Efficient Implementation of Adaptive Filters and Classifiers Using Multilayer Perceptron Feedforward Neural Network

A Thesis submitted for

the award of degree of

DOCTOR OF PHILOSOPHY

by

Sakshi

(951206001)

Under the guidance of

Dr. Ravi Kumar

Associate Professor, ECED

Department of Electronics & Communication Engineering

Thapar Institute of Engineering and Technology

Patiala-147004 (Punjab), INDIA

Certificate

I, Sakshi, hereby certify that the work which is being presented in this thesis titled, " **Efficient Implementation of Adaptive Filters and Classifiers using Multilayer Perceptron Feedforward Neural Network**", for the award of degree of **"Doctor of Philosophy"** submitted in Electronics and Communication Engineering Department of Thapar Institute of Engineering and Technology, Patiala, is an authentic record of my own work carried out under the supervision and guidance of Dr. Ravi Kumar and refers other researcher's works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree or diploma in this or any other university.

(Sakshi)
951206001

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(Ravi Kumar)
Associate Professor
Electronics and Communication Engineering Department
Thapar Institute of Engineering and Technology
Patiala

Abstract

Artificial Neural Networks (ANNs) are the mainstay of machine learning. It is hard to talk about pattern analysis in general without mentioning ANNs or its modern-day variants (e.g. deep networks). Used prolifically as a tool since decades, ANNs as a computational block have until recently been treated as a black box with a grain of suspicion about its actual efficacy. However, eventually the 'black box' has become larger and harder to fathom about since the advent of deep learning which has now propelled the machine learning research in some sort of a revolutionary path where everyone treading it is armed with some variant of a 'deep learning tool'. The time is now ripe to open up the black box of ANN and play around with its components to see how stuff works and may be make it work better. This thesis is a compilation of efforts made for improving the performance of a typical ANN with a basic text book architecture with an aim to extract an efficient and versatile performance an ANN with a minimalist architecture in order to ensure a hardware implementable design and an IC compatible implementation.

In general the following methodology was adopted for identifying an optimal architecture and measuring its performance. The data were first partitioned into training and test sets and several candidate architectures of the ANN were initialized. All the architectures were trained separately were trained repeatedly and Mean Squared Error (MSE) was recorded for each set of experiments. In order to avoid over-fitting K-fold cross validation scheme was adopted and the data in the training and test sets were replaced after a fixed set of experiments. The ANN architecture giving minimum average MSE was selected as the optimal architecture and as then subjected to test data for evaluating its generalization performance. In three out of four chapters describing backpropagation (BP) trained ANN, the above-mentioned methodology was adopted. The architecture was restricted to a single hidden layer with number of neurons varying from two to eight, the aim of the researcher being obtaining the best possible performance from the simplest and shallowest network.

This thesis is organized into the following sections:

Chapter 1 presents an overview of the current technological scenario in the backdrop of a resurgence of ANN underlining the importance of and motivations behind the presented work.

Chapter 2 proposes a principal component analysis (PCA) based learning rate variation methodology by making it dependent upon the output feature statistics rather than the input space as done traditionally. This technique also applies advantage of PCA to assess conditioning of the input data so as to evaluate the sensitivity of error with input data variation and hence evading the whole adaptation process in case of an ill-conditioned data. Simulation experiments performed with several benchmark data sets with variation in the number of attributes and classes reveal the efficacy of the proposed algorithm. The formula proposed in this work does not hurt the computational efficiency of the network, favoring its implementation on any physical environment.

Chapter 3 describes a novel weight pruning technique that not only knocks off the non-relevant weights but also upgrades the performance in terms of generalization and computational complexity. "Coefficient of dominance", a statistically dependent parameter has been introduced in the evaluation of complexity penalty term so as to scrutinize the relevance of weights which further helps to quantify the information content of the weight set. The weight with higher information content can bring out the major variations of the input data for better performance and hence will be retained. The analysis of the methodology used in this chapter has been conducted for datasets with a large diversity of attributes and in comparison to the well-known pruning strategies in terms of execution time, convergence rate, generalization and computational complexity.

Chapter 4 attempts to implement a hybrid learning approach which is a blend of a recently proposed unsupervised, biologically plausible algorithm for training the hidden layers and conventional least mean square (LMS) algorithm for the output layer. This amalgamation is aimed to improve the convergence by training all the neurons concurrently and bypassing the recursive computation of the local gradient. Training the hidden layer neurons using the unsupervised approach helps to provide a direct control over the hidden layer dynamics which also caters for faster training of the output neurons. The performance of the ANN has been investigated for various benchmark datasets with variation in number of hidden layer neurons, epochs, activation functions and parameters of the activation functions.

Chapter 5 aims to implement adaptive filter using an ANN contrasting its traditional design approach. This work commences by implementing a variable step-size adaptive filter with the

conventional design approach using variants of LMS and using direct, transpose and a novel area efficient structure. The analysis of filters has been carried out considering the estimated hardware utilization resources, actual area, delay and power targeting specific FPGAs. In the later part of this chapter, ANN based adapted filters are implemented using variants of LMS and trained using two different variants of BP viz. gradient descent with momentum BP and Levenberg Marquardt algorithm. This approach is investigated by comparing the original required signals with that of signals recovered after training the network. This study encompasses analyzing the two approaches presented in this chapter considering both the hardware overhead and computational resources.

Chapter 6 summarizes the key findings and significant contribution of this thesis and the possible future research directions.

Acknowledgment

First of all, I express my gratitude to the Almighty, who blessed me with the zeal and enthusiasm to complete this work successfully.

I thank my supervisor Dr. Ravi Kumar, Associate professor, ECED, Thapar Institute of Engineering and Technology, Patiala for his suggestions and constant support during my research. I am grateful to him for motivating and inspiring me to go deeply into this field and supporting me throughout the life cycle of PhD.

I am also thankful to Dr. Alpana Agarwal, Head, ECED, for her guidance through the early years of chaos and confusion. I am thankful to my Doctoral committee members for their constructive comments and regularly ensuring the progress of my research work. My deep regards to Dr. Prakash Gopalan, Director, Thapar Institute of Engineering and Technology, Patiala for providing access to facilities, which have been immensely helpful for the completion of my work.

I wish to thank the faculty and staff members of ECED, Thapar Institute of Engineering and Technology, Patiala for their cooperation and support. I am also very thankful to my friends for their continuous motivation and moral support.

My mother provided an environment that guided the first steps of the journey and encouraged the later ones. My brother helped me in every possible way at each and every step during last few years. My husband has been my pillar of strength throughout. He has played many roles- mentor, best-friend and foremost a continuous source of motivation. Finally, lots of love to my daughter Nysha Bajaj.

Sakshi

List of Publications

International Journals (Indexed by SCI)

1. Sakshi and Ravi Kumar, "A Computationally Efficient Weight Pruning Algorithm for Artificial Neural Network Classifiers," *Arab J Sci Eng*, Oct. 2017, doi.org/10.1007/s13369-017-2887-2. *(Published by Springer Heidelberg, Impact factor: 0.865, ISSN: 2193-567X)*

2. Sakshi and Ravi Kumar, "A Novel Design and FPGA Implementation of Filters Adapted Using LMS Variants. *J Circuit Syst Comp*, Nov. 2017, doi.org/10.1142/S0218126618501256. *(Published by World Scientific Publ. Co. Pte. Ltd., Impact factor: 0.481, ISSN: 0218-1266)*

Table of Contents

List of Figures

List of Tables

List of Abbreviations

ANN	Artificial Neural Network
NN	Neural Network
MSE	Mean Square Error
GA	Genetic Algorithm
SNR	Signal to Noise Ratio
BRNN	Bilinear Recurrent Neural Network
RNN	Recurrent Neural Network
EEG	Electroencephalogram
LM	Levenberg-Marquardt
BP	Back Propagation
IOT	Internet of Things
GPU	Graphics Processing Unit
LMS	Least Mean Square
DSP	Digital Signal Processing
ASIC	Application Specific Integrated Circuit
DNN	Deep NN
FPGA	Field Programmable Gate Array
VLSI	Very Large Scale Integration
HDL	Hardware Description Language
VHDL	Very High Speed Integrated Circuit HDL
GPP	General Purpose Processor
MAC	Multiply Accumulate Unit
DA	Distributed Arithmetic
LUT	Lookup Table
NLMS	Normalized LMS
RBF	Radial Basis Function
LA	Learning Automata
IALR	Inference Adjusting Learning Rate
MLP	Multi Layer Perceptron

ECG	Electrocardiography
PS	Premature Saturation
OBD	Optical Brain Damage
OBS	Optimal Brain Surgeon
CAPE	Cross-correlation Analysis of back-Propagated Errors
HLMS	Hebbian LMS
PCA	Principal Component Analysis
ANC	Adaptive Noise Canceller
PC	Principal Components
KPCA	Kernel PCA
UMPCA	Uncorrelated Multi-linear PCA
EKPCA	Efficient KPCA
RLS	Recursive Least Square
EKF	Extended Kalman Filtering
HN	Hidden Neurons
SPA	Successive Projections Algorithm
ELM	Extreme Learning Machines
UTI	Urinary Tract Infection
WDE	Weight Decay and Elimination
MAXCORE	Principle of Maximum Correlation of Errors
ADALINE	Adaptive Linear Neuron
SOP	Self Organizing Map
EM	Expectation Minimization
GDM	Gradient Descent with Momentum
SD	Sign-Data
SE	Sign-Error
SS	Sign-Sign
EMG	Electromyography
SP	Soft Plus
ISP	Inverse Soft Plus

Chapter 1
Introduction

This chapter provides an overview of the thesis along with the motivation to choose this particular area of research. The introduction to adaptive filters, aritificial neural networks, classifiers and function approximators and their interconnection is briefed initially, followed by the detailed literature review.

Based on the gaps identified in the study, objectives of the thesis are defined. The research methodology followed to achieve the formulated objectives is also described in detail.

1.1 Artificial Neural Network (ANN)

There are approximately 100 billion neurons in human brain that are connected through about 100 trillion connections that form the most complex object known in the universe. The major functions of the brain such as sensory information processing and cognition, are the results of emergent computations carried out by this massive Neural Network (NN). ANNs [1]–[3] are computational models that are inspired by the principles of computations performed by the biological NNs of the brain. NNs possess many attractive characteristics that may ultimately surpass some of the limitations in classical computational systems. The processing in the brain is basically parallel and distributed: the information is stored in connections, mostly in myeline layers of axons of neurons, and, hence, distributed over the network and processed in a large number of neurons in parallel. The brain is adaptive from its birth to its complete death and learns from exemplars as they arise in the external world. NNs have the ability to learn the rules describing training data, and from previously learnt information, respond to the novel patterns. They are fault-tolerant, in the sense that the loss of a few neurons or connections does not significantly affect their behavior, as the information processing involves a large number of neurons and connections. ANNs have found applications in many domains, for example, signal processing [4][5], image analysis [6]–[9], medical diagnosis systems [10][11], and financial forecasting [12][13]. The roles of NNs in the afore-mentioned applications fall broadly into two classes: pattern recognition [14][15], and functional approximation[16][17]. The fundamental objective of pattern recognition is to provide

a meaningful categorization of the input patterns. In functional approximation, given a set of patterns, the network finds a smooth function that approximates the actual mapping between the input, and the output. A vast majority of NNs are still implemented on software on sequential machines. Although this is not necessarily always a severe limitation, there is much to be gained from directly implementing NNs in hardware, especially if such implementation exploits the parallelism inherent in the NNs but without undue costs.

1.2 ANN as an adaptive filter, function approximators and classifier

Adaptive filters, are a class of filters which work without any prior information about the signal and noise statistics. The parameter adjustment can be accomplished by the means of an optimization algorithm. Moreover, due to the complexity involved in the optimization algorithms, almost all adaptive filters are digital filters. Adaptive filters have found their applications in diverse fields such as communications [18]–[20], controls [21][22], robotics [23][24], radar [25][26], biomedical engineering [27][28], and many more. Indeed, their applications can be classified into four basic categories: system identification [29][30], inverse modeling [31], prediction [32] and echo or interference cancelling [33].

Signal and noise statistics are required as a priori while designing any linear class of filters. The awareness of these factors beforehand can minimize the impact of noise by following a statistical criterion of minimizing the Mean Square Error (MSE). While designing adaptive filters, if the statistics of the signal are found to be stationary, the adaptive filters would converge to an optimum Weiner solution else, it will tracks them. To maintain tracking, the rate of adaptation must be faster than the rate at which the signal statistics are varying. However, the parameters of the adaptive filters are adapted continuously as data flows through it, hence, they are strictly non-linear in nature. Although, linear adaptive filters do exist which follows the classical Weiner filter theory, but can only deal with second order data statistics and are optimum for Gaussian input data distribution. Deviation from any of the parameters like dealing non-Gaussian input data distributions, and higher order statistical input information, it is mandatory to include nonlinearity in the structure of adaptive filters. ANN [34]–[36], fuzzy networks [37][38], and Genetic Algorithms (GA) [39]–[41] are some other non-linear forms of adaptive filters, the learning of

which does not depend upon the linear modeling techniques and can also handle non-Gaussian and higher dimensional input statistics.

Consider a dynamical system, of which the mathematical characterization is unknown. The only available parameters of the system is, a set of labeled input-output data generated by the system at discrete instant of time. Now, the task is to design a model of the above described system with multiple-input single-output by building it around a single linear neuron. This neuronal model will make required adjustments in the synaptic weights of the neurons by following a certain criterion and keeping the following points under consideration:

- The algorithm starts with a random weight initialization process.
- In accordance to the variations in the system's behavior, adjustments to the synaptic weights are made on a continuous basis.

The neuronal model described above is referred to as an adaptive filter.

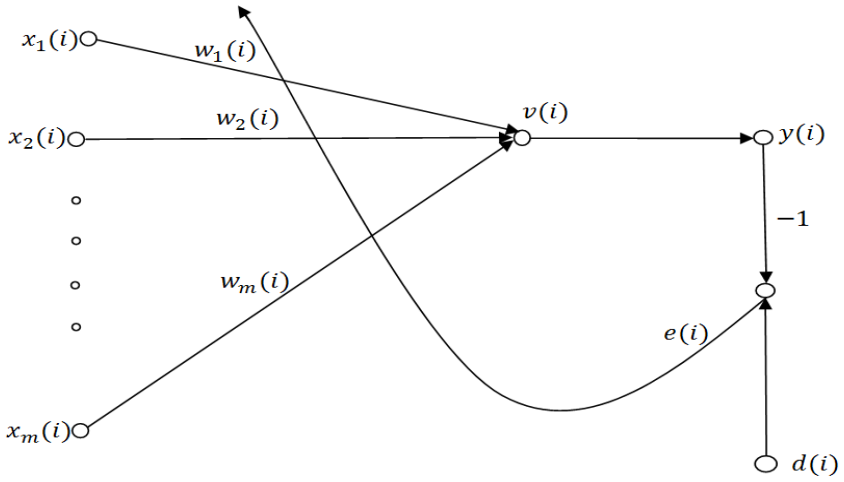


Figure1.1: Signal flow graph for the adaptive model.

Figure 1.1 shows signal flow graph of the adaptive filter representing two continuous processes:

1. Filtering process is the one in which two signals are evaluated. The first one is an output that is produced in response to the input vector and other one is an error signal generated by subtracting the actual output from the desired output produced by an unknown system.

2. Adaptive process, is the one in which the synaptic weights of the neuron are adjusted in accordance with the desired signal.

The combination of these two processes working together constitutes a feedback loop acting around the neuron. Since the neuron is linear, the output $y(i)$ is exactly the same as the induced local field $v(i)$, i.e.

$$y(i) = v(i) \tag{1.1}$$

$$where\ v(i) = \sum_{k=1}^{m} w_k(i)\, x_k(i)$$

where $w_1(i), w_2(i), w_3(i), \ldots\ldots w_m(i)$ are the m weights of the neuron, measured at time i. The neuron's actual output $y(i)$ is compared with the desired output $d(i)$ obtained from the unknown system at time i to generate an error signal called $e(i)$. The network adaptation algorithms aim at finding the synaptic coefficients which minimize the given satisfaction criterion involving, usually, the squared error $e(n)^2$. Thus, it is clear that adaptive filters and NNs are formally equivalent, and NNs [36][42], which are potentially capable of realizing non-linear input-output relations, are simple generalizations of the non-linear adaptive filters [43]–[45]. Till date, a variety of applications have employed NNs as adaptive filters. Xiao et al. [46] proposed NN adaptive filtering model based on wavelet transform to filter out the noise and proved to provide real time de-noising and hence improved the Signal to Noise Ratio (SNR) of the noisy signal. In [47], Cai et al. devised a NN based method for filtering impulse noise from an image. The method was found to be much simpler and provided improved performance than its conventional counterpart. Bilinear Recurrent NN (BRNN) were introduced by Park [48] to solve the time series prediction problems and provided promising results in comparison to Recurrent NN (RNN) but are computationally very expensive. Hence, Zhao et al. [49] proposed an adaptive filter based pipelined BRNN structure with low computational complexity. The performance improvement of the proposed system was justified for nonlinear system identification, channel equalization, nonlinear and non-stationary signal prediction applications in comparison to BLRNN and RNN filters. Later on, Manfred and Enke [50] formulated an adaptive NN filter to be used as noise forecaster. Electroencephalogram **(**EEG) signals, observe the electrical activity of brain, are contaminated by other biological signals which makes its examination very complex. In order to eliminate these unwanted signal and to generate a correct analysis and diagnosis of EEG, Quazi et al. [51] presented Firefly and

Levenberg-Marquardt (LM) optimization based algorithm for NN enhanced adaptive filtering model.

ANN as described above combines a set of input samples to generate the output that mimics the desired responses. This problem can also be viewed as function approximation problem where desired response is some fixed function of the input. The aim of the learning system is to discover the function. Multilayer feedforward NNs, are considered to be universal approximators [52][53], provided that approximated functions are univariate [54]. This approximation is valid provided, there is no restriction on the number of hidden layer neurons [55]. Activation function is another important criterion to be considered for the approximation accuracy of the function. It has been analyzed that sufficiently smooth i.e. continuous, bounded and non-constant activation functions are capable of accurate approximation to a function and its derivatives [55]. Many variations of NNs in a variety of function approximation application have been implemented till date [16][56]–[58] . In a relatively different application of function approximation, two variants of NN approaches were used to approximate the function for the retrieval of soil moisture and crop variables [59].

Classification is essential to separate large datasets for the purpose of decision making [60]–[66], pattern recognition [67]–[70], dimensionality reduction [71][72], prediction or forecasting [73]–[76] etc. It is one of the active research and application areas of NNs [77]. Back Propagation (BP) trained classifiers are even able to train a NN using minimal training set. This feature of ANN is useful especially in satellite and geographical information processing where a huge time and cost is involved in developing a set of training sites [78]. Pattern classification is considered to be a special case of function approximation in which the output of the function is restricted to one of the discrete values (or classes). These NNs typically have input and output neurons with the k^{th} output neuron trained to value one (while rest of the neurons are trained to value zero) for patterns belonging to the k^{th} class.

1.3 Motivation

The current technological scenario is witnessing a paradigm shift. Over the last couple of decades, there is a fundamental change in the basic concepts and experimental practises of a number of scientific disciplines. Forbes has listed automated banking [79][80], Internet of Things (IOT) [81]–

[83], big data [84][85] and self driving cars [86]–[90] among the four most disruptive technologies of the near future. A close scrutiniy may reveal some denominators common to all the abovementioned technologies. Needless to state, data analytics and`automation have already started driving the technological bandwagon. The general requirements of high performance data analysis and automation has led to massive proliferation in artificial intelligence research and no wonder, ANNs in general and deep learning in particular are the most sought after computational paradigms. Deep learning is but a hierarchical variant of traditional ANNS with a large number of layers in order to capture class descriptors/features in an unsupervised manner. Especially, for achieving high performance image and multimedia processing, some limitations of ANNs were overcome algorithimically. In addition to it, availability of large volumes of training data and Graphics Processing Unit (GPU) computing has made it possible to run the learning alogorithm through a large number of layers. However, there are several applications where data is sparse and/or there is a restriction on complexitiy of the network due to requirements of hardware implementation. In those cases, construction of custom networks and selection of an appropriate feature extractor is extremely important and most of the time is highly problem dependent too.

The fields of adaptive filtering and ANNs have been developing independently, but their structures and process of adaptation/learning are closely related [1][36][42]. A feedforward NN has an input layer which is fed with signal samples, an output layer which provides the estimated signal, along with one or more intermediate/hidden layers. In this sense, one can describe adaptive filter to be a multilayer feed-forward network with only one output, where the training process corresponds to the filter design. Adaptation algorithm, step size, filter length and structure are the important attributes of adaptive filter that affect their performance [91][92]. The BP algorithm [93][94] is the usual design procedure for an adaptive filter that represents a generalization of the Least Mean Square (LMS) algorithm [95] for feed-forward networks. The beauty of the training approach employing BP algorithm is that it is a systematic, step-by-step procedure that can be applied independent of the topology of the network, and the input dimensionality. Step size and filter length are the other two major parameters that affects the performance of the LMS adaptive filter in terms of its convergence rate and computational complexity. Length of adaptive filter is also a significant factor because it changes the number of iterations required to minimize error signal. In other words, larger the length, lesser will be the number of iterations to minimize error and vice-versa. Furthermore, the structure of the filter also plays a pivotal role in terms of power utilization,

silicon area consumption or speed performance while comparing it to its counterparts. In general, the direct-form adaptive filter have better filtering performance and power utilization than transpose-form adaptive filter whereas transpose-form adaptive filter have better silicon area consumption and speed performance as compared to direct-form adaptive filter. However, there is a great need to design a structure which can utilize the benefits offered by both kind of structures.

In the last few decades, the growing demand of digital audio devices, mobile phones, hearing aids etc. are the major drivers for the growth of portable and embedded Digital Signal Processing (DSP) systems with hard constraints. These applications also demand a faster time to market meeting all the stringent constraints. The possible approaches followed to implement these applications ranges from a general purpose processor, DSP processors to an Application Specific Integrated circuit (ASIC) custom chip. The computational cost (complexity) is an important issue to be addressed during evaluation of algorithms. Sometimes, the algorithm's effectiveness in providing a solution to a given problem requires the execution of complex computations and the use of excessive memory resources, which can create severe difficulties in real-time or low-cost applications. Due to this high computational complexity of adaptive filters, their hardware implementation is also not that easy task. However, it becomes indispensable incase of real-time execution. On certain occasions, the redundancy created by the network weights/coefficients increases the computational complexity of the network. By removing these excess components from the network makes the system compact and idle for the hardware implementation. The reduction in the network complexity makes it a suitable candidate for pruning the Deep NNs (DNN) which are massively connected. Moreover, the design with reduced system complexity have an edge from the implementation point of view in any physical environment. From the perspective of the hardware implementation, the complexity of the design can be defined in terms of its hardware (resource) utilization. Reducing the resource usage also improves the performance of any system on hardware not only in terms of area but also on power and delay front. Reconfigurable devices are gaining much popularity for hardware implementation of digital systems and prototyping [96]. Although, ASIC could yield the best solution that will meet all the strict constraints of its implementation but it does so at the cost of much longer design time and high cost involved. Since Field Programmable Gate Arrays (FPGA) produce both a programmable and a dedicated hardware option [97], they are considered an attractive choice for rapid prototyping. The suitability of FPGA as a hardware base for real-time audio processing had been studied and tested by Fohl et al. [98]. Technological

advances in Very Large Scale Integration (VLSI) fabrications have made FPGA a strong contender for hardware implementations of real-time DSP applications especially where timing requirement is strict. Hardware Description Languages (HDL) such as Very High Speed Integrated Circuit HDL (VHDL) or Verilog are used to realize these kind of implementations [99]. In addition to this, FPGA is also an interest for battery-operated devices as it consumes less clock system speed in comparison to a DSP or a General Purpose Processor (GPP) and consumes less power. Modern FPGAs contain many additional resources like on-chip multipliers, Multiply Accumulate Units (MAC) etc. optimized for low power consumption, high speed implementation and DSP applications [100]. Ramakrishna et al. [101] successfully implemented a low power adaptive noise canceller based on LMS algorithm on Xilinx XC3s200 FPGA. Allerd et al. [102] implemented a multiplier-less 32-tap LMS adaptive filter based on Distributed Arithmetic (DA) on Altera Stratix FPGA. The use of DA substitutes multiplication operations with a series of Lookup Table (LUT) accesses. The real time implementation of non-stationary noise cancellation adaptive Wiener filter was described for car environment application by Jheng et al. [103]. Mollaei [104] had implemented a Normalized LMS (NLMS) algorithm based adaptive filters on a TMS320C6711 board to analyze its performance for various frequency noises. High speed LMS based adaptive filter by Vella et al. [105] for echo cancellation was illustrated using different architectures. Elhossini et al. [106] have implemented an LMS adaptive algorithm for audio processing using three different architectures targeting Xilinx Virtex-II FPGA chip. Later on, Mustafa et al. [107] implemented a 64-tap 9-bit ANC on Cyclone II FPGA. Nekouei et al. [108] designed a constant step size and auto correcting LMS adaptive filter and implemented on a spartan3 FPGA board.

1.4 Literature Review

The structure of adaptive filters is similar to that of ANNs, and their process of adaptation/training can be studied under the same framework. The usefulness of NN filters can be efficiently investigated due to the existence of the BP algorithm, which is their usual design procedure, and represents a generalization of the LMS algorithm for feed-forward networks. Thus, the filter design can be viewed as a problem of unconstrained optimization that is iteratively solved by the method of steepest descent [93].

BP algorithm was originally invented by Paul Werbos in 1974 and later on redefined independently by Rumelhart and Parker [93][94]. Since its reinvention, it has been the popular training approach for multilayer feed-forward NNs. The beauty of learning/training using the BP algorithm lies in the fact that it is a systematic and step-by-step technique and can be applied independent of the network topology and input dimensionality. The general weight update equation of BP algorithm can be expressed as:

$$w(k+1) = w(k) + \eta \sum_{p=1}^{P} \nabla E^{p} \tag{1.2}$$

where η represents the learning rate, $w(k+1)$ represents the weight for the next iteration, ∇E is derivates of the error with respect to weight and p varies from 1 to P where P represents number of training patterns.

1.4.1 Issues associated with BP

The BP algorithm suffers from two major drawbacks; one is the slow rate of convergence [109] and other is trapping in local minima [110]. If the hunt for global minimum attains such a point in the search space and the algorithm will stop the search process further considering it to be the minimum point, while it would be desired that it continues towards a global minimum or at least a sufficiently low one. Furthermore, BP works well for simple training problems but its performance falls off rapidly as the dimension or complexity (high correlation in the input samples) of input data increases [111]. Since the discovery of BP algorithm, many researchers have tried and have been trying to resolve the foremost issues of slow convergence and trapping in local minima.

One year past the birth of BP, Sutton [112] proved that gradient descent is neither only nor the best learning procedure. The author presented in detail the two major problems associated with BP or in general, with all steepest descent learning procedures. The first problem is that all steepest descent learning algorithms fails when it comes to surfaces such as ravines. Ravines are the curves where gentle and steep slopes come across simultaneously and in different dimensions. The learning rate can be easily adjusted according to the changing gradient of the curve, whereas it becomes a tough choice to alter the learning rate parameter when it comes to ravines. The reason is that the curve is not only changing slope in one direction but in all directions. Therefore, the

problem cannot be solved by simply varying the learning rate over time. Instead, it has to be different in different directions. The other issue is the interference between learning with different patterns.

Another researcher claimed that BP is not an efficient algorithm when it comes to multilayered NNs. It was also pointed out that training bigger networks using sigmoid activation function is very time consuming [113]. Fukuoka et al. [110] dealt with one of the important problems while training using BP i.e. trapping into the local minima. Hence, they proposed a modified BP algorithm which keeps the activation function derivative large when the error signals are large so that weight adjustment terms will also be large. In this way, the algorithm converges to a global minimum in a reasonable amount of time. Another form of BP algorithm was proposed by Zweiri et al. [114] in which they have included an extra term called proportional factor in their weight update equation in addition to learning rate and momentum parameter. This inclusion actually reduced the number of epochs and provided the escape from local minima whilst maintaining the simplicity and efficiency of BP algorithm. Bi et al. [115] concluded that the local minima problem is due to the update disharmony between weights connected to the hidden layer and output layer. Therefore, they provided solution to this problem by adding another term to the error function of conventional BP. Overfitting of training data is majorly responsible for all the problems associated while training an NN with BP algorithm [116]. Overfitting is a situation when training error is driven to a very small value but when the new data is presented to the same network, the error comes out be large. This mechanism greatly affects the generalization ability of the network and is more common in networks with hidden layers. Generalization ability of a network implies how well the network performs when fed with samples outside the training patterns. Huang et al. [116] proposed a GA based BP model to avoid the overfitting problem. Two new algorithms viz. *AlgoRobust* and *AlgoGS* have been proposed by Wan et al. [117] to improve the generalization performance of the NNs by introducing additional process in the hidden layers. Lin et al. [118] used early stopping and statistical regression technique to improve the generalization performance of the Radial Basis Function (RBF) NN. Yang et al. [119] proposed a novel NN ensemble approach with an aim to enhance the generalization performance. Another researcher put forth a new information theoretic approach to simplify the learning procedure without strong inhibition yielding maximum information. This information change directly impacted the generalization performance of the network in a positive manner [120]. Srivastava et al. [121] employed

"dropout", to improve the generalization performance of the network. "Dropout", refers to removing units from the network along with all its incoming and outgoing connections. This technique improves the generalization performance of the network by reducing overfitting. Recently, many nature inspired meta heuristic algorithms have been developed which provide derivative free solution to optimize the complex tasks [122]–[124].

1.4.2 Parameter variation

There are many parameters that can be varied to alter the performance of the network trained using BP algorithm. They are learning rate, momentum term, initial weights and biases, network topology, activation function, and its gain and slope. Some authors have studied the interdependence of these parameters and the effect of their variations on the network performance [125][126]. The subsequent subsections will present a survey on the recent advancements in the variation of these parameters and their impact on the NN implementation.

1.4.2.1 Learning rate

It has been proved many times that a constant learning rate is not sufficient enough to handle a complex, and multidimensional error surface and it should be adapted continuously during the weight adjustment process [127]–[129]. Furthermore, the constant learning rate used in BP algorithm fails to optimize the search for the best weight combination [130]. Such a search methodology has been classified as a “blind-search”. Various heuristic techniques have been proposed till date for learning rate adaptation. Jacobs [127], Yamada [131] and Silva et al. [131] used the information obtained from the earlier iterations to adapt the learning rate parameter. The major drawback of their contribution was that they had to use a large number of iterations before reaching a point that would provide the suitable parameter. Unfortunately, these techniques only gave the direction of adjustment of the learning rates and the exact learning rate adaptation was still decided by trial and error. Hush and Salas [132] used the gradient reuse procedure to determine the new value of learning rate. They tried using the same gradient value in many iterations till the point the cost function start decreasing, and hence calculated the new step size using the reuse count value. However, the improvement achieved in convergence using this technique was

moderate. Weir [133] established a new height and gradient based self determination of learning rate parameter. Later on, Yu et al. in their series of research contribution [134]–[136] devised many ways for improving the performance of BP algorithm either by adapting the learning rates by utilizing the second order derivates of the cost function with respect to learning rate or by adapting the momentum term used in the weight update equation of BP or both. However, all these contributions suffered from a major drawback that the computational complexity is about 2 to 3 times the complexity involved in the conventional BP, with only 2 to 3% improvement in the results. Furthermore, a genetic based dynamic self adaptation technique to improve the convergence have been proposed by Solomen et al. [137]. The fundamental idea was to evaluate the cost function by mutating the learning rate parameter used in the previous step and choosing the one which reduces the cost function. The procedure terminated as soon as the reduction in the cost function came to a standstill. They also argued upon the use of this technique for other parameters of the network simultaneously. However, this method failed to mitigate the problem of being stuck in the local minima. Plagianakos et al. [138] also tried to improve the convergence of batch BP by providing an adaptive learning rate mechanism. The adapted learning rate parameter is different for every epoch and is calculated using the information obtained from the previous iteration's weights and gradient value. Mandic [139] in 2000 established a way to calculate the optimal learning rate value by using the taylor series expansion of the instantaneous error value. The derived optimal adaptive learning rate of a NN trained by BP exhibit behavior similar to that of NLMS. However, the computational complexity of that expression is so high that it is not practical to apply it on real world data sets. Some researchers have also tried learning rate adaptation techniques which does not involve the use of cost function. Meybodi et al. [140] postulated that the Learning Automata (LA) based schemes can automatically adapt the learning rate and hence provide optimum weights. Lee [141] concluded that one can calculate learning rate by fuzzy reasoning, but the algorithm needs to define a membership function for the fuzzy reasoning by trial and error. Zweiri et al. [114][142] presented a technique based on Inference Adjusting Learning Rate (IALR) originally devised by Barnard [143]. They devised a three-term weight updating equation, in which the learning rate can be automatically adjusted according to the characteristic datum length. The addition of new term other than the learning rate and the momentum term have actually sped up the convergence, and the algorithm was successfully able to escape from the local minima. Moreover, Wang et al. [144] provided a solution to improve the

learning performance of the NNs trained using BP algorithm. The model successfully combined a new learning rate adaptation with a weight adaptation frequency manner, so as to improve the convergence speed and the generalization performance of the network. The weight adjustment mechanism provided different learning rate for each weight. Besides this, the learning rate variation was dependent upon the current iteration weight adjustment and the datum. Li et al. [145] provided a self adaptation method of learning rate variation which is based on the topology of the network, training data and gradient of the error curve surface. The method provided convergence in much less iterations as compared to traditional constant learning approaches. Later on, many researchers have also come up with an idea to adapt the learning rate parameter of stochastic gradient descent algorithm so as to minimize the expected error at any one time and improve the performance [146][147].

1.4.2.2 Momentum Term variation

Rumelhart et al. [94] have established the advantages of including a momentum term in the basic weight update equation proposed in [148]. However, the momentum term required storage of the weight update terms from the previous iteration. Later on, the momentum term got incorporated into the weight update equation considering the performance improvement of BP algorithm due to its addition. The first version of original algorithm had a constant momentum term in the interval (0,1). However, it was deduced from the simulation results that momentum term should also be adapted throughout the entire training process like the learning rate parameter so as to produce better training results. This led to the emergence of various techniques for momentum term adaptation [149]–[154]. Recently, an adaptive momentum algorithm has been proposed by Hameed et al. [155] which is actually based on the algorithm proposed for learning rate variation in [156]–[159] and has been successfully used in various adaptive filter applications with better convergence and reduced cost function as compared to the earlier adaptive momentum algorithms.

1.4.2.3 Activation function parameters variation

The gain of the activation function can also be adjusted to improve the convergence of the BP. Eom et al. [126] concluded that adjusting the gain of the activation function is equivalent to changing the learning rate parameter, the weights and the biases. They have used a fuzzy logic framework to dynamically adjust this gain parameter so as to improve the performance of BP.

Nawi et al. [160] also introduced adaptation in the activation function gain so as to improve the performance of BP algorithm. The gain value changes adaptively for each node of the network. The variations in the slope parameter of the activation function have been studied by Rezgui et al. [161]. Ivanikovas et al. [162] introduced slope variation on a SAMANN [163] network which is basically used as a tool to visualize multidimensional data sets. It has been observed that the network's convergence performance has been enhanced using higher slope parameter for the activation function. Furthermore, the effects of the slope variation had a better effect on the network performance rather than the variation in the learning rate parameter. Yu et al. [164] devised a modified activation function with two free parameters in order to improve the learning speed of Multi Layer Perceptron (MLP) NNs. Ozbay et al. [165] proposed an ANN based Electrocardiography (ECG) classification method with adaptive activation function.

1.4.3 Neuron saturation

In BP algorithm, the final output has to attain one of the two extreme values of the activation function which can be represented as a sigmoid function. If the weighted sum to any output node is near the wrong extreme value, we say the node is “incorrectly saturated". When a node gets saturated, the gradient of the sigmoid function corresponding to that node becomes very small. This may eventually lead to even smaller changes in the weights of the network because the error has reduced to a minimal value. This phenomenon is also termed as Premature Saturation (PS) [166]. Balakrishnan et al. [167] explicated this mechanism as the occurrence of flat spots. Flat spots are the points on the curve where the derivative of the activation function becomes approximately zero. The saturated output nodes or the hidden nodes will not allow any improvements in the weights causing an increase in the number of epochs required to train a network. Hence, they provided a solution to handle the flat spots so that the network can learn even in the presence of such points. This situation is basically the reason behind the slow convergence of BP and trapping into a local minimum. Saturation of the output neuron is responsible for the slow convergence [166] whereas hidden layer neuron saturation creates the local minima problem [168]. It was found later by Lee et al. [166][169] that inappropriate setting of initial weights is responsible for the PS. Oh [170] used an adapted version of the error term to deal with the problem

of slow convergence by lowering the output nodes probability to be near the wrong extreme values. The reconstructed error term also generated a weak error signal for the output node near the desired value. Vitela et al. [171][172] also tried to find the reason behind trapping in these spots. They concluded that the momentum term is the major factor responsible for the occurrence of this situation. Thereafter, they proposed a modified BP algorithm that uses different values of momentum parameter when necessary conditions are satisfied. Subsequently, Lee et al. [173] proposed an error saturation prevention function to prevent the saturation of output nodes. An improved learning method for MLP has been proposed by Wang et al. [168] to avoid the saturation problem of hidden neurons. In this method, the activation functions used in the hidden layer were different for each training process which were adjusted during the learning procedure. This actually led to the prevention of the network getting trapped into local minima caused by the saturation of neurons in the hidden layer. In the very same year, Wang et al. [174] devised a modified error function for conventional BP to escape the saturation of hidden neurons. They appended an extra term with the original error term to coordinate the update of hidden layer weights with those of output layer weights. Moallem et al. [175] dictated that hidden neuron saturation is responsible for flat spots in the error surface which makes the flow of information in both the directions (forward and backward) to a complete halt. Therefore, they provided a complimentary method based on a suitable combination of anti-saturated hidden neurons, learning process, momentum term and parallel tangent technique.

1.4.4 Network Selection

Ever since the discovery of BP algorithm, network model selection or network optimization has been a hot topic of research. Smaller networks have optimal generalization performance, but their training process may require a lot of efforts due to the limited processing elements. Secondly, the reduced complexity of smaller networks systems has an edge from the implementation point of view in any physical environment. However, smaller architecture will not be able to learn the data properly, no matter which training algorithm is used. On the contrary, the network with too many parameters like weights, hidden layers, and hidden neurons are prone to learn undesirable characteristics of the training data, so offer poor generalization. On the other side, too many parameters help for faster learning avoiding the local minima problem as well. Therefore, both kind of architectures offer merits and demerits simultaneously. An optimal architecture is one

which is large enough to learn the problem and small enough to generalize well. Hence, different strategies have been adopted in previously to fulfill the above stated requirement.

The common approaches for network optimization are incremental, decremental, and hybrid. The incremental algorithms also known as constructive algorithms, are one in which new elements (hidden neurons/ weights/ a new layer of hidden neurons) are added to a minimum size network during training when the network is not able to meet the required design specification. The major issue concerning these approaches is that the ANNs obtained are usually huge and redundant information gets stored in the weights which cannot be eradicated. Secondly, the initial small network can easily be trapped into local minima and the training time may be increased [176]. Many constructive algorithms have been implemented so far for network selection [177]–[183] but cross correlation algorithm [177] is acknowledged by most researchers for its capability of learning both classification and continuous function approximation tasks [117][184][185]. Decremental algorithms also known as pruning algorithms [186] initiate with a larger network and remove the redundant layers, nodes, and weights during the ongoing training process. They are considered redundant because the presence of these elements has limited or no impact on the network performance. Instead, they increase the computational requirements of the system which may further affect the hardware implementation, eating up more resources on silicon. Castellano et al. [165] have shown that the total time required for training a large network and later on reducing it to a small size is comparatively much lower than the time required to train small networks. Consequently, it was deduced that initially, larger networks can be used for training and then their generalization can be improved by the process of pruning. The principal issue concerning the viability of MLP NN is " How large does the network be to perform a specific task?". Huang et al. [187] answered this by providing an upper restriction on the number of neurons of the hidden layer required to realize any function. Optical Brain damage (OBD) is the oldest pruning technique used for eliminating the excess weights from the network [188]. This technique uses second derivative of the objective function with respect to the parameters (weights) to compute the saliencies. The minimal increase in training error for weight elimination was allowed in OBD and it also assumes the Hessian matrix to be diagonal for computational simplicity. However, Hessian is non-diagonal for every problem and this lead to the elimination of wrong weights using OBD [189]. Optimal Brain Surgeon (OBS), allows correct weight elimination and improved generalization performance than any magnitude based pruning technique and OBD [189]. Until

now, a variety of pruning algorithms have been suggested which can be broadly classified into six different categories based upon the process involved in pruning the elements. These are penalty based [190], cross validation based [191], magnitude dependent [192][193], mutual information based [194]–[196], evolutionary methods of pruning [197][198] and sensitivity analysis based techniques [176][189][199]–[202]. All the above-mentioned techniques offer one benefit or the other depending upon the elements targeted for pruning the architecture. Augusta et al. [203] have given a comparative study of all the pruning algorithms developed so far. They have also devised a new algorithm of pruning neurons of both the input and hidden layer based on the significance measure of the nodes [204]. Another methodology based on the Cross-correlation Analysis of back-Propagated Errors (CAPE) signals has been proposed by Medeiros and Barreto [205]. CAPE defines the relevance of the weights with respect to the correlation between error signal of a given layer with the back propagated error signal to the previous layer. Recently, Tong et al. [206] formulated a new technique in which node pruning can be accomplished using the weight variation information. The third category of network optimization is the hybrid technique that uses both the incremental and decremental approach in their network design process [207][208].

1.4.5 Hybrid approach to training

Learning using a combination of unsupervised and supervised training methods was originally done to deal with high dimensionality problem of data [209]. Since then, there have been many attempts made by different researchers to blend these two approaches of learning for one application or the other [210]–[214]. Initially, this hybrid approach was applied, using unsupervised training for hidden layers and supervised to train the output layer [215]. However, this concept changed to applying unsupervised to the whole network first for rough clustering of data and subsequently applying supervised training to some layers for fine tuning the classification process [216]. Schwenker et al. [217] handled the problem of training with partially labeled data using a similar technique. Training the network for incompletely labeled data and using supervised learning can waste some useful information which may be hidden in the unlabeled data. Making use of the unlabelled data's hidden information is crucial for improving the classifier performance. Recently, Widrow et al. [218] suggested the amalgamation of unsupervised Hebbian learning and

supervised LMS algorithm. The union of these two algorithms generated a new learning paradigm called Hebbian LMS (HLMS) which has practical engineering applications. They also proposed that this new form of learning can be applied to MLP networks where unsupervised HLMS can be used to train hidden layers of the network and supervised LMS for training the output layer [219]. According to them, this new approach is more biologically plausible. The idea given by Widrow et al. [219] has already been implemented by Moody et al. [215] in 1989 for achieving faster learning using computationally efficient hybrid learning method. They have also proved that learning using their proposed hybrid approach is much faster than BP algorithm.

1.5 Gaps in the study

It is evident from recent developments in the broad domain of machine learning in general that the technological scenario is witnessing a paradigm shift driven by the advent of deep networks. However, this shift is largely governed by rapidly enhancing image and video processing market. Where on one hand, deep networks [220][221] possess 'magical' abilities to extract features relevant for classification, they are inherently data hungry and their hardware implementability remains a challenge till date. Having said this, shallow networks trained with BP are still an integral part of any deep NN (DNN) architecture albeit as a fine tuner. Another limitation of deep networks comes from the fact that even though a deep learning model can be interpreted as a kind of program, most programs cannot be expressed as deep learning models. Furthermore, mere scaling up current deep learning techniques by stacking more layers and more training data, can only superficially palliate some of these issues. Needless to say, shallow networks are fundamental building blocks of any huge deep net and any improvements in performance of shallow networks serve as a stepping stone to subsequent improvement in the more complex deep net version. Till date, a lot of work has been done to adapt the learning rate parameter with an aim to speed up the convergence. However, no one has come up with an idea of changing the learning rate with the changing statistics of the data. This kind of statistical information about the weight set can be a boon for designing the network pruning algorithms. Fisher information, which is a great parameter to measure the information content of any dataset, has never been used to quantify the viability or efficiency of any pruning strategy. It is also evident from the literature cited above that the performance comparison of LMS algorithm and its sign variants in terms of algorithm convergence and efficient hardware utilization have not been considered till date.

1.6 Objectives of the thesis

Based on the initial study, reported literature survey and the understanding established the following objectives are proposed:

1. Design and implementation of a general purpose efficient feedforward Multi Layer Perceptron (MLP) architecture with low computational complexity.
2. Implementation of an MLP based adaptive filter which uses multiplier of lower computational complexities.
3. Validate all the optimized parameters with standard tools.

1.7 Research Methodology

As outlined above, low architectural and computational complexity are quintessential objectives of this work. In order to achieve the *first objective*, a comprehensive review of different existing algorithms for lowering the computational complexity was conducted. The first parameter that is considered is the adaptation of learning rate to speed up the convergence. Many algorithms have been reported as observed in literature review that has altered the learning rate by one way or the other. However, a majority of them suffered from the problem of increase in the computational resources and training time. Some of the algorithms have even been reported to have two to three times increase in the usage of resources and computational time involved as compared to the conventional BP algorithm, with only 2 to 3% improvements in the results [222]. So, there is a need to design a technique which can speed up the convergence but not at the cost of increase in the computational cost of the design. A novel statistical method for learning rate adaptation has been proposed **(Chapter 2)** where, the adaptation is done by making the learning rate dependent upon changing statistical properties of the data using Principal Component Analysis (PCA) [223][224]. We have applied PCA on the output of the ANN during training phase in order to get an idea about the changing dynamics of the output feature space rather than concentrating only on the input feature space. However, the most novel aspect of the proposed technique is the application of PCA on the output data feature space rather than on the input data as done conventionally. Although, a standard formula for choosing an optimum value of learning rate has already been given by Mandic in [139] but the computational complexity of the formula is so high that it is not practical to apply on it real world data sets which take thousands of epochs to converge

with a BP algorithm trained classifier. Moreover, the optimal value of step size for a training an MLP is dependent upon the size and topology of a particular network. For a high dimensional data, its mathematical expression may become very complex which is computationally hard to resolve. The formula proposed in this work is simple to implement with almost no additional computational overhead except calculation of PCA after some iterations.

The second parameter that has been examined for the realization of *first objective* is the network selection. Constructive algorithms [182] increase the existing computational overhead of any algorithm. Therefore, destructive or pruning algorithms [203] have been given the priority for the network selection. The literature provides many successful algorithms for network pruning that involves pruning of hidden layers [117] [225], neurons[194][195], weights [199] and even input neurons for reducing the computational complexity of the design and improving the performance of the conventional BP algorithm. Recently, a weight pruning strategy has been implemented by Medeiros et al. [205] based on CAPE methodology. This technique has been considered as one of the most significant pruning strategies reported in the recent past and has outperformed OBS [189] and its variants in terms of computational cost. In this particular chapter, a weight pruning strategy has been proposed which is dependent upon the changing statistical properties of the weight set **(Chapter 3).** The weights are pruned according to their relevance which is defined in terms of complexity penalty and cost function. The complexity penalty term is defined for each weight and is further calculated considering the weight variation information during the BP training. The proposed algorithm has surpassed the CAPE in terms of its generalization performance and utilization of computational resources.

The third parameter that has been taken into consideration is the use of hybrid technique to train an ANN. In this, a recently proposed unsupervised technique by Widrow et al. [218] named HLMS have been used to train the hidden layers and supervised LMS is used for output layer. The main aim was to cluster the data at the output of final hidden layer using supervised learning and further on applying supervised learning for fine classification. HLMS is a new version of LMS which utilizes Hebb's teachings [226] by means of LMS algorithm. The benefit of using HLMS over other unsupervised techniques like K-means clustering [227] is that it does not require any prior information about the model parameters such as number of clusters. In fact, it requires only a

learning step value i.e. μ which is same as choosing step size for the supervised LMS. There are plenty of more benefits offered by HLMS which have been discussed in **Chapter 4**.

To achieve the second objective, a novel design of adaptive filter has been proposed. This design generated a set of vectorized delays eliminating the need to have a large number of storage registers for storing the delayed inputs. LMS adaptive filter and its sign variants [228] have been designed and implemented using two well known structures (direct and transpose form) and the proposed area efficient structure. The three structures have been analyzed using the critical path inspection. The structures have also been implemented targeting three different FPGAs to check the performance of the filters for resource utilization, silicon area, power dissipation and delay. NN based implementation of adaptive filter have also been analyzed in comparison to the conventional implementation approach **(Chapter 5)**.

The *last objective*, that is, the validation of the optimized parameters with standard tools has been achieved by analyzing and comparing the performance of the proposed methods against existing well-known approaches of the literature using Matlab. The performance of the PCA based learning rate adaptation for BP algorithm has been verified using several datasets with varying input attributes and varying number of classes. The analysis has been accomplished by comparing the performance with the conventional BP in terms of MSE and classification success rate. The weight pruning algorithm was implemented using the standard *trainlm* and *traingdm* functions of the Matlab and the performance was verified by comparing training, and generalization performance of the network trained using proposed pruning and conventional BP algorithm. The third algorithm which is design of an Adaptive Noise Canceller (ANC) using a novel adaptive filter structure was implemented first using Simulink. Later, Simulink HDL coder has been used to generate VHDL code of the design that is mandatory for implementing it on FPGA. This VHDL code was then simulated, synthesized using Xilinx 14.5 targeting a specific FPGA. The synthesized code is then implemented on three Xilinx FPGAs namely Spartan6, Virtex6 and Virtex7. The functionality of the implemented design on FPGA was then verified using Chipscope Pro.

1.8 Structure of Thesis

After providing an introduction to the thesis and literature review in Chapter 1, the rest of the thesis is structured as follows:

Chapter 2 discusses the PCA based technique along with a survey of the state of art in the utilization of PCA in training the ANNs. In this chapter, PCA based learning rate adaptation has been proposed, and the technique is compared with the conventional BP algorithm in terms of classification success rate and MSE using four benchmark datasets.

Chapter 3 presents the implementation of a computationally efficient weight pruning algorithm for NN design and optimization. Pruning of non relevant weights not only reduces the computational complexity but also improves the classification performance. The validation was done by comparing it with latest pruning techniques for training, generalization performance and computational complexity.

Chapter 4 details the implementation of an application of HLMS technique proposed by Widrow et al. [218] in which unsupervised Hebbian has been used to train the hidden layer, and supervised LMS algorithm trains the output layer. HLMS facilitates clustering of the input patterns in the hidden layer that helps faster learning of the output layer neurons using LMS algorithm. Simulation experiments have been performed with several benchmark datasets. The implementation issues related to this technique for multi layer NNs have also been discussed in this chapter.

Chapter 5 discusses the importance of efficient hardware implementation of an adaptive filter. A new area efficient structure has also been proposed and implemented for different variants of LMS algorithm for de-noising acoustic signals. The performance has been compared and analyzed considering hardware resource utilization, area, power and delay targeting three different FPGA's. It also presents the NN implementation of adaptive filter based on two different methodologies of BP and its comparative analysis with respect to the conventional filter design in terms of hardware estimation.

Finally, Chapter 6 gives the concluding remarks by highlighting the contributions of the work done. Future directions of the research have also been presented towards the end of this chapter. The highlights in brief the outcome of each chapter. It also underlines the methodological framework of this work, establishing a base where such analysis can be utilized for training DNNs in the future.

Chapter 2

Principal component analysis based learning rate adaptation

This chapter presents a novel technique to adapt the learning rate while training an ANN with BP algorithm. The adaptation is achieved by selectively incrementing the learning rate at every iteration based upon the information obtained from PCA of the output feature. This is because an optimally trained NN for classification problems has ideally low variance in the output since the output neuron corresponding to the parent class of the sample is supposed to fire at a value distinct to all other neurons in the output layer. On the other hand, misclassification is usually associated with a high variance in the output feature space with all the output neurons firing at random values. It is also proposed that learning rate would be varied only when the first three Principal Components (PC) of the input data space account for majority of the total variance. This is done to check the conditioning of the input data which describes the sensitivity of the error in the output to variations in the input data. Simulation experiments performed with various benchmark data sets reveal that the ANN trained with proposed technique exhibit better classification performance. Especially in one case, the improvement in testing phase success rate was found to be close to 26%.

2.1 Introduction

The BP algorithm is a standardized approach and can be utilized irrespective of the network topology and the input dimensionality [229]. The MLP training process and the weight specification is thus, aptly referred to as an optimization process where an error goal needs to be achieved [94]. The BP algorithm has been considered to be the most popular training approach that has the ability to solve numerous real-life problems. ANNs trained with BP algorithm are highly preferred due to their low complexity and ease of implementation. However, optimization using BP imposes a great risk of getting stuck to a local minimum [110], since the error surface is high dimensional. If the hunt for global minimum attains such a point in the search space and the algorithm will stop the search process further considering it to be the minimum point, while it would be desired that it continues towards a global minimum or at least a sufficiently low one.

Slow rate of convergence [109] and dependency on learning rate parameter [230] are the other two major issues associated with BP. However, the learning rate can be altered during the training process so as to achieve better results and hence a variety of algorithms for its adaptation have been found in literature [114][140][142][144][145][147] [153].

PCA (or the Karhunen-Lo`eve expansion) [223] is a technique widely used in the statistical community, primarily for descriptive [223][231] but also for inferential purposes. This technique is used to capture major variations in the data and reproduces strong patterns in the dataset. It helps to explore and visualize the multivariate data easily by representing the data onto a space of lower dimensionality, whilst attempting to preserve as much of the structural nature of the data as possible. This will help to analyze the data even on the computer screen by representing the information in much lesser dimensions than the original one (which is captured by the first two or three principal components) [232]. PCA has also now been used as a mechanism for dimensionality reduction in the weight space [233]. Consequently, it has been well studied, and standard references exist describing the method and its implementation.

Schenck et al. [234] used PCA to design an algorithm for adaptive learning rate control for Gaussian mixture models. Kompella et al. [235] used PCA for slow feature analysis which is used as a framework to extract features representing the underlying causes of the changes within a temporally coherent high-dimensional raw sensory input signal. Chitaliya et al. [233] have applied PCA in traffic surveillance system for reducing the extracted features of vehicle images that helped for identification of vehicles using Euclidian distance classifier and NN Classifier. Gallagher and Downs [236] proposed a method describing how visualization of the learning process in ANNs can be improved using PCA in order to observe the properties of the error surface along with the path taken by the network during training. They have also extended their algorithm for the visualization of high dimensional data which was difficult to achieve initially [232]. Abrahamsen and Hansen [237] have showed that dimensionality reduction by PCA and Kernel PCA (KPCA) can lack generalization due to small sample but high dimensional training data. So, they have provided a non-parametric renormalization scheme which can quite efficiently restore generalizability in KPCA. Lu et al. [238] have introduced a unique solution for unsupervised learning called Uncorrelated Multi-linear PCA (UMPCA), which has the capability of extracting uncorrelated features by successive variance maximization. Later on, Fan et al. [239] introduced

an Efficient KPCA (EKPCA) which is much more computationally efficient in extracting features than the original KPCA algorithm.

From the previous work, it is clear that apart from dimensionality reduction, PCA helps in visualizing the mappings produced by a network being trained. It is also evident from this work that progress of training through weight space can be depicted differently for different learning rates. However, for the same learning rates, the learning trajectory changes significantly when the first two PCs account for varying percentage of data variance. This has served as the motivation to devise a mechanism for varying the learning rate with changing percentage of data variance. However, the most novel aspect of the proposed technique is the application of PCA on the output data feature space rather than on the input data as done conventionally.

2.2 Back propagation algorithm and Principal component analysis

BP [93][94] also known as the generalized delta rule is a form of supervised learning algorithm for MLPs. Error (that compares the desired with the actual output) is fed back to the previous layers of the network in order to update the synaptic weights of all the previous network layers. Training an MLP is basically a two-phase process; first is the BP phase and second is the weight modification phase. Let us consider a set of P training patterns ($x^1. \ldots x^P$). Each input pattern x^p produces an output $y^p = (y_1^p \ldots . y_m^p)$ where m is the number of output neurons and p represents the output corresponding to p^{th} input pattern which ranges from 1 to P. The overall error E can be calculated by summing the errors generated by individual patterns and can be expressed mathematically as:

$$E = \sum_{p=1}^{P} E^p \qquad (2.1)$$

where E^p represents the error corresponding to the input pattern x^p

The most common error measure is the MSE and can be expressed as :

$$E^p = \frac{1}{2}\sum_{l=1}^{m}(d_l^p - y_l^p)^2 \qquad (2.2)$$

Where $d^p = d_1^p, \ldots \ldots \ldots \ldots d_m^p$ signifies the desired output corresponding to the input pattern x^p

Owing to the fact that *E* can be represented as a sum of the errors generated by individual patterns and both error terms *E* and E^p are functions of weights present in the network. Hence, the process of obtaining the derivatives of E with respect to the weights can be considered as calculating the derivatives of E^p with respect to the weights which is a much-reduced problem. The gradient ∇E of an error function *E* with respect to the weights consists of the partial derivatives $\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \ldots\ldots\ldots \frac{\partial E}{\partial w_w}$. The weights and biases of the network are initialized once before the commencement of the training process. Furthermore, a step size also called the learning rate parameter η must be chosen. The modification in the original set of weights can be done by adding a term $\Delta w(k)$ to the original weight term $w(k)$ where the modification term $\Delta w(k)$ is dependent on the learning rate parameter η. The simplest of these algorithms uses *gradient descent* approach for weight adjustment which updates the weights as follows:

$$w(k+1) = w(k) + \Delta w(k) \tag{2.3}$$

where $\Delta w(k) = -\eta \sum_{p=1}^{P} \nabla E^p$

Since this algorithm is based on gradient descent approach which implements a search for a local minimum, the chances for attaining the global minimum of the error function can be increased by executing several independent training procedures with randomly initialized weights. Another possibility would be to choose a more complex architecture with a larger number of weights, since the local minima are usually lower in this case. The BP algorithm provides an approximation to the trajectory in weight space computed by the method of steepest descent. The use of a smaller step size would result in smaller changes in the weights of the network from one iteration to the next, and hence provides a smoother trajectory in weight space. However, small values of step size or learning rate parameter lead to slower rate of learning. On the contrary, large value of step size parameter will speed up the training process but the resulting large changes in the weights of the network can make the network unstable [235].

PCA is a statistical technique which is generally used to map high-dimensional data onto a space of lower dimensionality [223]. This technique tries to capture the major variations in the data, by performing the equivalent rotation of the original data space. Using this approach, an orthogonal transformation of the original data set is being done so as to convert it onto a set of uncorrelated

variables called the PCs. The number of PCs obtained is less than or equal to the number of original variables. The transformation is defined in such a way that the first PC has the largest possible variance i.e. the PC corresponding to the largest principal value is in the direction of greatest variance in the data, and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to (i.e., uncorrelated with) the preceding components.

Let $w = (w_1, \ldots\ldots w_n)$ be a vector representing all the weights in a network, including bias weights. This vector defines a single point in weight space or equivalently on the error surface. Training a network for *s* epochs then produces a set of weight vectors $\{w^s\}$, which describes the trajectory followed by the learning process over the error surface. The usual coordinate system in weight space represents each *w* as a linear combination of a set of n orthonormal basis vectors

$$u_i w = \sum_{i=1}^{n} w_i u_i \tag{2.4}$$

which is transformed under PCA to form a new coordinate system given by

$$w = \sum_{i=1}^{n} z_i v_i \tag{2.5}$$

If we calculate the covariance matrix $\sum$ of the set of vectors $\{\boldsymbol{w}^s\}$

$$\Sigma = \sum (w^s - \bar{w})\,(w^s - \bar{w})^T \tag{2.6}$$

Then the PCs and principal values are given respectively by the eigenvectors and eigen values of the covariance matrix

$$\sum v_i = \lambda_i v_i \tag{2.7}$$

To reduce the dimensionality of any data to some value $d < n$, the *d* largest eigenvectors and their corresponding eigen values are chosen and remaining *n-d* values are discarded. In this work, PCA is applied on input and output features to obtain largest eigenvectors. However, no dimensionality reduction is done in the feature space. Instead, based upon the variance localization observed in the first few Principal Components, appropriate variation in learning rate is proposed. The following section formulates the problem of learning rate variation using statistics from input and

output feature spaces with justification of using information from PCA to modify the learning rate for better convergence.

2.3 Problem formulation

The problem of improving the rate of convergence and ability to ensure the global minimum of the BP algorithm has been investigated by a number of researchers [110] [114][115]. Some of the heuristic methods include adding a momentum term [94] to the original BP algorithm and standard numerical optimization techniques. However, the problems with these methods are the increased memory requirements and more computation time. Other algorithms for faster convergence like Recursive Least Square (RLS) [240], Extended Kalman Filtering (EKF) [241] have been proposed by authors but these are much more complex and difficult to implement in comparison to standard BP algorithm. Then, using lyapunov stability theory [242], an algorithm was proposed with accelerated convergence that is in many ways similar to the popular BP algorithm except for the fact that it employs adaptive learning rate. Furthermore, it has also been shown that by adapting the step size of the training algorithm, it is possible to drive out of a valley (local minimum) in the error surface by opposing the change in the direction of weight update [243].

A simple feed forward network with single neuron and a non-linear activation function is shown in figure 2.1. The training data consists of P patterns $\{x^p, y^p\}$, where p = 1, 2, ….., P and a weight vector $W \in R^n$. For a specific input pattern p with input vector x^p, the output is given as

$$y^p = f(W, x^p) \qquad p = 1,2, \ldots . P. \tag{2.8}$$

The instantaneous value of the error energy which is also termed as the cost function needs to be minimized so as to train the weight vector W and can be expressed as:

$$E = \frac{1}{2}\sum_{p=1}^{P}(d^p - y^p)^2 \tag{2.9}$$

where d^p signifies the desired output corresponding to the input pattern x^p

In order to analyze a weight update process where learning rate plays an important role, we first need to ensure global convergence of the process. Unlike in linear time-invariant systems, here, we cannot use transfer functions and pole positions to decide about stability of the training process. Hence, we apply Lyapunov stability theory which basically associates a 'function' which increases or decreases with the norm of the system vector according to a cost function under test.

Let us choose our very own cost function (error energy) as the candidate Lyapunov function for the system as follows:

$$E = \frac{1}{2}(\tilde{y}^T \tilde{y}) \tag{2.10}$$

where $\tilde{y} = [d^1 - y^1, \ldots\ldots, d^p - y^p, \ldots\ldots\ldots, d^P - y^P]^T$

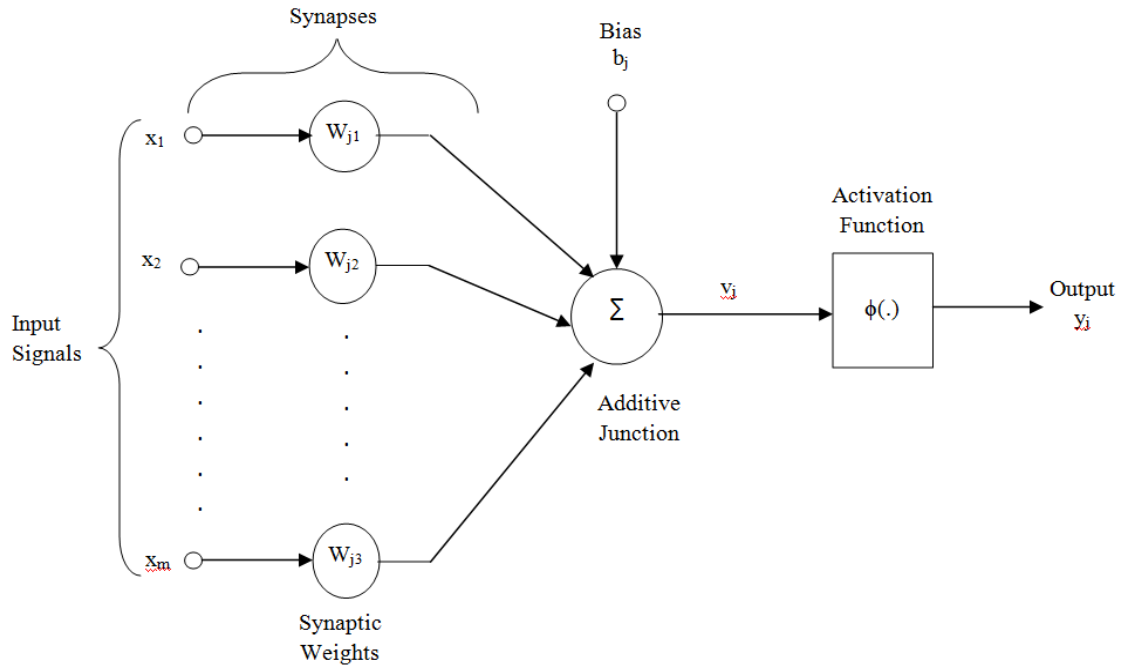


Figure 2.1: A single neuron model

In order to calculate the 'increment' or 'decrement' of the function we need to calculate time derivative of the Lyapunov function along the trajectories of the update process.

Theorem: The time derivative of the Lyapunov function is given by

$$\frac{\partial E}{\partial t} = -\tilde{y}^T J \frac{\partial W}{\partial t} \tag{2.11}$$

where $J = \frac{\partial y}{\partial W} \in R^{N \times n}$

Equation (2.11) can also be written as

$$\dot{E} = -\tilde{y}^T \frac{\partial y}{\partial W} \dot{w} = -\tilde{y}^T J \dot{w} \tag{2.12}$$

An arbitrary initial weight *w (0)* is updated over k iterations (replacing time steps t by k) as

$$w(k) = w(0) + \sum_{t=0}^{k} \dot{w} \tag{2.13}$$

where $\dot{\mathrm{w}} = \frac{\partial \mathrm{W}}{\partial \mathrm{t}} = \frac{\|\tilde{\mathrm{y}}\|^2}{\|\mathrm{J}^{\mathrm{T}}\tilde{\mathrm{y}}\|^2} \mathrm{J}^{\mathrm{T}}\tilde{\mathrm{y}}$

Then $\tilde{y}$ converges to zero under the condition that $\dot{W}$exists along the convergence trajectory. Using (2.13) in (2.11), we have

$$\dot{E} = -\|\tilde{y}\|^2 \leq 0 \tag{2.14}$$

where $\dot{E} < 0$ for all $\tilde{y} \neq 0$

The weight update law given in (2.13) is analogous to BP algorithm and using Lyapunov function, it can be derived as

$$\dot{w} = \frac{\partial W}{\partial t} = \frac{\|\tilde{y}\|^2}{\|J_p^T \tilde{y}\|^2} J_p^T \tilde{y} \tag{2.15}$$

where $\tilde{y} = d^p - y^p \in R$ and $J_p = \left[\frac{\partial y^p}{\partial W}\right] \in R^{1 \times m}$ is the instantaneous value of the Jacobian. Hence, the actual weight update equation is given by

$$w(k+1) = w(k) + \mu \dot{w}(k) = w(k) + \left[\mu \frac{\|\tilde{y}\|^2}{\|J_p^T \tilde{y}\|^2}\right] J_p^T \tilde{y} \tag{2.16}$$

where μ is a constant. Restating equation (2.16)

$$w(k+1) = w(k) + \eta J_p^T \tilde{y} \tag{2.17}$$

Comparing equation (2.17) with weight update equation of the standard BP algorithm given in (2.3), we get

$$\Delta w = -\eta \left(\frac{\partial E}{\partial w}\right)^T = \eta J_p^T \tilde{y} \tag{2.18}$$

The equations (2.17) and (2.3) are almost similar except the fact that the equation (2.3) has a fixed learning rate which is being replaced by its adaptive version η in equation (2.17) which is given by

$$\eta = \left(\mu \frac{\|\tilde{y}\|^2}{\|J_p^T \tilde{y}\|^2}\right) \tag{2.19}$$

2.3.1 Analytical Justification

Since, it is clear from the analysis presented above that adaptive learning rate is dependent on the Jacobian and Hessian of the error vector and error term is defined only for the output layer neurons. Therefore, any effort to make the learning rate adaptive utilizing error information should incorporate output neurons into the analysis.

Therefore, a heuristic for making the learning rate adaptive is being proposed where adaption is done by making the learning rate dependent upon changing statistical properties of the data using PCA. We have applied PCA on the output of the ANN during training phase in order to get an idea about the changing dynamics of the output feature space rather than concentrating only on the input feature space. In order to choose an optimum set of weights, it is important to look into the statistical properties of the weights converging on a hidden node.

The input to the hidden node is given by

$$y_j = w_{j1}x_1 + w_{j2}x_2 + \cdots w_{ji}x_i + b_i \tag{2.20}$$

where $x_i$is the i^{th} input and w_{ji} is the weight from i^{th} input to j^{th} hidden node. Also, the weights leading to a hidden node are regarded as random variables following normal distribution with mean m and variance v. Therefore, the mean of the input to the hidden node is given by

$$E\{y_i\} = E\{\sum_{i=0}^{n} w_{ji}x_i\} = \sum_{i=0}^{n}(E\{w_{ji}\}E\{x_i\} + Cov(\ w_{ji},\ x_i))$$ (2.21)

Replacing the covariance term by C, equation (2.21) can be written as

$$E\{y_i\} = \sum_{i=0}^{n} m.E\{x_i\} + C$$ (2.22)

The variance of y is given by

$$\sigma_y^2 = E\{(y)^2\} - E^2\{(y)\}$$ (2.23)

Using equation (2.21) in (2.23), we get

$$\sigma_y^2 = E\left\{\left(\sum_{i=0}^{n} w_{ji}x_i\right)^2\right\} - \left(\sum_{i=0}^{n} m.E\{x_i\} + C\ \right)^2$$ (2.24)

where, first expression of equation (2.24) can be calculated as follows:

$$E\left\{\left(\sum_{i=0}^{n} w_{ji}x_i\right)^2\right\} = \sum_{i=0}^{n}\left(E\left\{\left(w_{ji}\right)^2\right\}E\{(x_i)^2\} + cov\left({w_{ji}}^2, {x_i}^2\right)\right)$$ (2.25)

Replacing the covariance term by C_1, we get

$$E\left\{\left(\sum_{i=0}^{n} w_{ji}x_i\right)^2\right\} = \sum_{i=0}^{n} E\left\{\left(w_{ji}\right)^2\right\}.E\{(x_i)^2\} + C_1$$ (2.26)

The variance expression for any random variable x can also be written as

$$E\{(x)^2\} = E\{(x)\}^2 + \sigma_x^2(x)$$ (2.27)

where $\sigma_x^2(x)$ is the variance of random variable (x).

Input to hidden layer weight is regarded as a random variable with mean 0 and variance *a* and using equation (2.27) for input to hidden layer weight, we get

$$E\left\{\left(w_{ji}\right)^2\right\} = E\left\{\left(w_{ji}\right)\right\}^2 + \sigma_x^2\left(w_{ji}\right)$$ (2.28)

Therefore,

$$E\left\{\left(w_{ji}\right)^2\right\} = (m^2 + v)$$ (2.29)

Hence, the variance of the input to a hidden node is given by (using equation (2.24), (2.26) and (2.29)),

$$\sigma_y^2 = (\sum_{i=0}^{n}(m^2 + v).E\{(x_i)^2\} + C_1) - (\sum_{i=0}^{n} m.E\{x_i\} + C)^2 \quad (2.30)$$

where n denotes the number of weights converging on a particular node excluding the bias.

It has been clear from the above analysis that the variance of the incident weights is extremely important from convergence point of view and this variance is partly reflected in the PCA of the output feature. Thus, the information obtained from variance captured by prominent PCs can be used heuristically to alter the learning rate. Although, a standard formula for choosing an optimum value of learning rate is already given by Mandic in [139] which is given as follows:

$$\eta_{OPT}(k) = \frac{1}{2[\,\phi'\left(net_l^{(M)}(k)\right)^2 [\sum_{i\in L_{m-1}} z_i^2(k) + \cdots + \sum_{m\in L_1} \delta_m^2(k) \sum_{n\in L_0} x_n^2(k)]]} \quad (2.31)$$

for a multilayer ANN.

where ϕ is the activation function, z is the input to the i^{th} neuron in the $(L\text{-}1)^{th}$ layer, δ is the local gradient, x is the input vector, M is the output layer, L is the number of hidden layers, *net* signifies the activation of a neuron in the L^{th} hidden layer. The computational complexity of the formula being evidently high, it is not practical to apply on real world data sets which take thousands of epochs to converge with a BP algorithm trained classifier. Furthermore, the optimal adaptive learning rate for a general multilayer feedforward NN depends on the network size and topology. For a high dimensional data, its mathematical expression may become very complex which is computationally hard to resolve. Therefore, we apply the following logic to asymptotically obtain the η_{OPT} by capturing the essence of Eq. (2.31) in a computationally expedient and simple manner. A simplistic relationship between the optimum learning rate and the n-dimensional input feature for the k^{th} iteration can be stated as

$$\eta_{OPT}(k) = f\left(\frac{1}{\sum_{n\in L_0} x_n^2(k)}\right) \quad (2.32)$$

If we subject the input feature $x_n(k)$ to PCA, $x_n(k)$ can be completely described by $x_r(k)$ where, $r << n$. Therefore, high localization of variance in the first few principal components of the input

feature allows us to use a computationally simpler method. In a similar manner we can deduce from Eq. (2.31) that the optimum learning rate and the output features are inverse functions of each other, and the input and output features are in a multiplicative relationship.

We now propose a heuristic algorithm for obtaining a near optimal adaptive learning rate using the variance of the prominent PCs as depicted in figure 2.2. The network is first initialized with a small initial learning rate of 0.01(Line 2). PCA is applied on the input data so as to check the conditioning of the data which describes the sensitivity of the error in the output to variations in the input set. As we know, an ANN tends to get stuck in the local minima when encountered with an ill-conditioned (highly redundant) dataset. Incrementing the learning rate in that case is least likely to produce an optimum solution. However, when we are sure about the non-redundancy which can be checked by calculating the variance of first three PCs and if it accounts for the prescribed value of variance (Line 4), then the actual network training process starts (Line 6,7,8) with learning rate adaptation. Once the network is trained, PCA is applied on the output feature space (Line 10).

Input/Output:

1. Set of n-dimensional samples; $I = \{I_1, I_2 ... I_p\}$

2. Set of K classes; $C = \{C_1, C_2, ... C_k\}$

Initialization:

1. net= Neural network with one hidden layer

2. η (Learning rate) =0.01

3. V (variance localization) =98%

4. epochs=1000

Method:

Step 1: Apply PCA on I

Step 1.1: while $V > \in$

Evaluate/check V_3 i.e. %age variance covered by first 3 Principal components (PCs)

Step 1.1.1: if V_3 > V then

exit while **else**

V = V - 8%

Step 2: **If** V > ∈ then

Step 2.1: **For** i=1 to (epochs/10)

Step 2.1.1: net=train(net)

Step 2.1.2: output=sim(net)

Step 2.1.3: Apply PCA on output

Step 2.1.4: Update η after every 10 epochs such that

$$\eta(n+1) = \eta(n) + (A \times B)$$

where A: variance of I

B: variance covered by first two PCs of the output

Step 3: Evaluate the generalization performance

Figure 2.2: The proposed algorithm

Learning rate is incremented considering the variance of the input as well as the output feature space (Line11). If the first three PCs does not account for the prescribed value of variance required for well- conditioned data, the learning rate is not incremented. The learning rate updating formula has been proposed as:

$$\eta(n+1) = \eta(n) + A \times B \tag{2.33}$$

where *A*: variance of input data & *B*: Variance captured by first two principal components of the output.

An optimally trained NN for classification problems has ideally low variance in the output since the output neuron corresponding to the parent class of the sample is supposed to fire at a value distinct to all other neurons in the output layer (see Figure 2.3). However, misclassification is usually associated with a high variance in the output feature space with all the output neurons firing at random values (see Figure 2.4).

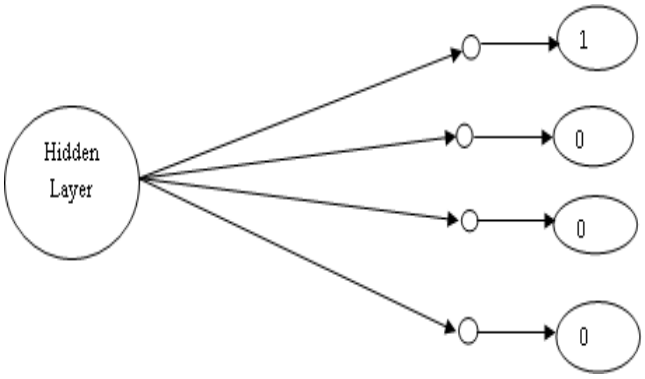


Figure 2.3: Optimum solution

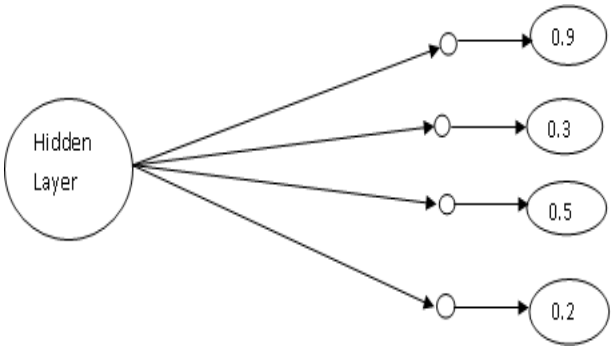


Figure 2.4: Non-optimum solution

After calculating the output from the trained NN, PCA is applied on the output data. If the variance captured by the first two PCs comes out to be a large value, this means output is highly uncorrelated which further implies a near optimal solution similar to the case shown in figure 2.3. Therefore, learning rate can be updated with a larger step size as the cost function is converging to the global minimum quickly. On the contrary, if the variance captured by the PCs is a small value which means non-optimum solution as shown in Figure 2.4 and the output is highly correlated, then the learning rate is required to be updated in small increments. The effect of learning rate parameter on generalization accuracy is an established fact since a too large learning rate often moves too far in the "correct" direction, that may result in overshooting a valley or minimum in the error surface, thus hurting accuracy. The parameter B has been introduced to fine tune successive increments in the learning rate based upon the variance of the outputs. If the variance of the outputs tends to zero, the learning rate at next iteration remains fixed at the initial value (which of-course is a rather small number). This methodology avoids the "overshooting" problem mentioned above.

The formula proposed in this work is simple to implement with almost no additional computational overhead except calculation of PCA after some iterations. We have chosen to increment the learning rate with the formula given in the algorithm if and only if the total (input) data variance captured by first three PCs is greater than $\in$ where $\in$ is percentage variance localization in first three principal components of the data. After rigorous experimentation and K-fold cross validation [244] with benchmark datasets, it was observed that improvement in generalization performance using the proposed technique were significant for those data sets which have around 55%-60% of the total variance localized in the first three PCs. Hence, learning rate is adapted only after checking the variance localization of the input data (algo. Line 3) whose default value is initialized at 98%. This requirement can be relaxed by decrementing the required variance localization in steps of 8% in a maximum of five successive steps. Thus, this algorithm can be expected to yield significantly better performance for data sets whose variance localization is $V > \in$=58%. Notwithstanding the lower bound on $\in$ the proposed technique exhibited improved performance with one of the data sets (Heart disease) whose variance localization was about 45%.

2.3.2 Description of dataset used

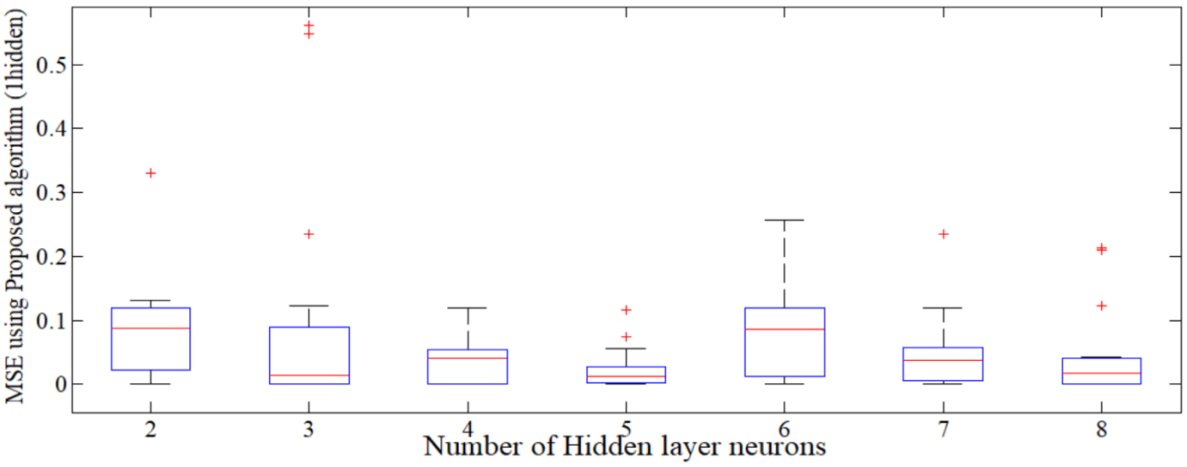
For the experiments, popular benchmark heart dataset from IDA repository was used. This dataset consists of 100 predefined splits into training and testing sets. Heart disease dataset contains 2 classes representing presence and absence of disease and has 13 attributes and 170 samples. Another dataset taken from the repository is the IRIS dataset which has 150 samples, 4 attributes and 3 classes, where each class refers to a type of iris plant and has equal data patterns. Out of 150 samples, 100 samples are used for training the NN and 50 samples are for testing. The attributes signify sepal length and width, petal length and width. Third dataset used is the gesture phase dataset with 18 attributes and 5 classes. The dataset consists of features extracted from 7 videos and each video consist of two files; one file contains the position of hands, wrists, head and spine of the user; and other file contains velocity and acceleration of hands and wrists. The 1227 samples from first file were taken for training and remaining for testing. Finally, the last dataset is the letter recognition dataset which has 16 attributes and 26 classes. It has 20,000 samples out of which 15,000 are used for training. The same data is shown in table 2.1.

2.4 Results and Discussion

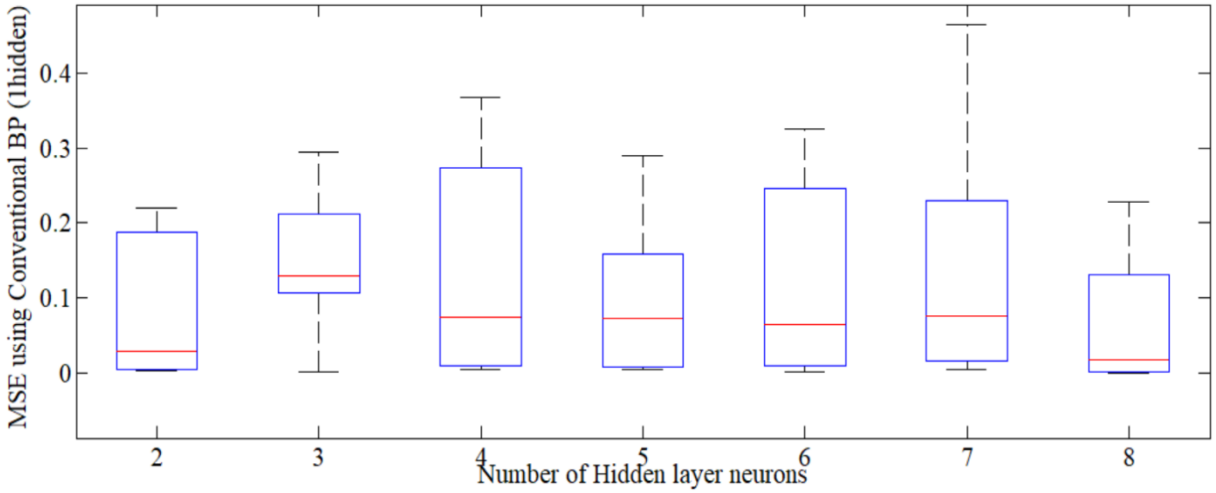
In this implementation, experiments were performed on four benchmark datasets for network with single and double hidden layer architecture and variation in the number of neurons from 2 to 8. The network was trained with the proposed algorithm and conventional BP algorithm for classifying the benchmark datasets. The datasets were first partitioned and 70% and 30% of the samples assigned to training and test sets respectively. Ten-fold cross validation was then performed to rule out over fitting. The number of hidden layer neurons was optimized experimentally both for single and double hidden layer NN. The variation with respect to number of training epochs was also included in the study with a range of 100-1000. The network was trained repetitively for 20 times for optimizing the architecture. The ANN's learning performance using the conventional and proposed algorithm for IRIS dataset with 1000 number of epochs is shown in the box plot in figure 2.5. Each box plot shows the MSE of the training phase with a variation in number of hidden layer neurons from 2-8. The plots clearly reveal that the optimum single and double hidden layer architecture for training using proposed algorithm is 4-5-3 and 4-4-4-3 respectively and for training using conventional algorithm is 4-8-3 and 4-2-2-3 respectively.

Table 2.1: Data description

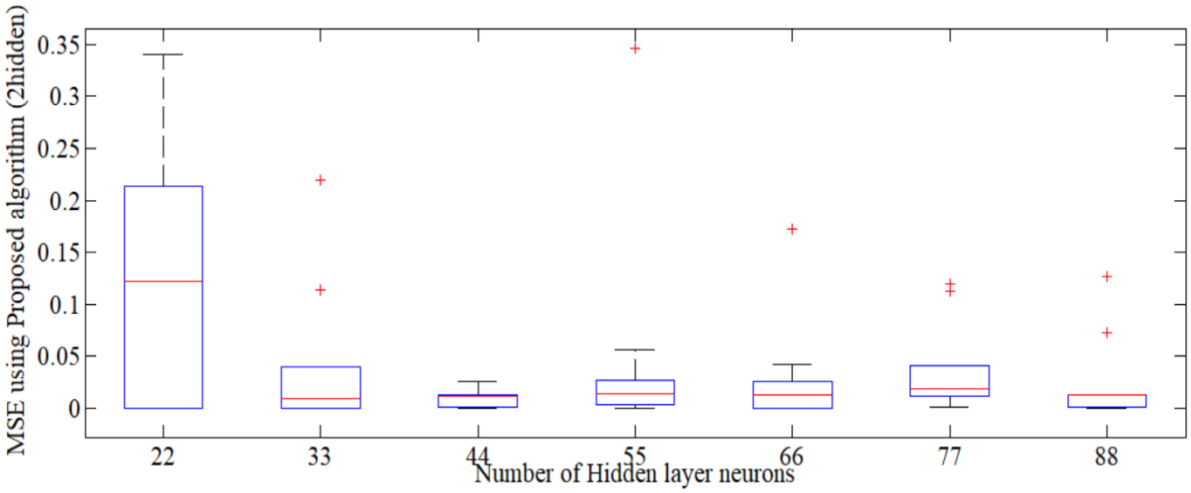
Name of the Dataset	**No. of Input dimensions**	**No. of Training Samples**	**No. of Test Samples**	**No. of Classes**
IRIS	04	100	50	03
Heart Disease	13	180	90	02
Gesture Phase	50	1227	520	05
Letter Recognition	16	15000	5000	26


MSE using Proposed algorithm (1hidden)
0.5
0.4
0.3
0.2
0.1
0
2
3
4
5
6
7
8
Number of Hidden layer neurons

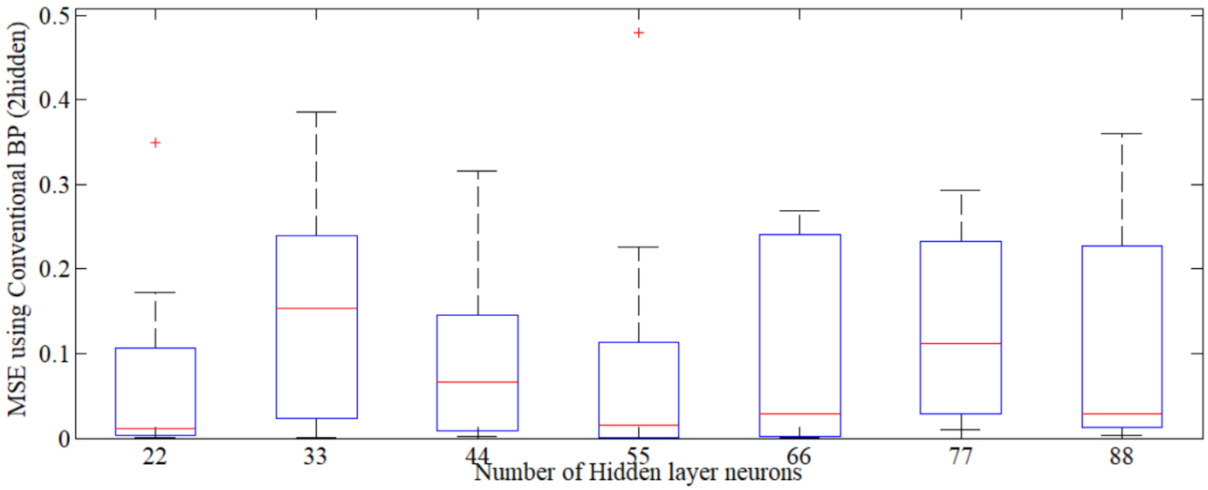
(a)


MSE using Conventional BP (1hidden)
0.4
0.3
0.2
0.1
0
2
3
4
5
6
7
8
Number of Hidden layer neurons

(b)


MSE using Proposed algorithm (2hidden)
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
22
33
44
55
66
77
88
Number of Hidden layer neurons

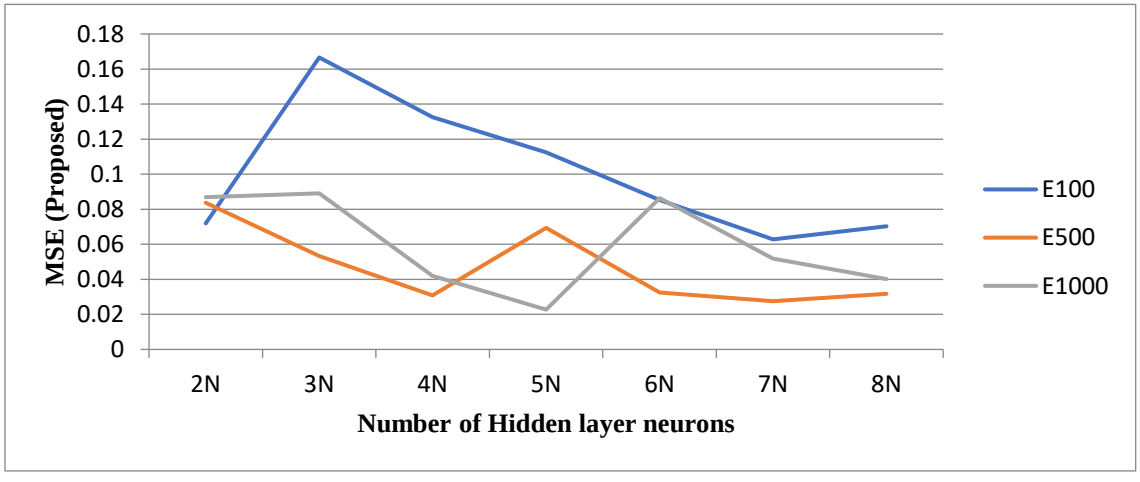
(c)



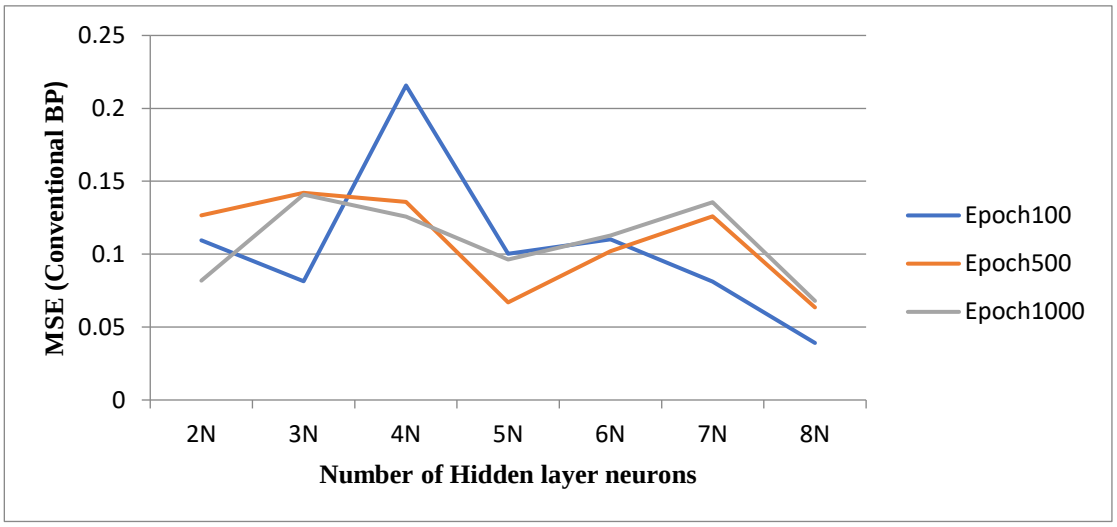
(d)

Figure 2.5: Box plot for the error obtained for IRIS dataset using (a) & (b) single and (c) & (d) double hidden layer and using conventional and proposed algorithm for training with 1000 epochs

The impact of increasing the number of epochs on learning using proposed and conventional BP algorithm is shown in figure 2.6. The graph dictates that the best learning performance using conventional BP could be achieved only with maximum number of epochs and more number of hidden layer neurons whereas it is not the case with the proposed algorithm which shows the best training behavior at lower number of epochs and less number of hidden layer neurons as well.



(a)

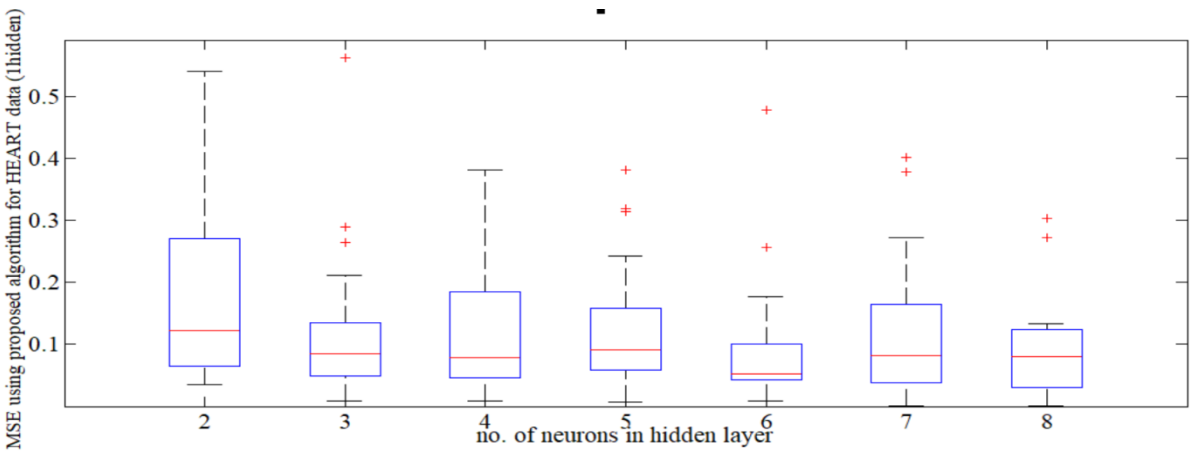


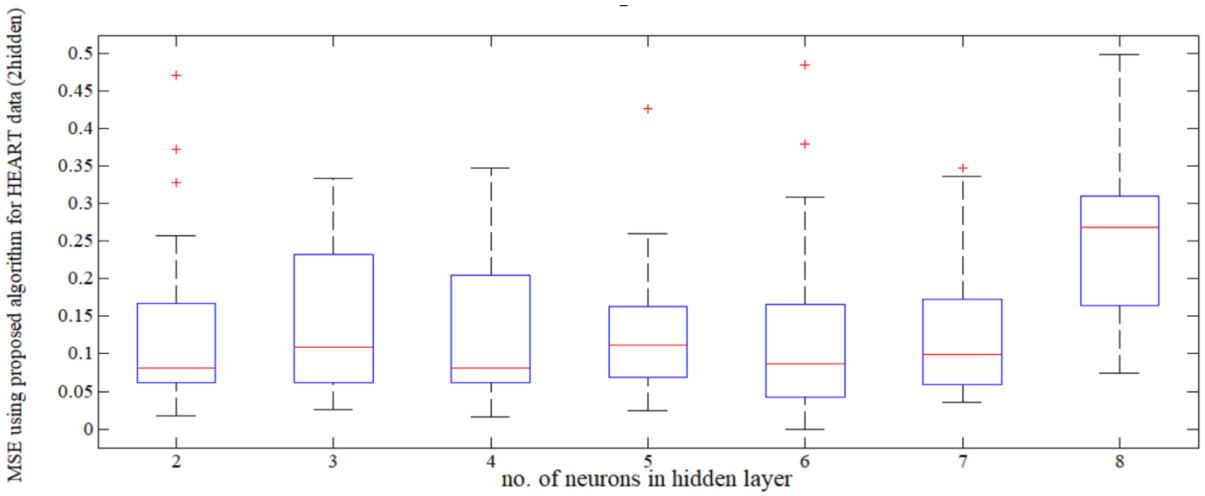
(b)

Figure 2.6: Variation of MSE with number of epochs for IRIS dataset: (a) using proposed algorithm (b) Conventional algorithm

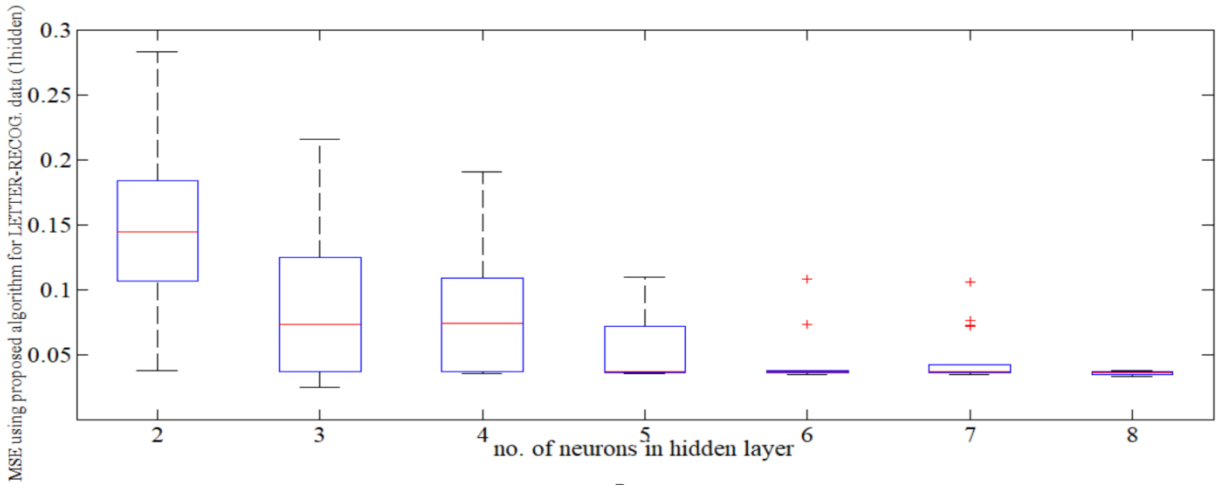
The network's training performance for optimizing the hidden layer neurons was also analyzed for the remaining datasets and using single and double hidden layer NN architecture and the network architecture with minimum training error for all datasets was found from the results shown in the box plot of figure 2.7. Figure 2.8 depict the error surface obtained after training the NN with proposed algorithm for single and double hidden layer architecture for all four datasets.

(a)

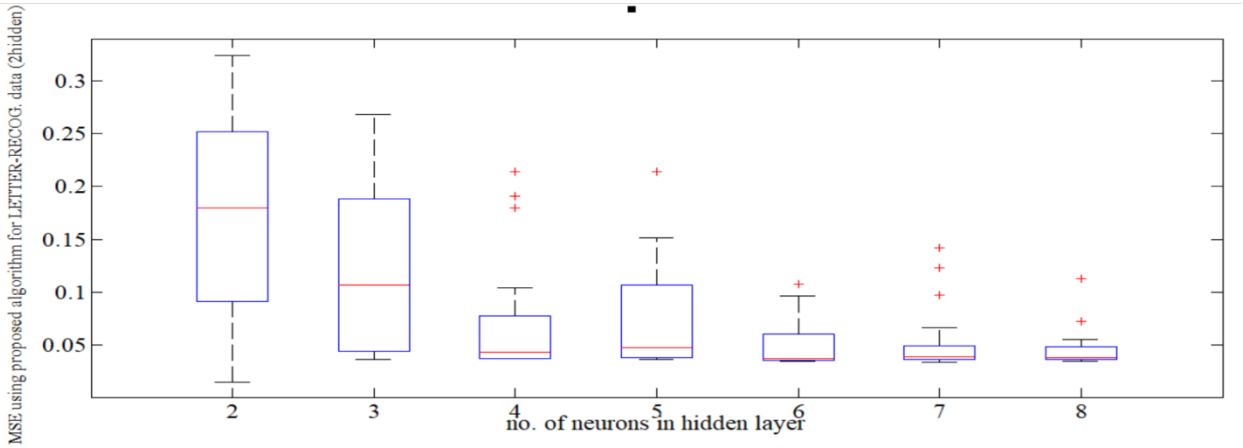



MSE using proposed algorithm for HEART data (2hidden)
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
2
3
4
5
6
7
8
no. of neurons in hidden layer

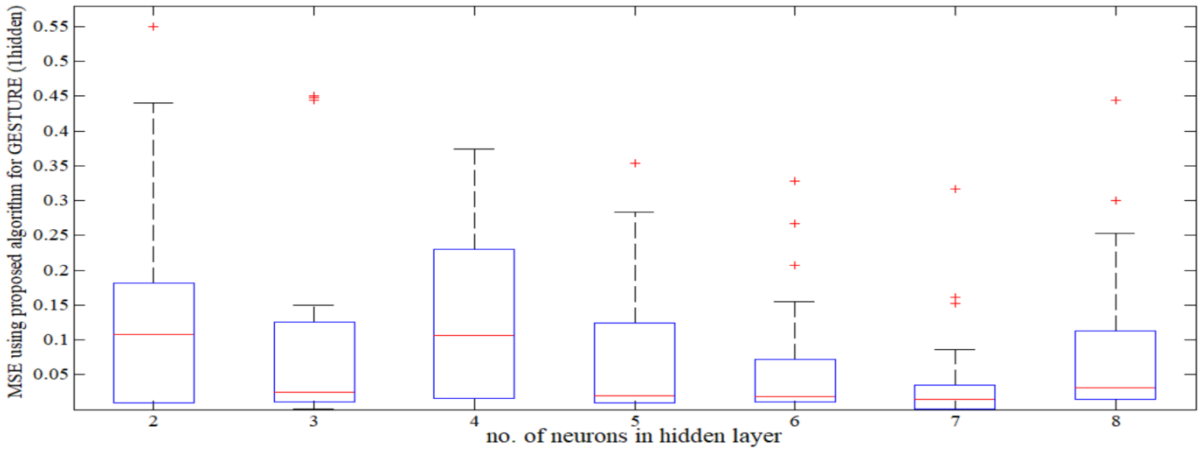
(b)


MSE using proposed algorithm for LETTER-RECOG. data (1hidden)
0.3
0.25
0.2
0.15
0.1
0.05
2
3
4
5
6
7
8
no. of neurons in hidden layer

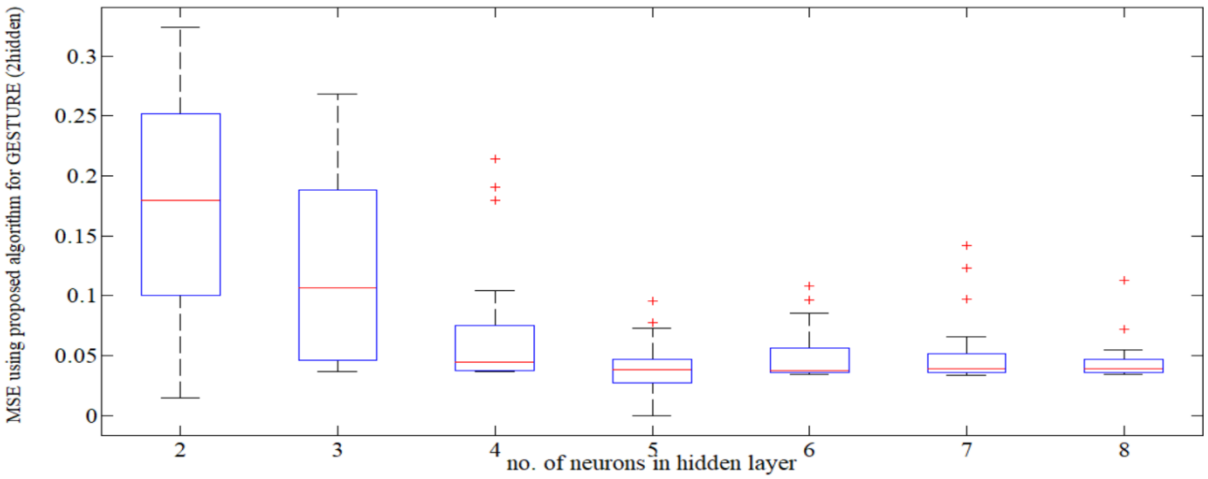
(c)


MSE using proposed algorithm for LETTER-RECOG. data (2hidden)
0.3
0.25
0.2
0.15
0.1
0.05
2
3
4
5
6
7
8
no. of neurons in hidden layer

(d)


MSE using proposed algorithm for GESTURE (1hidden)
0.55
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
2
3
4
5
6
7
8
no. of neurons in hidden layer

(e)


MSE using proposed algorithm for GESTURE (2hidden)
0.3
0.25
0.2
0.15
0.1
0.05
0
2
3
4
5
6
7
8
no. of neurons in hidden layer

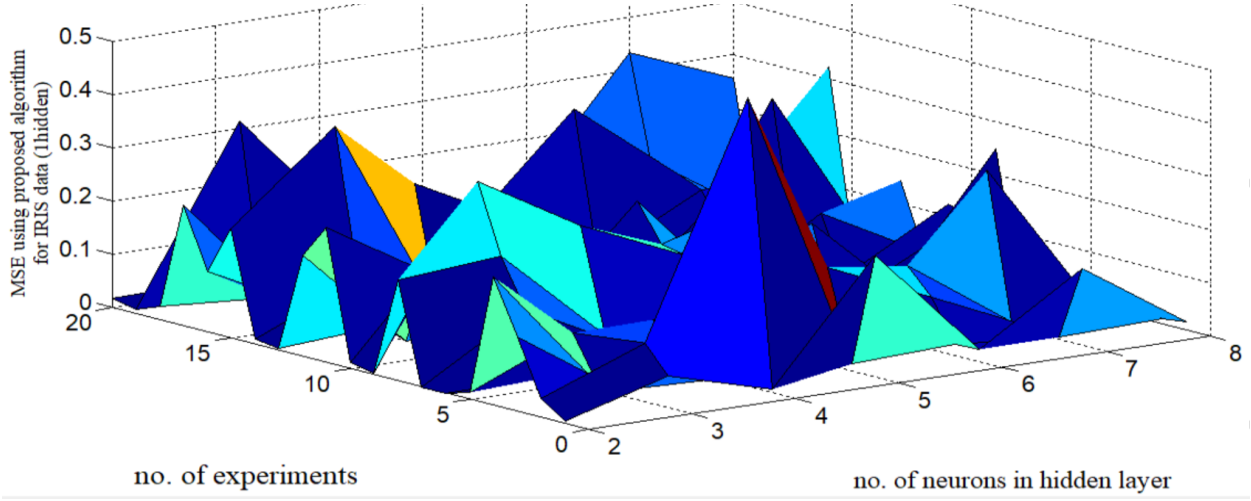
(f)

Figure 2.7: Box plot for the error obtained for different datasets using single and double hidden layer and using proposed algorithm (a)-(f)

However, the performance of the ANN cannot be analyzed only with the learning outcome. Both learning and testing performance have to be analyzed in unison. The testing phase performance is analyzed by means of R^2 coefficient which is defined as

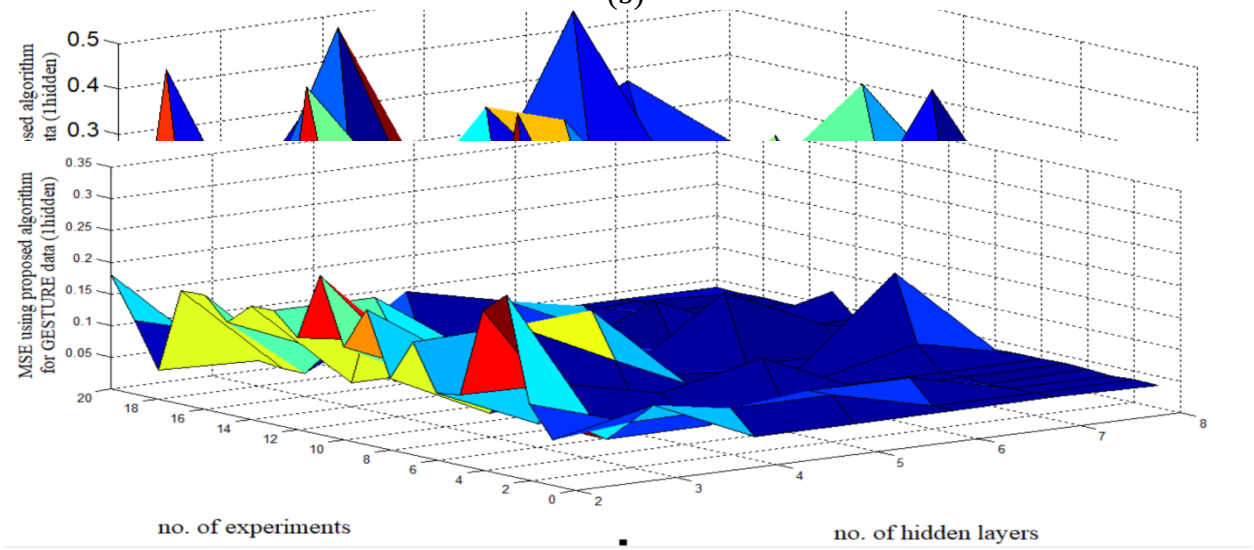
$$R^2 = 1 - \frac{\sum_{p=1}^{p=P}(t_p - y_p)^2}{\sum_{p=1}^{p=P}(t_p - \bar{t}_p)^2} \tag{2.33}$$

where t_p is the target value for a particular sample, y_p is the actual output for that input pattern and $\bar{t}_p$ is the mean of target values.

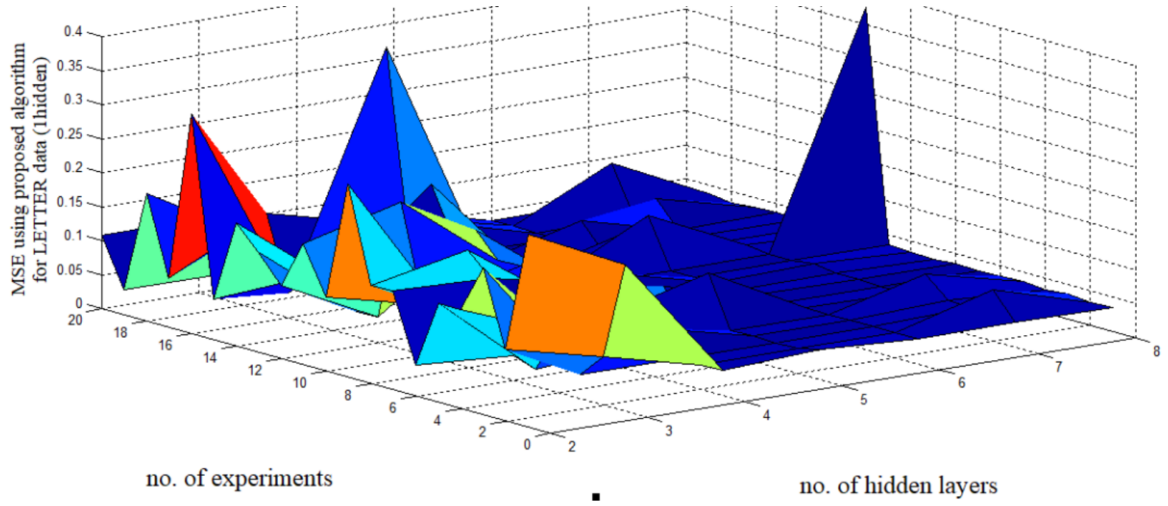


(a)

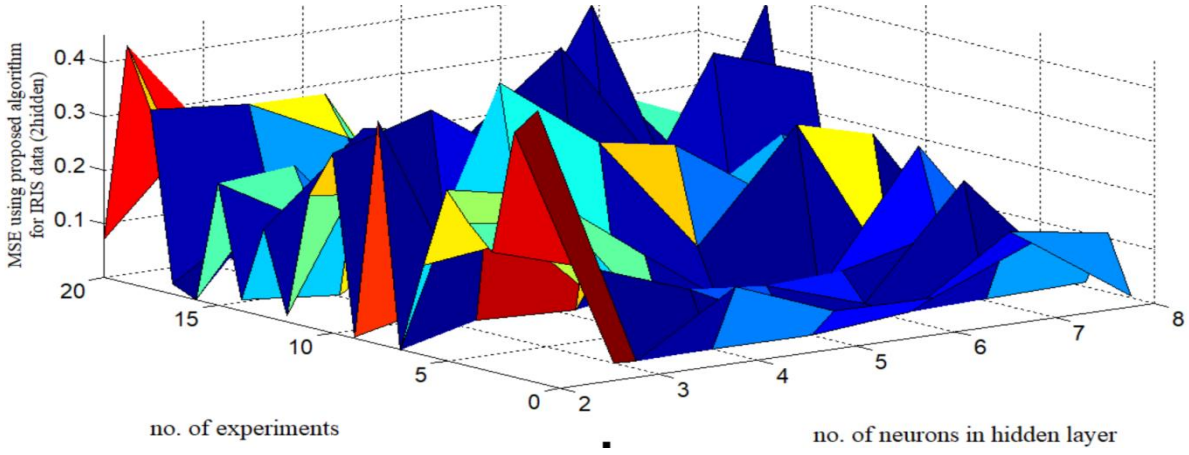
(b)


0.5
0.4
0.3
MSE using proposed algorithm for GESTURE data (1hidden)
0.35
0.3
0.25
0.2
0.15
0.1
0.05
20
18
16
14
12
10
8
6
4
2
0
2
3
4
5
6
7
8
no. of experiments
no. of hidden layers

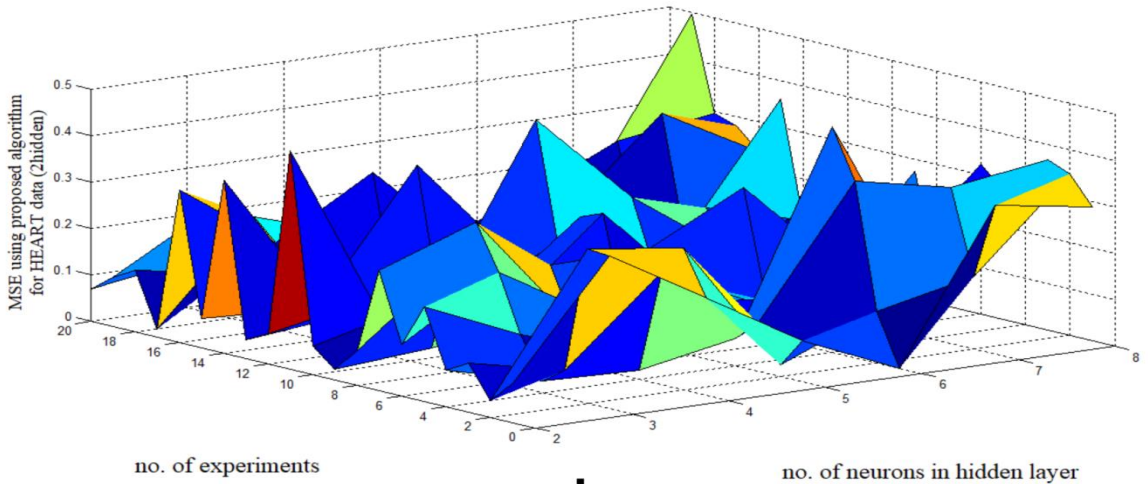
(c)


MSE using proposed algorithm for LETTER data (1hidden)
no. of experiments
no. of hidden layers

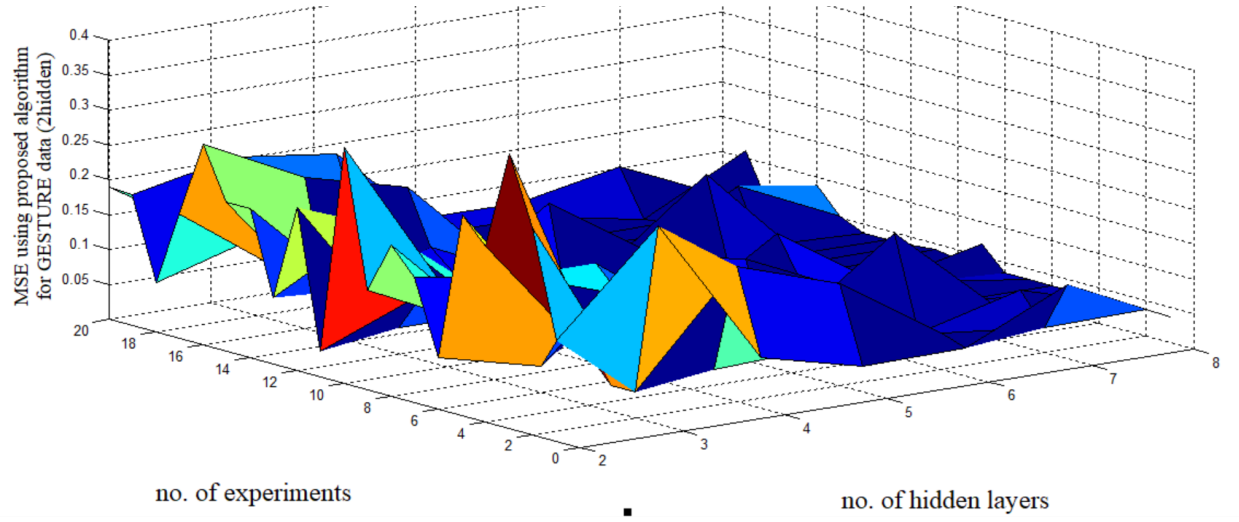
(d)


MSE using proposed algorithm for IRIS data (2hidden)
no. of experiments
no. of neurons in hidden layer

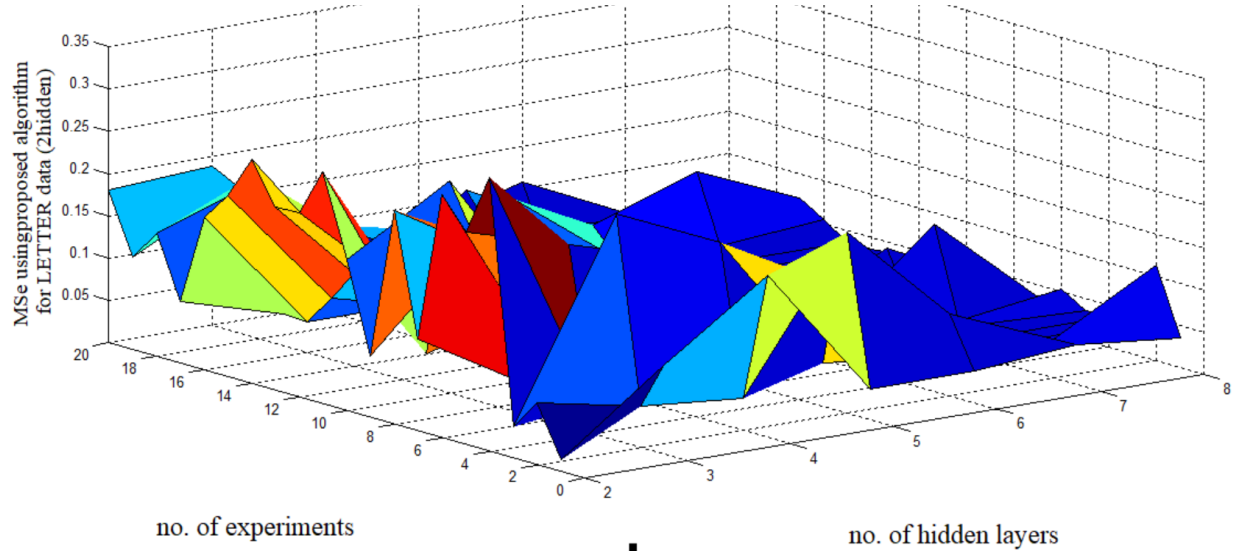
(e)


MSE using proposed algorithm for HEART data (2hidden)
no. of experiments
no. of neurons in hidden layer

(f)



(g)



(h)

Figure 2.8: Surface plots for the error obtained for different dataset using single and double hidden layer and using proposed algorithm (a)-(h)

The training phase is evaluated by MSE. Tables 2.2 and 2.3 summarize the performance of single and double hidden layer architectures trained by conventional and proposed algorithm in terms of

MSE and R^2. The tables include all the details including the variations with respect to number of training epochs and hidden layer neurons. The optimum network architecture can be found by comparing the outcome of table 2.2 and 2.3. The architecture which provides minimum MSE and maximum R^2 value with minimum number of neurons in the hidden layer and in least number of training epochs will be the network of choice

Table 2.2: MSE with variation in number of epochs and Hidden Neurons (HN) for IRIS dataset

			MSE						
Algorithm	**Architecture**	**Epochs**	**2 HN**	**3 HN**	**4 HN**	**5 HN**	**6 HN**	**7 HN**	**8 HN**
Conventional BP	Single Layer	E100	0.10943	0.08134	0.2157	0.10035	0.11028	0.08118	0.03916
		E500	0.12658	0.14207	0.13585	0.06702	0.10204	0.12587	0.06351
		E1000	0.08187	0.14085	0.12585	0.09634	0.11289	0.13567	0.06798
	Double Layer	E100	0.11149	0.11284	0.10744	0.13396	0.12967	0.07993	0.08863
		E500	0.09903	0.16514	0.06013	0.08543	0.13897	0.08931	0.05371
		E1000	0.07011	0.15633	0.09252	0.08849	0.09776	0.12919	0.09699
Proposed	Single Layer	E100	0.07198	0.16657	0.13258	0.11241	0.08525	0.06287	0.07031
		E500	0.08369	0.05320	0.03083	0.06933	0.03236	0.02759	0.03173
		E1000	0.08684	0.08908	0.04202	0.02273	0.08632	0.05191	0.04009
	Double Layer	E100	0.11163	0.11369	0.04560	0.05377	0.06119	0.07428	0.05566
		E500	0.12053	0.042417	0.031747	0.055947	0.03685	0.02539	0.06898
		E1000	0.119955	0.04296	0.01062	0.05010	0.02887	0.03662	0.02552

Table 2.3: Rsquare with variation in number of epochs and hidden neurons for IRIS dataset

			R^2						
Algorithm	**Architecture**	**Epochs**	**2 HN**	**3 HN**	**4 HN**	**5 HN**	**6 HN**	**7 HN**	**8 HN**
Conventional BP	Single Layer	E100	0.48977	0.62516	0.03696	0.50897	0.5336	0.63463	0.8299
		E500	0.32174	0.3389	0.33988	0.68259	0.5198	0.4421	0.6972
		E1000	0.59914	0.40152	0.43358	0.5549	0.52374	0.3916	0.7666
	Double Layer	E100	0.4518	0.4549	0.4968	0.41531	0.46278	0.61316	0.5932
		E500	0.54227	0.2711	0.69667	0.59565	0.4055	0.5653	0.73226
		E1000	0.65371	0.2908	0.565	0.57104	0.53015	0.44712	0.46419
Proposed	Single Layer	E100	0.63667	0.40559	0.40342	0.51771	0.59	0.69742	0.68281
		E500	0.66496	0.7603	0.8575	0.7363	0.8706	0.8715	0.8639
		E1000	0.5846	0.8375	0.793222	0.728	0.729	0.86	0.88228
	Double Layer	E100	0.47374	0.48915	0.799	0.74619	0.61814	0.6991	0.7627

	E500	0.47716	0.825	0.84992	0.7531	0.9003	0.86978	0.6946
	E1000	0.42426	0.8195	0.958	0.96235	0.896	0.8455	0.8955

Table 2.4: Optimum architecture analysis of IRIS dataset

Algorithm	**Architecture**	**Epochs**	**R^2**	**MSE**
Proposed	4-5-3	1000	0.728	0.022
Proposed	4-7-3	500	0.8715	0.027
Proposed	4-6-3	500	0.8706	0.032
Conventional BP	4-8-3	100	0.829	0.039
Proposed	4-4-4-3	1000	0.958	0.0106
Conventional BP	4-8-8-3	500	0.732	0.0537

Table 2.4 represents the performance of six best architectures of the whole simulated dataset for IRIS data. The top two architectures mentioned in table 2.4 have similar performance in the training phase. Both are single hidden layer NNs trained using the proposed algorithm. Despite having similar performance in the training phase, testing phase performance is worst for the one with lesser hidden layer neurons trained with comparatively a greater number of epochs. The third architecture in the list is a compromise between the first two in terms of MSE and number of hidden layer neurons. Its training performance is slightly poorer than the two, but the testing phase performance is far better than the first architecture. Hence, architecture 4-6-3 has been chosen to be the optimum single hidden layer ANN trained with proposed algorithm. The next favorable architecture is the single hidden layer NN trained using conventional BP algorithm. Considering performance using all the factors, only 4-8-3 structure gives minimum MSE, maximum R^2 with least number of epochs. The similar analysis was carried out for double hidden layer NN trained using proposed and conventional algorithm. It was noted that 4-4-4-3 architecture is found optimum for training using proposed algorithm and 4-8-8-3 using conventional algorithm. The results clearly reveal that network trained with conventional algorithm provide better performance only with maximum number of hidden layer neurons which in turn leads to increased

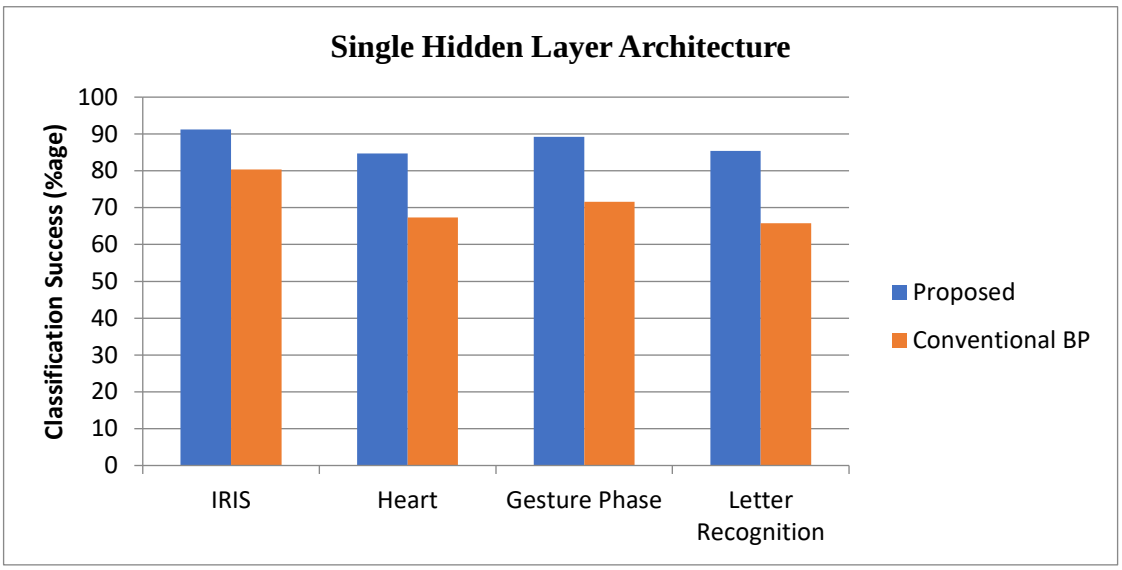
computational resources which is not the case with the network trained with proposed algorithm. The same experiment was also performed on other datasets given in table 2.1 using proposed and conventional BP algorithm and using single and double hidden layer network. The results for the optimized architecture for all the datasets are tabulated in table 2.5.

The testing phase was further evaluated by analyzing the generalization performance of both the algorithms. Hence, average of percentage classification obtained in each fold has been chosen as the performance metric. The testing phase performance of the proposed and conventional network is summarized in tables 2.5(a) and (b) respectively. The classification performance comparison both for proposed and conventional algorithms are also depicted in the form of a bar graph as shown in figure 2.9(a) & (b). It is clear from the results shown in figure 2.9 that the proposed algorithm achieves better generalization performance than the conventional BP algorithm for all the datasets. The algorithm performed exceptionally well for the datasets where the number of classes and test samples were very large.

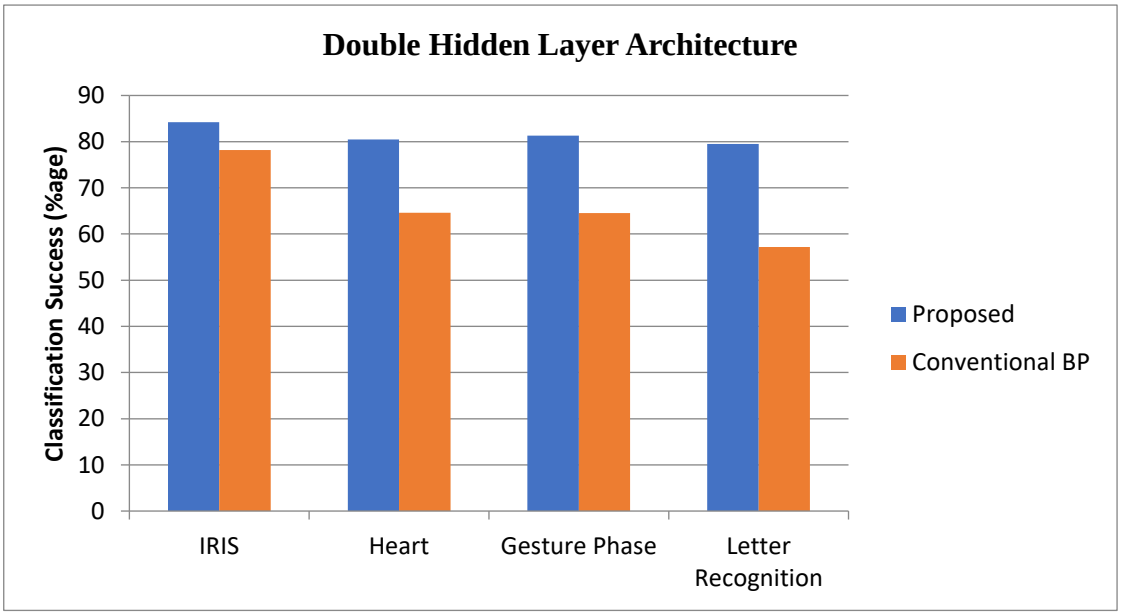
Table 2.5: Testing phase performance of the proposed and conventional algorithm

(a) Proposed algorithm (b) Conventional algorithm

Optimum Network Architecture	**Data set**	**Classification Success (%age)**	**Optimum Network Architecture**	**Data set**	**Classification Success (%age)**
4:6:3	IRIS	91.2	4:8:3	IRIS	80.4
13:6:2	Heart	84.7	13:7:2	Heart	67.3
18:7:5	Gesture Phase	79.2	18:8:5	Gesture Phase	71.6
16:6:26	Letter Recognition	75.4	16:8:26	Letter Recognition	65.8
4:4:4:3	IRIS	84.2	4:8:8:3	IRIS	78.2
13:6:6:2	Heart	80.5	13:8:8:2	Heart	64.6
18:5:5:5	Gesture Phase	75.3	18:7:7:5	Gesture Phase	64.5
16:6:6:26	Letter Recognition	69.5	16:8:8:26	Letter Recognition	57.2



(a)



(b)

Figure 2.9: Bar graph for the classification success rate obtained for different dataset and for both proposed and conventional algorithm: (a) using single hidden layer architecture; (b) using double hidden layer architecture

The results were further evaluated using paired sample t-test to assess the effectiveness of the proposed algorithm for different network architectures and using different datasets. In the current study, the null hypothesis assumes that there is no significant improvement in the classification

success rate of the proposed algorithm relative to the conventional algorithm. Conversely, the alternate hypothesis assumes a significant improvement in the success rate using the proposed algorithm. In our case, 0.05 level of significance (α) was considered. If the P-value is less than (or equal to) α, then the null hypothesis is rejected in favor of the alternative hypothesis. And, if the P-value is greater than α, then the null hypothesis is not rejected. Data was taken as the classification success rate of proposed and conventional algorithm for 20 set of experiments. The results of the t-test are summarized in Table 2.6 which clearly indicates that for every dataset either single hidden layer or double hidden layer, null hypothesis is rejected which justifies the efficacy of the proposed technique.

Table 2.6: T-test results

Network Architecture	**Dataset**	**P-value (α=0.05)**	**Null Hypothesis**
Single Layer	IRIS	0.0133	Rejected
	Heart	0.002	Rejected
	Gesture Phase	0.0256	Rejected
	Letter Recognition	0.0325	Rejected
Double Layer	IRIS	0.02434	Rejected
	Heart	0.0186	Rejected
	Gesture Phase	0.0154	Rejected
	Letter Recognition	0.0023	Rejected

2.5 Summary

A novel technique has been proposed and implemented to make the learning rate adaptive using the information obtained from the PCA of the output feature space. The results shown in tables 2.5 & 2.6 reveal the efficacy of the proposed technique. Networks with varying architectures have been trained using several data sets with variation in the number of attributes and classes. However, the testing phase performance has been found to be better significantly in all the cases. It can be observed that with IRIS data set, the performance of the proposed classifier was only marginally better. This can be attributed to inherent simplicity of the IRIS data set where even conventional

classifiers perform well. However, with all the remaining datasets the improvement has been found significant. Since, the number of output classes in case of HEART data was two, it is expected that the proposed technique may give better performance for binary classification problems only. In addition to this, the classification performance becomes better with higher number of output classes as clearly shown in figure 2.9. The proposed technique for learning rate variation applies the advantage of PCA to output feature space for assessing the requirement of enhanced learning.

The next chapter focuses on the design and implementation of a novel pruning technique with an aim to reduce the computational complexity of the NN design making it suitable for pruning massively connected networks (e.g., deep networks) in future.

Chapter 3

A computationally efficient weight pruning algorithm for ANN classifiers

This chapter describes a novel technique to prune the weights of ANN while training with BP algorithm. Iterative update of weights through gradient descent mechanism does not guarantee convergence in a specified number of epochs. Pruning of non-relevant weights not only reduces the computational complexity but also improves the classification performance. This algorithm

first defines the "relevance" of initialized weights in a statistical sense by introducing a coefficient of dominance for each weight converging on a hidden node and subsequently employing the concept of complexity penalty. Based upon complexity penalty for each weight, a decision has been taken to either prune or retain the weight. It has been shown analytically that a weight with higher complexity penalty has a high degree of Fisher information which further implies its ability to capture the variations in the input set for better classification. Simulation experiments performed with five benchmark data sets reveal that ANNs trained after being pruned using the proposed technique exhibit higher convergence, lower execution time and higher success rate in the test phase and yields substantial reduction in computational resources. For complex architectures, early convergence was found to be directly correlated with percentage of weights pruned. The efficacy of the technique has been validated on several benchmark datasets having large diversity of attributes.

3.1 Introduction

ANNs trained with BP algorithm [94] are highly popular due to their low complexity and ease of implementation. However, the algorithm suffers from two major drawbacks owing to the use of gradient descent approach [94]; one is the learning process which is prone to getting stuck in local minima thereby yielding suboptimal solutions and other is extremely slow convergence [245]. The convergence issue of gradient descent has been addressed by many researchers but recently Khazaei et al. [231] have proposed a new learning algorithm for approximating non- linear systems. This method has helped to achieve much faster convergence with reduced root MSE. However, there are additional difficulties associated with finding an optimal architecture for a given task. Furthermore, BP works well on simple training problems but its performance falls off rapidly as the dimension or complexity (high correlation in the input samples) of input data increases [111]. The problem of training a network using BP has two aspects. First, the task of finding the optimal number of hidden layer neurons. Second, the determination of weights and biases of the ANN. Specifying the weights of a NN is mostly viewed as an optimization process,

where an error goal needs to be achieved. However, optimization using BP imposes a great risk of getting stuck in a local minimum, since the error surface might not be convex. It is also clear that all the weights do not contribute equally to pattern classification. Thus, there are some set of weights which have less or no impact on the pattern separability. These type of weights are redundant and are referred to as excess weights [246]. Therefore, the primary issue addressed in this work is elimination of redundant weights. This aspect has been addressed by many researchers in a plentitude of ways. A pruning strategy for Extreme learning machine has been proposed using the Successive Projections Algorithm (SPA), as an approach to automatically find the number of hidden nodes [247]. Several pruning strategies have been proposed to find a compact structure of feedforward NNs with high generalization ability in classification problems [117] [203]. Pruning is a crucial step in "network model selection" which is still a popular topic of research and can be defined as the task of identifying the optimal architecture which not only trains well but also generalizes well. However, if we assume that the network is not over fitting, early convergence also foretells possibly high success rate in the generalization phase which is also called early stopping. Although, this method can provide a noisy estimate of generalization performance if the validation set is small [248]. Pruning is one of the methods to provide smallest generalization error by finding the optimal architecture. The current scenario in neural computing has been dominated by DNNs having hundreds of millions of connections making their implementation computationally and memory intensive. Han et al. [249] proposed an energy efficient interface engine for deep compression which works on a "weight sharing" scheme which in turn is quite similar to weight pruning. Similar pruning strategies for deep compression and convolutional networks have been efficiently implemented on hardware [250]–[253]. However, a general-purpose pruning strategy is easier to validate on BP trained ANNs before it could be adapted to prune a DNN. A brief overview of the relevant prior work on weight pruning is presented in the next section.

3.2 Related work

A comprehensive review of network pruning algorithms in the early stages of development of ANN training paradigms has been presented by Reed in [186]. More recently, researchers have employed statistical input pruning for environmental modeling [254]. Sabo et al. [255] have proposed a simple pruning algorithm based upon cross validation and sensitivity analysis in order

to get an optimal NN topology which results into a significantly lower computational complexity. Constructive algorithms have been reported in [117][181] which start from a leanest possible architecture and add neurons during training phase to arrive at an optimal one. Xiang et al. [256] have proposed the relation between the geometrical shape of a function and an optimal architecture for its approximation. Trenn in [257] throws valuable insights in finding an optimal ANN architecture based on the order of the approximated function. Augasta et al. [203] have conducted a survey of existing pruning techniques that optimize the architecture of NNs. They have also evaluated the effectiveness of various pruning techniques based on sensitivity analysis, mutual information and significance on four real datasets. Apart from network and weight pruning approaches, researchers are also opting for statistical methods to prune the input features for simpler network architecture since the number of input layer neurons is equal to the input data dimension. Fan *et al.* [258] have employed $L_{1/2}$ regularization method to prune Extreme Learning Machines (ELM). The authors have applied a preprocessing stage to determine the suitable number of hidden nodes before running the ELM. GA based search has been employed by Mantzaris *et. al.* in [259] to eliminate excess weights from an ANN used to estimate Urinary Tract Infection (UTI) from medical examination data. Workers have also reported novel evolutionary algorithms for pruning ANNs constructed for prediction applications [260]. However, evolutionary algorithms require significant computational effort especially for large architectures hence they are less attractive for model selection purposes. In this course of development, OBS algorithm first proposed by Hassibi & Stork [189] requires special mention since it employs computation of weight saliency which in turn ensures error minimization. However, inverting the Hessian matrix, as required in OBS, is a computationally massive task. Christiansen *et al.* [261] have applied OBS algorithm for automatically generating a topologically optimized ANN based upon the statistical properties of the input data set. The Weight Decay and Elimination (WDE) algorithm proposed by Weigend et al. in [262] introduces a regularization method which modifies the error function itself by introducing a penalty term. The merit of this approach lies in the fact that relevance of individual weights is not only defined in terms of MSE but also in terms of complexity of the network as a function of the weight magnitudes. Gomez *et al*. [263] establishes the role of function complexity in ANN architecture selection. Similarly, Nakamura *et al.* in [264] have compared several information criteria employed in model selection. The above two references underline the importance of other criteria apart from MSE for effective model selection. Etminani et al. [265]

have proposed several pruning strategies in order to reduce the number of nodes created in the branch and bound tree.

3.2.1 Need of improvement

Medeiros and Barreto in [205] have proposed a novel weight pruning methodology for MLP classifiers. The proposed method is based on the observation that relevant synaptic weights tend to generate higher correlations between error signals associated with the neurons of a given layer and the error signals propagated back to the previous layer whereas non-relevant (i.e. prunable) weights tend to generate smaller correlations. This observation has been termed as Principle of Maximum Correlation of Errors (MAXCORE) principle by the authors and has been introduced as the guiding principle behind the CAPE methodology. CAPE has emerged as one of the most significant weight pruning strategy reported in the recent past. In terms of computational complexity, CAPE has outperformed OBS and its variants. However, this work contests the claim of generality of the MAXCORE principle. Let us analyze a hypothetical ANN trained with BP algorithm whose backward pass is depicted in figure 3.1. Let the errors generated by M output neurons of the network presented with N training examples be represented in the form of an $N \times M$ matrix

$$E_o = \begin{bmatrix} e_1^o(1) & e_2^o(1) & e_3^o(1) & \cdots & e_M^o(1) \\ e_1^o(2) & e_2^o(2) & e_3^o(2) & \cdots & e_M^o(2) \\ e_1^o(3) & e_2^o(3) & e_3^o(3) & \cdots & e_M^o(3) \\ . & . & . & \cdots & . \\ . & . & . & \cdots & . \\ . & . & . & \cdots & . \\ e_1^o(N) & e_2^o(N) & e_3^o(N) & \cdots & e_M^o(N) \end{bmatrix}_{N \times M} \tag{3.1}$$

where, each row of E_o corresponds to the error generated by the output neurons for a given training pattern.

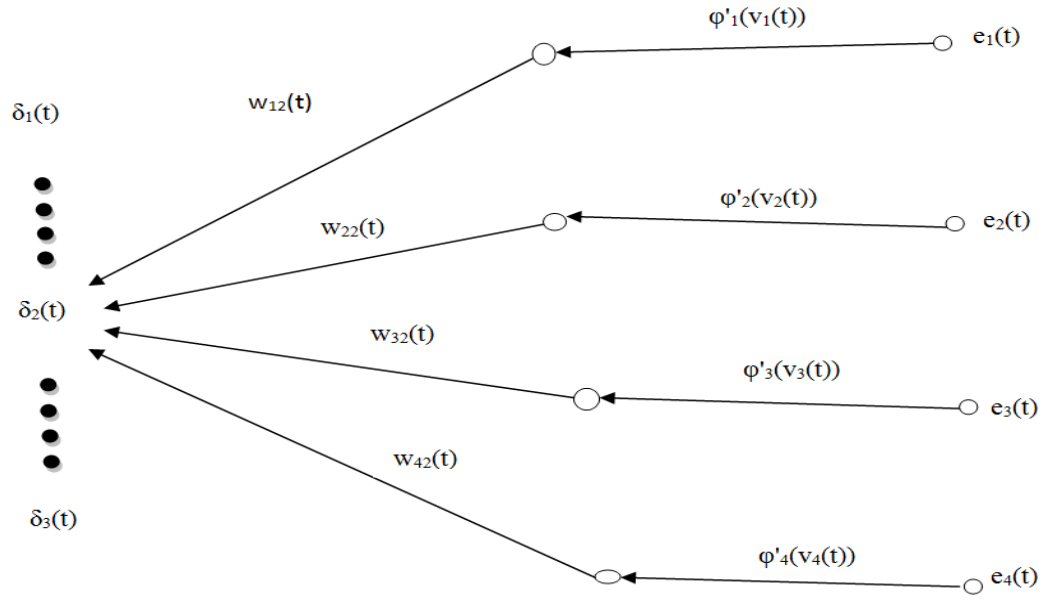


Figure 3.1: Backward pass of BP algorithm

Similarly, backpropagated errors associated with $(Q+1)$ hidden neurons of the same network can be represented in the form of an $N \times (Q+1)$ matrix

$$E_h = \begin{bmatrix} e_0^h(1) & e_1^h(1) & e_2^h(1) & \dots & e_M^h(1) \\ e_0^h(2) & e_1^h(2) & e_2^h(2) & \cdots & e_M^h(2) \\ e_0^h(3) & e_1^h(3) & e_2^h(3) & \cdots & e_M^h(3) \\ . & . & . & \cdots & . \\ . & . & . & \cdots & . \\ . & . & . & \cdots & . \\ e_0^h(N) & e_1^h(N) & e_2^h(N) & \cdots & e_M^h(N) \end{bmatrix} \tag{3.2}$$

The correlation matrix can be evaluated as

$$C_{oh}[k,i] = \sum_{t=1}^{N} e_k^o(t) e_i^h(t) \tag{3.3}$$

where, $C_{oh}[k,i]$is the entry in the correlation matrix corresponding to the scalar product of the k[th] column of E_o with the i[th] column of E_h

Let us examine $C_{oh}[3,2]$ and $C_{oh}[2,1]$ for the network shown in Figure 3.1

Let us assume $C_{oh}[3,2] > C_{oh}[2,1]$. According to the MAXCORE principle, w_{32} is more relevant than w_{21} if

$$\sum_{t=1}^{N} e_3^o(t) e_2^h(t) > \sum_{t=1}^{N} e_2^o(t) e_1^h(t) \tag{3.4}$$

where, $e_2^h(t) = \sum_{k=1}^{4} \delta_k(t) w_{k2}(t)$ (3.5)

and, $e_1^h(t) = \sum_{k=1}^{4} \delta_k(t) w_{k1}(t)$ (3.6)

As evident from above equations, higher or lower values of $e_i^h(t)$ depends not only upon w_{k1} but also upon local gradients δ_k of the output neurons connected to the hidden layer neuron *i*. Since, local gradient of the output neuron *k* is given by

$$\delta_k(t) = e_k^o(t) \varphi'(v_k(t)) \tag{3.7}$$

where, $v_k(t)$ is the induced local field and φ' is the differential of the activation function. Most of the time, a sigmoidal activation function is chosen which maps the synaptic weights $w_{ki}(t)$ nonlinearly in the form of induced local field passed through $\varphi'(.)$. Since, the contribution of $e_i^h(t)$ in the magnitude of correlation coefficient $C_{oh}[k,i]$ depends upon $\delta_k(t)$ which in turn are non linear functions of synaptic weights, $w_{ki}(t)$, a high value of $C_{oh}[k,i]$ neither guarantees a high magnitude nor a high degree of relevance for $w_{ki}(t)$. Furthermore, it is always better to define "relevance" of a synaptic weight either in a statistical sense or with respect to a performance metric. The standard practice of complexity regularization defines a risk function as

$$R(w) = \xi_s(w) + \lambda \xi_c(w) \tag{3.8}$$

where, $\xi_s(w)$ is the standard performance measure (e.g. MSE) and $\xi_c(w)$ is the complexity penalty term which depends upon the network model alone. As regularization parameter, λ represents the relative importance of the complexity penalty term with respect to the MSE. Assuming λ to have a moderate value somewhere between zero and infinity, the relative importance of complexity penalty term with respect to MSE is also moderate. Therefore, we can define "relevance" of individual synaptic weights both in terms of MSE and in terms of complexity penalty. However, to compute the relevance of weights in terms of $\xi_s(w)$, we need to run the BP algorithm first (at least the forward pass). For the networks which have not yet undergone training, it would be better to first compute $\xi_c(w)$ in terms of complexity penalty.

In the light of the above discussion, novel contributions of this work can be enlisted as follows:

- This work seeks to define relevance of the weights in a statistical sense.
- A coefficient of dominance has been proposed to identify nodes carrying higher information content and to determine complexity penalty for each weight.
- This work defines risk function both for the network and for the individual weights.
- The user can start with any architecture and prune the non-relevant weights just after initialization.
- Significantly lower computational cost as compared to some recent pruning schemes.
- Suitability for future implementation on DNNs.

3.3 Problem formulation

3.3.1 The proposed Methodology

The pruning scheme has been depicted in the form of figure 3.2. The proposed methodology still relies on starting with a random architecture. However, it seeks to subsequently optimize the network by pruning individual weights according to their “relevance”. This “relevance” has been defined in terms of both MSE and complexity penalty. After calculating the relevance of an individual weight in terms of risk function, weights are pruned according to a thresholding scheme. Each individual stage has been described in detail in subsequent sections.

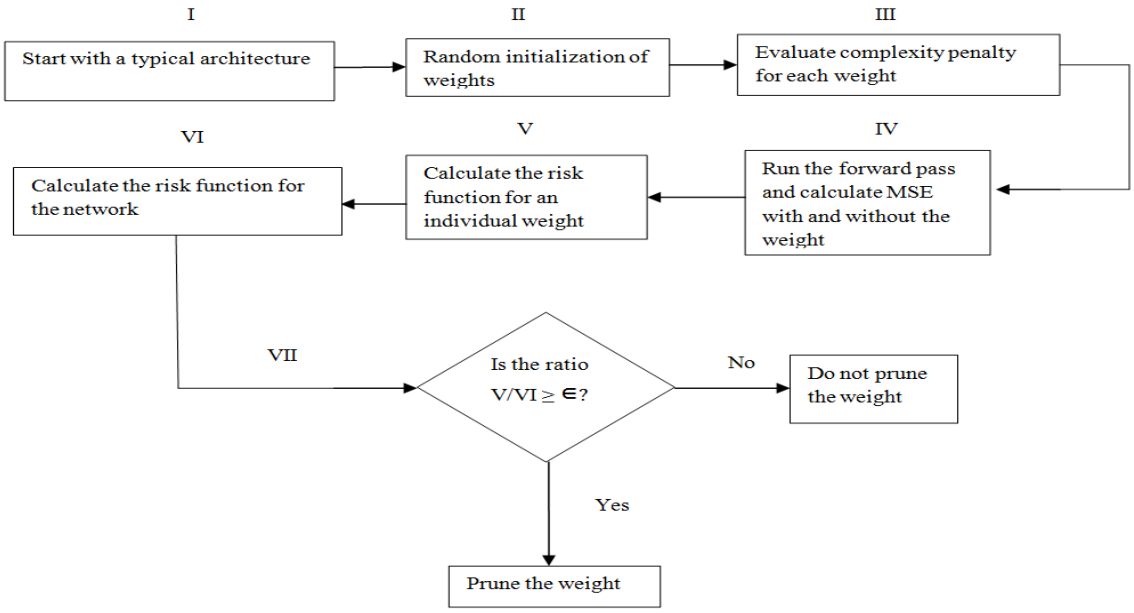


Figure 3.2: The proposed methodology

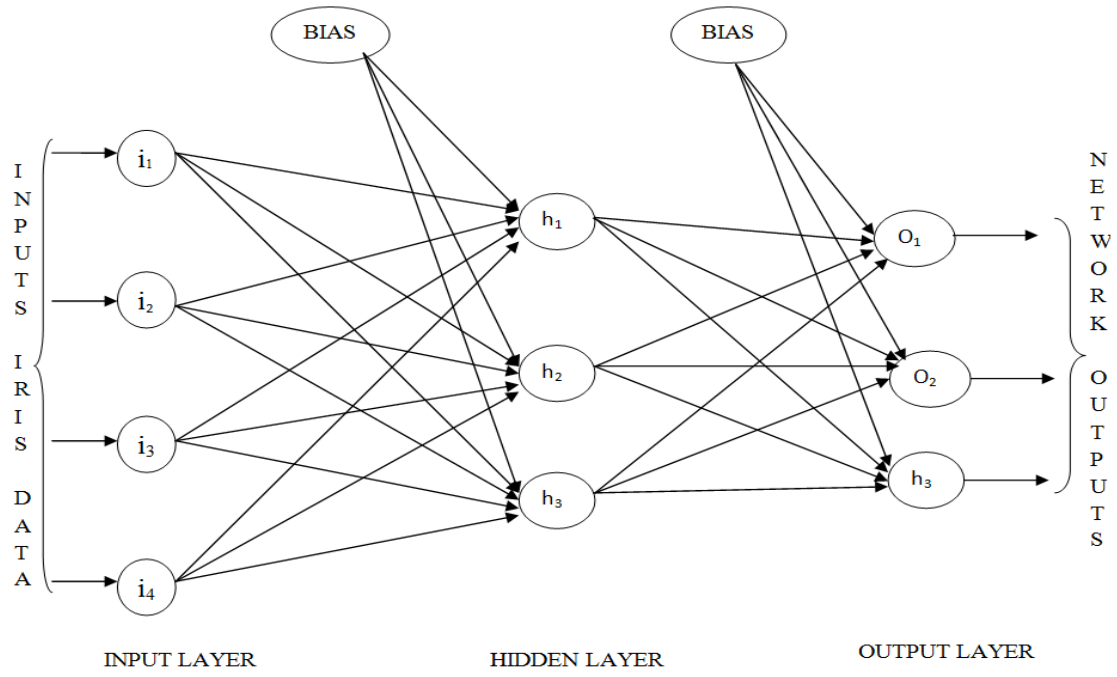


Figure 3.3: ANN architecture for IRIS data

As an illustration of the first stage of the methodology let us consider an ANN with single hidden layer and three target classes subjected to IRIS data as input. The number of hidden layer neurons

has been chosen to be three. As shown in figure 3.3, there are a total of 27 connections including 20 weights and 7 biases. All the connections have been initialized randomly. The connections would be evaluated as a part of weight pruning scheme which is described in detail with analytical justifications in the next section. The details of all individual stages of the methodology are described in the implementation section.

3.3.2 Description of dataset used

The simulations have been carried out on several benchmark datasets obtained from UCI machine learning repository. All the datasets chosen for this study represent pattern classification problems with varying number of samples, input dimensions and target classes details of which is shown in table 3.1.

Table 3.1: Data description

Name of the DataSet	No. of Input dimensions	No. of Training Samples	No. of Test Samples	No. of Classes
IRIS	04	75	75	03
Breast Cancer	09	350	349	02
Solar Flare	09	500	500	03
Letter Rec.1	16	10000	10000	20
Letter Rec.2	16	14000	6000	20

During the course of simulation, number of training and test samples was kept equal in most of the cases except in the case of *letter recognition* where the results varied significantly when the number of test samples was changed.

3.4 Implementation

3.4.1 The complexity penalty

The objective of pruning is to downsize the model to the level of least complexity that still provides the best generalization. In general, the capability of feed forward NNs for data fitting increases as the number of connection weights increase, because an increase in the number of free parameters enables the network to finely capture the distinctive features. However, a model with too many parameters can pick up nonessential information in the sample data and cause the problem of over

fitting. This leads to poor generalization. Therefore, application of appropriate pruning techniques can contribute to improvement in the generalization ability. Furthermore, a reduction in the network components is advantageous for hardware implementation of large-sized ANNs. The weights which are insensitive to the error changes can be discarded after each step of training. The pruned network is of smaller size and is likely to give higher accuracy than before its trimming. ANN use network pruning strategies which can be classified into two categories viz. weight decay and weight elimination. In both the strategies, a complexity penalty term is defined which appropriately quantifies the degree of redundancy in a weight. In weight decay procedure, if the initial weight vector is *w*, the complexity penalty term is defined as the squared norm of the weight vector *w* (i.e. all the free parameters) in the network and is expressed as

$$\xi_c(w) = \|w\|^2 = \sum_{i \in \zeta_{total}} {w_i}^2 \tag{3.9}$$

where the set ζ_{total} refer to all the synaptic weights in the network. However, in this scheme, some weights in the network have relatively larger values than others. This results in making smaller weights redundant since they have little influence on the training outcome. To overcome this limitation, the complexity penalty term is now redefined as follows

$$\xi_c(w) = \sum_{i \in \zeta_{total}} \left(\frac{\left(\frac{w_i}{w_o}\right)^2}{1+\left(\frac{w_i}{w_o}\right)^2}\right) \tag{3.10}$$

where w_o is the pre-assigned weight change parameter defined according to the problem and w_i refers to the i^{th} weight in the network. It is proposed here that the optimality of a weight vector would be defined in the statistical sense. Higher the complexity penalty for a weight, more important is the weight for the learning process. If the weights converging on a hidden neuron have variance more than the variance of total weights in the network, then that hidden node will contribute more to the network. The real challenge in this case lies in selection of an appropriate w_o. As can be inferred from figure 3.4, the choice of parameter w_o is crucial aspect of network pruning.

In this work, a new pruning method has been developed based on the weight variation information during the BP training. Every node in the hidden layer is evaluated to determine the effective variance of weights. In order to compute the complexity penalty term as defined in equation (3.10)

for each weight, a coefficient of dominance called D_{si} is calculated for each neuron and is expressed as

$$D_{si} = \frac{\overline{V}_{total}}{V_{si}} \tag{3.11}$$

where D_{si} corresponds to w_o of equation (3.10),

$\overline{V}_{total}$ is the effective variance of all the weights in the hidden layer and can be calculated as:

$$\overline{V}_{total} = \frac{\sum_{i,j \in h_{total}} (w_{ji} - \overline{w})^2}{h_{total}} \tag{3.11a}$$

where h_{total} represents the total number of weights in the hidden layer and $\overline{w}$ represents mean of weights of the hidden layer

and V_{S_i} is the variance of the weights of i^{th} hidden node and can be calculated as:

$$V_{si} = \frac{\sum_{i \in n} (w_{ji} - \overline{w_i})^2}{n} \tag{3.11b}$$

where n is the number of weights of the i^{th} neuron of the hidden layer and $\overline{w_i}$ is the mean of the weights of i^{th} hidden layer neuron.

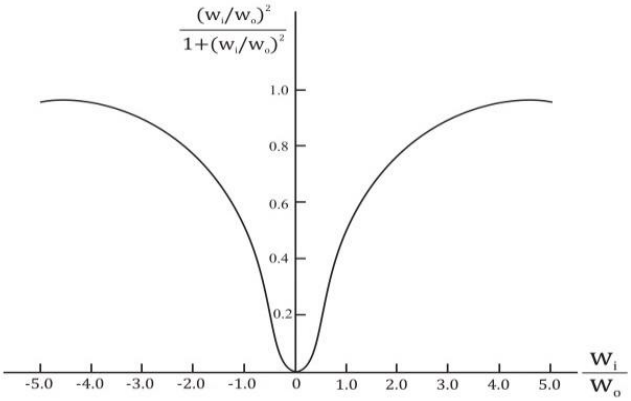


Figure 3.4: Variation of complexity penalty term with respect to w_i/w_o [246]

In the similar way, all the remaining hidden layer neurons are evaluated to find the coefficient of dominance and a set D is defined as

$$D = \{D_{S_1}, D_{S_2}, D_{S_3} \dots \dots \dots D_{S_i}\} \quad (3.12)$$

The complexity penalty for each weight in the network is defined as :

$$\xi_{wi} = \frac{\left(w_i/D_{S_i}\right)^2}{1+\left(w_i/D_{S_i}\right)^2} \quad (3.13)$$

The variance dependent weight pruning technique described above helps to increase the information content of the weight set convergent on a less dominant neuron. The introduction of coefficient of dominance into the complexity penalty term as described in Eq. (3.13) holds the essence of the pruning criteria. We basically want to prune a weight which contributes least to the network in terms of information content. The complexity penalty ξ_{wi} can be justified on the basis of a metric called Fisher information [266] which is defined as a way of measuring the amount of information that an observable random variable X carries about an unknown parameter θ of a distribution that models X and can be stated as

$$\Gamma(\theta) = E[(\frac{\partial \ln(f(X;\theta))}{\partial \theta})^2|\theta] \quad (3.14)$$

where ln is the natural logarithm of the probability function.

3.4.2 Fisher Information

In order to analyze the information content about the weight set w_{ji} (analogous to θ in equation (3.14)) in the input to hidden layer neuron (X in equation (3.14)) , it is essential to take into account the analysis in Chapter 2 from equation (2.20) to (2.31). It is a fact that the variance of total convergent input to a particular hidden layer neuron given in equation (2.31) is also dependent upon the underlying distribution of the input feature. Therefore, the following subsections will analyze the fisher information considering two possible distributions of the input samples. The first case pertains to a pre-processed set of input samples which are normalized to lie uniformly in the interval [0:1]. Such distributions are also encountered when the inputs are generated using a pseudo-random generator seed. The second case is more general which assumes the inputs to be normally distributed with an arbitrary mean and variance. According to central limit theorem we

can safely assume this distribution of inputs when the number of sample points are sufficiently large.

3.4.2.1 Case1: Input samples uniformly distributed in the interval [0:1]

For a uniform random variable lying within [0:1],

$$E\{(x_i)\} = \frac{1}{2} \; and \; \sigma_x^2(x_i) = \frac{1}{12} \tag{3.15}$$

Therefore, using equation (3.15) in (2.27) and (2.22), we get

$$E\{(x_i)^2\} = \frac{1}{4} + \frac{1}{12} = \frac{1}{3} \tag{3.16}$$

$$E\{y_j\} = \sum_{i=0}^{n} \frac{m}{2} + C = (n+1)(\frac{m}{2} + C) \tag{3.17}$$

The probability distribution function of the convergent inputs to hidden layer node is given as

$$f(\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-(y-\mu)^2}{2\sigma^2}} \tag{3.18}$$

where μ corresponds to the mean of convergent inputs to a hidden node (and is represented by equation (3.17))

$$\mu = E\{y_j\} = (n+1)(\frac{m}{2} + C) \tag{3.19}$$

And

$$y_j = \sum_{i=0}^{n} w_{ji} x_i \tag{3.20}$$

where x_i is the data input and w_{ji} is the weight connecting from i^{th} input node to j^{th} hidden node. According to equation (3.14), the information content about the weight set in the hidden layer neuron (which is the above defined random variable in equation (3.20)) can be calculated as follows:

$$lnf = \ln(\frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-(y_j)^2}{2\sigma^2}}) = ln\frac{1}{\sqrt{2\pi\sigma}} - \frac{(y_j-\mu)^2}{2\sigma^2} \tag{3.21}$$

$$\frac{\partial \, lnf}{\partial w} = -\frac{(y_j-\mu)}{\sigma^2}(\frac{\partial \, y_j}{\partial w}) \quad \text{where } \frac{\partial \, y_j}{\partial w} = \sum_{i=0}^{n} x_i = (n+1)E(x_i) \tag{3.22}$$

Using equation (3.15) and (3.19) in (3.22),

$$\frac{\partial\, lnf}{\partial w} = \frac{\left((n+1)(\frac{m}{2}+C)-y_j\right)}{\sigma^2}.\frac{(n+1)}{2} \tag{3.23}$$

Solving further for fisher information,

$$(\frac{\partial\, lnf}{\partial w})^2 = \frac{(n+1)^2}{4\sigma^4}((\frac{m}{2}+C)^2(n+1)^2 + y_j^2 - 2(\frac{m}{2}+C)(n+1)y_j) \tag{3.24}$$

$$\mathrm{E}(\frac{\partial\, lnf}{\partial w}^2) = \frac{(\frac{m}{2}+C)^2(n+1)^4}{4\sigma^4} - \frac{(\frac{m}{2}+C)(n+1)^3E(y_j)}{2\sigma^4} + \frac{(n+1)^2E({y_j}^2)}{4\sigma^4} \tag{3.25}$$

Also, from equation (2.26)

$$E\left({y_j}^2\right) = E\left\{\left(\sum_{i=0}^{n} w_{ji}x_i\right)^2\right\} = \sum_{i=0}^{n} E\left\{\left(w_{ji}\right)^2\right\}.E\{(x_i)^2\} + C_1 \tag{3.26}$$

Using equation (3.19) and (3.26) in (3.25), we get

$$\mathrm{E}(\frac{\partial\, lnf}{\partial w}^2) = \frac{(\frac{m}{2}+C)^2(n+1)^4}{4\sigma^4} - \frac{(\frac{m}{2}+C)^2(n+1)^4}{2\sigma^4} + \frac{(n+1)^2}{4\sigma^4}(\sum_{i=0}^{n} E\left\{\left(w_{ji}\right)^2\right\}.E\{(x_i)^2\} + C_1) \tag{3.27}$$

Also using equation (2.30) and (3.16), we get

$$\mathrm{E}(\frac{\partial\, lnf}{\partial w}^2) = \frac{(n+1)^2}{4\sigma^4}\sum_{i=0}^{n}(\frac{m^2+v}{3} + C_1) - \frac{\left(\frac{m}{2}+C\right)^2(n+1)^4}{4\sigma^4} \tag{3.28}$$

Final fisher information expression becomes:

$$\mathrm{E}(\frac{\partial\, lnf}{\partial w}^2) = \frac{(n+1)^3}{4\sigma^4}\left\{\left(\frac{m^2+v}{3} + C_1\right) - (n+1)(\frac{m}{2}+C)^2\right\} \tag{3.29}$$

Replacing the other terms by a constant K and variance of the input to hidden layer neuron (which is represented by σ in the equation (3.29)) by the coefficient of dominance given in equation (3.11) and considering two different hidden nodes with V_{s1} and V_{s2} as respective variances of weights convergent on them, the fisher information for the two nodes is given by

$$\Gamma_1(w_{ji}) = f(K\left(\frac{V_{s1}}{V_{total}}\right)^2) \tag{3.30}$$

and

$$\Gamma_2(w_{ji}) = f(K\left(\frac{V_{s2}}{V_{total}}\right)^2) \quad (3.31)$$

If $V_{s1} > V_{s2}$, fisher information of node 1 will be more compared to that of node 2 which implies that lower coefficient of dominance makes the complexity penalty higher (weight more relevant) and also enhances the information content of the convergent connections at a node.

3.4.2.2 Case 2: Input samples following normal distribution with mean m_1 and variance *v_1*

If the training samples are considered to follow a normal distribution with mean m_1 and variance v_1, then

$$E\{(x_i)\} = m_1 \; and \; \sigma_x^2(x_i) = \; v_1 \quad (3.32)$$

Using equation (3.32) in (2.27) and (2.22), we get

$$E\{(x_i)^2\} \; = (m_1^2 + v_1) \quad (3.33)$$

$$E\{y_j\} = \sum_{i=0}^{n} mm_1 + C = (n+1)(mm_1 + C) \quad (3.34)$$

The probability distribution function of the convergent inputs to hidden layer node is given as

$$f(\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-(y-\mu)^2}{2\sigma^2}} \quad (3.35)$$

where μ corresponds to the mean of convergent inputs to a hidden node (and is represented by equation (3.34))

$$\mu = E\{y_j\} = (n+1)(mm_1 + C) \quad (3.36)$$

And

$$y_j = \sum_{i=0}^{n} w_{ji} x_i \quad (3.37)$$

where x_i is the data input and w_{ji} is the weight connecting from i^{th} input node to j^{th} hidden node. According to equation (3.14), the information content about the weight set in the hidden layer neuron (which is the above defined random variable in equation (3.37)) can be calculated as follows:

$$lnf = \ln\left(\frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-(y_j)^2}{2\sigma^2}}\right) = ln\frac{1}{\sqrt{2\pi\sigma}} - \frac{(y_j-\mu)^2}{2\sigma^2} \quad (3.38)$$

$$\frac{\partial\, lnf}{\partial w} = -\frac{(y_j-\mu)}{\sigma^2}\left(\frac{\partial\, y_j}{\partial w}\right) \quad \text{where } \frac{\partial\, y_j}{\partial w} = \sum_{i=0}^{n} x_i = (n+1)E(x_i) \tag{3.39}$$

Using equation (3.32) and (3.36) in (3.39),

$$\frac{\partial\, lnf}{\partial w} = \frac{\left((n+1)(mm_1+C)-y_j\right)}{\sigma^2}.(n+1).m_1 \tag{3.40}$$

Solving further for fisher information,

$$\left(\frac{\partial\, lnf}{\partial w}\right)^2 = \frac{(n+1)^2 {m_1}^2}{\sigma^4}\left((mm_1+C)^2(n+1)^2 + y_j^2 - 2(mm_1+C)(n+1)y_j\right) \tag{3.41}$$

$$\mathrm{E}\left(\frac{\partial\, lnf}{\partial w}^2\right) = \frac{(mm_1+C)^2(n+1)^4 {m_1}^2}{\sigma^4} - \frac{2(mm_1+C)(n+1)^3 E(y_j){m_1}^2}{\sigma^4} + \frac{(n+1)^2 E({y_j}^2){m_1}^2}{\sigma^4} \tag{3.42}$$

Using equation (3.26) and (3.34), equation (3.42) becomes

$$\mathrm{E}\left(\frac{\partial\, lnf}{\partial w}^2\right) = \frac{(mm_1+C)^2(n+1)^4 {m_1}^2}{\sigma^4} - \frac{2(mm_1+C)^2(n+1)^4 {m_1}^2}{\sigma^4} + \frac{(n+1)^2 {m_1}^2}{\sigma^4}\left(\sum_{i=0}^{n} E\left\{\left(w_{ji}\right)^2\right\}.E\{(x_i)^2\} + C_1\right) \tag{3.43}$$

Also using equation (2.30) and (3.33), we get

$$\mathrm{E}\left(\frac{\partial\, lnf}{\partial w}^2\right) = \frac{{m_1}^2(n+1)^2}{\sigma^4}\sum_{i=0}^{n}\{(m_1^2+v_1).(m^2+v)+C_1\} - \frac{(mm_1+C)^2(n+1)^4 {m_1}^2}{\sigma^4} \tag{3.44}$$

Final fisher information expression becomes:

$$\mathrm{E}\left(\frac{\partial\, lnf}{\partial w}^2\right) = \frac{{m_1}^2(n+1)^3}{\sigma^4}\{(m_1^2+v_1).(m^2+v)+C_1-(n+1)(mm_1+C)^2\} \tag{3.45}$$

Replacing the other terms by a constant K and variance of the input to hidden layer neuron (which is represented by σ in the equation (3.45)) by the coefficient of dominance given in equation (3.11) and considering two different hidden nodes with V_{s1} and V_{s2} as respective variances of weights convergent on them, the fisher information for the two nodes is given by

$$\Gamma_1\left(w_{ji}\right) = f\left(K\left(\frac{V_{s1}}{V_{total}}\right)^2\right) \tag{3.46}$$

and

$$\Gamma_2(w_{ji}) = f(K\left(\frac{V_{s2}}{V_{total}}\right)^2) \quad (3.47)$$

If $V_{s1} > V_{s2}$, fisher information of node 1 will be more compared to that of node 2 which implies that lower coefficient of dominance makes the complexity penalty higher (weight more relevant) and also enhances the information content of the convergent connections at a node.

The above analysis of fisher information considering variations in the input sample distribution clearly justifies that variance of the incident weights is extremely important and a node with higher weight variance is relatively more informative and relevant than the node with lower weight variance. Hence, the inclusion of the variance in the complexity penalty term and the particular choice of coefficient of dominance is justified by the above analysis.

3.4.3 The pruning scheme

We illustrate the scheme restricting ourselves to single hidden layer architectures for all the data sets of table 3.1. Let us randomly initialize an architecture. For each initialization, run the forward pass once and calculate the MSE. Let us denote this MSE $\xi_s(w)$ which means MSE with all initialized weights intact. Now, make $w_i = 0$ and note down the MSE running the forward pass once. Let us denote this MSE $\xi_s(w_i)$ which means MSE with the i^{th} weight set to zero. We now modify the risk function defined in equation (3.8) as:

$$R(w_i) = \xi_s + \lambda\xi_{w_i} \quad (3.48)$$

where, ξ_{w_i} is defined in equation (3.13) and $\xi_s = \xi_s(w) - \xi_s(w_i)$. According to equation (3.47) the risk function of an individual weight takes into account the difference between the MSEs obtained with and without that particular weight. A higher value of ξ_s implies a higher contribution of that weight to MSE. On the other hand, the complexity penalty ξ_{w_i} defined above also contributes to the risk function. We can now calculate the risk function for the whole network as sum of risk functions for individual weights.

Thus, $R(w) = \sum_{i \in C} R(w_i)$ (3.49)

where, C is the total number of connections in the network.

We can now set the threshold for pruning. The weight w_i will be pruned if $\frac{R(w_i)}{R(w)} \geq \in$. A particular choice of $\in$=0.1 as the threshold has been experimentally obtained and has been found to work well for the data sets considered in the study. The merit of the technique lies in the fact that we can start with any architecture and obtain better results by pruning according to the ratio of the risk functions. This method gives due consideration both to MSE and network complexity.

3.4.4 Training the pruned ANN

A single hidden layer NN was trained using two different versions of BP algorithm for classifying five benchmark datasets. The network was first trained using ***traingdm*** function of Matlab. ***Traingdm*** was chosen as the training function because it gives the highest degree of freedom for the investigator to fine tune the training parameters since both learning rate and momentum constant can be varied experimentally using this function. The network was also trained with LM algorithm using ***trainlm*** function of Matlab. Since, LM algorithm is known to outperform simple gradient descent and other conjugate gradient methods in a plentitude of problems, it was necessary to examine whether the proposed technique can improve the performance of LM algorithm. A ten-fold cross validation scheme was used to prevent over fitting. The training phase performance of the network was evaluated with respect to four parameters viz. convergence success rate (% of time the simulation reaches error goal), mean epochs, execution time, and computational complexity. However, the generalization performance is the ultimate performance metric which gives the classification success rate. All the parameters have been evaluated both for unpruned and pruned networks.

3.5 Results and Discussion

3.5.1 Convergence and Generalization

The training phase performance of the ANN trained with ***traingdm*** are given in table 3.2. Results have been shown both for pruned and unpruned networks. It can be observed that the pruned architectures perform consistently better than their unpruned counterparts for all the three training parameters.

Table 3.2 Training phase performance **(traingdm)**

Data Set	Convergence Success (%)		Epoch Mean		Execution Time (Sec)	
	Unpruned Network	**Pruned Network**	**Unpruned Network**	**Pruned Network**	**Unpruned Network**	**Pruned Network**
IRIS	100	100	402.5	400	208	200
Breast Cancer	90.5	94.5	1180	1000	420	400
Solar Flare	88.5	92.5	1260	1130	542	500
Letter Rec.1	70.5	78.5	5805.5	4972.5	1200	960
Letter Rec.2	66	76	7280	6740	2300	1870

The convergence success rate has been defined as the percentage of time the training error goal is reached after repeatedly initializing and training the network. In the case of ***traingdm,*** an optimal combination of learning rate and momentum constant need to be specified. We have obtained this optimal combination from the error surfaces depicted in figure 3.5 and 3.6 for unpruned and pruned networks respectively. For the sake of convenience, we have plotted error surface only for the IRIS data. Similar surfaces can be plotted for other datasets as well. It can be seen from figure 3.6, that the lowest average error was obtained at a learning rate of 0.6 and momentum constant of 0.4.

Ten-fold cross validation was then performed, and the network was retrained to obtain final average values of convergence success, epoch mean and execution time. The networks were subsequently presented with the test samples. The effect of the proposed techniques on LM algorithm has also been investigated. The pruned networks have again performed consistently better with ***trainlm*** as shown in table 3.3. Although ***trainlm*** by default exhibits higher convergence success and a lower epoch mean, pruning of weights have improved these parameters as can be observed in table 3.3.

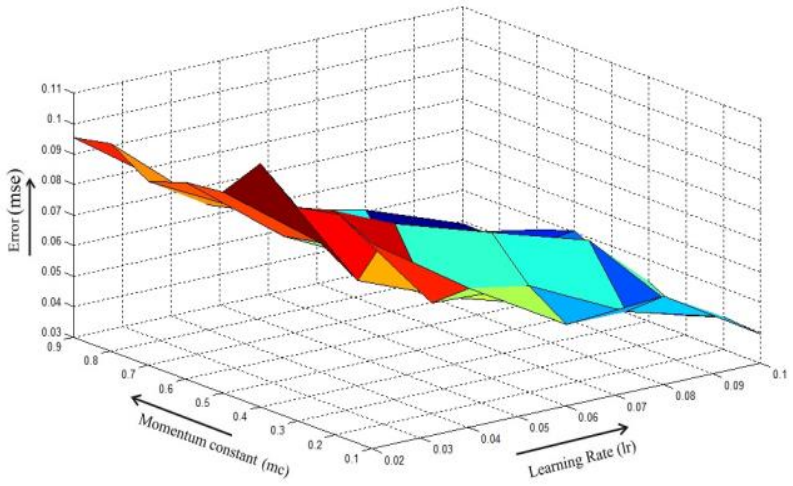


Figure 3.5: Surface plot for error obtained for the unpruned network trained using traingdm (IRIS data)

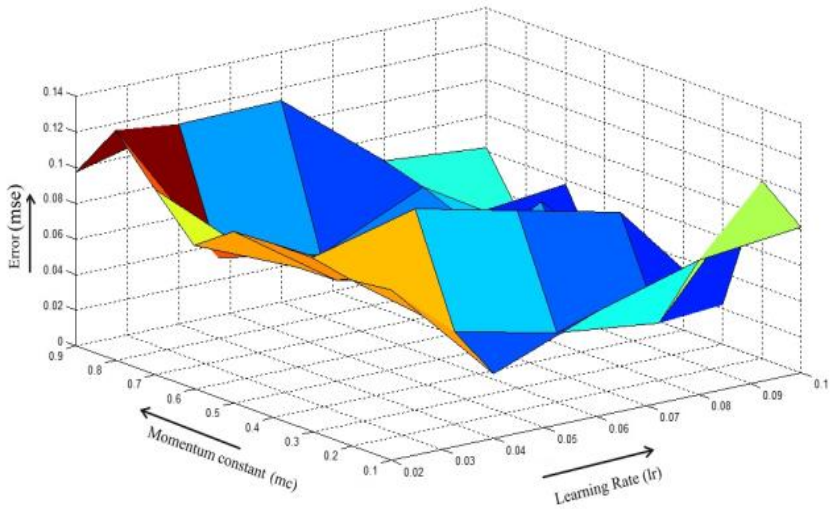


Figure 3.6: Surface plot for error obtained for the pruned network trained using traingdm (IRIS data)

Table 3.3 Training phase performance **(trainlm)**

Data Set	Convergence Success (%)		Epoch Mean		Execution Time (Sec)	
	Unpruned Network	Pruned Network	Unpruned Network	Pruned Network	Unpruned Network	Pruned Network
IRIS	100	100	56	50	14	13
Breast Cancer	100	100	255	200	22	20

Solar Flare	96	98	388	325.5	28	26
Letter Rec.1	80	88	1059.5	1002	66	64
Letter Rec.2	78	84.5	1987	1526	110	100

Table 3.4: Generalization performance **(traingdm)**

Data Set	**Total no. of test samples**	**No. of Correctly Classified Samples/Success Rate (%)**	
		Unpruned Network	**Pruned Network**
IRIS	75	66/88	66/88
Breast Cancer	349	158/45.27	188/53.87
Solar Flare	500	195/39	227/45.40
Letter Rec.1	10,000	4254/42.54	5502/55.02
Letter Rec.2	6000	2642/44.03	2972/49.53

Table 3.5: Generalization performance **(trainlm)**

Data Set	**Total no. of test samples**	**No. of Correctly Classified Samples/Success Rate (%)**	
		Unpruned Network	**Pruned Network**
IRIS	75	70/93.33	70/93.33
Breast Cancer	349	258/73.92	278/79.65
Solar Flare	500	295/59	320/64
Letter Rec.1	10,000	5270/52.70	6208/62.08
Letter Rec.2	6000	3642/60.70	4006/66.76

The reflection of better convergence in the training phase on the generalization performance was as expected and the results are shown in table 3.4 and 3.5 for ***traingdm*** and ***trainlm*** respectively. It is encouraging to note that magnitude of improvement in the performance metric is more significant in complicated architectures e.g. “letter recognition-1 and 2”. More complex architectures have been found to have larger number of prunable weights and less information content than simpler ones.

3.5.2 Comparison with other pruning techniques

In this section, the proposed pruning strategy is compared with two other well-known pruning methods viz. OBS [189] and CAPE [205]. The methods like OBS and its variants are often very

time consuming and need additional computational resources for computation of inverse Hessian matrix of the error function. However, there are no such complex computations required in the proposed method as it only involves examination of the relevance of the synaptic weights by evaluating MSE and complexity penalty for the network with or without the inclusion of weight. CAPE algorithm is based on the cross-correlation of back propagated errors which is again a computationally intensive task. The computational requirements for CAPE have also been compared with the proposed algorithm in the subsequent section.

Table 3.6 shows the comparative performance of the proposed algorithm with the other two pruning strategies on five different datasets with the total number of classes varying from 2 to 20. It is clear from the results presented in table 3.6 that the proposed algorithm achieves better generalization performance than OBS and CAPE in almost all the cases except for some datasets where the performance was comparable to OBS algorithm. The algorithm performed exceptionally well for the datasets where the number of classes and test samples were very large.

Table 3.6: Comparison of classification success rate of the proposed algorithm with other algorithms

Data Set	**Total no. of test samples**	**Classification Success Rate (%)**			
		Unpruned Network	**Proposed Pruning method**	**OBS method** [189]	**CAPE method** [205]
IRIS	75	93.33	93.33	93.33	93.33
Breast Cancer	349	73.92	79.65	79.65	76.50
Solar Flare	500	59	64	61	62
Letter Rec.1	10,000	52.70	62.08	50.27	51.86
Letter Rec.2	6000	60.70	66.76	55.45	62.65

The testing performance is further analyzed using paired sample t-test to assess the effectiveness of the proposed strategy with other two pruning strategies viz: OBS and CAPE and unpruned network and using different datasets. In the current study, the null hypothesis assumes that there is no significant improvement in the classification success rate of the proposed algorithm relative

to other pruning techniques and unpruned network. Conversely, the alternate hypothesis assumes a significant improvement in the classification success rate using the proposed technique.

Table 3.7: T-test results

Comparison	Dataset	P-value (α=0.05)	Null Hypothesis
Proposed & unpruned	IRIS	0.5	Rejected
	Breast Cancer	0.00244	Rejected
	Solar flare	0.00054	Rejected
	Letter Recog-1	.0000163	Rejected
	Letter Recog-2	0.0000412	Rejected
Proposed & CAPE	IRIS	0.5	Rejected
	Breast Cancer	0.0077	Rejected
	Solar flare	0.00074	Rejected
	Letter Recog-1	.0000553	Rejected
	Letter Recog-2	0.000191	Rejected
Proposed & OBS	IRIS	0.5	Rejected
	Breast Cancer	0.5	Rejected
	Solar flare	0.0069	Rejected
	Letter Recog-1	0.0000046	Rejected
	Letter Recog-2	0.00024	Rejected

In our case, 0.05 level of significance (α) was considered. If the P-value is less than (or equal to) α, then the null hypothesis is rejected in favor of the alternative hypothesis. And, if the P-value is greater than α, then the null hypothesis is not rejected. Data was taken as the classification success rate of all pruning strategies including the unpruned technique for different datasets for 20 set of experiments. The results of the t-test are summarized in Table 3.7 where null hypothesis is rejected for all the combinations which clearly justifies the efficacy of the proposed pruning algorithm in comparison to others.

3.5.3 Computational cost

The computational cost (complexity) is an important issue to be addressed during evaluation of algorithms. Sometimes, the algorithm's effectiveness in providing a solution to a given problem requires the execution of complex computations and the use of excessive memory resources, which can create severe difficulties in real-time or low-cost applications. Let us first analyze the computational complexity of the weight pruning stage described in section 3.4.1. Since we have developed the weight pruning scheme as an alternative to CAPE [205], we would now compare the computational cost for these two methodologies. Table 3.8(a) & 3.8(b) enlist the number of operations required by the proposed scheme and CAPE respectively as a function of number of input samples (N), input attributes (P), hidden neurons (Q) and output neurons (M). Number of operations required by the proposed methodology and CAPE was computed considering an MLP with N=140 input samples, P=2 input attributes, M=1 output neuron and the number of hidden neurons Q varying from 1 to 10. The same MLP was taken as reference by the inventors of CAPE. It can be easily observed from the table 3.8(a) & 3.8(b) that the proposed scheme requires lesser number of multiplication operations than CAPE. This is due to the fact that CAPE requires calculation of correlation coefficients to determine the relevance of weights whereas we define relevance in terms of complexity penalty which involves fewer operations. Furthermore, unlike the proposed technique CAPE involves running backward pass of BP which in turn involves computations through a non-linear activation function incurring additional cost as shown in the last row of the table 3.8(a) & 3.8(b). Figure 3.7 shows the number of operations required as a function of number of weights (connections). For the sake of simplicity, all the operations are considered to have the same computational cost. From the graph, it is apparent that the proposed scheme demands fewer resources than the CAPE algorithm. It is also clear from table 3.9 that using proposed weight pruning strategy, there is substantial reduction in the computational complexity and this reduction tends to become consistent with increasing number of connections which makes the proposed technique suitable for pruning massively connected networks (e.g. deep networks) in future.

Table 3.8: Computational operations required by (**a)** the proposed and **(b)** CAPE method

(a) **(b)**

Operation	**Number of operations as a function of *N, P, M, Q***

Addition	*N(3PQ+3Q+M)*
Subtraction	*N(3PQ+3Q+M)*
Multiplication	*N(2PQ+2Q+M)*
Division	*N(PQ+2Q)*
Tanh	-

Operation	**Number of operations as a function of *N, P, M, Q***
Addition	*N(3PQ+3QM-P+2Q+3M-2)-PQ-QM-M-1*
Subtraction	*N(Q+2M)*
Multiplication	*N(4PQ+4QM+5Q+5M)*
Division	-
Tanh	*N(Q+M)*

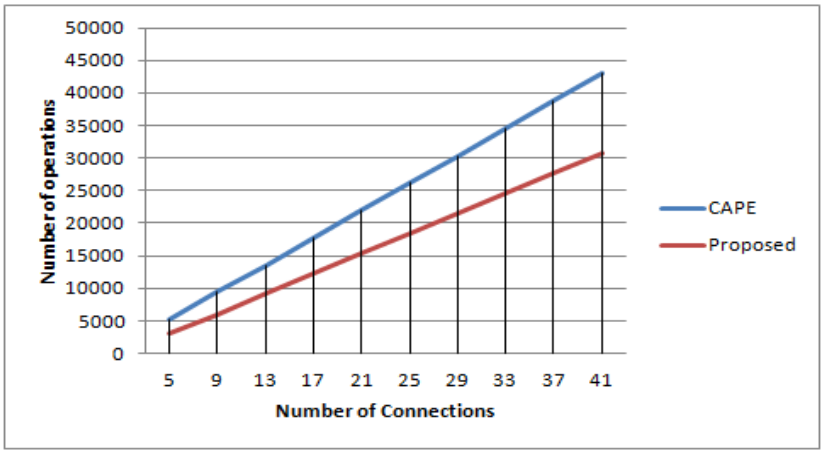


Figure 3.7: Estimated computational costs for CAPE and proposed scheme when pruning an MLP

Table 3.9: Percentage reduction in complexity by the proposed algorithm

No. of connections	**%age reduction in complexity (w.r.t. CAPE)**
5	42.41545894
9	37.25992318
13	31.9036038
17	30.65405831
21	29.8820744
25	29.35779817
29	28.97848931
33	28.69132373

37	28.46636216
41	28.28536835

The major contributions of this work as listed in section 3.2.1 have been successfully implemented and the performance results clearly reveal the efficacy of the proposed technique. Figure 3.2 describes how the first four points of our contribution has been incorporated into the implementation strategy starting with choosing a NN architecture, then evaluating the relevance of weights by finding complexity penalty corresponding to each weight. This penalty term is defined in a statistical sense. The network was subsequently evaluated for its performance in terms of a risk function which accounts for the change in MSE as well as the complexity penalty term. This risk function is an amalgamation of risk functions of individual node weights. The results shown in table 3.9 also reveal that the proposed algorithm outperforms some well-known pruning algorithms in terms of computational requirements. The reduction in the computationally complexity of the network makes it a suitable candidate for pruning the DNNs which are massively connected.

3.6 Summary

A novel technique based on the statistical information of the weight set for weight pruning has been designed and implemented. The results presented confirm the efficacy of the proposed technique in terms of convergence, execution time, and generalization performance. It is also evident that weight pruning contributes significantly to improvement in performance of the BP trained ANN. This improvement is reflected not only in ANNs trained with traditional gradient descent with momentum term (***traingdm***) but also with a more advanced algorithm like LM (***trainlm***). Reduction in the epochs required for convergence indicates that search about the weight space has yielded near optimal solutions. This work underpins the importance of defining both global and local risk functions since the decision to prune a particular weight should take into account its effective contribution to the total risk function. Evaluation of the risk function also points towards a correlation between statistical "relevance" of weights and their fitness in terms of lower MSE. The computational complexity of the proposed weight pruning stage has been proven to be better than the existing weight pruning strategy CAPE. Substantial reduction in

computational complexity has been observed which tends to become consistent with increasing number of connections making the proposed technique suitable for pruning massively connected networks (e.g. deep networks) in future.

The next chapter will focus on the implementation of a biologically plausible variant of LMS algorithm proposed by Widrow et al. [218] for supervised learning. The implementation issues related to this technique for multilayer NNs have also been discussed in this chapter.

Chapter 4
Implementation of Hebbian LMS algorithm for supervised learning

This chapter presents implementation of Hebbian-LMS proposed by Widrow et al. in 2015. The method is an amalgamation of LMS algorithm and Hebbian rule formally called the HLMS rule /which is an unsupervised learning approach. Depending upon the application in hand, HLMS can be used either as a stand-alone unsupervised learning strategy or as a complementary technique

in combination with the conventional LMS algorithm. In this work, HLMS is applied for unsupervised tuning of hidden layer parameters whereas conventional LMS has been used for the output layer. Use of HLMS has made feasible some kind of a control over the hidden layer dynamics which was not possible using conventional BP algorithm. Additionally, HLMS facilitates clustering of the input patterns in the hidden layer that helps faster learning of the output layer neurons using LMS algorithm. Simulation experiments performed with several benchmark datasets confirm that using this hybrid strategy eventually leads to faster convergence of the overall algorithm and better performance in comparison to conventional BP. The implementation issues related to this technique for multi layer NNs have also been discussed in this chapter.

4.1 Introduction

Hebbian learning is one of the oldest learning paradigms and is based in large parts on the dynamics of biological systems. "Fire together wire together" , the popular rule given by Hebb [226] simply means that if two neurons in the network are firing simultaneously, the synaptic weight connecting the two is strengthened. The implementation of this concept in artificial neural environment sometimes causes the weights to grow out of bound which are then controlled by constraints to achieve stability [267]. This very idea of "Fire together wire together" presented by Hebb for synaptic adaptation, learning and training is fundamental to the cognitive process. That is why, Hebbian learning is almost universally regarded as something biologically plausible. The evidence of its possible chemical and biological processes have already been studied [268]. Hebb's rule of strengthening the synaptic connection between two neurons is feasible if and only if pre-synaptic neuron is excitatory. It fails when the pre-synaptic neuron is excitatory. The rule was amended later to incorporate excitatory neurons into the learning process. "Fire together unwire together" is termed as anti-hebbian learning which means reducing the synaptic strength between the pre-synaptic and postsynaptic neurons if the pre-synaptic neuron is inhibitory [269]. P. Foldiack has designed a hebbian network with anti-hebbian feedback connections which can learn easily without using the statistical properties of the input data [270]. In the very same year, "anti-Hebbian" rule can be proposed for linear networks by Rubner et al. [271]. They have described that anti-hebbian rule was employed only locally in the network for adapting the output layer neurons. Later on, Carlson have described an anti-hebbian technique for learning in non-linear networks [272]. Sudjianto et al. [273] have provided a mathematical proof for hebbian learning in

non-linear networks for a fundamental understanding of this technique. Hebbian learning also called correlation-based learning has a major limitation of making the synaptic weights grow till they reach their maximum value or decay to zero which makes the network unstable. These kind of constraints have been discussed by Miller and MacKay in [267]. Kempter et al. [274] have proposed a new Hebbian rule on the basis of spiking neurons instead of correlation between neuron firing rate. McClelland raised two very important issues related to Hebbian learning that are (1) how much capable a network is to work as a computational model? and (2) What is the relation of Hebbian learning to human learning [275]. He also concluded that the learning introduced by Hebb is incapable of dealing with all the aspects of human learning. The effect of Hebbian learning have also been studied for RNNs in [276].

The goal of supervised learning is to train the network so as to produce outputs that match the desired ones. This also involves learning only the necessary things by avoiding or skipping the irrelevant ones. This makes training with a supervisor (teacher) all the more important. Although supervised learning is clearly psychologically relevant, and a majority of psychological models exist to represent this form of learning. However, evidences regarding its biological plausibility does not exist. The reason may be the need to back propagate the error signal which is highly inconsistent with known neurobiological properties [277]. Rosenblatt's Perceptron [278] and Adaptive Linear Neuron (ADALINE) are some of the earlier models developed for learning with a teacher. The training algorithm used for ADALINE was LMS invented by Widrow and Hoff in 1959 [95] which was based on the principle of steepest descent, a gradient method used to minimize the cost function i.e. mean square error. A year later, Widrow and his students devised Madaline Rule I, the earliest learning rule for feedforward networks with multiple adaptive elements. The major extension of the feedforward NN beyond Madaline I took place in 1971, when Paul Werbos developed a BP algorithm for training MLP NNs. BP was further refined by Rumelhart, Hinton, and Williams in 1985 [94] and till then it is the most popular and widely used learning algorithm in training supervised networks. The iterative procedure of calculating the local gradient used in BP algorithm is a cumbersome task and cannot be avoided. The beauty of BP algorithm lies in its simplicity. However, the limitations still exist which encouraged the researchers to carry on improving the existing algorithm. Being a gradient search, it is possible for weight vectors to converge to a local rather than global minima, resulting in sub-optimal performance. Learning rate parameter is also a major factor affecting the performance of the

algorithm. A small value of learning rate will smoother the trajectory of the weight space at the cost of slower learning whereas large value of learning rate will faster the training process but large changes in the synaptic weights sometimes make the system unstable. Secondly, there are chances of skipping the global minima. A lot of variants of BP algorithm considering the above stated factors have already been discussed in the previous chapters.

4.2 Hybrid algorithms

Researchers have tried several alternative approaches to speed the convergence and improve the classification performance of multi-layer NNs. In 1996, Jeong et al. [279] have devised a new learning mechanism combining conventional BP and Hebb's learning. The hidden layer neurons are deliberately saturated by adding an extra gradient term to the conventional error term. This additional gradient term happens to follow the hebbian learning. This new algorithm converges much faster than BP algorithm and generalization performance of the network has also improved significantly. Naseri et al. [212] have designed a hybrid model for daily load forecasting. They have used Self Organizing Map (SOP) NN which uses unsupervised learning for clustering of data. It maps high dimensional data into a lower dimension by keeping similar elements together to form clusters. Once the clusters are formed, supervised learning is applied for load forecasting. The effectiveness and superiority of the hybrid technique has been proved by comparing it with a network that uses unclustered data. It has also been demonstrated that the hybrid technique improves the forecasting results in comparison to the technique which uses unclustered data. A similar kind of mechanism have been developed by Sheela et al. [214] for predicting the wind speed. They have also shown that the hybrid technique has better convergence rate as compared to the single models. Widrow et al. [280] have proposed "No-Prop" algorithm in which the output error is not back propagated to alter the hidden layer weights so as to minimize the MSE. Instead, the hidden layer weights are chosen randomly and remains constant and only output layer neurons are adapted using standard LMS algorithm. The performance of the No-Prop algorithm is at par with the BP algorithm as long as the number of training patterns are less than the LMS capacity. LMS capacity is defined as the number of input patterns that can be trained using the NN with zero mean square error. If the patterns are more than the capacity, equivalent performance of the algorithm can be obtained by increasing the network capacity. This can be done by increasing the number of neurons of the hidden layer that drives the output layer. This algorithm is simpler, easier

to implement and have faster convergence than its conventional counterpart. Two years later, the same group of researchers have devised an unsupervised technique called HLMS [218] which is an amalgamation of two popular algorithms: LMS and Hebb's learning. It is a different form of LMS algorithm which is biologically plausible. This technique has tried to fill the gaps in Hebb's learning by incorporating inhibitory neurons into the analysis rather than concentrating only on the excitatory neurons. Motivated by the suggestion of Widrow et al. [218] to use HLMS (for training the hidden layers) and the supervised/conventional LMS (for training the output layer), this chapter summarizes training and generalization performance of the suggested architecture over several benchmark data sets. We believe that the results and implementation details will prove to be useful for future work with an aim for further improvement in HLMS performance.

4.3 HLMS algorithm

Widrow and his colleagues have devised an unsupervised learning algorithm called HLMS which seems to be biologically more plausible. It is an implementation of Hebb's teaching by means of LMS algorithm. It has many practical engineering applications with an assumption that this is the nature's algorithm performed by the biological neurons to interact with each other. Figure 4.1 represents a single neuron trained with HLMS algorithm.

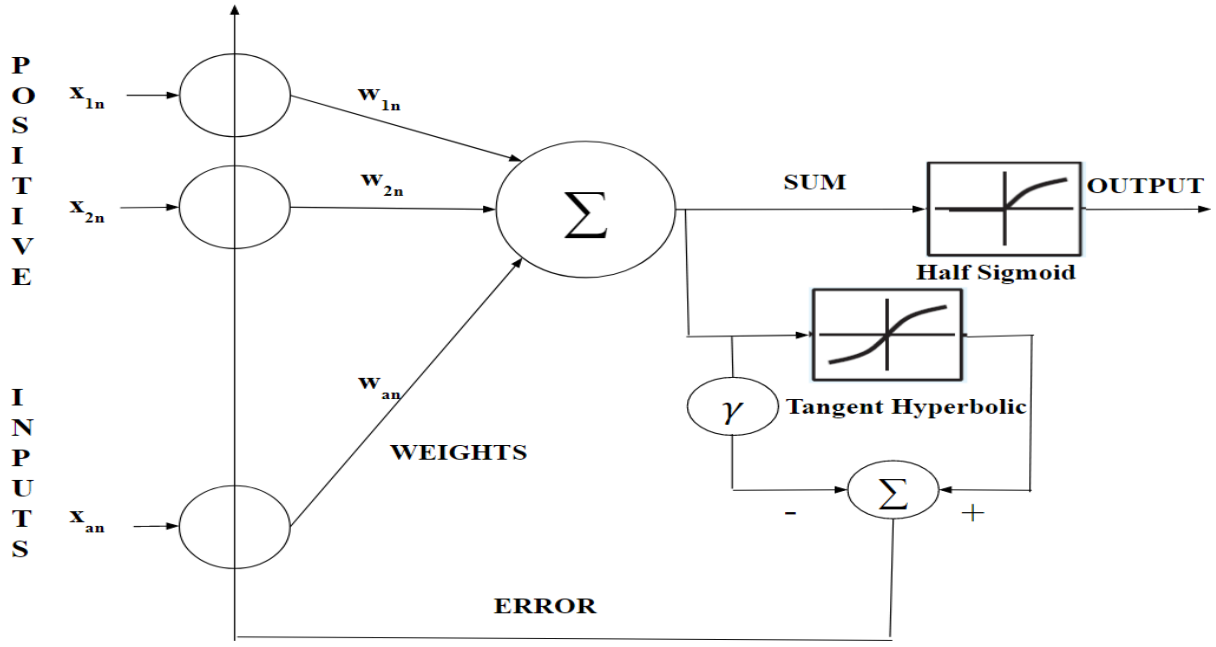


Figure 4.1: A single neuron trained with HLMS

Mathematically, the algorithm can be expressed as

$$w_{n+1} = w_n + 2\mu e_n x_n \tag{4.1}$$

$$e_n = sgm(y_n) - \gamma(y_n) = sgm(x_n^T w_n) - \gamma({x_n}^T w_n) \tag{4.2}$$

where *sgm()* is the standard sigmoid function and γ is a constant and can have any positive value less than the initial slope of the sigmoid function.

The output of the neuron can be given as:

$$(OUT)_n = \begin{cases} sgm(x_n^T w_n), & x_n^T w_n \geq 0 \\ 0, & x_n^T w_n < 0 \end{cases} \tag{4.3}$$

The HLMS algorithm assumes the following essential:

- The inputs from the presynaptic neurons can never be negative. It can either be zero or positive corresponding to neuron not firing and firing respectively.
- Synaptic weights should be positive.

- The half sigmoidal function used at the output neuron will keep the output to be positive or zero corresponding to postsynaptic neuron firing or not firing.

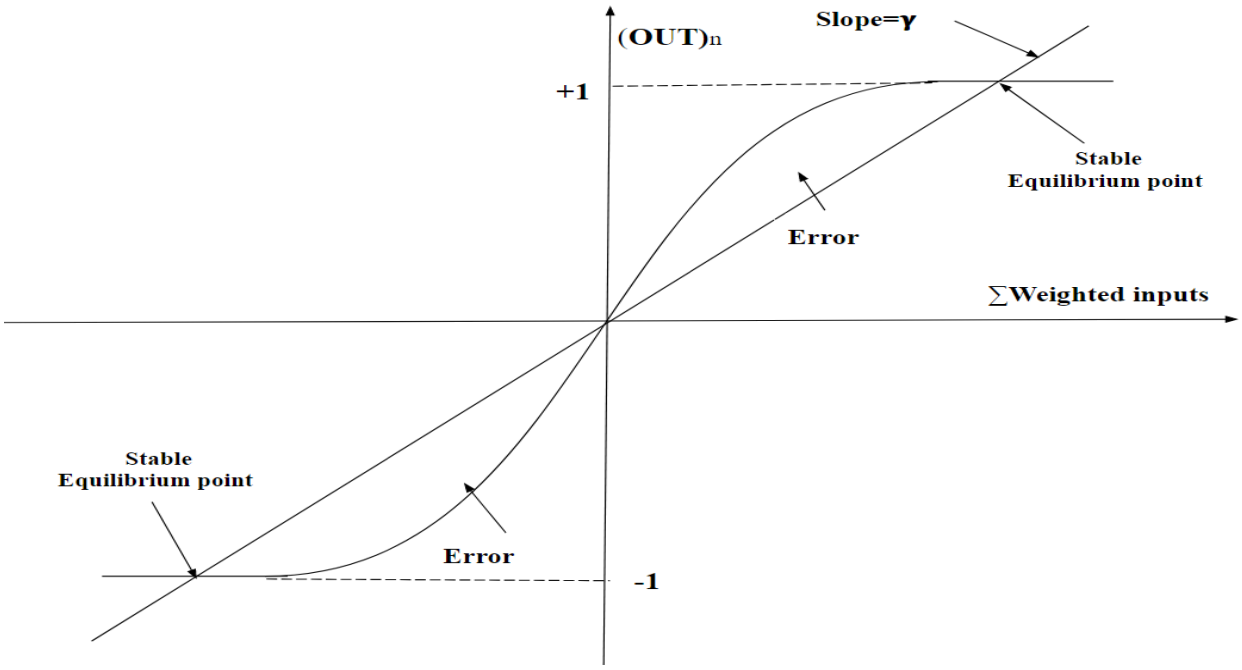


Figure 4.2: Output vs weighted sum of inputs

The LMS algorithm used will not allow the weights to grow out of bound as was with original Hebb's rule. If the weights get saturated during the adaptation process, it is not a permanent process. They will be brought back to their natural positive range by the LMS algorithm. The final output of the neuron versus the weighted sum of inputs is plotted in figure 4.2.

The process of maintaining the weights within their normal range is termed as "Homeostasis". It is a regularization phenomenon occurring in living systems for maintaining a stable state under conflicting situations. HLMS also follows this strategy by keeping the weights to be in the permissible range by reversing the error signal when it happens to cross the stable equilibrium point. This very process can be visualized in figure 4.2 when error crosses any of the stable equilibrium point, the LMS algorithm manages to bring it back to the stable region. Thus, the assumptions made above in fact establish the biological plausibility of HLMS algorithm.

Hebbian-LMS can work with all configurations of networks. The network can be layered structure, or it can be interconnected in random configurations. When the training patterns are given to the

network trained with HLMS algorithm, the patterns will try to cluster near the two stable equilibrium points. Error corresponding to every pattern will be small and the MSE for all the patterns will be minimized by the LMS algorithm. Hence, training patterns will be divided into two classes without supervision. This technique will not allow the output corresponding to weighted sum of inputs as shown in figure 4.2 to increase out of bound.

4.3.1 Need of HLMS

According to Hebb's teaching "Fire together, wire together ", weights will go on increasing until they reach a saturation value. This eventually leads to an impractical design. Therefore, Hebb's principle cannot alone be used to design an ANN. LMS algorithm in the HLMS maintains a stable control and avoids saturation of the weights by preventing the weights and weighted sum of inputs from increasing unrestrictedly. Secondly, BP algorithm is not biologically plausible. This new form of LMS provides insights into learning in the living NNs. HLMS is an unsupervised learning algorithm that can be used for clustering of input samples. The performance of the HLMS is at par with the well-known clustering algorithms like K-means, Expectation Minimization (EM). However, all the existing clustering techniques require to determine manually or heuristically some prior information about the model parameters such as number of clusters. On the contrary, HLMS require only a learning step value i.e. μ which is same as choosing step size for the supervised LMS. Homeostasis phenomenon have also been incorporated into the HLMS so that the weights and weighted sum of inputs should remain within two equilibrium points. Furthermore, HLMS not only specifies the direction of weight change but also the rate of change. The rate of weight change is given by

$$Rate \propto (input\ signal \times error\ signal) \tag{4.4}$$

The sign of the error represents the direction of change and the magnitude represents the rate of change. As the error decreases, rate of weight change also decreases which indirectly specifies that the algorithm is approaching one of the stable equilibrium points. In figure 4.2, when the error is negative, the synaptic weight strength is decreased so as to approach the positive stable equilibrium point or vice versa for the positive error. This is called anti-hebbian learning (fire together, unwire together), and is assimilated in the HLMS algorithm. When implemented for MLPs, the rate of convergence of the algorithm can be greatly improved. The HLMS algorithm will be applied

independently to all the neurons so that the learning becomes decentralized. All the neurons in the network will be trained and adapted simultaneously, so the speed of convergence of the network will not be more than the convergence rate of a single neuron. The number of clusters that the network can differentiate depends upon the network capacity which is defined as the number of synaptic weights of a neuron of the output layer. It can further be improved by increasing the network capacity by means of increasing the number of hidden layer neurons which drives the output layer.

The literature of hybrid algorithms and the benefits offered by the HLMS as mentioned above served as a motivation to further analyze the application of HLMS algorithm which is a supervised learning algorithm. This supervised approach uses HLMS for the hidden layers of the network and LMS for the output layer. It will offer reduced computational cost in terms of calculation of local gradient for each and every neuron in the network and faster rate of convergence in comparison to BP algorithm.

4.4 Modified HLMS

Widrow et al. in [218] have proposed that a multi-layer NN could be trained with both supervised and unsupervised methods. The unsupervised method is the one proposed by them in the same paper i.e. HLMS and supervised is the original LMS algorithm. According to the application, it was stated that a network with hidden layers (one or more) could be trained with his proposed HLMS method and the final layer which is the output layer can be trained with original LMS algorithm. The input pattern would produce an individual binary word at the final hidden layer. This binary word can be used to train the output layer neuron. The neuron with the largest output would represent the class of the pattern the input data belongs to. The advantage of using HLMS instead of original LMS for hidden layers is that the recursive computation of local gradient can be avoided which is mandatory for the adaptation of hidden layer neuron synapses. Secondly, all the synapses in the hidden layers will be adapted simultaneously, so the speed of convergence will improve. Clustering of input patterns happens at the hidden layer output using HLMS which is further classified using the LMS at the output layer. This not only provides faster convergence but also provides better classification than the conventional BP algorithm. Lastly, conventional BP algorithm used for training the MLP minimizes the output error only and has no direct control over

the hidden layer neurons adaptation. Training the hidden layer using HLMS not only provides a direct control over the hidden layer neurons adaptation but the simultaneous adaptation of all the hidden layer weights which also leads to faster convergence.

4.5 Description of dataset used

The simulations have been carried out on several benchmark datasets obtained from UCI machine learning repository. All the datasets chosen for this study represent pattern classification problems with varying number of samples, input dimensions and target classes details of which is shown in Table 4.1.

Table 4.1: Data description

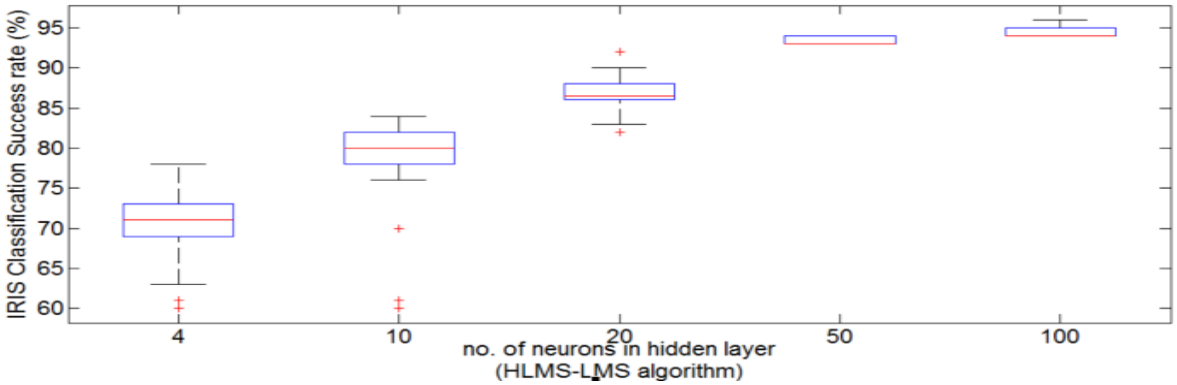
Data Set	No. of Input dimensions	No. of Training Samples	No. of Classes
Heart Disease	13	170	02
IRIS	04	100	03
Wine	13	178	03
Gesture Phase	18	1227	05

4.6 Implementation and Results

Simulations were performed to demonstrate the effectiveness of the modified HLMS using network with single and double hidden layer architecture. The number of hidden layer neurons was varied from number of attributes to the value of 100. A set of benchmark datasets as shown in table 4.1 were classified using modified HLMS and conventional BP to scrutinize the potency of the HLMS in comparison to LMS. The network was trained repetitively for 20 times for classifying the datasets. The variations with respect to number of training epochs was also included in the study. The ANN's learning performance using the conventional and modified HLMS for IRIS dataset with 1000 number of epochs is shown in the box plot in figure 4.3 (a) and (b). Each box plot shows the performance of the training phase with a variation in number of hidden layer neurons from 4-100. The plots clearly reveal that the optimum single and double hidden layer

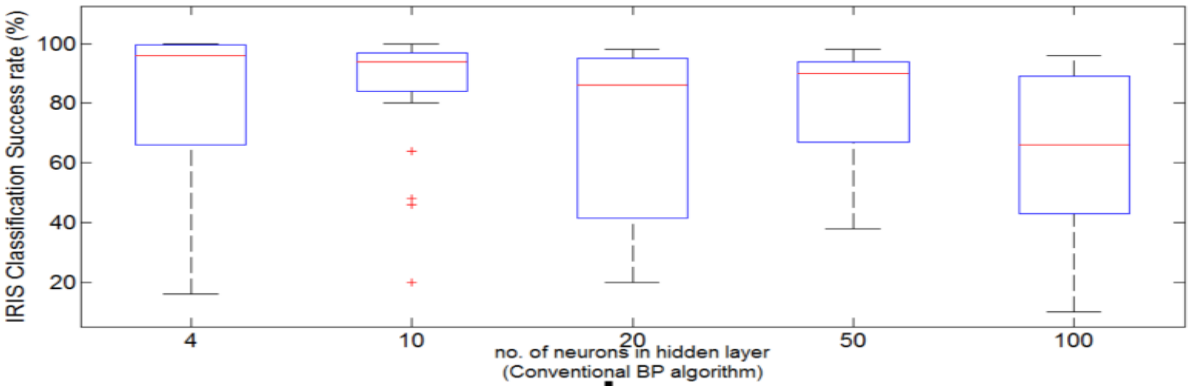
architecture for training using modified HLMS is the one with maximum number of hidden layer neurons. Similarly, the optimum single and double hidden layer architecture trained with conventional BP is found to be 4-100-3 for both.

The network capacity as stated earlier is the number of hidden layer neurons that drives the output layer. If the number of training patterns is less than the network capacity, MSE will be minimum



and best classification performance can be achieved [218]. Since in our analysis, the number of training patterns were always much more than the capacity, the performance of the network can be improved by increasing the number of hidden layer neurons only. The average classification success (%age) for BP and modified HLMS for all the datasets listed in table 4.1 are shown in tables 4.2.

4.3(a)



4.3 (b)

Figure 4.3: Box plot for %age classification success of single hidden layer architecture for IRIS data using (a) modified HLMS and (b) conventional BP algorithm

The results clearly indicate that the performance of the network improves with the increase in number of hidden layer neurons for modified HLMS. The network trained with conventional BP

has no order of performance improvement. However, the training time has been adversely affected as the number of hidden layer neurons approached 100 for both kinds of algorithms. The training process was extended to hours for maximum number of hidden neurons for all datasets for double hidden layer architecture trained with conventional BP. Therefore, those columns in Tables 4.2 are empty. The classification success improvement of both single- and double-layer architecture trained with modified HLMS and BP are also shown diagrammatically in figure 4.4 and 4.5.

Table 4.2: Classification success for all dataset using modified HLMS and conventional BP

4.2(a)

Architecture	**IRIS data Average Classification Success (%age)**				
	HN= Number of attributes	**HN=10**	**HN=20**	**HN=50**	**HN=100**
Single hidden layer (Modified HLMS)	70.55	80.7	89.15	94.85	96.3
Single hidden layer (Conventional BP)	76.6	92.6	85.6	51.4	75.8
Double hidden layer (Modified HLMS)	55.6	71.5	78.8	89.3	94
Double hidden layer (Conventional BP)	68	71	80.7	74.8	--

4.2(b)

Architecture	**Heart data Average Classification Success (%age)**			
	HN= Number of attributes	**HN=20**	**HN=50**	**HN=100**
Single hidden layer (Modified HLMS)	79.92	81.994	83.76	84.694
Single hidden layer (Conventional BP)	76.99	76.356	77.822	72.221
Double hidden layer (Modified HLMS)	73.291	75.054	77.994	78.406
Double hidden layer (Conventional BP)	86.95	86.878	87.466	--

4.2(c)

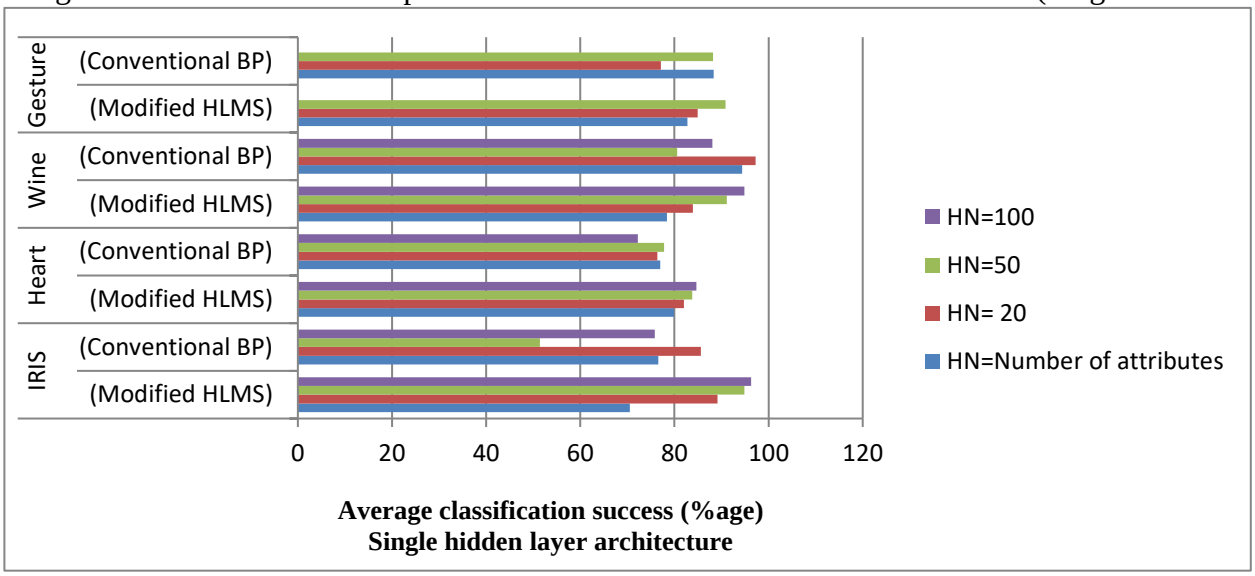
Architecture	**Wine data Average Classification Success (%age)**			
	HN= Number of attributes	**HN=20**	**HN=50**	**HN=100**

Single hidden layer (Modified HLMS)	78.408	83.89	91.15	94.87
Single hidden layer (Conventional BP)	94.39	97.22556	80.62	88.04111
Double hidden layer (Modified HLMS)	79.77125	84.19375	91.07625	94.94
Double hidden layer (Conventional BP)	93.153	77.368	90.78	--

4.2(d)

Architecture	**Gesture data Average Classification Success (%age)**		
	HN= Number of attributes	**HN=20**	**HN=50**
Single hidden layer (Modified HLMS)	82.759	84.971	90.884
Single hidden layer (Conventional BP)	88.313	77.094	88.235
Double hidden layer (Modified HLMS)	83.852	83.852	91.098
Double hidden layer (Conventional BP)	87.566	73.332	63.785

Figure 4.4: Performance comparison of modified HLMS and conventional BP (Single hidden layer architecture)



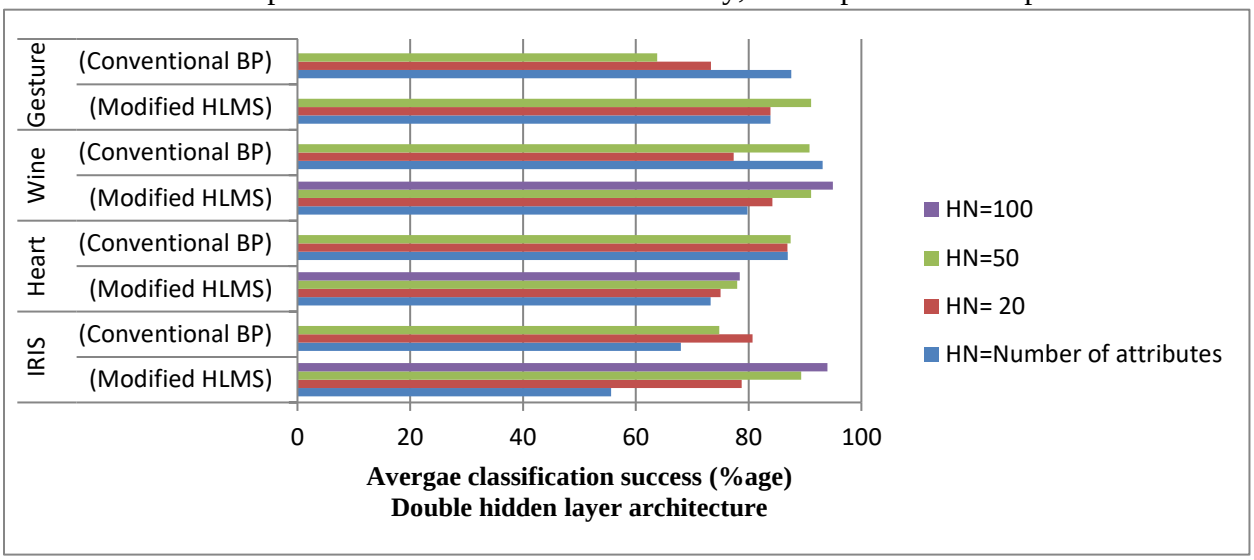
4.6.1 Data preparation

The results for the modified HLMS and conventional BP algorithm for IRIS dataset corresponding to an ordered and random representation are tabulated in Table 4.3. The results shown in table 4.3 clearly indicate that the network trained with BP algorithm behaves randomly and an optimum architecture for it was chosen by executing several experiments by varying the number of hidden layer neurons. On the contrary, the HLMS algorithm's performance increases as the number of hidden layer neurons start increasing. Secondly, the classification success rate obtained using HLMS is almost constant whereas in BP, it is random. By random, we mean that for some training samples, the performance is as good as 100% and for some it is even worse than 10%. So, the average value drops down to 64% in some of the cases. The average performance does not go beyond 84% even after applying the ten-fold cross validation to the BP trained network. Order of data representation for training the NN is a dominant factor from the performance point of view. The classification success in case of HLMS is better for the ordered representation of the data than random representation whereas the trend performance growth with increase in number of hidden neurons remains the same as shown in figure 4.3(a). Contrastingly, the performance for the conventional BP algorithm is arbitrary as the architecture which was chosen as the optimum

architecture for ordered representation of data becomes the most non-optimum architecture for random representation of dataset.

Figure 4.5: Performance comparison of modified HLMS and conventional BP (Double hidden layer architecture)

In order to further evaluate the efficacy of the technique, the network performance was evaluated with increasing number of epochs. All benchmark datasets have been trained using conventional BP and HLMS algorithm for single hidden layer architecture and analyzed with variation in the number of epochs from 100 to 5000. The results shown in figure 4.6 indicate that the improvement in performance of the network (trained with modified HLMS) with increase in epochs is marginal for all datasets except for IRIS dataset. On the contrary, the improvement in performance with



increase in the epochs for conventional BP is consistent as shown in figure 4.7. In datasets with maximum number of training patterns and attributes in comparison to other datasets, a hefty amount of time has been dissipated for training the network for maximum number of epochs (5000) as the number of hidden layer neurons approached its maximum value set for analysis

Table 4.3: Classification success for different representation of IRIS dataset

No. of Hidden layer Neurons (Single Hidden layer architecture)	**Average Classification Success (%age)**			
	Conventional BP algorithm		**Modified HLMS algorithm**	
	Ordered Representation	**Random Representation**	**Ordered Representation**	**Random Representation**

4	81.4	62.8	57.9	62.85
10	84.1	80.2	82.05	73.05
20	64.1	86	83.1	81
50	82.4	72.4	90.8	84.3
100	69.6	73.5	94	88.1

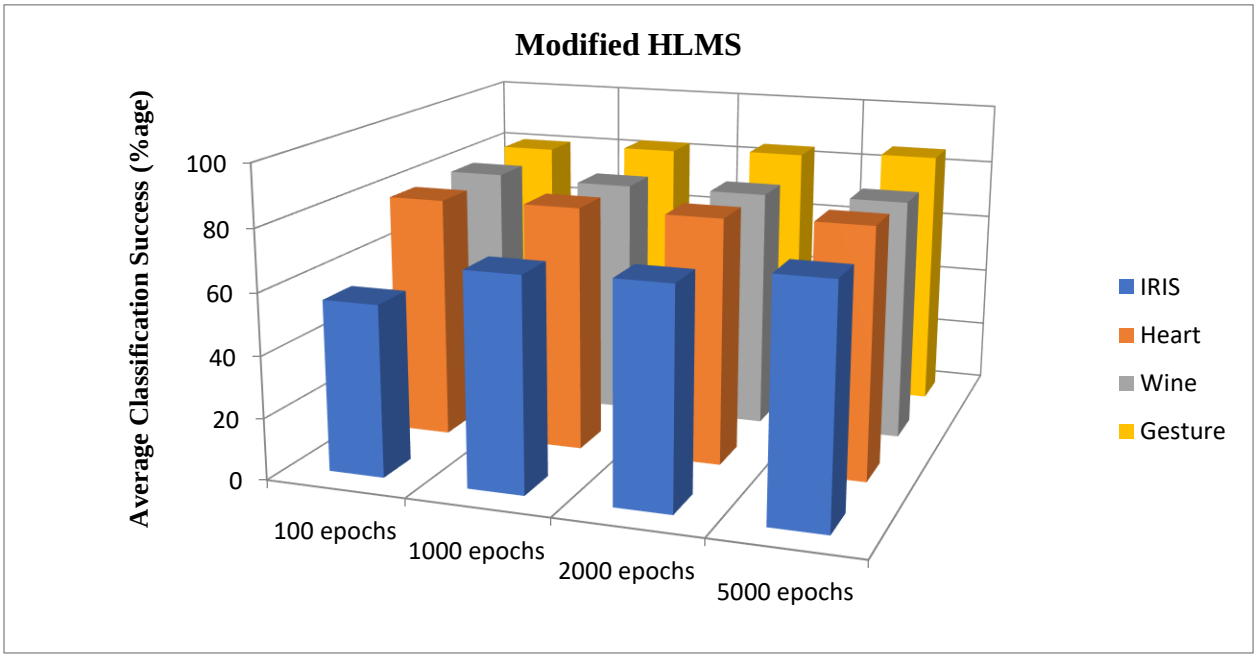


Figure 4.6: Performance of modified HLMS with variation in epochs

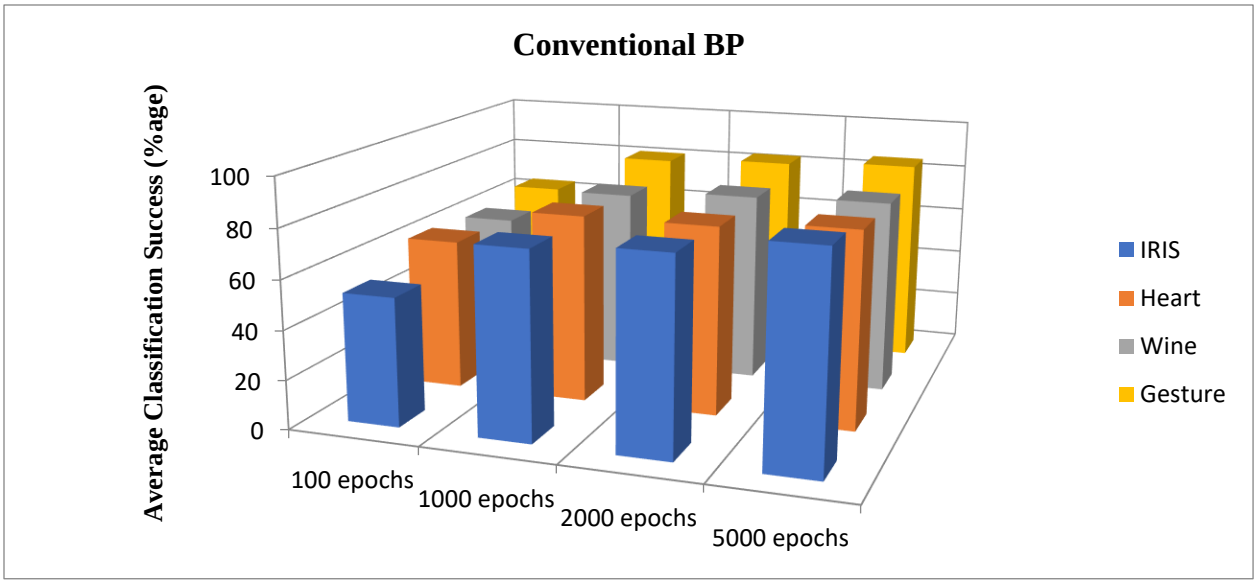


Figure 4.7: Performance of conventional BP with variation in epochs

4.6.2 Variation in slope of the activation function

The activation function of a neuron affects the output of a neuron over a wide range of input variations depending upon the nature of the function. It has been proved for the BP algorithm that changing the gain parameter of the activation function is equivalent to changing the learning rate parameter, weights and biases and is an effective means for performance improvement in general BP algorithm [33]. This has served as a motivation to evaluate the variation in network performance with a variation in slope of the activation function. A common choice for the activation function used in MLP NN is the sigmoid function represented as:

$$f(x) = \frac{\gamma}{1+e^{-\beta\gamma}} \tag{4.5}$$

The parameter β is called the gain and βγ is the slope of the activation function. If γ=1, the gain is same as the steepness of the activation function. Many other variations of the activation function have been reported in literature such as linear, step, logistic, hyperbolic tangent etc. Out of all the functions listed above, hyperbolic tangent (tanh) and logistic sigmoid function are the most commonly used activation functions in feed forward NNs because of its nonlinearity and computational simplicity of its derivative and the same had been used in the analysis. They are defined as a strictly increasing function that exhibits a graceful balance between linear and nonlinear behavior. The response corresponding to logistic sigmoid and hyperbolic tangent

functions are almost similar i.e. S-shaped except the fact that the output of sigmoid for negative signals is zero. The type of the nonlinear function governs the nonlinear characteristics of each neuron and usually has a special feature. This means that the shape of the function is fixed and cannot reach its saturated level without infinitely large inputs. The shape of this function can be varied by trial and error basis or by some auto shaping algorithm so as to increase the flexibility and learning ability of the network. Using this effort of variation of slope of the activation function in our study, the optimal shape of the nonlinear saturated function can be easily tracked which will further lead to neuron saturation prevention, and thus avoids the necessity of using the scaling procedure.

To accomplish this task, the network was trained with modified HLMS algorithm with a variation in slope of the hyperbolic tangent function and logistic sigmoid. The performance of the network was evaluated for all datasets with number of neurons in the hidden layer equal to the number of attributes of the dataset.

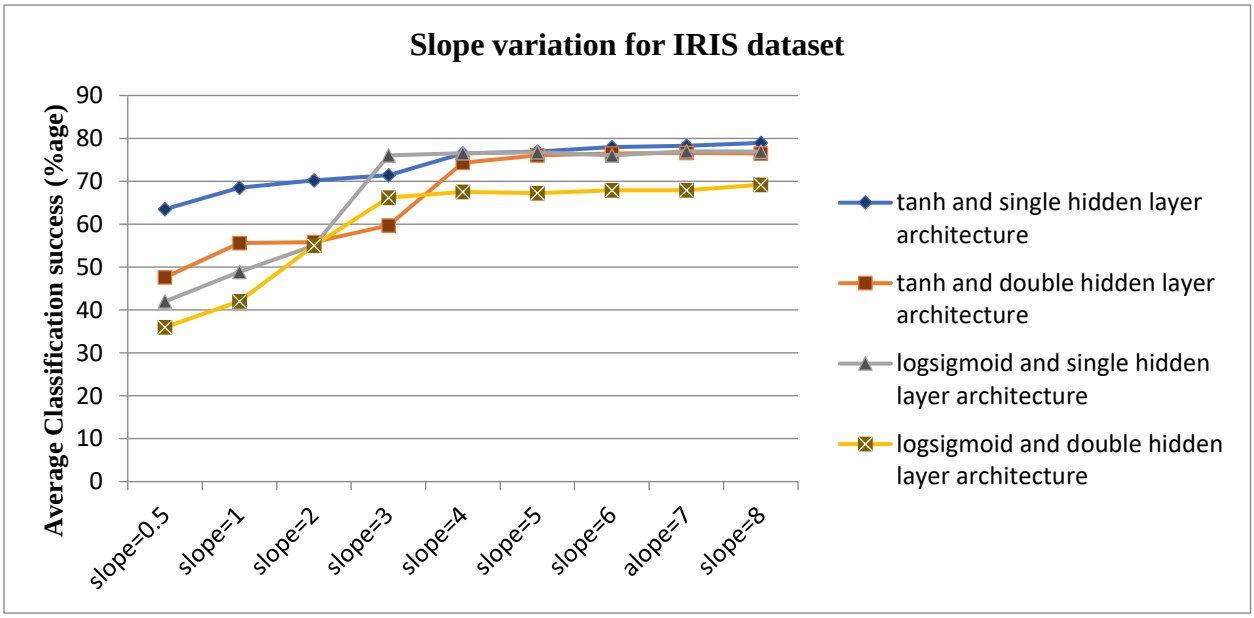


Figure 4.8: Classification success of modified HLMS with variation in slope of tanh and sigmoid activation function for single and double hidden layer architecture for IRIS dataset

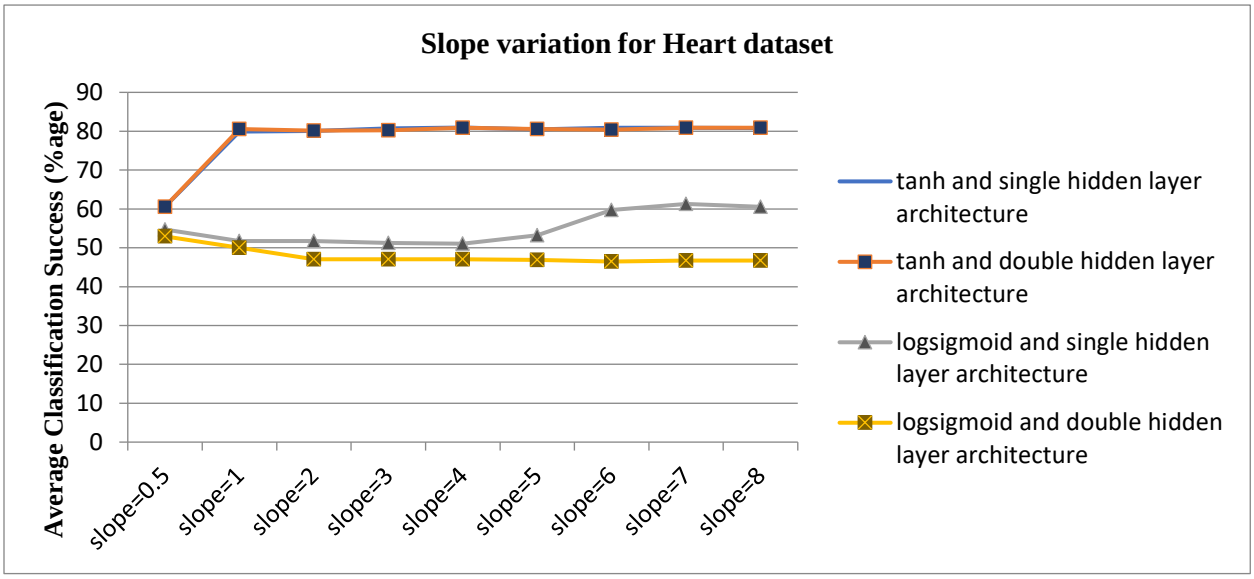


Figure 4.9: Classification success of modified HLMS with variation in slope of tanh and sigmoid activation function for single and double hidden layer architecture for HEART dataset

A set of 20 simulation experiments were conducted on the algorithm and the average performance result was evaluated. Figure 4.8 - 4.11 demonstrates the performance of the modified HLMS algorithm with a variation in slope of the activation function for single and double hidden layer architecture for all datasets respectively. The results of the experiments clearly indicate that there is a significant improvement in the training performance of the network using enlargement of the slope parameter of the neuron activation function. However, the same order of performance has been attained with the use of large number of hidden neurons with complex and computationally intensive network implementation which also lead to extended training time. The usage of the larger values of slope for neuron activation function enabled us to get better classification performance for both the activation functions i.e. hyperbolic tangent and logistic sigmoid from the standard value which is one whereas it deteriorates with decrease in slope. In order to further analyze the performance variation with slope, the activation curves were generated for both log sigmoid and tanh functions for all datasets separately. The reason is, for every dataset, the input data variation is different. For IRIS data, the maximum value is 7.9 and minimum is 0.1. Hence

the x-axis variation used to generate the activation function curve is kept between 0 and 8. Similarly, the range for x-axis variation for all the datasets is tabulated in Table 4.4. For WINE and GESTURE datasets, the input data was sparse, and number of attributes were more, so normalization was done and hence the range was chosen to be between 0 and 1. The curves for both activation functions and for all datasets are shown in figures 4.12 - 4.15.

Table 4.4: X-axis range chosen for different datasets for generating activation function curves

Dataset	Max	Min	Range
IRIS	7.9	0.1	(0 - 8)
HEART	3.97	-3.39	(-4 - 4)
WINE	1	0.06	(0 - 1)
GESTURE	1	0.25	(0 - 1)

After comparing the performance curves shown in figure 4.8 - 4.11 with the corresponding activation curves shown in figures 4.12 - 4.15, it can be deduced that the saturation in the improvement of classification success rate for all the datasets have been attained at almost similar slope values for which the activation curves attain their saturation

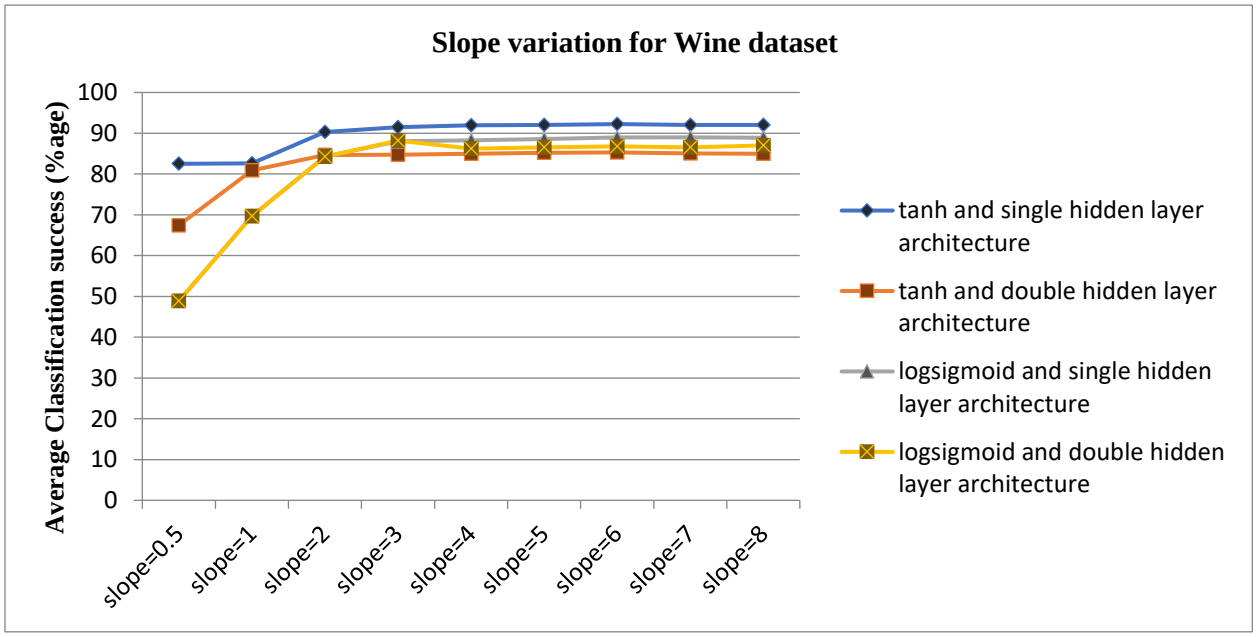


Figure 4.10: Classification success of modified HLMS with variation in slope of tanh and sigmoid activation function for single and double hidden layer architecture for WINE dataset

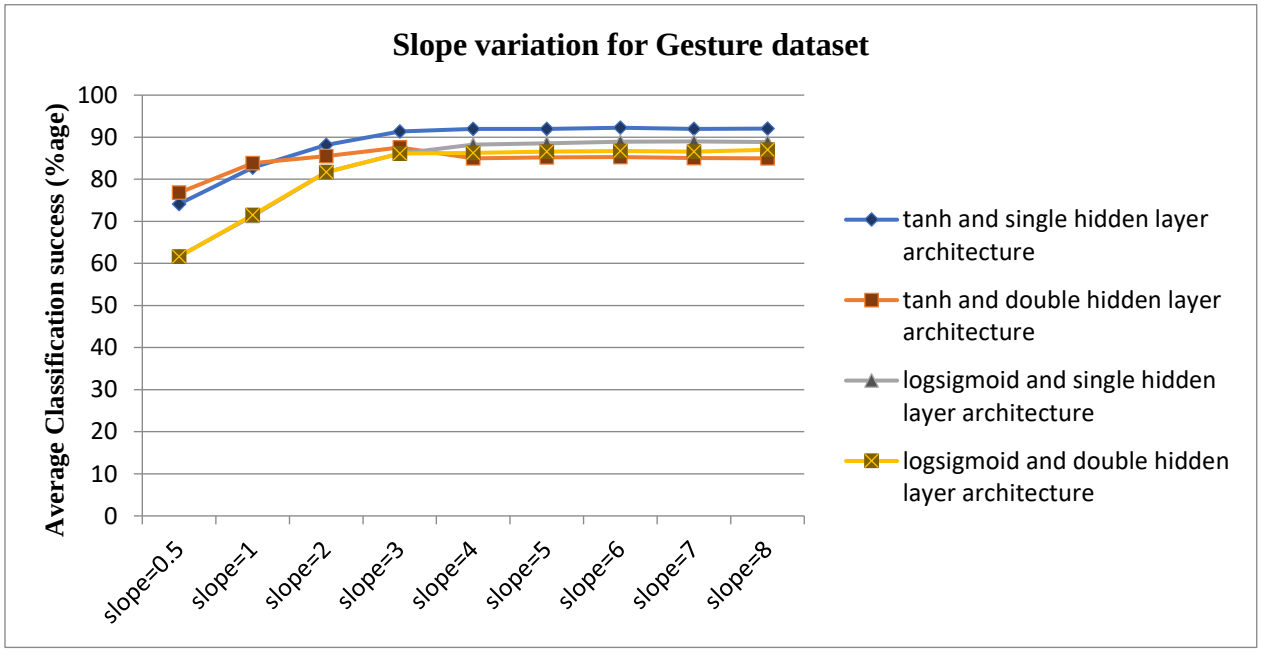

Slope variation for Gesture dataset
Average Classification success (%age)
100
90
80
70
60
50
40
30
20
10
0
slope=0.5
slope=1
slope=2
slope=3
slope=4
slope=5
slope=6
slope=7
slope=8
tanh and single hidden layer architecture
tanh and double hidden layer architecture
logsigmoid and single hidden layer architecture
logsigmoid and double hidden layer architecture

Figure 4.11: Classification success of modified HLMS with variation in slope of tanh and sigmoid activation function for single and double hidden layer architecture for GESTURE dataset

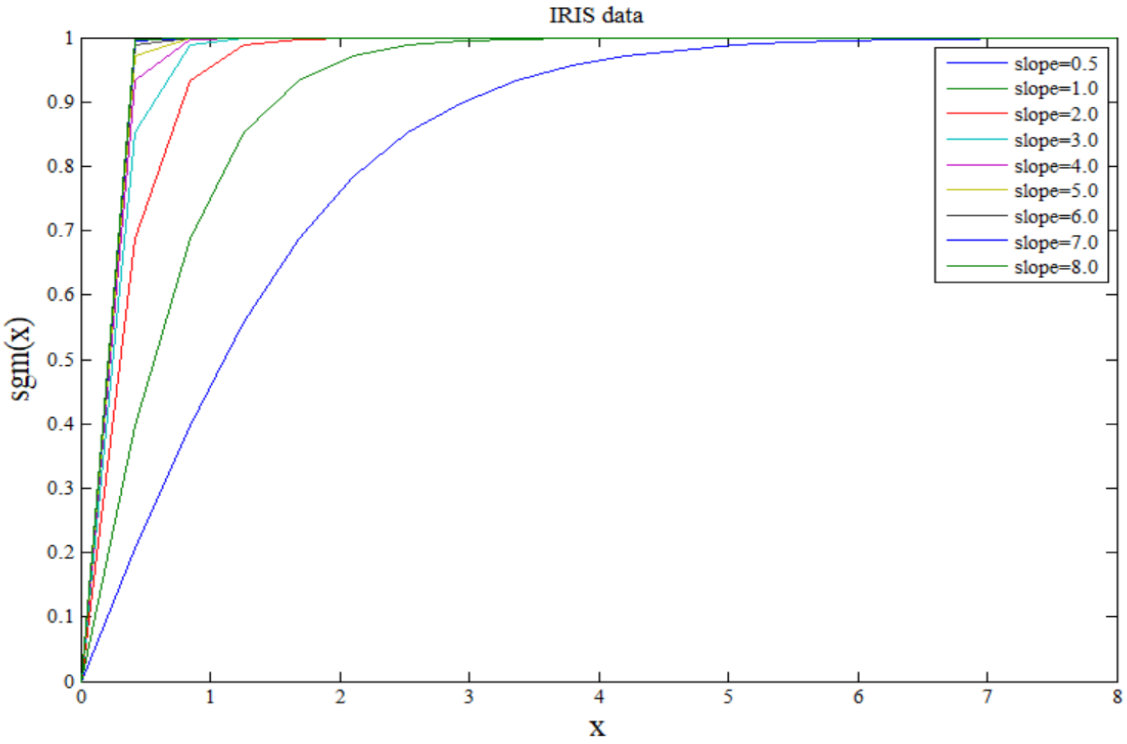


Figure 4.12: Plot of log sigmoid activation function for IRIS data variation and with variation in slope

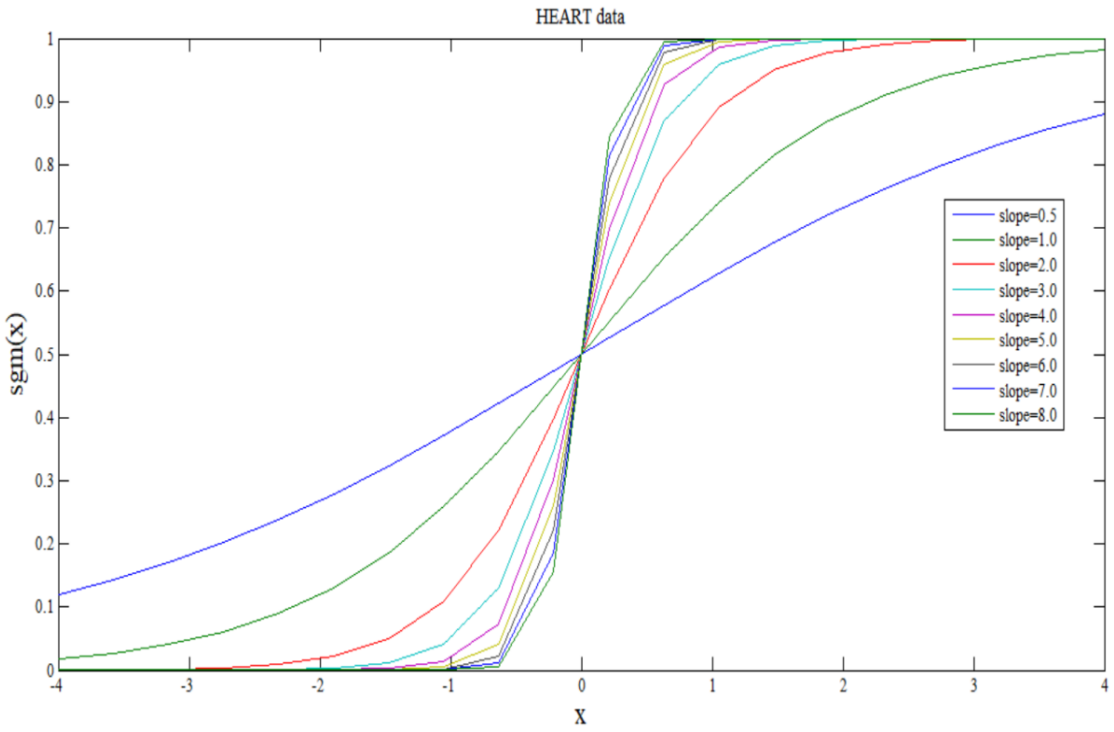


Figure 4.13: Plot of log sigmoid activation function for HEART data variation and with variation in slope

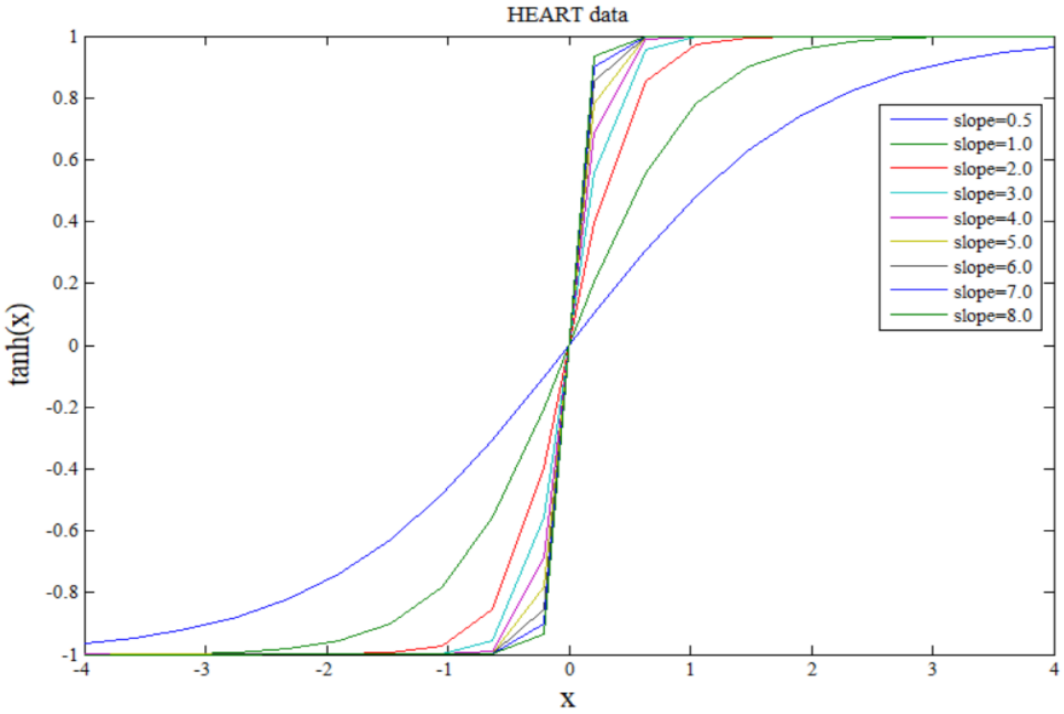


Figure 4.14: Plot of tanh activation function for HEART data variation and with variation in slope

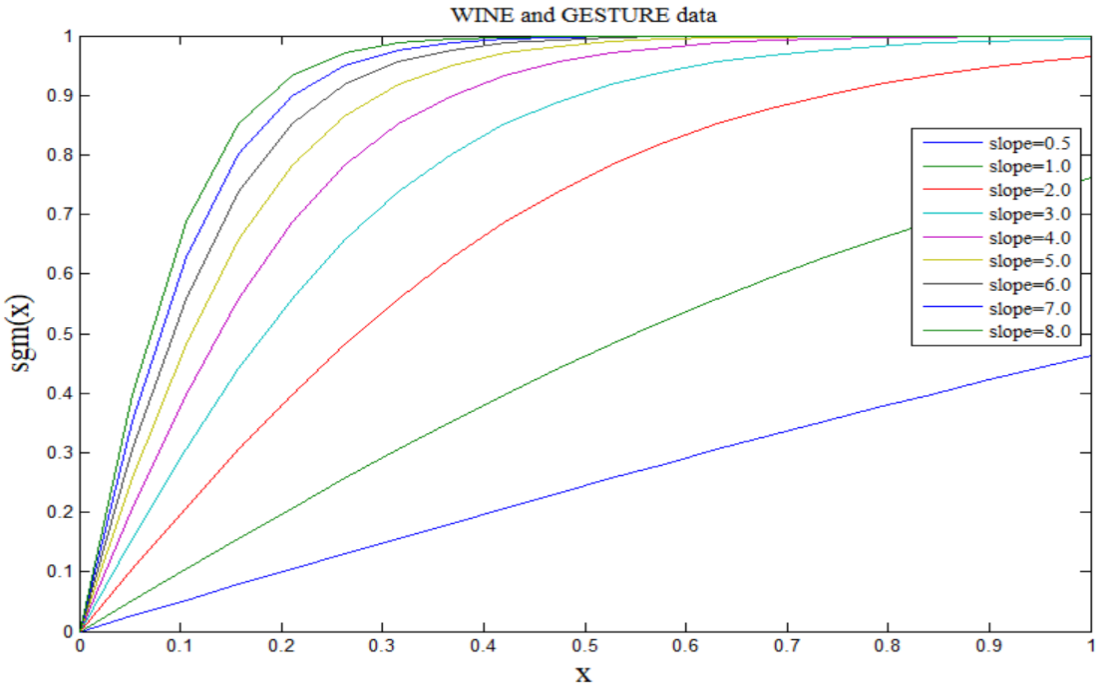


Figure 4.15: Plot of log sigmoid activation function for WINE and GESTURE data variation and with variation in slope

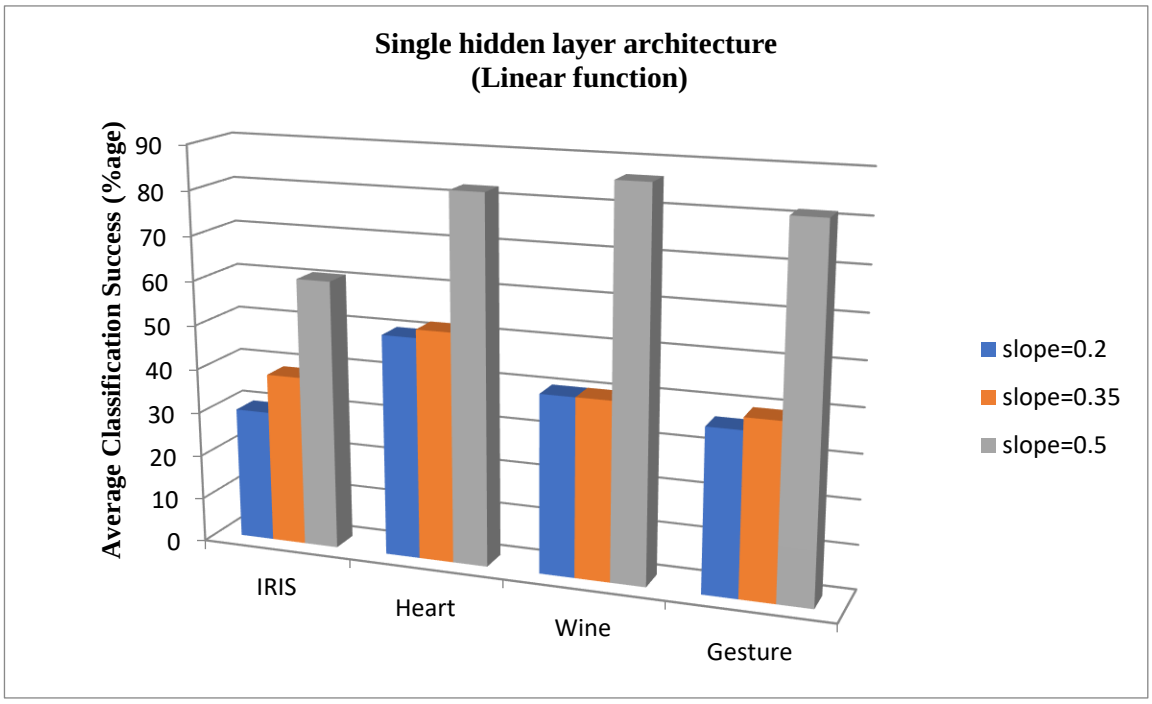


Figure 4.16: Classification success of modified HLMS with variation in slope of linear function (Single hidden layer architecture)

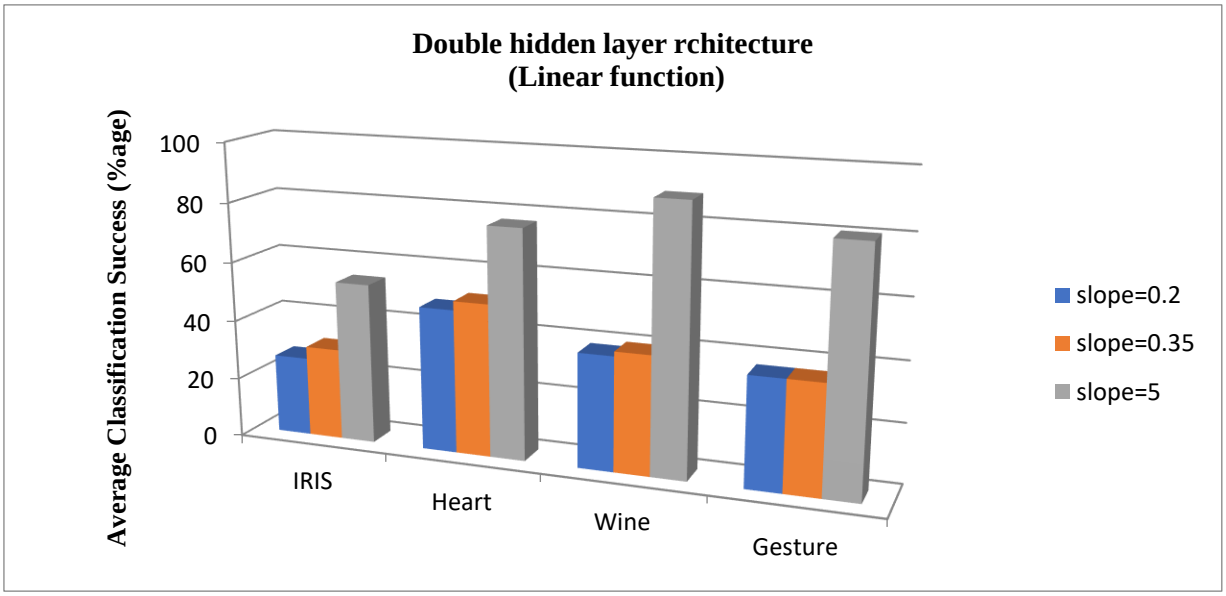


Figure 4.17: Classification success of modified HLMS with variation in slope of linear function (Double hidden layer architecture)

This study has also investigated the performance of the NN with a variation in the slope of the linear activation function and the results are shown in figure 4.16 and 4.17 for single and double hidden layer architecture. The trend performance growth with linear activation function is same as with logistic sigmoid and tanh function. However, the variation in slope parameter of linear function is only from 0.2 to 0.5. Beyond this value, the network does not learn and hence the performance of the network was approximately zero. Therefore, 0.5 is the maximum value that can be used for the analysis of linear activation function.

The analysis of the network performance by varying the slope of the neuron's activation function dictates that slope variation not only improves the average classification success but also offers other benefits. This kind of variation in the network does not change the network topology and does not require extra computation. However, similar improvement in the performance can be obtained by increasing the number of hidden layer neurons but that happens at the cost of excessive computation and usage of resources. Therefore, variation of slope of neuron's activation function is a favorable strategy for performance improvement over variation of number of hidden neurons that demands additional computational resources and time.

4.7 Summary

This chapter presents an implementation of modified HLMS which is a fusion of HLMS, an unsupervised learning algorithm and LMS, a well-known supervised learning mechanism. The union of these two learning algorithms resulted into a new supervised mechanism for training ANNs. The proposed algorithm seems to somehow taken a control over the hidden layer dynamics which was not apparent while using the conventional BP algorithm. By applying HLMS to the hidden layers, the input patterns were clustered in the hidden layer which were finely classified using LMS at the output layer. The recursive computation of the local gradient is not required which was otherwise imperative for adaptation of hidden layer weights and responsible for slow convergence. Additionally, all the synapses in the hidden layers are adapted simultaneously that cater to improve the speed of convergence. Different topologies of ANN (single and double hidden layer) have been trained using both modified HLMS and conventional BP algorithms for several

benchmark data sets. The performance of the ANN has been investigated with a variation in the number of hidden layer neurons and epochs. The simulation results revealed that the network performance showed an upsurge with an increase in the number of hidden layer neurons with an undue usage of network resources, excessive computation and increased training time. On the other hand, similar order of performance has been achieved by simply increasing the slope of the neuron's activation function with least number of hidden layer neurons and no additional overheads. Experiments were also conducted using a variety of activation functions and the effect of varying the slope of these activation functions has been analyzed. It has been concluded from the results that the performance of the algorithm with increasing slope of the activation function is at par with the performance attained with increase in the number of hidden layer neurons without undue overheads. Hence, slope variation can be considered as a better alternative for performance enhancement.

The next chapter discusses about the implementation of MLP as an ANC for de-noising acoustic signals. Apart from this, a novel area efficient structure of adaptive filter has been proposed and implemented for LMS and all its sign variants. The performance of the proposed structure in terms of hardware utilization, delay, power and FPGA family has been done in comparison to other structures viz. direct and transpose form. Furthermore, NN based adaptive filter implementation have also been described along with its estimated hardware utilization.

Chapter 5

Implementation of neural network adaptive filters based on LMS and its sign variants

This chapter presents the design and implementation of NN adaptive filters based on LMS and its sign variants using two different methodologies of BP viz. Gradient Descent with Momentum (GDM) and LM. The analysis has been done to optimize the NN architecture, methodology and algorithm to design the adaptive filters. Furthermore, hardware implementation of a variable step size adaptive filter for de-noising acoustic signals is also presented. Initially, LMS and its sign variants were employed using three different structures viz. direct form, transpose form and the proposed structure and their de-noising performances were compared. The filters are implemented on three different FPGAs viz. Spartan 6, Virtex 6 and Viretx 7 to find out the best device family that can be used to implement an ANC by comparing speed, power and area utilization. Further, with an aim to impart more adaptability and computational simplicity to the design, the adaptive filter was implemented in the form of a single hidden layer ANN trained with BP. The performance and hardware requirements were compared across different methodologies of BP and variants of LMS.

5.1 Introduction

Adaptive filters possess the ability of adaptation to an unknown environment. This family of filters has been widely applied because of its versatility (capable of operating in an unknown system) and low implementation cost compared to their non-adaptive counterparts [281][282]. Adaptive filters have a wide range of signal processing and control applications owing to its capability of tracking time variations of input statistics [283]. Out of all the application areas described in the introduction, active noise cancelling, also called adaptive noise cancelling belongs to the interference cancellation class. Most of the non-deterministic signals occur in real world scenarios such as in case of an airplane cruise where the noise characteristics continuously changes with plane's speed, height and environmental conditions. In such a case, it is imperative to remove noise from the pilot's microphone to ensure proper communication between the pilot and the ATC tower. Adaptive filters come handy in these situations. A real-time, adaptive noise cancellation filter had

been designed and tested under various environmental conditions to repress the broadband, non-stationary, and intense noise often encountered in military tracked vehicles and high-performance aircraft with a wide range of filter parameters being investigated, as surveyed by Kang et al. [284]. The basic structure of ANC is shown in Figure 5.1. Adaptive filters are known to give high performance and are robust to any sort of noise environment [285].

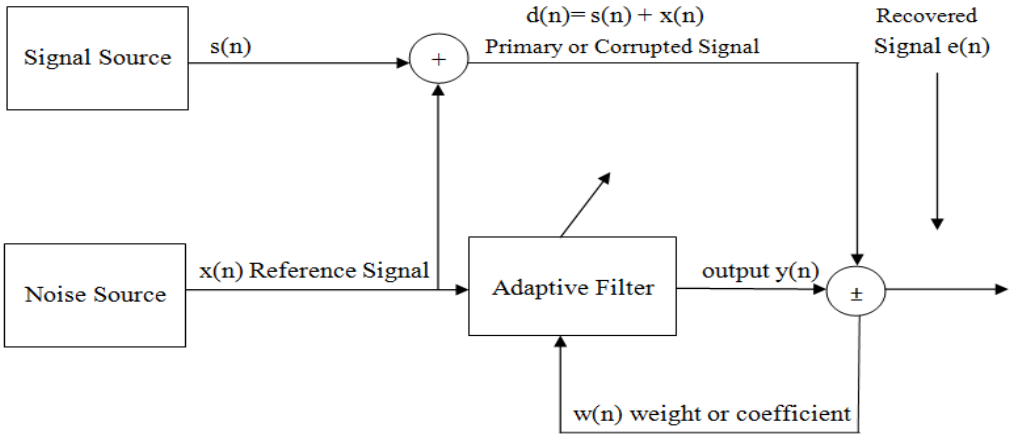


Figure 5.1: Adaptive noise canceller (ANC)

Adaptation algorithm, step size, filter length and structure are some important attributes of adaptive filter that drastically affect their performance [91][92]. There are a large number of adaptation algorithms such as LMS [95], NLMS [286], sign variants of LMS such as Sign-Data (SD) , Sign-Error (SE) and Sign-sign (SS) [228]. Out of all these, SS [287] algorithm is found to be the simplest algorithm to implement it on hardware whereas the performance of Normalized LMS [286] algorithm has been found to be better than that of the SS algorithm. Although, LMS algorithm is a compromise between the SS and NLMS algorithms, there are ample reasons supporting the usage of LMS algorithm in most of the DSP applications. Firstly, it is easy to implement both in software and hardware owing to its less computational complexity. Secondly, it is robust in the sense, it can perform well in the presence of numerical errors caused by finite precision arithmetic. Lastly, its behavior has been analytically characterized to the point where a user can easily set up the system to obtain adequate performance with only limited knowledge about the input and desired response signals [286]. Comparing LMS and its sign variants, it has been noted that SD and SE variants are less complex than LMS whereas more complex than their counterpart; SS algorithm. However, the performance of SD algorithm is as good as LMS algorithm whereas the performance of SE and

SS algorithm is poor. Step size and filter length are the two major parameters that affects the performance of the LMS adaptive filter. If the step size is large, then the coefficients of the adaptive filter will diverge instead of converging and hence the filter will not be able to remove noise and decrease error. On the contrary, if the step size is too small then it will increase the convergence time of the LMS filter coefficients and hence the adaptive filter will remove noise or decrease error with a slower speed [286]. This ambiguity can be resolved by using a variable step size adaptive filter. Many algorithms with variable step size and selective coefficients update and partial update have been reported for reduced computational complexity of LMS variants [288][289]. Length of adaptive filter is also important because it changes the number of iterations required to minimize error signal. Larger the length, lesser will be the number of iterations to minimize error and vice-versa. The structure of adaptive filter also plays an important role in filtering like in some cases basic (Direct-Form) adaptive filters exhibit better filtering performance over transposed form ones [290].

Conventionally, ANNs were used to cater to the classification or function approximation tasks by mapping input feature space to their respective output feature vectors. However, they can also be efficiently used to obtain reasonably good accuracy in removal of noise or elegantly filtering out the desired signals. At a high level, the filtering problem is a special class of function approximation problem in which the function values are represented using time series. Conventional parametric approaches of filtering involve mathematical modeling of the signal characteristics, which is then used to accomplish the filtering. The approach itself is a complex task containing many steps for instance model hypothesis, identification and estimation of model parameters and their verification. However, using a NN, the modeling phase can be bypassed, and nonlinear and nonparametric signal filtering can be performed. Therefore, using an ANN is another viable option for filtering unwanted information [291]. It would be interesting to compare the performance of an ANN as an adaptive filter with that of the filters implemented conventionally both in terms of computational requirements and hardware overhead. Motivated by this possibility, this chapter first describes implementation of a conventional adaptive noise canceller using variants of LMS and thereafter outlines employment of an ANN with some improvisations to achieve the same filtering objectives.

In terms of choice of an adaptation methodology it is evident from the literature that researchers have by and large ignored the performance comparison of different variants of LMS algorithm for a given de-noising task. This has served as a motivation to implement and analyze four different variants of the standard LMS algorithm using two popular NN methodologies viz. GDM and LM. The highlights of this chapter are outlined as:

- Design, implementation and analysis of an adaptive filter using LMS algorithm and its sign variants with two different methodologies of BP; GDM and LM in order to identify the best training methodology, algorithm and architecture to work upon.
- Design and verification of an Adaptive noise canceller using three filter structures (direct, transposed and a novel structure) and using LMS algorithm and its sign variants in SIMULINK.
- Hardware implementation of the designed ANC on three Xilinx FPGAs namely Spartan6, Virtex6 and Virtex7.
- Comparative performance analysis of speed, power and device utilization parameters of the ANC implemented on individual device families.

5.2 Adaptive filter algorithms and structures

5.2.1 LMS algorithm

LMS algorithm [95] is used to imitate a desired filter by adjusting the filter parameters that will try to generate the least mean squares of the error signal. It is a stochastic gradient descent method in the sense, the filter is only adapted based on the error at the current time. During the n^{th} iteration, the filter coefficients are adjusted according to the following equation:

$$w(n+1) = w(n) + \mu.e(n).x(n) \tag{5.1}$$

where, *e(n)* is the error signal as explained in equation (5.1) and μ is the step size or the adaptation coefficient and *y(n)* in the error signal is the filter output at the n^{th} iteration and is given as

$$y(n) = x(n)^T.w(n) \tag{5.2}$$

where *x(n)* and *w(n)* corresponds to input sample vector and weight vector of the n^{th}order LMS adaptive filter at the n^{th} iteration, respectively.

5.2.2 Sign- LMS algorithm

Since the creation of LMS, several modifications have been proposed to the original LMS formulation such as the signum function is applied either at input signal vector or at error signal vector or at both the signal vectors. These algorithms came into existence for real valued data, is due to its ease of implementation. The computational cost of sign based LMS algorithm will be much less than that of the original LMS algorithm [228]. This is because, the evaluation of these $\mu x(n)sign[e(n)]$ or $\mu sign[x(n)]e(n)$ or $\mu sign[x(n)]sign[e(n)]$ product terms can be very efficiently digitally implemented by means of shift registers, and the multiplication terms can be ignored for a generic η(step-size). This simplification is not possible for LMS because it does not use sign of data or error or both.

Sign based LMS algorithm are categorized into three sub divisions: SD, SE and SS algorithm.
SD Algorithm: - This algorithm applies the signum function only to the input vector *x(n)* and updates the adaptive filter coefficients according to the following equation:

$$w(n+1) = w(n) + \eta . sign(x(n)) . e(n) \tag{5.3}$$

SE Algorithm: - This algorithm applies the signum function only to the error vector *e(n)* and updates the adaptive filter coefficients according to the following equation:

$$w(n+1) = w(n) + \eta . sign(e(n)) . x(n) \tag{5.4}$$

SS Algorithm: - This algorithm applies the signum function to both the error signal *e(n)* and the input signal vector *x(n)* and updates the adaptive filter coefficients according to the following equation:

$$w(n+1) = w(n) + \eta . sign(e(n)) . sign(x(n)) \tag{5.5}$$

5.2.3 Adaptive filter structures

For the implementation of adaptive filters, several structures have been employed as shown in Figures 5.2 - 5.3. It had been observed that the silicon area consumption, hardware utilization efficiency and clock speed of transpose form adaptive filters are better than the direct form and cross coupled form adaptive filters. However, the power dissipation (mainly dynamic power) of

transpose form adaptive filter is higher as compared to direct form and cross coupled form adaptive filter as its clock speed is higher.

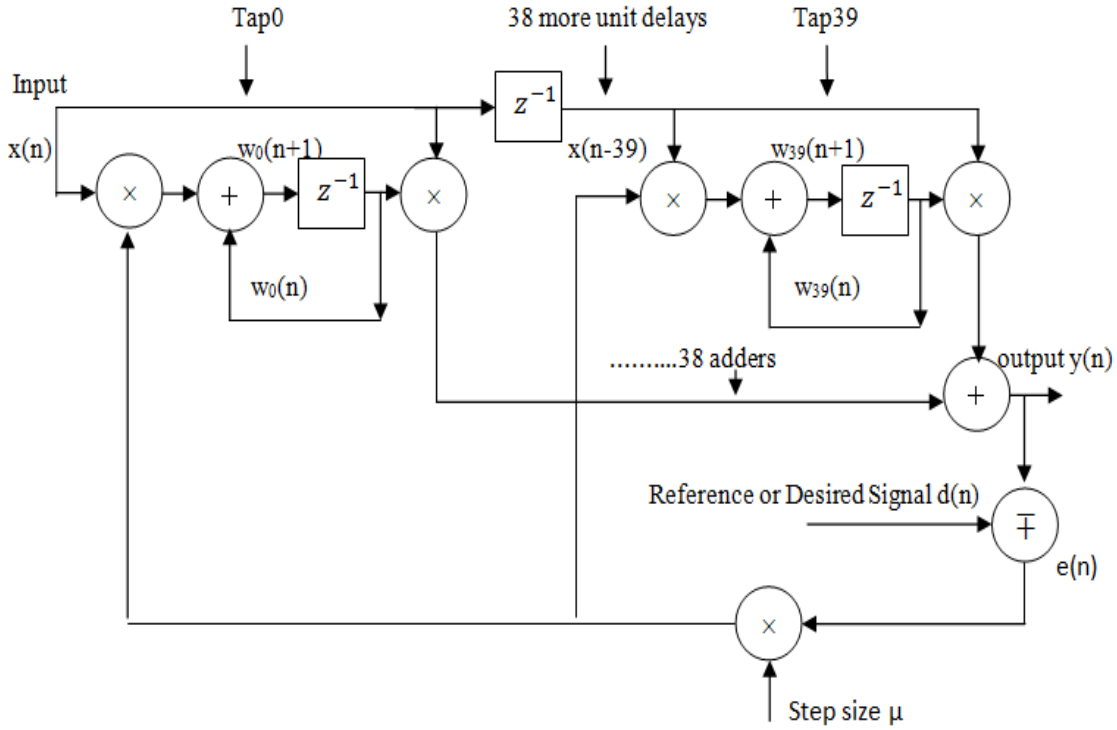


Figure 5.2: Direct-form adaptive Filter

A new structure has been proposed as shown in figure 5.4 that combines the advantage of both the direct form and transpose form adaptive filter. It had better silicon area utilization, hardware utilization efficiency and clock speed as compared to direct form adaptive filters and its power dissipation is lower than transpose form adaptive filter. The new structure has been designed by modifying direct form structure; by replacing the chain of cascaded unit delays by a single delay block of specific value. The value of this single delay is dependent upon the length of the filter. In the proposed structure, delay block delays the input by the specified number of sample periods and outputs all the delayed versions. In other words, it is used to discretize a signal in time or resample a signal at a different rate.

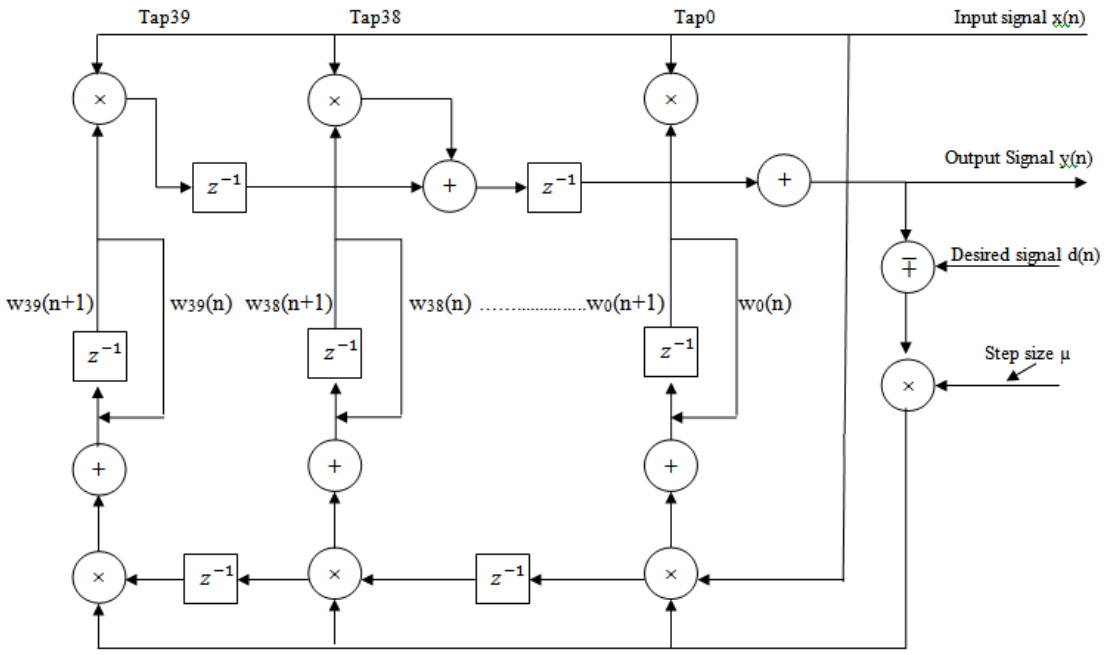


Figure 5.3: Transpose-form adaptive filter

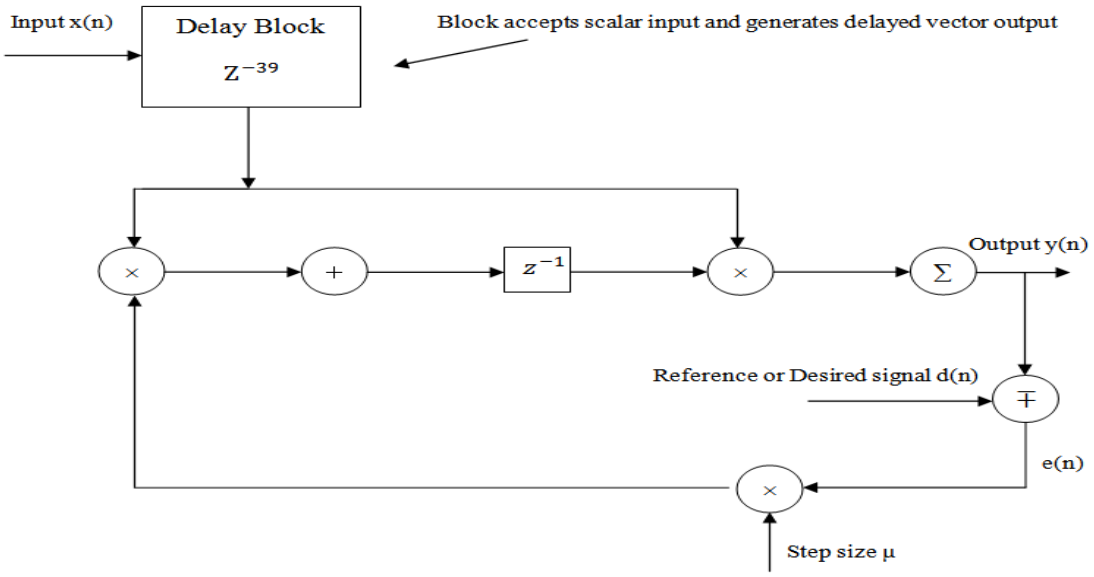


Figure 5.4: Proposed-form adaptive filter

5.3 Design of ANC using different filter structures

5.3.1 Implementation

An adaptive noise canceller was designed using the three filter structures (having filter length 40); Direct, transpose and the proposed form. Primarily, there are two ways to design an ANC in SIMULINK. First one involves the use of a digital filter as a block which is then cascaded with an adaptive algorithm. Second technique is based on the implementation of a weight loop equation of adaptive algorithm using different structures (Direct, transpose and the proposed form) and using different blocks like multipliers, adders, multiplexers etc. that will act as an unit adaptive block. This unit block will be duplicated m times in order to generate an m tap filter. No doubt, the first method is quite simple and takes very less time to implement but it cannot be used for denoising of complex input signals. So, the second design approach is used in our implementation. The adaptive filters are first designed using standard cells in Simulink then simulated with noise corrupted acoustic signals. The denoising performance of the filters was evaluated repeatedly. Adaptive algorithm used in the filter structures have been designed using LMS algorithm and its sign variants.

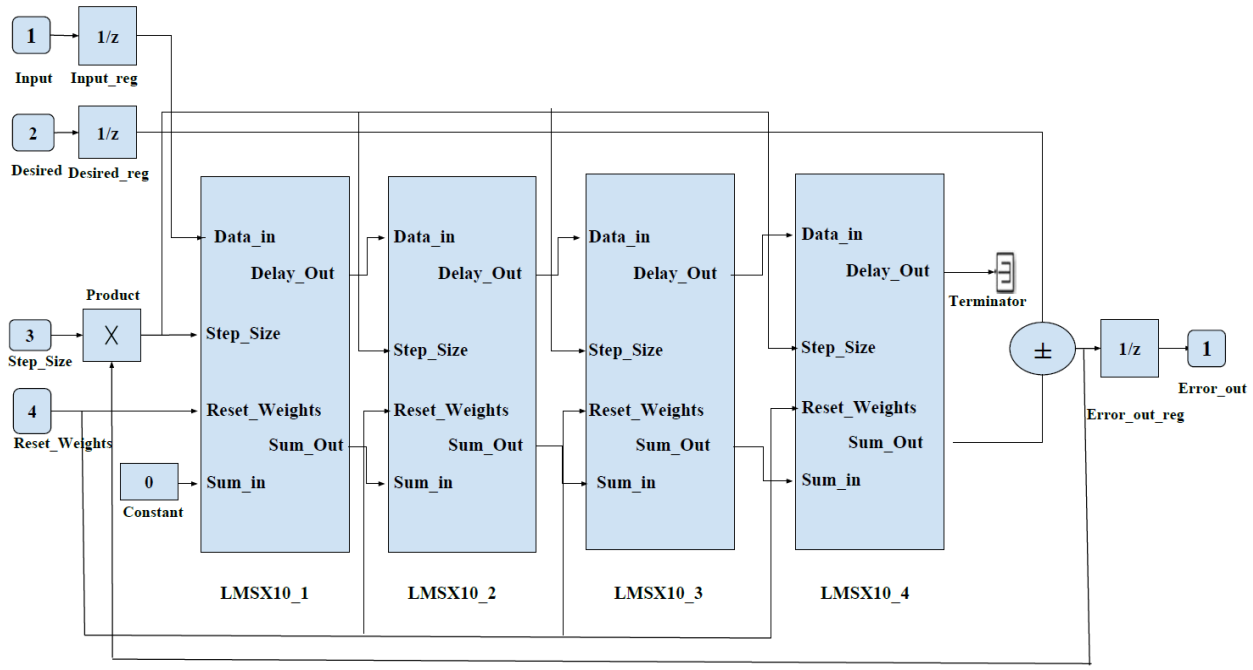


Figure 5.5: SIMULINK model of a 40 tap LMS adaptive filter implemented using Direct-form and Transpose- form structure (Outer structure)

A 40-tap LMS adaptive filter implemented using Direct-form is shown in Figure 5.5. The four blocks namely LMSx10_1, LMSx10_2, LMSx10_3 and LMSx10_4 represents a 10 tap adaptive filter. The internal structure of these blocks for the direct form structure is shown in Figure 5.6. The 10 blocks namely LMS_Tap1 to LMS_Tap10 represents a unit tap adaptive filter. This unit tap adaptive filter is nothing but the implementation of the weight update loop equation of the algorithm used in designing the adaptive filter. The unit order LMS adaptive filter implemented using direct-form is shown in Figure 5.7.

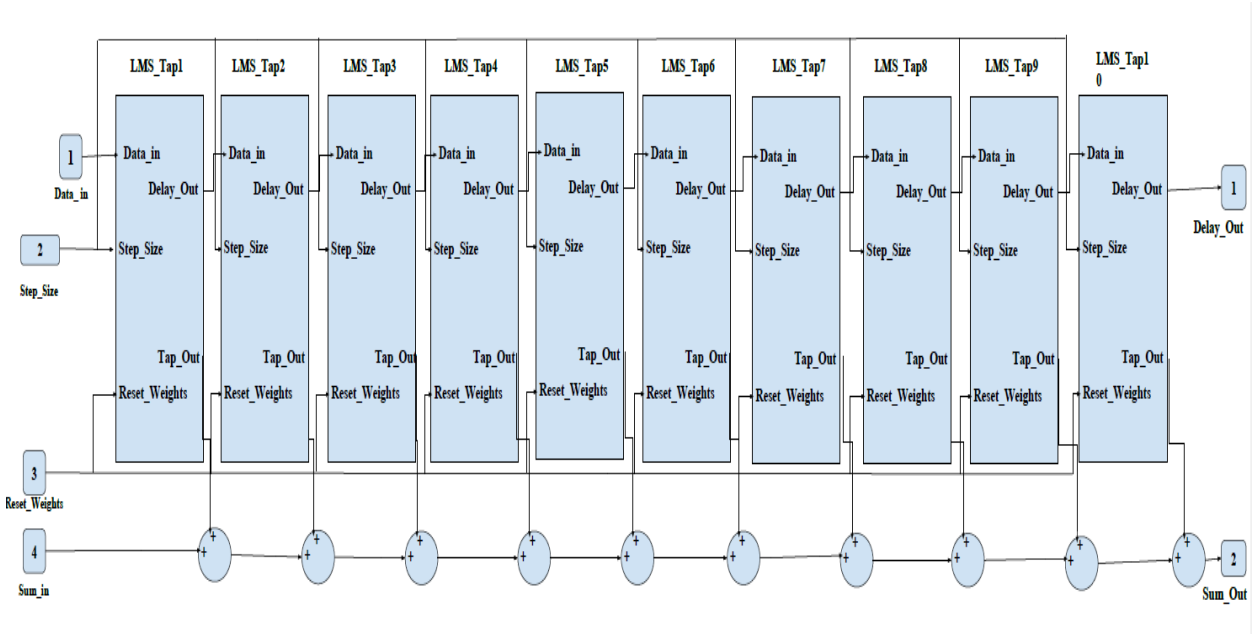


Figure 5.6: SIMULINK model of a 10 tap LMS adaptive filter implemented using Direct-form structure

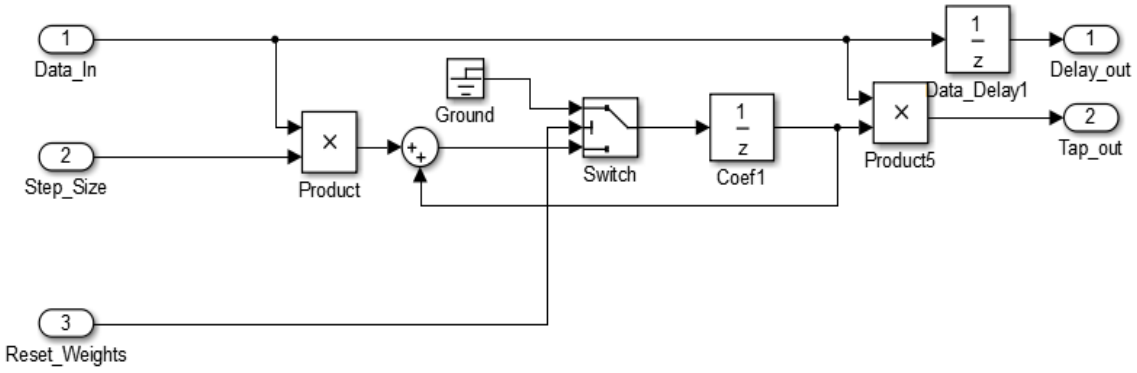


Figure 5.7: SIMULINK model of a unit order LMS adaptive filter implemented using Direct-form structure

Furthermore, the simulink model of the 40 tap LMS adaptive filter using the proposed structure is also shown in Figure 5.8. The proposed structure uses a delay block that accepts scalar input and generates vector delayed outputs. The designed adaptive filter using all the structures and algorithms mentioned above was then used to implement an ANC which was used to remove the noise from the audio signal. For this, an acoustic environment was also created in SIMULINK. The noise source used in the acoustic environment is Gaussian noise which is then converted into discrete form and is sent as input signal to the adaptive filter through the exterior mic port present in the acoustic environment. Figures 5.9 (a) to 5.9 (c) illustrates the audio input, the noise signal and noise corrupted input signal respectively. The signal recovered from noise when LMS and its variants have been employed are shown in Figures 5.9 (d) to 5.9 (g). After the ANC was designed and simulated in the SIMULINK it needs to be implemented on an FPGA/ASIC. For this, the VHDL code of the designed system was generated using Matlab HDL coder. This VHDL code was then simulated, synthesized using Xilinx 14.5. The synthesized code was then implemented on three Xilinx FPGAs namely spatran6, virtex6 and virtex7. The functionality of the implemented code on FPGA was then verified using Chipscope Pro.

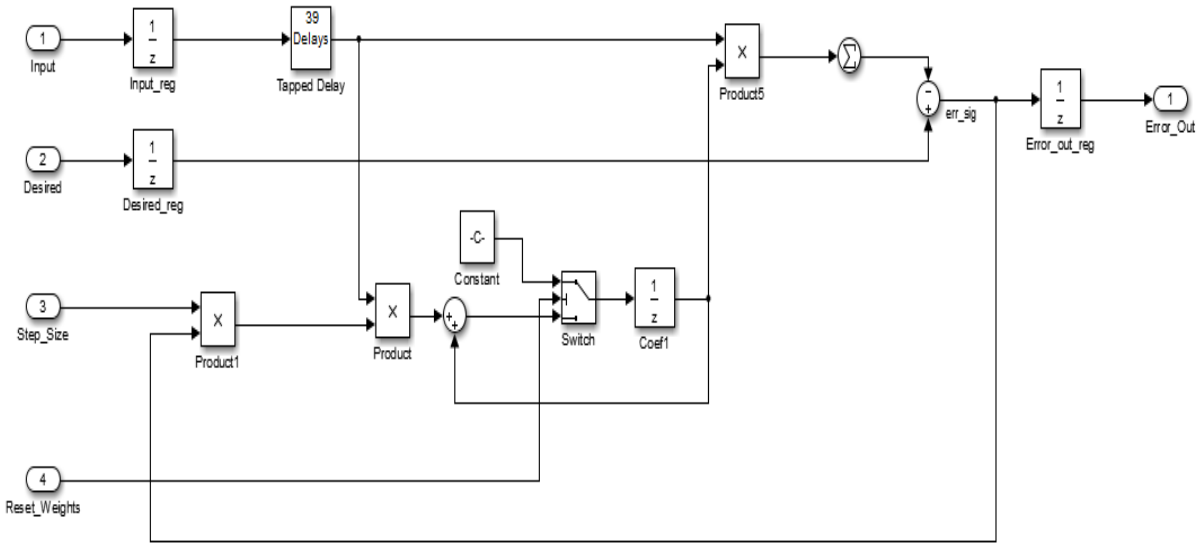
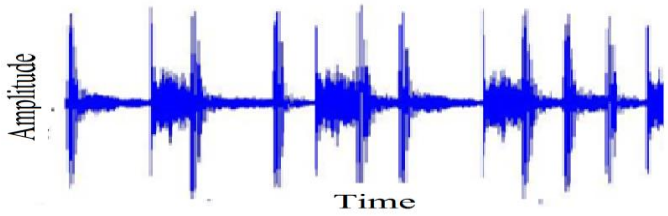
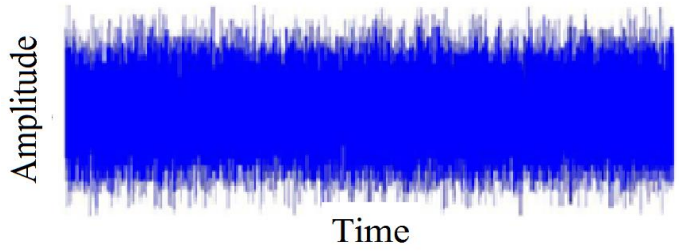


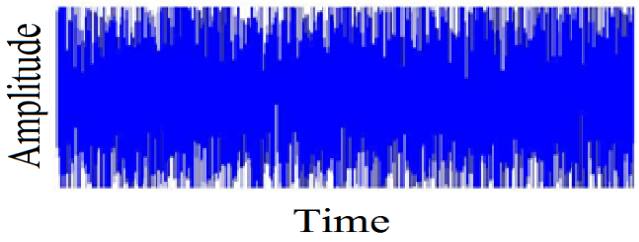
Figure 5.8: SIMULINK model of a 40 tap LMS adaptive filter implemented using Proposed-form structure



5.9(a): Input signal (Audio wave)

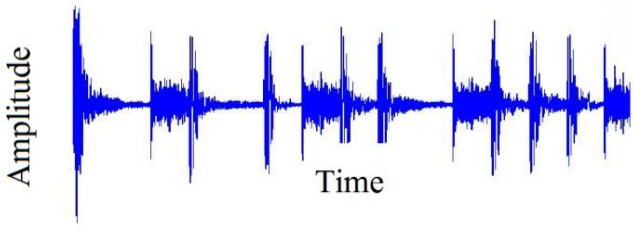


5.9(b): Low frequency noise signal



5.9(c): Noise and input signal

5.9(d): LMS output



5.9(e): SD output

5.9(f): SE output

5.9(g): SS output

Figure 5.9: Input and output signal diagram of LMS and its sign variants

5.3.2 Critical path analysis

In general, the critical path [292] of an LMS adaptive filter implemented using the direct and transpose form can be represented as:

$$T_{CRITICAL} = T_{ERROR} + T_{UPDATE} \tag{5.6}$$

where, T_{ERROR} and T_{UPDATE} are, the time required to calculate the error and to update the weights respectively.

5.3.2.1 Direct form LMS adaptive filter

During implementation of LMS adaptive filters, it is required that a new arithmetic operation starts only when the previous operation has been completed and the result is available for the next computation. For example, the addition operation in a multiply-add operation starts as soon as the complete product is available to the adder. Considering the above requirement, the critical path for a direct form LMS adaptive filter can be estimated as:

$$T_{CRITICAL} = 2T_{mult} + (N + 1)T_{add} \tag{5.7}$$

This estimation of critical path is quiet high that might even exceed the required sample period in many cases. Therefore, there is a great need to reduce this delay which will be discussed as follows.

The delay analysis typically requires two calculations as stated in equation (5.6). Considering the error computation part of the structure shown in figure 5.2, it is clear that all the multiplications are executed concurrently, and addition operations will be executed in an adder tree. As all the multiplication operations happen simultaneously, the delay is only one T_{mult}. However, the addition operation is executed in a tree fashion as shown in figure 5.10, Hence, the delay of adder tree is $log_2 N$ times the delay of a single adder where N is the number of bits to be added in a tree. Therefore, the total delay for the error computation becomes:

$$T_{ERROR} = T_{mult} + (log_2 N + 1)T_{add} \tag{5.8}$$

The extra T_{add} is due to the extra adder/ subtractor required for the computation of the final error signal.

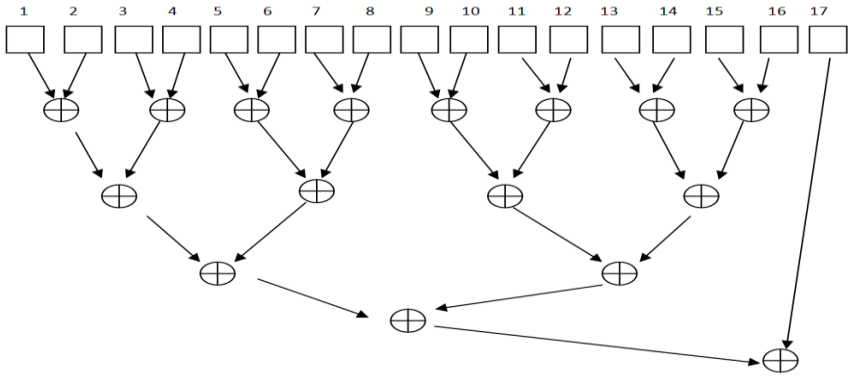


Figure 5.10: Adder tree

For the calculation of T_{UPDATE} of equation (5.6), if learning rate/ step-size parameter μ is assumed to be a power of *2* fraction, then multiplication of μ with error term can be implemented by a hardwired shift without involving additional hardware and hence delay. Thus, the critical path for this block will be only one multiply and add operation and can be stated as

$$T_{UPDATE} = T_{mult} + T_{\text{add}} \tag{5.9}$$

Using equation (5.8) & (5.9) in (5.6), the critical path of the Direct form structure becomes

$$T_{CRITICAL} = 2T_{mult} + (log_2 N+2)T_{add} \tag{5.10}$$

5.3.2.2 Transpose form LMS adaptive filter

In the transpose form structure shown in figure 5.3, all the multiplication operations are again performed simultaneously, and the product is transferred through the registers to be added with the subsequent products. The critical path of this complete unit is comprised of the last multiply and add unit. Multiplication operation is independent whereas the adder is dependent upon the result of previous addition operation. Therefore, the last adder has to wait for the previous adder results. In addition to this, the addition operation starts as soon as the least significant bits of the product are available. Therefore, the delay due to the addition operation till the calculation of error signal is $log_2(N+1)T_{add}$. Hence, the critical path of the error computation part can be estimated as:

$$T_{ERROR} = T_{mult} + log_2(N+1)T_{add} \tag{5.11}$$

Similarly, the critical path T_{UPDATE} can be calculated as:

$$T_{\text{UPDATE}} = T_{mult} + T_{add} \tag{5.12}$$

Using equation (5.11) and (5.12) in (5.6), the critical path for the Transpose form LMS adaptive filter would be

$$T_{CRITICAL} = 2T_{mult} + [\{\log_2(\text{N}+1)\}+1]\,T_{add} \tag{5.13}$$

Comparing equations (5.10) and (5.13), it is clear that the critical path of Transpose form is comparable although slightly higher than the Direct form LMS adaptive filter for smaller values of N. However, this difference becomes significant for larger values of N.

5.3.2.3 The proposed form LMS adaptive filter

In the proposed structure shown in figure 5.4, the multiplication operations are performed again concurrently, and all the product terms are fed to the adder tree. The process is almost similar to the direct form implementation of the LMS adaptive filter. The difference lies in the simultaneous generation of delayed vector inputs that avoids the requirement of extra registers for storing the intermediate delayed inputs which incurs a significant benefit in terms of silicon area reduction if implemented on hardware.

Therefore, the critical path for the proposed structure is similarly found to be

$$T_{CRITICAL} = [T_{mult} + (log_2 N+1)T_{add}] + [T_{mult} + T_{add}] \quad (5.14)$$

This can be further simplified as

$$T_{CRITICAL} = 2T_{mult} + (log_2 N+2)T_{add} \quad (5.15)$$

It is evident from equation (5.15) that the critical path of the proposed structure is comparable to that of the transpose form. However, the major benefit of the proposed structure is in terms of reduction in the requirement of continuous delay elements registers. Undoubtedly, these elements are one of the most critical functional blocks with respect to chip area and power consumption. For that reason, a reduction in the number of registers will significantly minimize the hardware requirement which is a boon for the today's era of portable devices.

5.4 Implementation of NN adaptive filter

Adaptive filtering is of central importance in many applications of signal processing, such as modeling, estimation and detection of signals as discussed in section 1.2. Till date, the bulk of adaptive filtering theory is devoted to classical approach of adaptive filtering which rely on linear modeling techniques and have been extensively studied, and appropriate for many purposes in signal processing. Linear approach is widely used for system and signal modeling, due to their

simplicity, and due to the fact, that in many cases (such as the estimation of gaussian signals), they are optimal. Despite its popularity, this approach remains inappropriate for modeling nonlinear systems [293][294]. Unfortunately, when dealing with non-linear systems, no general adaptation algorithm is available, so that heuristic approaches are used. Biomedical signals such as ECG, Electromyography (EMG) etc. are nonlinear signals generated from a nonlinear system-the human body and it is difficult to adapt to a nonlinear signal using a linear model. Since, ANNs are inherently non-linear models [295] and are known as universal approximants [52][53], that can be used to approximate any non-linear function, ANN-based filtering methods are potentially useful for signals with inherent nonlinearity. Furthermore, the process of adaptation of filters parameters and the training of NNs involve gradient techniques as explained in section 1.2. Hence, NNs can be regarded as a class of non-linear adaptive filters, which are quite general because of the ability of feedforward nets to approximate non-linear functions [296]. There are evidences of the use of ANN as an adaptive filter in the recent past. Monfared and Enke [50] have proved that NN based adaptive filters are able to remove noise and hence extract information from non-stationary and random walk processes. Recently, Quazi et al. [51] investigated a hybrid learning approach for NN enhanced adaptive filtering to eliminate the artifacts from the EEG dataset. They have concluded that this NN based hybrid approach prove to provide better signal to noise ratio in comparison to other recently proposed approaches.

5.4.1 Design methodology

In this implementation of NN adaptive filter, two well approved training methodologies: LM and GDM have been employed. It is a well-established fact that BP algorithm dispersed the dark clouds on the field of ANN and could be regarded as one of the most significant breakthroughs for training NNs. LM and GDM algorithms were developed to overcome the limitations imposed by BP that are slow convergence and trapping in local minima. The momentum term in GDM allows the network to ignore small features in the error surface and hence help to slide through such minima. The coefficients of the filter using GDM are updated according to the following equation:

$$w(n+1) = w(n) + \alpha.\Delta w(n-1) + \eta.g(n) \tag{5.16}$$

where $g(n)$ is the first order derivative of the error vector

α is the momentum term

$\Delta w(n-1)$ is the weight update term in the previous iteration.

LM is the fastest BP algorithm available and has stable convergence. However, it has drawback of memory and computational overhead caused due to calculation of the gradient and approximated Hessian Matrix. It is also the default training function in Matlab. The coefficients of the filter using LM are updated according to the following equation:

$$w(n+1) = w(n) + (J^T(n)J(n) + \eta.I)^{-1}J(n).e(n) \quad (5.17)$$

where $J(n)$ is the Jacobian of the error vector

5.4.2 Results and Discussion

In this implementation, adaptive filters have been designed and implemented in Matlab with LMS and its sign variants and using both the training methodologies described above. Experiments were performed to filter out the noise from the corrupted signal using network with single hidden layer architecture with number of neurons in the hidden layer varying from 2-8. Actual desired signal, noise corrupted signal and noise signal were generated using Matlab. The noise vector is normally distributed random numbers with zero mean and unity standard deviation and is generated using *randn* function of Matlab. The input was made two dimensional by considering square of the signal as the second dimension. The dataset was first partitioned and 70% and 30% of the samples were assigned to training and test sets respectively. Five-fold cross validation was then performed to rule out over fitting. The number of hidden layer neurons was optimized experimentally. The target was preprocessed using Soft Plus (SP) function as shown in figure 5.11. The function is described by the following equation:

$$y = \ln(1 + e^x) \quad (5.18)$$

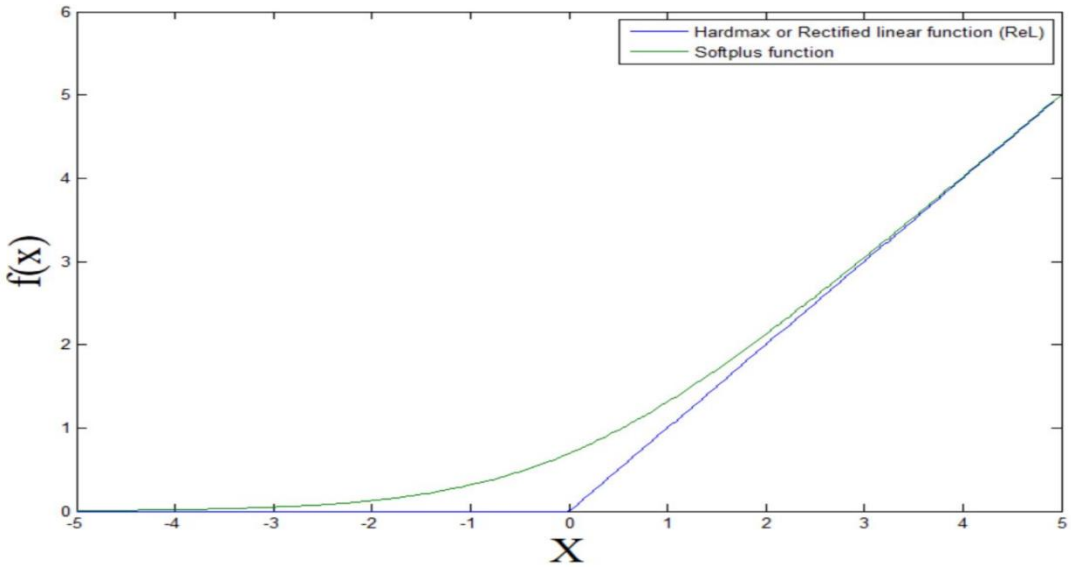


Figure 5.11: The soft plus (SP) function

The reconstructed signal (noise free signal) i.e. the output of the filter is made to pass through an Inverse Soft Plus (ISP) function so as to obtain the original signal. Figure 5.12 and 5.13 shows the training performance in terms of MSE of adaptive filter with variation in number of hidden layer neurons for both methodologies. It has been found that the MSE is approximately same for LMS and SD in LM methodology which implies that the training performance of minimally configured size architecture is at par with other architectures. So, 2-2-1 configuration can be considered as an optimum one for filtering. Similarly, the optimum architecture for SE, SS algorithms for LM methodology is also found to be 2-2-1.

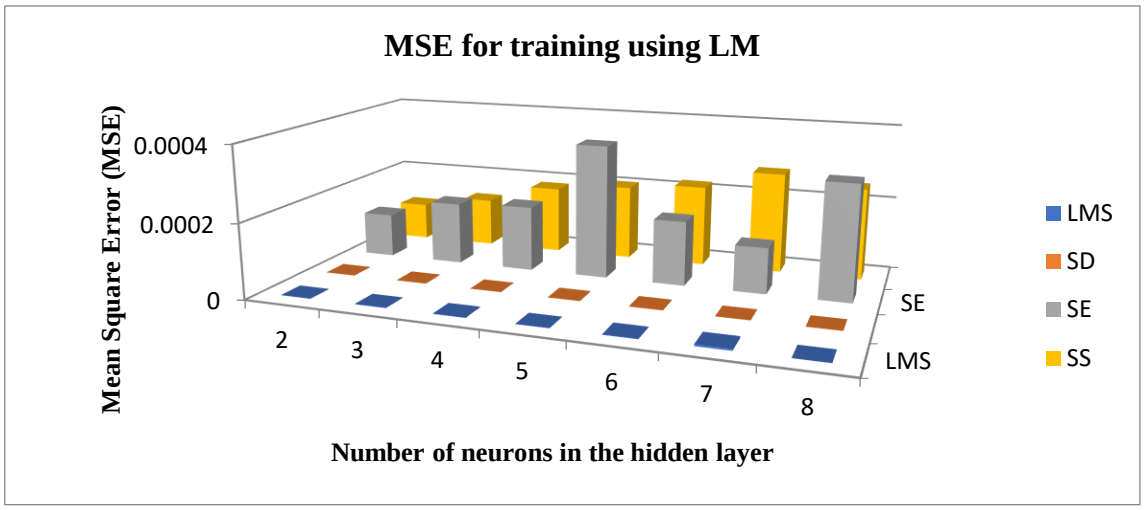


Figure 5.12: Training performance of LM for all variants of LMS

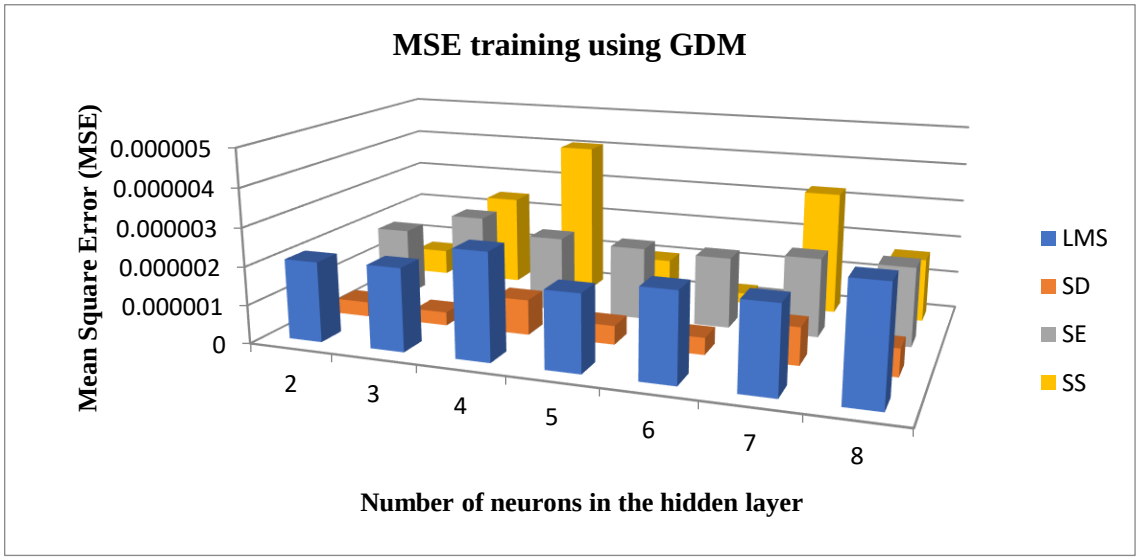
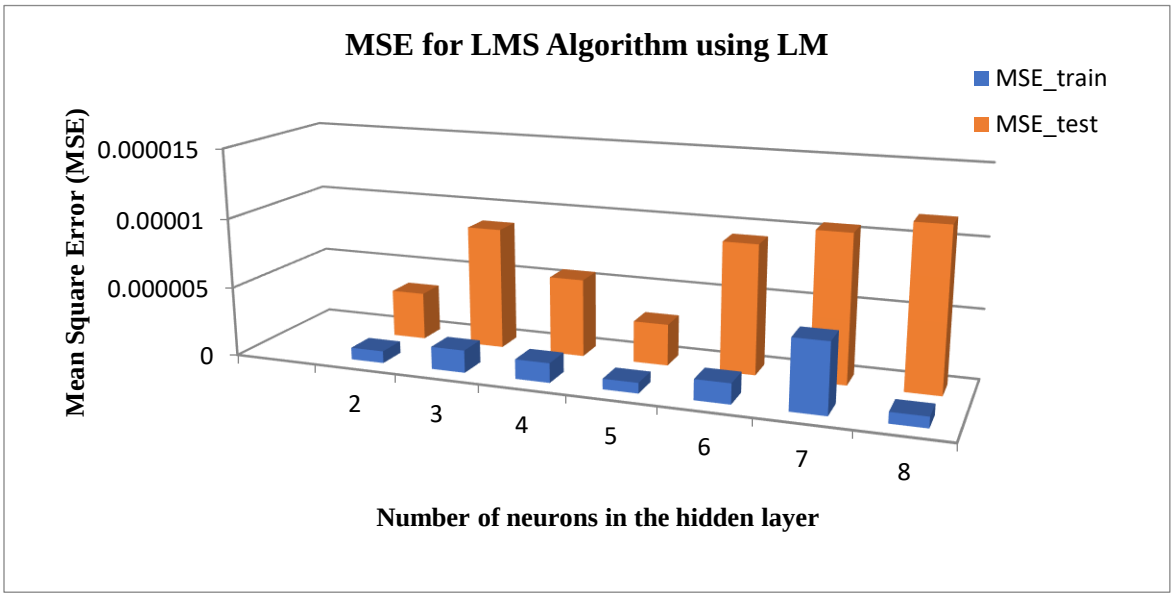
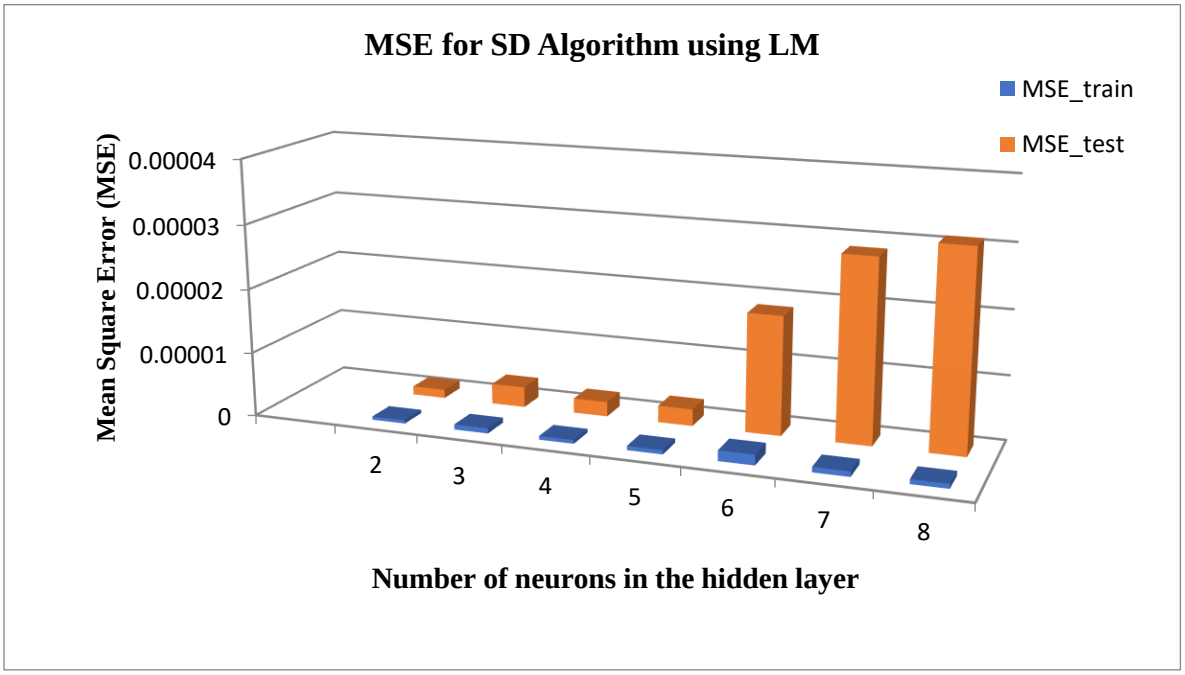


Figure 5.13: Training performance of GDM for all variants of LMS

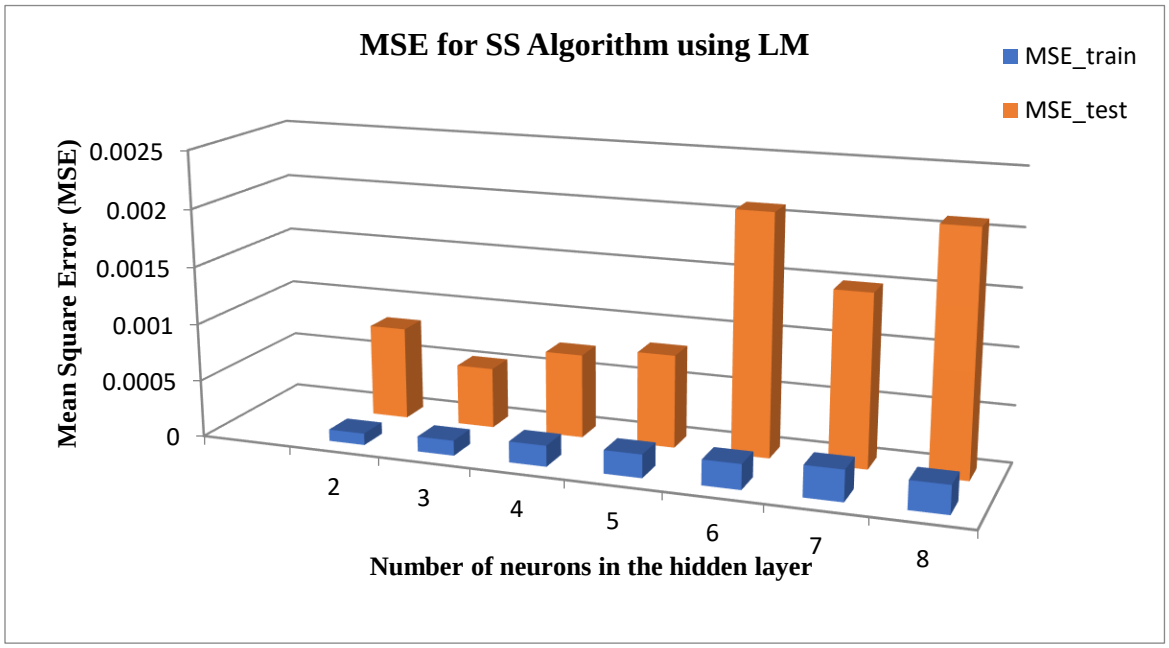
The testing performance comparison of both methodologies with respect to training on MSE grounds for LMS and its sign variants can be separately visualized in figures 5.14 and 5.15. It can be interpreted from the figures that the testing MSE of all architectures for both the methodologies and all variants follow the training MSE trend in 6 out of 8 cases.


MSE for LMS Algorithm using LM
MSE_train
MSE_test
Mean Square Error (MSE)
0.000015
0.00001
0.000005
0
2
3
4
5
6
7
8
Number of neurons in the hidden layer

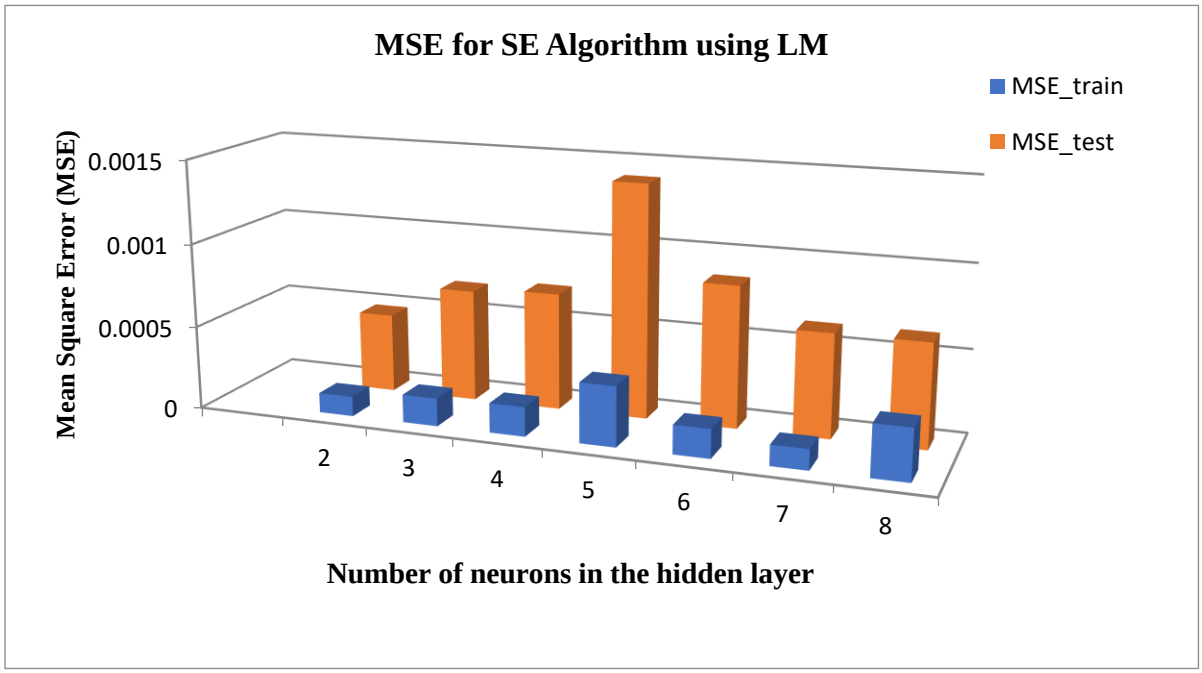
(a)


MSE for SD Algorithm using LM
MSE_train
MSE_test
Mean Square Error (MSE)
0.00004
0.00003
0.00002
0.00001
0
2
3
4
5
6
7
8
Number of neurons in the hidden layer

(b)

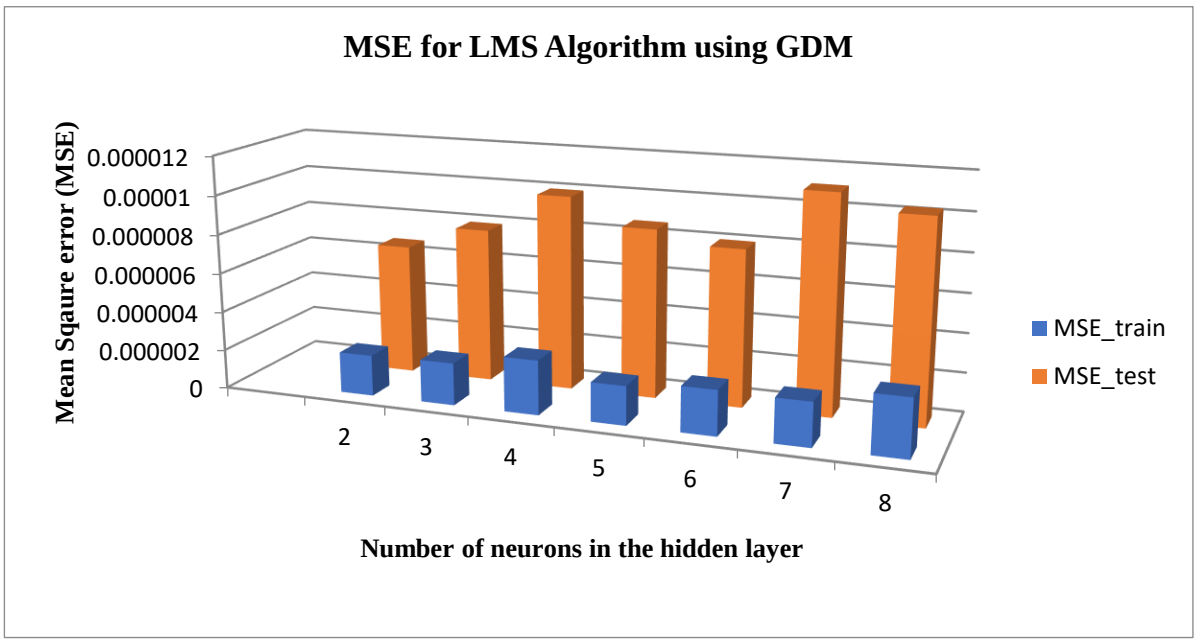


(c)

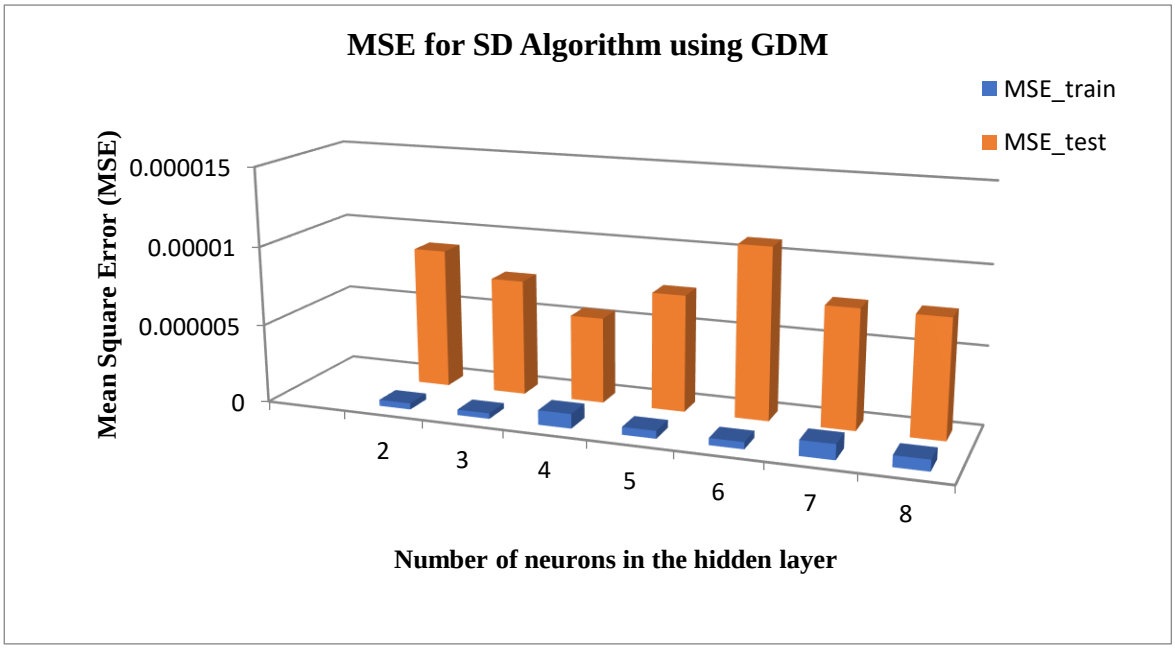


(d)

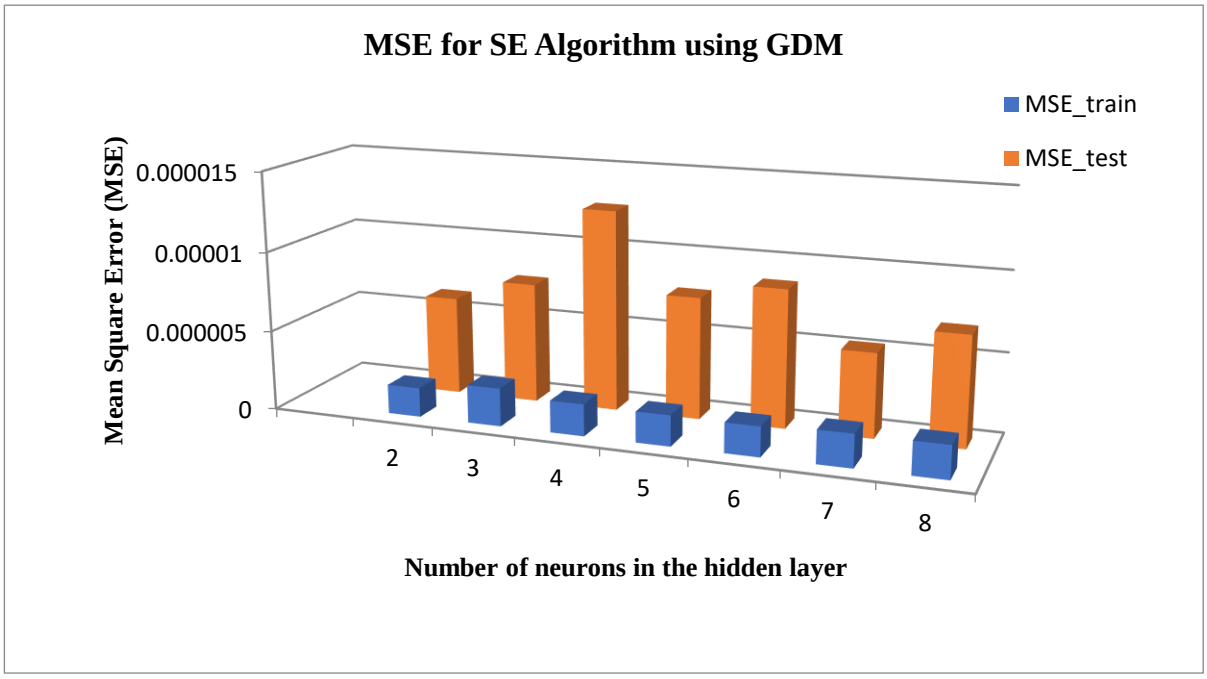
Figure 5.14: MSE for training and testing using LM for all variants of LMS (a) - (d)


MSE for LMS Algorithm using GDM
Mean Sqaure error (MSE)
0.000012
0.00001
0.000008
0.000006
0.000004
0.000002
0
2
3
4
5
6
7
8
Number of neurons in the hidden layer
MSE_train
MSE_test

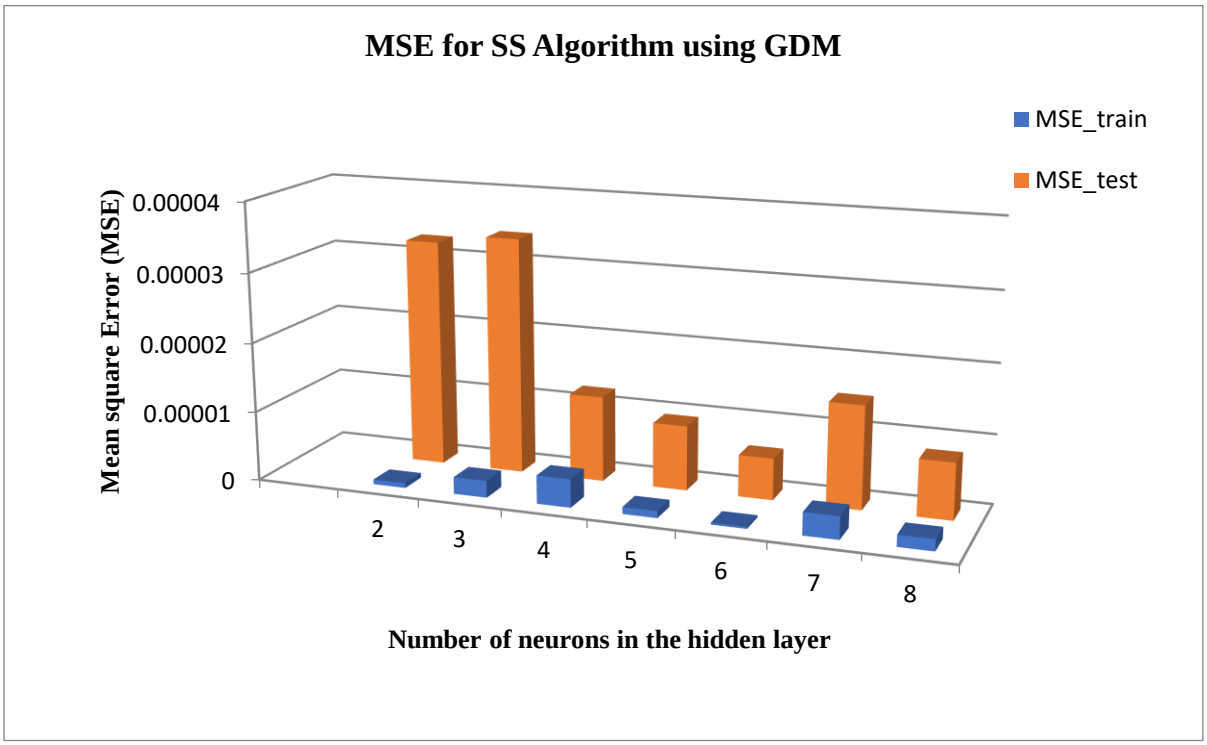
(a)


MSE for SD Algorithm using GDM
Mean Square Error (MSE)
0.000015
0.00001
0.000005
0
2
3
4
5
6
7
8
Number of neurons in the hidden layer
MSE_train
MSE_test

(b)



(c)



(d)

Figure 5.15: MSE for training and testing using GDM for all variants of LMS (a) - (d)

The testing output signals were recovered after applying 5-fold cross validation and the average of all the folds has been taken for analyzing the training and testing MSE as represented in figures 5.12 - 5.15. These signals were recovered by setting up a threshold of 0 and 1 for signal values less than 0.5 and more than 0.5 respectively when the target was preprocessed passing through a SP function. The testing performance was assessed by analyzing how many samples have been correctly recovered and is in coordination with the actual required signal. In order to further track, whether the reconstructed signals are also in phase with the actual required ones, the output was made to pass through an ISP function. The threshold values for these signals were taken as -0.5 if the signals are in range of (-0.5 - 0), -1 if they are in range (-1 - -0.5), 0 if in (0 - 0.5) and 1 for (0.5 - 1). The testing performance in terms of percentage success rate of LM and GDM methodologies can be envisaged from figures 5.16-5.19. Figure 5.16 and 5.17 represent the performance when the output is obtained after preprocessing the target. It has been observed from the bar graphs that LMS and SD algorithms performed best and equally well whereas SE and SS perform also performed equally well but worst among all variants of LMS for LM methodology. However, this is not the case with GDM methodology where all variants have approximately similar performance for all kinds of architectures. Figures 5.18 and 5.19 depict the performance of these methodologies when the recovered signals were made to pass through an ISP function. These graphs also indicate that LMS and SD in LM methodology and LMS and SE in GDM are the star performer algorithms whereas SS in both methodologies performs the worst. The testing and training performance of optimum architectures of both methodologies can also be analyzed by comparing the actual required signal and reconstructed signal graphically as depicted in figures 5.20 -5.27. The line graphs corresponding to testing performance of LMS and its sign variants for LM methodology clearly shows that LMS and SD algorithm outperforms the rest of the algorithms. However, the testing performance of all optimum architectures corresponding to LMS and its sign variants for GDM methodology is equally likely and is in phase with their training performance. Figure 5.20 and 5.21 represent the required and reconstructed signals for both training and test phases of LMS and SD algorithm for LM methodology. The plots are evidences of the effectiveness of the two algorithms for LM methodology. The poor test performance of SE and SS algorithms as evident from figure 5.22 and 5.23 can be explained on the basis of the poor training performance of the same algorithms as shown in the training phase of figures 5.22 and 5.23. As a consequence of

poor training, clearly the test phase MSE is also high in Figures 5.14(c) and (d) which correlates well to the required and reconstructed signals for all variants of LMS in LM methodology.

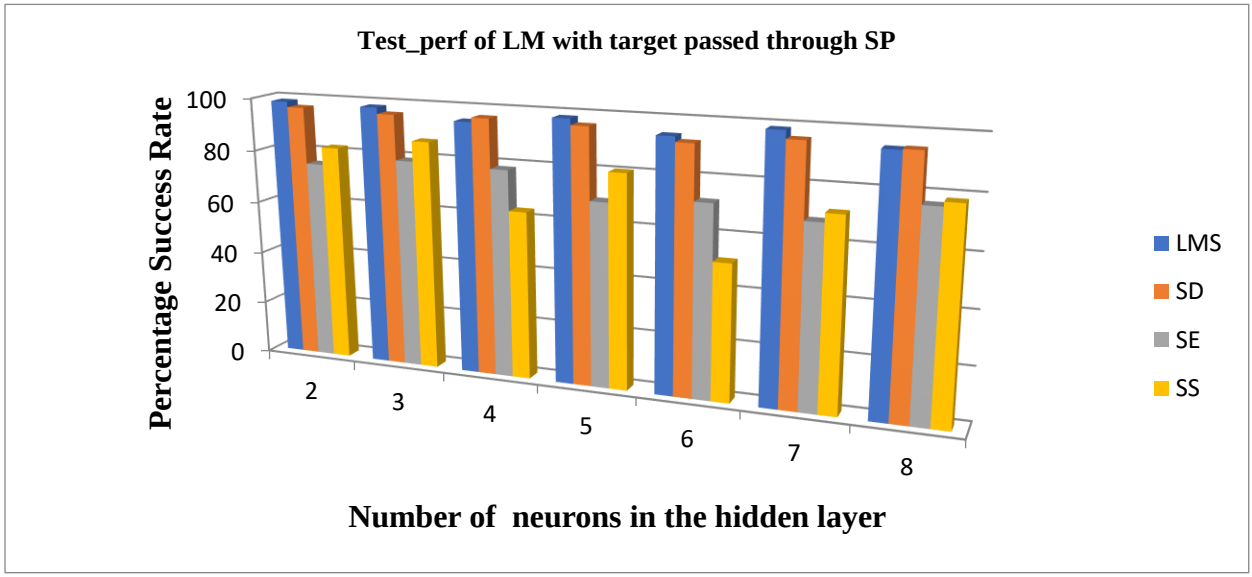


Figure 5.16: Testing performance of LM methodology for LMS and its sign variants with target preprocessed

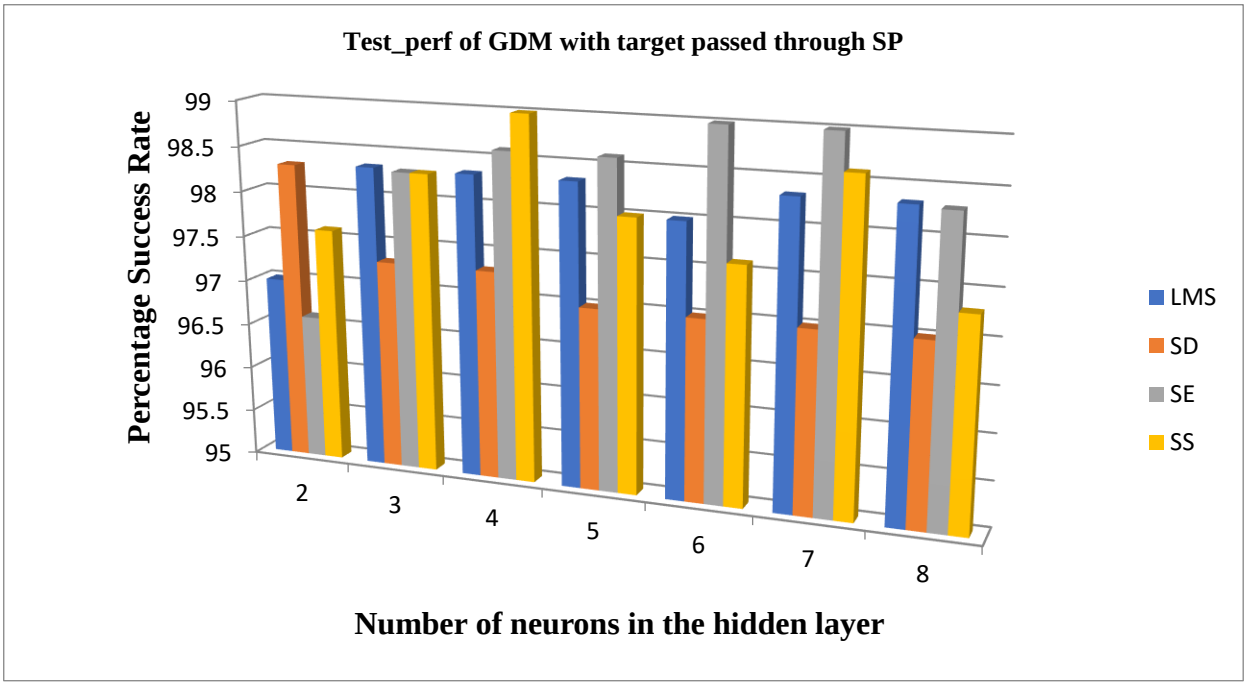


Figure 5.17: Testing performance of GDM methodology for LMS and its sign variants with target preprocessed

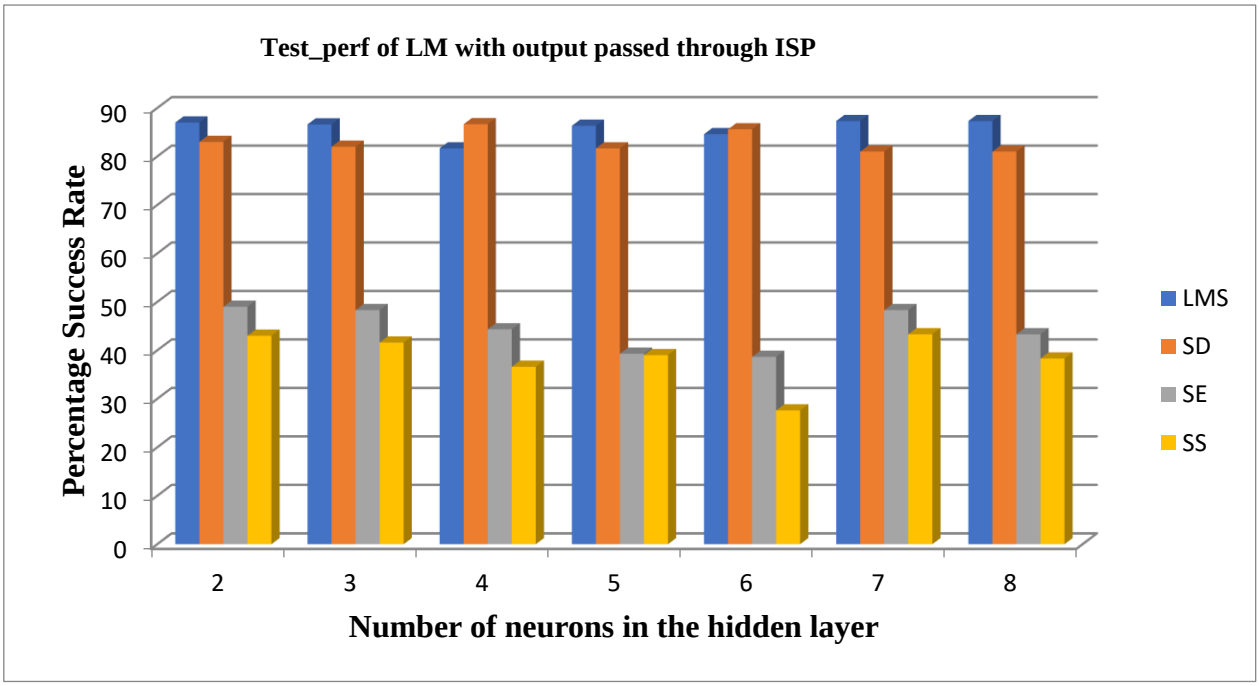


Figure 5.18: Testing performance of LM methodology for LMS and its sign variants with output recovered using ISP

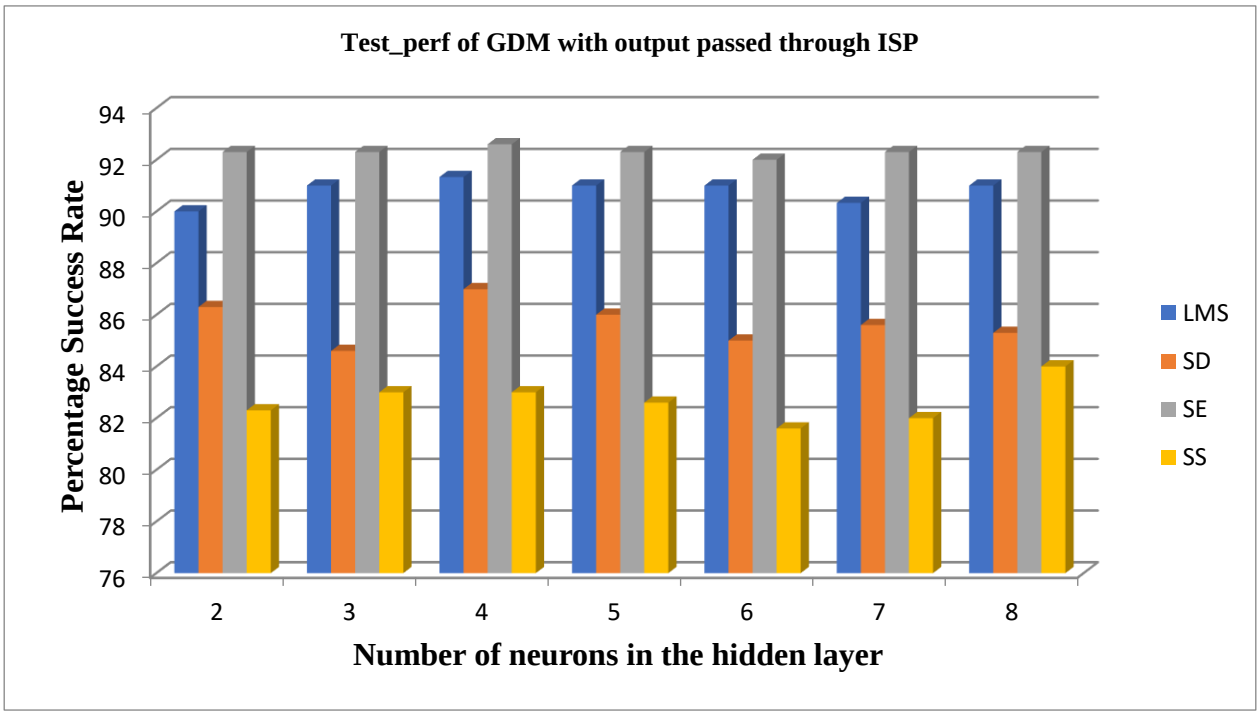
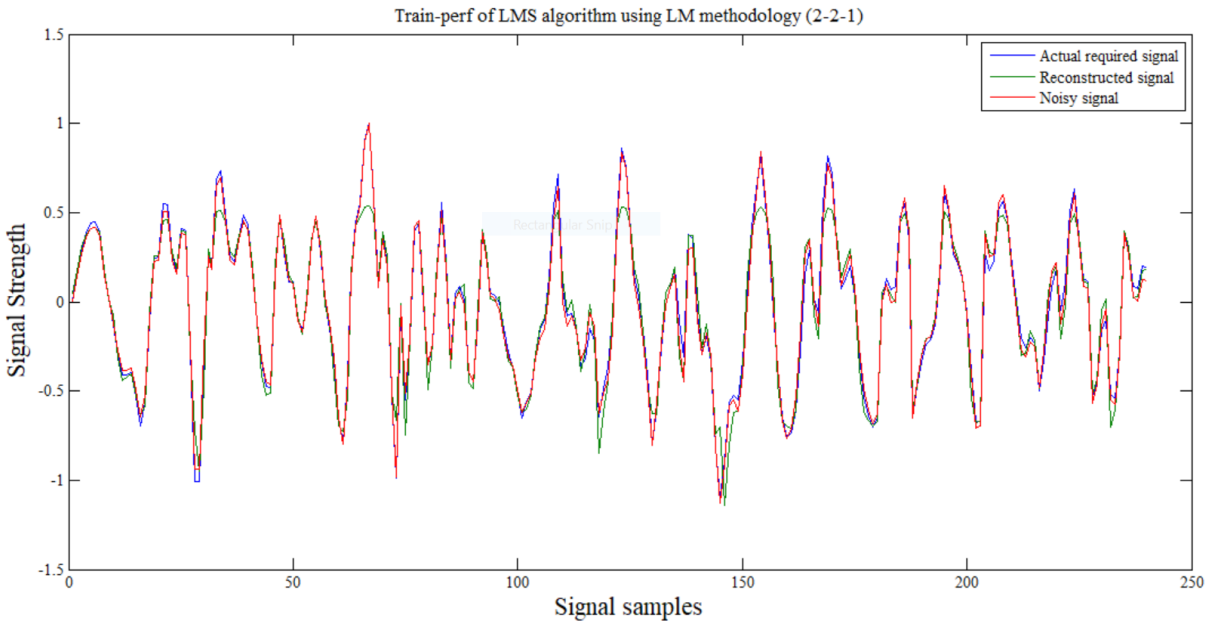
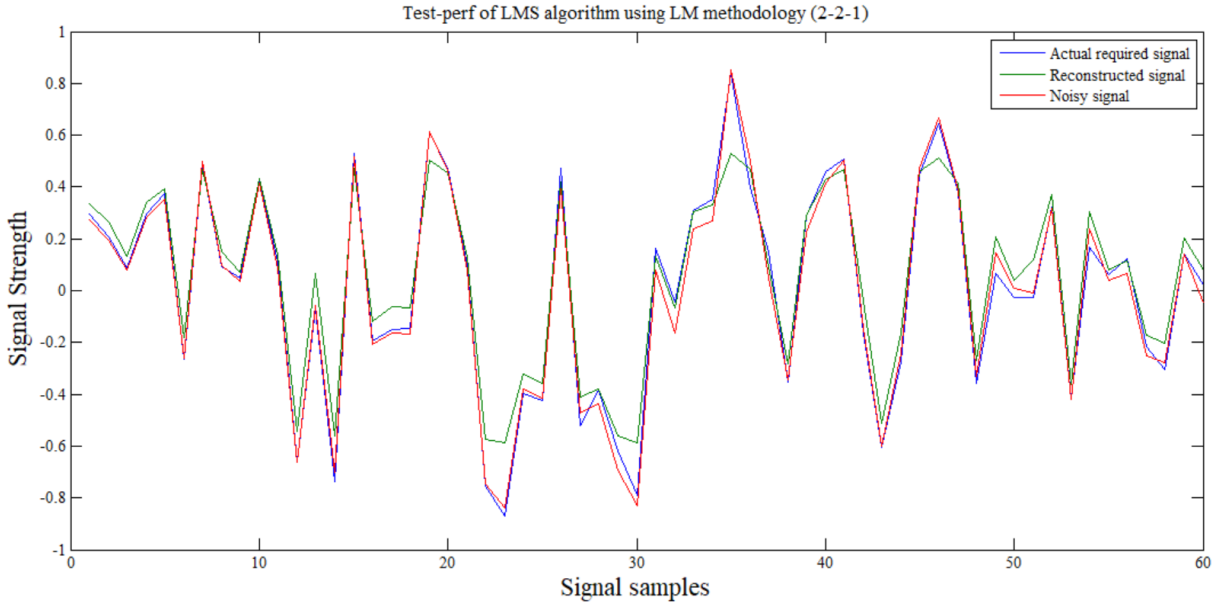


Figure 5.19: Testing performance of GDM methodology for LMS and its sign variants with output recovered using ISP

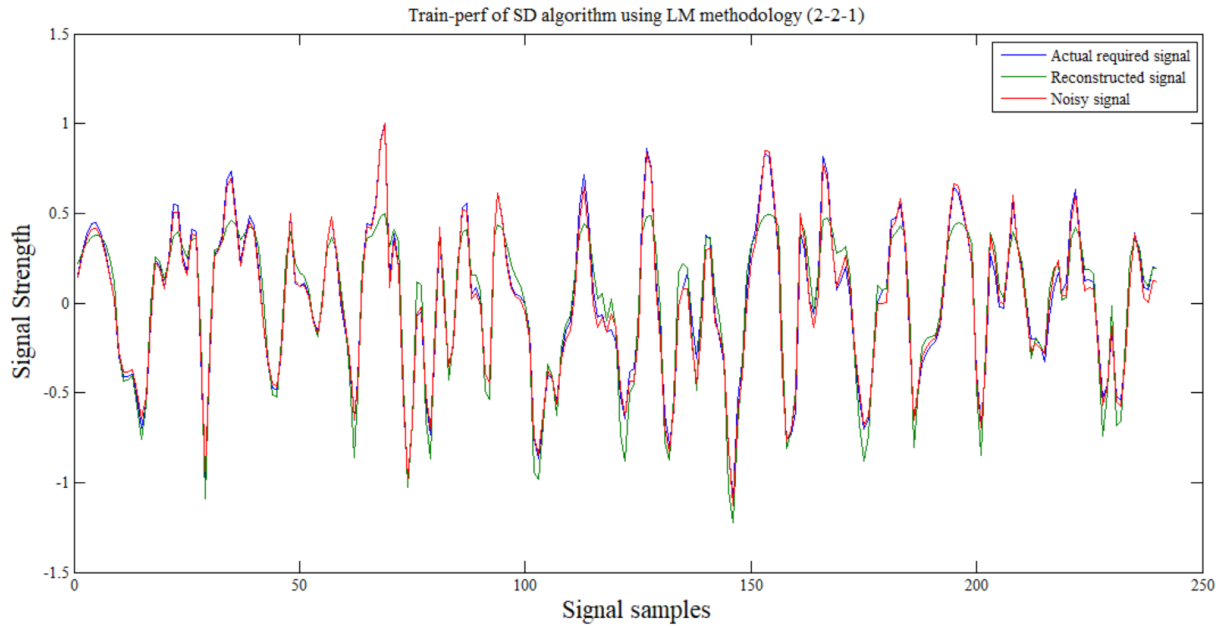


(a)

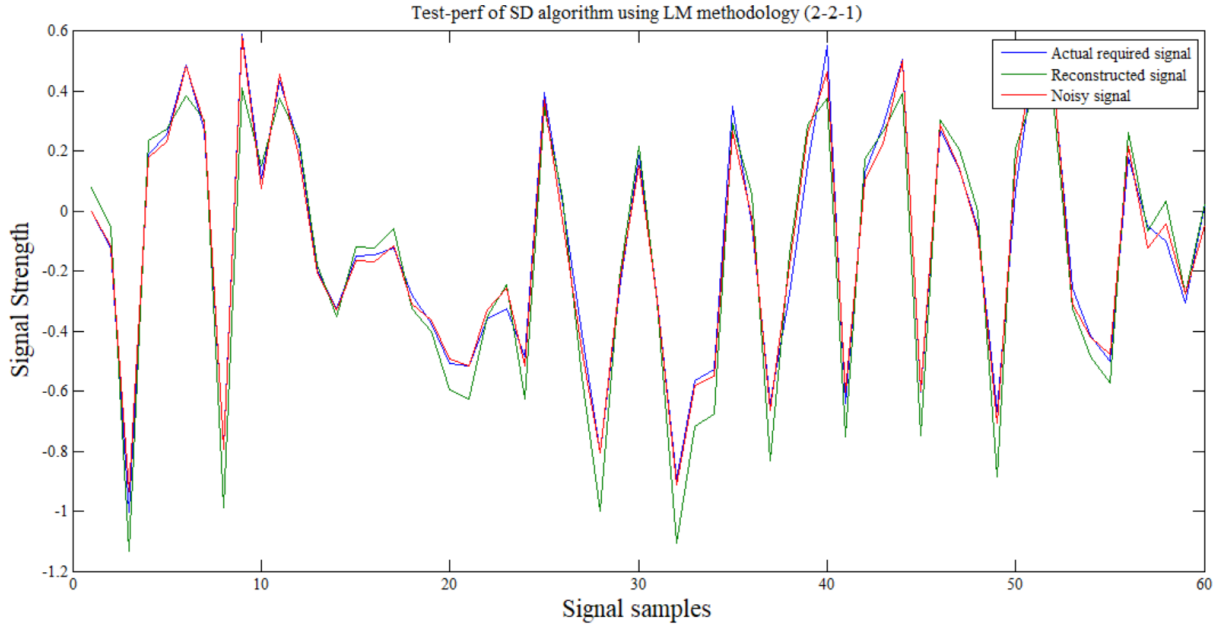


(b)

Figure 5.20: Training and testing performance of optimum architecture of LM methodology using LMS algorithm

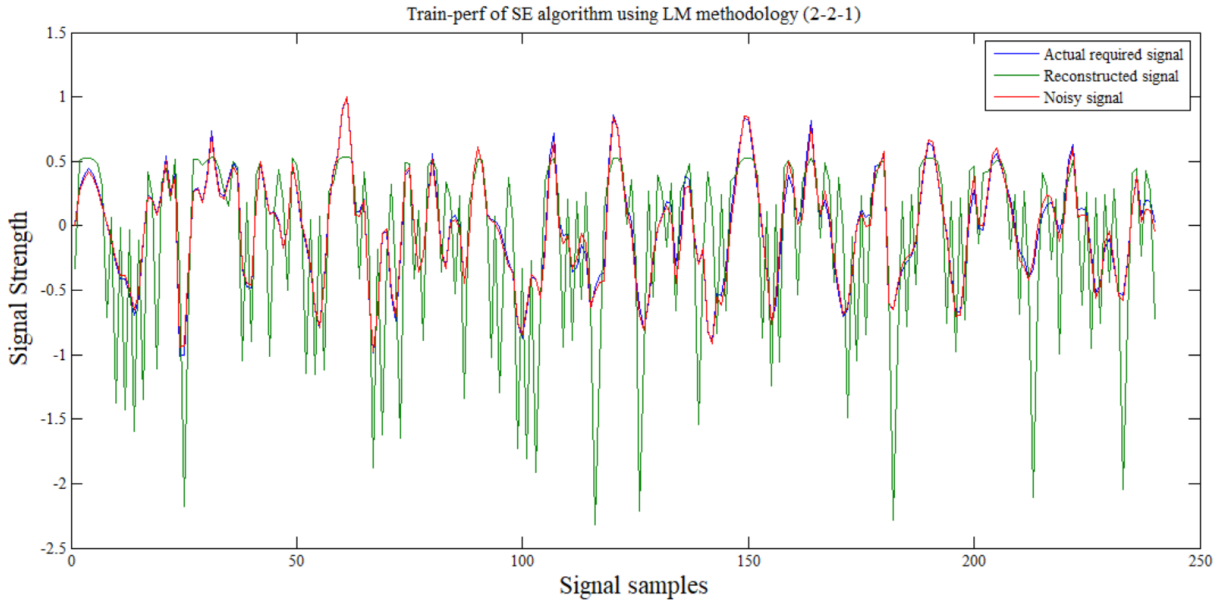


(a)

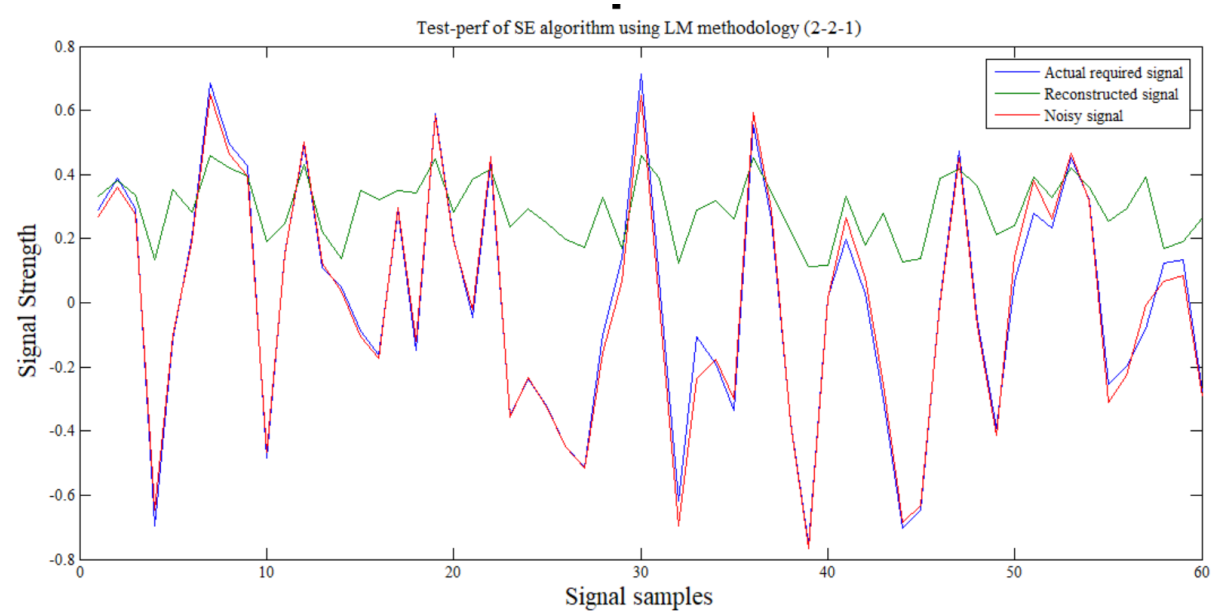


(b)

Figure 5.21: Training and testing performance of optimum architecture of LM methodology using SD algorithm

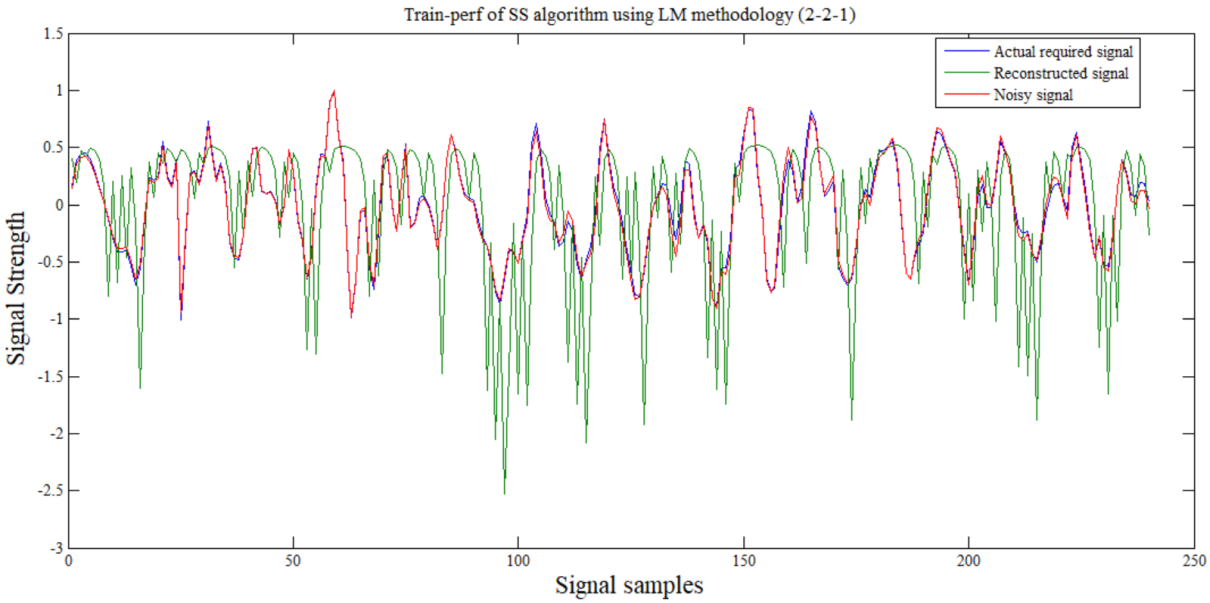


(a)

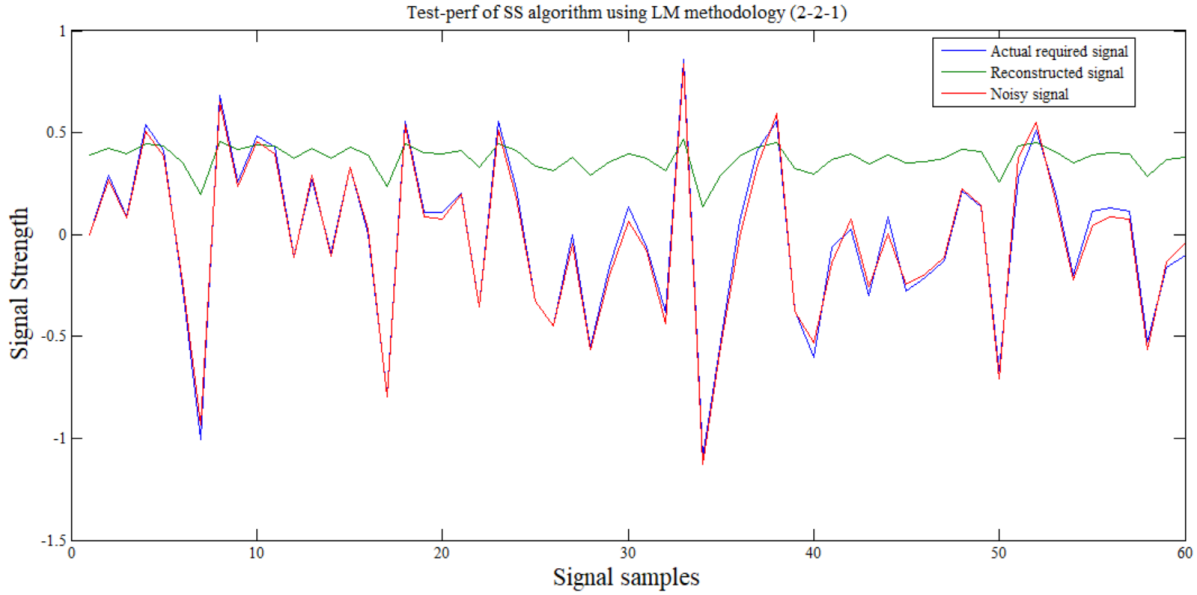


(b)

Figure 5.22: Training and testing performance of optimum architecture of LM methodology using SE algorithm

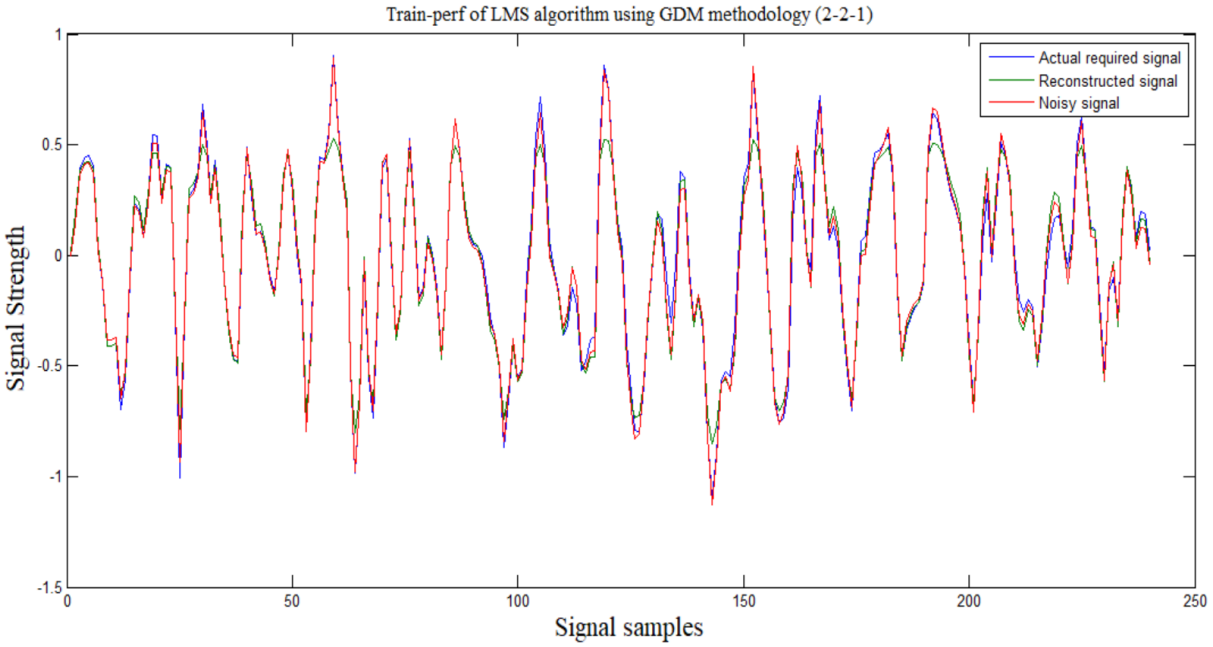


(a)

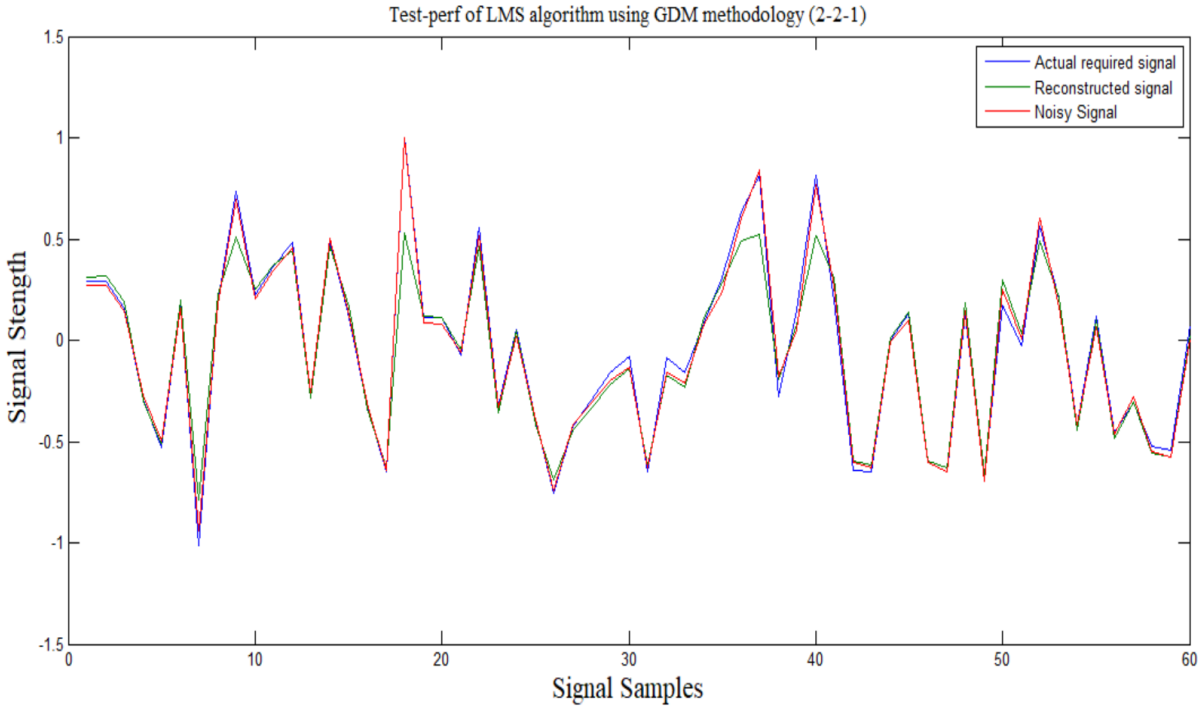


(b)

Figure 5.23: Training and testing performance of optimum architecture of LM methodology using SS algorithm

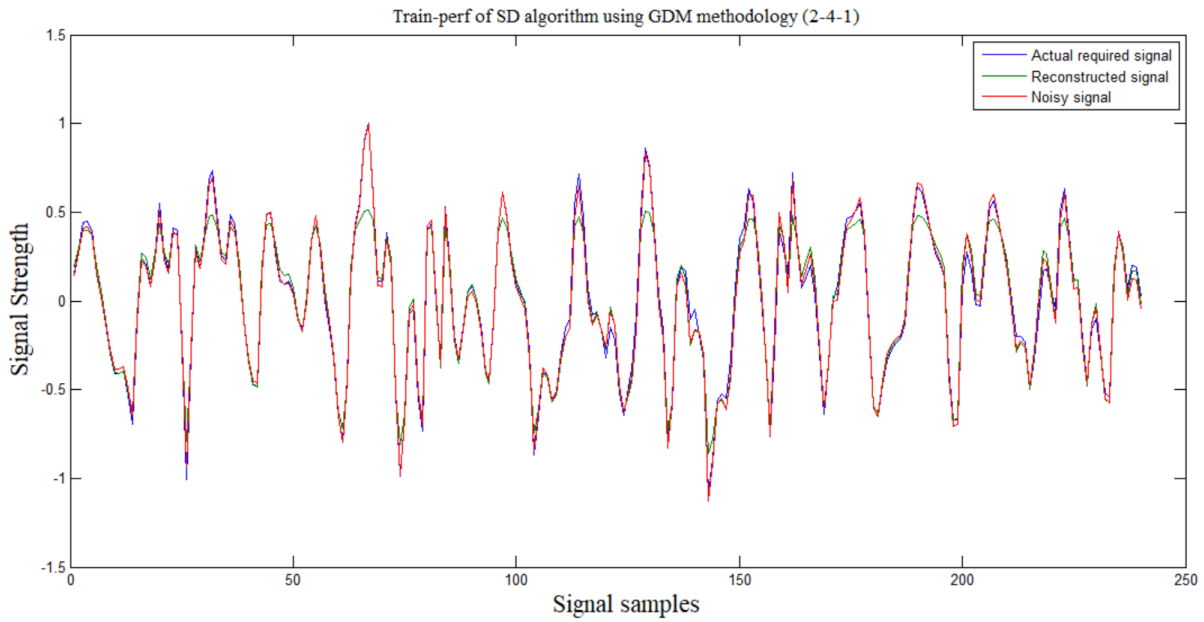


(a)

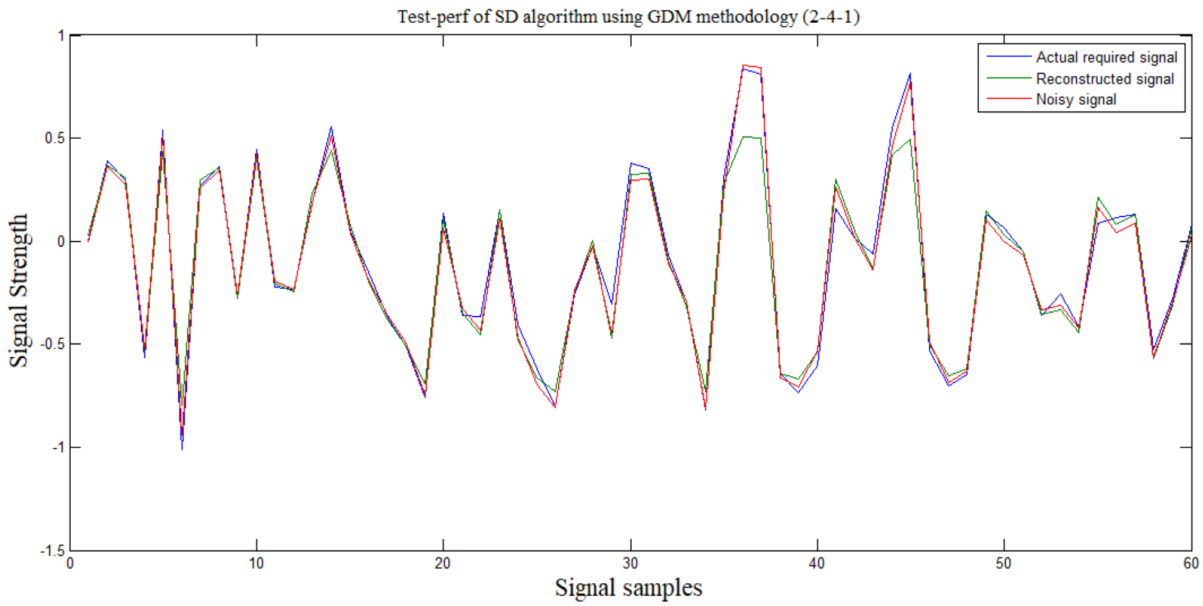


(b)

Figure 5.24: Training and testing performance of optimum architecture of GDM methodology using LMS algorithm

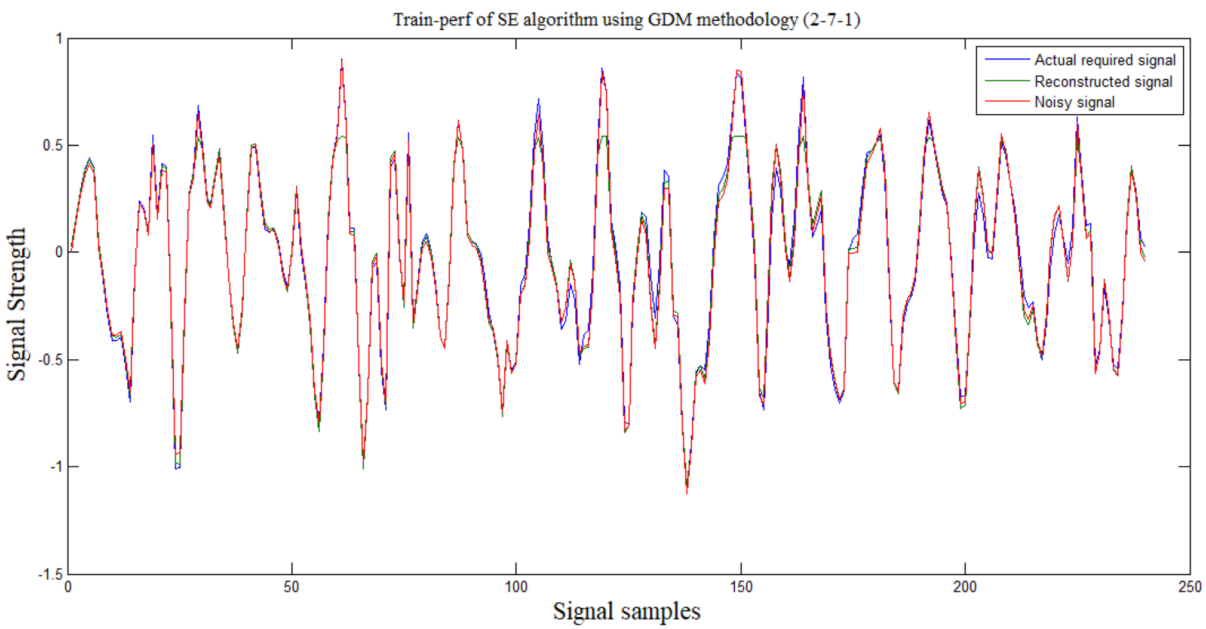


(a)

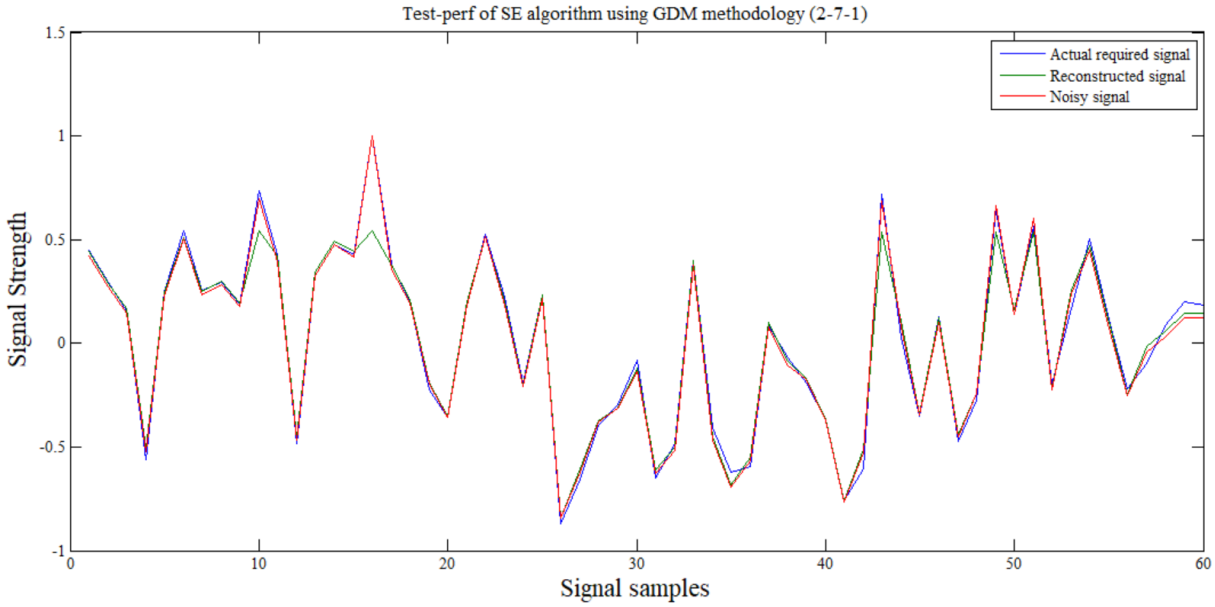


(b)

Figure 5.25: Training and testing performance of optimum architecture of GDM methodology using SD algorithm

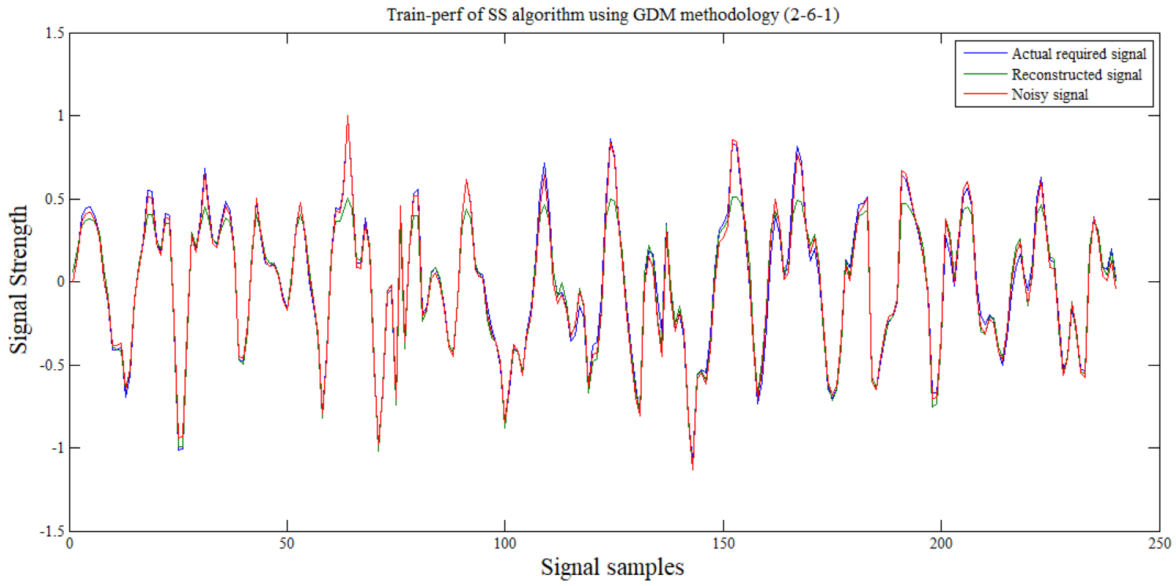


(a)

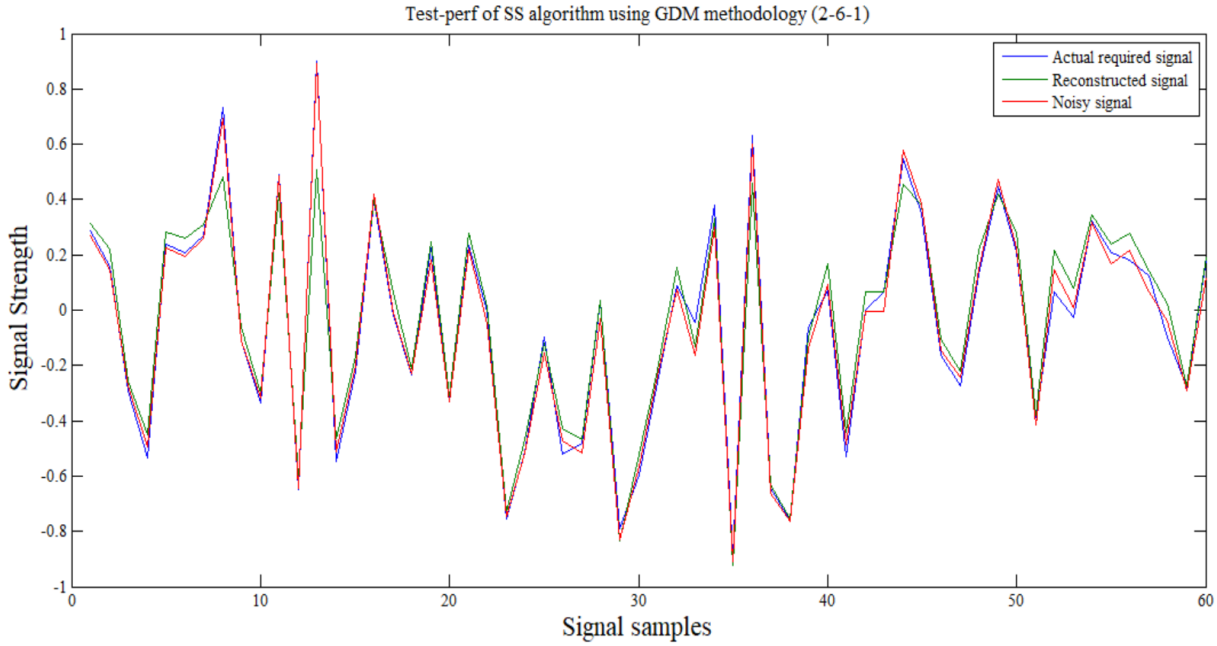


(b)

Figure 5.26: Training and testing performance of optimum architecture of GDM methodology using SE algorithm



(a)



(b)

Figure 5.27: Training and testing performance of optimum architecture of GDM methodology using SS algorithm

The filter designed using GDM methodology improved the performance of LMS and its sign variants especially SE and SS algorithm. The required and reconstructed signals of test phase shown in figures 5.24-5.27 directly correspond to the test MSE for GDM methodology and for

all variants of LMS algorithm. The testing performance is approximately similar for all variants of GDM methodology and is in phase with their corresponding training performance of all optimum architectures.

The testing performance for all sign variants of LMS using both LM and GDM methodology when the recovered signal is made to pass through an ISP function were further evaluated using paired sample t-test. This test has been done to assess the effectiveness of the variants of LMS algorithm for different NN methodologies and for variation in the hidden layer neurons. In the current study, the null hypothesis assumes that there is no significant improvement in the success rate of the specific variant of LMS using the given NN methodology relative to the other variant of LMS. Conversely, the alternate hypothesis assumes a significant improvement in the success rate using that variant of LMS and using the given NN methodology. In our case, 0.05 level of significance (α) was considered. If the P-value is less than (or equal to) α, then the null hypothesis is rejected in favor of the alternative hypothesis. And, if the P-value is greater than α, then the null hypothesis is not rejected. Data was taken as the success rate of all the sign variants of LMS using both LM and GDM methodology when the output signal is recovered using the ISP function for the number of hidden neurons varying from 2 to 8. The results of the t-test are summarized in Table 5.1 where null hypothesis is rejected for all the combinations which clearly justifies that LMS and SD in LM methodology and LMS and SE in GDM are the star performer algorithms for all variations of hidden neurons.

NN Methodology	**Algorithm Comparison**	**P-value (α=0.05)**	**Null Hypothesis**
LM	LMS and SE	1.8555E-07	Rejected
	LMS and SS	4.11122E-08	Rejected
	SD and SE	9.5032E-07	Rejected
	SD and SS	7.6655E-07	Rejected
GDM	LMS and SD	8.09803E-06	Rejected
	LMS and SS	5.97464E-08	Rejected

	SE and SD	2.9364E-07	Rejected
	SE and SS	1.15745E-08	Rejected

Table 5.1: T-test results

5.5 Results and discussion of ANC implemented using different structures

This work makes use of Simulink HDL coder which offers several important benefits like quick assessment of hardware implementability of the algorithm which ultimately helps to choose the best one. This approach works as a better alternative for FPGA prototyping than the VHDL description which is a complex and time-consuming task.

After the successful completion of HDL code generation step, a resource utilization report was generated that describes the requirement of adders, comparators, multiplexers etc., required to implement that design in hardware. The resource utilization report of the implemented ANC is shown in Table 5.1. The generated code was then synthesized targeting specific FPGA using ISE 14.5 and the synthesis results representing hardware utilized, timing analysis and power analysis are shown in table 5.2, 5.3 and 5.4 respectively. For performance comparison, three different Xilinx devices have been targeted namely, a low cost Spartan6 and two leading edge devices from

Virtex6 and Virtex7 device families. The present work seeks to evaluate the performance of the adaptive filter with respect to the algorithm chosen, the filter structure, and the implementation technology. In all the three cases, clock speed, delay, hardware utilization and power have been the common performance matrices.

5.5.1 Performance analysis in terms of algorithm chosen

It is evident from Tables 5.2-5.5 that the performance of LMS algorithm in terms of speed is best among all which however is achieved at the cost of maximum number of LUTs and slices consumed on an FPGA. Logic occupation is very similar and maximum for LMS and SE except for the usage of multiplexers and comparators units which is maximum for SD and SS algorithm. Apart from that, Post synthesis results also show that ANC implemented using the LMS consumes more power which again is due to the usage of more number of on-chip resources. In addition to this, the performance of SE algorithm in terms of speed and power is at par with the LMS whereas SD and SS do not perform well.

Table 5.2: Resource utilization of 40-tap ANC

Algorithm	**Structure**	**Adder**				**Mux's**		**Registers**		**Comparators**	
		16 bit	32 bit	33 bit	>34 bit	16 bit 2:1	32 bit 2:1	16 bit	>32 bit	>16 bit	<16 bit
LMS	Direct	1	161	39	0	124	158	83	0	0	0
	Transposed	1	240	0	0	200	80	120	1	0	0
	Proposed	1	161	20	19	84	82	3	2	0	0
SD	Direct	1	81	39	0	125	238	83	0	2	0
	Transposed	1	160	0	0	201	160	120	1	2	0
	Proposed	1	81	20	19	85	162	3	2	2	0
SE	Direct	1	160	39	0	125	160	83	0	2	0
	Transposed	1	239	0	0	201	82	120	1	2	0
	Proposed	1	160	20	19	85	84	3	2	2	0

SS	Direct	1	80	39	0	126	240	83	0	4	0
	Transposed	1	159	0	0	202	162	120	1	4	0
	Proposed	1	80	20	19	86	164	3	2	4	0

5.5.2 Performance analysis in terms of filter structure

It is well known that popular filter structures yield better performance according to one metric or the other. Thus, the choice of an appropriate structure might give the required performance in terms of a parameter of interest. It is evident from Table 5.2 and 5.3 that area consumption in terms of use of resources (adders, comparators etc.) or LUT consumption on an FPGA is least for the proposed technique. Inspection of Table 5.4 and Table 5.5 reveal that among the above-mentioned structures the direct form have resulted the least power consumption but at the cost of lower clock speed making it suitable for low power applications. Furthermore, due to reduction in critical path, the clock speed of the transposed structure is significantly increased making it suitable for high speed applications. However, the increase in the clock speed is at the expense of increased power dissipation. On the other hand, the performance of proposed structure is a compromise between the two. The authors believe that strategy proposed in Figure 5.4 combines the advantages of other two implemented structures thus striking a balance between different requirements of an optimum adaptive filter.

Table 5.3: Device utilization summary

Algorithm	**Structure**	**Used Slice Registers**			**Used Slice LUTs**			**Occupied Slices**		
		Spartan 6	Virtex 6	Virtex 7	Spartan 6	Virtex 6	Virtex 7	Spartan 6	Virtex 6	Virtex 7
LMS	Direct	1312 (1%)	1342 (1%)	1342 (1%)	8450 (9%)	8326 (5%)	8415 (3%)	2851 (12%)	2916 (7%)	2829 (4%)
	Transposed	1957 (1%)	1986 (1%)	1986 (1%)	8490 (9%)	8290 (5%)	8381 (3%)	2920 (12%)	2919 (7%)	2886 (4%)
	Proposed	1942 (1%)	1942 (1%)	1942 (1%)	8112 (8%)	7930 (5%)	8016 (3%)	2720 (11%)	2713 (6%)	2716 (4%)
SD	Direct	764 (1%)	752 (1%)	742 (1%)	7289 (7%)	7131 (4%)	7183 (2%)	2677 (11%)	2902 (7%)	2747 (4%)

SE	Transposed	1434 (1%)	1434 (1%)	1434 (1%)	7669 (8%)	7547 (4%)	7642 (2%)	2812 (12%)	2981 (7%)	3180 (4%)
	Proposed	1410 (1%)	1382 (1%)	1382 (1%)	6849 (7%)	6703 (4%)	6754 (2%)	2444 (10%)	2671 (6%)	2338 (3%)
	Direct	1328 (1%)	1328 (1%)	1328 (1%)	8545 (9%)	8336 (5%)	8424 (3%)	3075 (13%)	2850 (7%)	2844 (4%)
	Transposed	1986 (1%)	1986 (1%)	1986 (1%)	8556 (9%)	8293 (5%)	8378 (3%)	3014 (13%)	2868 (7%)	2876 (4%)
	Proposed	1928 (1%)	1928 (1%)	1928 (1%)	8139 (8%)	7918 (5%)	8019 (3%)	2762 (11%)	2610 (6%)	2720 (4%)
SS	Direct	740 (1%)	738 (1%)	738 (1%)	7323 (7%)	7132 (4%)	7184 (2%)	2682 (11%)	2443 (6%)	2724 (4%)
	Transposed	1434 (1%)	1434 (1%)	1434 (1%)	7698 (8%)	7550 (4%)	7643 (2%)	2849 (12%)	2848 (7%)	3224 (5%)
	Proposed	1397 (1%)	1368 (1%)	1368 (1%)	6871 (7%)	6703 (4%)	6746 (2%)	2471 (10%)	2500 (6%)	2340 (2%)

For example, the simultaneous generation of delayed vector output does away with the need of multiple delays as in the case of other structures mentioned here. This results in enhanced clock speed as compared to direct form structure, reduced power consumption as compared to transpose structure, and least silicon area consumption among all the structures. It was also noticed from the results that unlike the direct form and transpose form structures, the proposed structure uses higher bit adders such as 37 bit and 38 bit that will help in faster addition. In addition to this, logic resources such as multiplexers and registers in the proposed form are reduced by a significant amount as compared to other.

Table 5.4: Timing analysis summary

Algorithm	**Structure**	**Clk. Freq. (Max.) (MHz)**			**Delay(min.) (nsec)**		
		Spartan 6	Virtex 6	Virtex 7	Spartan 6	Virtex 6	Virtex 7
LMS	Direct	8.334	13.798	14.245	119.996	72.475	70.198
	Transposed	31.027	51.431	55.943	32.230	19.450	17.875
	Proposed	24.939	40.265	42.617	40.097	24.835	23.465
SD	Direct	8.252	13.766	14.236	121.176	72.641	70.247
	Transposed	28.664	49.029	53.596	34.887	20.396	18.658

	Proposed	23.971	39.994	42.760	41.718	25.004	23.368
SE	Direct	8.198	13.687	14.163	121.988	73.060	70.607
	Transposed	29.466	49.966	54.663	33.938	20.014	18.294
	Proposed	23.933	39.395	41.892	41.783	24.384	23.871
SS	Direct	8.101	13.618	14.110	123.448	73.434	70.870
	Transposed	27.326	47.712	54.420	36.595	20.959	19.077
	Proposed	23.009	39.112	42.008	43.461	25.568	23.805

5.5.3 Performance analysis in terms of device family

The proposed structure along with the other two was implemented on three different device families viz. Spartan 6, Virtex 6, and Virtex 7. All the three mentioned device families have something to offer either in terms of low power and low cost of implementation or in terms of increased system performance and accelerated design productivity. This has motivated the authors to investigate speed, power and device utilization parameters of the filters implemented on individual device families. Being the latest among the three, Virtex 7 combines very high level of system integration (28 million logic cells) with an unprecedented performance bandwidth (up to 2.8 Tb/sec of total serial bandwidth). The synthesis results shown in Tables 5.3-5.5 also verified that Virtex 7 implementation gives a higher clock speed, with a marginal increase in the area and power consumption than the other two device families.

Table 5.5: Power analysis summary

Algorithm	**Structure**	**Static Power (W)**			**Dynamic Power (W)**			**Total Power (W)**		
		Spartan 6	Virtex 6	Virtex 7	Spartan 6	Virtex 6	Virtex 7	Spartan 6	Virtex 6	Virtex 7
LMS	Direct	0.114	0.319	0.247	0.025	0.036	0.025	0.139	0.355	0.272
	Transposed	0.116	0.322	0.248	0.094	0.140	0.099	0.210	0.462	0.347
	Proposed	0.115	0.321	0.249	0.073	0.106	0.073	0.188	0.427	0.322
SD	Direct	0.114	0.318	0.247	0.024	0.036	0.024	0.138	0.354	0.271
	Transposed	0.116	0.322	0.248	0.084	0.129	0.092	0.200	0.451	0.340

	Proposed	0.115	0.321	0.249	0.067	0.101	0.069	0.182	0.422	0.318
SE	Direct	0.114	0.319	0.247	0.024	0.036	0.025	0.138	0.355	0.272
	Transposed	0.116	0.322	0.248	0.089	0.136	0.097	0.205	0.458	0.345
	Proposed	0.115	0.321	0.249	0.070	0.103	0.072	0.185	0.424	0.321
SS	Direct	0.114	0.318	0.247	0.023	0.035	0.024	0.137	0.353	0.271
	Transposed	0.116	0.322	0.248	0.080	0.126	0.093	0.196	0.448	0.341
	Proposed	0.115	0.321	0.249	0.065	0.099	0.068	0.180	0.420	0.317

5.5.4 Hardware estimation for NN adaptive filter

The computational complexity as described in section 3.5.3 is of paramount importance for any algorithm. The hardware estimate of adaptive filters designed using direct, transpose and proposed form of structures have been generated by Matlab HDL coder and is given in Table 5.2. In order to assess the efficacy of the NN approach for the design of adaptive filters, it is obligatory to quantify the requirement of hardware resources for the same. For this analysis, the researcher has represented the data in 12-bit fixed point fractional number format with 4 bits as the integer part and remaining 8 bits as the fixed fractional part. This very representation has been opted from [297] where the authors have proposed an efficient way of implementing sigmoid activation function using DA. The hardware estimate of optimum architectures for LM and GDM methodology have been computed in terms of operations required in the execution of one pass (forward and backward pass) of the algorithm. The mathematical operations required to represent the complete process are defined in terms of addition/ subtraction, multiplication, division and sigmoid activation function. The same operations are defined in terms of their gate count considering the method described in [297] to compute the gate count of multiplication and sigmoid activation function. The comparative analysis of the total count demanded by the two methodologies (LMS and GDM) with the count generated from table 5.2 for three structures (direct, transpose and proposed) have been shown in figure 5.28.

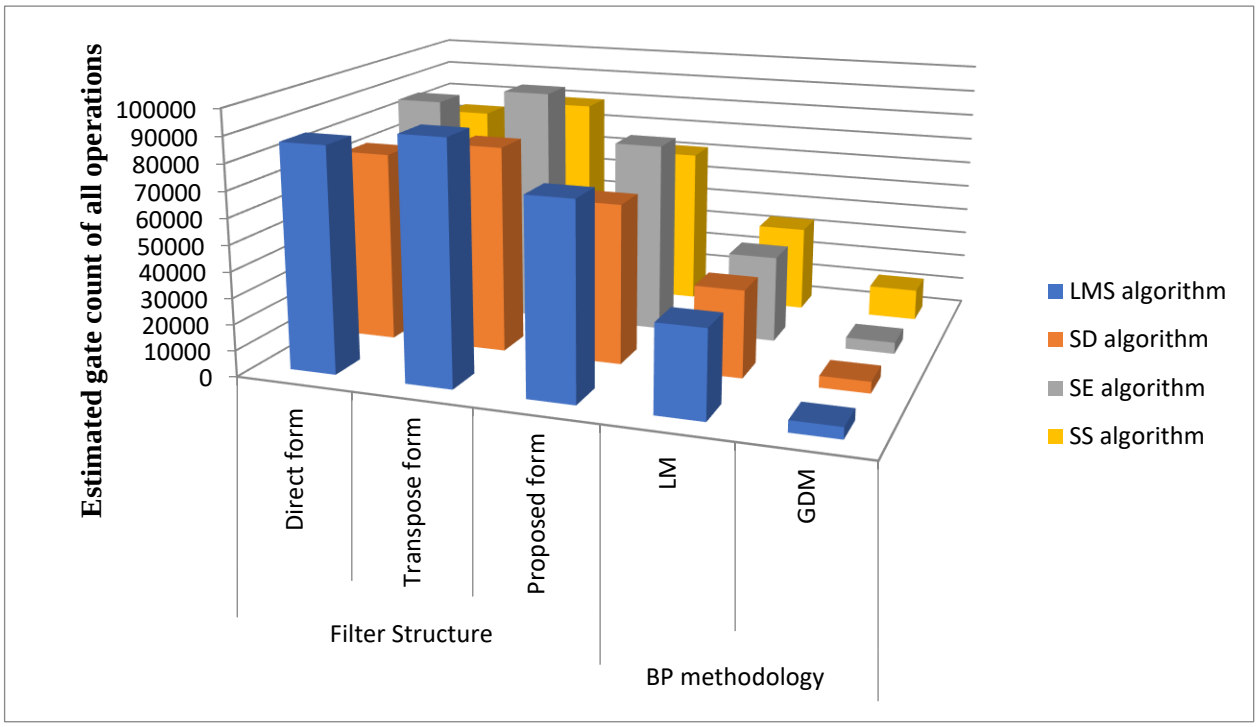


Figure 5.28: Comparison of estimated gate count of adaptive filters implemented using LM and GDM methodology with implementation using three structures

Figure 5.28 clearly justify the efficacy of NN implementation of adaptive filter with GDM methodology outperforming all the techniques. Furthermore, LM methodology also performs well relative to the conventional approach of adaptive filter implementation. However, the increased usage of resources in LM relative to GDM methodology owes itself to the calculation of Jacobian and its inverse.

5.6 Summary

This chapter establishes the performance of an ANN working as an adaptive filter and compares its performance with that of another adaptive filter implemented in a conventional manner. The filter itself was employed with LMS and its different variants in an attempt to evaluate the variation in the performance of the ANN based filter with respect to the conventional one. The ANN based filter was trained with two popular methodologies of the BP algorithm. In LM methodology, the analysis of testing performance of optimum and other architectures reveals the fact that an architecture with minimal configuration like 2-2-1 is able to perform well for a particular de-

noising task for all variants of LMS algorithm resulting in reduced computational time and resources thereby underpinning a possible minimalist hardware implementation of the design. However, using the GDM methodology the testing performance of the filter was more or less invariant to the architecture. All architectures with all variants of LMS algorithm perform equally well and hence again a minimally configured architecture can be chosen for its implementation on any physical environment.

In the later section of this chapter, a comprehensive study related to the performance parameters of hardware implementable variable step size adaptive filters has been done. The use of simulink HDL coder made the whole process quite simpler and less time consuming as a fast design entry can be done without the need of extensive hardware implementation knowledge. Also, the hardware implementation of an ANC using the HDL coder is an easier design approach as an inexperienced hardware designer would do. A novel filter structure for the generation of the vectorized delay has been proposed and implemented targeting different FPGAs. In a nut shell, following conclusions can be drawn from the synthesis results presented above.

(1) The proposed structure gives an optimum tradeoff in terms of used silicon area irrespective of all the four variants of LMS algorithm employed. This reduction in area owes itself to reduction in the number of multiplexers and registers which in turns is due to the generation of vectorized delay.
(2) The proposed structure performed better than the direct form structure in terms of speed where a three-fold increase in the speed has been witnessed. However, when compared to the transformed form structure (one with the minimum delay) the reduction in speed was not more than 15%.
(3) Despite a three-fold increase in the speed compared to the direct form structure, the proposed structure exhibits a significantly lower penalty in terms of power which makes it a good alternative for more efficient ASIC and FPGA implementations in the future.
(4) The synthesis results for the implementation of ANC on three device families reveal that Virtex 7 implementation gives a higher clock speed with a moderate increase in silicon area and power than other FPGAs.

With the hardware implementation results at our disposal, we were able to find that it is even more area efficient to implement the filter using an ANN rather than its implementation using conventional approach. Both the NN approaches perform exceptionally well in comparison to the rest with GDM outperforming among all. The increased hardware complexity of LM in comparison to GDM is due to the calculation of Jacobian and its inverse which itself are hard to compute. This problem is something which could be worked upon in future to enable more efficient hardware of ANN filters using LM methodology which is proven to converge better and is definitely faster.

The next chapter gives the concluding remarks by highlighting the contributions of the work done and future directions.

Chapter 6

Conclusion and Future Scope

This thesis is an attempt to improve the performance of a hardware implementable ANN and to investigate appropriate techniques which enable a simple ANN to perform optimally in its versatile avatars, not only as a classifier but also as a filter and function approximator. Novel techniques were developed and tested with an aim to reduce computational complexity and hardware requirement while at the same time obtaining an overall better generalization performance. The author has made use of the simplest possible ANN architecture and one of the most popular training algorithms i.e. BP (except chapter 4 where efficient learning in semi-supervised mode is attempted) throughout the work being fully aware about the requirements and constraints of subsequent implementation on hardware for real time deployment.

This work kicks-off by making the learning rate adaptive for better training and generalization performance of the network. In this part, experimental investigations and analysis have established a correlation between the information content in the output feature space of the ANN (inside a particular band of iterations of the training algorithm) and the requirement of an optimal learning rate in that particular band of iterations. It can be concluded from the obtained results that the

variance of the output feature space can be made use of for tuning learning rate adaptively inside different iteration bands. This work seems to open up further avenues for future work in terms of establishing a direct mapping between the learning rate, principal components of the output feature and error trajectory analytically.

The next part of the work as presented in chapter three was able to establish the relevance of weights in the ANN in a statistical sense thereby quantifying the information content of the weights convergent at each individual node in the network. The obtained results and analysis lead us to conclude that that pruning of non-relevant weights during the training phase is a viable option and saves a lot of time otherwise consumed in arriving at an optimal architecture through trial and error. It can also be concluded that labelling individual nodes with a statistically defined coefficient of dominance helps in easy identification of the nodes carrying higher information and corresponding complexity penalty associated with an individual weight. The network pruned using the methodology proposed in chapter three exhibited better convergence and generalization performance than the one pruned using other well-known methods. The results obtained using the proposed methodology can be scaled up in future to be implemented for pruning in deep networks.

One of the parts of this work as presented in chapter four is devoted to semi supervised learning using a recently proposed algorithm proven to be more biologically plausible. The author has implemented HLMS in a supervised manner by employing HLMS, which itself is unsupervised, for tuning the hidden layers and standard LMS for the output layer. Early clustering of the input patterns at the output of the hidden layer has made the classification process faster. The recursive computation of local gradient has been completely avoided which otherwise was indispensable while training using conventional BP algorithm. Simultaneous adaptation of synapses using HLMS has improved the convergence speed. The simulation results lead us to the conclusion that it is justified to use slope variation with least number of hidden neurons instead of increasing the number of neurons in the hidden layer for enhancing the overall performance of the ANN. The network trained with modified HLMS with a variation in slope of the activation function has also proved to be a preferred alternative without excessive usage of network resources and computation time.

Finally, chapter five seeks to present a comparative analysis of adaptive filters implemented using an ANN with their conventional design approach. The ANN based adaptive filters were designed

using LMS and its sign variants and then trained employing two popular methodologies of BP (LM and GDM). It can be inferred from the simulation results that filters can be designed using an ANN with a minimally configured architecture. In the traditional design approach, a new area efficient structure for hardware implementable variable step-size adaptive filter has been proposed. The proposed structure provided an optimum area trade-off irrespective of the algorithm used to design the filter and a three-fourth improvement in speed than direct form has been witnessed with a significant lower penalty in terms of power which can make it a good alternative for more efficient ASIC and FPGA implementations in the future. The estimated hardware requirement of both approaches (ANN and conventional) presented in the chapter leads us to deduce that ANN based adaptive filter design approach is even more area efficient than its conventional counterpart.

A holistic view of this work leads us to contemplate the following take home points:

- A simple ANN (An ANN with a simple architecture) is a useful and versatile computational entity whose efficient performance is highly important since such an ANN can also serve to be a basic building block of a more complex deep network hence it is all the more important to make it more efficient both computationally and in terms of hardware.
- There are smart ways to manipulate the parameters of the ANN (e.g. adapting the learning rate, pruning the non-relevant weight) for obtaining better overall performance.
- Activation functions are important for semi-supervised style of training where adaptation of its parameters such as slope can revamp the ANN's overall performance without undue usage of network resources and computational time which otherwise was beyond the bounds of possibility with hidden neuron variation.
- Apart from classifiers, ANN as a function approximator can work as an adaptive filter capable of noise cancellation with a minimal hardware and reduced number of computations. However, appropriate choices in terms of training methodology and algorithm variants still remain important.

References

[1] F. C. Pessoa, “Morphological/rank neural networks and their adaptive optimal design for image processing,” in *Acoustics, Speech, and Signal Processing (ICASSP-96)*, 1996, vol. 6, pp. 3398–3401.

[2] M. T. Hagan and M. B. Menhaj, “Training feedforward networks with the Marquardt algorithm,” *IEEE Trans. Neural Networks*, vol. 5, no. 6, pp. 989–993, 1994.

[3] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, “Learning and transferring mid-level image representations using convolutional neural networks,” in *Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 1717–1724.

[4] S. I. Amari and A. Cichocki, “Adaptive blind signal processing-neural network approaches,” in *Proceedings of IEEE*, 1998, vol. 86, no. 10, pp. 2026–2048.

[5] J. Herault and C. Jutten, “Space or time adaptive signal processing by neural network models,” in *AIP Conference Proceedings*, 1986, vol. 151, no. 1, pp. 206–211.

[6] M. Covell, M. Arjovsky, Y. C. Lin, and A. Kokaram, “Optimizing Transcoder Quality

Targets Using a Neural Network with an Embedded Bitrate Model," in *Electronic Imaging*, 2016, vol. 7, pp. 1–7.

[7] V. Devendran, A. Wahi, and H. Thiagarajan, "ANN and SVM based Image classification using Wavelet Decomposition," *Asian J. Inf. Technol.*, vol. 6, no. 11, pp. 1174–1180, 2007.

[8] B. Shi, X. Bai, and C. Yao, "An end-to-end trainable neural network for image-based sequence recognition and its application to scene text recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 11, pp. 2298–2304, 2017.

[9] O. John and S. Prince, "Classification of flame and fire images using feed forward neural network," in *Electronics and Communication Systems (ICECS)*, 2014, pp. 1–6.

[10] A. Bhardwaj and A. Tiwari, "Breast cancer diagnosis using genetically optimized neural network model," *Expert Syst. Appl.*, vol. 42, no. 10, pp. 4611–4620, 2015.

[11] A. T. Azar and A. E. Hassanien, "Dimensionality reduction of medical big data using neural-fuzzy classifier," *Soft Comput.*, vol. 19, no. 4, 2015.

[12] R. Adhikari and R. K. Agrawal, "A combination of artificial neural network and random walk models for financial time series forecasting," *Neural Comput. Appl.*, vol. 24, no. 6, pp. 1441–1449, 2014.

[13] T. D. Chaudhuri and I. Ghosh, "Artificial Neural Network and Time Series Modeling Based Approach to Forecasting the Exchange Rate in a Multivariate Framework," *arXiv Prepr. arXiv1607.02093*, 2016.

[14] R. Kumar, R. R. Das, V. N. Mishra, and R. Dwivedi, "A radial basis function neural network classifier for the discrimination of individual odor using responses of thick-film tin-oxide sensors," *IEEE Sens. J.*, vol. 9, no. 10, pp. 1254–1261, 2009.

[15] R. Kumar, R. R. Das, V. N. Mishra, and R. Dwivedi, "Fuzzy entropy based neuro-wavelet identifier-cum-quantifier for discrimination of gases/odors," *IEEE Sens. J.*, vol. 11, no. 7, pp. 1548–1555, 2011.

[16] P.Andras, “HighDimensional Function Approximation With Neural Networks forLarge Volumes of Data,” *IEEE Trans. Neural Networks*, vol. 29, no. 2, pp. 500–508, 2018.

[17] Z. Moravej, D. N. Vishwakarma, and S. P. Singh, “Application of radial basis function neural network for differential relaying of a power transformer.,” *Comput. Electr. Eng.*, vol. 29, no. 3, pp. 421–434, 2003.

[18] L. T. Tran, H. Schepker, S. Doclo, H. H. Dam, and S. Nordholm, “Proportionate NLMS for adaptive feedback control in hearing aids,” in *Acoustics, Speech and Signal Processing (ICASSP)*, 2017, pp. 211–215.

[19] L. T. T. Tran, H. Schepker, S. Doclo, H. H. Dam, and S. Nordholm, “Improved practical variable step-size algorithm for adaptive feedback control in hearing aids,” in *Signal Processing and Communication Systems (ICSPCS)*, 2016, pp. 1–8.

[20] N.V.George and A. Gonzalez, “Convex combination of nonlinear adaptive filters for active noise control,” *Appl. Stat.*, vol. 76, pp. 157–161, 2014.

[21] M. S. Aslam and M. A. Z. Raja, “A new adaptive strategy to improve online secondary path modeling in active noise control systems using fractional signal processing approach,” *Signal Processing*, vol. 107, pp. 433–443, 2015.

[22] F. D. Freijedo, J. Doval-Gandoy, O. Lopez, P. Fernandez-Comesana, and C. Martinez-Penalver, “A signal-processing adaptive algorithm for selective current harmonic cancellation in active power filters,” *IEEE Trans. Ind. Electron.*, vol. 56, no. 8, pp. 2829–2840, 2009.

[23] K. C. Veluvolu and W. T. Ang, “Estimation and filtering of physiological tremor for real-time compensation in surgical robotics applications,” *Int. J. Med. Robot. Comput. Assist. Surg.*, vol. 6, no. 3, pp. 334–342, 2010.

[24] E. Ohara, K. I. Yano, S. Horihata, T. Aoki, and Y. Nishimoto, “Tremor suppression control of meal-assist robot with adaptive filter,” in *Rehabilitation Robotics*, 2009, pp. 498–503.

[25] F. Colone, D. W. O’hagan, P. Lombardo, and C. J. Baker, “A multistage processing

algorithm for disturbance removal and target detection in passive bistatic radar," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 45, no. 2, pp. 698–722, 2009.

[26] T. Higgins, S. D. Blunt, and A. K. Shackelford, "Time-range adaptive processing for pulse agile radar," in *Waveform Diversity and Design Conference (WDD)*, 2010, pp. 115–120.

[27] M. Z. U. Rahman, R. A. Shaik, and D. R. K. Reddy, "Efficient sign based normalized adaptive filtering techniques for cancelation of artifacts in ECG signals: Application to wireless biotelemetry," *Signal Processing*, vol. 91, no. 2, pp. 225–239, 2011.

[28] Y. Wu, R. M. Rangayyan, Y. Zhou, and S. C. Ng, "Filtering electrocardiographic signals using an unbiased and normalized adaptive noise reduction system," *Med. Eng. Phys.*, vol. 31, no. 1, pp. 17–26, 2009.

[29] E. S. Nejevenko and a. a. Sotnikov, "Adaptive modeling for hydroacoustic signal processing," *Pattern Recognit. Image Anal.*, vol. 16, no. 1, pp. 5–8, 2006.

[30] R. Yu, Y. Song, and M. Nambiar, "Fast system identification using prominent subspace LMS," *Digit. Signal Process. A Rev. J.*, vol. 27, no. 1, pp. 44–56, 2014.

[31] M. A. Quraan and D. Cheyne, "Reconstruction of correlated brain activity with adaptive spatial filters in MEG," *Neuroimage*, vol. 49, no. 3, pp. 2387–2400, 2010.

[32] A. H. Sayed, *Fundamentals of Adaptive Filtering*. New Jersey: John Wiley & Sons, 2003.

[33] A. B. Diggikar and S. S. Ardhapurkar, "Design and Implementation of Adaptive filtering algorithm for Noise Cancellation in speech signal on FPGA," in *Computing, Electronics and Electrical Technologies (ICCEET)*, 2012, pp. 766–771.

[34] M. B. I. Reaz and L. S. Wei, "Adaptive linear neural network filter for fetal ECG extraction," in *Intelligent Sensing and Information Processing*, 2004, pp. 321–324.

[35] Y. Wu and R. M. Rangayyan, "An unbiased linear artificial neural network with normalized adaptive coefficients for filtering noisy ECG signals," in *Electrical and Computer Engineering*, 2007, pp. 868–871.

[36] B. Widrow and R. Winter, “Neural nets for adaptive filtering and adaptive pattern recognition,” *Computer (Long. Beach. Calif).*, vol. 21, no. 3, pp. 25–39, 1988.

[37] A. Toprak and İ. Güler, “Impulse noise reduction in medical images with the use of switch mode fuzzy adaptive median filter,” *Digit. Signal Process.*, vol. 17, no. 4, pp. 711–723, 2007.

[38] K. Y. Lee, “Complex fuzzy adaptive filter with LMS algorithm,” *IEEE Trans. signal Process.*, vol. 44, no. 2, pp. 424–427, 1996.

[39] S. C. Ng, S. H. Leung, C. Y. Chung, A. Luk, and W. H. Lau, “The genetic search approach. A new learning algorithm for adaptive IIR filtering,” *IEEE Signal Process. Mag.*, vol. 13, no. 6, pp. 38–46, 1996.

[40] N. Karaboga, “A new design method based on artificial bee colony algorithm for digital IIR filters,” *J. Franklin Inst.*, vol. 346, no. 4, pp. 328–348, 2009.

[41] T. Kaur and S. Agrawal, “Adaptive traffic lights based on hybrid of neural network and genetic algorithm for reduced traffic congestion,” in *Enginnering and Computational Sciences*, 2014.

[42] O. Nerrand, P. Roussel-Ragot, L. Personnaz, G. Dreyfus, and S. Marcos, “Neural networks and nonlinear adaptive filtering: Unifying concepts and new algorithms,” *Neural Comput.*, vol. 5, no. 2, pp. 165–199, 1993.

[43] N. Ansari and Z. Zhang, “Generalized adaptive neural filters,” *IEEE Electron. Lett.*, vol. 20, no. 4, pp. 342–343, 1993.

[44] H. Hanek, N. Ansari, and Z. Zhang, “Comparative study on the generalized adaptive neural filter with other nonlinear filters,” in *Acoustics, Speech, and Signal Processing (ICASSP-96)*, 1993, pp. 649–652.

[45] Z. Z. Zhang and N. Ansari, “Structure and properties of generalized adaptive neural filters for signal enhancement,” *IEEE Trans. Neural Networks*, vol. 7, no. 4, pp. 857–868, 1996.

[46] Q. Xiao, G. Ge, and J. Wang, “The neural network adaptive filter model based on wavelet transform,” in *Hybrid Intelligent Systems*, 2009, pp. 529–534.

[47] G. Cai, H. Li, D. Xu, and H. Zhou, “Impulse noise filtering by using an adaptive single-linking pulse coupled neural network,” in *Software Engineering and Service Sciences (ICSESS)*, 2010, pp. 107–110.

[48] D. C. Park, C. N. Tran, and Y. Lee, “Short-term load forecasting using multiscale bilinear recurrent neural network,” in *Artificial Intelligence*, 2006, pp. 329–338.

[49] H. Zhao, X. Zeng, and Z. He, “Low-complexity nonlinear adaptive filter based on a pipelined bilinear recurrent neural network,” *IEEE Trans. Neural Networks*, vol. 22, no. 9, pp. 1494–1507, 2011.

[50] S. A. Monfared and D. Enke, “Noise Canceling in volatility forecasting using an adaptive neural network filter,” *Procedia Comput. Sci.*, vol. 61, pp. 80–84, 2015.

[51] M. H. Quazi and S. G. Kahalekar, “Artifacts removal from EEG signal: FLM optimization-based learning algorithm for neural network-enhanced adaptive filtering,” *Biocybern. Biomed. Eng.*, vol. 37, no. 3, pp. 401–411, 2017.

[52] K. Hornik, M. Stinchcombe, and H. White, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[53] S. Sonoda and N. Murata, “Neural network with unbounded activation functions is universal approximator,” *Appl. Comput. Harmon. Anal.*, vol. 43, no. 2, pp. 233–268, 2017.

[54] N. J. Guliyev and V. E. Ismailov, “On the approximation by single hidden layer feedforward neural networks with fixed weights,” *Neural Networks*, vol. 98, pp. 296–304, 2018.

[55] K. Hornik, “Approximation capabilities of multilayer feedforward networks,” *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.

[56] Y. Tassa and T. Erez, “Least squares solutions of the HJB equation with neural network value-function approximators,” *IEEE Trans. Neural Networks*, vol. 18, no. 4, pp. 1031–

1041, 2007.

[57] Z. Zainuddin and O. Pauline, “Modified wavelet neural network in function approximation and its application in prediction of time-series pollution data,” *Appl. Soft Comput.*, vol. 11, no. 8, pp. 4866–4874, 2011.

[58] T. H. Teng, “Function approximation for large markov decision processes using self-organizing neural networks,” in *Neural Networks (IJCNN)*, 2015, pp. 1–8.

[59] R. Prasad, R. Kumar, and D. Singh, “A radial basis function approach to retrieve soil moisture and crop variables from X-band scatterometer observations,” *Prog. Electromagn. Res. B*, vol. 12, pp. 201–217, 2009.

[60] R. Kumar, R. R. Das, V. N. Mishra, and R. Dwivedi, “Wavelet coefficient trained neural network classifier for improvement in qualitative classification performance of oxygen-plasma treated thick film tin oxide sensor array exposed to different odors/gases,” *IEEE Sens. J.*, vol. 11, no. 4, pp. 1013–1018, 2011.

[61] R. Kumar, R. R. Das, V. N. Mishra, and R. Dwivedi, “A neuro-fuzzy classifier-cum-quantifier for analysis of alcohols and alcoholic beverages using responses of thick-film tin oxide gas sensor array,” *IEEE Sens. J.*, vol. 10, no. 9, pp. 1461–1468, 2010.

[62] S. Chen, Y. Guo, and D. Wang, “Use of Engineering Fuzzy Sets, BP Neural Network and Genetic Algorithm for Intelligent Decision-Making,” in *Intelligent Control and Automation*, 2006, pp. 3052–3056.

[63] H. Shi, “Bid/no-bid decision-making using rough sets and neural networks,” in *Control and Decision Conference*, 2009, pp. 6075–6079.

[64] I. McLoughlin, H. Zhang, Z. Xie, Y. Song, and W. Xiao, “Robust sound event classification using deep neural networks,” *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 23, no. 3, pp. 540–552, 2015.

[65] H. Mechlih and M. M. Mechlih, “Neural network based medical decision making using wearable technology,” in *Learning and Technology Conference*, 2015, pp. 36–38.

[66] A. R. Sereshkeh, R. Trott, A. Bricout, and T. Chau, “EEG Classification of Covert Speech Using Regularized Neural Networks,” *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 25, no. 12, pp. 2292–2300, 2017.

[67] M. Anthimopoulos, S. Christodoulidis, L. Ebner, A. Christe, and S. Mougiakakou, “Lung pattern classification for interstitial lung diseases using a deep convolutional neural network,” *IEEE Trans. Med. Imaging*, vol. 35, no. 5, pp. 1207–1216, 2016.

[68] Y. Zhao, Z. Zhen, X. Liu, Y. Song, and J. Liu, “The neural network for face recognition: Insights from an fMRI study on developmental prosopagnosia,” *Neuroimage*, vol. 169, pp. 151–161, 2018.

[69] M. Ariyanto *et al.*, “Finger movement pattern recognition method using artificial neural network based on electromyography (EMG) sensor,” in *Automation, Cognitive Science, Optics, Micro Electro-Mechanical System, and Information Technology (ICACOMIT)*, 2015, pp. 12–17.

[70] M. M. Khedkar and S. A. Ladhake, “Robust human iris pattern recognition system using neural network approach 78-83). IEEE,” in *Information Communication and Embedded Systems*, 2013, pp. 78–83.

[71] Y. Tsoy, “Using neural networks for dynamic reduction of the features space dimensionality,” in *Strategic Technology*, 2012, pp. 1–6.

[72] V. A. Golovko, L. U. Vaitsekhovich, P. A. Kochurko, and U. S. Rubanau, “Dimensionality reduction and attack recognition using neural network approaches,” in *Neural Networks*, 2007, pp. 2734–2739.

[73] M. P. Naeini, H. Taremian, and H. B. Hashemi, “Stock market value prediction using neural networks,” in *Computer Information Systems and Industrial Management Applications (CISIM)*, 2010, pp. 132–136.

[74] Y. Y. Lin, J. Y. Chang, and C. T. Lin, “Identification and prediction of dynamic systems using an interactively recurrent self-evolving fuzzy neural network,” *IEEE Trans. neural networks Learn. Syst.*, vol. 24, no. 2, pp. 310–321, 2013.

[75] E. Sipos and L. N. Ivanciu, “Failure analysis and prediction using neural networks in the chip manufacturing process,” in *Electronics Technology (ISSE)*, 2017, pp. 1–5.

[76] J. Tang, F. Liu, W. Zhang, R. Ke, and Y. Zou, “Lane-changes prediction based on adaptive fuzzy neural network,” *Expert Syst. Appl.*, vol. 91, pp. 452–63, 2018.

[77] G. P. Zhang, “Neural Networks for Classification : A Survey,” vol. 30, no. 4, pp. 451–462, 2000.

[78] G. Hepner, T. Logan, N. Ritter, and N. Bryant, “Artificial neural network classification using a minimal training set- Comparison to conventional supervised classification,” *Photogramm. Eng. Remote Sensing*, vol. 56, no. 4, pp. 469–473, 1990.

[79] A. Lipton, D. Shrier, and A. Pentland, *Digital Banking Manifesto: The End of Banks?*. Massachusetts, 2016.

[80] R. Padalino, N. Pinnell, P. C. Shinn, and W. Yu, “System and method for automated debiting and settling of financial transactions,” U.S. Patent 8,650,122, 2014.

[81] X. Cui, “The internet of things,” in *Ethical Ripples of Creativity and Innovation*, 2016, pp. 61–68.

[82] O. Etzion, “When artificial intelligence meets the internet of things , pp. 246, June. 2015.,” in *Distributed Event-Based Systems*, 2015, pp. 246–246.

[83] B. H. Dobkin, “A Rehabilitation-Internet-of-Things in the Home to Augment Motor Skills and Exercise Training ,” *Neurorehabil. Neural Repair*, vol. 31, no. 3, pp. 217–227, 2017.

[84] S. Amini, I. Gerostathopoulos, and C. Prehofer, “Big data analytics architecture for real-time traffic control,” in *Models and Technologies for Intelligent Transportation Systems*, 2017, pp. 710–715.

[85] L. Xie, E. J. Draizen, and P. E. Bourne, “Harnessing big data for systems pharmacology,” 2017.

[86] M. Hengstler, E. Enkel, and S. Duelli, “Applied artificial intelligence and trust—The case

of autonomous vehicles and medical assistance devices," *Technol. Forecast. Soc. Change*, vol. 105, pp. 105–120, 2016.

[87] Y. Tian, K. Pei, S. Jana, and B. Ray, "DeepTest: Automated testing of deep-neural-network-driven autonomous cars," *arXiv Prepr. arXiv1708.08559*, 2017.

[88] L. Jones, "Driverless cars: when and where?," *Eng. Technol.*, vol. 12, no. 2, pp. 36–40, 2017.

[89] Z. Wadud, "Fully automated vehicles: A cost of ownership analysis to inform early adoption," *Transp. Res. Part A Policy Pract.*, vol. 101, pp. 163–176, 2017.

[90] D. Zhao and H. Peng, "From the Lab to the Street: Solving the Challenge of Accelerating Automated Vehicle Testing," *arXiv Prepr. arXiv1707.04792*, 2017.

[91] K. K. Parhi and T. Nishitami, *Digital Signal processing for multimedia systems*. CRC Press, 1999.

[92] B. Widrow, J. M. McCool, M. G. Larimore, and J. C. Richard Johnson, "Stationary and Nonstationary Learning Characteristics of the LMS Adaptive filter," 1976, vol. 64, no. 8, pp. 1151–1162.

[93] D. B. Parker, "Learning Logic: Technical Report TR-47," Cambridge, 1985.

[94] david E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," San Diego, 1985.

[95] B. Widrow and M. E. Hoff, "Adaptive switching circuits," in *Neurocomputing: foundations of research*, 1988, pp. 123–134.

[96] G. Spivey, S. S. Bhattacharyya, and K. Nakajima, "Logic foundry: rapid prototyping for FPGA-based DSP systems," *EURASIP J. Appl. Signal Processing*, pp. 565–579, 2003.

[97] K. Benkrid, D. Crookes, and A. Benkrid, "Design and implementation of a novel algorithm for general purpose median filtering on FPGAs," in *Circuits and Systems*, 2002, pp. 425–428.

[98] W. Fohl and J. Matthies, “An FPGA-based adaptive noise cancelling system,” in *Digital Audio Effects*, 2009, pp. 1–9.

[99] U. M. Baese, *Digital Signal Processing With Field Programmable Gate Arrays*. Berlin, Germany: Springer-Verlag, 2007.

[100] A. Rosado-Munoz, M. Bataller-Mompean, E. Soria-Olivas, C. Scarante, and J. F. Guerrero-Martinez, “FPGA Implementation of an Adaptive Filter Robust to Impulsive Noise: Two Approaches,” *IEEE Trans. Ind. Electron.*, vol. 58, pp. 860–870, 2011.

[101] V. Ramakrishna and T. A. Kumar, “Low Power VLSI Implementation of Adaptive Noise Canceller Based on Least Mean Square Algorithm,” *2013 4th Int. Conf. Intell. Syst. Model. Simul.*, vol. 2, no. 4, pp. 276–279, 2013.

[102] D. J. Allred, W. Huang, V. Krishnan, and D. V. Anderson, “An FPGA Implementation for a High Throughput Adaptive Filter Using Distributed Arithmetic,” in *Field-Programmable Custom Computing Machines*, 2004, no. 4, pp. 324–325.

[103] H. Y. Jheng, Y. H. Chen, S. J. Ruan, and Z. Qi, “FPGA Implementation of High Sampling Rate In-Car Non-Stationary Noise Cancellation Based on Adaptive Wiener Filter,” in *VLSI and System-on-chip*, 2011, pp. 114–117.

[104] Y. Mollaei, “Hardware implementation of adaptive filters,” in *Research and Development*, 2009, pp. 45–48.

[105] M. Vella and C. J. Debono, “The implementation of a high speed adaptive FIR filter on a field programmable gate array,” in *Mediterranean Electrotechnical Conference*, 2006, pp. 113–116.

[106] A. Elhossini, S. Areibi, and R. Dony, “An FPGA Implementation of the LMS Adaptive Filter for Audio Processing,” *2006 IEEE Int. Conf. Reconfigurable Comput. FPGA’s (ReConFig 2006)*, no. September, pp. 1–8, 2006.

[107] R. Mustafa, M. a. Mohd Ali, C. Umat, and D. a. Al-Asady, “Design and implementation of least mean square adaptive filter on Altera Cyclone II Field Programmable Gate Array for

active noise control," *2009 IEEE Symp. Ind. Electron. Appl.*, no. Isiea, pp. 479–484, 2009.

[108] F. Nekouei, Neda Zargar Talebi, Y. S. Kavian, and A. Mahani, "FPGA implementation of LMS self correcting adaptive filter (SCAF)," *Proc. 2012 8th Int. Symp. Commun. Syst. Networks Digit. Signal Process. CSNDSP 2012*, pp. 3–7, 2012.

[109] S. E. Fahlman, "An empirical study of learning speed in back-propagation networks," *Neural Networks*, vol. 6, no. 3, pp. 1–19, 1988.

[110] Y. Fukuoka, H. Matsuki, H. Minamitani, and Akimasa Ishida, "A modified back-propagation method to avoid false local minima," *Neural Networks*, vol. 11, no. 6, pp. 1059–1072, 1998.

[111] A. K. Srivastava, K. K. Shukla, and S. K. Srivastava, "Exploring neuro-genetic processing of electronic nose data," *Microelectronics J.*, vol. 29, no. 11, pp. 921–931, 1998.

[112] R. S. Sutton, "Two problems with backpropagation and other steepest-descent learning procedures for networks," *Proc. 8th annual conf. cognitive science society*. pp. 823–831, 1986.

[113] J. Šíma, "Back-propagation is not efficient," *Neural Networks*, vol. 9, no. 6, pp. 1017–1023, 1996.

[114] Y. H. Zweiri, J. F. Whidborne, and L. D. Seneviratne, "A three-term backpropagation algorithm," *Neurocomputing*, vol. 50, pp. 305–318, 2003.

[115] W. Bi, X. Wang, Z. Tang, and H. Tamura, "Avoiding the local minima problem in backpropagation algorithm with modified error function," *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.*, vol. E88–A, no. 12, pp. 3645–3653, 2005.

[116] C. Y. Huang, L. H. Chen, Y. L. Chen, and F. M. Chang, "Evaluating the process of a genetic algorithm to improve the back-propagation network: A Monte Carlo study," *Expert Syst. Appl.*, vol. 36, no. 2 PART 1, pp. 1459–1465, 2009.

[117] W. Wan, S. Mabu, K. Shimada, K. Hirasawa, and J. Hu, "Enhancing the generalization

ability of neural networks through controlling the hidden layers," *Appl. Soft Comput.*, vol. 9, no. 1, pp. 404–414, 2009.

[118] C. L. Lin, J. F. Wang, C. Y. Chen, C. W. Chen, and C. W. Yen, "Improving the generalization performance of RBF neural networks using a linear regression technique," *Expert Syst. Appl.*, vol. 36, no. 10, pp. 12049–12053, 2009.

[119] J. Yang, X. Zeng, S. Zhong, and S. Wu, "Effective neural network ensemble approach for improving generalization performance," *IEEE Trans. neural networks Learn. Syst.*, vol. 24, no. 6, pp. 878–887, 2013.

[120] R. Kamimura, "Simplified and gradual information control for improving generalization performance of multi-layered neural networks," in *Neural Networks (IJCNN)*, 2015, pp. 1–7.

[121] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.

[122] N. M. Nawi, A. Khan, and M. Z. Rehman, "A new back-propagation neural network optimized with cuckoo search algorithm," in *Computational Science and Its Applications*, 2013, pp. 413–426.

[123] N. M. Nawi, A. Khan, and M. Z. Rehman, "A new Levenberg Marquardt based back propagation algorithm trained with cuckoo search," *Procedia Technol.*, vol. 11, pp. 18–23, 2013.

[124] N. M. Nawi, A. Khan, and M. Z. Rehman, "A new bat based back-propagation (BAT-BP) algorithm," in *Advances in Systems Science*, 2014, pp. 395–404.

[125] G. Thimm, P. Moerland, and E. Fiesler, "The interchangeability of learning rate and gain in backpropagation neural networks," *Neural Comput.*, vol. 8, no. 2, pp. 451–460, 1996.

[126] K. Eom, K. Jung, and H. Sirisena, "Performance improvement of backpropagation algorithm by automatic activation function gain tuning using fuzzy logic," *Neurocomputing*,

vol. 50, pp. 439–460, 2003.

[127] R. A. Jacobs, “Increased rates of convergence through learning rate adaptation,” *Neural Networks*, vol. 1, pp. 295–307, 1988.

[128] C. Yu and B. Liu, “A backpropagation algorithm with adaptive learning rate and momentum coefficient,” in *Neural Networks*, 2002, pp. 1218–1223.

[129] Y. Li, Y. Fu, H. Li, and S.W.Zhang, “The improved training algorithm of back propagation neural network with self-adaptive learning rate,” in *Computational Intelligence and Natural Computing*, 2009, vol. 1, pp. 73–76.

[130] N. A. Hamid, N. M. Nawi, R. Ghazali, and M. N. Salleh, “Learning efficiency improvement of back propagation algorithm by adaptively changing gain parameter together with momentum and learning rate,” in *Software Engineering and Computer Systems*, 2011, pp. 812–824.

[131] K.Yamada, H. Kami, and J. Tsukumo, “Handwritten numeral recognition by multi-layered neural network with improved learning algorithm,” in *Neural Networks*, 1989, pp. 259–266.

[132] D. R. Hush and J. M. Salas, “Improving the learning rate of backpropagation with the gradient reuse algorithm,” in *Neural Networks*, 1988, pp. 441–447.

[133] M. K. Weir, “A method for self-determination of adaptive learning rates in back propagation,” *Neural Networks*, vol. 4, no. 3, pp. 371–379, 1991.

[134] X. H. Yu, G. A. Chen, and S. X. Cheng, “Acceleration of backpropagation learning using optimized learning rate and momentum,” *Electron. Lett.*, vol. 29, no. 14, pp. 1288–1290, 1993.

[135] X. H. Yu, G. A. Chen, and S. X. Cheng, “Dynamic learning rate optimization of the backpropagation algorithm,” *IEEE Trans. Neural Networks*, vol. 6, no. 3, pp. 669–677, 1995.

[136] X. H. Yu and G. A. Chen, “Efficient backpropagation learning using optimal learning rate

and momentum," *Neural Networks*, vol. 10, no. 3, pp. 517–527, 1997.

[137] R. Solomen and J. L. Van Hemmen, "Accelerating backpropagation through dynamic self-adaptation," *Neural Networks*, vol. 9, no. 4, pp. 589–601, 1996.

[138] V. P. Plagianakos, D. G. Sotiropoulos, and M. N. Vrahatis, "An Improved Backpropagation Method with Adaptive Learning Rate," pp. 1–8, 1998.

[139] D. Mandic and J. Chambers, "Towards the optimal learning rate for backpropagation," *Neural Process. Lett.*, pp. 1–5, 2000.

[140] M. R. Meybodi and H. Beigy, "A note on learning automata-based schemes for adaptation of BP parameters," *Neurocomputing*, vol. 48, pp. 957–974, 2002.

[141] C. H. Lee and Y. C. Lin, "An adaptive neuron-fuzzy filter design via periodic fuzzy neural network," *Signal Processing*, vol. 85, pp. 401–411, 2005.

[142] Y. H. Zweiri, L. D. Seneviratne, and K. Althoefer, "Stability analysis of threeterm backpropagation algorithm," *Neural Networks*, vol. 18, pp. 1341–1347, 2005.

[143] E. Barnard, "Optimization for Training Neural Nets," *IEEE Trans. Neural Networks*, vol. 3, no. 2, pp. 232–240, 1992.

[144] C. H. Wang, C. H. Kao, and W. H. Lee, "A new interactive model for improving the learning performance of back propagation neural network," *Autom. Constr.*, vol. 16, no. 6, pp. 745–758, 2007.

[145] Y. Li, Y. Fu, H. Li, and S. W. Zhang, "The improved training algorithm of back propagation neural network with self-adaptive learning rate," in *Computational Intelligence and Natural Computing*, 2009, pp. 73–76.

[146] E. Moulines and F. R. Bach, "Non-asymptotic analysis of stochastic approximation algorithms for machine learning," in *Advances in Neural Information Processing Systems*, 2011, pp. 451–459.

[147] T. Schaul, S. Zhang, and Y. LeCunn, "No more pesky learning rates," in *Machine Learning*,

2013, pp. 343–351.

[148] D. E. R. J. L. McClelland and P. R. Group, *Parallel distributed processing*. Cambridge, MA: MIT Press, 1987.

[149] L. W. Chan and F. Fallside, “An adaptive training algorithm for backpropagation networks, Computer Speech and Language,” *Comput. Speech Lanhuage*, vol. 2, pp. 205–218, 1987.

[150] G. Qiu, M. R. Varley, and T. J. Terrell, “Accelerated training of backpropagation networks by using adaptive momentum step,” *IEEE Electron. Lett.*, vol. 28, no. 4, pp. 377–379, 1992.

[151] X. Wu, N. K. Loh, and W. C. Miller, “A new acceleration technique for the back-propagation algorithm,” in *Neural Networks*, 1993, vol. 3, pp. 1157–1161.

[152] E. Istook and T. Martinez, “Improved backpropagation learning in neural networks with windowed momentum,” *Int. J. Neural Syst.*, vol. 12, no. 03n04, pp. 303–318, 2002.

[153] C. C. Yu and B. D. Liu, “A backpropagation algorithm with adaptive learning rate and momentum coefficient,” in *Neural Networks (IJCNN’02)*, 2002, pp. 1218–1223.

[154] H. M. Shao and G. F. Zheng, “A new BP algorithm with adaptive momentum for FNNs training,” in *WRI Global Congress on Intelligent Systems*, 2009, pp. 16–20.

[155] A. A. Hameed, B. Karlik, and M. S. Salman, “Back-propagation algorithm with variable adaptive momentum,” *Knowledge-Based Syst.*, vol. 114, pp. 79–87, 2016.

[156] M. S. Ahmad, O. Kukrer, and A. Hocanin, “An Efficient Recursive Inverse Adaptive Filtering Algorithm for Channel Equalization,” in *Wireless Conference*, 2010, pp. 88–92.

[157] M. S. Ahmad, O. Kukrer, and A. Hocanin, “Recursive inverse adaptive filtering algorithm,” *Signal Processing*, vol. 21, no. 4, pp. 491–496, 2011.

[158] M. S. Ahmad, O. Kukrer, and A. Hocanin, “A 2-d recursive inverse adaptive algorithm,” *Signal, Image Video Process.*, vol. 7, pp. 221–226, 2013.

[159] A. Hameed, M. S. Salman, and B. Karlik, “A New 2-D Convex Combination of Recur- sive

Inverse Algorithms," in *Electronics and Nanotechnology*, 2014, pp. 273–276.

[160] N. M. Nawi, R. S. Ransing, M. N. M. Salleh, R. Ghazali, and N. A. Hamid, "An improved back propagation neural network algorithm on classification problems," in *Database Theory and Application, Bio-Science and Bio-Technology*, 2010, pp. 177–188.

[161] A. Rezgui, "The effect of the slope of the activation function on the back propagation algorithm," in *Neural Networks*, 1990, pp. 707–710.

[162] S. Ivanikovas, G. Dzemyda, and V. Medvedev, "Influence of the neuron activation function on the multidimensional data visualization quality," in *Applied Stochastic Models and data Analysis (ASMDA)*, 2009, pp. 299–303.

[163] J. W. Sammon, "A Nonlinear Mapping for Data Structure Analysis," *IEEE Trans. Comput.*, vol. 18, pp. 401–409, 1969.

[164] C. C. Yu, Y. C. Tang, and B. D. Liu, "An adaptive activation function for multilayer feedforward neural networks," in *Computers, Communications, Control and Power Engineering*, 2002, vol. 1, pp. 645–650.

[165] Y. Özbay and G. Tezel, "A new method for classification of ECG arrhythmias using neural network with adaptive activation function," *Digit. Signal Process.*, vol. 20, no. 4, pp. 1040–1049, 2010.

[166] Y. Lee, S.-H. Oh, and M. W. Kim, "The effect of initial weights on premature saturation in\nback-propagation learning," *IJCNN-91-Seattle Int. Jt. Conf. Neural Networks*, vol. i, no. August, p. 4169130, 1991.

[167] K. Balakrishnan and V. Honavar, "Improving convergence of back-propagation by handling flat-spots in the output layer," in *Artificial Neural Networks*, 1992, pp. 1003–1009.

[168] X. G. Wang, Z. Tang, H. Tamura, M. Ishii, and W. D. Sun, "An improved backpropagation algorithm to avoid the local minima problem," *Neurocomputing*, vol. 56, no. 1–4, pp. 455–460, 2004.

[169] Y. Lee, S. H. Oh, and M. W. Kim, “An analysis of premature saturation in back-propagation learning,” *Neural Networks*, vol. 6, pp. 719–728, 1993.

[170] S. H. Oh, “Improving the error backpropagation algorithm with a modified error function,” *IEEE Trans. Neural Networks*, vol. 8, no. 3, pp. 799–803, 1997.

[171] J. Vitela and J. Reifman, “Enhanced backpropagation training algorithm for transient event identification. Transactions of the American Nuclear Society;(United States), 69,” *Trans. Am. Nucl. Soc.*, vol. 69, pp. 148–149, 1993.

[172] J. E. Vitela and J. Reifman., “Premature saturation in backpropagation networks: mechanism and necessary conditions,” *Neural Networks*, vol. 10, no. 4, pp. 721–735, 1997.

[173] H. M. Lee, C. M. Chen, and T. C. Huang, “Learning efficiency improvement of back-propagation algorithm by error saturation prevention method,” *Neurocomputing*, vol. 41, pp. 125–143, 2001.

[174] X. G. Wang, Z. Tang, H. Tamura, and M. Ishii, “A modified error function for the backpropagation algorithm,” *Neurocomputing*, vol. 57, no. 1–4, pp. 477–484, 2004.

[175] P. Moallem and S. A. Ayoughi, “A Complementary Method for Preventing Hidden Neurons’ Saturation in Feed Forward Neural Networks Training." relation 2 (2010): 1.,” *Iran. J. Electr. Comput. Eng.*, vol. 9, no. 2, pp. 127–133, 2010.

[176] A. P. Engelbrecht, “A new pruning heuristic based on variance analysis of sensitivity information,” *IEEE Trans. Neural Networks*, vol. 12, no. 6, pp. 1386–1399, 2001.

[177] S. E. Fahlman and C. Lebiere, “The cascade-correlation learning architecture,” in *Advances in neural information processing systems*, 1990, vol. 2, pp. 524–532.

[178] R. Setiono and L. C. K. Hui, “Use of a Quasi-Newton Method in a Feedforward Neural Network Construction Algorithm,” *IEEE Trans. Neural Networks*, vol. 6, no. 1, pp. 273–277, 1995.

[179] J. Moody and P. J. Antsaklis, “The dependence identification neural network construction

algorithm," *IEEE Trans. Neural Networks*, vol. 7, pp. 3–15, 1996.

[180] J. Zhang and A. Morris, "A sequential learning approach for single hidden layer neural networks," *Neural Networks*, vol. 11, pp. 65–80, 1997.

[181] R. Parekh, J. Yang, and V. Honavar, "Constructive neural-network learning algorithms for pattern classification," *IEEE Trans. Neural Networks*, vol. 11, no. 2, pp. 436–451, 2000.

[182] O. Aran, O. T. Yildiz, and E. Alpaydin, "An incremental framework based on cross-validation for estimating the architecture of a multilayer perceptron," *Int. J. Pattern Recognit. Artif. Intell.*, vol. 23, no. 2, pp. 159–190, 2009.

[183] J. L. Subirats, L. Franco, and J. M. Jerez, "C-Mantec: A novel constructive neural network algorithm incorporating competition between neurons," *Neural Networks*, vol. 26, pp. 130–140, 2012.

[184] B. Hammer, A. Micheli, and A. Sperduti, "Universal approximation capability of cascade correlation for structures," *Neural Comput.*, vol. 17, no. 5, pp. 1109–1159, 2006.

[185] F. Dandurand, V. I. Berthiaume, and T. R. Shultz, "A systematic comparison of flat and standard cascade-correlation using a student–teacher network approximation task," *Conn. Sci.*, vol. 19, no. 3, pp. 223–44, 2007.

[186] R. Reed, "Pruning Algorithms-A Survey," *IEEE Trans. Neural Networks*, vol. 4, no. 5, pp. 740–747, 1993.

[187] S. C. Huang and Y. F. Huang, "Bounds on the number of hidden neurons in multilayer perceptrons," *IEEE Trans. Neural Networks*, vol. 2, pp. 47–55, 1991.

[188] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, 1990, pp. 598–605.

[189] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Advances in Neural Information Processing Systems*, 1993, pp. 164–171.

[190] R. Setiono, "A penalty function approach for pruning feedforward neural networks," *Neural*

Comput., vol. 9, no. 1, pp. 185–204, 1997.

[191] T. Q. Huynh and R. Setiono, "Effective neural network pruning using cross validation," in *Neural Networks*, 2005, pp. 972–977.

[192] J. Sietsma and R. J. F. Dow, "Neural net pruning: why and how," in *Neural Networks*, 1988, vol. 1, pp. 325–333.

[193] M. Hagiwara, "A simple and effective method for removal of hidden units and weights. Neurocomputing," *Neurocomputing*, vol. 6, pp. 207–218, 1994.

[194] S. Tamura, M. Tateishi, M. Matumoto, and S. Akia, "Determination of the number of redundant hidden units in a three layered feedforward neural network," in *Neural Networks*, 1993, pp. 335–338.

[195] L. Fletcher, V. Katcovnik, F. E. Steffens, and A. P. Engelbrecht, "Optimizing the number of hidden nodes of a feed forward neural network," in *Neural Networks*, 1998, pp. 1608–1612.

[196] H. J. Xing and H. B. Gang, "Two phase construction ofmultilayer perceptrons using Information Theory," *IEEE Trans. Neural Networks*, vol. 20, no. 4, pp. 715–721, 2009.

[197] D. Whitely and C. Bogart, "The evolution of connectivity: pruning neural networks using genetic algorithms," in *Neural Networks*, 1990, pp. 134–137.

[198] P. G. Benardos and G. C. Vosniakos, "Optimizing feedforward artificial neural network architecture," *Eng. Appl. Artif. Intell.*, vol. 20, no. 3, pp. 365–382, 2007.

[199] B. Hassibi, D. G. Stork, and G. J. Wolff, "Optimal Brain Surgeon and general network pruning," in *IEEE*, 1993, pp. 293–299.

[200] X. Zeng and D. S. Yeung, "Hidden neuron pruning of multilayer perceptrons using a quantified sensitivity measure," *Neural Comput.*, vol. 69, pp. 825–837, 2006.

[201] P. Lauret, E. Fock, and T. A. Mara, "A node pruning algorithm based on a fourier amplitude sensitivity test method," *IEEE Trans. Neural Networks*, vol. 17, no. 2, pp. 273–293, 2006.

[202] J. Xua and W. C. H. Daniel, “A new training and pruning algorithm based on node dependence and Jacobian rank deficiency,” *Neurocomputing*, vol. 70, pp. 544–558, 2006.

[203] M. G. Augasta and T. Kathirvalavakumar, “Pruning algorithms of neural networks - a comparative study,” vol. 3, no. 3, pp. 105–115, 2013.

[204] M. G. A. T. Kathirvalavakumar, “A Novel Pruning Algorithm for Optimizing Feedforward Neural Network of Classification Problems,” *Neural Process. Lett.*, vol. 34, no. 3, p. 241, 2011.

[205] M. S. Medeiros and G. A. Barreto, “A novel weight pruning method for MLP classifiers based on the MAXCORE principle,” *Neural Comput. Appl.*, vol. 22, pp. 71–84, 2013.

[206] Z. Tong and G. Tanaka, “A pruning method based on Weight Variation information for Feedforward Neural Networks,” *IFAC-PapersOnLine*, vol. 48, no. 18, pp. 221–226, 2015.

[207] J. Patez, “Reducing the number of neurons in Radial Basis Function networks with dynamic decay adjustment,” *Neurocomputing*, vol. 62, pp. 79–91, 2004.

[208] P. N. L., W. H. Delashmitb, M. T. Manrya, J. Lic, and F. Maldonado, “An integrated growing-pruning method for feedforward network training,” *Neurocomputing*, vol. 71, pp. 2831–2847, 2008.

[209] N. Intrator, “Combining exploratory projection pursuit and projection pursuit regression with application to neural networks,” *Neural Comput.*, vol. 5, no. 3, pp. 443–455, 1993.

[210] N. Intrator, D. Reisfeld, and Y. Yeshurun, “Face recognition using a hybrid supervised/unsupervised neural network,” *Pattern Recognit. Lett.*, vol. 17, no. 1, pp. 67–76, 1996.

[211] M. Boudour and A. Hellal., “Combined use of unsupervised and supervised learning for large-scale power system static security assessment,” *Int. J. power energy Syst.*, vol. 26, no. 2, p. 157, 2006.

[212] A. Naseri, M. Reza, and A. R. Soroush, “Combined use of unsupervised and supervised

learning for daily peak load forecasting," *Energy Convers. Manag.*, vol. 49, no. 6, pp. 1302–1308, 2008.

[213] F. Theljani, K. Laabidi, M. Lahmari-Ksouri, and S. Zidi, "New approach for systems monitoring based on semi-supervised classification," in *Communications, Computing and Control Applications (CCCA)*, 2011, pp. 1–6.

[214] K. G. Sheela and S. N. Deepa, "Neural network based hybrid computing model for wind speed prediction," *Neurocomputing*, vol. 122, pp. 425–429, 2013.

[215] J. Moody and J. D. Christian, "Fast learning in networks of locally-tuned processing units," *Neural Comput.*, vol. 1, no. 2, pp. 281–294, 1989.

[216] K. R. Hsieh and W. T. Chen, "A neural network model which combines unsupervised and supervised learning," *IEEE Trans. Neural Networks*, vol. 4, no. 2, pp. 357–360, 1993.

[217] F. Schwenker and E. Trentin, "Pattern classification and clustering: A review of partially supervised learning approaches," *Pattern Recognit. Lett.*, vol. 37, pp. 4–14, 2014.

[218] B. Widrow, K. Youngsik, and D. Park, "The Hebbian-LMS learning algorithm," *ieee Comput. Intell. Mag.*, vol. 10, no. 4, pp. 37–53, 2015.

[219] B. Widrow, "Hebbian learning and the LMS algorithm," in *Cognitive Informatics & Cognitive Computing (ICCI* CC)*, 2016, pp. 2–2.

[220] M. Long, Y. Cao, J. Wang, and M. Jordan, "Learning transferable features with deep adaptation networks," in *Machine Learning*, 2015, pp. 97–105.

[221] C. Dong, C. C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *Computer Vision*, 2014, pp. 184–199.

[222] Y. F. Yam and W. S. Chow, "Extend least squares based algorithm for training feedforward networks," *IEEE Trans. neural networks*, vol. 8, no. 3, pp. 806–810, 1997.

[223] I. T. Jolliffe, "Principal component analysis and factor analysis," in *Principal component analysis*, Springer, 2002, pp. 150–166.

[224] W. Kasprzak and W. Skarbek, “Adaptive learning algorithm for principal component analysis with partial data,” *Cybern. Syst.*, vol. 2, pp. 1014–1019, 1996.

[225] J. Tu, Y. Zhan, and F. Han, “A neural network pruning method optimized with PSO algorithm,” in *Computer Modeling and Simulation*, 2010, vol. 3, pp. 257–259.

[226] D. O. Hebb, *The Organization of Behavior*. New York, USA: Wiley, 1949.

[227] R. Kumar and R. Dwivedi, “Quaternion Domain k-Means Clustering for Improved Real Time Classification of E-Nose Data,” *IEEE Sens. J.*, vol. 16, no. 1, pp. 177–184, 2016.

[228] E. Eweda, “Comparison of RLS, LMS, and Sign Algorithms for Tracking Randomly Time-Varying Channels,” *IEEE Trans. signal Process.*, vol. 42, no. 11, pp. 2937–2944, 1994.

[229] P. J. Werbos, “Back-propagation: Past and future,” in *Neural Networks (ICANN)*, 1988, pp. 343–354.

[230] F. M. Silva and L. B. Almeida, “Speeding up backpropagation,” in *Advanced Neural Computers*, 1990, pp. 151–158.

[231] M. Khazaei, H. S. Hosseini, A. Marjaninejad, and S. Daneshwa, “A Radial Basis Function Neural Network Approximator with Fast Terminal Sliding Mode-Based Learning Algorithm and Its Applications in Control Systems,” in *Electrical Engineering (ICEE)*, 2017, pp. 812–816.

[232] M. Gallagher and T. Downs, “Visualization of learning in multilayer perceptron networks using principal component analysis,” *IEEE Trans. Syst. Man Cybern. Part B*, vol. 33, no. 1, pp. 28–34, 2003.

[233] N. G.Chitaliya and A. I. Trivedi, “Feature extraction using wavelet-pca and neural network for application of object classification & face recognition,” in *Computer Engineering and Applications (ICCEA)*, 2010, vol. 1, pp. 510–514.

[234] W. Schenck, R. Welsch, A. Kaiser, and M. Ralf, “Adaptive learning rate control for ‘ neural gas principal component analysis ,’” vol. 277, no. April, pp. 28–30, 2010.

[235] V. R. Kompella, M. Luciw, and J. Schmidhuber, “Incremental slow feature analysis,” in *Artificial Intelligence*, 2011, pp. 1354–1359.

[236] M. Gallagher and T. Downs, “Visualization of Learning in Neural Networks using principal component analysis,” in *Computational Intelligence and Multimedia Applications*, 1997, pp. 327–331.

[237] T. J. Abrahamsen and L. K. Hansen, “A Cure for Variance Inflation in High Dimensional Kernel Principal Component Analysis,” *J. Mach. Learn. Res.*, vol. 12, pp. 2027–2044, 2011.

[238] H. Lu, K. N. Plataniotis, and A. N. Venetsanopoulos, “Uncorrelated Multilinear Principal Component Analysis for Unsuper- vised Multilinear Subspace Learning,” *IEEE Trans. Neural Networks*, vol. 20, no. 11, pp. 1820–1836, 2009.

[239] Z. Fan, J. Wang, B. Xu, and P. Tang, “An efficient KPCA algorithm based on feature correlation evaluation,” *Neural Comput. Appl.*, vol. 24, no. 7–8, pp. 1795–1806, 2014.

[240] O. M. Machhi and N. J. Bershad, “Adaptive recovery of a chirped sinusoid in noise. I. Performance of the RLS algorithm,” *IEEE Trans. Signal Process.*, vol. 39, no. 3, pp. 583–594, 1991.

[241] S. J. Julier and J. K.Uhlmann, “A new extension of the Kalman filter to nonlinear systems,” in *Signal processing, sensor fusion, and target recognition VI*, 19987, vol. 3, no. 26, pp. 182–194.

[242] H. K. Khalil, *Noninear Systems*, vol. 2, no. 5. New Jersey: Prentice-Hall, 1996.

[243] L. Behera, S. Kumar, and A. Patnaik, “On adaptive Learning rate that guarantees Convergence in feedforward Networks,” *IEEE Trans. Neural Networks*, vol. 17, no. 5, pp. 1116–1125, 2006.

[244] R. Kohavi, “A study of cross-validation and bootstrap for accuracy estimation and model selection,” in *Artificial Intelligence (IJCAI)*, 1995, vol. 14, no. 2, pp. 1137–1145.

[245] M. Riedmiller, “Advanced supervised learning in multi-layer perceptrons—from

backpropagation to adaptive learning algorithms," *Comput. Stand. Interfaces*, vol. 16, no. 3, pp. 265–278, 1994.

[246] S. Haykins, *Neural Networks and learning machines*. USA: Pearson, 2007.

[247] D.P.Mesquita, J. Gomes, L. R. Rodrigues, and R. K. Galvao, "Pruning extreme learning machines using the successive projections algorithm," *IEEE Lat. Am. Trans.*, vol. 13, no. 12, pp. 3974–3979, 2015.

[248] R. J. Legg, *The SAGE Handbook of Spatial Analysis edited by A. Stewart Fotheringham and Peter A. Rogerson*. Wiley, 2010.

[249] S. Han *et al.*, "EIE: efficient inference engine on compressed deep neural network," in *Computer Architecture*, 2016, pp. 243–254.

[250] S. K. Esser *et al.*, "Convolutional networks for fast, energy-efficient neuromorphic computing," in *National Academy of Sciences of United States of America (PNAS)*, 2016, vol. 113, no. 41, pp. 11441–11446.

[251] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv Prepr. arXiv1510.00149.*, 2015.

[252] J. Qiu *et al.*, "Going deeper with embedded fpga platform for convolutional neural network," in *Field-Programmable Gate Arrays*, 2016, pp. 26–35.

[253] A. Shafiee *et al.*, "ISAAC: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," in *Computer Architecture*, 2016, pp. 14–26.

[254] G. B. Kingston, H. R. Maier, and M. F. Lambert, "A statistical input pruning method for artificial neural networks used in environmental modelling," in *Biennial Meeting of the International Environmental Modelling and Software Society*, 2004, pp. 87–92.

[255] D. Sabo and X. H. Yu, "A new pruning algorithm for neural network dimension analysis," in *Neural Networks (IJCNN)*, 2008, pp. 3313–3318.

[256] C. Xiang, S. Q. Ding, and T. H. Lee, “Geometrical interpretation and architecture selection of MLP,” *IEEE Trans. Neural Networks*, vol. 16, no. 1, pp. 84–96, 2005.

[257] S. Trenn, “Multilayer perceptrons: Approximate order and necessary number of hidden units,” *IEEE Trans. Neural Networks*, vol. 19, no. 5, pp. 836–844, 2008.

[258] Y. T. Fan, W. Wu, W. Y. Yang, Q. W. Fan, and J. Wang, “A pruning algorithm with L 1/2 regularizer for extreme learning machine,” *J. Zhejiang Univ. Sci. C*, vol. 15, no. 2, pp. 119–125, 2014.

[259] D. Mantzaris, G. Anastassopoulos, and A. Adamopoulos, “Genetic algorithm pruning of probabilistic neural networks in medical disease estimation,” *Neural Networks*, vol. 24, no. 8, pp. 831–835, 2011.

[260] S. H. Yang and Y. P. Chen, “An evolutionary constructive and pruning algorithm for artificial neural networks and its prediction applications,” *Neurocomputing*, vol. 86, pp. 140–149, 2012.

[261] N. H. Christiansen, J. H. Job, K. Klyver, and J. Hogsbrg, “Optimal brain surgeon on artificial neural networks in nonlinear structural dynamics,” in *Nordic Seminar on Computational Mechanics*, 2012.

[262] A.S.Weigend, D. E. Rumelhart, and B. A. Huberman, “Generalization by weight-elimination with application to forecasting,” in *Advances in neural information processing systems*, 1991, pp. 875–882.

[263] I. Gómez, L. Franco, and J. M. Jerez, “Neural network architecture selection: can function complexity help?,” *Neural Process. Lett.*, vol. 30, no. 2, pp. 71–87, 2009.

[264] T. Nakamura, K. Judd, A. I. Mees, and M. Small, “A comparative study of information criteria for model selection,” *Int. J. Bifurc. Chaos*, vol. 16, no. 8, pp. 2153–2175, 2006.

[265] K. Etminani, M. Naghibzadeh, and A. R. Razavi, “Effective pruning strategies for branch and bound Bayesian networks structure learning from data,” *Sci. Iran.*, vol. 20, no. 3, pp. 682–694, 2013.

[266] S. M. Kay, *Fundamentals of statistical signal processing*. USA: Prentice Hall, 1993.

[267] K. D. Miller and D. J. C. MacKay, “The role of constraints in Hebbian learning,” *Neural Comput.*, vol. 6, no. 1, pp. 100–126, 1994.

[268] C. U. A. Kappers, G. C. Hube, and E. C. Crosby, “The comparative anatomy of the nervous system of vertebrates, including man.” 1936.

[269] Y. Choe, “Anti-Hebbian learning,” *Computational Neuroscience*. Springer-Verlag, pp. 1–4, 2014.

[270] P. Földiak, “Forming sparse representations by local anti-Hebbian learning,” *Biol. Cybern.*, vol. 64, no. 2, pp. 165–170, 1990.

[271] J. Rubner and K. Schulten, “Development of feature detectors by self-organization,” *Biol. Cybern.*, vol. 62, no. 3, pp. 193–199, 1990.

[272] A. Carlson, “Anti-Hebbian learning in a non-linear neural network,” *Biol. Cybern.*, vol. 64, no. 2, pp. 171–176, 1990.

[273] A. Sudjianto and M. H. Hassoun, “Nonlinear Hebbian rule: A statistical interpretation,” in *Neural Networks*, 1994, pp. 1247–1252.

[274] R. Kempter, W. Gerstner, and J. L. Van Hemmen, “Hebbian learning and spiking neurons,” *Phys. Rev. E*, vol. 59, no. 4, p. 4498, 1999.

[275] J. L. McClelland, “How far can you go with Hebbian learning, and when does it lead you astray,” *Process. Chang. brain Cogn. Dev. Atten. Perform. xxi*, vol. 21, pp. 33–69, 2006.

[276] J. R. Movellan, “Contrastive Hebbian learning in the continuous Hopfield model,” in *Connectionist Models*, 1991, pp. 10–17.

[277] P. Mazzoni, R. A. Andersen, and M. I. Jordan, “A more biologically plausible learning rule for neural networks,” in *National Academy of Sciences of United States of America (PNAS)*, 1991, vol. 88, no. 10, pp. 4433–4437.

[278] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain,” *Psychol. Rev.*, vol. 65, no. 6, pp. 386–408, 1958.

[279] J. Dong-Gyu and L. Soo-Young, “Merging Back-propagation and Hebbian Learning Rules for Robust Classifications,” *Neural Networks*, vol. 9, no. 7, pp. 1213–1222, 1996.

[280] B. Widrow, A. Greenblatt, Y. Kim, and D. Park, “The No-Prop algorithm: A new learning algorithm for multilayer neural networks,” *Neural Networks*, vol. 37, pp. 182–188, 2013.

[281] S. Kelkar and R. Kamal, “Adaptive fault diagnosis algorithm for controller area network,” *IEEE Trans. Ind. Electron.*, vol. 61, no. 10, pp. 5527–5537, 2014.

[282] B. Widrow and S. Stearns, *Adaptive Signal Processing*. Englewood Cliffs, New Jersey: Prentice Hall, 1985.

[283] S. Haykin, *Adaptive Filter Theory*. Englewood Cliffs, New Jersey: Prentice Hall, 1986.

[284] G. S. Kang and L. J. Fransen, “Experimentation with an Adaptive Noise-Cancellation Filter,” *IEEE Trans. Circuits Syst.*, vol. 34, no. 7, pp. 753–758, 1987.

[285] K. J. Lee, J. P. Lee, D. Shin, D. W. Yoo, and H. J. Kim, “A Novel Grid Synchronization PLL Method Based on Adaptive Low-Pass Notch Filter for Grid-Connected PCS,” *IEEE Trans. Ind. Electron.*, vol. 61, pp. 292–301, 2014.

[286] D. T. M. Slock, “On the Convergence Behavior of the LMS and the Normalized LMS Algorithms,” *IEEE Trans. signal Process.*, vol. 41, no. 9, pp. 2811–2825, 1993.

[287] S. Choudhary, P. Mukherjee, M. Chakraborty, and S. S. Rath, “A SPT Treatment to the Realization of the Sign-LMS Based Adaptive Filters,” *IEEE Trans. Circuits Syst.*, vol. 59, pp. 2025 – 2033, 2012.

[288] K. Mayyas, “A Variable Step-Size Selective Partial Update LMS Algorithm Jordan University of Science and Technology,” *Digit. Signal Process.*, vol. 23, pp. 75–85, 2013.

[289] K. Mayyas, “Performance analysis of the selective coefficient update NLMS algorithm in an undermodeling situation,” *Digit. Signal Process. A Rev. J.*, vol. 23, no. 6, pp. 1967–1973,

2013.

[290] D. L. Jones, “Learning Characteristics of Transpose-Form LMS Adaptive Filters,” *IEEE Trans. Circuits Syst. II Analog Digit. Signal Process.*, vol. 39, no. 10, pp. 745–749, 1992.

[291] V. R. Mankar and A. A. Ghatol, “Design of Adaptive Filter Using Jordan/Elman Neural Network in a Typical EMG Signal Noise Removal,” *Adv. Artif. Neural Syst.*, pp. 1–9, 2009.

[292] P. K. Meher and S. Y. Park, “Critical-path analysis and low-complexity implementation of the LMS adaptive algorithm,” *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 61, no. 3, pp. 778–788, 2014.

[293] S. Chen, G. J. Gibson, C. F. N. Cowan, and P. M. Grant, “Adaptive equalization of finite non-linear channels using multilayer perceptrons,” *Signal Processing*, vol. 20, no. 2, pp. 107–119, 1990.

[294] T. McCannon, N. Gallagher, D. Minoo-Hamedani, and G. Wise, “On the design of nonlinear discrete-time predictors,” *IEEE Trans. Inf. Theory*, vol. 28, no. 2, pp. 366–371, 1982.

[295] A. Landi, P. Piaggi, M. Laurino, and D. MenicucciI, “Artificial neural networks for nonlinear regression and classification,” in *Intelligent Systems Design and Applications (ISDA)*, 2010, pp. 115–120.

[296] O. Nerrand, P. Roussel-Ragot, L. Personnaz, G. Dreyfus, and S. Marcos, “Neural networks and non-linear adaptive filtering: Unifying concepts and new algorithms,” *Neural Comput.*, vol. 5, no. 99, pp. 165–197, 1993.

[297] R. Kumar and S. Sharma, “Hardware Efficient Second Order Implementation of Sigmoid Function using Distributed Arithmetic,” *Accept. Publ. IEEE VLSI Syst. Lett.*, 2018.