MATHEMATICAL MODELLING OF HIDDEN LAYER ARCHITECTURE IN ARTIFICIAL NEURAL NETWORKS

Nimalka Mihirini Wagarachchi

(118024B)

Degree of Doctor of Philosophy

Department of Computational Mathematics

University of Moratuwa
Sri Lanka

August 2018

MATHEMATICAL MODELLING OF HIDDEN LAYER ARCHITECTURE IN ARTIFICIAL NEURAL NETWORKS

Nimalka Mihirini Wagarachchi

(118024B)

Thesis submitted in partial fulfillment of the requirements for the degree Doctor of Philosophy

Department of Computational Mathematics

University of Moratuwa

Sri Lanka

August 2018

Declaration

I declare that this is my own work and this thesis does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief, it does not contain any material previously published or written by another person except where the acknowledgement made is made in the text.

Also, I hereby grant to University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books)

Signature: Date:

The above candidate has carried out the research for the PhD thesis under my supervision.

Signature of the supervisor: Date:

Dedicated to

My beloved Father and Mother

Acknowledgements

Many people have helped their best to successfully completion of this research. I acknowledge all of them for their valuable thoughts and constant encouragement given me to make my project a reality.

First and foremost, I acknowledge my supervisor Senior Professor Asoka Karunananda for accepting me as his research student and giving excellent support and advices. Prof. Karunananda is a great mentor who guided whilst giving me all the freedom and encouragement to accompany with my own ideas. Without his patient listening and creative thoughts this work would not have been possible at all.

Very special thank goes to Prof. Sarath Pieris and Dr. Uditha Rathnayake for their invaluable comments and guidance as my examiners of bi-annual review panels.

I acknowledge all the office bearers and the staff of the HETC project for granting me financial assistance by awarding the HETC scholarship to smooth functioning of this research. Also express my sincere thanks to the staff of OTS office, university of Ruhuna for their wholehearted support.

I graciously acknowledge the support of Senior Prof. Gamini Senanayake, The Vice Chancellor, Senior Prof. Susirith Mendis, former Vice Chancellor, Dr. Nayana Alagiyawanna, former Dean/ Faculty of Engineering and the present Deputy Vice Chancellor University of Ruhuna for selecting me as an HETC candidate of the University of Ruhuna and giving their utmost support and guidance throughout.

I wish to extend my sincere thanks for the support I received from all the members of the administration office and members of the Faculty of Information Technologies, University of Moratuwa. Especially I thank Ms. Dilini Kulawansa, Dr. Subha Fernando and Dr. Thushari Silva for their important roles.

Also, I thank all the academic and non-academic staff of Faculty of Engineering, University of Ruhuna for their kind-hearted help to fulfill my research work.

My graciously acknowledgment to the friendly assistance given by Dr. M. K. Abeyrathne, Dr. Subashi, Ms. Malkanthi, Mr. Samantha and all my colleagues of the Department of Interdisciplinary Studies, Faculty of Engineering, University of Ruhuna.

Very special and heartfelt thanks for Budditha and Chinthanie for their gracious associations throughout the last couple of years.

I acknowledge the sacrificial dedication of my family members, especially my husband Pramud and our daughter Dinithi Navodya for their encouragement and corporate by managing all the works while I was busy with my works on this research.

Abstract

The performance of an Artificial Neural Network (ANN) strongly depends on its hidden layer architecture. The generated solution by an ANN does not guarantee that it has always been devised with the simplest neural network architecture suitable for modeling the particular problem. This results in computational complexity of training of an ANN, deployment, and usage of the trained network. Therefore, modeling the hidden layer architecture of an ANN remains as a research challenge. This thesis presents a theoretically-based approach to prune hidden layers of trained artificial neural networks, ensuring better or the same performance of a simpler network as compared with the original network.

The method described in the thesis is inspired by the finding from neuroscience that the human brain has a neural network with nearly 100 billion neurons, yet our activities are performed by a much simpler neural network with a much lesser number of neurons. Furthermore, in biological neural networks, the neurons which do not significantly contribute to the performance of the network will naturally be disregarded. According to neuroplasticity, biological neural networks can also solicit activations of neurons in the proximity of the active neural network to improve the performance of the network. On the same token, it is hypothesized that for a given complex-trained ANN, we can discover an ANN, which is much more simplified than the original given architecture.

This research has discovered a theory to reduce certain number of hidden layers and to eliminate disregarding neurons from the remaining hidden layers of a given ANN architecture. The procedure begins with a complex neural network architecture trained with backpropagation algorithm and reach to the optimum solution by two phases. First, the number of hidden layers is determined by using a peak search algorithm discovered by this research. The newly discovered simpler network with lesser number of hidden layers and highest generalization power considered for pruning of its hidden neurons. The pruning of neurons in the hidden layers has been theorized by identifying the neurons, which give least contribution to the network performances. These neurons are identified by detecting the correlations regarding minimization of error in training. Experiments have shown that the simplified network architecture generated by this approach exhibits same or better performance as compared with the original large network architecture. Generally, it reduces more than 80% of neurons while increasing the generalization by about 30%. As such, the proposed approach can be used to discover simple network architecture relevant to a given complex architecture of an ANN solution. Due to its architectural simplicity, the new architecture has been computationally efficient in training, usage and further training.

Keywords: Artificial neural networks, backpropagation algorithm, delta value, hidden layer architecture, neuroplasticity

CONTENTS

List of Figures

List of Tables

Abbreviations

ADALINE – Adaptive linear neuron

AI – Artificial Intelligence

ANN – Artificial neural network

Bi-search algorithm – Binary search algorithm

CNS – Central nervous system

$\gamma_{\delta_h,E}$ – Correlation coefficient of the sum of the delta values of h^{th} hidden layer and the output error

etc. – etcetera

i.e. – That is

LTD – long term depression

LTP – long term potentiation

MADALINE – Many ADALINE

MBP – Magnitude based pruning

MLP – Multilayer perceptron

MRI – The Magnetic Resonance Imagine

NN – Neural network

OBD – Optimal brain damage

OBS – Optimal brain Surgeon

PNS – Peripheral nervous system

PSA – Peak search algorithm

PSDV – Peak search and delta value algorithm (The proposed algorithm)

RBF – Radial basis function

SOM – Self Organizing Map

SVZ – Sub ventricular zone

CHAPTER 1

INTRODUCTION

1.1 Prolegomena

The field of Artificial Neural Networks (ANNs) has become one of the most cited areas of Artificial Intelligence (AI). Because of their capability of modelling and processing parallel on nonlinear relationships of inputs and outputs ANNs can be used in many real-world problems, which could not be solved otherwise. The first ANN model was introduced in 1940s and grown through 1950s. Throughout last 50 years with its significant developments, nowadays ANN has become the champion of Machine Learning. However, there are numerous challengers in designing and developing of ANNs and determining the most suitable architecture has been identified as one of the central issues. This thesis examines the current approaches on designing the ANN architectures and provides a novel approach to determine more efficient and simpler ANN architecture by pruning hidden layer neurons in a trained ANN.

This chapter discusses the problem that addresses in this thesis with the background and motivation. Further, it provides aims and objectives followed by the other's works and their limitations. Next, provides a brief description of the approach to the solution by emphasizing the hypothesis. At the end, the structure of organization of this thesis will be presented.

1.2 Aims and Objectives

The aim of this PhD thesis is to discover an approach to prune hidden layer neurons of a trained network to get a simplified network with same or better performance, compared the resultant network with the original trained network, enhance the generalization ability and improve the efficiency of the network.

To achieve the above aims the following objectives have been recognized.

1. Critical review of ANNs and their uses.

2. In depth study of current approaches to model hidden layer in ANN.
3. Develop an approach to prune hidden layer architecture of ANNs.
4. Evaluate of the novel approach.

1.3 Background and Motivation

The generalization power of an Artificial Neural Network (ANN) strongly depends on the number of hidden layers. In general, as there are enough data to capture the complexity of the given task, multi-layered architectures show better performance than shallow ones for many real valued applications [1], [2]. However, this solution of architecture may not be computationally optimized. In addition, networks with too large and too small number of hidden neurons show advantages as well as disadvantages. When the network is too large, it learns fast [3]. In addition large networks form complex decision regions as problem requires and show better fault tolerance in damage conditions [4]. However, when there are too many parameters, generalization ability declines as it fails to distinguish similar neurons. In contrast, networks with too few parameters show better generalization, nevertheless neurons in these networks do not learn data properly [5], [6].

Artificial Neural Networks are mathematical and computational models for predicting and decision making, inspired by the functions of the biological central nervous system. They are very advanced modelling systems capable to solve many highly complex tasks. The human brain has amazing features that can memorize and learn from data [7]. Also, when a part of the central nervous system is damaged, some other neurons maximize their functions to compensate for the damaged ones. This process is known as fault tolerance. Likewise, ANNs are designed and developed with similar architecture as structure of the biological nervous system (Figure 1.1). In addition, as the human brain does, ANNs are able to remember the features of specific tasks and use them to predict on that task in the future. Because of these properties nowadays neural networks have been applied to many real world problems in various fields such as medicine, agriculture, finance, and engineering. Certainly, when there is a problem on prediction or classification, neural networks

are being used because ANNs have shown promising results in solving non-algorithmic complex problems.

Although there are many advantages of neural networks, the complexity of its error surface has become a significant problem. The complexity of the error surface occurs because of some barriers like local minima, flat spot plateaus, and saddle points. So that the training network has become more complex and also crucial in performance [8], [9]. In this context, modeling hidden layer architecture is very important to achieve a better performance for the given problem. Therefore, this research addresses the problem clearly and approach the hidden layer architecture by using a pruning method inspired by the concepts of neuroplasticity [10].

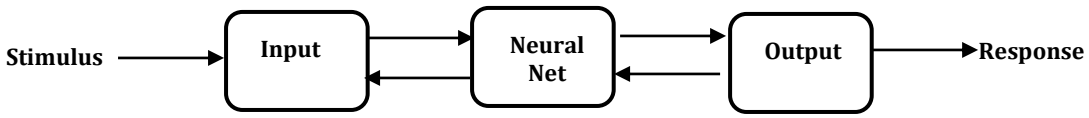


Figure 1.1: Diagram representation of central nervous system

1.4 The Problem in Brief

Despite many advantages, there are major difficulties in applying ANNs in real-world problems. One of the major issues in application of neural networks is determining the size of the most appropriate neural network architecture, which used in the particular problem, i.e. the size of the neural network architecture is a decisive factor.

The number of training cycles (epochs) and the generalization power are the two main measures to determine the performance of the network. Generalization refers, how the network performs for any data which was not used in training process [11]. It has been observed that the complex models with too many hidden neurons show poor generalization as it could not distinguish very close parameters. When there are too few neurons, the network may not learn properly and yields inaccurate solutions [12]. On the other hand, generalization power can be improved by using more than one hidden layer. However, too large networks increase the complexity of the

architecture and there is a high probability to have local minima problem [5]. Moreover, networks with many hidden layers arise complex computations and it needs much training time.

Therefore, the determining the optimum ANN architecture for modelling the given task is very important and still it is a research challenge as the available methods do not yield the optimized solution with the proper theoretical background.

1.5 Current Approaches to Modelling Hidden Layer Architecture

Generally, there are two fundamental approaches to determine the most appropriate hidden layer architecture, namely constructive and pruning algorithms. Constructive algorithms generate with a minimum number of neurons and iteratively increase neurons and connection weights of hidden layers to reach the most appropriate architecture [13],[14]. In contrast, Pruning algorithms start with an oversized network and iteratively eliminate unimportant neurons from hidden layers until the optimum solution occurs [15],[16].

There are many approaches based on these techniques. Setiono [17] has proposed a penalty term method to prune the network, where at the end of training, the terms with smallest values become zero. Also in magnitude based pruning (MBP) methods assume that smaller weights are irrelevant and eliminate them form the network structure [18]. Further, optimal brain damage (OBD) [16] and optimal brain surgeon (OBS) [19] have proposed to determine the less salience neurons based on the second derivative of the cost function. In addition, many constructive algorithms had been proposed in numerous ways to obtain the optimal architecture [13], [20]. There are also hybrid methods which used both addition and deletion of hidden neurons [21], [22].

However, both constructive and pruning methods have advantages as well as disadvantages. The constructive algorithms are computationally economic because they initialize with simple networks. Nevertheless, these solutions are more likely to

have the local minima problem as error surface of small networks are more complicated than error surface of large networks [23]. On the other hand, successively train of smaller network until the smallest one occurs is time consuming. In the pruning methods, normally large networks allow reasonable fast learning and while reduced, network performs better generalization. However, the pruning techniques are hypothesized, that the initially large sized network allows the network to learn with less accuracy for the initial conditions [24]. In addition training a large network is not computationally economic as it takes more training time. Also, inappropriate deletion of nodes and connections may cause to loose information. But authors of [12] have shown that the overall time taken to prune large network to a smaller one is relatively very favorable with that of training a small network with fewer number of neurons. Hence, it has observed that pruning techniques are widely used in improving the generalization.

1.6 The Proposed Solution

The approach to the solution is motivated by the finding from neuroscience that the human brain is a neural network with more than hundred billion neurons [25], yet our activities are performed by a simpler network with a much lesser number of neurons. Furthermore, in biological neural networks, the neurons that do not significantly contribute to the network performance will be naturally disregarded [26]. According to neuroplasticity, biological neural networks can also solicit activations of neurons in the proximity of the active neural network to improve the performance of the network [27]. By the same token, it is hypothesized that for a given complex-trained artificial neural network, it can discover a network, which is much more simplified than the original complex architecture but still performs same or better than the original one.

Another inspiration fact that a single hidden layer architecture does not yield the best solution in every instance. This idea was stimulated by the functions of neocortex in the human brain. As shown in the Figure 1.2, the neocortex is the largest part of the outermost layer of the mammalian brain, which believes

responsible for intelligence such as perception, imagination, language, art, music, mathematics, and planning [28]. It is assumed that the neocortex area of the human brain has a columnar structure with six layers which contains billions of neurons [29]. Although artificial neural networks do not perform exactly similar way as the neocortex, this is a momentous factor to have some large number of hidden layers to neural network architecture.

The approach describe in this thesis is based on the hypothesis that any large ANN could be pruned to a smaller sized network without lowering the performance. Thus, the procedure starts with a complex network with a large number of hidden layers and hidden neurons which was trained by backpropagation algorithm [30],[31].

The procedure of achieving the optimum architecture has two phases. Firstly, it designs an algorithm to determine the number of hidden layers in network which shows the highest generalization. This is done by using the Peak-Search Algorithm (PSA) which will be described in Chapter 5. In the second phase, the newly discovered simpler network with the highest generalization power is considered for pruning of neurons in its hidden layers. The pruning of neurons in the hidden layers has been theorized by identifying the neurons that give least contribution to the error decay process. These neurons are identified by detecting correlation ($\gamma_{\delta_h,E}$) [32] between the sum of the delta values of the hidden layer *h* and the output error (*E*) of the training cycle, regarding minimization of error in training. While pruning disregarded neurons, synaptic weights attached to the removable neuron will be merged with weights of a neuron which is having similar weight vector. The new method obtained by using Peak Search algorithm and Delta Value is accronymed as the PSDV algorithm.

Experiments showed that the simplified ANN architectures generated by new approach exhibit the same or better performance as the original large network architecture. As such, our approach can be used to discover simpler network architecture relevant to a given complex architecture of an ANN problem. Due to its

architectural simplicity, the new architecture has been computationally efficient in training, usage, and further training.

The results proved by experimentally using some bench mark problems. The experimental results show that the network obtained by newly introduced method performs better generalization with lesser number of hidden neurons compared to the original network.

1.7 Resource requirements

To reach the goals of the above approach, some software and hardware have been needed. MATLAB has been used in training the backpropagation algorithm. The proposed methodology needs an adequate hardware such as processor of 2.50 GHz and 4GB RAM.

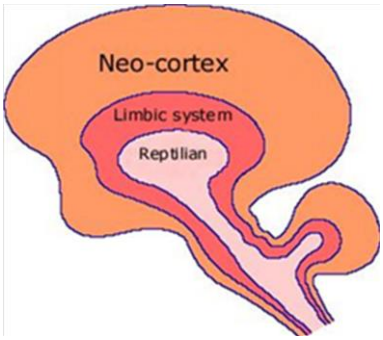


Figure 1.2: The structure of the human brain

1.8 Organization

The rest of this thesis is arranged as follows. A detailed description of artificial neural networks will be discussed in chapter 2, which includes the different types of networks, training methodologies and evolution of ANNs. Chapter 3 presents brief but necessary overview of modelling techniques in artificial neural networks.

Basically, these approaches considered under two major titles pruning and constructive algorithms. Chapter 4 contains the concept of neuroplasticity and synaptic pruning. It briefly discusses how these concepts relate to the ANNs. The methodology is presented in chapter 5. It discusses the peak-search algorithm and pruning method applied in this thesis. The experiments and results appear in chapter 6. The details of data sets and the results of all the benchmark problems present in this section. In recent years concept of deep learning networks has become a hot topic in machine learning and using the results for some deep learning networks will discuss in chapter 7. Finally, the conclusion is given in chapter 8. It discusses in brief how the model performs for different data sets and limitations arise while training the data sets.

1.9 Summary

This chapter briefly discussed the background and motivation of the research problem by highlighting the importance of modelling of the hidden layer architecture. In addition adequate description of ANN was provided. Further, a brief description of pruning and constructive methodologies and their limitations were given. It clearly stated the aims and objective of this research and briefly explained the proposed solutions. Finally, it discussed the structure of the rest of this thesis. The next chapter discusses the fundamental concepts on artificial neural networks that used in our approach.

CHAPTER 2

FUNDAMENTALS OF ARTIFICIAL NEURAL NETWORKS

2.1 Introduction

The previous chapter gave an introduction to the context of thesis by stating the aims and objectives, the problem in brief and the method of achieving the solution of the hidden layer architecture in artificial neural networks. The Artificial neural networks are densely interconnected, parallel computational models for the human brain. Some of the most important features of ANNs are their adaption to the nature, ability of learn by experience and the fault tolerance. This chapter focuses on these basic aspects together with the key development stages of ANNs. In addition, various types of neural network structures and learning rules of ANNs are taken to the discussion.

2.2 Preamble to the Artificial Neural Networks

An Artificial Neural Network generally referred to as a neural network (NN) is an information processing system that inspired by the functions of the human central nervous system. It is a massively parallel and highly connected distributive processor, made up with artificially designed units called artificial neurons, which have capability to store the acquired knowledge from the environment and use it when necessary. The knowledge is saved in the adjustable interconnected weights called synaptic weights [33]. In other words, functions of neural networks are very similar to the behavior of human brain and they yield the corresponding output when the input is presented.

The human brain, which is formed by elementary units called neurons, is highly complex, massively parallel and nonlinear information-processing system. The brain is capable to manage its own structure to perform certain tasks like pattern recognition and classification faster than any modern computer [34] . On the same line, artificial neural networks are created as mathematical and computational model to simulate the functions of biological nerve cells and their interconnection. The

most important feature of an artificial neural network is the adaptability to the environment by changing its structure by experience [35], [36].

Because of these characteristics, ANNs have become very useful and they have been applied to solve variety of problems in pattern recognition, optimizing and associative memory in many areas such as medicine, agriculture, physics and geology. Although some conventional approaches were applied to solve such problems, many of them were not successfully performed well for their input domains. However, almost all the applications could benefit from using ANNs as they show exciting alternative results [37], [38].

The artificial neural networks were introduced once the people realized that the functions of von Neumann machines are far different from the human brain [33]. Both human brain and computers have many similarities such as increase their memory, transmit the signals, use electrical signals to send messages, able to solve mathematical and logical problems, and need energy and much more [39]. However, they use different techniques in these functions. For example, brain uses chemicals to transmit the signals while computers use electricity. The brain acquires energy from nutrients, but computers need electricity to work. Moreover, the capability of the modern computer's on solving some complex mathematical based problems is much faster than that of humans. On the contrary, they are not able to perform in some perceptual instance such as face recognition. We, human can easily recognize the face of a friend even in crowd place or recognize him by his voice without seeing him. However, those fastest modern digital computers available today are not capable to do such because the architectural design of them is totally different from the topology of the human brain and these differences strongly affect the performance of the system. The significant differences of such functions are presented by jain et al. [38] as shown in the below Table 2.1.

Table 2.1: Comparison of Von Neumann computer and the human brain

Function	**Von Neumann computer**	**Human brain**
Processor	Complex High speed One or few	Simple Low speed A large number
Memory	Separate from a processor Localized Non-content addressable	Integrated internal processor Distributed Content addressable
Computing	Centralized Sequential Stored programs	Distributed Parallel Self-learning
Reliability	Very vulnerable	Robust
Expertise	Numerical and symbolic manipulations	Perceptual problems
Operating environment	Well defined Well constrained	Poorly defined Unconstrained

These differences highly affect on the performance of the function. Thus, realizing these differences made significant role in revealing mechanism of the neural information processing. As the result of that artificial neural networks were created by adopting such functions of the human nervous system.

Artificial neural networks are massively parallel distributive structures which can use to solve parameters involving non-linear and noisy, complex data. In addition, they are powerful tools which mimic the learning process of the human brain in modelling, especially when the relationship between input and target is unknown. ANNs has ability to recognize the correlated patterns and after training, it can be used to predict the output for new input data. Another important characteristic of an ANN is its adaptivity to the nature. They have a built-in capability to adapt the surrounding environment by changing the synaptic weights. Also, neural networks

are fault tolerant. That is, if a group of neurons or their connecting links are damaged, the network has ability to recall the stored pattern to compensate for the damage ones and performs without degrading the quality. Therefore, nowadays ANNs have been recognized as very sophisticated modelling technique capable of model the highly complex nonlinear models [33]. So that they have been used to solve problems in classification and pattern recognition in a very wide range of domains.

2.3 The History

The basic concept and some background work on the field of neural networks happened in the late 19th and early 20th centuries with the fundamental works of Hermann Von Helmholtz, Ernst Mach and Ivon Pavlov in different areas such as physics, psychology and neurophysiology. This primary works are highlighted the theories of fields such as learning, vision, and conditioning, but have not emphasized a mathematical model of neuron operations [40]. The beginning of neurocomputing can be traced back to the research article published in 1943 [41] by American neurophysiologist Warren McCulloh and logician Walter Pitts. In this article, they had shown that how the brain could produce highly complex patterns by using many basic cells called neurons that are connected together. In addition, it emphasized that even a simple model of artificial neurons could, in principle, compute any arithmetic or logical function. The first model of ANN that McCulloh and Pitt was presented is shown in the following Figure 2.1.

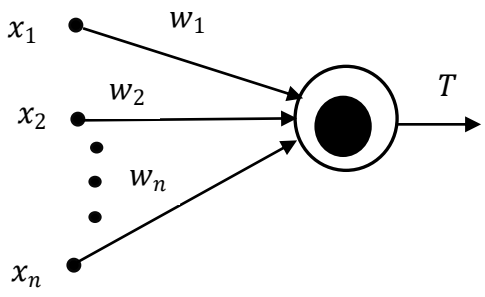


Figure 2.1: Model of ANN proposed by McCulloh and Pitt

In 1949 Donald Hebb claimed in his book entitled “The Organization of Behavior” [42] that classical psychological conditioning is presented because of the properties of an individual neuron. However, this idea was initially proposed by Ivan Pavlov (1849-1936) and Hebb developed it by proposing a specific learning rule for the synapses of neurons.

The first practical application of the artificial neural network was introduced with the invention of perceptron rule by Frank Rosenblatt, Charles Whiteman and their colleagues [43] in late 1950s. This perceptron network was successfully used in pattern recognition. At the same time Bernard Widraw and Ted Hoff introduced a new learning algorithm, Widrow-Hoff learning rule [44] to train adaptive linear neural networks (ADALINE) which is still using. Both these networks struggled with their inherent limitations and they were not able to successfully modify the network to overcome those limitations. Thus, in the late 1960s, people believed that the ANN has reached to its dead end. Until 1980 interest of neural networks faltered due to several reasons, such as lack of new ideas and insufficiency of powerful computers to continue experiments. In 1980, with availability of newly developed personal computers, research on ANNs significantly increased and researchers came with several new concepts. The Backpropagation learning rule published by David Rumelhart [30] is considered as one of the most important concepts of the rebirth of neural networks.

With the invention of the backpropagation algorithm, the research interest on ANNs was dramatically increased. Over the last few decades, it has been successfully applied in a wide area including, medicine, engineering, geology, physics, finance, and biology.

2.4 Structure of Artificial Neural Networks

The two main components of a neural network are processing elements and connections. Processing elements are known as neurons or nodes. The link between any two neurons is called the synaptic weight connection. A weight parameter is

assigned to each connection and the weight of the connection from i^{th} neuron to j^{th} neuron is denoted by w_{ji}. Each neuron receives input signal x_i from the environment or an adjacent neuron, multiplied by the weight vector and sum up all such terms. Then passes the summation through an activation function f to outside or another neuron connected to it. The neurons that receive signals from the environment are known as input neurons and neurons send to signals outside are known as output neurons [45].

In the model of McCulloh and Pitts each neuron computes the weighted sum of n inputs $x_j : j = 1, \dots\dots, n$ and corresponding synaptic weights w_j; associates with each input and produce the output y. The output is 1 if the sum is greater than a certain threshold value u, otherwise 0.
The mathematical expression of the output is

$$y = \theta\left(\sum_{j=1}^{n} w_j x_j - u\right), \tag{2.1}$$

where $w_j : j = 1, \dots\dots, n$ are the weights, y is the output of the network and $\theta(.)$ represents the unit step function. The positive weights correspond to excitatory synapses and negative weights represent inhibitory ones. For computational simplicity threshold value u is considered as another weight w associates to an input with value 1.

In general, an artificial neural network can be represented as a weighted directed graph, where artificial neurons are nodes and synaptic weights are the directed edges. Each network necessarily contains a certain input neurons $x_1, x_2, \cdots, x_n$ and output neurons $y_{1,}\, y_2, \cdots, y_m$, and it assumes that these neurons lie in layers. So that the basic structure of an ANN contains an input layer, an output layer as shown in the Figure 2.2

Based on the architecture, artificial neural networks can be divided into two types, such as

- Feedforward networks
- Recurrent networks

2.4.1 Feedforward networks

The directed graphs with no loops are feedforward networks. Neurons on these networks are organized into layers. They have static behavior. That is, rather than provide a sequence of outputs, feedforward networks give only one output at a time for a particular input. The main categories of feedforward architectures are single layer feedforward networks, multilayer feed forward networks and radial basis function nets. These networks are known to be 'memoryless' as each input is independent of the previous positions. The simplest form of these networks has two layers, input layer and output layer. However, the most widely used types of feedforward networks are multilayer perceptron (MLP), which contains a certain number of hidden layers in addition to input and output layers.

Generally, neurons in the input layer receive signals of information from the environment and the output layer is responsible to produce the output of the network. The number of neurons in the input and output layers are fixed and equal to the number of elements in the input and output vectors of the data set respectively. The section of hidden layers is the most crucial part of the network. This imitates the functions of the human brain. That is most of the internal processing are carried out by the hidden part of the network. The number of hidden layers and the number of neurons in each hidden layer is uncertain and the solution of the network strongly depends on the hidden layer architecture of a neural network.

Single Layer Feedforward Networks

The expansion of the research on feedforward layered networks began in late 1950s with the presenting of single layer feedforward networks on perceptron by

Rosenblatt [43] and ADALINE by Widrow [46]. Generally, a single layer feed forward architecture has only an input layer and an output layer as shown in the Figure 2.2. These networks are used in solving only linearly separable problems. This limitation led to researchers to improve the system and hence, multilayered feedforward networks were introduced.

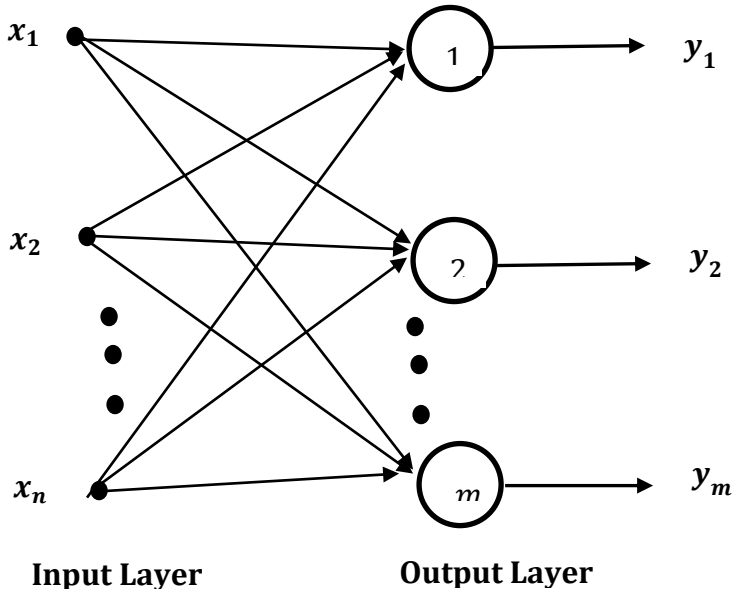


Figure 2.2: Single layer feedforward network

Multilayer Feedforward Networks

The feedforward networks with multiple hidden layer units, where all are adaptive, are known as multilayer feedforward neural networks (Figure 2.3). Unlike single hidden layer networks, multilayer networks can be used in a wide range of applications such as pattern classification, face recognition, function approximation optimization and many more. These structures must contain at least one hidden layer. Hidden nodes of the network, which contains only one hidden layer receive signals from the input layer and pass the processed signal to the output layer. When there is more than one hidden layer, the signals received by the input layer, pass to the first hidden layer and each hidden neuron pass the processed signal to the neuron in adjacent hidden layer neurons. Finally, the neurons in the last hidden layer send the signals to the output layer. Having more hidden neurons, network enable to

extract more information rather than simple ones due to the extra set of synaptic weights and more connections [33]. In multilayer network, it is not necessary to connect each node with all the other nodes in the adjacent layer. However, the feedforward network with all the possible connection is called a fully connected network. But there are instances that some links may missing. Those networks are known as partially connected networks.

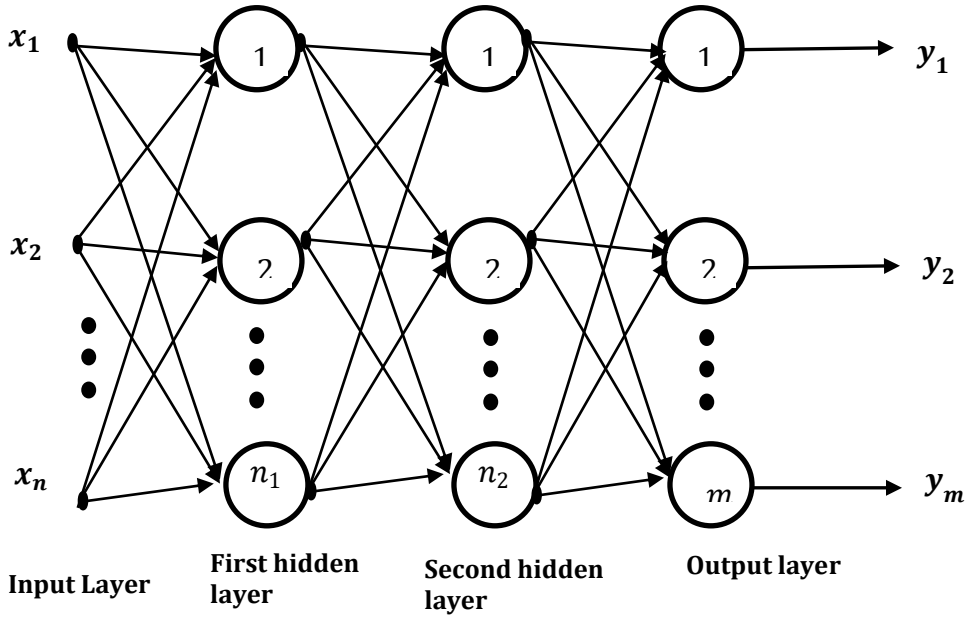


Figure 2.3: Multilayer feed forward network

Radial Basis Function Neural Nets

Radial basis function (RBF) neural nets are the two layer feed forward neural nets whose activations functions are radial basis functions such as Gaussian function $\left(f(r) = e^{-(\varepsilon r)^2}\right)$ [47]. As in MLP the output nodes apply the linear summation of functions (Figure 2.4). The networks train in two stages. First, adjust the weights form input layer to hidden layer, then adjust the weights from hidden layer to output layer [48].

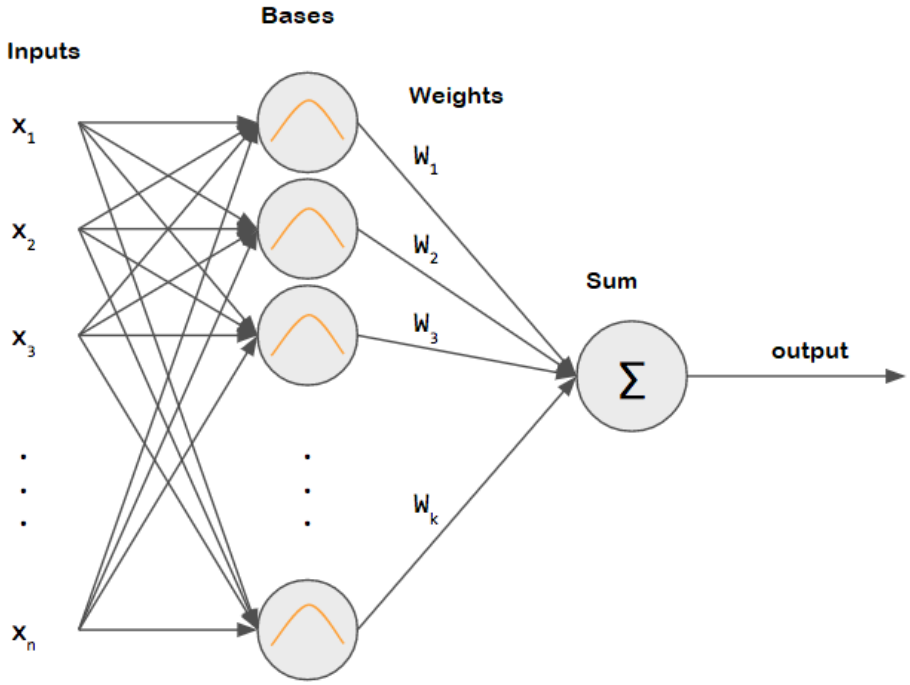


Figure 2.4: Radial basis function network

2.4.2 Recurrent networks

The recurrent networks contain feedback loops and hence, their procedure differs from the feedforward networks (Figure 2.5). The most common recurrent networks are competitive networks, Kohonen's Self Organizing Map (SOM) and Hopfield networks. Unlike feed forward networks, recurrent networks are dynamic systems. Once the output computed for an input neuron, it may feed back to the same neuron and hence, input neuron is altered. Thus, the network leads a new state [38].

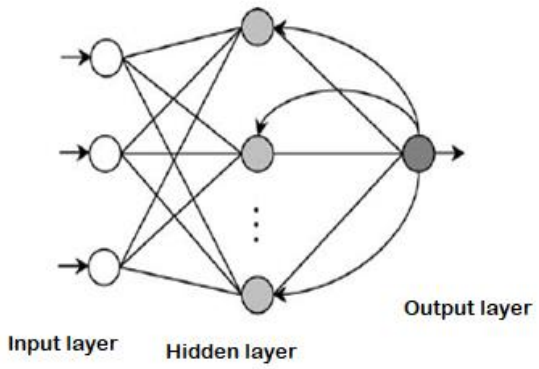


Figure 2.5: Recurrent network

The classification of neural networks accordance their architecture is described in the Figure 2.6. The different network architectures use different learning methods and sections 3.6 and 3.7 briefly discuss the learning process of neural networks.

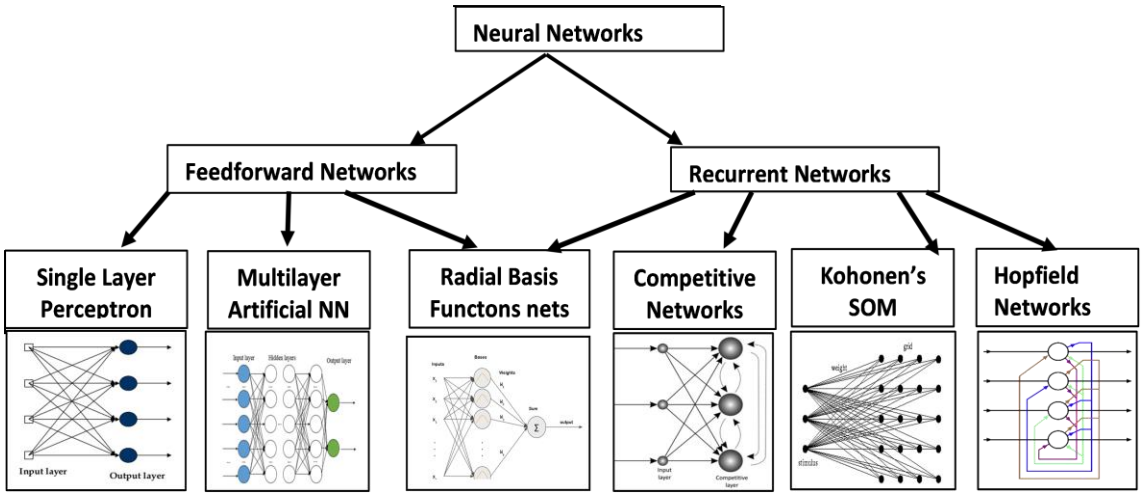


Figure 2.6: Classification of Neural Networks by architecture

2.5 Activation Functions

In a neural network an activation function, also referred as a transfer function or a cost function plays a major role in transferring the signals. It controls the information propagation of each layer by using its nonlinear property. Different activation functions have different characteristics and thus, they work in different ways. In general, the same activation function is applied to all the neurons in a layer. In this section we discuss some widely used activation functions, namely hard limit function, linear function and sigmoid functions.

2.5.1 Hard limit activation functions

The activation of the type

$$f(x) = \begin{cases} 1; & \text{if } x \geq 0 \\ 0; & \text{if } x < 0 \end{cases} \tag{2.2}$$

is called a threshold function. This function is also known as hard limit transfer function, which is illustrated in the Figure 2.7. In NN, the total input of the j^{th} neuron v_j is given as

$$v_j = \sum_{i=1}^{n} w_{ji} x_i + b_j, \tag{2.3}$$

where x_i and w_{ji}, $i = 1, 2 \ldots n$ are input vectors and corresponding weights respectively. b_j is the bias vector.

So that the output of the j^{th} neuron is stimulated as

$$y_j = \begin{cases} 1; & \text{if } v_j \geq 0 \\ 0; & \text{if } v_j < 0 \end{cases} \tag{2.4}$$

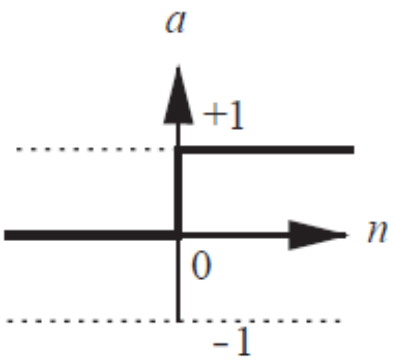


Figure 2.7: Hard limit function

2.5.2 Linear function

In the linear function, the output of any particular neuron equals to its input. As shown in the Figure 2.8, the function is written as $f(x) = x$.

When the input of the neuron j is v_j, the output $y_j = v_j$. This function is used in the ADALINE neural networks.

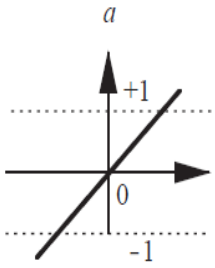


Figure 2.8: Linear function

2.5.3 Sigmoid functions

A strictly increasing function, whose graph is S-shape is called a sigmoid function. The two sigmoid functions, logsigmoid and tansigmoid which depict in the Figure 2.9 are the most commonly used cost functions, especially for the construction in hidden layers. The log sigmoid function is given as

$$f(x) = \frac{1}{1 + e^{-\lambda x}}, \tag{2.5}$$

where λ is the slope of the curve. By changing the parameter λ, curves with different slopes can be obtained. When λ tends to infinity, sigmoid functions becomes hard limit function. However, for finite values of λ sigmoid function is differentiable. But hard limit function is not.

The tan sigmoid function, given by the equation 2.3 lies between -1 and +1.

$$f(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.6}$$

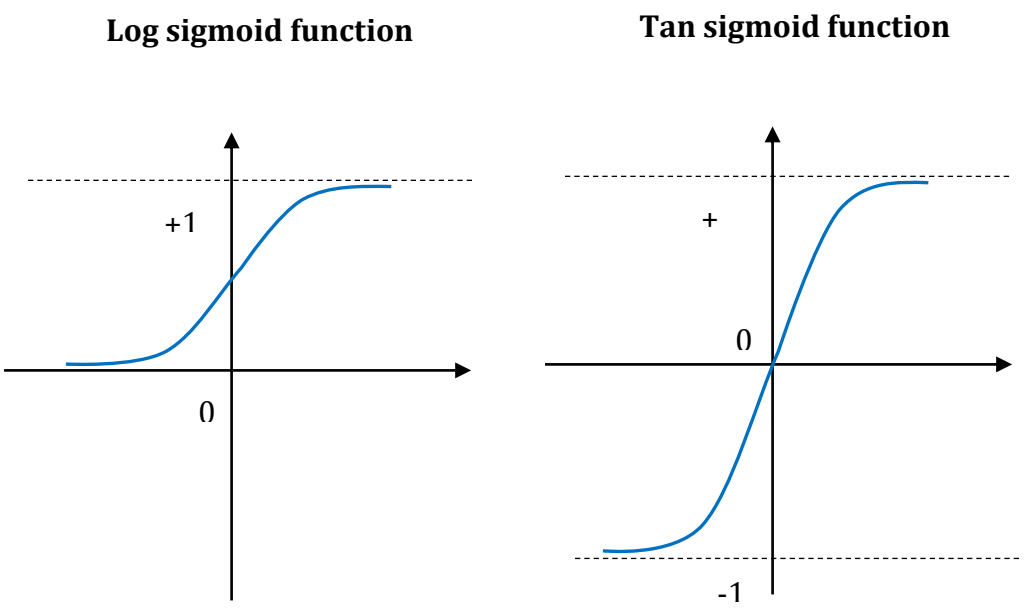


Figure 2.9: Sigmoid functions

2.6 Neural Network Learning

One of the basic common features of the human brain and artificial neural networks is their ability to learn. We human beings are able to learn from our surrounding environment by using different techniques. Sometimes we used to learn with a teacher and sometimes learn ourselves, without a teacher. So, it is with artificial neural networks. They can learn or train in similar ways and perform as humans achieve their goals. So that, in artificial neural network's framework learning is the updating the topology of the network by iteratively changing its neurons and synaptic connection to achieve its specific tasks successfully. One of the major advantages of artificial neural networks is unlike conventional expert systems, they use the underlying rules in the certain set of examples, instead of following a set of rules specified by human experts [38]. In the context of ANN, the learning with a teacher is viewed as supervised learning. The learning without a teacher has two subdivisions known as unsupervised training and reinforcement learning.

2.6.1 Supervised learning

Supervised learning also referred as 'learning with a teacher' is inspired by the concept that the teacher has the knowledge on environment. This knowledge is represented by input-output data set $\{(x_1, t_1), \dots \dots (x_n, t_n)\}$. Where $x_1, \dots x_n$ are inputs and $t_1, \dots t_n$ are corresponding target outputs. When inputs apply to the network, it computes the output and measure the *error signal*. Error signal refers the difference between actual and target output. Then it adjusts the synaptic weights and biases to minimize the error signal (Figure 2.10).

2.6.2 Reinforcement learning

The reinforcement learning is like supervised learning. However, here algorithm gives a scalar or a grade instead the correct output for each input. There is no teacher or exact output at each step of learning and the main goal of this learning procedure is to minimize the expectation of the cumulative cost of action on the

steps of iterations. These applications are not much more common as supervised learning.

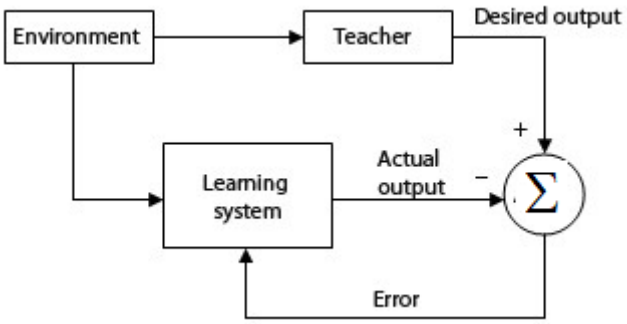


Figure 2.10: Block diagram for supervised learning

2.6.3 Unsupervised learning

As shown in the Figure 2.11 below, in unsupervised learning, there is no teacher or a critic to observe the learning process. Hence, in unsupervised learning algorithms, instead of providing specific task to learn, it introduces a task independent measure to be learned and the network is self-organized by adjusting the synaptic weights and biases to achieve those measures. By applying this procedure sequentially one layer at time, it is able to extend to train large networks with many hidden layers. For a particular task, once the network has become tuned to the statistical regularities, it is able to form internal representation which encodes features of the input in a more explicit or simple form [49].

Unsupervised learning algorithms are used in competitive-learning rule, where the nodes compete with each other give the chance to respond to a subset of the input data [50]. In its simplest form network works according to the winner-takes-all strategy. That is neuron with the greatest input wins and turn it on while rest of all the other neuron turn off [33].

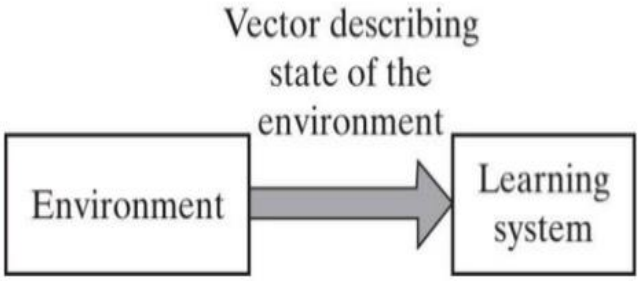


Figure 2.11: Block diagram for unsupervised learning

2.7 Learning Algorithms

The machine learning concept was initially presented by Nilson [51] in 1965. The learning process of artificial neural networks is very similar to that in biological neural system. Some neurons can be trained only by using local signals, while some may require knowledge of output neurons. As discussed in previous section, some neurons may need a teacher to learn and some are able to self-learning without a teacher. We can notice that all these learning are done accordance to a set of systematic rules. Here we discuss the some of the basic learning rules such as

- Hebbian learning
- Error correction learning
- Boltzmann learning
- Competitive learning

2.7.1 Hebbian learning

Hebbian Learning algorithm, referred as *Hebb's postulate of learning* is the oldest learning algorithm which was proposed by the neurophysiologist Donald O. Hebb [42] in 1949. This was initially introduced as a possible mechanism for synaptic modification for the brain cells and since then, it was used to train artificial neural networks. Hebb's book stated the following postulate (pg. 62).

"When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change

takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

This change in Hebb's learning was proposed for associative learning. Hebb's postulate expand and rephrase in to two-part rule as follows [33].

1. If two neurons on either side of a synapse (connection) are activated simultaneously (i.e., synchronously), then the strength of that synapse is selectively increased.

2. If two neurons on either side of a synapse are activated asynchronously, then that synapse is selectively weakened or eliminated.

These synapses are called Hebbian synapses. Mathematically Hebbian learning can be expressed as follows.

Let the signals of two neurons j and k be x_j and y_k respectively, and connection weight from neuron j to neuron k be w_{kj}. Then x_j and y_k are called pre-synaptic and post-synaptic respectively. Thus the weight update of w_{kj} can be written as a function of both pre-synaptic and post-synaptic signals. That is

$$\Delta w_{kj}(n) = f\big(y_k(n).x_j(n)\big) \tag{2.7}$$

Function f has many forms all such with the Hebb's postulate. So that the simplest form of above equation (2.7) can be written as

$$\Delta w_{kj}(n) = \eta y_k(n).x_j(n) \tag{2.8}$$

where η is a positive constant known as learning rate. The most important feature of this relation is weight update depends only on the output signals of two neurons. This rule is defined for unsupervised learning and it does not require the information of desired output.

2.7.2 Error correction learning rules

In supervised learning, the network is trained for input/output patterns $\{(x_1, t_1), \ldots \ldots (x_n, t_n)\}$, where $x_1, \ldots, x_n$ are inputs and $t_1, \ldots, t_n$ are desired outputs. The main objective of the learning is to minimize the error between actual output and the desired output. Actual output y_i is the output generated by the network for the input x_i. The error signal of the output neuron j is defined as

$$e_j = y_j - t_j.$$

The fundamental concept of error correction rules is to gradually reduce e_j by upgrading the connection weights. The first practical error correction learning rule is the perceptron learning rule.

2.7.2.1 Perceptron Learning Rule

The perceptron was the first algorithmically illustrated network and hence, it takes very special place in neural network learning. The algorithm was introduced by Rosenblatt [43] in the late 1950s. The features of this network is similar to the network introduced by McCulloh and Pitts [41]. In the model of McCulloh and Pitts, the output is determined by comparing the weighted sum of the input signals to a threshold value. If the sum is greater than the certain threshold value output is 1, otherwise the output is zero (0). The key feature introduced in perceptron network is the learning rule in pattern recognition. The learning was simple and automatic. Also, it can learn for random weights and biases. The learning occurs only when network gives an error (Figure 2.12). Rosenblatt proved in his literature that this learning rule always converge to the correct weights. This is known as the "*perceptron convergence theorem"*.

Variable and Parameters

$x(n)$: Input vector

$w(n)$: Weight vector

$b(n)$: Bias

$y(n)$: Actual output

$d(n)$: Desired output

η: learning rate parameter, (a number between 0 and 1)

Initialization

$w = 0;$
$b = 0;$

Do the following until stopping condition is satisfied

Compute $y(n)$ the output for each input $x(n)$

error $(e(n))= d(n) - y(n)$

$$w(n+1) = w(n) + \eta \times e(n) \times x(n)$$

$$b(n+1) = b(n) + \eta \times e(n)$$

Figure 2.12: Perceptron Algorithm

The perceptron model, which is shown in the below Figure 2.13 contains only a single neuron and limited only for pattern classification problems. The node computes the sum of the linear combination of all inputs and corresponding synaptic weights and bias. The resulting sum applied to the hard limiter. If this resulting sum is positive or zero network gives $+1$, otherwise the output is -1.

As shown in the Figure 2.13 inputs to the single node are represented by $x_1, x_2, \ldots\ldots, x_n$, and corresponding synaptic weights are $w_1, w_2, \ldots\ldots, w_n$. The externally applied bias is denoted by b. The net input for the single neuron is

$$v = \sum_{i=1}^{n} w_i x_i + b \tag{2.9}$$

Then output y is given as

$$y = \begin{cases} +1; & \text{if } v_j \geq 0 \\ -1; & \text{otherwise} \end{cases} \tag{2.10}$$

In two class classification problem, classes are separated by the decision boundary given by the linear equation

$$v = 0 \tag{2.11}$$

If $y = 1$ in the above equation (2.10), perceptron assign input to one particular class and if $y = -1$ to the other class.

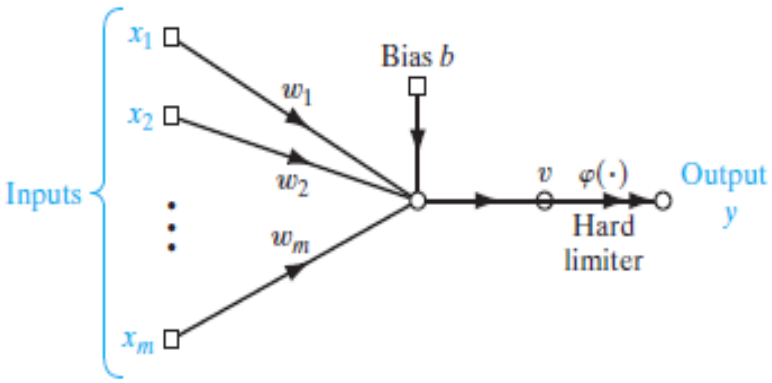


Figure 2.13: The perceptron learning rule

However, the perceptron learning algorithm has inherent limitations, which were highlighted in the book written by Marvin Minsky and Saymour Papert [52]. They argue that the perceptron network fails in applying certain elementary functions to understand more complex networks.

Until 1980s there was no significant development of ANNs to overcome this problem. With the presented of backpropagation algorithm, ANN became very popular and a dramatically increase of research of ANN can be seen. Nevertheless, the perceptron learning algorithm is considered as a very important algorithm as understanding of perceptron network helps to provides good basic knowledge to learning process of ANNs.

2.7.2.2 Backpropagation Learning

The backpropagation algorithm, depicts in the Figure 2.14, is the most widely used learning algorithm in artificial neural networks. Although, it was firstly introduced

in 1970, the importance of the algorithm was fully appreciated with the paper [30] published by David Rumelhart, Geoffrey Hinton and Ronal Williams in 1986 [53]. The intend of the backpropagation algorithm is to minimize the error of the network is given by the cost function

$$E = \sum_{j=1}^{m} \left(o_j - t_j\right)^2, \tag{2.12}$$

where o_j and t_j are actual and desired outputs of the neuron j of the output layer. m is the number of neurons in the output layer. The algorithm expresses $\partial E / \partial w$ of the cost function with respect to any weight w and this expression tells that how quickly cost changes with respect to the weights and the bias. Backpropagation is a fast learning algorithm. Not only that, but also it tells in detail that how the changes of the bias affects in the overall behavior of the network during the training.

The procedure initializes with choosing random weights. After choosing weights, network computes the cycle error by using the error function shown in equation 2.7. The algorithm contains four major steps as follows [54],

1. Feedforward computation
2. Backpropagation to the output layer
3. Backpropagation to the hidden layers
4. Weights update

The algorithm stops when the error decreased to the required level.

Feedforward Computation

The input vector $[x_1, x_2, \dots\dots, x_p]^T$ presents to the network. Weights are initialized and evaluated derivatives of the activation functions are also fed at each neuron. The input of the j^{th} neuron of the output layer at the n^{th} iteration is

$$net_j(n) = \sum_{i=1}^{n_h} w_{ji}(n) f_h(net_i(n)) \tag{2.13}$$

h is the number of hidden layers and w_{ji} is the connection weight between i^{th} neuron of h^{th} hidden layer and j^{th} neuron of the output layer. Number of neurons in h^{th} hidden layer is n_h. When the activation function of the output layer is f_o, the output of neuron j at n^{th} iteration is

$$y_j(n) = f_o\left(net_j(n)\right) \tag{2.14}$$

Then the error of j^{th} neuron of output layer at n^{th} iteration is

$$e_j(n) = t_j(n) - y_j(n) \tag{2.15}$$

Where $t_j(n)$ is the expected value of the j^{th} neuron at n^{th} iteration.

Then the instantaneous total error of the whole network is defined as

$$E(n) = \frac{1}{2}\sum_{j=1}^{m}\left(t_j(n) - y_j(n)\right)^2$$

$$= \frac{1}{2}\sum_{j=1}^{m} e_j^2\,(n) \tag{2.16}$$

Where m is the total number of neurons in the output layer.

According to steepest descent algorithm [55], [56] the change of weight vector is proportional to

$$\frac{\partial E}{\partial w_{ji}}$$

i.e.

$$\Delta w \propto \frac{\partial E}{\partial w_{ji}}$$

$$\Longrightarrow \qquad \Delta w = -\eta\frac{\partial E}{\partial w_{ji}}, \tag{2.17}$$

where η is the learning rate. As the change of weight spaces, reduces the error it holds the minus sign.

By applying the chain rule

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial e_j}.\frac{\partial e_j}{\partial y_j}.\frac{\partial y_j}{\partial\left(net_j\right)}.\frac{\partial\left(net_j\right)}{\partial w_{ji}} \tag{2.18}$$

By differentiating equation (2.16) with respect to e_j

$$\frac{\partial E}{\partial e_j} = e_j \tag{2.19}$$

Differentiate equation (2.14) with respect to y_j. Then we get

$$\frac{\partial e_j}{\partial y_j} = -1 \tag{2.20}$$

Then by differentiating equation (2.14) with respect to net_j

$$\frac{\partial y_j}{\partial (net_j)} = f_o'(net_j) \tag{2.21}$$

and by differentiating equation (2.13) with respect to w_{ji}

$$\frac{\partial (net_j)}{\partial w_{ji}} = f_h(net_i) \tag{2.22}$$

Then equation (2.18) implies that

$$\frac{\partial E}{\partial w_{ji}} = e_j(-1) f_o'(net_j) f_h(net_i) \tag{2.23}$$

From equations (2.17) and (2.23)

$$\Delta w = \eta e_j f_o'(net_j) f_h(net_i) \tag{2.24}$$

Now define δ_j as follows

$$\delta_j = e_j f_o'(net_j) \tag{2.25}$$

Backpropagation to the output layer

When *j* is an output layer neuron, it is straight forward and easy to compute. In this case the change of synaptic weights calculates according to equation (2.27) below by applying the delta value derived by equation (2.25).

Backpropagation to the hidden layers

When *j* is a neuron in hidden layer h, a desired value is not assigned to this neuron. Then the error signal is computed recursively and backward by using the error signals of all the neurons in layer $(h+1)$, that connected to neuron j directly. Then, as described in [30] and [33] delta value $\delta_j^h(n)$ for hidden neurons *j* of layer *h* can be given as

$$\delta_j^h(n) = f_h'\left(net_j^h(n)\right)\sum_{k=1}^{h_n}\delta_k^{h+1}(n)w_{kj}^{h+1}(n), \tag{2.26}$$

where h_n is the number of hidden neurons in the h^{th} layer and δ_k^{h+1} is the delta value of k^{th} neuron of $(h+1)^{\text{th}}$ layer.

Weights update

After computing delta values, synaptic weights update in the negative gradient direction as shown in the equation (2.17). The new weight of w_{ji} is expressed as

$$w_{ji}(n+1) = w_{ji}(n) + \eta_j\delta_j f_h(net_i) \tag{2.27}$$

Weights are updated only after the computation of backpropagation error in all the neurons. Otherwise, corrections intertwined with the backpropagation of the error and weights updated do not correspond to the negative gradient direction [54].

Backpropagation Learning Algorithm

1. Insert the input vector of the first pattern
2. Initialize the random weights
3. Propagate the signal forward through the network to compute the error

$$e_j = t_j - y_j$$

4. Compute δ values for the output layer

$$\delta_j = e_j f_o'(net_j)$$

5. Compute δ values of preceding layers by propagating the errors backward

$$\delta_j^h = f_h'(net_j^h)\sum_{k=1}^{h_n}\delta_k^{h+1}w_{kj}^{h+1}$$

6. Update the weights

$$w_{ji}(n+1) = w_{ji}(n) + \eta_j\delta_j f_h(net_i)$$

7. Repeat the process with the next pattern until the error becomes less than the pre-defined value or the maximum number of iterations is reached.

Figure 2.14: Backpropagation learning algorithm

2.7.3 Boltzmann learning

The Boltzmann learning was invented by Hinton et al. [57] in 1985. Neural networks, which use Boltzmann learning, are known as Boltzmann machines. They are symmetric recurrent networks, which make stochastic decisions whether to be on or off ($+1$ for 'on' and -1 for 'off'). Symmetric refers that the synaptic weight of neurons i to neuron j equals to the synaptic weight of neuron j to neuron i. That is $w_{ij} = w_{ji}$. The neurons of Boltzmann machine are portioned into two functional groups; *visible* and *hidden* as shown in the Figure 2.15.

The visible units make the interface between network and the environment. During the training all the visible units are clamped for specific states by the environment. Also, in testing any subset of visible units may be clamped. But the hidden units behave in opposite way. If they exist, they never clamp and always activate freely [58].

The objective of Boltzmann learning is to produce a neural network that accurately models input patterns according to the Boltzmann learning. In the Boltzmann learning the correction of the weight w_{ji} from i^{th} neuron to j^{th} neuron is given as

$$\Delta w_{ji} = \eta\left(\bar{\rho}_{ji} - \rho_{ji}\right), \tag{2.28}$$

where η is the learning rate. $\bar{\rho}_{ji}$ and ρ_{ji} are the corrections between the states of neurons i and j when the network operates the clamped mode and non-clamped mode respectively. These values are computed by Monte Carlo experiments [59], which is known as a very slow process.

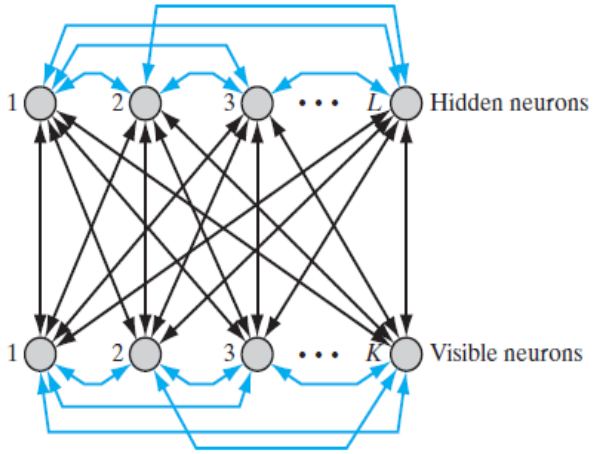


Figure 2.15: Boltzmann Machine

Boltzmann learning is very similar to error correction learning procedure. Instead of computing error between desired and actual outputs, Boltzmann machines consider the difference of error of the correlation between the outputs of two neurons under clamped and free running mode [38].

2.7.4 Competitive learning

In the competitive learning output units compete with each other for activation. At the end, neuron with the greatest total input wins the competition and turns on while all the other neurons turned off. So that at any instance only one output neuron is activated. This is called the '*winner takes all'* strategy.

Competitive learning clusters input data. By using the correlation of data, it automatically groups the similar patterns and represents them by a single neuron. Each output unit $x_j\ ; j = 1, \dots, n$ connects to all the input units x_i including x_j itself and connection weight between x_j and x_i is w_{ji}. According to the learning rule the neuron i^* with the largest input wins the competition, when $w_{i^*}x \geq w_i.x, \ \forall i$ [38].

2.8 Summary

This chapter briefly discussed the fundamental concepts of artificial neural networks. Feedforward networks are the most commonly used NNs. Due to its

inherent properties such as learn by experience and fault tolerance ANNs have been used to solve many real-world problems. The networks can train in supervised or unsupervised manner and backpropagation learning algorithm has widely used to train the multilayer feedforward networks. The next chapter will focus on the current approaches in modelling the hidden layer architecture and their strengths and limitations.

CHAPTER 3

CHALLENGES IN DESIGNING OF NEURAL NETWORKS

3.1 Introduction

The previous chapter discussed the fundamentals of artificial neural networks. Further it pointed out how ANNs learn with different types of learning rules. The current approaches related to our research are reviewed in this chapter. Firstly, it briefly discusses about the problem on designing the optimal architecture. Next, focus on the various approaches in modelling the hidden layer architecture in ANNs with their adopted technologies, strengths and weakness of each method. Finally, a summary will be given on the available researches, which helped to enhance our research work on this context.

3.2 The Problem of Designing the Optimal Architecture in ANN

The determining the hidden layer architecture is crucial in ANNs as some inappropriate architectures increase the training time, indicate poor generalization and cause non-convergence [60], [61]. It is known that, networks with higher number of hidden layers give more generalized solution. However, this architecture may not be very economical. Optimal neural network architecture reduces the computational complexity whilst improving the generalization ability. Therefore, before neural network employs its structure must be known. Determining the minimal network architecture is known as a difficult task and hence, often it comes down as a trial an error work [62]. To address this problem, researchers have applied different types of neural networks structures such as feedforward neural networks, recurrent neural networks and radial basis function etc., for various types of applications.

Because of its flexibility, good representational capabilities, and availability of large number of training algorithms, the feedforward neural networks are the most common and widely used network architectures [63]. It is known that feedforward

networks with large number of hidden neurons are able to learn fast by avoiding local minima. Also massively parallel networks form more complex decision regions [4] and they exhibit certain degrees of faulty tolerance under damage conditions [64], [65]. Nevertheless, when there are too many hidden neurons, hidden layers, and connections data may over-fit and show poor generalization. In addition, if the network is too large, it bears more nodes, more hidden layers and connections than required thus, it yields the unnecessary computational cost and arithmetic computations. In contrast, when the network is too small, it saves expensive hardware implementation time, but may not be able to learn the input/output relationships properly. Hence, design the optimal architecture i.e., the architecture which is large enough to learn maximum data and small enough to perform good generalization, is very important and remains as a research challenge. There are a certain number of approaches to overcome this problem. In general, hidden layer architecture of neural networks is determined by various pruning, constructive, pruning-constructive and some evolutionary techniques.

A one of the most common methods to approach the minimal architecture is by pruning unimportant hidden neurons [12]. This process is starting with a network larger than needed and trim down to the optimal solution [15], [19], [62],. The second approach is reaching to optimal solution by a constructive algorithm, where optimal architecture obtains by adding neurons and connection weights to a minimal network until the acceptable approximation accuracy is achieved [13], [14], [66]. There are number of hybrid methods of pruning and constructive algorithms to achieve the hidden layer architecture [67], [68]. These algorithms decide the neurons to be added and then gradually remove unnecessary neurons from the network. In addition, some evolutionary techniques have been used in optimizing the hidden layer architecture [69],[70].

3.3 Pruning Algorithms

Most of the pruning methods approach the solution by a brute-force algorithm, which systematically searches all the possibilities for the solution and checking

whether all the answers satisfy the required error conditions. When the output error is too large, either weights are updated or remove the corresponding neurons from the system. While the signal is forwarded through each layer, it takes $O(W)$ time, where W is the number of weights. For N training patterns, the total time taken to pass the signal is $O(NW^2)$. Thus, this process is very slow and time consuming as many neurons have to be trained [15], [71].

Most of the available pruning algorithms belong to two broad classes, sensitivity methods and penalty term methods. The first class estimates the sensitivity of the error function of the removable neurons where, sensitivity factor is defined as the derivative of the output error with respect to the connection weight [16], [62]. Thus, the nodes with least effect are identified as removable nodes and remove them from the network. The penalty term methods add terms to objective function (Activation function) that tends network to an efficient solution. The adding term proportional to the sum of all the weight magnitude [15]. Apart of those, magnitude-based pruning (MBP) [18], [72] evolutionary pruning methods [12] have been used to eliminate irrelevant neurons. MBP is the deletion of the connection weights with less saliency. It assumes that deletion of weights with less salience will cause only minor effect on the performance of the entire result. In most of the methods, backpropagation algorithm is used to train the network and the end of the training for reasonable error, the connection weights with the smallest magnitude are removed and the resultant network is trained until it tends to the desired error.

3.3.1 Sensitivity calculation method

Mozer and Simolensky [73] proposed a method to automatically trim least relevance units and construct a skeleton version of the network. The relevance ρ_i of unit i is measured based on the error (E) of the linear function

$$E = \sum_p \sum_j \left| t_{pj} - o_{pj} \right|, \tag{3.1}$$

where p is an index over patterns j over output units. t_{pj} and o_{pj} are target and actual outputs respectively, and ρ_i defined as

$$\rho_i = E_{without\ unit\ i} - E_{with\ unit\ i} \tag{3.2}$$

Where E is the error of the network on the training set. Before calculating this, it introduces activity of unit j, by using additional strength of α_i , which is considered as gating of activity of the unit

$$o_j = f\left(\sum_i w_{ji}\, \alpha_i o_i\right), \tag{3.3}$$

Where w_{ji} is the connection strength from i to j and f is the sigmoid squashing function. When α_i is zero, the unit i has no influence on the rest of the network. If α_i=1, unit i is a conventional unit. Thus, the relevance of the unit i is rewritten as

$$\rho_i = E_{\alpha_i=0} - E_{\alpha_i=1} \tag{3.4}$$

This is approximated by the derivative

$$\hat{\rho}_i = -\frac{\partial E}{\partial \alpha_i} \tag{3.5}$$

This derivative can be obtained by error propagation [74] procedure which is very similar to backpropagation method. When $\hat{\rho}_i$ comes down to a certain treshold value, the unit can be deleted. In this study authors claim that $\partial E/\partial w$ fluctuates strongly in time, thus exponentially decaying time average of the derivative, to minimize the fluctuations the following formula.

$$\hat{\rho}_i(t+1) = 0.8\hat{\rho}_i(t) + .2\frac{\partial E(t)}{\partial \alpha_i} \tag{3.6}$$

End of the pruning it observed that the pruned system also performs in the same efficiency even though there are less parameters. However, learning process of this network is considerably slow and hence, hard to apply for large sized networks.

The optimal brain damage (OBD) [16] and the optimal brain surgeon (OBS) [19] are the most popular sensitivity based pruning algorithms. To determine the unimportant neurons OBD measures '*the saliency*' of neurons by using the second derivative of the error with respect to the connection weights. When the objective function E is approximated by Tailor's series and the weight W is perturbed, change in the objective function is given by

$$\delta E = \sum_i g_i \, \delta w_i + \frac{1}{2} \sum_i h_{ij} \, \delta w_i^2 + \frac{1}{2} \sum_{i \neq j} h_{ij} \, \delta w_i \delta w_j + O(\|\delta W\|^3), \tag{3.7}$$

where g_i's are the components of gradient of E. That is

$$g_i = \frac{\partial E}{\partial w_i} \tag{3.8}$$

δw_i's are the components of the weight correction δW and h_{ij}'s are the elements of the Hessian matrix H [75].

$$h_{ii} = \frac{\partial^2 E}{\partial w_i \partial w_j} \tag{3.9}$$

The main objective of the algorithm is to find parameters whose removal will cause to minimize the error E. When the network is large Hessian matrix becomes enormous. Hence, authors assume that the matrix is diagonal and so that the cross terms are ignored. Therefore, the third term of equation 3.1 is eliminated. Also, pruning is done on a well-trained network in order to obtain minimum E. Hence, the first term of the equation is zero. This leads to

$$\delta E = \frac{1}{2} \sum_i h_{ij} \, \delta w_i^2 \tag{3.10}$$

Then define salience as

$$s_i = \frac{h_{ii} w_i^2}{2} \tag{3.11}$$

and remove low salience parameters from the network.

Authors in [76] presents a quantitative results on the performance of OBD and highlighted that certainly OBD increases the learning performance and improves the

generalization ability of the network. However, in practice Hessian matrix is not diagonal for all the instances. However, by proposing “Optimal Brain Surgeon” (OBD) Hassabi et al. [19], [77] argue that the Hessian matrix is strongly non diagonal for all the instances and hence, it may eliminate incorrect weights. Although, OBD and OBS derived by the same theoretical approach, based on the second order derivatives, OBS is more complex than OBD as it does not make any assumption on Hessian matrix [78]. Authors argue that OBS is significantly better than other magnitude-based pruning algorithms. However, this approach is quite slow and requires much memory and yields much complex computations especially when it deals with the inverse of the Hessian matrix.

G. Castellano et al. [12] proposed a method to iteratively prune hidden neurons from a feed-forward neural network. The proposed method solves a linear system in the least square sense using pre-conditioned conjugate gradient procedure. Authors claim that the algorithm formulates the pruning problem in terms of defining a system of linear equations in a very efficient conjugate gradient least square procedure and removes weights by preserving overall network behavior.

They claim that this algorithm shows number of new features as follows.

- It does not make use of any working parameters as some other algorithms describe in the literatures [72] , [79] and [80] . Hence, it does not require the tuning phase.
- It does not require any training phase after pruning as in ‘Optimal Brain Surgeon’ and it requires far less computations.
- Algorithm can be applied to any arbitrary topology and eliminates hidden neurons as well as weight connections.

This algorithm has similarities with some other algorithms such as the method proposed by Sietsma and Dow [72] to remove redundant hidden neurons and adjust remaining weights. Nevertheless, authors reach to the goal by solving a linear system without considering the redundancy of individual neurons. In addition,

network designer enables to monitor the behavior of network pruning and hence, its own stopping criteria can be defined.

However, the proposed method has been applied for very small architectures with one hidden layer. The maximum number of hidden neurons is less than 5. Hence, there is a possibility to change these results for topologies with large number of hidden neurons. On the other hand, when the network is large, matrix on the system of linear equations may have deficiency rank and hence, infinite number of solution may occur [8].

Apart of the above approaches, Lauret et al. [81] proposed a technique to prune a single hidden layer network based on extended Fourier amplitude sensitivity test [82] (EFAST). Although, this method shows some acceptable performance in to smaller sized networks, there are limitations to extend the criteria for multilayered architectures. Zeng and Yeung [83] presents a pruning method with help of quantified sensitivity measure. The method removes neurons with least relevance from hidden layers of multilayered perceptron. The concept is based on that the less relevance neuron's contribution to the network is negligible and thus, removal of least relevance neuron does not degrade the performance. This method discusses only neuron pruning and there is no theory to determine the number of hidden layers.

Sabo and Yu [62] presents an algorithm called hybrid sensitivity analysis with re-pruning (HSAR) by combining the advantages of local sensitivity analysis, local variance sensitivity analysis [84] and cross validation punning method [85]. The sensitivity is estimated by

$$S_{j,i} = \sum_{t=0}^{T-1} \left[\Delta w_{j,i}(t)\right]^2 \frac{w_{j,i}^f}{\eta\left(w_{j,i}^f - w_{j,i}^t\right)}, \tag{3.12}$$

where T is the total number of iterations needed to minimize the objective function and η is the learning rate. $\Delta w_{j,i}$ is the weight correction of $w_{j,i}$. $w_{j,i}^f$ and $w_{j,i}^t$ are the final value and the t^{th} iteration of weight $w_{i,j}$. Then it Computes the value of local

parameter variance nullity (LPVN) and prunes the neurons with parameter whose LPVN is less than a certain a threshold value. The pruned network is retrained and its performance is evaluated. If the pruned network shows better performance, continue process for a smaller network until there is no parameter to prune. If there is no any improvement, continue pruning process with the old network.

The main drawback of this approach is that the network processes with only weight connection pruning. The neuron pruning is not considered. However, if all the connections are possible to prune, neuron will be removed automatically. But practically, this needs many iterations and hence, this solution is not very feasible.

Suzuki et al. [86] discussed on synthesizing filters using a multilayer neural networks. This approach reduces both neurons and hidden layers based on the network error. After removing neurons, network retrain to repair the damage occurred while reducing. Fnaiech et al [87] have been approached to hidden layer architecture in feedforward neural networks based on that the feedforward neural networks could be represented by Volterra series [88], [89] such as a input/output model. It approaches the minimal architecture in 3 steps. Firstly, create a nonlinear activation function of hidden neurons as Taylor's expansion, then express the network output as a NARX (nonlinear auto regressive with exogenous input) model and finally, use the existing algorithm for nonlinear order selection. This literature selects the most relevant signal of the NARX and uses backpropagation algorithm to prune the hidden nodes.

In [90] authors had been approached to hidden layer architecture in Madeline [91] by using a sensitivity-based algorithm. Adeline (adaptive linear neuron) is a single layer artificial neural network (Figure 3.1) whose activation function is hard limit function which, gives two output values +1 and -1. Madeline consists of many Adaline's arranged in a multilayer net. The difference between standard neural networks and Adaline is in the standard neural networks weighted sum is passed to the activation function and the weights are adjusted by using function's output. In

the Adaline the weights in the learning phase are adjusted according to the weighted sum of the inputs [92].

According to the definition, sensitivity is to analyze the dependency relationship between output variation of the network and its parameter disturbance. Although, many existing approaches on sensitivity based pruning algorithms consider weight disturbance, authors in [90] discuss the sensitivity computation with architecture variation. In this method, firstly train the network by using Sensitivity-Based Adaptive Learning Rules (SBALR) [93] and prune Adaline by using the formula until it reaches to target architecture.

This approach proves that for some instance, multilayer architectures perform with better generalization and it can reduce to target network by pruning Adeline. However, it has not discussed the argument of deciding the target architecture and evidence to accept it as the most appropriate solution.

Augasta et al. [64] presented a pruning method called Neural Network Pruning by Significance (N2PS) by combining the advantages of both Variance Nullity Pruning (VNP) [94] and the Xing-Hu method on construction multilayer perceptron using information theory [95]. Although it performs well in single hidden layer networks, the method has not discussed implementation on multilayered structures.

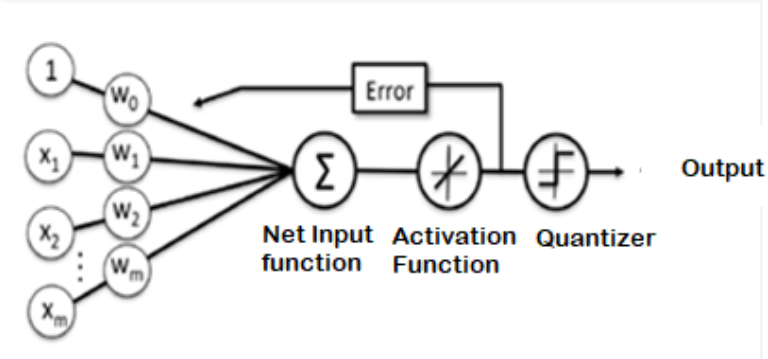


Figure 3.1: Adaptive linear neuron

Xu and Ho [96] have proposed an algorithm called subset-based training and pruning on the node dependent and Jacobean rank deficiency. At each iteration, it identifies the dependent nodes by applying column permutation to the output nodes. Then makes weights on output nodes to zero and only independent nodes train by Levenberg–Marquardt (LM) algorithm [97]. End of the training, a unit-based optimal brain surgeon pruning method applies to remove the insensitive hidden neurons and reduce the size of the network. There are several advantages of this method such as due to subset-based training and pruning method, only a subset of independent hidden nodes is trained. This is time saving and saves computational cost on training excess data. Secondly, an extra term is not added to the cost function and hence, there is no lengthy tuning phase. In addition, re-training is no need after pruning. At the end, authors claim that their proposal could be applied to multilayer perceptron. However, it does not clearly explain the way of determining the correct number of hidden layers in the optimum architecture.

3.3.2 Penalty methods

The Penalty methods reach the solution by adding a penalty term to the cost function to minimize the weights. So that, small weights ultimately become zero. Therefore, weights are removed from the network when they reach to a certain threshold.

Yves Chauvin [98] introduces a cost function

$$C = \mu_{er} \sum_{j}^{P} \sum_{i}^{O} \left(d_{ij} - o_{ij}\right)^2 + \mu_{en} \sum_{j}^{P} \sum_{i}^{H} e\left(o_{ij}^2\right) \tag{3.13}$$

where e is a positive monotonic function. Sums take over output units O and set of hidden units H. Number of patterns considered is P. the first term of this cost function is a standard error function in backpropagation algorithm. The second term is called the energy function, which measures the average energy spends by the set of hidden neurons. The parameters μ_{er} and μ_{en} are used to balance two terms. The minimum of the above function obtains when desired and actual outputs ($d_{ij}$and o_{ij} respectively) are same and when energy of hidden units is zero.

In [99], ji et al. present a method by adding two terms to backpropagation learning rule. The first term removes all possible nodes form the network while maintaining the acceptable level of error on the output layer. The other term creates to minimize the weights as much as possible. The process starts by training a feedforward network having one input unit, a single layer of N hidden sigmoidal units, which is larger than the necessary and one linear output. The M data of training set are given by $\{(x^{\pi}, y^{\pi});\ \pi = 1, \dots, M\}$, where the desired output of x^{π} is y^{π}. The output of the network is given by

$$g(x: \mathbf{w}, \boldsymbol{\theta}) = \sum_{i=1}^{n} v_i\, f(u_i x - \theta_i). \tag{3.14}$$

Where u_i and v_i are the input and output weights of the i^{th} hidden unit respectively. θ_i is the corresponding threshold value. $\mathbf{w} = (u_1, \dots, u_N, v_1, \dots, v_N)^t$, $\boldsymbol{\theta} = (\theta_1, \dots, \theta_N)^t$ and $f(x) = {}^{1}/_{1 + e^{-x}}$.

Then the standard error function given by Rumelhart et al. [30] can be written as

$$\varepsilon_0(\mathbf{w}, \boldsymbol{\theta}) = \sum_{\pi=1}^{M} [g(x^{\pi}; \mathbf{w}, \boldsymbol{\theta}) - y^{\pi}]^2. \tag{3.15}$$

A hidden unit defined as significant if it connects to both input and output units with weights greater than one and quantity of the significance is given as

$$S_i = \sigma(u_i)\sigma(v_i), \tag{3.16}$$

where

$$\sigma(w) = \frac{w^2}{1 + w^2}. \tag{3.17}$$

Then error $\varepsilon(\mathbf{w}, \boldsymbol{\theta})$ obtains by adding a term to $\varepsilon_0(\mathbf{w}, \boldsymbol{\theta})$ as follows:

$$\varepsilon(\mathbf{w}, \boldsymbol{\theta}) = \eta\varepsilon_0(\mathbf{w}, \boldsymbol{\theta}) + \lambda\varepsilon_1(\mathbf{w}), \tag{3.18}$$

where η and λ are corresponding learning rates and

$$\varepsilon_1(\mathbf{w}) = \sum_{i=1}^{N} \sum_{j=1}^{i-1} S_i\, S_j \tag{3.19}$$

After applying gradient decent learning rule, weights and threshold values are updated as

$$w_i^{n+1} = w_i^n - \eta \frac{\partial \varepsilon_0}{\partial w_i}(\mathbf{w}^n, \boldsymbol{\theta}^n) - \lambda \frac{\partial \varepsilon_1}{\partial w_i}(\mathbf{w}^n) \tag{3.20}$$

$$\theta_i^{n+1} = \theta_i^n - \eta \frac{\partial \varepsilon_0}{\partial \theta_i}(\mathbf{w}^n, \boldsymbol{\theta}^n) \tag{3.21}$$

Here

$$\frac{\partial \varepsilon_1}{\partial w_i}(\mathbf{w}^n) \propto \frac{2w_i}{(1+w_i^2)^2}$$

Therefore, $\partial \varepsilon_1(\mathbf{w}^n) \Big/ \partial w_i$ becomes zero for large $|w_i|$ and hence, dominant weights will stable. Somehow, conflict between two gradiant terms of the above equation may disturb the stability and hence, suggested to add the last term after the network has learned well. Then, λ is modified as

$$\lambda = \lambda(\varepsilon_0) = \lambda_0 e^{-\beta \varepsilon_0}, \tag{3.22}$$

where β^{-1} is defined as a characteristic standard error. Then the second modification of weights and threshold value are given by the following equations that reduce the larger weights from the network.

$$w_i^{n+1} = w_i^n - \eta \frac{\partial \varepsilon_0}{\partial w_i}(\mathbf{w}^n, \boldsymbol{\theta}^n) - \lambda \frac{\partial \varepsilon_1}{\partial w_i}(\mathbf{w}^n) - \mu \tanh(w_i^n) \tag{3.23}$$

$$\theta_i^{n+1} = \theta_i^n - \eta \frac{\partial \varepsilon_0}{\partial \theta_i}(\mathbf{w}^n, \boldsymbol{\theta}^n) - \mu \tanh(\theta_i^n) \tag{3.24}$$

It takes the term μ as

$$\mu = \mu_0 |\varepsilon_0(\mathbf{w}^n, \boldsymbol{\theta}^n) - \varepsilon_0(\mathbf{w}^{n-1}, \boldsymbol{\theta}^{n-1})| \tag{3.25}$$

Once the performance reaches to an acceptable level, nodes whose weights are smaller than certain level will remove from the network and hence, the network becomes simpler. Nevertheless, it has noted that modified error function increases

the training time. Moreover, the method is built for single hidden layer networks and needs some modifications to extend for multilayers neural networks.

Rudy Setiono [17] has presented a penalty function

$$P(w,v) = \epsilon_1 \sum_{m=1}^{h} \left(\sum_{l=1}^{n} \frac{\beta w_{ml}^2}{1+\beta w_{ml}^2} + \sum_{p=1}^{C} \frac{\beta v_{pm}^2}{1+\beta w_{pm}^2} \right) + \epsilon_2 \sum_{m=1}^{h} \left(\sum_{l=1}^{n} w_{ml}^2 + \sum_{p=1}^{C} v_{pm}^2 \right) \qquad (3.26)$$

by combining the works in [100] and [101]. Where ϵ_1 and ϵ_2 are small weight decay constants. C is the number of output units. h is the number of hidden units in the network and n represents the number of inputs. w_{ml} is the weight from l^{th} input to m^{th} hidden unit and v_{pm} is the weight from m^{th} hidden unit to p^{th} output neuron. Author of this paper claims that first part of the function controls having unnecessary connections while the second term prevents on getting large amount for these weights. However, there is no guarantee that this function is not removing weights which are significant. The eliminating such weights might cause for local minima on the error surface.

All the above methods have significantly contributed to develop the techniques in modelling of hidden layer architecture of ANNs. However, each method has its own advantages and drawbacks. Some techniques remove even important nodes while they remove the irrelevant ones. For example, MBP methods eliminate some relevant neurons as they remove all the neurons with small magnitudes. Another major barrier of many methods is their low efficiency. In case of OBD and OBS, although they remove cluster of neurons at once, they are not computationally economic when the networks are large. The sensitive analysis methods are based on the assumption that both the inputs of the network and output of hidden neurons are no mutually independent. So that they are not guaranteed to remove all redundant processing elements [65].

3.4 Constructive Methods

A Constructive neural network starts with a minimal network architecture and dynamically increases the network by adding hidden layers, hidden neurons and connection weights while training until the satisfactory solution. Constructive learning algorithms alter the network architectures as learning proceeds, producing a network with the proper size [102]. These algorithms extend the searching for solution to whole possible structure. Firstly, they search for a simple solution and extend it for near minimal architecture which exactly suits for the given task. Once it successful, the algorithm can extend to estimate the solutions for more complex practical problems. Different constructive algorithms can be used to manage learning measures such as training time, network size and accuracy [103].

The main advantage of constructive algorithms is they are easy to initialize. But in pruning methods it is not known initially how large network should be taken. In addition, constructive algorithms are more computationally economical as they train small networks and the suitable network is chosen during the training [104]. Another advantage of constructive algorithm is, since constructive algorithms give smallest possible network it reduced the training time to find optimum network which gives the best generalization [105]. There are many constructive neural networks and some of them are listed below.

3.4.1 Cascade correlation algorithm

The most well-known and widely used constructive algorithm is Cascade-correlation algorithm which published by Scott Fahlman and Christina Lebiere [13] in 1990. Cascade correlation is designed by combining two key ideas, namely ‘cascade architecture’ and ‘learning algorithm’. Cascade architecture is referred that, adding hidden neurons one at a time to the network and they do not change once they have been added. Learning algorithm creates and installs new hidden units and for each hidden unit it tries to maximize the magnitude of the correlation between the output of the hidden neuron and residual error signal.

The method starts with a minimal network and automatically trains and adds new hidden units one by one and builds a multi-layer structure. Once a new node is added to the network, it becomes a permanent feature-detector by freezing the input weights and then its output weights are able to create a more complex feature detector. The main objective of this algorithm is to attempt to solve several problems and limitations on the backpropagation learning algorithm as they observed that the way of the training network by a backpropagation algorithm is slow. The authors claim that two major problems cause of the slowness of the backpropagation algorithm, namely ‘step size problem’ and the ‘moving target problem’.

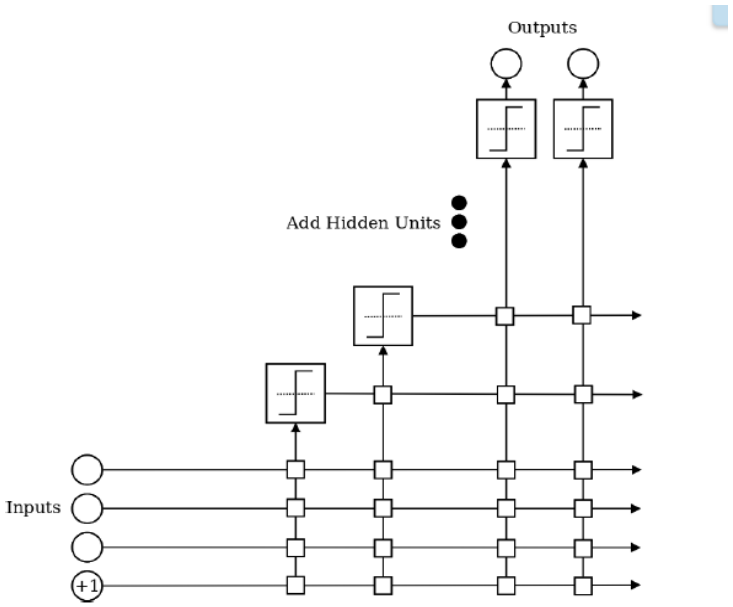


Figure 3.2: Structure of the cascade algorithm

In backpropagation algorithm, the first partial derivative $\partial E/\partial w$ in each weight computes to minimize the error at every step. So that choosing infinitesimal step size may cause to local minima and too large step size will not converge to a good solution. To determine the reasonable step size, sufficient information such as curvature of the step function, the vicinity of the current point must be known. But these are not available in the standard backpropagation algorithm. As the second weakness of the backpropagation, the authors show that each neuron in the network

is trying to change its feature in every iteration. This process is highly complicated as the hidden units in the given layer cannot communicate on another directly. So that it must allow only a few of the weights in the network to evolve at once while keeping others constant. However, the cascade correlation algorithm decides to have an extreme version of this technique such as, only one unit changes its features at any given time. The model of the network of the cascade-correlation algorithm is very similar to the pyramid structure which describes below. The main difference is instead of output layer cascade architecture adds new nodes to hidden layers as shown in the above Figure 3.2 [106].

3.4.2 Dynamic node creation algorithm

The dynamic node creation (DNC) algorithm [14] is supposed to be the first constructive algorithm for designing single layer feedforward networks dynamically and there after many other constructive algorithms such as [107],[108] and [109] were derived of the technique of this algorithm. Here a single hidden unit is added to the same hidden layer, one at a time and whole network trains after adding each unit. This method is simple and convergent for a small sized architecture. However, some computational limitations arise when the network is large.

3.4.3 Tiling algorithm

The tiling algorithm is proposed by Mark Mézard and Jean-Pierre Nadal [110], which constructs a layered network where neurons are added like tiles when they are needed. Each layer has two types of units as shown in the Figure 3.3; the first type has major role is called the master unit and all the other units in the layer are ancillary units. The certain master neuron classifies more patterns than the master neuron of the previous layer and the process continues until the master unit of a certain level is classifying all the training examples. Otherwise, add ancillary units, which are known as tiles, until the layer becomes faithful. Then use the pocket algorithm with ratchet [111] to train new ancillary units and a new layer [106].

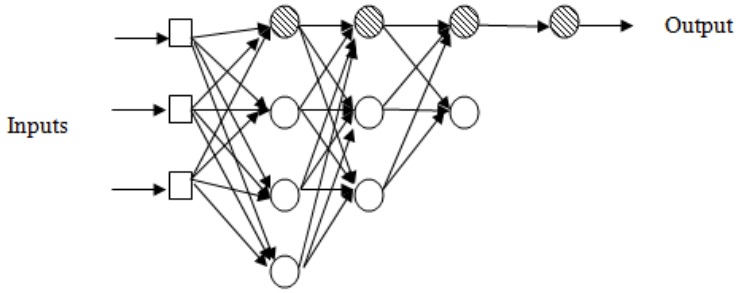


Figure 3.3: Tilling Algorithm

3.4.4 Tower algorithm

The tower algorithm, which firstly presented by Stephen Gallent [111] builds a tower of threshold logic units by making single-cell learning. As shown in the following Figure 3.4, the bottom most neuron receives signals from original inputs and the neuron immediately below it. Each neuron is trained separately by using the pocket algorithm with ratchet and after trained, the connection weights of the particular neurons are fixed. Then this output and the neurons in the input layer become inputs to the next neuron in the tower and process continues until the desired classification accuracy is obtained [106].

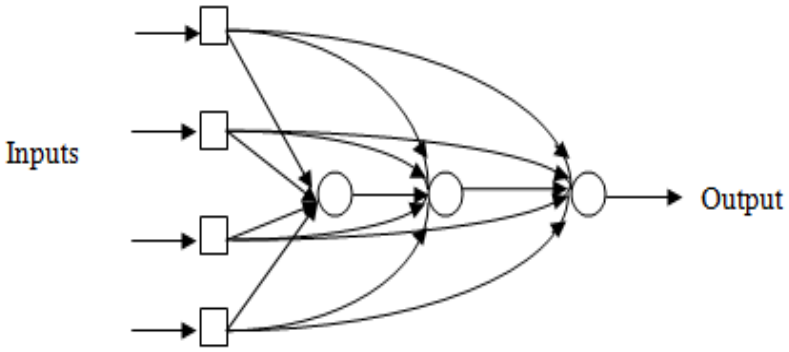


Figure 3.4: Tower Algorithm

3.4.5 Pyramid algorithm

The pyramid algorithm also created by Stephen Gallent [111] and it is very similar to the above tower algorithm. The difference of this algorithm is the new neuron is connected to all the previously trained neurons as depicts in the Figure 3.5. The pyramid algorithm generates single cell model by using the pocket algorithm and train it by pocket algorithm with ratchet. If the network shows better performance freeze the weights of newly added neuron and add another neuron to the structure.

Apart the above, Friedman [112] has proposed a constructive technique projection pursuit regression (PPR), which is similar to DNC. Also [113],[114],[115], [116] and [117] are presented constructive methods inspired by PPR. Unlike DNC, these algorithms add more complication functional form hidden units and instead of retraining whole network, train only the newly added hidden units. Rivals and Personnaz [118] discussed a model selection procedure based on the least square estimation and statistical tests to determine the optimal number of neurons in one hidden layer networks. The complete procedure has two phases. The first phase based on bottom-up strategy, increase the number of neurons and extend up to its Jacobean matrix is sufficiently well conditioned. The second phase use top-down strategy with statistical Fisher test to refine the selection.

Generally, in constructive algorithms, initial network is simple. Thus, they are computationally economical and they can train faster than large networks. However, iteratively train few neurons, until the optimal solution is obtained is time consuming. Not only that, but also as initially there are small number of neurons, they are more sensitive to initial conditions and other training parameters [119]. So that, there is high probability to trap with local minima.

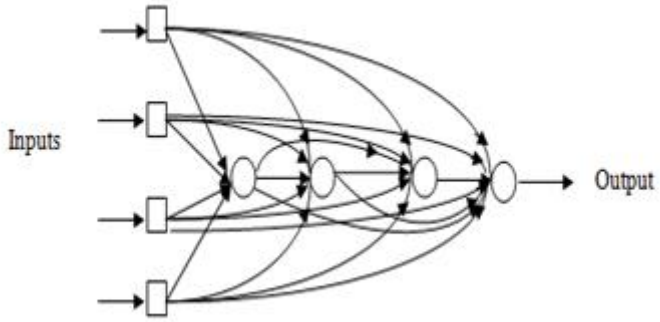


Figure 3.5: Pyramid Algorithm

3.5 Evolutionary Methods

Liu et al. [1] proposed a method to approach the minimal hidden neurons based on the estimation of the signal-to–noise-ratio figure (SNRF). In this study they suggest that as SNRF can quantitatively measure the useful information in data that not learned, without validation set, over-fitting problem automatically detected. By using bench-mark data sets it shows that this method reduces the over-fitting problem in the network.

By creating network with two hidden layers Asthana et al. [120] proposed a method to recognize multi-script number on postcard. In this research they could achieve more than 95% accuracy. Authors claim that the best accuracy can be obtained by using the same number of neurons in both layers. Their work was tested in five different popular Indian scripts namely Hindi, Urdu, Tamil, English and Telugu and obtained 96% accuracy under ideal condition. Karsolia [121] shows that the accuracy of the performance improves when the number of layers increases up to 3 and claims that by increasing the number of layers training time and complexity increase many folds. Also, in his work, he has shown that by adding more hidden neurons unnecessarily network leads to an over-fitting problem.

In these problems, evaluations have been done only with few data sets, which is not sufficient to come with strong decisions. Author in [121], presented all the results

based on that three layer networks are the most suitable topologies without any theoretical background.

3.6 Summary

The main objective of this chapter was to discuss the past to present developments of the design of the artificial neural networks. There are number of approaches to overcome the problem on the modelling of hidden layers in ANNs. Most of them belong to either category of pruning, constructive or evolutionary method.

Various research models in each method were analyzed by explaining their mechanism, strengths and drawbacks. Most of the existing approaches have been used various techniques and achieved momentous solutions. However, there are several limitations on existing approaches. Some of them cause to huge complex computations. Several methods are time consuming and not based on a proper theoretical basis. Hence, none of the method achieves the solution of the problem of hidden layers in ANNs with good theoretical basis. Therefore, the problem on modelling the hidden layer architecture still remains as a research challenge. The next chapter will be discussed the theoretical background of modelling the hidden layer architecture.

CHAPTER 4

A THEORETICAL BASIS FOR MODELING HIDDEN LAYERS

4.1 Introduction

The previous chapter discussed the current technologies of modelling the hidden layer architecture of ANNs by highlighting their strengths and limitations. The artificial neural networks are built to resemble the functional behavior of the human brain. Of course, it is impossible to model all the functions of the human brain with ANNs, due to its complexity and activities, some parts of the brain are still mysterious. So, this research has been adopted few of the functions of the human brain to model the hidden layer architecture and this chapter will discuss its theoretical basis. Firstly it introduces the context of neuroplasticity. Then briefly discuss on the functional behavior of human brain followed by the different types of neuroplasticity and its various effects. Finally, it gives a concise mapping of the human brain and the artificial neural networks.

4.2 The History of the Neuroplasticity

One of the most important and fascinating properties of the human brain is its ability to adapt to the surrounding environment by changing its neural structure. Until recently, scientists and the philosophers in the field of neuroscience worked with the notion that the human brain is immutable and hard wired. It was postulated that no new neurons are born and functions of the brain structures are fixed [122]. The recent studies show that, these assumptions are no longer correct and brain functions change throughout one's life [123]. It is one of the important capabilities of the human brain to organize its structure and functions itself in order to provide an output for various inputs received from its surrounding environment [124]. This change of brain neurons and its pathways to adapt to the surrounding environment is called the neuroplasticity [125] and also referred as the brain plasticity [126], [127].

The word neuroplasticity is originated by two words 'neuron' and 'plastic'. A neuron stands for the nerve cell, which is formed by an axon and dendrites. Two

neurons are combined by a small space known as synapses (Figure 4.1). The term plasticity refers the native property of nervous system that alters the structure of neurons and synapses to maximize the brain functions and to adapt the changes of environment. In other words plasticity means the development of the system to react or adjust to both internal and external environmental situations under certain conditions [128]. Hence, neuroplasticity stands for the ability of the brain to change by altering its neurons and pathways to adjust to the surrounding environment.

The basic concept of neuroplasticity arose in the 19th century. But it was not emerged in the field and highly neglected by the scientists because the concept of '*one function one location*' was dominated in the field of neuroscience [129]. At that time, it was assumed that the brain is static. It believed that if a part of brain devoted for some function was damaged, then it cannot be recovered. However, with the interest of neuroscientists, this concept emerged in the society. In 1887 Spanish pathologist and neuroscientist Santiago Ramon Cajol proposed that the plasticity occurs in the nervous system and he published his important article, focus on the nervous system of lower animals, *"Estructura de los Centros Nerviosos de las Aves"*, (Structure of the Nervous Centers in Birds) based on the practical analysis and claim that a nerve

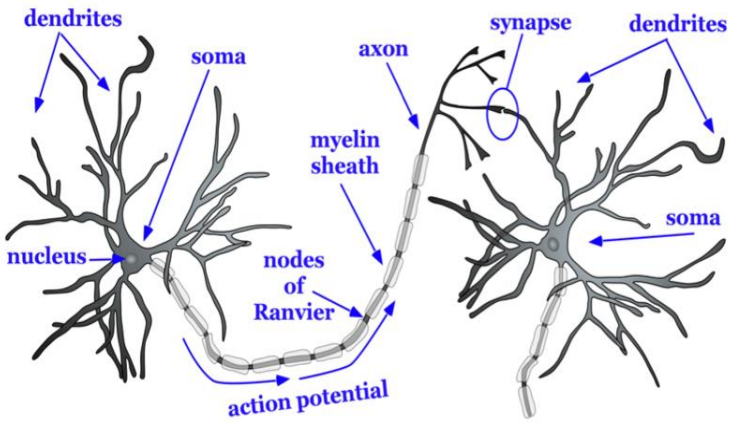


Figure 4.1: Structure of a biological nerve cell

cell is an absolutely autonomous physiological canton and dendrites and axons are end freely [130]. William James was the first to suggest theory of neuroplasticity in 1890, by his book '*Principles of Phycology*' [131]. In this book he has mentioned about the training cerebral hemisphere and used both physical and psychic factors to justify his idea.

For the first time, the term neuroplasticity was used by polish neuroscientist Jerzy Konorski [132] at the middle of 20^{th} century. At this stage it was assumed that the lower brain and neocortical areas are fixed after childhood and when people learn, neuronal changes occur only in the hippocampus in the adult's brain [133]. However, new findings show that all the parts of brain are plastic and claim that a large number of neurons are being added daily to the primate prefrontal and temporal lobes.

4.3 Types of Neuroplasticity

The neuroplasticity theory says that thinking, learning, meditating and some of the physical exercises such as dancing and playing musical instruments change the brain's physical structure and the functional abilities. These activities cause the plasticity of the human brain and which occurs in different ways. The main three types of neuroplasticity are activity dependent plasticity, competitive plasticity, and positive and negative plasticity.

4.3.1 Activity - dependent plasticity

The brain's ability of adapting to the environment yields to human specialized in special target by continuous training and involving different activities. For example, a child without born-talent in music is able to train to play a musical instrument by regular training. A person, who suffers with a voice problem as a result of a neurological disorder, will be able to talk by doing speech therapy exercises. In other words, by training the lost functions of the nervous system can be retrieved. This ability of the brain is known as activity-dependent plasticity. This concept

begins with American neuroscientist Paul Bach Y. Rita in late 1960s [134]. He designed several models and conducted experiments to show evidence to prove the hypotheses that the brain is capable of changing by different activities by including a vision substitution system for blind people.

Activity-dependent plasticity plays a significant role in learning and ability of understanding new things. It is responsible for helping to adapt an individual's brain, according to the relative amount of usage and functioning.

4.3.2 Competitive plasticity

The plasticity of the brain is able to change its function constantly. Each part of the brain has some specialized function. These functions change constantly when they activate to achieve human needs. This is called competitive plasticity. Neuroscientists' old saying "use it or lose it" is more suitable to describe the brain's functions than it is of any other part of the body. In other words, the human brain is the way that because humans need to be able to process certain information, various areas of the brain are specially adapted to processing different types of information. But at the same time any area of the brain is capable of processing almost any type of information. The idea, that we use only a small part of the brain, is simply wrong. Any part of the brain that is not being used, will tend to be taken over for the processing of other information. For example, blind people have a better sense of other parts like ears, fingertips, etc. People who have lost their arms are able to manage many of their works with legs which normally other cannot. Because, the neurons of the other parts tend to take and maximize their functions of inputs of the disable parts. This has been explained by famous Canadian psychiatrists and psychoanalyst Norman Doidge as follows [135].

"The competitive nature of plasticity affects us all. There is an endless war of nerves going on inside each of our brains. If we stop exercising our mental skills, we do not just forget them: the brain map space for those skills is turned over to the other skills we practice instead. If you ever ask yourself, 'How often must I practice

French, or guitar, or math to keep on top of it?' You are asking a question about competitive plasticity. You are asking how frequently you must practice one activity to make sure its brain map space is not lost to another."

4.3.3 Positive and negative plasticity

There are many factors affect on the alternations in the brain, such as behavior, environment, learning and mental and physical health conditions including injuries. Like all other changes, the brain's structural and functional changes also work in positive and negative ways on human beings. For example, if the individual's brain functions and neural pathways and neurochemistry associated with experiencing in pessimistic thoughts are strengthened then his neural pathways and neurochemistry associated with optimistic thoughts are weakened. And the other side of this also happens. Same goes to anxious versus relaxation, anger versus peacefulness and depression versus happiness etc.

Positive neuroplasticity is the physiological ability of the brain to form and strength dendritic connections, produce temporal changes, release neuromodulators and increase cognitive reserve. Neuromodulators is the physiological process, which a given neuron uses one or more neurotransmitters to regulate diverse populations of neurons. Various physical activities, learning, social interactions and cognitive remediation are some factors for positive plasticity.

In contrast, negative plasticity refers to the same physiological ability of the brain to atrophy and weaken dendritic connections produce detrimental morphological connection and decrease cognitive reserve. Negative plasticity may occur as the result of poor health conditions, lack of sleep, bad food habits, depression and some feeling like anxiety.

4.4 Structure of the Biological Neuron

A biological neuron has very specialized structure, as shown in the Figure 4.2, which includes dendrites and axons. As many other cells it contains a cell body. Two neurons connect to each other by synapses. More details on the neuronal structure are discussed below.

Cell Body

Neural cell body, also known as 'soma' is the spherical end of a neuron that contains the nucleus. Cell body connects to the dendrites and axon. All the protein for dendrites, axon and synaptic terminals are made in the cell body.

Axon

The lengthy fiber that transmits signals from cell body to terminals of the other neurons, muscles and glands, is known as the axon. In some sensory neurons, such as those for touch and warmth, axons transmit electrical impulse from the periphery to the cell body. The length of axon varies from 0.1 millimeter to 1 meter, and lengthy axons transmit the signal faster. Axons are covered by a fatty substance called myelin, which protects the axon and help to transmit the speed the transmission.

Dendrites

The dendrites are the treelike extension at beginning of neurons that play a key role in transmitting signals. Dendrites receive chemical signals from the axon of other cells and convert to electrical impulse and passes to the cell body or soma. Generally, dendrites are short, highly branched and covered with synapses.

Synapses

Each neuron ends with terminal buttons and the gap between terminal buttons are known as a synapse. Neural transmitters pass the electrical and chemical signals through the synapses as depicts in the Figure 4.2 below. Neurotransmitters are located in the vesicles in the terminal buttons. When an electrical signal reaches to

the terminal gap, it converts the electrical impulse to a chemical signal and passes through the synapse to other nerve cells [136], [137] .

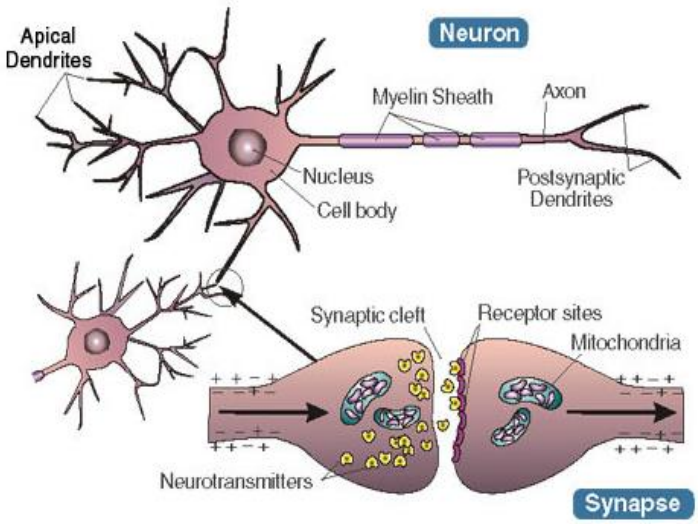


Figure 4.2: Structure of a neurons and a synapse

Synapses are the basic functional and structural units locate in between two neurons to transmit the signals. A neuron that transmits the electrical signal towards the synapse is called the pre-synaptic neuron and the neuron that conduct electrical signal away from the synapse is the post-synaptic neuron. Probably a neuron may have more than thousands of synaptic junctions and some neurons associate with the brain contains more than 100,000 synaptic contacts. The most common synapses in the human brain are chemical synapses. A chemical synapse converts a pre-synaptic electrical signal in to chemical signal and then to a post-synaptic electrical signal [138].

4.5 Neuronal Structure of the Human Brain

Neurons are the most important specialized cells in the nervous system that transmits the signals throughout the body. They are known as information-processing units in the brain, which responsible for receiving and transmitting information. Neurons act in different ways in sensing external and internal stimuli,

in transmitting the information and controlling the muscle actions. In the human brain, there are three main kinds of neurons.

1. **Sensory neurons**: These neurons are combined with receptors and convert external stimuli in to internal chemical impulse.

2. **Motor Neurons**: Motor neurons control the functions of muscles. They are responsible for all the activity such as movements, speech, etc.

3. **Interneurons**: In between sensory and motor neurons, interneurons are introduced. Interneurons are found in small scale in the central nervous system because they can find only the brain and the spinal cord but not in the peripheral system.

At birth, the infant brain consists of more than 80 billion of neurons and each neuron has about 7,500 connections. These neurons and synaptic connections grow rapidly until 2 years of age. Generally, the synaptic connections of the 2-year-old infant are double as that of an adult brain. While he grows the weaker neurons and synapses removed from the brain while strengthening the remaining.

4.6 The Anatomy of the Human Brain

The biological nervous system composed with central nervous system (CNS) and peripheral nervous system (PNS). The CNS consists of the brain and the spinal cord. The PNS consists of nerves and ganglia outside the brain and the spinal code.

The brain is the most amazing and complex organ in the human body. It is generally viewed as a black box that receives input signals from the environment and emits the corresponding response. As shown in the Figure 4.3, the brain consists of three regions called, cerebrum, cerebellum and brainstem. The most substantial part among them is the cerebrum, which is split longitudinally into two large hemispheres; the left hemisphere and the right hemisphere. The cerebrum has both

gray and white matter. The gray matter, which is the outermost layer of the cerebrum, is called the cerebral cortex. The cerebral is divided into four lobes; frontal, parietal, temporal and occipital. The neocortex is the largest part of the cerebral cortex and many functions such as intelligence, memory, creativity, emotions, touch, vision, hearing and speech, etc. are controlled by the neocortex. The functions of neocortex will be discussed in the next section.

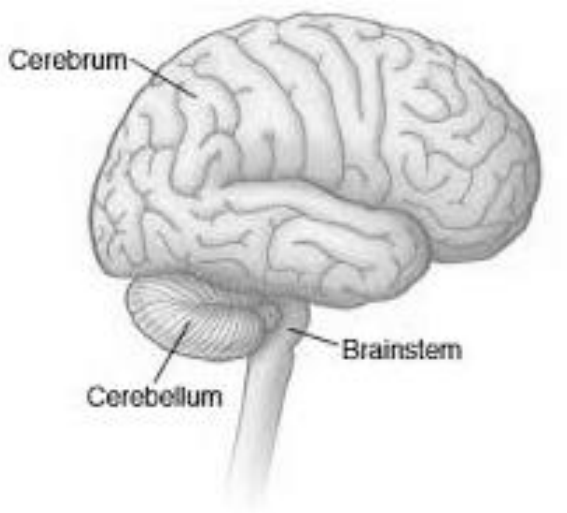


Figure 4.3: Structure of the brain

Under the cerebrum, cerebellum is located and it controls the movement of muscles, posture and balance of the human body. The brain stem connects cerebrum and cerebellum to the spinal cord. The three major parts of the brainstem are midbrain, pons and medulla. Many automatic events such as breathing, digesting, heart rate, waking and sleeping cycle are functioned by the brain stem. Usually, brain neurons are five to six times of magnitudes slower than the silicon logic gates. The reaction of silicon chips to an event is measured in nanoseconds while the reaction of brain neuron is measured in milliseconds. However, the brain gets its amazing features as it is made with massively interconnected neurons. It is shown that a typical brain consists of ten billion neurons and sixty trillion of synaptic connections [139]. Hence, the brain has become an enormously efficient structure and the energetic

efficiency of the brain is measured as 10^{-16} joules per operation per second and that is much greater than that of a modern computer.

In the brain anatomical organization can be found in both small scales and large scale. There different functions take place at lower levels and higher level. Figure 4.4 shows the hierarchy of those levels given in Haykin [33] and references therein.

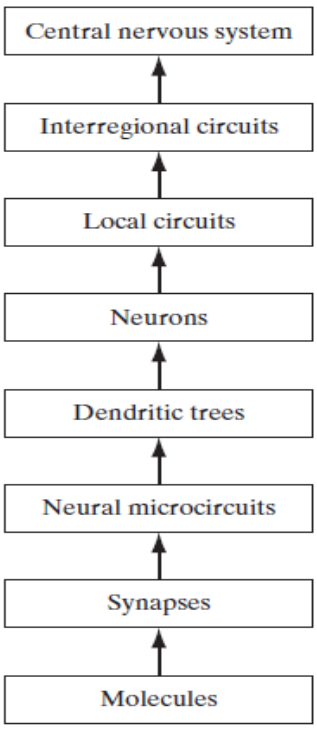


Figure 4.4: The structural organization of levels in the brain

In this structure, molecules are in the bottom level and synapses are depends on the actions of molecular ions. Neural microcircuits are in the next level. Neural Microcircuit refers the assembly of synapses organized into pattern of connectivity between neurons within the region. They get many different forms depending on the cellular and synaptic designs of microcircuits [140]. The size of microcircuit is measured in micrometers (μm) and their speed is measured in milliseconds.

In the next level of complexity we have dendrite trees and neurons. The *dendritic subunits* are formed by grouping the neural microunits with in the dendritic trees of

each neuron. The size of the whole neuron with several dendritic subunits is approximately 100 μm [33].

At the next level, there are local circuits which are formed by neurons with similar or different properties. Generally, size of a local circuit is about 1mm. Next we have interregional circuits, which made up of columns, pathways and topography maps. Interregional circuits involve in several regions located in all the sensory system of the brain. Topographic maps act as an information transfer from thalamic way stations to different areas of the brain [141]. Finally, at the top level of complexity we find the central nervous system.

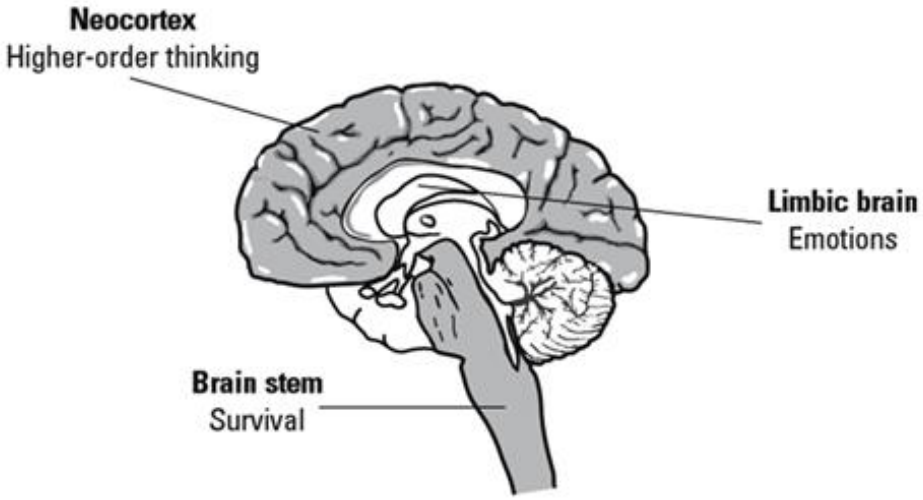


Figure 4.5: The hierarchy of the brain

4.7 Functions of the Neocortex

The neocortex, also referred as the isocortex is the largest part of the cerebral cortex (Figure 4.5). In the human brain, this is the part that involves in higher order brain functions such as cognition, rational thinking, planning and sensory perception etc. It is the outermost part of the cerebral hemisphere with thickness of 2-4 mm. The neocortex consists the 'grey matter' of the brain or neuronal cell bodies and unmyelinated fibers surrounding the 'deeper white matter' in the cerebrum [142].

It is interesting to observe that the layered structure of the neocortex. Generally, neocortex of the mammal's brain consists 6 layers and each layer has its own function different from others as depicts in the Figure 4.6.

Layer I, the outermost layer which contains only few inhibitory cells is called the molecular layer. Basically, this layer contains dendrites and axons of neurons from deeper layers. These axons and dendrites spread horizontally in this layer. In this layer, the intra-columnar axon are supposed to connect to the pyramidal cells [143].

Layer II contains both pyramidal cells and inhibitory cells. As this is one of the most outer layers, it is called the external granular cell layer. This layer contains dendrites whose cell bodies are in layers V and VI and the main function of these cells is receiving the input signals from other layers.

In layer III majority of the cells are pyramidal cell. However, it contains all most all the other variety of cells which can find in the other areas of neocortex. Since the majority of the cells of this layer are pyramidal, this layer is called the *external pyramidal cell layer*. This layer cells are responsible to receive signals from other cortical regions and transmit them to other cortical columns.

Layer IV, also known as *internal granule layer*, is composed with granule cells and this layer is located in the deeper in the neocortex. This layer contains some inhibitory cells and small excitatory cells known as spiny stellate cells. Layer IV is the main layer that receives signals coming from thalamus. The granule cells receive sensory input and transmit them to adjacent neocortex column.

Layer V composed with larger pyramidal cells and it contains only few inhibitory cells. Some pyramidal cells of this layer have long axons which transmit signals to basal ganglia, brain stem, and spinal cord [142]. The basal ganglia refer the group of subcortical neurons which are strongly interconnected with several areas of the

brain including cerebral cortex, thalamus, and brainstem. Basal ganglia are located at the base of forehead. Cells in layer V mainly involve in motor movements.

Layer VI is called the *multiform layer* as it contains many different cells in white matter. The structure of this function is not homogeneous. This layer also contains some inhibitory cells called *Martinotti* cells. Martinotti cells have long axonal outputs which project signal across all the layers of neocortex. Most of the cells in this layer are large pyramidal cells which project their axons to the thalamus. Other than layer IV, layer VI is the next main target of inputs of thalamus to the neocortex. Thus, the main function of layer VI neurons are receive and integrate the signal from brain stem and transmit them to the thalamus [144].

The six-layered structure of the neocortex is unique to the mammalian brain structure. Other than mammalians only fish and reptiles have neocortex and they have only three layered structure.

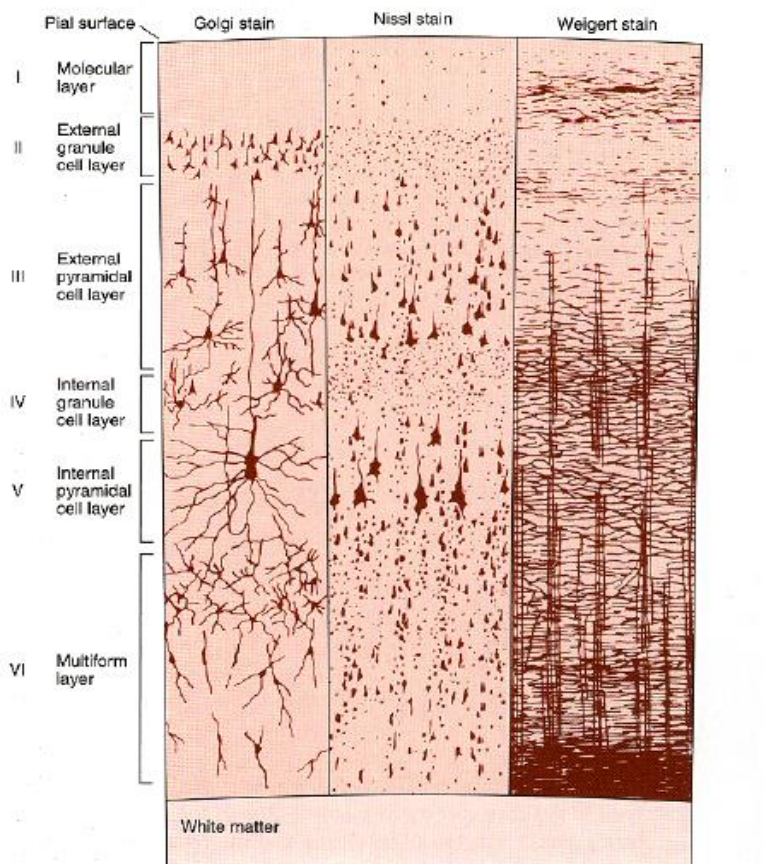


Figure 4.6: The layered structure of the neocortex

4.8 Classification of Effect of Neuroplasticity

The neuroplasticity is the general term to describe the changes of the brain. For further analysis, it is distinguished into two broad categories; structural neuroplasticity and functional neuroplasticity.

4.8.1 Structural changes in Human brain

The development of the human brain begins in the embryonic state and continue throughout the lifespan [145]. However, most of the dynamic changes of the brain occur in the early childhood, and it assumes that the brain reaches to 80% its weight during the first two years. The functional and structural changes in the human brain can be occurred due to the various types of behavior of neurons such as

neurogenesis, neural migration, neural cell death, synaptogenesis and synaptic pruning [146], [147].

4.8.1.1 Neurogenesis

Neurogenesis is the creating new neurons from the neural stem. There are number of behavioral and environmental factors affecting in neurogenesis and it caused to change of synapses and neural pathways. Usually, neurogenesis takes place in two regions of the brain called the sub ventricular zone (SVZ) and the hippocampus.

The hippocampus (Figure 4.7), where neurogenesis occurs throughout the life span of the brain plays a major role in learning thinking, and problem solving. Reducing this part has been found to have some neuro-degenerative diseases such as Parkinson's and cognitive disorders like depression, amnesia and dementia. The newly born cells in the SVZ form a lining of the lateral ventricles. The neurogenesis takes place in hippocampus forms the 'denate gyrus' which is the part of the hippocampus responsible for memories of events [148], [149]. The neurogenesis happens from the embryonic period to adulthood and it largely occurs in the developing brain. The recent studies show that there are evidences in neurogenesis happen in the adult brain, but it is limited to some parts of the brain. Joseph Altman [150] shows in his studies that adult neurogenesis takes place only in the hippocampus

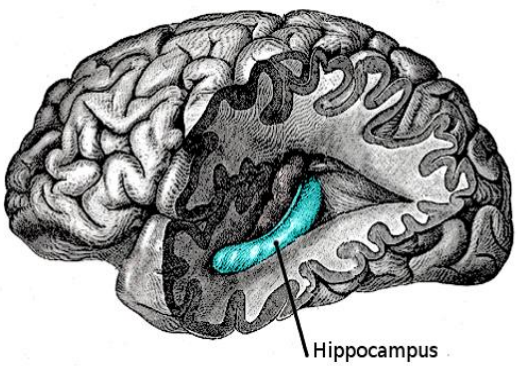


Figure 4.7: Hippocampus area of the brain

4.8.1.2 Neural Migration

The position of neurons in the central nerve system plays a key role in determining their functions. Most of the neurons in the human brain are not born in the same place as they are finally located. Neurons are generated by one part of the brain and sometimes travel long distances along complicated routes to reach their target locations. Especially, neurons in the peripheral nervous system come to their final location after having a long journey from the embryonic position. However, neurons in the central nervous system limit their movement to the neural tube. This process of moving neurons from its birth place to another location is called the neural migration. Due to neural migration, different classes of neurons locate together and hence, they can interact appropriately [151]. Although majority of neuronal migration takes place in all stages of embryonic development, few neurons continue the process until adulthood [152], [153].

The two major modes of neural migration are radial migration and tangential migration. The neurons have radial migration, originate in the ventricular zone of the pallium (cortex) and form 'glutamatergic pyramidal neurons'. The tangentially migrating neurons that originate in the ventricular subpallium form GABA ($\gamma -$ aminobutyric acid).

4.8.1.3 Neural Cell Death

At the developing period of the brain, about one and half time of the neurons that in the adult brain are created. The process of destroying such excess of neurons is known as neural cell death. Neuroscientists have been identified three different ways of neural cell death. Firstly, it occurs during the nerves system developing period. That is, during the embryonic and early postnatal period. A large percentage of neurons, approximately 50% in each region of nervous system die in this stage. The timing of the process may vary from region to region. But this process is normal and known as 'apoptosis.' The second way of neural cell death happens as the result of various neurodegenerative disorders like Alzheimer's disease. In this stage significant number of cells die, but the process continues several years. Hence,

the daily death rate of neurons is infinitesimal. Finally, cell death appears after hypoxia that accompanies strokes. In this case large number of neurons dies within short period and hence, it leads to very abnormal behavior in central nervous system [20].

4.8.2 Synaptic plasticity

Synaptic plasticity is the ability of changing activities of the synapses over time with the effect of the synaptic transmission. That is, enhances or depresses the synaptic transmission by activity [154]. Synaptic plasticity plays a key role in the early developing period. Generally, there are two types of synaptic plasticity called intrinsic and extrinsic. Intrinsic refers the changes of the strength of synapses due to its own activities while extrinsic is the changes of activities due the activity of others behavior. The changes of synapses may occur as short term or long term. The short-term process lasts from milliseconds to minutes. The long term synaptic plasticity, which is known as long term potentiation (LTP) or long-term depression (LTD), lasting hours or days. Synaptic plasticity can occur in two ways: creating new synapses (synaptogenesis) and removing existing synapses (synaptic pruning), which are described below.

4.8.2.1 Synaptogenesis and Synaptic Pruning

The elimination of unnecessary synapses from the central nervous system is known as synaptic pruning. Although this process lasts through the life span, majority of the synapses eliminate from the human brain between the child birth and the puberty. At the birth, human brain consists of more than 80 billion of neurons. During the first two years after child's birth, size of the brain grows significantly. In this period there is no much neurogenesis take place. The growth of the brain occurs as the result of creation of new synapses and myelination of nervous fiber. Myelination refers the forming white substance surrounding the axon. Creation of new synapses is called the synaptogenesis.

At the child's birth, a neuron consists of approximately 2,500 connections. In two years it becomes about 15,000 and this is far more than the functionality needed. When synaptogenesis reaches to a peak level it starts to prune weak and unnecessary synapses from the central nervous system as describe in the Figure 4.8 [155]**.** Pruning occurs due to environmental factors and learning. While infant is learning, weak synapses are eliminating by strengthening the functions of remaining ones. Pruning process lasts until the death of healthy persons, but significantly occurs until the adolescence. At the end of this process brain contains about 50% synapses that were in a two-year-old child.

4.8.3 Functional neuroplasticity

The changes happen in the brain due to learning and memory is called the functional neuroplasticity. The Magnetic Resonance Imaging (MRI) based structural imaging techniques evidence that learning and memory yields both short term and long-term changes in synapses. While learning and memory, as a result of structural adjustments and intracellular biomechanical process, permanent changes appear in synaptic relationships between neurons [133].

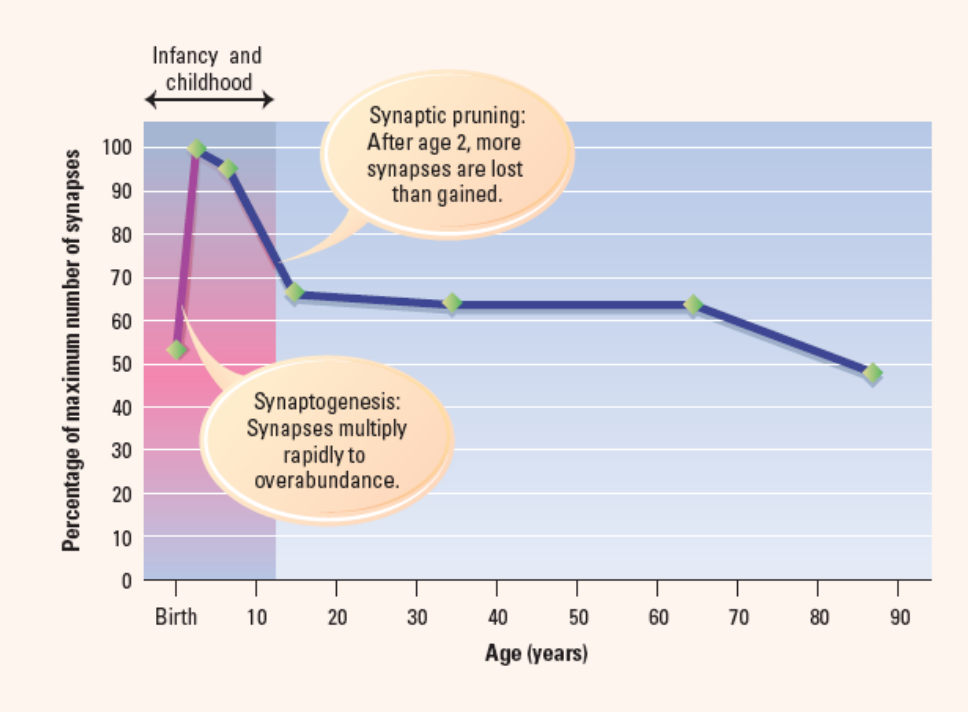


Figure 4.8: Changes of synapses

4.9 Positive and Negative Outcomes of Neuroplasticity

Now it is clear that the brain continues its changes throughout one's lifespan. These changes can happen in both positive and negative directions to respond to intrinsic and extrinsic influences [156]. The majority of the remodeling of the human brain takes place from infant level to adolescence. But this process continues until death. In previous sections we discussed how neuroplasticity occurred, types of different plasticity of brain and the factors which promote this plasticity.

The structure and the functions of the brain depend on several parameters like activity, education, environment, food habits, etc. So that, the remodeling of brain shows both positive and negative outcomes that depend on the above factors.

4.9.1 Positive outcomes of neuroplasticity

Positive neuroplasticity improves the brain and body health. Also, it enhances the capacity of the creativity and the memory. It has been shown that by improving synaptic plasticity, new skills can be developed. Physical exercises and meditation cause for better cognition and maximize the functions of the aging brain [157]. When a child has some disorder in a particular function such as hearing, the brain removes those neuron and axons which does not serve him and replaced the new neurons which are able to develop new skills. This gives a child a second chance to develop his skills.

In addition, more efficient communication between sensory and motor pathways, slowing down pathological processes, promoting recovery of sensory losses and improved motor control are some positive outcomes of the neuroplasticity.

4.9.2 Negative outcomes of neuroplasticity

Researchers show that, in the infancy level the size and the weight of the brain increase very rapidly and the significant factor for this incensement is the growth of synapses (synaptogenesis) between neurons (gray matter) and myelination of nerve

fibers (white matter) [158]. This process continues until adolescence and after adolescence, synaptic pruning begins. Synaptic pruning can be described as a learning mechanism and it is largely happened due to the environmental influences [159]. The neurologists explain that in the human brain memories are formed at structures that are known by dendritic spines which communicate other brain cells through synapses. This process continues until the end of someone's life. Nevertheless, about half of the brain connection removes after puberty.

However, synaptic pruning always does not yield only positive outcomes. The loss of extra neurons and pathways may cause difficulties in recovering from a brain injury. Eliminating excess neurons limit the ability to develop new pathways to bypass the damaged neurons. Moreover, under-pruning of synapses slows down the functions of the human brain. For instance, it believes that children and adolescents get mental disorder such as autism due to having excess synapses in the brain. Because synapses are the end point of neurons and neurons connect and communicate with each other through synapses, excessive synapses may maximize the effects of theses brain functions. Thus, having synapses more than necessary can cause some symptoms such as oversensitivity to noise and social experiences. In addition, it may cause to the mental disorder known as 'epileptic seizures' due the more electrical signals being transmitted through neurons [160]. Guomet Tang, a professor in neurology shows how neurons appear autistic brain and a normal brain in the following Figure 4.9.

Schizophrenia is another severe disorder that affects to person's social behavior, thinking ability and feelings. Neurologists show that this happens due to mal-synaptic pruning. The schizophrenia occurs in the late adolescence or early adulthood [161]. Electron microscopy (EM) studies indicate that healthy brain of a human shows cortical synaptic density reaches a maximum at 2 – 4 years of age [162] and then starts to reduce excess neuronal synapses and this mainly happens during the adolescence. However, some studies show that synaptic elimination continues [163] throughout the third decay of the life before stabilization of the

synaptic density in adult brain. So that, the neurologists strongly believe that growth of abnormal synaptic pruning towards severe mental disorders at the early adulthood [164] [165].

In addition, the other mental disorders such as decline in brain function, altered motor control, impaired performance of activities of daily living and amplified perception of pain are some effects of the inappropriate synaptic pruning.

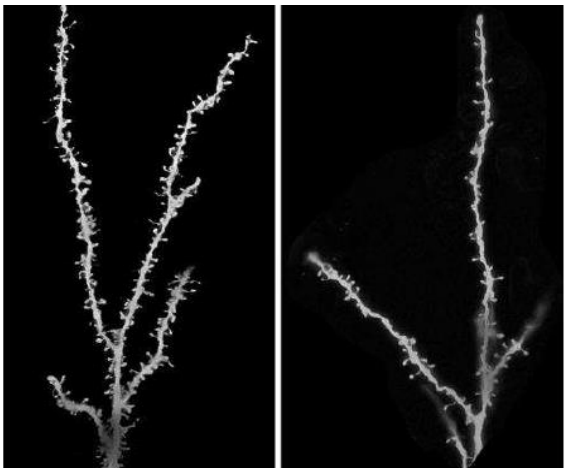


Figure 4.9: Neurons of autistic (left) and normal brains (right)

4.10 Artificial Neural Networks and Human Brain

Human brain, which is having the phenomenal power, is the most complex organ in the human body. The extraordinary power of human brain is far beyond than that of any supercomputer today. The mechanism of the human brain is absolutely different from the conventional 'Von Neumann' architectural computer. A Von Neumann computer works step by step sequentially through an algorithm [23].

The brain is a massively parallel and highly complex information-processing structure. Among a big crowd in a town we can recognize a friend, or identified a voice in a noisy station. Is there any machine to model such complex behavior? The artificial neural networks are developed to mimic the some such fascinating features of the human brain.

In the human brain, dendrites, which project from the cell body receive signals and pass them to the cell body of another neuron. When accumulated signals in cell body reach to a certain threshold limit, the neuron fires and electrical impulses are passed through the axon. At the end, each axon is branched into number of synaptic knobs, also known as axon terminals. With synapses it connects to other neighboring neurons and the signal passes to those adjacent neurons through the synapses. Some synapses get positive outcomes from dendrites and they influence neurons to fire while some get negative outcomes and they weaken the signals. Approximately, a single neuron connects to 10^5 synapses and it believes that the human brain contains about 10^{16} synaptic connections.

Artificial neural networks are created to model this functional behavior of the human brain by directly transferring the concept of neurons. The neurons or basic elements are represented by nodes or artificially designed neurons. The axons are corresponded to the connections between neurons. Dendrites are represented by activation functions. The synaptic weights of artificial neural networks represent the synapses of central nervous system. The concept of training of artificial neural networks came from the psychologist Donald O. Hebbs famous theory "*When an axon of cell A is near enough to excite cell B or repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased*" [22].

However, it is still a challenge to model human brain artificially. The Biological neurons and neuronal activities are far more complex than artificially created neurons. Generally, neurons in human brain do not simply sum the weighted inputs and the dendritic mechanisms in biological systems are much more elaborate. Also, real neurons do not stay on until the inputs change and the outputs may encode information using complex pulse arrangements.

4.11 Summary

This chapter briefly discussed the theoretical basis towards the modelling of hidden layers in ANNs. The ANNs are created by copying the functional behavior of the human brain. The human brain is an immensely parallel and highly complex information-processing dynamic structure. Changes of the human brain occur due the neuroplasticity and synaptic plasticity. These changes take place mainly in the embryonic period and early childhood but extend throughout the life-span. There are positive and negative outcomes of such changes, especially improper synaptic pruning can cause to some severe mental disorders like autism and Schizophrenia. In the next chapter we will discuss the hypothesis and theories that we applied in modeling the hidden layer architecture.

CHAPTER 5

A NOVEL APPROACH TO MODELLING HIDDEN LAYERS

5.1 Introduction

The previous chapter discussed the theoretical basis of achieving the hidden layer architectures in ANN. The chapter 3 was discussed the different approaches on modeling the hidden layer architectures. It showed that the determining the hidden layer architecture in artificial networks is a great challenge. Even though there are several approaches, these methods have various shortcomings. Hence, still the modelling of hidden layers in ANNs remains as an unsolved problem. This chapter discusses the approach on modelling the hidden layers by applying a Peak Search algorithm and Delta Value of hidden layers (PSDV approach) by highlighting the hypothesis. Additionally, it includes inputs, outputs and the various steps of the approach.

5.2 The Hypothesis

This research postulates that '*any given large ANN can be reduced to a smaller-sized ANN by trimming hidden layers and neurons in hidden layers such that the resultant network shows same or better performance*'.

This approach is inspired by the fact that 'the nature is always overestimated.' Naturally, nature consists more than sufficient elements. For instance, animals have two eyes, two ears, etc. Not only that, by also they bear billions of neurons and trillions of connections on their brain. However, if they lose part of these, still their lives would be survived, because when some neurons are damaged, others maximize the functions to compensate for the damaged neurons. In the same line it assumes that a large artificial neural network can make smaller by eliminating neurons and connection which are not very important.

5.3 Inputs

The input of the process will be a large network (network with H hidden layers and N hidden neurons) trained by the backpropagation algorithm. That is this input network consists of N hidden neurons distributed among H hidden layers. The hidden layer i contains n_i hidden neurons, where $i = 1, 2, \dots, H$.

Then

$$n_1 + n_2 + \cdots n_i + \cdots + n_H = N. \tag{5.1}$$

The distribution of neurons among the hidden layers has done in 3 different ways as follows.

- 'Equal' hidden neurons:

  Each layer has same number of neurons. That is

  $$n_1 = n_2 = \cdots = n_i = \dots = n_H .$$

- 'Ascending' hidden neurons

  Hidden neurons in each layer arrange in the ascending order.

  $$n_1 < n_2 < \cdots n_i < \dots < n_H .$$

- 'Arbitrary' hidden neurons

  Each layer contains an arbitrarily chosen number and there is no any special pattern among $n_1, n_2, \dots n_i, \dots, n_H$.

At the first stage, process initializes with this input and determines the number of hidden layers of the highest generalized network. Then the resultant network uses as the input for the second phase. In this stage, it identifies neurons which do not contribute in error decay process and eliminate them from the network.

5.4 Outputs

In the first stage the output is the number of hidden layers in the network which gives the highest generalization. The resultant network has fewer number of hidden layers and hidden neurons comparing with the original one. But still it may have some unnecessary neurons. So that the second phase removes all unnecessary neurons. Thus, the final output of the process is a fully connected network with

$j(< H)$ hidden layers whose total number of neurons is $n'_1 + n'_2 + \cdots n'_j$, where $n'_i < n_i,\ \forall\ i = 1, 2, \ldots, j$

5.5 Process of the New Method

There are two phases in the process. At the first phase network cut down all the excess hidden layers to obtain the minimal number of hidden layers which gives the best generalization. In the second phase it eliminates all the weak neurons that do not contribute to minimize the output error. The whole process is described below.

5.5.1 The Peak Search Algorithm

The process of modeling the hidden layers starts with a large network, trained by backpropagation algorithm. Nonetheless, if there is no such trained network for the given data set, it is able to create and train a network with arbitrary large number of hidden layers (H) and hidden neurons (N). No matter how poor generalization it shows.

The generalization of the network with *h* hidden layers is defined as

$$g_h = \frac{\text{Number of correct response in the testing set}}{\text{Total number of data sets in the testing set}} \times 100\% \qquad (5.2)$$

By the experiments done previously, It was observed that by adding layers, generalization power can be improved up to a certain level and then decreases or retain at the maximum level [32] as shown in the following Figure 5.1, where *H* is the number of hidden layers in the initial network.

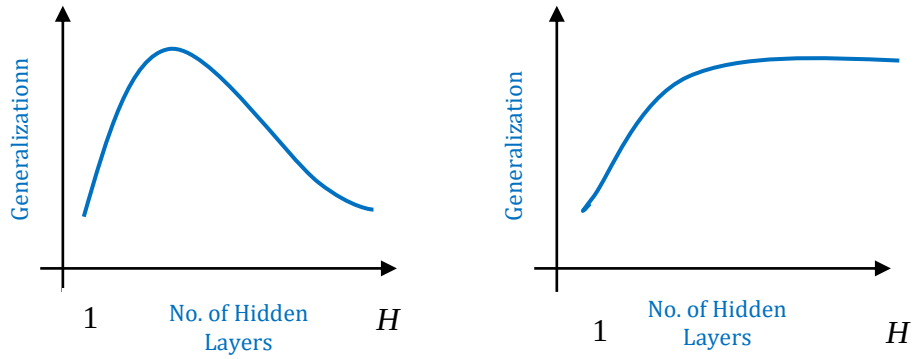


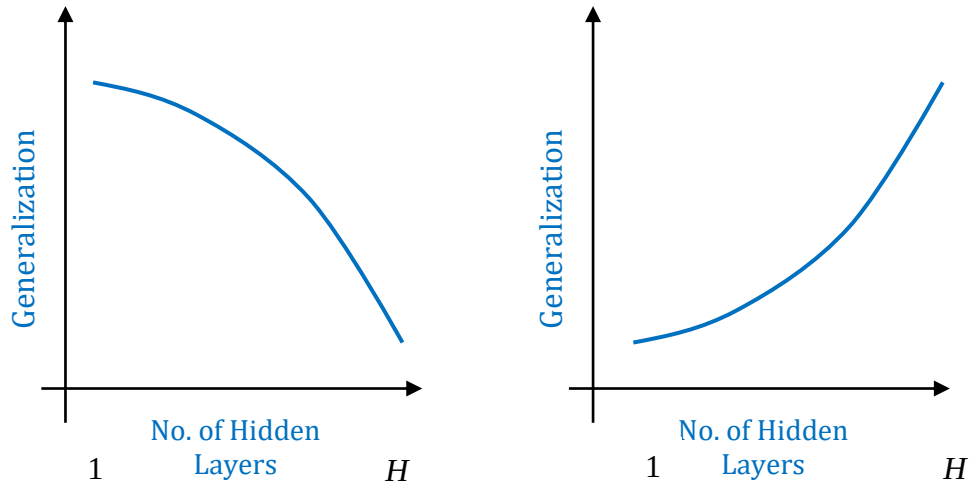
Figure 5.1: Change of the generalization with the number of hidden layers

However, this graph is always not expected to obtain the peak at the middle of the given range. There is a possibility to have the optimum solution for a single hidden layer network or the initially considered architecture with a large number of hidden layers (say *H*) as shown in the Figure 5.2(a) and Figure 5.2(b) respectively.

The initial network contains H number of hidden layers and N total number of hidden neurons. The generalization of initial network g_H is known. The proposed algorithm searches the number of hidden layers (say j) in the architecture which shows the best generalization. On the other words the main objective of this algorithm is to determine an integer j such that $j \in [1, H]$ and

$$g_j = \max\{g_1, g_2, \dots, g_H\}$$

.



(a): A single hidden layer network gives the best generalization

(b): The initial network with *H* hidden layers gives the best

Figure 5.2: Change of generalization with hidden layers

The concept was influenced by the algorithms on binary search (bi-search) [166] and peak search [167]. However, bi-search algorithms are used to approach target value in a sorted array. Most of the existing peak search algorithms reach to the peak level by considering the local maxima or by comparing the maximum value in three consecutive numbers.

The proposed method in this research, reaches to the target by comparing the generalization of the middle value with the generalization of two end points of the interval. According to the inequality holds, lower or upper part of the interval eliminates and repeats the procedure until it reaches to the target value.

The algorithm is based on the assumption that while adding hidden layers to the network, generalization of any data set is increased up to a peak level and then it remains at that level or decreases. The main objective of this algorithm is to determine an integer value j between 1 and H, where j is the number of hidden layers in the network which gives this peak value of generalization. That is, the algorithm searches j such that $j \in [1, H]$ and $g_j = \max\{g_1, g_2, \dots, g_H\}$.

Initially, consider the interval with left end $L \equiv 1$ and right end $R \equiv H$. If the single hidden layer network, which is the simplest architecture, shows generalization 100%, then it is considered as the required network. i.e if $g_1 = 100$, the solution for the given problem is the network with one hidden layer and n_1 hidden neurons. Otherwise, compute the generalization of middle value m of the interval $[L, R]$, g_m. As the number of hidden layers is a positive integer and always we prefer to obtain the minimal architecture, m is chosen as

$$m = \left[\frac{L + R}{2}\right], \tag{5.3}$$

where $[x]$ denotes the integer part of x.

Then the process continues by comparing g_m with the values of generalization of two end points g_L and g_R.

The possibilities of having inequalities among g_L, g_m and g_R are as follows.

Case I: $g_L > g_m \geq g_R$

This can happen either $g_L > g_m = g_R$ or $g_L > g_m > g_R$. In both the instances peak lies in between L and m as depicts in the Figure 5.3. Then it

removes the interval $(m, R]$. So that the new right end of the interval is m. Hence, replace R by m and continue the process.

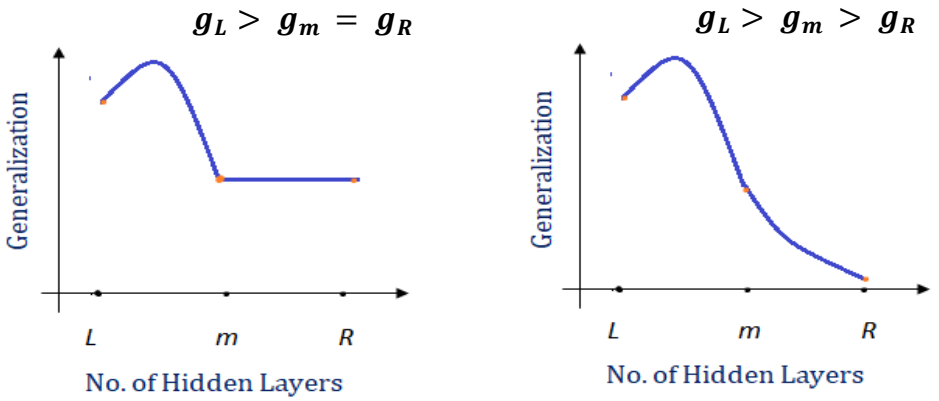


Figure 5.3: Graphs for $g_L > g_m \geq g_R$

Case II: $g_L \leq g_m < g_R$

This is opposite of the case I. Hence, peak lies in the interval $[m, R]$ and then interval $[L, m)$ will be removed. As shown in the Figure 5.4, this case happens when $g_L = g_m < g_R$ and $g_L < g_m < g_R$. Here, replace L by m and continue the process.

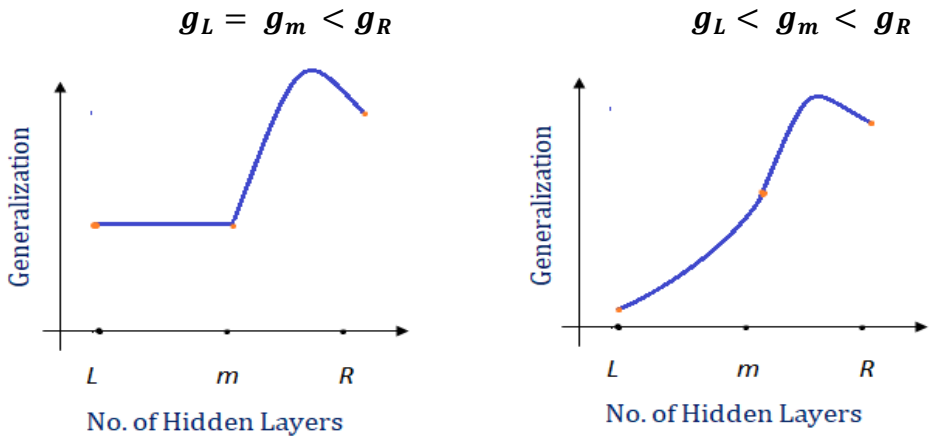


Figure 5.4: Graphs for $g_L \leq g_m < g_R$

Case III: $g_L \leq g_m \geq g_R$

This is the worst scenario. This case could be occurred at four instances. Such as

$g_L < g_m = g_R$, $g_L = g_m > g_R$, $g_L = g_m = g_R$ and $g_L < g_m > g_R$. In all these cases, the peak can find anywhere in the interval $[L, R]$. Then to find the peak value we compute m_1 and m_2 as follows and continue the process.

$$m_1 = \left[\frac{L+m}{2}\right] \quad and \quad m_2 = \left[\frac{m+R}{2}\right]$$

Then one of the following 6 cases can be occurred.

(i) When $g_L < g_{m_1} > g_m \geq g_{m_2} \geq g_R$.

i.e. $g_{m_1} = \max\{g_L, g_{m_1}, g_m, g_{m_2}, g_R\}$. In this case peak lies in the left and right intervals of m_1 . i.e in the interval $[L, m_1]$ or $[m_1, m]$ as illustrated in the following Figure 5.5. So that the peak value can be found in anywhere in the interval $[L, m]$. i. e. peak is in the interval which contains m_1. Therefore, the interval $(m, R]$ can be removed and R is replaced by m. Now m_1 becomes the middle point of the new interval and it agrees the condition of Case III: $g_L \leq g_m \geq g_R$.

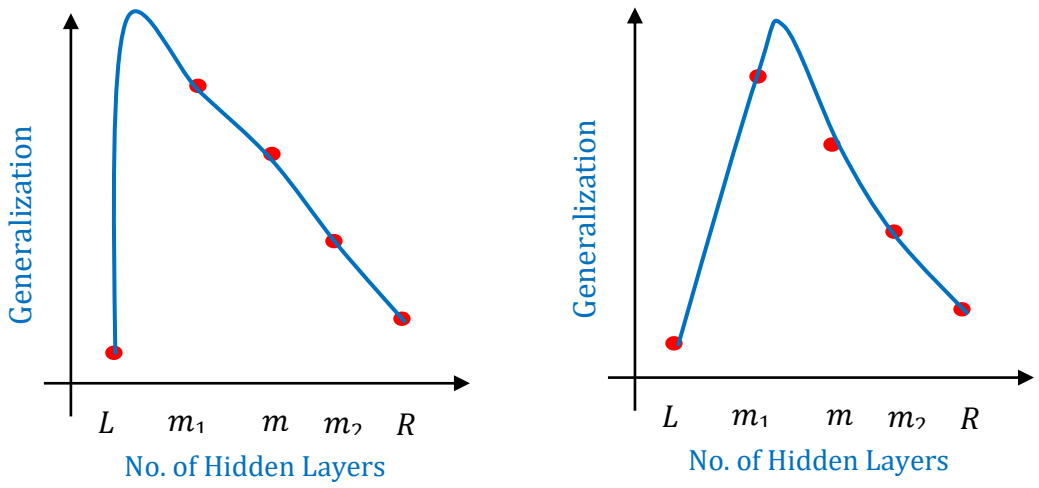


Figure 5.5: Graphs when g_{m_1} is the maximum

(ii) When $g_L \leq g_{m_1} \leq g_m < g_{m_2} > g_R$.

i.e. $g_{m_2} = \max\{g_L, g_{m_1}, g_m, g_{m_2}, g_R\}$. As shown in the following Figure 5.6, this is the opposite of the above (i). Hence, the peak lies any interval which contains m_2. By removing the interval $[L, m)$ and by replacing L by m and the middle point m by m_1 and it can convert to the Case III.

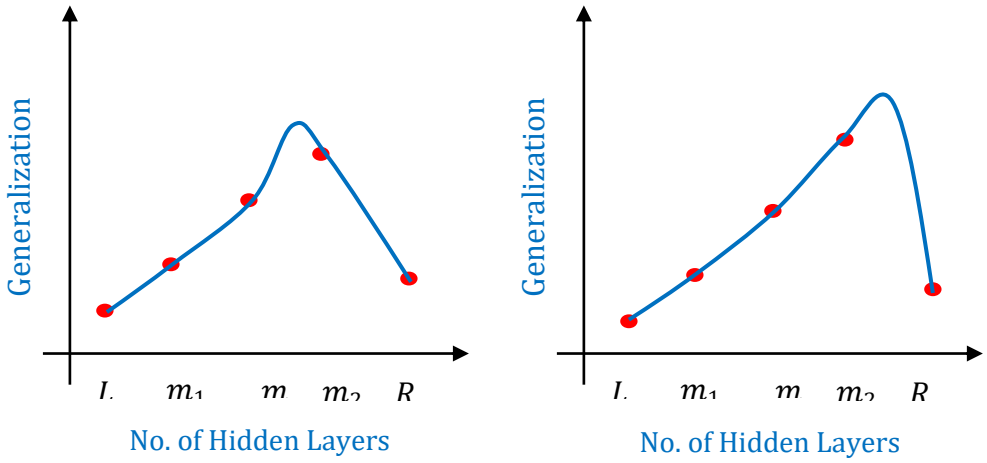


Figure 5.6: Graphs when g_{m_2} is the maximum

(iii) When $g_L \leq g_{m_1} < g_m > g_{m_2} \geq g_R$

i.e $g_m = \max\{g_L, g_{m_1}, g_m, g_{m_2}, g_R\}$. Similar to above (i) and (ii) In this case peak lies in an interval including m. i.e in the interval $[m_1, m]$ or $[m, m_2]$ (Figure 5.7). Hence, peak lies anywhere in $[m_1, m_2]$. Then replace L by m_1 and R by m_2.

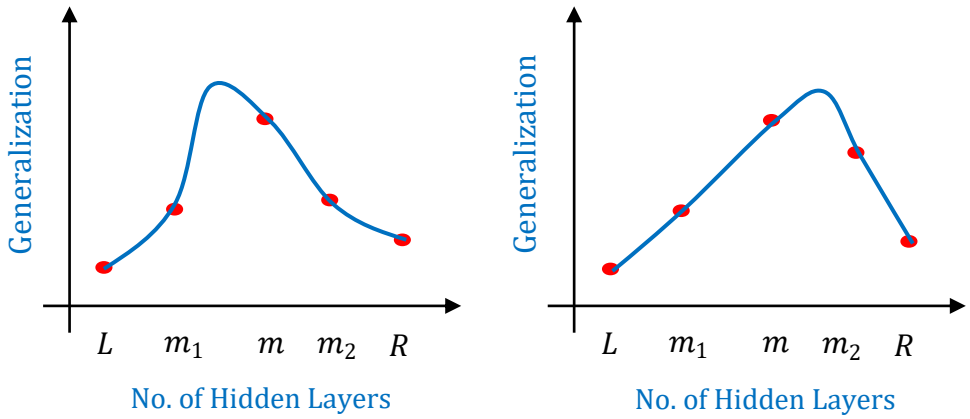


Figure 5.7: Graphs when g_m is the maximum

(iv) When $g_L \leq g_{m_1} = g_m > g_{m_2} \geq g_R$

i.e. $g_{m_1} = g_m = \max\{g_L, g_{m_1}, g_m, g_{m_2}, g_R\}$. As discussed in the previous cases, here peak lies in any interval contains $m_1$and m. i.e in the one of the intervals $[L, m_1], [m_1, m]$ or $[m, m_2]$. In otherwords peak lies in anywhere in the interval $[L, m_2]$. Hence, remove the interval $(m_2, R]$ and continue the procedure by replacing R by m_2.

(v) When $g_L \leq g_{m_1} < g_m = g_{m_2} \geq g_R$

i.e. $g_{m_2} = g_m = \max\{g_L, g_{m_1}, g_m, g_{m_2}, g_R\}$. This is opposite of the above (iv). In this case peak lies in the interval $[m_1, R]$. Hence, remove $[L, m_1)$ and continue by replace L by m_1.

(vi) When $g_L \leq g_{m_1} = g_m = g_{m_2} \geq g_R$

In this case it is not possible to determine the peak. Hence, it needs to check the generalization of middle point of each of the interval $[L, m_1], [m_1, m], [m, m_2]$ and $[m_2, R]$.

The process stops when $L - R = 1$ or L, m, R are consecutive numbers. Then the number of hidden layers in the architecture is

$$j = \{g_j = \max\{g_L, g_R\}\} \tag{5.4}$$

or

$$j = \{g_j = \max\{g_L, g_m, g_R\}\} \tag{5.5}$$

The following Figure 5.8 is a flow diagram to describe the procedure of obtaining the number of hidden layers in the most appropriate network and Figure 5.9 describes the peak search algorithm.

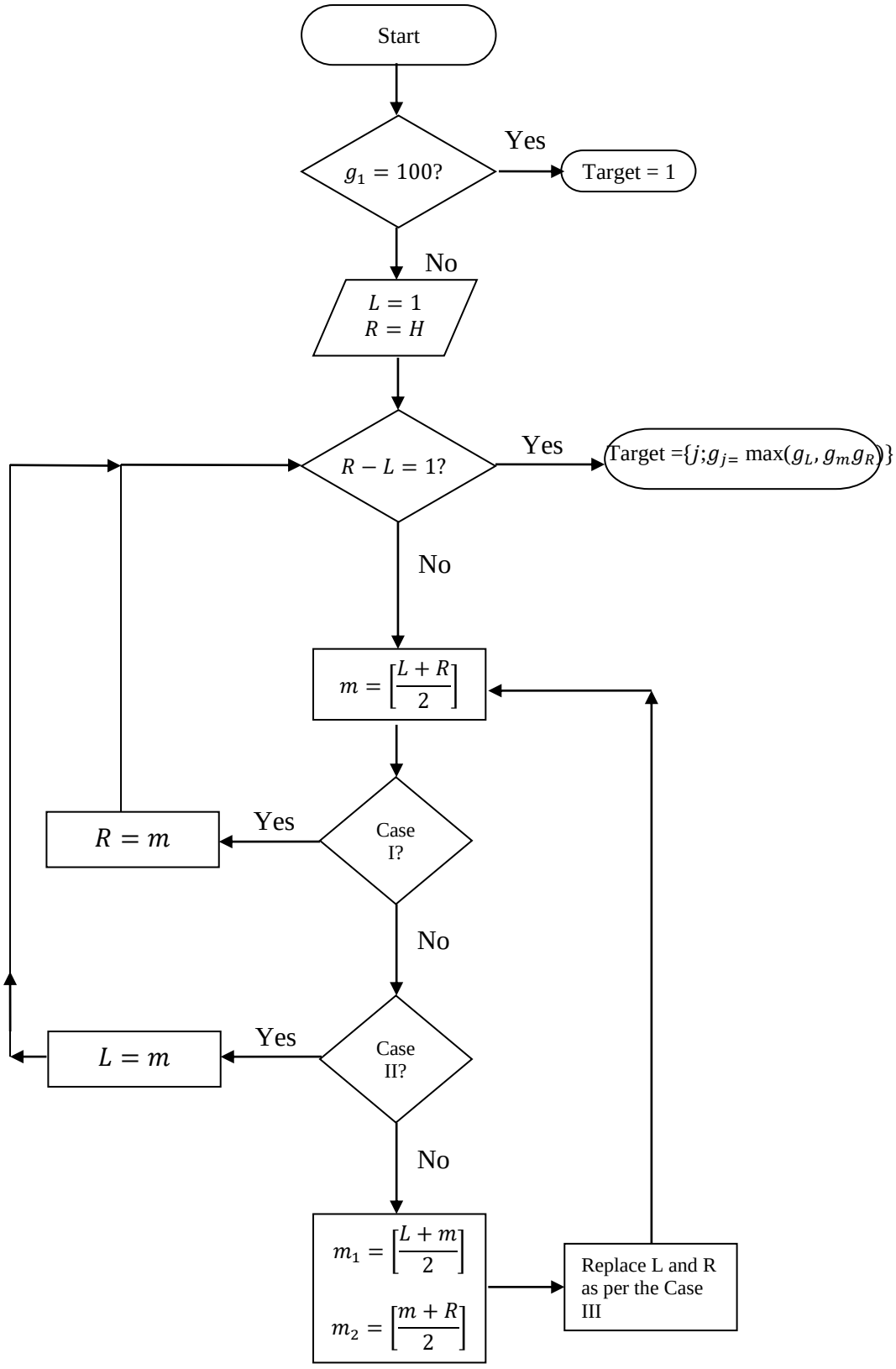


Figure 5.8: Flow diagram for peak search algorithm

The Peak Search Algorithm

Consider initial network with H hidden layers and $\{n_1, n_2, \dots, n_H\}$ hidden neurons

Start

Input : H, g_H

Output : j such that $g_j \geq g_u, \forall u \in [1, H]$

Step 1

Compute g_1

If $g_1 = 1$, then $j = 1$ {*The required architecture has only one hidden layer*}

Else $L = 1,\ R = H$

Step 2

While $R - L > 1$, compute the middle value m of L and R

$$m = \left[\frac{L + R}{2}\right]$$

Step 3

Train the network with m hidden layer and $\{n_1, n_2, \dots, n_m\}$ hidden neurons

Compute g_m

If $g_L > g_m \geq g_R$, replace R by m and repeat Step 2 {*Peak lies in the interval* [L, m]}

Else if $g_L \leq g_m < g_R$, replace L by and repeat Step 2{*Peak lies in the interval* [m, R]}

Else ($g_L \leq g_m \geq g_R$) compute middle values of $[L, m]$ and $[m, R]$

$$m_1 = \left[\frac{L + m}{2}\right], \qquad m_2 = \left[\frac{m + R}{2}\right]$$

Replace L by m_1 and R by m_2 and repeat the Step 2

Step 4

Repeat Step 2 and Step 3 until L, m and R are consecutive numbers

End

The number of layer with maximum generalization is j where

$$g_j = \max\{g_L, g_m, g_R\}$$

Figure 5.9: The peak search algorithm

5.5.2 Performance of the algorithm

The proposed algorithm is design to search number j corresponds to the maximum generalization $g_{j,}$ based on the concept that while increasing the number of hidden layers of an ANN, generalization is increasing and reaches to a peak level and then decreasing to a lower level. To analyses the performance of the algorithm, it needs to compute the number of iterations that takes to reach the target.

The process starts with the middle element $\left[{(1+H)}/{2}\right]$ of the array $[1,2,\dots,H]$, where $[x]$ denotes the integer part of the number x. The procedure is represented by a binary comparison tree shown in the Figure 5.10. The lower half of the array represents the left part of middle element and upper part represents the right side and extends the tree in similar fashion. Iteration continues selecting the middle elements of the range where maximum generalization occurred. The process terminates when there is no integer between two values which provide the highest generalization. The worst case arises when tree contains maximum branches. The number of iterations of the worst case is $H-1$, which happens only if all the networks with layers $1,\dots,H$ show the same performance.

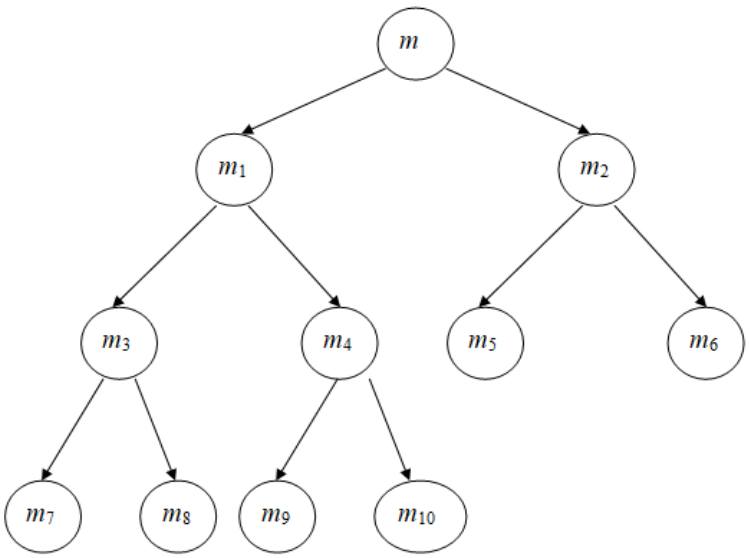


Figure 5.10: Binary comparison tree

5.5.3 Upper limit for the hidden Layers

To avoid getting huge values for output error, normalized weights are chosen for synaptic weights. So that we can assume the input of any neuron in the output layer lie in the interval $[-1, 1]$. Normally, the activation function chosen output layer is the linear function $f(x) = x$. Therefore, by the equation 2.23, the delta value of any output neuron equals to the error between target output and actual output. In addition, the Figure 5.11 shows that by applying the log sigmoid function, output of each neuron converges to a value in between 0 and 1. If the activation function is tan sigmoid, then this value lies between -1 and +1. The following Figure 5.12 illustrates that the derivative of log sigmoid is always less than 0.25 and that of tan sigmoid is less than 1.

Now consider the delta values of hidden layers given by the equation 2.26,

$$\delta_j^h(n) = f_h'\left(net_j^h(n)\right)\sum_{k=1}^{h_n}\delta_k^{h+1}(n)w_{kj}^{h+1}(n)$$

Then, dividing the equation by δ_j^{h+1} , the ratio of δ_j^h to δ_j^{h+1} can be obtain as

$$\frac{\delta_j^h}{\delta_j^{h+1}}(n) = f_h'\left(net_j^h(n)\right)\sum_{k=1}^{h_n}\frac{\delta_k^{h+1}}{\delta_j^{h+1}}(n)w_{kj}^{h+1}(n) \tag{5.6}$$

The experimental results show that delta values in a one particular layer is almost same [32].

Therefore,

$$\frac{\delta_j^h(n)}{\delta_j^{h+1}} \approx f_h'\left(net_j^h(n)\right)\sum_{k=1}^{h_n}w_{kj}^{h+1}(n) \tag{5.7}$$

But as weights are normalized

$$\sum_{k=1}^{h_n}w_{kj}^{h+1}(n) \approx 1. \tag{5.8}$$

Hence, the ratio of the delta value of any particular neuron in layer *h* to layer *h* + 1 approximately equals to the derivative of the activation function. That is

$$\frac{\delta_j^h(n)}{\delta_j^{h+1}} \approx f_h'\left(net_j^h(n)\right) \tag{5.9}$$

Since, maximum values of derivatives of log sigmoid and tan sigmoid functions are 0.25 and 1.

Therefore,

$$\frac{\delta_j^h(n)}{\delta_j^{h+1}} < 1$$

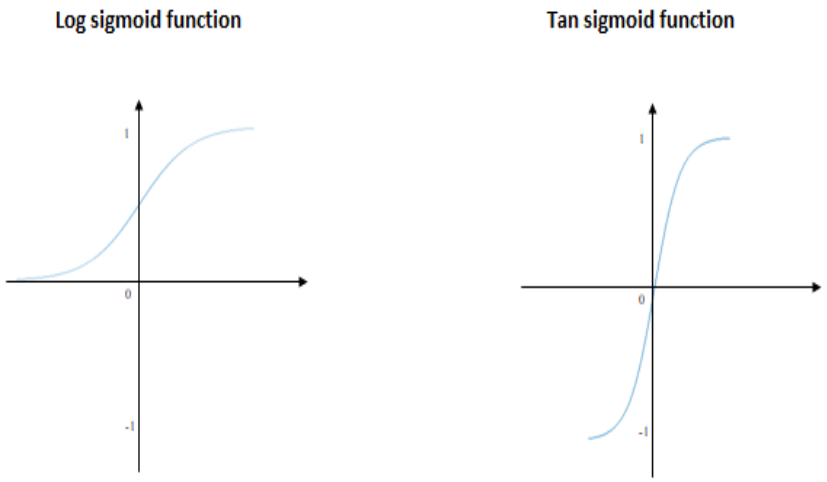


Figure 5.11: Sigmoid functions

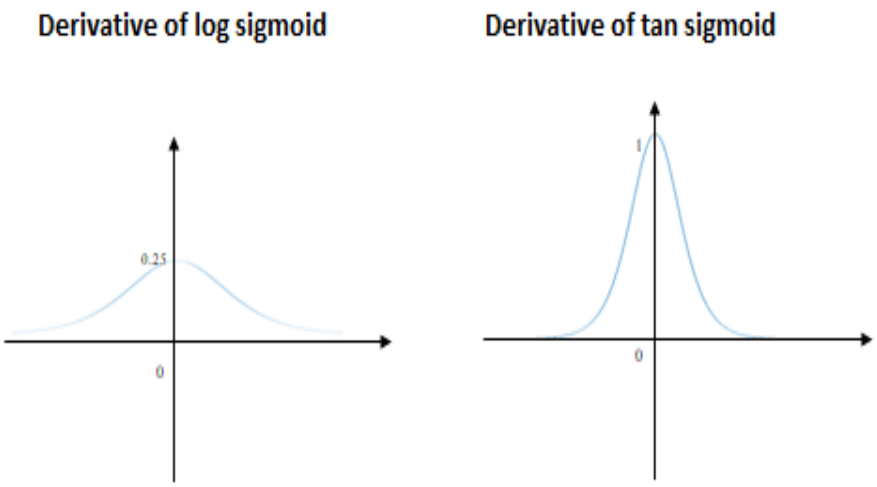


Figure 5.12: Sketch of the derivatives of sigmoid functions

Thus, delta values of each layer is less than that of the previous layer. Hence, when there are many layers, delta values of initial layers become infinitesimal. So that, the correction of synaptic weights are negligible and hence, there is no update of connection weights in the very first layers. Therefore, when network consists of large number of hidden layers, always it shows the same performance as weights are not updating. So that when an integer k is some large value, all the network architectues contain k or more than k hidden layers shows the same generalization.

In addition, suppose that there is a large number of hidden layers, which repeatedly applies the same sigmoid function (log sigmoid or tan sigmoid).

Now consider the two sequences

$$x_{p+1} = \text{logsig}(x_p), \quad \text{and } y_{p+1} = \text{tansig}(y_p) \; p = 1,2 \ldots \ldots$$

Then

$$\limsup_{n\to\infty}(x_n) = 0.66$$

and

$$\limsup_{n\to\infty}(y_n) = 0$$

The Figure 5.13 shows how the sequence x_n converges to its limit for four different initial values 0.0, 2.7, 0.5 and 1.0. Generally, the sequence $x_{p+1} = \text{logsig}(x_p), \; p = 1,2, \ldots.$ reaches its limit 0.66 by 7 or 8 iterations and $y_{p+1} = \text{tansig}(y_p)$, takes more time. Therefore, when the activation function is log sigmoid, ANN architecture with 8 or more layers supposed to give the same output and thus, the performance of any large network is same as the network with 8 layers.

So that, this process can start with a trained network with any number of hidden layers. However, when there are only raw data, the process can start by creating a network with $H(>8)$ hidden layers, $n_1, n_2, \ldots n_H$ hidden neurons.

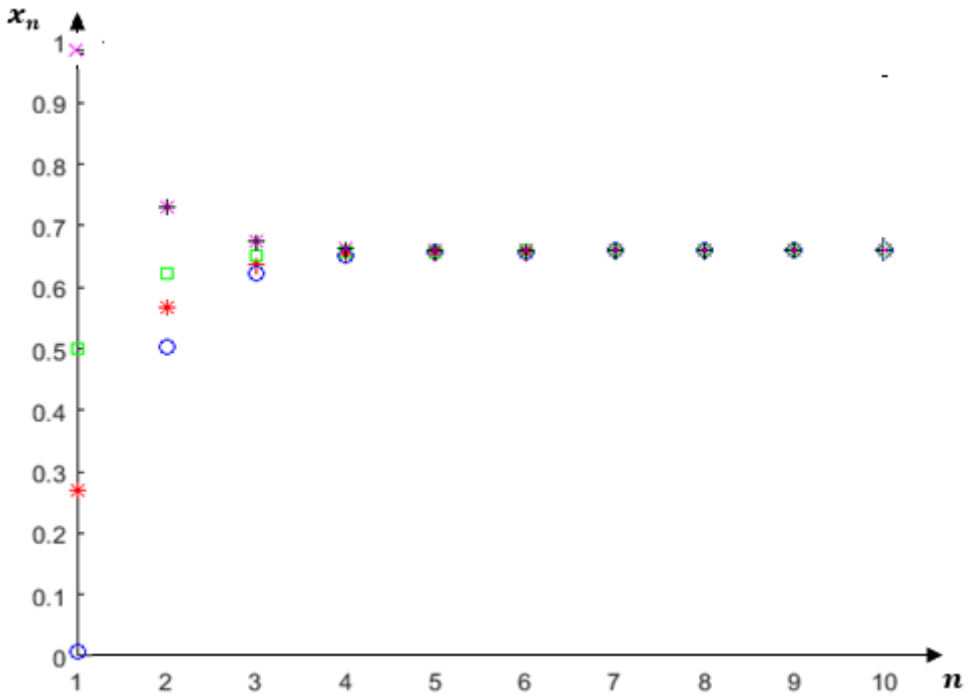


Figure 5.13: x_n for different initial values

5.5.4 Determining number of hidden neurons

The traditional method of training a feed-forward artificial neural network is backpropagation algorithm which can be used successfully in many real world problems. However, as many other training algorithms, it shows some weaknesses such as the problem of local minima. When it reaches to local minima, network is unable to learn and hence, it is a serious barrier for successful training of a network to obtain the desired task.

This research approaches to the minimal architecture by modifying the backpropagation algorithm. The previous section 5.5.1 discussed how to determine the number of hidden layers by starting with an over-sized network. However, still the network may contain some unnecessary neurons. At this stage the network is just like the human brain which is pruned inappropriately. Thus, it recognizes the unimportant neurons while training the network and remove them as synaptic pruning occurs in the human brain. That is in this stage it identifies the neuron which do not contribute to the error decay process and eliminate them from the

network. So that, a new algorithm is proposed to remove unnecessary neurons from the network and fine tune by merging the possible weights to achieve the desired task.

The backpropagation algorithm is the most well-known and widespreaded algorithm among many numerous algorithms that have been proposed to train artificial neural networks. The basic idea behind the train a network by backpropagation algorithm is to obtain weight matrices in order to minimize the error of n^{th} training cycle $E(n)$, which is given by the following equation.

$$E(n) = \frac{1}{2}\sum_{j=1}^{m}\left(t_j(n) - y_j(n)\right)^2, \tag{5.10}$$

where t_j and y_j are the target and actual outputs of the j^{th} neuron of the output layer. m is the number of neurons in the output layer.

For N number of input/output training patterns error becomes

$$E(n) = \frac{1}{2N}\sum_{k}^{N}\sum_{j=1}^{m}\left(t_{jk}(n) - y_{jk}(n)\right)^2 \tag{5.11}$$

The proposed algorithm prunes neurons as much as possible from the hidden layers of over-sized network while maintaining the same error rate as initially given network. Pruning is done by using the delta values of hidden layers. If the network contains h hidden layers, the delta value of the i^{th} neuron of the hidden layer h (the last hidden layer) is given by

$$\delta_i = f_h' \quad (net_i)\sum_{k}\delta_k\, w_{ki}, \tag{5.12}$$

where f_h' is the pre-defined activation function of the h^{th} hidden layer. w_{ki} is the connection weight of the neurons i of the last hidden layer and neuron k of the output layer. δ_k is the delta value of k^{th} neuron in the output layer, which is defined by

$$\delta_k = f_O'(net_k)(t_k - y_k), \tag{5.13}$$

where, f_O' is the activation function defined for output layer. t_k and y_k are the desired and actual outputs respectively.

This value is used to update the connection weights as follows.

$$w_{ki}^h(n+1) = w_{ki}^h(n) + \eta\delta_i^h(n)f_h(n), \tag{5.14}$$

where η is the learning rate.

Then the error $E(n+1)$ of the training cycle $(n+1)$ is calculated by using updated weights obtained from the equation (5.14). The intension of this algorithm is to update weights to reduce the output error at the each training cycle. However, the above equation implies that, zero delta value means there is no update of the particular weight. It implies that the hidden neurons with zero delta values do not contribute to decrease the error. So that the hidden neurons with zero delta values are identified as less salience neurons and eliminate them from the ANN architecture does not affect to the performance of the network.

Empirical results show that very often, there is a correlation between summation of delta values of hidden layers and the output error which can be positive or negative [168]. Thus, we use this correlation to identify the removable neurons. Let this correlation denote by $\gamma_{\delta_h,E}$, where,

$$\gamma_{\delta_h,E} = corr\left(\sum_{k=1}^{n_h}\delta_k, E\right) \tag{5.15}$$

Therefore, to obtain a more precise network, the correlation defined in the above equation is used. If the correlation is positive, sum of the delta can be reduced by removing neurons with positive delta values. But according the equation (5.14) neurons with zero delta values are recognized as unimportant neurons. Therefore, when $\gamma_{\delta_h,E}$ is positive, the minimal architecture obtained by removing neurons with positive infinitesimal delta values. On the other hand, when the correlation is negative, neurons with negative delta values, which are very close to zero, will be removed to obtain the desired architecture.

The pruning has the same meaning of synaptic pruning in neuroscience. It facilitates changes the neural configuration by removing weak neurons and synapses while strengthening the remaining. In synaptic pruning while pruning the weak neurons from the nervous system it merges the similar neurons to strengthen their functions. In same manner to enhance the accuracy of the network, removing weights are merged with the similar weights.

5.5.5 Merge the similar neurons

The whole process of pruning neurons is inspired by the concepts of neuroplasticity and synaptic pruning. While pruning unnecessary neurons from the human brain, it increases the functions of the remaining ones. In similar, this process maximizes the weights of synaptic connection while removing the unimportant neurons. Although, the particular neuron does not contribute to reduce the output error, the contribution of the input and output weights attached to that neuron are not negligible. However, while removing the desired neuron, all its input and output connections also removed from the network. Thus, the weights of the removing connections will be merged with the *similar weights* to obtain more efficient network. The 'similar weights' are the weight vectors with the same orientation.

Let j^{th} neuron of hidden layer h be identified as a removable neuron. Suppose $V = (v_{ji})_{p\times q}$ and $W = (w_{kj})_{q\times r}$ are the input and output vectors of layer h respectively.

Let $V = \begin{bmatrix} V^{R_1} \\ V^{R_2} \\ \vdots \\ V^{R_p} \end{bmatrix}$ and $W = [W^{C_1}, W^{C_2}, \dots, W^{C_r}]$.

Where $V^{R_1}, V^{R_2}, \dots\dots, V^{R_P}$ are the row vectors of the input weight matrix and $W^{C_1}, W^{C_2}, \dots, W^{C_r}$ are column vectors of the output matrix of the layer h. When removing j^{th} neuron, the row vector $V^{R_j} = [v_{j1}, v_{j2}, \dots, v_{jq}]$ and the column vector $W^{C_j} = [w_{1j}, w_{2j}, \dots, v_{qj}]^T$ will be removed. While removing, they merge with similar vectors as described below.

When two input vectors V^{R_j} and V^{R_k} are similar, then

$$\langle \frac{V^{R_j}}{\|V^{R_j}\|}, \frac{V^{R_k}}{\|V^{R_k}\|} \rangle = 1, \tag{5.16}$$

where $< u, v >$ denotes the scalar product of two vectors u and v.

Thus, if neuron j in layer h is identified as the removable neuron, and V^{R_k} and W^{C_l} are the similar vectors to V^{R_j} and W^{C_i} respectively, then V^{R_k} merges with V^{R_j} and W^{C_j} merges with W^{C_l}.

5.5.6 The new algorithm

The process starts with the ANN structure obtained by the PSA. Still it may contain some unimportant neurons. Let the number of hidden layer be h and the total number of hidden neurons is N. Let the number of different input/output vectors in training set be p. Assume the number of hidden neurons in layer h is n_h.

Step 1

Initialize random normalized weights.

Train the network once by backpropagation algorithm.

Step 2:

Compute the correlation between summation of delta values of each hidden layer and output layer.

$$\gamma_{\delta_h,E} = corr\left(\sum_{k=1}^{n_h} \delta_k, E\right)$$

Identify the removable nodes according to the value of $\gamma_{\delta_h,E}$ and remove them from the network.

Step 3:

Merge the weights with similar weight vectors

Step 4:

Train the network for fine tuning until the desired output.

The state diagram of removing unimportant neurons from the architecture is depicted in the Figure 5.14. Once the process satisfies the stopping criterion, that is when the error defined by the equation 2.11 becomes less than the given value, the network stops training.

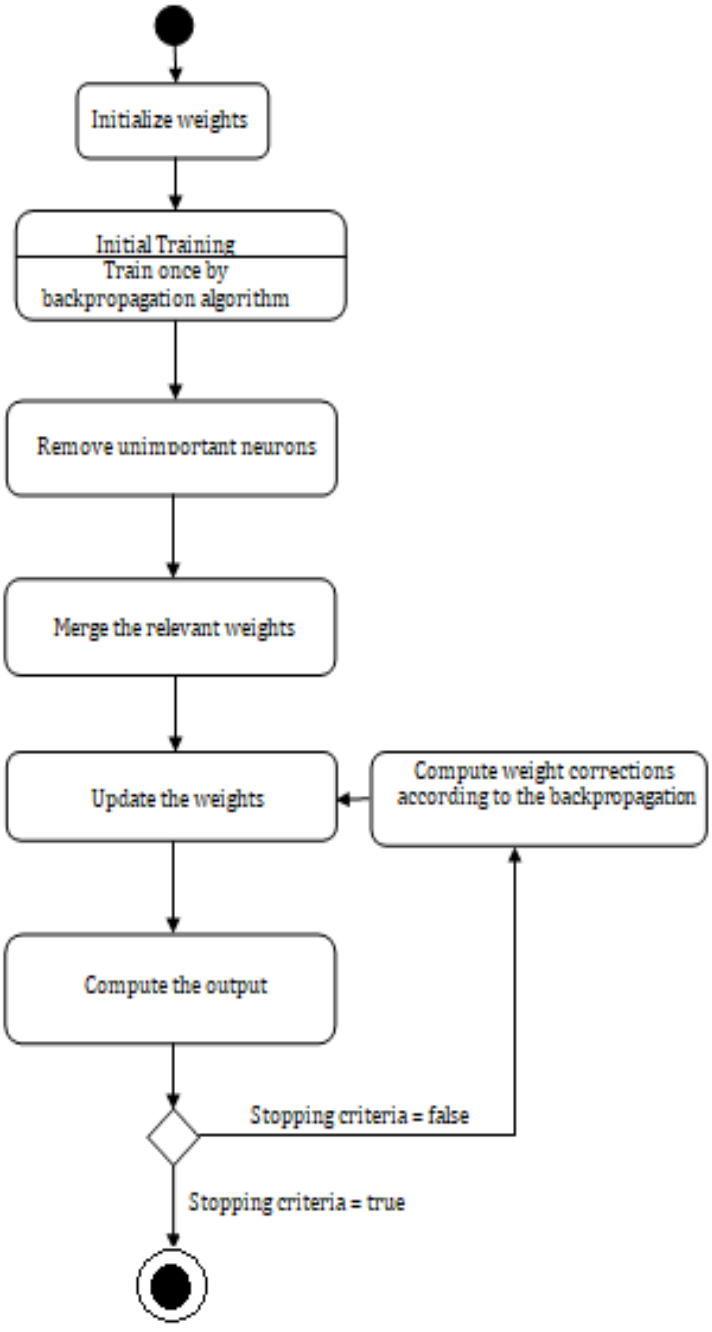


Figure 5.14: Illustration of removing unimportant neurons

5.6 Summary

This chapter focused on the methodology on the designing the optimal solution of problem of hidden layer architecture in artificial neural networks. The process of modelling the hidden layer architecture is based on the hypothesis that any large network will be able to reduce a small sized one by trimming its layers and neurons by managing the same or better performance of the original network. The solution of optimal architecture consists of two main stages. First it determines the number of layers by cutting down the inappropriate layers from a large-sized network. Then eliminates the unnecessary neurons which do not contribute to error decay process. The procedure was inspired by the facts of neuroplasticity and synaptic pruning. In the next chapter the all the experiments carried to justify the hypothesis and evaluation method will be discussed.

CHAPTER 6

EXPERIMENTAL DESIGN AND RESULTS

6.1 Introduction

The previous chapter discussed the methodology of novel approach on designing the hidden layer architecture in ANNs. The evaluation which describes the mechanism of using this methodology is the most important section in the research. So that this chapter presents the experiments carried on the project to evaluate the modeling of the hidden layer architecture. Firstly, it shows that the performance of the network can be improved by increasing the number of hidden layers. Then describes how to achieve the optimal solution by using the novel approach.

6.2 Experimental Design

Most of the existing methods are based on the assumption that a single hidden layer ANN is sufficient to solve many real world problems. However, initially this project shows that the multilayer perceptron of ANN performs better than the single hidden layer networks.

Next it reaches to the optimal solution. This process is done by the hypothesis that a smaller sized network can be obtained by a given a large sized network by trimming down its hidden layer architecture where the resultant network performs same or better than the original one.

The following sections describe the experimental setup, test cases and testing strategies made on this design.

6.2.1 Experimental setup

In order to test the above, some benchmark problems from different domains were considered. All the benchmark problems were chosen from UCI machine learning repository [169]. UCI machine learning repository maintains more than 400 data

sets. Among those it was chosen data sets which have used by other researchers for the purpose of comparison of the results. In addition it was chosen data sets with large and small number of instances, attributes and classes (outputs). Thus, altogether it was considered 34 data sets belong to 19 different domains.

Each set is divided in to two classes for training and testing purposes. In order to examine the different features, 5 benchmark problems namely Cancer, Card, Diabetes, Flare and Knowledge were chosen and 4 different network configuration of each data set were taken in to account. For example 4 different network architectures of Cancer problem namely Cancer I, Cancer II, Cancer III and Cancer IV were designed. The first 3 sets consist of 75%, 50% and 25% data samples in the training set and the last set consists only 20 samples. The details of all the data sets are depicted in the following Table 6.1.

The performance of an artificial neural network depends on several parameters such as hidden layer architecture, learning rate, and the activation function, etc. Since, this research concerns the hidden layer architecture of the network, all the other parameters except number of hidden layers and the number of hidden neurons, made constants throughout the training and testing process. The backpropagation algorithm used to train each network. The logsigmoid function used as the activation function for hidden layers and linear function used for the output layer. When it was necessary, attribute values were recalled to be a real number between -1 and +1. For each case, learning rate of backpropagation algorithm was fixed as 0.1. Initially, all the weights were chosen randomly and normalized. The stopping criteria was decided as the error given by the equation (2.14) is 10^{-4} or pre-decided maximum number of iterations.

To test the above hypothesis, input was a large network trained by the backpropagation algorithm. These were collected by different users who involve in the research on the field of ANN.

Table 6.1: Information of Data Sets

	Data Set	No. of Inputs	No.of outputs	No. of instance	Training patterns	Testing patterns
1	Banknote	4	2	1372	1029	343
2	Cancer I	9	2	699	525	174
3	Cancer II	9	2	699	350	349
4	Cancer III	9	2	699	175	524
5	Cancer IV	9	2	699	24	689
6	Card I	51	2	690	518	172
7	Card II	51	2	690	345	345
8	Card III	51	2	690	172	518
9	Card IV	51	2	690	12	678
10	Cardio	23	3	2126	1594	532
11	Climate	14	2	540	405	135
12	Diabetes I	8	2	768	576	192
13	Diabetes II	8	2	768	384	192
14	Diabetes III	8	2	768	192	576
15	Diabetes IV	8	2	768	50	716
16	Flare I	25	3	1066	800	266
17	Flare II	25	3	1066	533	533
18	Flare III	25	3	1066	266	800
19	Flare IV	25	3	1066	50	1016
20	Glass	9	7	214	160	54
21	Heberman	3	2	306	230	76
22	Iris	4	3	150	112	38
23	Knowledge I	4	4	403	302	101
24	Knowledge II	4	4	403	201	202
25	Knowledge III	4	4	403	100	303
26	Knowledge IV	4	4	403	20	383
27	Monk’s 1	6	2	556	124	432
28	Monk’s 2	6	2	602	170	432
29	Monk’s 3	6	2	554	122	432
30	Seeds	7	3	210	158	52
31	Statlog	13	2	270	202	68
32	Thyroid	5	3	215	161	54
33	Tissue	9	2	106	80	26
34	Yeast	8	9	1484	1113	371

6.2.2 Test cases

In this chapter, first it will show that deep networks perform better than shallow ones. In order to show this it has chosen above mentioned 5 bench mark problems Cancer, Card, Diabetes, Flare and Knowledge. In addition, as mentioned in above, different configuration of each of the sets was considered to test the behavior of the network for different sizes of training sets. The descriptions of all 5 data sets are given here while the others have described in the Appendix A.

6.2.2.1 Breast Cancer Wisconsin data set (Cancer)

The data set was introduced by Dr. William H. Wolberg of the University of Wisconsin Hospital, Madison to diagnose the breast cancer and classify that a tumor as either benign or malignant level [170],[171],[172]. The decision is made based on the information gathered by microscopic examination of 9 features.

The data set contains 699 continuous examples, where 65.5% are in benign stage [173]. To examine the performance of the proposed method different 4 types of data sets, namely Cancer I, Cancer II, Cancer III and Cancer IV were considered with distinct testing and training sets. The first group contains 75% of data in the training set while other 25% used for testing purpose. The 2^{nd} set considered with 50% data in training and the 3^{rd} set contains 25% as training data. Finally, a very small group of data (20 sets) trained and tested the performance.

6.2.2.2 Credit card approval data set (Card)

A database to predict the approval or rejection of credit card of an applicant is presented here. Each example represents the details supplied by a real applicant and output shows whether the corresponding organization granted a credit card to the client or not. The decision makes based on 51 inputs with continuous values and 690 examples. Out of 690 applicants 44.5% show positive output [174]. Four different networks architectures Card I, Card II, Card III and Card IV were designed from this data base as shown in the following Table 6.3.

6.2.2.3 Pima Indians diabetes data set (Diabetes)

This data was originally created by the National Institute of Diabetes and Digestive and Kidney Diseases to binary classification on whether a patient has diabetes. There are records of 768 patients and out of 500 (≈ 65.1%) shown positive for diabetes. All the patients here are females of above 21 years old of Pima Indian heritage. Four different networks, namely Diabetes I, II, III and IV carry 75%, 50%, 25%, and 2% samples respectively created for testing [175].

6.2.2.4 Solar flare data set (Flare)

This database has been created to predict the solar flare which will occur in next 24 hours by using the information on past 24 hour period. In the set there are 1389 attributes and results are classified in three different classes, common flare, moderate flare and severe flare [173].

There are 10 attributes in the input set. First 3 inputs are given as the alphabetical characters whilst rest are integers. Before the training process, alphabetic characters converted to integers.

6.2.2.5 User knowledge modeling data set (Knowledge)

The dataset is about the users' learning activities and knowledge levels on subjects of DC Electrical Machines. Information of 403 users with 5 attributes including the study time and exam performance considered for analysis. According to the information, uses knowledge was classified into four classes, very low, low, middle and high [176].

6.2.3 Testing strategies

Each of the above case was tested for their different configurations. As discussed in chapter 5 generalization remains unchanged for layers greater than l, where generally l is greater than 8. Hence, the maximum number of layers H set as a number greater than 8 and less than 20. Normally, the network with H hidden layers

consists of N hidden neurons. Where N is approximately same as its training patterns, but it can differ according to the user. The number of hidden layers in the layer h is denoted by n_h. Hidden neurons chose in 3 different ways;

- 'Equal' hidden neurons:

  Number of hidden neurons in each layer is same. That is

  $$n_1 = n_2 = n_3 = \dots\dots\dots = n_H$$

- 'Ascending' hidden neurons

  Hidden neurons in each layer arrange in the ascending order.

  $$n_1 < n_2 < n_3 < \dots\dots\dots < n_H$$

- 'Arbitrary' hidden neurons

  Each layer contains an arbitrary chosen number and there is no any special pattern among $n_1, n_2, n_3, \dots, n_H$.

Generally, in each case

$$n_1 + n_2 + n_3 + \dots\dots\dots + n_H \cong N$$

Initially, it was tested how the generalization changes while number of layers is increasing. In order to test this, the above mentioned 5 bench mark problems trained for different number of hidden layers varying from 1 to H.

Next, the performance of novel algorithm was tested by using all the 34 data sets. In the first step it determines the most appropriate number of hidden layers in the network by using the peak search algorithm (PSA). Then eliminate irrelevant neurons by considering the correlation coefficient of summation of delta values of each hidden layer and the output error.

6.3 Experimental Results

6.3.1 The variation of network performance with the number of layers.

The below Table 6.2 shows the total number of samples in the training set (***P***) and how the generalization changes with the number of hidden layers in each of the Cancer problem. These results show that after reaching its maximum value, the generalization decreases to a certain level and retain there while hidden layers add

to the network. Cancer I, II and III show their best generalization at 4 hidden layers and then decreases until 6-9 layers. After this level although hidden layers are added generalization is remained as constant. The Figure 6.1 shows the how the performance of Cancer I changes while increasing the hidden layers. It is clear that while adding hidden layers to the network generalization increases to a peak value and then decreases to a certain level. The Figure 6.2 compares the performance of all the Cancer problems. It indicates that when the training set is large, network shows better performance. For example, Cancer I problem has 525 samples in the training set. Initially, consider network with 20 hidden layers. As it was explained in the section 6.2.3 the total number of hidden neurons chosen as a number close to 525. When these neurons divided equally among 20 layers, each layer consists of $\left[\mathbf{525}/\mathbf{20}\right] = \mathbf{26}$ hidden neurons. This set shows the maximum generalization 99.4%. Further Cancer II and Cancer III perform in the same way and obtain their maximum generalization 98.6% and 95.0% respectively. However, Cancer IV problem has only 20 neurons in the training set and each layer contains only one hidden neuron. The performance of this set is very weak and the maximum generalization it shows is 66.7%.

Table 6.2: Changing performance with hidden layers in Cancer problems

Data Set	***P***	**No. of Hidden Layers**											
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**12**	**20**
Cancer I	525	97.7	97.7	98.9	**99.4**	99.4	96.0	94.3	95.4	62.6	62.6	62.6	62.6
Cancer II	350	97.1	97.4	97.7	**98.6**	97.7	97.7	65.6	65.6	65.6	65.6	65.6	65.6
Cancer III	175	94.7	94.7	94.7	**95.0**	94.7	65.8	65.8	65.8	65.8	65.8	65.8	65.8
Cancer IV	20	63.5	**66.7**	65.4	65.4	65.4	65.4	65.4	65.4	65.4	65.4	65.4	65.4

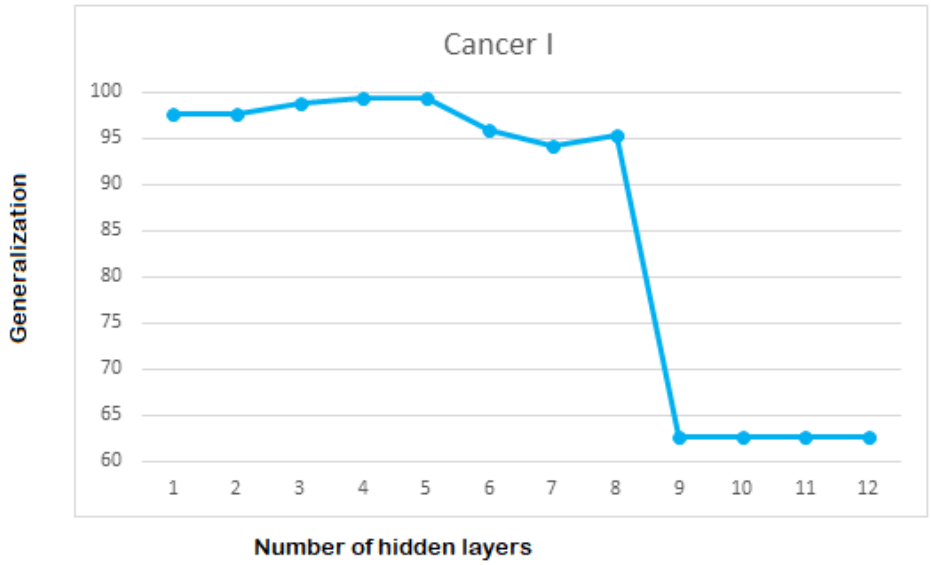


Figure 6.1: Changing performance with hidden layers in Cancer I problem

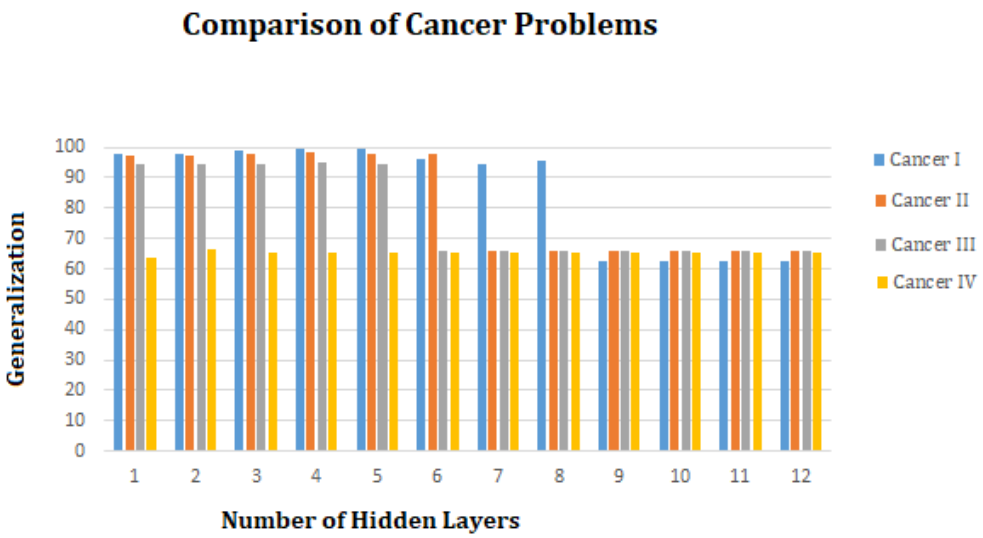


Figure 6.2: Generalization comparison of Cancer problems

Card and diabetes problems also show the similar pattern (Table 6.3 and Table 6.4). In the card problem there are 51 attributes and comparatively Cancer sets generalization of these configurations is weak. The best performance that Card problems show is 88.1% when there are 75% of samples in the training set and 6 hidden layers in the network. Card I and II increase the generalization while adding hidden layers to the network and reach to a similar peak value at 6 hidden layer network and then decline. Card III gets its maximum generalization 84.8% for 5 hidden layered networks. Further in Card IV, when there are 20 samples in the training set, it shows very poor performance. Although it reaches to the peak value 65.7% with 3 layer network, this is much lesser than the peak values of the other configurations (Figure 6.3). The diabetes problems show their highest performance in Diabetes I with 6 hidden layers. The Figure 6.4 depicts that when there are more data in training set, it also shows better generalization.

Table 6.3: Changing performance with hidden layers in Card problems

Data Set	***P***	**No. of Hidden Layers**											
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Card I	518	81.4	81.4	84.9	86.6	86.6	**88.1**	53.4	53.4	53.4	53.4	53.4	53.4
Card II	345	75.1	76.5	84.6	84.6	85.8	**87.2**	55.1	55.1	55.1	55.1	55.1	55.1
Card III	172	68.7	75.7	79.1	84.4	**84.8**	54.8	55.1	54.8	54.8	54.8	54.8	54.8
Card IV	20	59.4	62.2	**65.7**	55.3	55.3	55.3	55.3	55.3	55.3	55.3	55.3	55.3

Table 6.4: Changing performance with hidden layers in Diabetes problems

Data Set	***P***	**No. of Hidden Layers**											
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Diabetes I	576	78.6	79.7	82.8	82.8	81.8	**81.8**	63.5	63.5	63.5	63.5	63.5	63.5
Diabetes II	384	76.0	76.0	77.3	77.9	**80.5**	79.9	63.3	63.3	63.3	63.3	63.3	63.3
Diabetes III	192	71.7	**78.3**	78.3	78.3	63.7	63.7	63.7	63.7	63.7	63.7	63.7	63.7
Diabetes IV	50	71.1	**77.5**	77.5	65.2	65.2	65.2	65.2	65.2	65.2	65.2	65.2	65.2

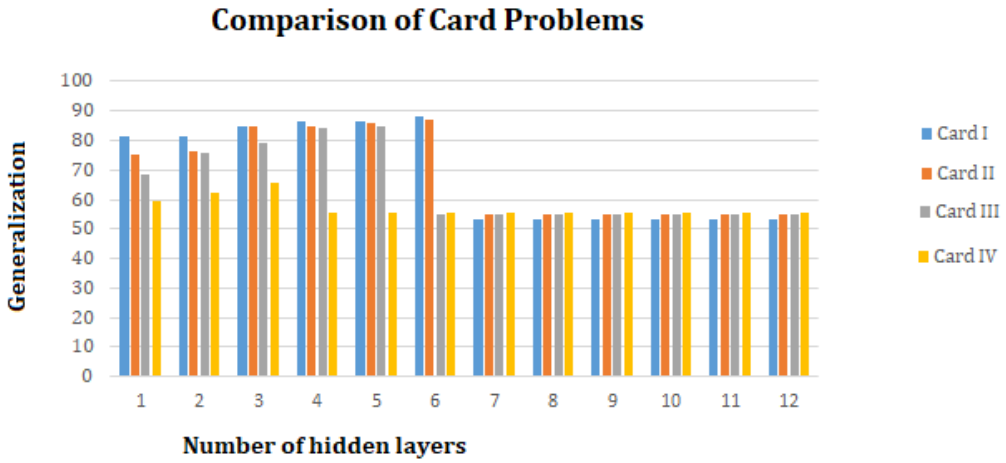


Figure 6.3: Generalization comparison of Card problems

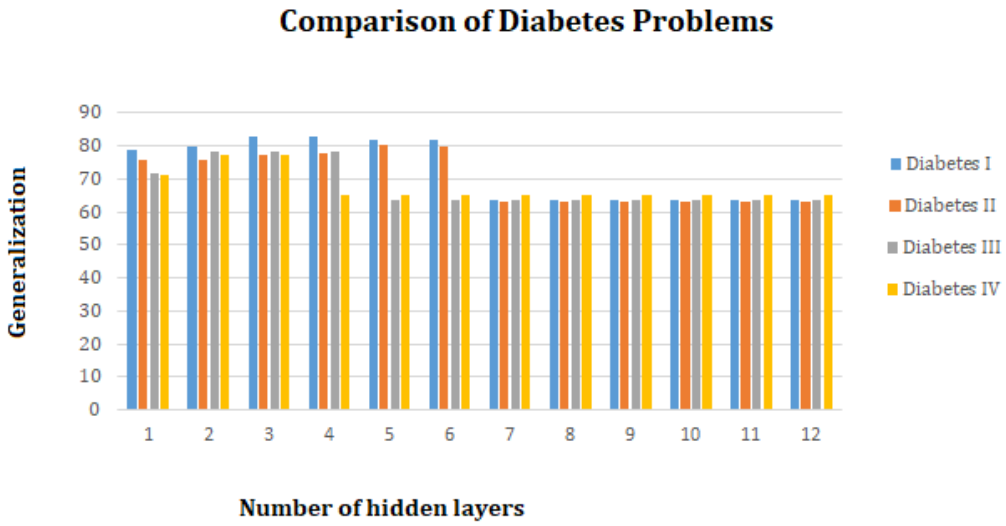


Figure 6.4: Generalization comparison of Diabetes problems

The Flare problems also interpret that the higher number of hidden layers gives improved generalization than shallow networks (Table 6.5). However, in these problems once it reaches to the peak value generalization retain there even for more layers. For example Flare I – III reach to their maximum generalization at 7 layers. The Flare IV reaches to this level at 6 hidden layers. However, as depicts in the Figure 6.5, after reaching their highest values, generalization do not declines and it retains the same value for higher number of layers.

Table 6.5: Changing performance with hidden layers in Flare problems

Data Set	***P***	**No. of Hidden Layers**											
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Flare I	800	71.1	74.4	78.2	82.0	80.1	80.1	**89.4**	89.4	89.4	89.4	89.4	89.4
Flare II	533	81.8	79.2	80.1	83.0	83.5	87.4	**92.7**	92.7	92.7	92.7	92.7	92.7
Flare III	266	82.5	79.1	83.2	82.8	88.9	82.8	**92.1**	92.1	92.1	92.1	92.1	92.1
Flare IV	50	68.4	83.1	80.2	81.6	80.7	**91.1**	91.1	91.1	91.1	91.1	91.1	91.1

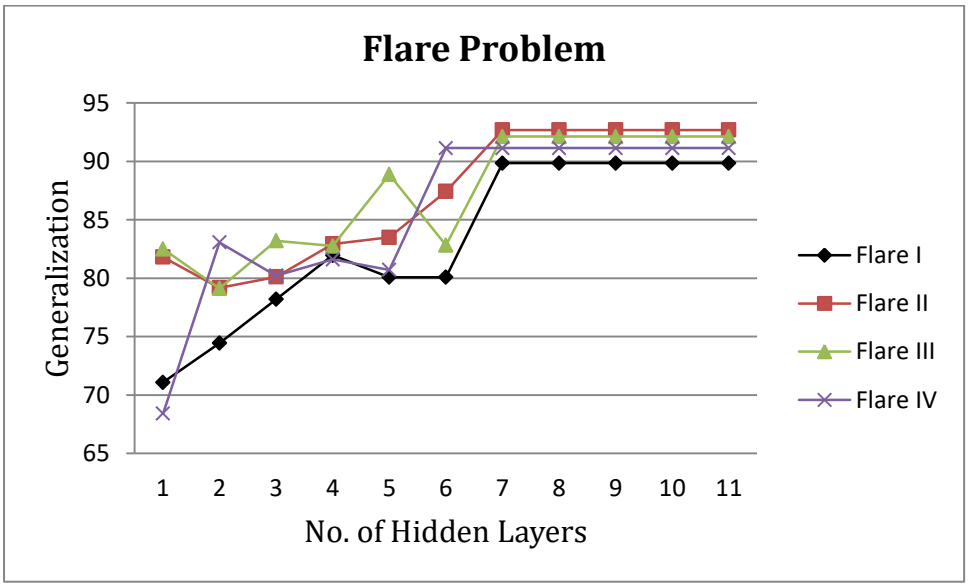


Figure 6.5: Generalization comparison of Flare problems

Apart the above, efficiency of the network can be described by the time cost of the training and this reflects by the number of iterations. Especially when it is difficult to determine the performance by the generalization, training time could be considered. For example, in the Knowledge I problem all the configurations show 100% performance for testing sets. Thus, in this case the time cost will be used to determine the best architecture. The Figure 6.6 points up the number of epochs taken to train the different sizes of networks. According to this result 3 layer network shows the best performance as it could train by only 400 iterations while others needs more than thousand of that.

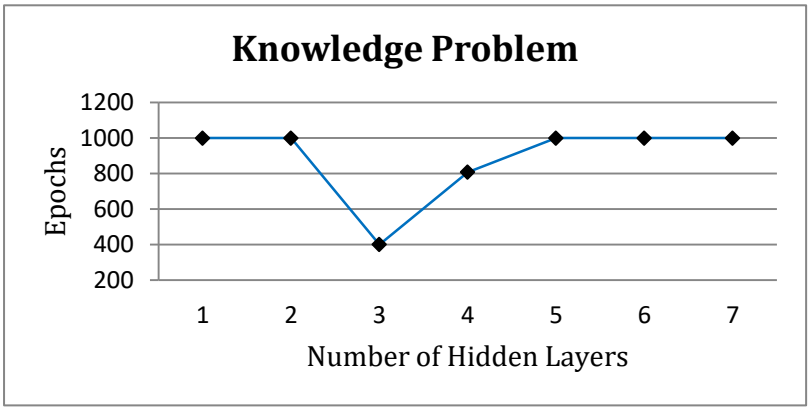


Figure 6.6: No. of epochs take to train Knowledge I Problem

The Knowledge I problem shows 100% for all the configurations with 303 data samples (75% samples from the whole set). However, while reducing the size of the training set generalization for single hidden layer networks decline, nevertheless they reach rapidly to their maximum generalization 100% and retain that for higher number of hidden layer architectures (Figure 6.7).

As in other benchmark problems, generalization power is not a good measure for Knowledge problems as they show almost 100% performances for each configuration. Thus, the training time and number of epochs could be taken into account to determine the best architecture. By comparing the training time, it is observed that the single hidden layer network in Knowledge I problem takes 131.13 seconds to train the network while 3 layer network trains within 123.43 seconds.

Both of these configurations give 100% performance but 3 layer network is considered to be the best architecture as it is more economic.

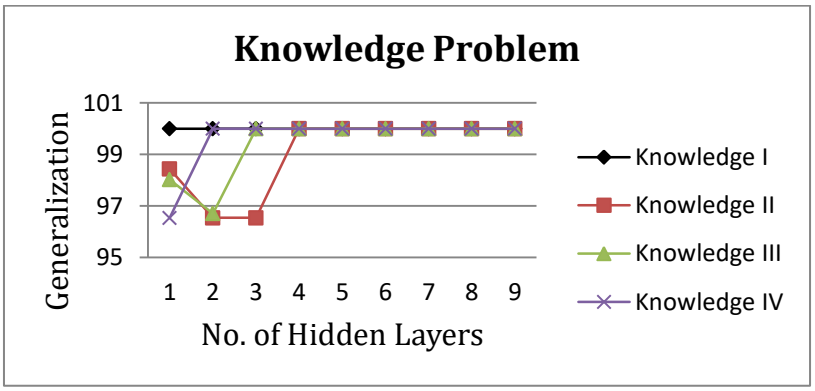


Figure 6.7: Generalization comparison of Knowledge problems

All the above data sets show that generalization improves while increasing the number of hidden layers. On the other words, single hidden layer network is not a good solution for those problems. Further, data sets of the same domain show different peak values for different sizes of training sample. Most probably, training sets with large data sets show better performance.

In the next section we discuss the experiments done in determining the number of layers by using the peak search algorithm. In order to find this, all the 34 data sets given in the Table 6.1 were used.

6.3.2 Determining the number of hidden layers

The process of determining the number of hidden layers in the most appropriate solution for the given problem was started with a network, trained by the backpropagation algorithm for an arbitrary large number of hidden layers *H* (say). The Table 6.6 shows the number of hidden layers, generalization and the way of choosing hidden neuronal structure of all the input data sets. The neuronal structure of each architecture was decided by the user who involved to this experiment. We conducted all the experiments based on the hypothesis that, “any network with large number of hidden layers can be reduced to a smaller sized network without

degrading its performance." The *Peak Search Algorithm* (PSA) described in the previous chapter 5 was used to determine the number of hidden layers of each of the network. To illustrate the procedure, the data set of cancer I and Flare I were chosen and their process of determining the number of hidden layers is described as follows. The process of achieving the number of hidden layers of all the other data sets is described in the Appendix 2.

Cancer I Problem

The breast cancer I contains 699 data samples and 525 were chosen for training purpose. The process started with a network with 20 hidden layers trained by the backpropagation algorithm. The number of hidden neurons in each layer is assumed to be same and this number is

$$\left[\frac{525}{20}\right] = 26$$

Hence, the total number of hidden neurons is $26 \times 20 = 520$.

174 data sets were used in the testing purpose. The generalization or the percentage of correct responses of the initial network with 20 layers $g_{20} = 62.6$. g_1 was observed as 97.7. As $g_1 \neq 100$ continued the procedure. Initially, $L = 1$ and $R = 20$. Then

$$m = \left[\frac{1 + 20}{2}\right] = 10$$

g_{10} was computed as 62.6. It agrees with the statement Case I: $g_L > g_m \geq g_R$ in the section 5.2. Hence, peak lies between 1 and 10. Then removed the interval (10,20] and replaced R by 10 and new

$$m = \left[\frac{1 + 10}{2}\right] = 5$$

and $g_5 = 99.4$.

Therefore, $g_1 < g_5 > g_{10}$. According to the Case III: $g_L \leq g_m \geq g_R$, the peak can lie in interval $[1, 5]$ or [5, 10]. Then compute the mid points of two intervals m_1 and m_2 such that,

$$m_1 = \left[\frac{1+5}{2}\right] = 3 \quad \text{and} \quad m_2 = \left[\frac{5+10}{2}\right] = 7\,.$$

It observed that

$$g_3 = 98.9 \text{ and } g_7 = 94.3.$$

Then peak lies in the interval, where $\max\{g_1, g_3, g_5, g_7, g_{10}\} = g_5$ lies.

Therefore, peak lies in the interval $[3, 7]$ and since, $g_3 < g_5 > g_7$, the peak can lie in interval $[3, 5]$ or [5, 7]. Then compute the mid points of two intervals m_1 and m_2 such that,

$$m_1 = \left[\frac{3+5}{2}\right] = 4 \quad \text{and} \quad m_2 = \left[\frac{5+7}{2}\right] = 6\,.$$

Then peak lies in the interval, where $\max\{g_3, g_4, g_5, g_6, g_7\} = g_4$ lies.

That is in the interval [3,5] and since $3, 4, 5$ are consecutive numbers, the number of layers which gives highest generalization is

$$\{j: g_j = \max\{g_3, g_4, g_5\}\} = 4$$

(Note: In this case $g_4 = g_5$, so that it chooses smaller network as the optimimum solution.)

To achieve the peak value of the above Cancer I problem, only 7 networks were trained. That is networks with layers 1,3,4,5,6,7 and 10. The whole process of determining the number of hidden layers which gives the best architecture in Cancer problem by the peak search algorithm is depicted by the Figure 6.8.

Table 6.6: Details of Initial networks

	Data Set	No. of Hidden Layers	Total no. of hidden neurons	Type of hidden neurons	Generalization (%)
1	Banknote	12	1014	Ascending	100
2	Cancer I	20	520	Equal	62.6
3	Cancer II	20	360	Equal	65.6
4	Cancer III	20	180	Equal	65.8
5	Cancer IV	20	20	Equal	65.4
6	Card I	12	516	Equal	53.4
7	Card II	12	348	Equal	55.1
8	Card III	12	168	Equal	54.8
9	Card IV	12	12	Equal	55.3
10	Cardio	12	1560	Ascending	58.9
11	Climate	10	400	Equal	92.6
12	Diabetes I	15	570	Equal	63.5
13	Diabetes II	15	390	Equal	63.2
14	Diabetes III	15	195	Equal	63.7
15	Diabetes IV	15	45	Equal	65.1
16	Flare I	12	780	Ascending	89.8
17	Flare II	12	546	Ascending	92.7
18	Flare III	12	234	Ascending	92.1
19	Flare IV	12	78	Ascending	91.1
20	Glass	10	160	Equal	100
21	Heberman	10	230	Equal	71.4
22	Iris	12	108	Equal	100
23	Knowledge I	12	300	Equal	100
24	Knowledge II	12	204	Equal	100
25	Knowledge III	12	96	Equal	100
26	Knowledge IV	12	24	Equal	100
27	Monk’s 1	10	120	Equal	100
28	Monk’s 2	12	169	Arbitrary	67.1
29	Monk’s 3	12	120	Equal	100
30	Seeds	10	130	Equal	0.0
31	Statlog	15	195	Equal	55.9
32	Thyroid	12	156	Equal	66.7
33	Tissue	10	80	Equal	100
34	Yeast	12	1092	Ascending	100

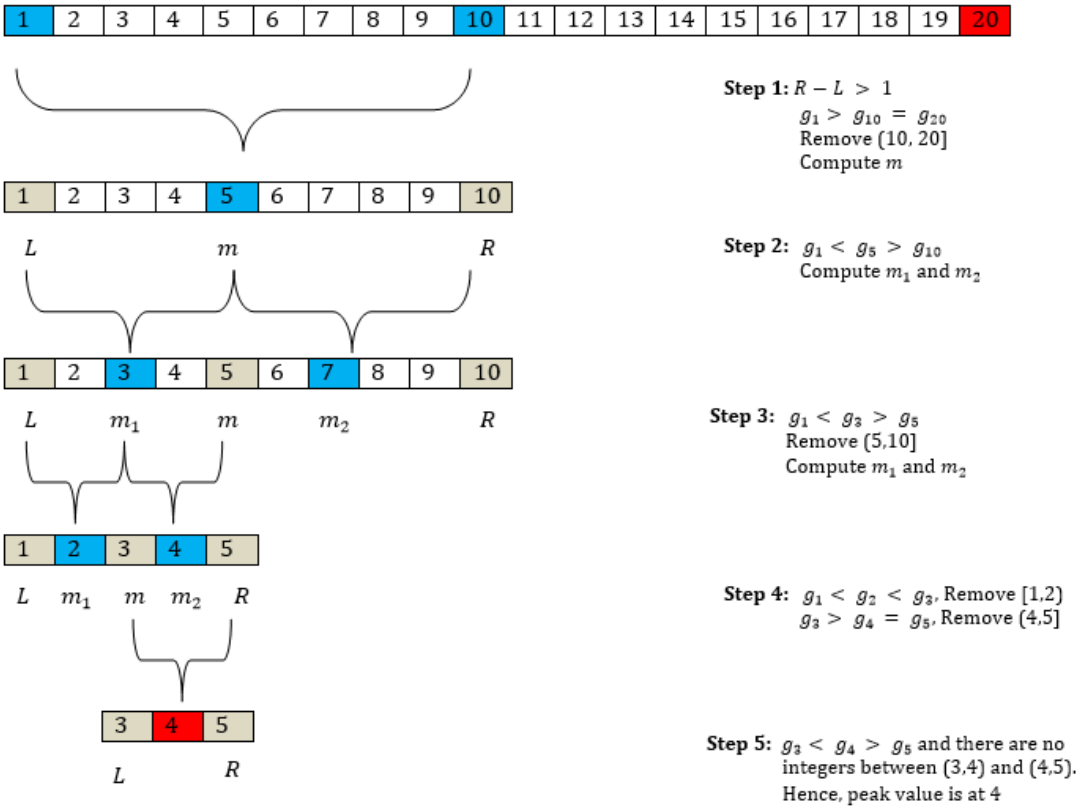


Figure 6.8: Determining number of hidden layers in Cancer I

Flare I Problem

The Flare problem consists of 800 training patterns and a network was created with 12 layers, where neurons are distributed ascending order as shown in the following table. The total number of hidden neurons was chosen as a number close to the number of training patterns. Hence, as given in the table Table 6.5 the total number of hidden neurons was taken as 780. Initially set $L = 1$ and $R = 12$. The generalization of the network of 12 hidden layers was given as

$$g_{12} = 89.4$$

The generalization of one hidden layer network is computed as

$$g_1 = 71.1$$

Table 6.7: Distribution of hidden neurons in Flare I data set

No. of hidden Layers	1	2	3	4	5	6	7	8	9	10	11	12
No. of hidden neurons	10	20	30	40	50	60	70	80	90	100	110	120

As $g_1 \neq 100$ and $L - R > 1$, the process continued and m is computed as

$$m = \left[\frac{1+12}{2}\right] = 6$$

$g_6 = 80.1$ and therefore, $g_1 < g_6 < g_{12}$.

Hence, according to the case II in section 5.5.2 the peak lies in [6,12] and the interval [1,6) can be removed and replace L by 6. The new m is computed as

$$m = \left[\frac{6+12}{2}\right] = 9$$

$g_9 = 89.4$ and $g_6 < g_9 = g_{12}$, by p_3 in section 5.5.2, m_1 and m_2 are computed.

$$m_1 = \left[\frac{6+9}{2}\right] = 7 \text{ and } m_2 = \left[\frac{9+12}{2}\right] = 10$$

$g_7 = g_{10} = 89.4$, Hence, $g_6 < g_7 = g_9 = g_{10} = g_{10}$. Now 6 can be removed and thus L is replaced by 7. However, now this process has come to its worst scenario. That is as all the corresponding values are equal, generalization of each possible value of interval should be computed. However, it shows

$$g_8 = g_{11} = 89.4$$

Now as every value in the interval [7,12] show generalization 89.4, the number of hidden layers in most appropriate network is taken as the least number in the corresponding interval, i.e, number of hidden layers in the most appropriate network is 7.

The process is described in the following Figure 6.9

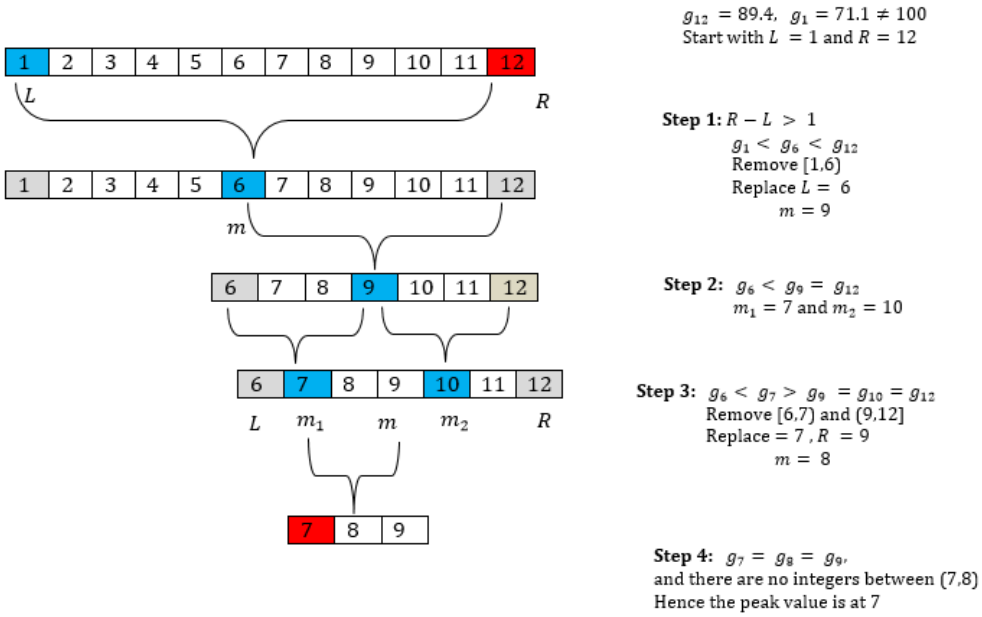


Figure 6.9: Determining number of hidden layers in Flare I problem

The below Table 6.8 shows the results obtained by the peak search algorithm. It clearly shows that multilayer architectures of ANNs give better generalization than the single hidden layer networks. In these results only the 'Yeast' data set reaches to its best performance with single hidden layer network. All the other problems needed two or more hidden layers. However, about 70% data sets reach to their maximum generalization within 5 hidden layers.

Table 6.8: Details of New architecture obtained by the Peak Search Algorithms

	Data Set	No. of Hidden Layers	Total no. of hidden neurons	Amount of neuron reduction (%)	Generalization %	Generalization Improvement (%)
1	Banknote	2	172	83.3	100	0.0
2	Cancer I	4	104	80.0	99.4	36.8
3	Cancer II	4	72	80.0	95.7	30.1
4	Cancer III	4	36	80.0	95.0	29.2
5	Cancer IV	2	2	90.0	65.4	0.0
6	Card I	6	258	50.0	87.2	33.7
7	Card II	6	145	58.3	85.8	31.0
8	Card III	5	70	58.3	85.3	30.4
9	Card IV	3	3	75.0	65.7	10.4
10	Cardio	4	200	87.2	72.9	14.0
11	Climate	2	80	80.0	97.8	5.2
12	Diabetes I	3	114	80.0	82.8	19.3
13	Diabetes II	5	130	66.7	80.5	17.2
14	Diabetes III	4	52	73.3	78.3	14.6
15	Diabetes IV	3	9	80.0	77.5	12.4
16	Flare I	7	280	64.1	89.8	0.0
17	Flare II	7	196	64.1	92.7	0.0
18	Flare III	7	84	64.1	92.1	0.0
19	Flare IV	6	21	73.1	91.1	0.0
20	Glass	4	64	60.0	100	0.0
21	Heberman	6	138	40.0	77.9	6.5
22	Iris	4	36	66.7	100	0.0
23	Knowledge I	3	75	75.0	100	0.0
24	Knowledge II	4	68	66.7	100	0.0
25	Knowledge III	3	24	75.0	100	0.0
26	Knowledge IV	2	4	83.3	100	0.0
27	Monk’s 1	7	84	30.0	100	0.0
28	Monk’s 2	4	78	53.9	86.8	19.1
29	Monk’s 3	6	60	50.0	100	0.0
30	Seeds	4	52	60.0	90.6	90.6
31	Statlog	8	104	46.7	100	44.1
32	Thyroid	4	52	66.7	94.4	27.8
33	Tissue	5	40	50.0	100	0.0
34	Yeast	1	14	91.7	100	0.0

6.3.3 Correlation between the sum of Delta values and the output error

When the number of hidden layers is fixed, the network starts the pruning process. The one of the main factors considered on pruning was the correlation between the sum of delta values of each layer h and the output error E which is denoted by $\gamma_{h,E}$. Empirical results show that, most probably the value of $\gamma_{h,E}$ for the last hidden layer shows a negative value. In this section we discuss about this correlation for some of the above benchmark applications.

6.3.4 Correlation between the sum of Delta values and the output error

When the number of hidden layers is fixed, the network starts the pruning process. The one of the main factors considered on pruning was the correlation between the sum of delta values of each layer h and the output error E which is denoted by $\gamma_{h,E}$. Empirical results show that, most probably the value of $\gamma_{h,E}$ for the last hidden layer shows a negative value. In this section we discuss about this correlation for some of the above benchmark applications.

Cancer I problem shows its better performance for 4 hidden layer network. So that, we consider this architecture to prune unnecessary neurons. The Table 6.9 and Figure 6.10 depict the corresponding results of correlations after the first iteration of this Problem. These results imply that there is a significant relation between the sum of delta values and the output error and hence, this result is able to use in identifying the neurons such that removing those neurons, output error will decrease, i.e, by eliminating such neurons generalization power of the ANN would be increased.

Table 6.9: Correlations of the Cancer I data set

	$\gamma_{1,E}$	$\gamma_{2,E}$	$\gamma_{3,E}$	$\gamma_{4,E}$
Correlation	0.8913	-0.8853	0.8332	-0.8771

This figure shows that although summations of delta values are different, all the layers have same pattern, but it alternates the sign at each layer. A similar results can be observed in Cancer II, III and IV, which depict in the Table 6.11.

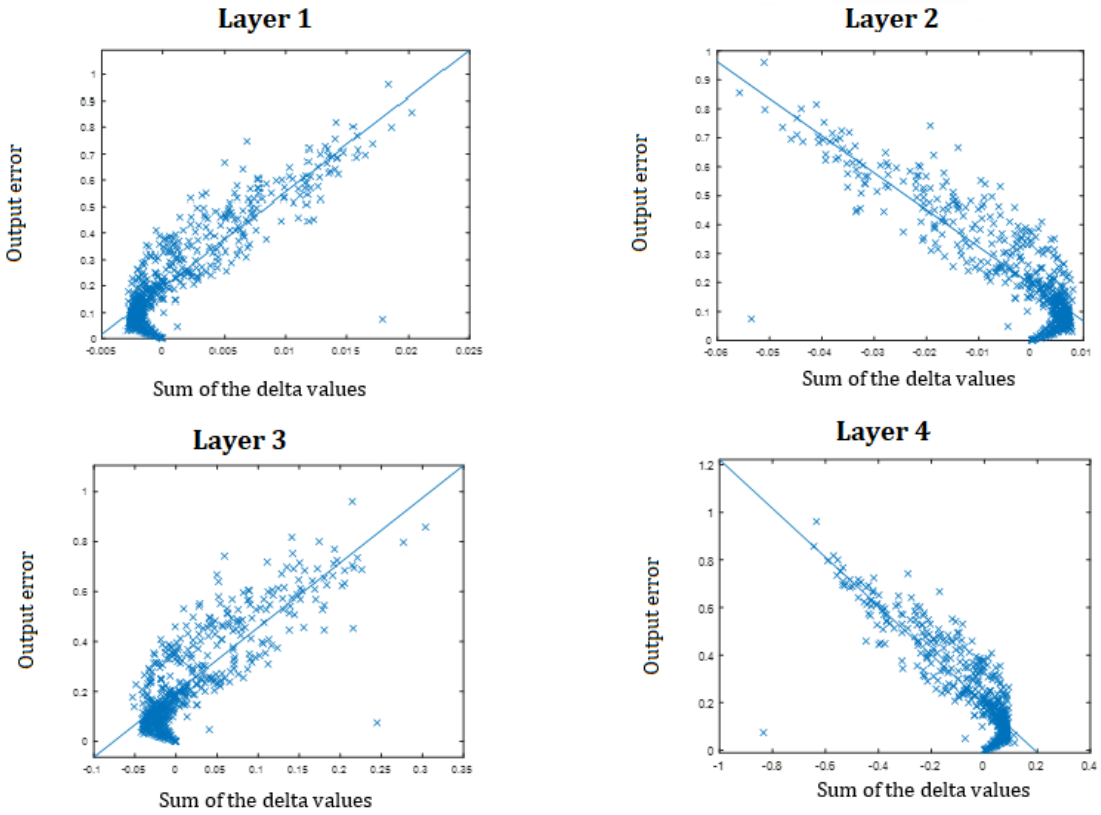


Figure 6.10: Correlations of Cancer I problem

The Card I problem shows its highest performance when there are 6 layers in the network and sum of the delta values in all 6 layers show significant correlation with the output error (Table 6.10 and Figure 6.11) The two problems Card II and Card III reach to their peaks with 6 and 5 hidden layers respectively and all these sets show significant correlation in $\gamma_{h,E}$ for each hidden layer h.

Table 6.10: Correlations of the Card I data set

	$\gamma_{1,E}$	$\gamma_{2,E}$	$\gamma_{3,E}$	$\gamma_{4,E}$	$\gamma_{5,E}$	$\gamma_{6,E}$
Correlation	0.8046	-0.8047	0.8088	-0.7993	0.8138	-0.8160

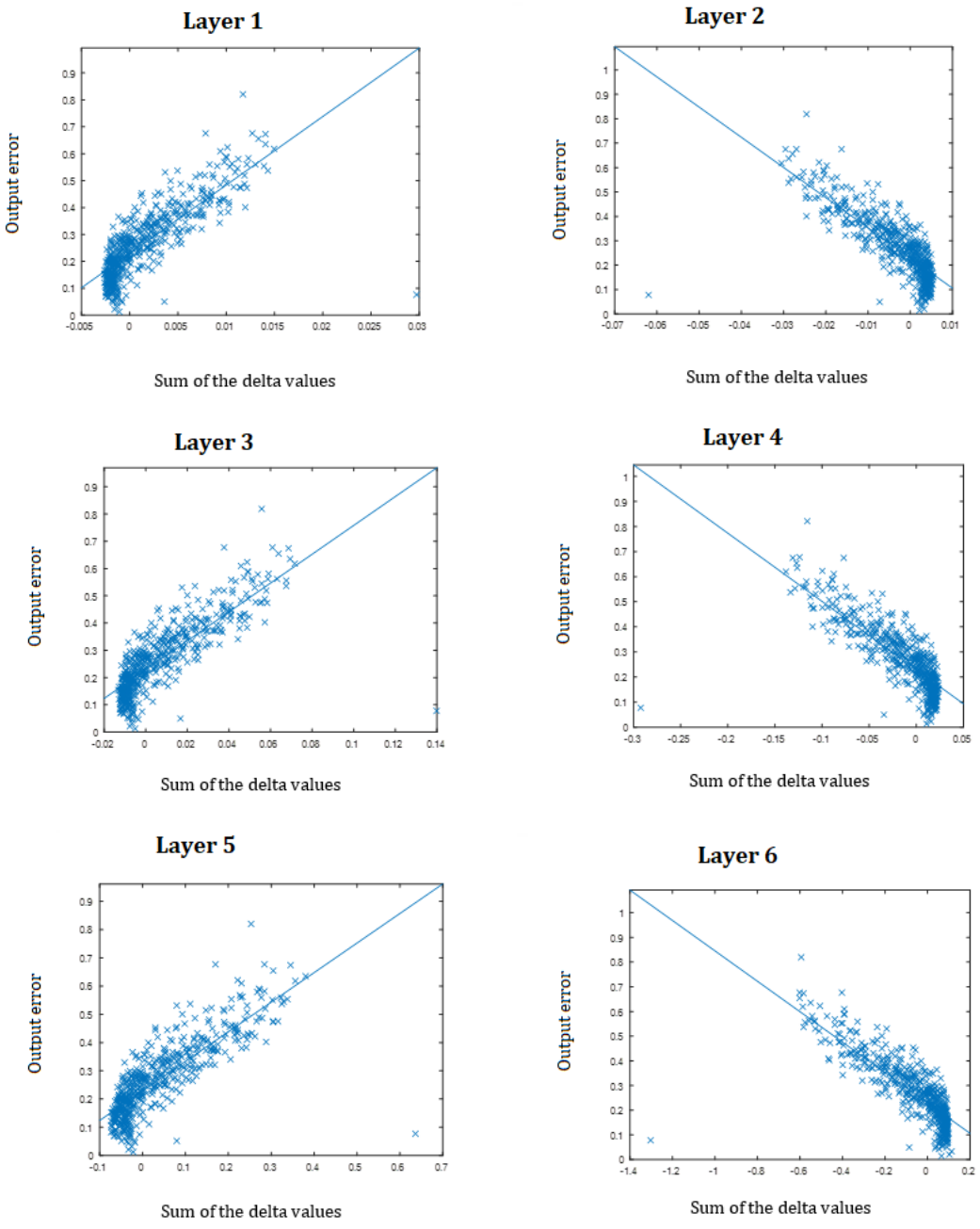


Figure 6.11: Correlations of Card I problem

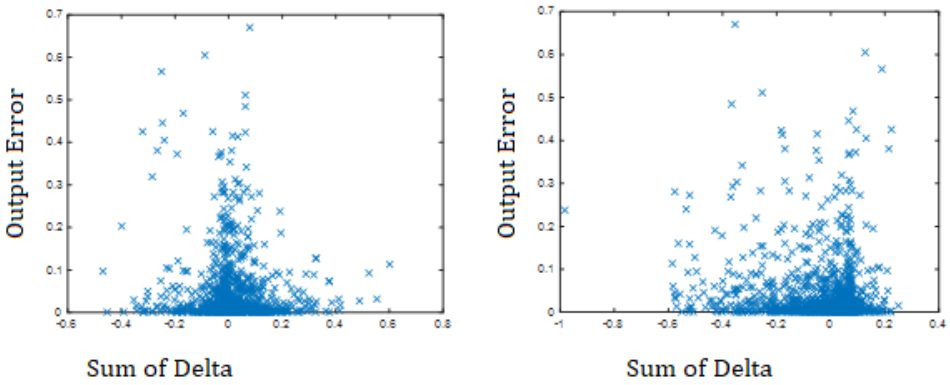


Figure 6.12: Correlation of the Banknote problem

The correlations of all such data sets are shown in the following Table 6.11. This shows that most of the datasets show considerable correlation $\gamma_{h,E}$. That is more than 78% show significant correlation. Other than banknote, every other network's changes its sign of $\gamma_{h,E}$ at each layer. Therefore, in these networks delta values and their signs are taken into consideration when removing the neurons from the hidden layers.

Banknote, Flare I, II, III, IV and Statlog problems have poor correlations. In addition, unlike the other networks, the correlations of both the layers of Banknote have the same (negative) sign. In these problems, sign of delta values are not considered while eliminating neurons. So that, in these cases in order to improve the performance, neurons which have positive or negative infinitesimal delta values are remove from the network.

Table 6.11: Correlation between sum of delta values and output error

	Data Set	Number of Hidden Layers								
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
1	Banknote	-0.0657	-0.0871							
2	Cancer I	0.8913	-0.8853	0.8332	-0.8771					
3	Cancer II	0.9396	-0.9384	0.9242	-0.9362					
4	Cancer III	0.9575	-0.9587	0.9475	-0.9672					
5	Cancer IV	0.6200	-0.6256							
6	Card 1	0.8046	-0.8047	0.8088	-0.7993	0.8138	-0.8160			
7	Card II	-0.8478	0.8487	-0.8509	0.8736	-0.8662				
8	Card III	-0.9660	0.9642	-0.9761	0.9743	-0.9747				
9	Cardio	0.6685	-0.6679	0.6810	-0.6776					
10	Climate	0.8839	-0.9024							
11	Diabetes 1	-0.8324	0.8314	-0.8327						
12	Diabetes 2	-0.8156	0.8332	-0.8006	0.8397	-0.7755				
13	Diabetes III	0.8650	-0.8626	0.8602	-0.9016					
14	Diabetes IV	-0.9694	0.9705	-0.9699						
15	Flare 1	0.1470	-0.1522	0.1520	-0.1512	0.1512	-0.1143	0.1284		
16	Flare II	0.1491	-0.1540	0.1543	-0.1528	0.1587	-0.1201	0.1548		
17	Flare III	0.2913	-0.3149	0.3161	-0.3128	0.3118	-0.2795	.2786		
18	Flare IV	-0.5623	0.5648	-0.5707	0.5676	-0.5738	0.5209			
19	Glass	0.8255	-0.8328	0.8216	-0.8608					
20	Heberman	0.8595	-0.8595	0.8629	-0.8650	0.8714	-.8867			
21	Iris	0.8448	-0.7906	0.8760	-0.9691					
22	Knowledge I	-0.6889	0.6613	-0.7366						
23	Knowledge II	0.8519	-0.8481	0.8646	-0.8422					
24	Knowledge III	-0.9213	0.9298	-0.9543						
25	Knowledge IV	0.6327	-0.7295							
26	Monk's 1	-0.8592	0.8607	-0.8568	0.8604	-0.8720	0.8903	-0.9079		
27	Monk's 2	-0.7754	0.7738	-0.7744	0.7671					
28	Monk's 3	0.8233	-0.8222	0.8249	-0.8258	0.8424	-0.8415			
29	Seeds	-0.7893	0.8442	-0.8501	0.8512	-0.8513	0.8741	-0.8843		
30	Statlog	0.1445	-0.1432	0.1408	-0.1456	0.1602	-0.1587	0.1814	-0.0878	
31	Thyroid	0.8863	-0.8863	0.8914	-0.8846					
32	Tissue	-0.6813	0.6866	-0.6270	0.5226	-0.8259				
33	Yeast	-0.7964								

6.3.5 Removing neurons

In order to attain the optimal architecture, the unimportant neurons in the error decay process must be removed from the network architecture. In previous chapter, it was explained the procedure of eliminating neurons.

Basically, we use the correlation coefficient shown in the above Table 6.11 to determine the removable neurons. If there is significant positive correlation value for $\gamma_{h,E}$, the error could be minimized by removing the neurons with positive delta values. In this case, the neurons, which show infinitesimal positive for every training sample in the data set will be identified as removable neurons. In contrast, when there is a considerable negative correlation, neurons, which show the negative delta value whose absolute value is very close to zero for each training sample in the set will be removed.

While removing the unnecessary neurons from the network it merges the neurons with similar weight vectors as described in the previous section 5.5.6. The Table 6.12 below depicts the experimental results and it compares the resultant network with the network obtained by the PSA and the initial network.

The Cancer I problem started with 20 hidden layers and 520 hidden neurons divided equally among the layers. The generalization of initial network was 62.6%. The network obtained by PSA contained 4 layers and 104 hidden neurons. (Each layer consisted of 26 neurons). This network showed generalization 99.4%. By using delta values of hidden neurons it was identified 10 unnecessary neurons (3, 4, 2 and 1 from each layer) and removed them from the network. Hence, the optimal solution of this problem contains only 4 layers and 94 neurons. The hidden layer architecture was 23 – 22 – 24 – 25. The proposed method has reduced 9.6% percent of hidden neurons from the network obtained by PSA. But this stage generalization had not increased. Comparing with the original network, it has reduced 81.9% neurons and improved generalization by 36.8%.

Similar results can be seen on the other data sets such as Cancer II, Card I, II, Diabetes I-III, Heberman and Seeds etc. The data sets such as Banknote, Flare I-IV and Glass, Iris have initially reached to their maximum generalization. Hence, by new algorithm it has lessened only the size of the network, which will enable to reduce test time. However, if we use very small data set as Cancer IV and Diabetes IV we cannot expect any improvement and process will last with the same network structure and the generalization.

Table 6.12: Neural network architectures obtained by the new model

	Data Set	No. of Hidden Layers	Total no. of hidden neurons	Reduction of neurons		Generalization %	Improvement	
				Relative to Phase I	Relative to the original network		Relative to phase I	Relative to the original network
1	Banknote	2	26	33.3	97.4	100	0.0	0.0
2	Cancer I	4	94	9.6	81.9	99.4	0.0	36.8
3	Cancer II	4	38	47.2	89.4	98.6	0.9	32.1
4	Cancer III	4	31	13.9	82.8	95	0.0	29.2
5	Cancer IV	2	2	0.0	90.0	65.4	0.0	0.0
6	Card I	6	239	7.4	53.7	87.8	0.6	34.3
7	Card II	6	110	24.1	68.4	84.4	0.6	31.3
8	Card III	5	70	0.0	58.3	85.3	0.0	30.4
	Card IV	3	3	0.0	75.0	65.7	0.0	10.4
9	Cardio	4	160	20.0	89.7	77.0	4.1	18.1
10	Climate	2	64	20	84.0	97.8	0.0	5.2
11	Diabetes I	3	100	12.3	82.5	84.4	1.6	20.9
12	Diabetes II	5	130	0.0	66.7	80.5	0.0	17.2
13	Diabetes III	4	46	11.5	76.4	78.6	0.3	14.9
14	Diabetes IV	3	9	0.0	80.0	77.5	0.0	12.4
15	Flare I	7	23	91.8	97.0	89.8	0.0	0.0
16	Flare II	7	150	23.5	72.5	92.7	0.0	0.0
17	Flare III	7	80	4.8	65.8	92.1	0.0	0.0
18	Flare IV	6	21	0.0	73.1	91.1	0.0	0.0
19	Glass	4	60	6.3	62.5	100	0.0	0.0
20	Heberman	6	101	26.8	56.1	79.2	1.3	7.8
21	Iris	4	18	50.0	83.3	100	0.0	0.0
22	Knowledge I	3	42	44	86	100	0.0	0.0
23	Knowledge II	4	39	42.6	80.9	100	0.0	0.0
24	Knowledge III	3	24	0.0	75	100	0.0	0.0
25	Knowledge IV	2	4	0.0	83.3	100	0.0	0.0
26	Monk's 1	7	60	28.6	50.0	100	0.0	0.0
27	Monk's 2	4	67	14.1	60.4	87.1	0.3	20.0
28	Monk's 3	6	47	21.7	60.8	100	0.0	0.0
29	Seeds	7	42	19.2	67.7	93.2	2.6	93.2
30	Statlog	8	68	34.6	65.1	100	0.0	0.0
31	Thyroid	4	38	26.9	75.6	94.4	0.0	27.8
32	Tissue	5	33	17.5	58.8	100	0.0	0.0
33	Yeast	1	9	35.7	99.1	100	0.0	0.0

6.6 Data Analysis

The summary of the hidden layers that gives the best performance is depicted in the Figure 6.13. According to the results all the networks show their best architecture within 8 hidden layers and more than 67% of them reach to the best architecture within 5 hidden layers. Moreover, it implies that the single hidden layer network is not the best solution for most of the problems, because only one data set (3%) shows its best performance with a single hidden layer ANN architecture. According to these results the average of the hidden layers in the most appropriate network is 4. However, very few data sets (about 3%) need more than 7 layers to show their best performance.

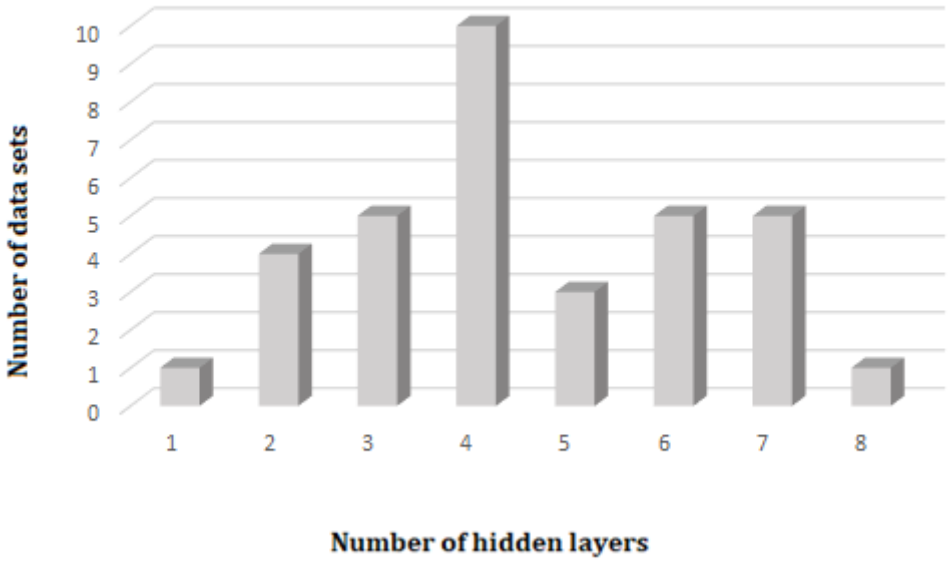


Figure 6.13: Summary of Peak search algorithm

The number of hidden layers, which gives the best performance for each of the problem is given in the Table 6.8. It implies that the given method reduces network to a small one relative to the initial network. For example, in climate problem, initial network consists of 10 hidden layers with 400 neurons. But the resultant network contains only 64 neurons which are distributed in two layers. Comparing with the original network 84% of neurons have been removed while increasing the

generalization from 92.6% to 97.8%. In some problems such as Cancer, Diabetes and Flare, there is a huge reduction of neurons while increasing the generalization.

The data set Yeast seems to train easily and they give 100% correct responses for single hidden layer networks. In this problem initial network contains 10 layers, which gives 100% generalization. The PSA firstly compute the generalization for a single hidden layer network and it was observed as 100%. Then the process stopped after computing the performance of single hidden layer networks. The Banknote Authentication set showed zero correct mappings for the single hidden layer network. Hence, procedure of PSA continued, and it was observed that each ANN architecture with higher number of layers gives 100% correct mappings. Thus, always right half of the interval removed and process ended with two-layer network. Unlike the other problems, Seeds data set starts with zero performance with 10 hidden layers. However, PSA shows that this set gives its highest generalization 90.6% at 7 hidden layers.

The data sets of Flare I – IV, have been shown their maximum generalization for the initially input network architectures. Therefore, once the peak search results achieve this level, it retains there. Similar results show in the data sets of Glass, Iris, Knowledge I – IV, Monk's 1, 3 and Tissue with 100% of their highest performance. Thus, in each of these problems, the network with the smallest number that shows this highest performance is considered as the best architecture. In these problems, even though there is no improvement of generalization by comparing with original network, the performance is upgraded due to the training time is reduced by making the network smaller.

The results obtained by the above results verified the assumption that 'any trained network can be reduced to a smaller sized network without degrading its performance'. Because according the experimental results, all 34 data sets have been reduced their sizes by at least 25% while improving the performance. The Figure 6.14 shows that, 17 samples that are 50% percent of the data sets remove more than 75% of the hidden neurons while achieving the optimal solution. Another

47% (16 samples) were able to remove 51% -75% neurons and only 1 set (3%) reduces its size by 26% - 50%.

In addition, among the 17 data sets which reduce their size over 75%, two problems have been increased the generalization between 10% - 30%. Another 6 problems show 0% - 10% improvement of the performance. However, 7 data sets do not show any upgrading of their generalization as they have started with their highest generalization. For example, Flare I and Yeast problem. The initial configuration of these sets shows their maximum generalization 89.8% and 100% respectively. During the elimination process they reduce their hidden layer structure and make the data sets smaller. But their generalization retain at the same value.

The Figure 6.15 explains that how the generalization has been improved in the above 34 data sets from their initial configuration. It shows that 9 data sets (26% of data) improve their generalization above 20%. Another 6 and 2 data sets upgrade their performance between 10% - 20% and 0%-10% respectively. Nevertheless, 17 data sets (50%) begin the procedure with their maximum generalization and hence, they do not show any improvement of their generalization.

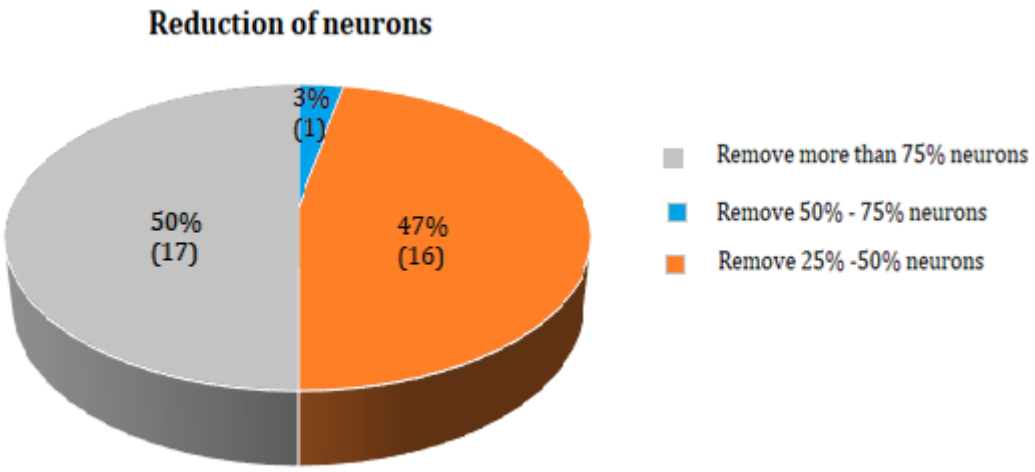


Figure 6.14: Reduction of neurons from the initial network configuration

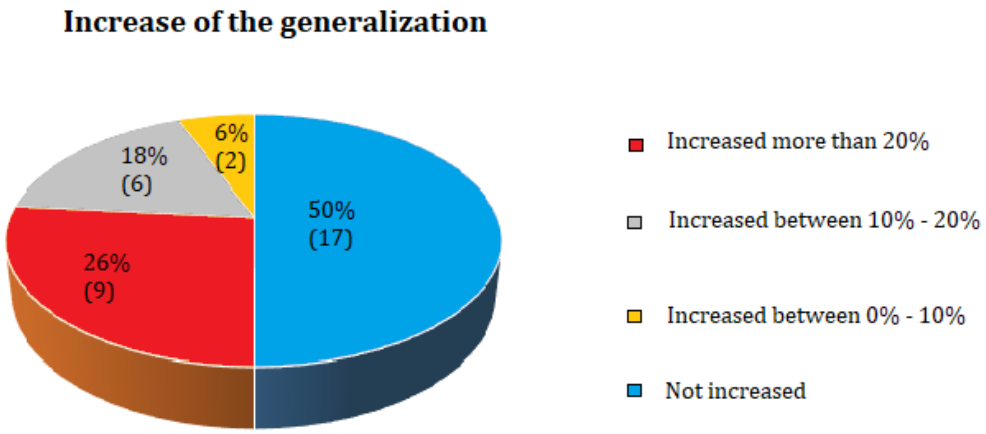


Figure 6.15: Increase of the generalization comparing with the initial network

6.4 Comparison with Existing Method

It was discussed in the Chapter 2 that, OBS and OBD are the two well-known methods in pruning, but they need extra computations for accommodate the Hessian matrix. The efficiency of Skeletonization method [73] is low as it always choose the neuron with least salience at each iteration. In addition, the productivity of MBP methods are not up to the standards, because they concern only on the magnitude of the connection weights in pruning [64]. The method of N2PS perform better than several other methods. However, it has restricted only for single hidden layer networks. A constructive method called rule extraction from ANNs (REANN) is also give significant results in medical diagnose problems, nevertheless it has not discussed the solution for large set of output vectors.

The proposed method (PSDV) does not have complex computations and hence, it has easy accommodation in any large data set. On the other hand, while applying the peak search algorithm it removes all the neurons in the particular layer. Moreover, in the second part it recognizes all the removable neurons at once to trim the ANN architecture. Hence, rather than eliminating one neuron at a time PSDV

removes cluster of neurons at once. Therefore, relative to other approaches, the proposed method reaches to the optimal architecture of hidden layers faster.

The following Table 6.13 depicts the comparison of the generalization PSDV with some existing methods. '-'indicates that information is not available. It implies that this method gives best generalization relative to many other methods. Hence, PSDV is a very promising approach in determining the hidden layer architecture in ANNs.

Table 6.13: Generalization of PSDV and other existing methods

Method / Dataset	VNP	N2PS	OBD	OBS	MBP	REANN	PSDV
Cancer	97.8	97.1	92.5	90	94.2	96.28	99.4
Diabetes	70.3	69.1	68.6	65.4	68.9	76.56	84.4
Iris	97.7	98.7	98	98	98	-	100

The following Figure 6.16 illustrates the results of the above table. It is clear that the proposed method PSDV has achieves highest accuracy in all 3 problems Cancer, Diabetes and Iris. In the other entire methods network consist only one single layer. Thus, it is obvious that generalization of network can be improved by adding more layers to the network.

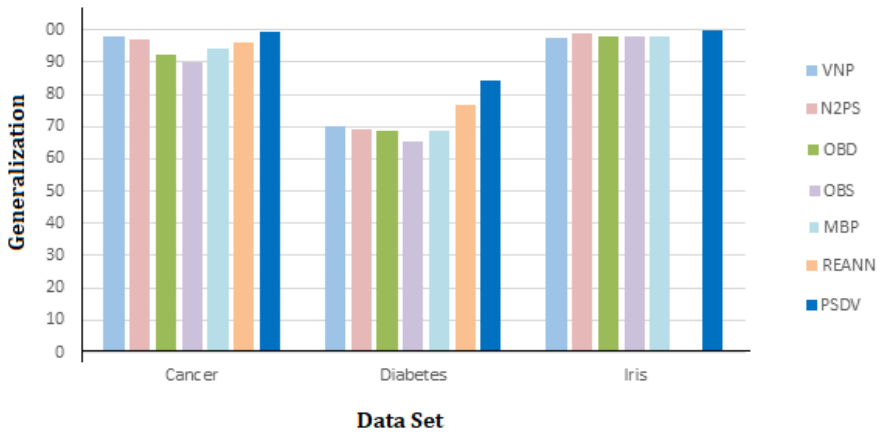


Figure 6.16: Comparison of PSDV with the other existing methods

6.5 Summary

This chapter emphasizes the experimental results carried out to prove the hypothesis that any large network can be pruned to a smaller one by removing the hidden layers and hidden neurons without degrading its performance. 34 data sets were analyzed. From the entire network we were able to get a smaller network that the resultant network show same or better performance. Experimental results show that the single hidden layer network is not a good architecture for ANNs and some networks need 8 – 9 layers to give a considerable solution. The significant number of data sets reach to their best architecture within 4 hidden layers and probability of having such network is 0.56. At the end it compared with the existing methods and observed that the novel approach described here always gives best generalization. The next chapter will discuss how to extend the proposed PSDV method for deep leaning networks.

CHAPTER 7

USING PSDV FOR DEEP NEURAL NETWORKS

7.1 Introduction

The previous chapter discussed the evaluation of newly introduced Peak Search and Delta Value (PSDV) method. It was observed that any large, trained network could be pruned to a more efficient network with lesser number of hidden layers and hidden neurons. Experimental results implied that, generally multilayered feedforward networks show better generalization than shallow networks.

This chapter discusses how the PSDV method could be applied for deep neural networks. First, it introduces basic concepts of deep neural networks followed by the two widely used deep neural networks namely Convolutional Neural Networks and Deep Belief Networks. Next, it will discuss how to implement the proposed PSDV method to very common two deep nets namely, convolutional networks and deep belief networks.

7.2 Preamble to Deep Neural Networks

Deep learning neural networks that refer the artificial neural networks which consist of large number of hidden neurons arranged in several layers with different levels of abstraction [177], [178]. The concept of deep learning algorithms was inspired by the hierarchical structures of human speech perception and production systems [179]. The networks can be trained as supervised or unsupervised manner. The techniques of the deep learning widely used in image recognition, natural language processing, transcribe speeches in to text and match new items etc. These networks can be performed without interference of human. They are able to work closer to or sometimes better than human. The conventional machine learning process is required a feature extractor that could transform the raw data, such that pixel value to image etc. into numerical vector form, which could represent an input pattern.

The deep learning methods are representative learning methods which transforms input data of one level to a more abstract level. For instance, in image recognition, the input is an array of pixel values. The first layer represents the existence or non-existence of edges at particular positions in the image. The second layer detects the patterns by spotting the particular arrangements of edges. The third layer arranges the patterns of main objects and higher order layers would detect objects as these parts [178].

Creating a simple network with one hidden layer is too simple to model complex structures on many real world problems. Because of the hardness of using multi-layers, until recently, most researches restricted their researches for only single hidden layered networks. However, for better interpretation in the human brain sensory cortex, it needs an efficient way of adapting synaptic weights of multi-layers of feature detection neurons. Because, active features in the higher layers are much better guide to activation of appropriate action than the lower level features [180]. For example, the visual system has multi-layers and it is able to generate features better than shallow networks [181]. However, in early 1990s, while backpropagation based training of multi-layered neural networks found to be difficult, deep learning showed feasible results on unsupervised learning for some extend. Since then deep learning neural networks became very popular and recent years it has won many contests in pattern recognition and machine learning [182].

Based on the mathematical operations and requirement of parameters to perform the output, deep learning networks can be divided to several categories. Some widely used architectures in deep learning [181], [183] are as follows.

- Feedforward Neural Networks (FNN)
- Convolutional Neural Networks (CNN)
- Deep Belief Networks (DBN)

Throughout the previous six chapters, the obtaining optimal architecture for feedforward neural networks by using PSDV method was discussed, and it was concluded that deep feedforward neural networks show much generalized solution

relative to shallow networks. Therefore, This chapter will discuss how to apply the PSDV method to two widely used deep neural networks namely, convolutional and deep belief neural networks.

However, deep neural network architectures are not restricted only for the above architectures and similar way it can apply the proposed method to other existing architectures too.

7.2.1 Convolutional neural networks

Like feedforward neural networks, Convolutional neural networks (CNNs) also consist of neurons with learnable connection weights. The major difference here is, they have used convolutional operation in some layers instead of matrix multiplication. The CNNs are originally designed to work with images and aim of them is to use spatial information between the pixels of image [184]. One of the first convolutional networks was LeNet-5, introduced by LeCun et al., which could classify handwritten digits. Other than the input layer, LeNet-5 contained 7 layers with trainable parameters (weights) and can be trained by backpropagation algorithm [185]. Generally, CNNs are based on discrete convolution [186].

The convolution of two functions f and g is defined as follows.

When f and g are continuous functions,

$$\int (f * g)(t) = \int_{-\infty}^{\infty} f(x)g(t-x)dx = \int_{-\infty}^{\infty} f(t-x)g(x) \qquad (7.1)$$

When the functions are discrete, integral is replaced by the sum

$$(f * g) = \sum_{m=-\infty}^{\infty} f(m)g(n-m) = \sum_{m=-\infty}^{\infty} f(n-m)g(m). \qquad (7.2)$$

There are numerous different architectures in CNNs. But they all have same basic components. Usually a CNN has 3 types of layers namely, convolution layers, pooling layers and fully connected layers.

The convolutional layers learn feature maps representation of their inputs. An input of a CNN is an element with 3 parameters, height, width and colour (R, G, B, colours). Then the input proceeds sequentially through layers and each layer transforms signals using convolutional filters (kernels) [187] as shown in the Figure 7.1. As the retinal of human eye does, convolutional operators absorb features of the image by dividing them into small slices [188].

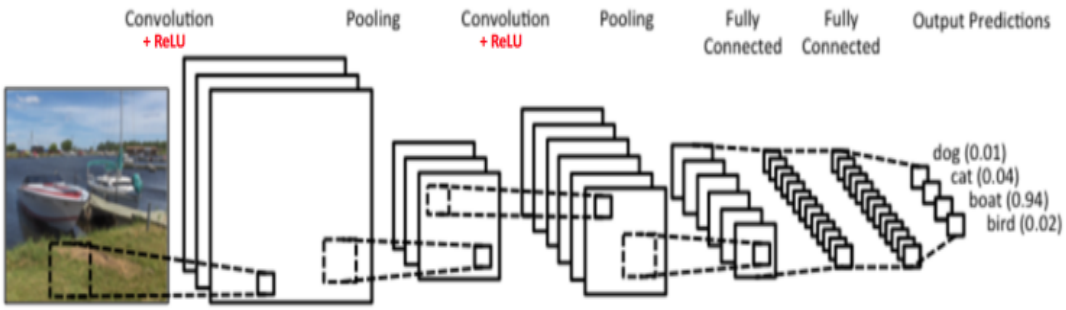


Figure 7.1: The basic architecture of a convolutional neural network

A pooling layer is placed in between two convolutional layers to achieve the shift – invariance by reducing the resolution of the feature maps. The pooling layers resize the spatiality in inputs by independently operates on every depth slice of the input. The widely used types of pooling are max pooling and average pooling. In between convolutional and pooling layers, the activation function such as, ReLU exists.

After many convolutional and pooling layers, there may some number of fully connected layers. The higher order cognitive part in the neural network is functioning through these fully connected layers. Generally, at present available networks have used maximum three hidden layers in this part. Neurons in these layers have connections to all activation in previous convolutional and pooling layers as in feedforward artificial neural networks. These activations are done with

matrix multiplication. The activation functions such as softmax [189] or sigmoid are used to classify the output classes.

7.2.2 Deep belief neural networks

Deep belief network (DBN) was introduced by Geoffrey E. Hinton [190] to overcome the limitations of deep feedforward artificial neural networks. The two most important properties of DBNs are

- There is a layer by layer procedure for learning the top-down, generative weights that determines how the variables in one layer depend on the variables in the layer above.
- After learning vales of latent variables in every layer can be inferred by a single, bottom-up pass that starts with an observed data vector in the bottom layer and uses the generative weights in the reverse direction.

Unlike backpropagation algorithm deep nets leaning networks can be trained by unsupervised learning algorithms for feature detection. However, after unsupervised leaning DBN can further fine-tuned through supervised learning to perform classification or regression. This greedy-layered wise training is mostly performed by using backpropagation or other gradient decent algorithm [191]. In these networks each layer acts as a feature generator and converts the input to more abstract representation. A DBN consists of two different types of networks namely Belief Networks and Restricted Boltzmann Machines (RBM).

Belief Network: Belief network is a directed acyclic graph that composed with layers of stochastic binary units with weighted connections. These stochastic binary units have values 0 or 1. The probability to obtain 1 is decided by the bias and weights of the inputs. Thus, the probability equation of these units is as

$$p(x_j = 1) = \frac{1}{1 + exp(-\sum w_{ji} x_i)} \tag{7.3}$$

Where x_i and x_j are the inputs of neurons i and j respectively. w_{ji} is the synaptic weight between i^{th} and j^{th} neurons.

Restricted Boltzmann Machines: Boltzmann machine is an undirected graph with stochastic binary units. The restricted type of Boltzmann machine composed with one visible layer to represent the data and one hidden layer. The hidden layer represents the features that capture higher order correlation in data. These input and visible layers are connected by a symmetric weight matrix. It says 'restricted', because there are no connections between the neurons in the same layer [192], [193]. The restricted connectivity between hidden units makes learning easier.

7.3 Using PSDV for Deep Neural Networks

Deep learning refers the multiple levels of representation and feature detection. Hence, it always looks for deeper and deeper networks for better accuracy of the solution. It is known that deeper networks are difficulty to train. On the other hand, they need unaffordable training time. Therefore, it can observe that, in recent past researchers have started to discuss the optimization of deep learning architectures [192], [193]. However, several research works are undergoing and there are only limited literatures and many researchers have used trial and error method to determine optimal network architecture.

Although the PSDV method was introduced to FNNs, there are some rooms to extent this method to other deep learning architecture too. Hence, this section will discuss the applying of PSDV method to other deep neural architectures by considering two widely used deep leaning architectures namely convolutional neural networks and deep belief networks as examples.

7.3.1 Applying PSDV to convolutional neural networks

The first part of a convolutional network is composed with some convolutional layers and pooling layers alternatively. This part of the network is used for feature detection. It has observed that having more layers in the network cause to obtain better generalized solution. But sometimes adding layer may face over-fitting. In addition, complexity of the network will increases with the number of layers. In

such a case Peak Search Algorithm described in Chapter 5 can apply to obtain the optimal neural network architecture.

However, this application is completely depends on the given problem. Because, training a convolutional network is very resource hungry. Therefore, application of PSDV involving additional training for the entire architecture (pooling layers and fully connected layers) of the CNN is counterproductive. On the other hand, pooling layers in a CNN are introduced on the basis of some interested features of inputs. Therefore, it is rather irrational to attempt to drop some layers and neurons from pooling layers. Therefore, PSDV can be applicable only for fully connected network part of a CNN.

As already stated, latter part of CNNs there are composed of fully connected layers, and they are trained by using the backpropagation algorithm. Therefore, latter part of fully connected layers in a CNN behaves like feedforward artificial neural networks. We have already shown PSDV works for feedforward networks. As such fully connected network within a CNN should work with PSDV. In such instance, it is possible reduce the size of network by reducing the number of hidden layers and removing less important hidden neurons (if exist) by using the delta values as describe in the Chapter 5.

7.3.2 Applying PSDV to deep belief neural networks

A deep belief network is an unsupervised learning networks trains by greedy leaning algorithm to discover new features. As stated, PSDV cannot be applied for unsupervised learning networks. Therefore, PSDV is not applicable for belief networks as they are.

However, after detecting features, some DBNs execute fine tuning of the results, by a subsequent supervised learning network which uses backpropagation training algorithm. Given that PSDV is applicable for supervised learning, DBNs involving a phase of supervised leaning could be optimized by PSDV.

7.4 Summary

The objective of this chapter was to discuss the concept of deep learning and to explain how PSDV could be applied for deep neural networks. In this sense, first, the concept of deep neural networks was discussed in relation to deep Convolutional Neural Networks (CNN) and Deep Belief Networks (DBN). Secondly, it was explained that PSDV could be applied for the phase involving supervised learning in CNN and DBN. As such fully connected network in CNN, and DBN associating with supervised learning could be optimized by PSDV.

The next chapter will conclude the results and discuss the future works.

CHAPTER 8

CONCLUSION AND FUTURE WORKS

8.1 Introduction

The interest in applying artificial neural networks of many fields including medicine, economics agriculture and engineering has been increased within the last few decades. Hence, the modelling of hidden layer architecture in ANNs has become crucial and important area of research. Throughout this research, a new method on designing hidden layer architecture of ANNs was discussed. The previous chapter discussed the experimental designing results of the newly introduced method. This chapter provides a summary of the thesis. The next gives the achievement of objectives mentioned in the chapter 1 and then concludes the experimental results. Further, it discusses the limitations and future works.

8.2 Modelling Hidden Layer Architecture in ANN

Artificial neural networks are widely used in many real world problems including classification and pattern recognition. Despite many advantages of ANNs, choose the most appropriate architecture which gives the optimal solution for the given task is crucial. This thesis has addressed this problem by critically reviewing the existing many approaches. Although there are variety of approaches inspired by different optimization theories, still they have several drawbacks. The converging to undesired local minima has been identified as the one of main limitation of the current methods. This problem may be avoided by using the global optimization. However, these algorithms may not computationally economical [8].

The proposed novel approach to modelling hidden layers in ANN is inspired by the facts of neuroplasticity. Experimental results show that any large network can be trimmed down to a smaller one by pruning its hidden layers and neurons. However, the number of hidden layers in the architecture is significant and it depends on several parameters such as type of data, size of input vector and number of samples in the training set.

The accuracy of the result obtained by an ANN strongly depends on the number of hidden layers in the network. Generally, multilayered perceptron's give better performance than single hidden layer networks. However, there is an upper limit on the number of hidden layers and the maximum number of hidden layers found in a network in this research was 9 for Monk's 3 problem. The majority of the data sets achieved their best performance by 4 hidden layers.

The proposed method successfully pruned the network without degrading its performance. By the neurons elimination process, 5% - 50 % have been removed from the networks obtained by the PSA. As a whole, the process was able to eliminate about 80 % while improving the generalization. It is obvious that new method gives a better solution than the training an arbitrary sized network with back propagation algorithm. In addition, it shows improved results in generalization, comparing with the many existing methods. Moreover, this method does not have complex arithmetic and hence, it is easy to accommodate. Not only that, but also this approach eliminates cluster of neurons at once instead of removing one at each iteration. Thus, it reaches the optimal solution faster.

8.3 Objectives-wise Achievement

The aim of our research is to design most appropriate neural network architecture to solve the given problem. We achieved this goal while fulfilling the following objectives mentioned in the Chapter 1.

Critical review of artificial neural networks and their uses: The chapter 2 discussed the fundamental concepts of ANNs and critically reviews their uses. Further this was focused on different structure of ANNs and neural network learning by highlighting the activation functions and different learning rules associate with supervised learning, unsupervised learning and reinforcement learning. Moreover, it emphasized that the backpropagation learning rule is the most

widely learning algorithms and with the invention of the backpropagation algorithm, the research interest on ANNs was dramatically increased.

In depth study of current approaches to model hidden layer architecture in ANNs: The Chapter 2 discussed various methods of modelling hidden layer architecture. Generally, these approaches have developed under 3 major techniques, pruning, constructive and evolutionary. It was considered methods from early 1940 to 2015. Also, most famous approaches include OBD, OBS were taken in to account. The methodologies, strengths and weakness were discussed in depth. At the end strengths and limitations of deep learning algorithms were discussed.

Develop an approach to prune hidden layer architecture of ANNs: The Chapter 5 addresses the requirement of this objective. This is the most important chapter as the achievement of the whole research totally depends on this approach. The process was initiated with the hypothesis that any large network could make smaller one without degrading its performance by trimming down the neurons and weight connections. Two algorithms were designed to reach the optimal architecture and the first algorithm which determines the number of hidden layers was encouraged by bi-search algorithms. The second algorithm design to fine tune the network obtained by the first part by removing the irrelevant neurons. The removable neurons on the second phase were determined based on the delta values of hidden neurons. The layered structure, neurons elimination and merging were inspired by the facts of the neuroplasticity and the synaptic pruning.

Evaluate of the novel approach: Evaluate the model which has obtained in the previous section is also one of the main tasks of this research. The evaluation process of this research was achieved by chapter 6. For this model, the evaluation was done with 34 real-world applications in 19 different domains. For all the data, it was able reduce the number of hidden layers and hidden neurons from the ANN architecture. Hence, the modified architecture is computationally economic. Among the all data sets, 25% was able to reduce its size in more than 80%. In addition 15 data sets increased their accuracy while reducing the size. In the new

model it increases the accuracy mainly from 10% to 30%. However, some of the datasets started with their highest accuracy, and hence, it was not able to improve their generalization using this model. So, as the overall results, it shows that, this new method is able to cut down a large sized network to a smaller network while improving the performance. Not only that, it also depicts that, the proposed method gives better performance than many of the existing approaches.

Extension of PSDV Method into Deep Neural Networks: In recent years, deep neural networks have become one of the hot topics in pattern recognition and machine learning. However, training a deep network is hard and researches have started to see the possibilities of optimize the architectures for much efficient solution. Chapter 7 considered this matter and discussed how to extend the PSDV method to deep neural architectures. The convolutional neural networks and deep belief networks are two of widely used deep learning architectures and to identify unnecessary neurons and reduce some layer of these networks PSDV method can be applied. Also these results can be extended to the other existing deep nets too.

8.4 Limitations and Future Directions

The proposed method is successfully reached to the optimal solution with less computation relative to the other existing methods. However, this may not be the best achievement as still some limitations are there and they should be improved to enhance the applications of this method.

Activation Function: The Back propagation algorithm was used in all the training process and the log sigmoid function was used as activation function in the hidden layers whilst linear function was used in the output layer. The log sigmoid function attains its limsup at about 8 repetitions and thus, the output of each network squeeze after a certain number of layers. So that in deeper networks, more knowledge will be lost, which will be caused on a huge error. To avoid this fault, some other activation function such as the rectified linear unit (ReLU) has been used in some deep networks. Nevertheless, ReLU fails at some points as it can be fragile and die

during the training. Hence, experiments need to be done with a carefully chosen activation function.

Theoretical Aspects: Most of the decisions in this project derived based on the empirical results. Hence, the conclusion made here may diverge if we employ a different set of data. Hence, it encourages obtaining mathematical evidence on these aspects. For example, rather than showing network reach to a peak while increasing the hidden layers, it will be more appropriate if we can obtain an analytical solution.

The ultimate goal of modeling hidden layer architecture is to come across a solid method, which enables to apply in solving the extremely complex data sets in the real-world problems.

8.5 Summary

This chapter concludes works done throughout this thesis. It summarized how it achieved the main objectives given in the Chapter 1. Further it discussed the achievements of newly introduced PSDV algorithm and how to extend to deep neural networks. Finally, it highlighted the limitations and future works.

REFERENCES

[1] Y. Liu, J. A. Starzyk, and Z. Zhu, "Optimizing number of hidden neurons in neural networks.," in *Artificial Intelligence and Applications*, 2007, pp. 138–143.

[2] J. Amini, "Optimum Learning Rate in Back-Propagation Neural Network for Classification of Satellite Images (IRS-1D)," *Sci. Iran.*, vol. 15, no. 6, pp. 558–567, Dec. 2008.

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

[4] R. P. Lippmann, "Anintroduction to computing with neural nets,"," *IEEE Assp Mag.*, vol. 4, no. 2, pp. 4–22, 1987.

[5] Y. L Cun, "Generalization and Network Design Strategies," *Connect. Perspect.*, no. Pfeifer, Schreter, Fogelman and Steels (eds) "Connectionism in perspective," Elsevier, pp. 143–155, 1989.

[6] J. Denker, D. Schwartz, B. W ittner, S. Solla, R. Howard, and L. Jackel, "Large Automatic Learning, Rule Extraction, and Generalization," *Complex Syst.*, vol. 1, pp. 877–922, 1987.

[7] "Brain Plasticity: How learning changes your brain," *SharpBrains*, 26-Feb-2008. [Online]. Available: https://sharpbrains.com/blog/2008/02/26/brain-plasticity-how-learning-changes-your-brain/. [Accessed: 08-Jun-2019].

[8] A. D. Anastasiadis, "Neural networks training and applications using biological data," PhD Thesis, Birkbeck, University of London, UK, 2006.

[9] G. D. Magoulas, M. N. Vrahatis, and G. S. Androulakis, "Effective backpropagation training with variable stepsize," *Neural Netw.*, vol. 10, no. 1, pp. 69–82, 1997.

[10] V. Demarin and S. Morović, "Neuroplasticity," *Period. Biol.*, vol. 116, no. 2, pp. 209–211, 2014.

[11] O. Mahmoud, F. Anwar, and M. J. E. Salami, "Learning algorithm effect on multilayer feed forward artificial neural network performance in image coding," *J Eng Sci Technol*, vol. 2, no. 2, pp. 188–199, 2007.

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

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

[14] T. Ash, "Dynamic node creation in backpropagation networks," *Connect. Sci.*, vol. 1, no. 4, pp. 365–375, 1989.

[15] R. Reed, "Pruning algorithms-a survey," *IEEE Trans. Neural Netw.*, vol. 4, no. 5, pp. 740–747, 1993.

[16] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in neural information processing systems*, 1990, pp. 598–605.

[17] R. Setiono, "A penalty-function approach for pruning feedforward neural networks," *Neural Comput.*, vol. 9, no. 1, pp. 185–204, 1997.

[18] M. Hagiwara, “A simple and effective method for removal of hidden units and weights,” *Neurocomputing*, vol. 6, no. 2, pp. 207–218, 1994.
[19] 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.
[20] S. M. Kamruzzaman and M. D. Islam, “An algorithm to extract rules from artificial neural networks for medical diagnosis problems,” *ArXiv Prepr. ArXiv10094566*, 2010.
[21] P. L. Narasimha, W. H. Delashmit, M. T. Manry, J. Li, and F. Maldonado, “An integrated growing-pruning method for feedforward network training,” *Neurocomputing*, vol. 71, no. 13–15, pp. 2831–2847, 2008.
[22] H.-G. Han and J.-F. Qiao, “A structure optimisation algorithm for feedforward neural network construction,” *Neurocomputing*, vol. 99, pp. 347–357, 2013.
[23] S. Lawrence, C. L. Giles, and A. C. Tsoi, “What size neural network gives optimal generalization? Convergence properties of backpropagation,” 1998.
[24] D. R. Wilson and T. R. Martinez, “The need for small learning rates on large problems,” in *Neural Networks, 2001. Proceedings. IJCNN’01. International Joint Conference on*, 2001, vol. 1, pp. 115–119.
[25] S. Herculano-Houzel, “The human brain in numbers: a linearly scaled-up primate brain,” *Front. Hum. Neurosci.*, vol. 3, p. 31, 2009.
[26] J. T. Bruer, “Neural connections: Some you use, some you lose,” *Phi Delta Kappan*, vol. 81, no. 4, pp. 264–277, 1999.
[27] G. Chechik, I. Meilijson, and E. Ruppin, “Synaptic pruning in development: A novel account in neural terms,” in *Computational Neuroscience*, Springer, 1998, pp. 149–154.
[28] T. Dean, “A computational model of the cerebral cortex,” in *Proceedings of the National Conference on Artificial Intelligence*, 2005, vol. 20, p. 938.
[29] L. N. Long and A. Gupta, “Scalable massively parallel artificial neural networks,” *J. Aerosp. Comput. Inf. Commun.*, vol. 5, no. 1, pp. 3–15, 2008.
[30] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” California Univ San Diego La Jolla Inst for Cognitive Science, 1985.
[31] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *Cogn. Model.*, vol. 5, no. 3, p. 1, 1988.
[32] N. M. Wagarachchi and A. S. Karunananda, “Optimization of multi-layer artificial neural networks using delta values of hidden layers,” in *2013 IEEE Symposium on Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB)*, 2013, pp. 80–86.
[33] S. Haykin, *Neural Networks and Machine Learning*, Third Edition. Prentice Hall.
[34] P. J. Fox, “Massively parallel neural computation,” PhD Thesis, University of Cambridge, 2013.
[35] A. Azzini, “A New Genetic Approach for Neural Network Design and Optimization,” Ph. D. Dissertation. Universita Degli Studi Di Milano, Unpublished, 2006.
[36] S. N. Sivanandam and M. PaulraJ, *Introduction to Artificial Neural Networks*. Vikas Publishing House Pvt Ltd.

[37] J. A. Hertz, A. S. Krogh, and R. G. Palmer, *Introduction to the Theory of Neural Computation*. Citeseer.
[38] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: A tutorial," *Computer*, no. 3, pp. 31–44, 1996.
[39] "Similarities Between the Computer and the Brain," *PC Dreams*, 15-Jun-2016. .
[40] H. B. Demuth, M. H. Beale, O. De Jess, and M. T. Hagan, *Neural network design*. Martin Hagan, 2014.
[41] W. S. McCulloch and W. Pitts, "Complete Gradient Clustering Algorithm for Features Analysis of X-ray Images," *Bull. Math. Biophys.*, vol. 5, no. 4, pp. 115–133, 1943.
[42] D. O. Hebb, *the organization of behavior - Google Search*. .
[43] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain.," *Psychol. Rev.*, vol. 65, no. 6, p. 386, 1958.
[44] B. Widrow, M. E. Hoff, and others, "Adaptive switching circuits," in *IRE WESCON convention record*, 1960, vol. 4, pp. 96–104.
[45] Q. J. Zhang and K. C. Gupta, *Neural Networks for RF and Microwave Design*. Boston: Artech House, 2000.
[46] Bernard Widrow and Michael A. Lehr, "Adaptive Neural Networks and Their Applications," *Int. J. Intellegence Syst.*, vol. 8, pp. 453–507, 1993.
[47] J. A. Bullinaria, *Radial Basis Function Networks: Introduction*. 2004.
[48] M. J. Orr, *Introduction to radial basis function networks*. Technical Report, Center for Cognitive Science, University of Edinburgh, 1996.
[49] S. Becker, "Unsupervised learning procedures for neural networks," *Int. J. Neural Syst.*, vol. 2, no. 01n02, pp. 17–33, 1991.
[50] J. L. McCLELLAND, D. E. RUMELHART, and G. E. HINTON, "The Appeal of Parallel Distributed Processing," *MIT Press*, vol. 01, pp. 151–193., 1986.
[51] N. J. Nilson, *Learning Machines: Foundations of Trainable Pattern Classifiers*. New York: McGraw Hills, 1965.
[52] Marvin L Minskey and Seymour A. Papert, *Perceptrons*. USA: The Science Press.
[53] M. A. Nielson, *Neural Networks and Deep Learning*. 2015.
[54] R. Rojas, "Neural Networks: a systematic introduction," 2009.
[55] R. M. Freund, "The steepest descent algorithm for unconstrained optimization and a bisection line-search method," *J. Mass. Inst. Technol. U. S. Am.*, 2004.
[56] J. C. Meza, "Steepest descent," *Wiley Interdiscip. Rev. Comput. Stat.*, vol. 2, no. 6, pp. 719–722, 2010.
[57] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for Boltzmann machines," *Cogn. Sci.*, vol. 9, no. 1, pp. 147–169, 1985.
[58] G. Hinton, "Boltzmann machines," in *Encyclopedia of Machine Learning*, Springer, 2011, pp. 132–136.
[59] P. Paxton, P. J. Curran, K. A. Bollen, J. Kirby, and F. Chen, "Monte Carlo experiments: Design and implementation," *Struct. Equ. Model.*, vol. 8, no. 2, pp. 287–312, 2001.
[60] G. Thimm and E. Fiesler, "Pruning of neural networks," IDIAP, 1997.

[61] J. Ghosh and K. Tumer, “Structural adaptation and generalization in supervised feed-forward networks, d,” *J. Artif. Neural Netw.*, vol. 01, pp. 431–458, 1994.
[62] D. Sabo and X.-H. Yu, “A new pruning algorithm for neural network dimension analysis,” in *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, 2008, pp. 3313–3318.
[63] G. G. Rigatos and S. G. Tzafestas, “Attractors and energy spectrum of neural structures based on the model of the quantum harmonic oscillator,” in *Complex-Valued Neural Networks: Utilizing High-Dimensional Parameters*, IGI Global, 2009, pp. 376–410.
[64] M. G. Augasta and T. Kathirvalavakumar, “A novel pruning algorithm for optimizing feedforward neural network of classification problems,” *Neural Process. Lett.*, vol. 34, no. 3, p. 241, 2011.
[65] M. G. Augasta and T. Kathirvalavakumar, “Pruning algorithms of neural networks—a comparative study,” *Cent. Eur. J. Comput. Sci.*, vol. 3, no. 3, pp. 105–115, 2013.
[66] S. Marsland, J. Shapiro, and U. Nehmzow, “A self-organising network that grows when required,” *Neural Netw.*, vol. 15, no. 8, pp. 1041–1058, 2002.
[67] M. M. Islam and K. Murase, “A new algorithm to design compact two-hidden-layer artificial neural networks,” *Neural Netw.*, vol. 14, no. 9, pp. 1265–1278, 2001.
[68] S. Cohen and N. Intrator, “Forward and backward selection in regression hybrid network,” in *International Workshop on Multiple Classifier Systems*, 2002, pp. 98–107.
[69] P. A. Castillo, M. G. Arenas, J. J. Castillo-Valdivieso, J. J. Merelo, A. Prieto, and G. Romero, “Artificial neural networks design using evolutionary algorithms,” in *Advances in Soft Computing*, Springer, 2003, pp. 43–52.
[70] J. R. McDonnell and D. Waagen, “Determining neural network hidden layer size using evolutionary programming,” DTIC Document, 1993.
[71] Z. Reitermanov´a, “Feedforward Neural Networks – Architecture Optimization and ... - Google Search,” *WDS08 Proc. Contrib. Pap.*, vol. 1, pp. 159–164, 2008.
[72] J. Sietsma and R. J. Dow, “Neural net pruning-why and how,” presented at the IEEE international conference on neural networks, 1988, vol. 1, pp. 325–333.
[73] M. C. Mozer and P. Smolensky, “Skeletonization: A technique for trimming the fat from a network via relevance assessment,” 1989.
[74] K. O. Arras, “An introduction to error propagation: Derivation, meaning and examples of cy= fx cx fx,” 1998.
[75] S. Haykin, “The Least Mean Square Algorithm,” in *Neural Networks and Learning Machines, .*
[76] J. Gorodkin, L. K. Hansen, A. Krogh, C. Svarer, and O. Winther, “A quantitative study of pruning by optimal brain damage,” *Int. J. Neural Syst.*, vol. 4, no. 02, pp. 159–169, 1993.
[77] B. Hassibi, D. G. Stork, and G. J. Wolff, “Optimal brain surgeon and general network pruning,” in *Neural Networks, 1993., IEEE International Conference on*, 1993, pp. 293–299.

[78] T. Kavzoglu and P. M. Mather, “Assessing artificial neural network pruning Algorithms,” *Proc. 24th Annu. Conf. Exhib. Remote Sens. Soc. UK*, pp. 603–609, 1998.
[79] J. Sietsma and R. J. F. Dow, “Creating artificial neural networks that generalize,” *Neural Netw.*, vol. 4, pp. 67–79, 1991.
[80] J. Sietsma and R. J. Dow, “Creating artificial neural networks that generalize,” *Neural Netw.*, vol. 4, no. 1, pp. 67–79, 1991.
[81] P. Lauret, E. Fock, and T. A. Mara, “A node pruning algorithm based on a Fourier amplitude sensitivity test method,” *IEEE Trans. Neural Netw.*, vol. 17, no. 2, pp. 273–293, 2006.
[82] A. Saltelli, S. Tarantola, and K.-S. Chan, “A quantitative model-independent method for global sensitivity analysis of model output,” *Technometrics*, vol. 41, no. 1, pp. 39–56, 1999.
[83] X. Zeng and D. S. Yeung, “Hidden neuron pruning of multilayer perceptrons using a quantified sensitivity measure,” *Neurocomputing*, vol. 69, no. 7, pp. 825–837, 2006.
[84] P. V. S. Ponnapalli, K. C. Ho, and M. Thomson, “A formal selection and pruning algorithm for feedforward artificial neural network optimization,” *IEEE Trans. Neural Netw.*, vol. 10, no. 4, pp. 964–968, 1999.
[85] T. Q. Huynh and R. Setiono, “Effective neural network pruning using cross-validation,” in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, 2005, vol. 2, pp. 972–977.
[86] K. Suzuki, I. Horiba, and N. Sugie, “A simple neural network pruning algorithm with application to filter synthesis,” *Neural Process. Lett.*, vol. 13, no. 1, pp. 43–53, 2001.
[87] F. Fnaiech, N. Fnaiech, and M. Najim, “A new feedforward neural network hidden layer neuron pruning algorithm,” in *Acoustics, Speech, and Signal Processing, 2001. Proceedings.(ICASSP’01). 2001 IEEE International Conference on*, 2001, vol. 2, pp. 1277–1280.
[88] “volterra series - Google Search.” [Online]. Available: https://www.google.lk/?gws_rd=ssl#q=volterra+series. [Accessed: 28-Jul-2016].
[89] L. Carassale and A. Kareem, “Modeling nonlinear systems by Volterra series,” *J. Eng. Mech.*, vol. 136, no. 6, pp. 801–818, 2009.
[90] J. Sai, P. Yang, S. Zhong, J. Wang, and J. Uk Kim, “A Novel Adaptive Architecture Pruning Algorithm for Madalines,” *Int. J. Hybrid Inf. Technol.*, vol. 9, no. 6, pp. 303–316, 2016.
[91] R. Winter and B. Widrow, “MADALINE RULE 11: A Training Algorithm for Neural Networks,” presented at the IEEE International Conference on Neural Networks 1988, 401–408., 2015.
[92] “ADALINE,” *Wikipedia, the free encyclopedia*. 07-May-2016.
[93] S. Zhong, X. Zeng, S. Wu, and L. Han, “Sensitivity-based adaptive learning rules for binary feedforward neural networks,” *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 3, pp. 480–491, 2012.
[94] A. P. Engelbrecht, “A new pruning heuristic based on variance analysis of sensitivity information,” *IEEE Trans. Neural Netw.*, vol. 12, no. 6, pp. 1386–1399, 2001.

[95] H.-J. Xing and B.-G. Hu, “Two-phase construction of multilayer perceptrons using information theory,” *IEEE Trans. Neural Netw.*, vol. 20, no. 4, pp. 715–721, 2009.
[96] J. Xu and D. W. Ho, “A new training and pruning algorithm based on node dependence and Jacobian rank deficiency,” *Neurocomputing*, vol. 70, no. 1, pp. 544–558, 2006.
[97] M. I. Lourakis, “A brief description of the Levenberg-Marquardt algorithm implemented by levmar,” *Found. Res. Technol.*, vol. 4, no. 1, 2005.
[98] Y. Chauvin, “A Back-Propagation Algorithm with Optimal Use of Hidden Units.,” in *NIPS*, 1988, vol. 1, pp. 519–526.
[99] C. Ji, R. R. Snapp, and D. Psaltis, “Generalizing smoothness constraints from discrete samples,” *Neural Comput.*, vol. 2, no. 2, pp. 188–197, 1990.
[100] “Back-propagation, weight elimination and time series prediction,” *ResearchGate*. [Online]. Available: https://www.researchgate.net/publication/245578996_Back-propagation_weight_elimination_and_time_series_prediction. [Accessed: 11-Apr-2017].
[101] G. E. Hinton, “Connectionist learning procedures,” *Artif. Intell.*, vol. 40, no. 1–3, pp. 185–234, 1989.
[102] S. K. Sharma and P. Chandra, “Constructive neural networks: a review,” *Int. J. Eng. Sci. Technol.*, vol. 2, no. 12, pp. 7847–7855, 2010.
[103] R. Parekh, J. Yang, and V. Honavar, “Constructive neural-network learning algorithms for pattern classification,” *IEEE Trans. Neural Netw.*, vol. 11, no. 2, pp. 436–451, 2000.
[104] S. S. Sridhar and M. Ponnavaikko, “Improved Adaptive Learning Algorithm forConstructive Neural Networks,” *Int. J. Comput. Electr. Eng.*, vol. 3, no. 1, p. 30, 2011.
[105] T.-Y. Kwok and D. Y. Yeung, “Constructive feedforward neural networks for regression problems: A survey,” 1995.
[106] M. V. M. Figueredo, “A Learning Algorithm for Constructive Neural Networks Inspired on Decision Trees and Evolutionary Algorithms.”
[107] J. Moody, “Prediction risk and architecture selection for neural networks,” in *From Statistics to Neural Networks*, Springer, 1994, pp. 147–165.
[108] M. R. Azimi-Sadjadi, S. Sheedvash, and F. O. Trujillo, “Recursive dynamic node creation in multilayer neural networks,” *IEEE Trans. Neural Netw.*, vol. 4, no. 2, pp. 242–256, 1993.
[109] B.-T. Zhang, “An incremental learning algorithm that optimizes network size and sample size in one trial,” in *Neural Networks, 1994. IEEE World Congress on Computational Intelligence., 1994 IEEE International Conference on*, 1994, vol. 1, pp. 215–220.
[110] M. Mézard and J.-P. Nadal, “Learning in feedforward layered networks: The tiling algorithm,” *J. Phys. Math. Gen.*, vol. 22, no. 12, p. 2191, 1989.
[111] S. I. Gallant, “Perceptron-based learning algorithms,” *IEEE Trans. Neural Netw.*, vol. 1, no. 2, pp. 179–191, 1990.
[112] J. H. Friedman and W. Stuetzle, “Projection pursuit regression,” *J. Am. Stat. Assoc.*, vol. 76, no. 376, pp. 817–823, 1981.

[113] J. H. Friedman, E. Grosse, and W. Stuetzle, “Multidimensional additive spline approximation,” *SIAM J. Sci. Stat. Comput.*, vol. 4, no. 2, pp. 291–301, 1983.
[114] J.-N. Hwangy, S.-R. Layy, M. Maechlerz, D. Martin, and J. Schimert, “Regression Modeling in Back-Propagation and Projection Pursuit Learning3,” *IEEE Trans Neural Netw.*, no. 3, pp. 324–353, 1994.
[115] C. B. Roosen and T. J. Hastie, “Automatic smoothing spline projection pursuit,” *J. Comput. Graph. Stat.*, vol. 3, no. 3, pp. 235–248, 1994.
[116] Y. Shin and J. Ghosh, “Ridge polynomial networks,” *IEEE Trans. Neural Netw.*, vol. 6, no. 3, pp. 610–622, 1995.
[117] J.-L. Yuan and T. L. Fine, “Forecasting demand for electric power,” *Adv. Neural Inf. Process. Syst.*, pp. 739–739, 1993.
[118] I. Rivals and L. Personnaz, “A statistical procedure for determining the optimal number of hidden neurons of a neural model,” in *Proceedings of the Second ICSC Symposium on Neural Computation NC*, 2000.
[119] K. Alam, B. Chandra Karmokar, and M. Siddiquee, “A COMPARISON OF CONSTRUCTIVE AND PRUNING ALGORITHMS TO DESIGN NEURAL NETWORKS,” *Indian J. Comput. Sci. Eng.*, vol. 2, Jun. 2011.
[120] S. Asthana, R. K. Bhujade, N. Sharma, and R. Singh, “Handwritten Multiscript Pin Code Recognition System having Multiple hidden layers using Back Propagation Neural Network,” *Int. J. Electron. Commun. Comput. Eng. ISSN Online*, 2011.
[121] S. Karsoliya, “Approximating number of hidden layer neurons in multiple hidden layer BPNN architecture,” *Int. J. Eng. Trends Technol.*, vol. 3, no. 6, pp. 713–717, 2012.
[122] L. Vollmer, “Change Your Mind: Neuroplasticity & Buddhist Transformation,” 2010.
[123] A. Pascual-Leone, A. Amedi, F. Fregni, and L. B. Merabet, “The plastic human brain cortex,” *Annu Rev Neurosci*, vol. 28, pp. 377–401, 2005.
[124] J. J. Eggermont, *The Correlative Brain: Theory and Experiment in Neural Interaction*. New York: Springer-Verlag, 1990.
[125] P. S. Churchland and T. J. Sejnowski, *The Computational Brain*. Cambridge,MA: MIT Press, 1992.
[126] Y. Perwej and F. Parwej, “A Neuroplasticity (Brain Plasticity) Approach to Use in Artificial Neural Network.”
[127] J. Chrol-Cannon and Y. Jin, “Computational modeling of neural plasticity for self-organization of neural networks,” *BioSystems*, vol. 125, pp. 43–54, 2014.
[128] S. Trojan and J. Pokorny, “Threoretical aspects of neuroplasticity,” *Physiol. Res.*, vol. 48, pp. 87–98, 1999.
[129] D. Krech, M. R. Rosenzweig, and E. L. Bennett, “Chemical and anatomical plasticity of brain,” *Science*, vol. 146, pp. 610–619, 1964.
[130] F. R. Ferreira, M. I. Nogueira, and J. DeFelipe, “The influence of James and Darwin on Cajal and his research into the neuron theory and evolution of the nervous system,” *Front. Neuroanat.*, vol. 8, 2014.
[131] J. William, *The principles of psychology*. New York: Holt, 1890.
[132] J. Konorski, “Conditioned reflexes and neuron organization.,” 1948.

[133] V. Demarin and S. Morović, “Neuroplasticity,” *Period. Biol.*, vol. 116, no. 2, pp. 209–211, 2014.
[134] P. Bach-y-Rita and S. W. Kercel, “Sensory substitution and the human–machine interface,” *Trends Cogn. Sci.*, vol. 7, no. 12, pp. 541–546, 2003.
[135] “The Amazing Mind – Brain and Its Neuroplasticity | My Becoming Aware.” [Online]. Available: https://mybecomingaware.wordpress.com/2016/02/08/the-amazing-mind-brain-and-its-neuroplasticity/. [Accessed: 04-Mar-2018].
[136] O. Bukalo and A. Dityatev, “Synaptic cell adhesion molecules,” in *Synaptic Plasticity*, Springer, 2012, pp. 97–128.
[137] J. E. Heuser and T. S. Reese, “Structure of the synapse,” *Compr. Physiol.*, 1977.
[138] G. M. Shepherd, “The Synaptic Organization of the,” *BrainOxford Univ. Press NY 1979*, 1990.
[139] G. M. Shepherd, *The Synoptic Organization of the Brain*, 3rd ed. New York: Oxford University Press, 1990.
[140] G. Silberberg, S. Grillner, F. E. LeBeau, R. Maex, and H. Markram, “Synaptic pathways in neural microcircuits,” *Trends Neurosci.*, vol. 28, no. 10, pp. 541–551, 2005.
[141] J.-P. Thivierge and G. F. Marcus, “The topographic brain: from neural connectivity to cognition,” *Trends Neurosci.*, vol. 30, no. 6, pp. 251–259, 2007.
[142] R. B. Wells, “Cortical neurons and circuits: a tutorial introduction,” *Unpubl. Pap. Www Mrc Uidaho Edu*, 2005.
[143] V. B. Mountcastle, “The columnar organization of the neocortex.,” *Brain*, vol. 120, no. 4, pp. 701–722, 1997.
[144] A. M. Thomson, “Neocortical layer 6, a review,” *Front. Neuroanat.*, vol. 4, 2010.
[145] J. Stiles and T. L. Jernigan, “The basics of brain development,” *Neuropsychol. Rev.*, vol. 20, no. 4, pp. 327–348, 2010.
[146] B. J. Casey, J. N. Giedd, and K. M. Thomas, “Structural and functional brain development and its relation to cognitive development,” *Biol. Psychol.*, vol. 54, no. 1, pp. 241–257, 2000.
[147] J. Stiles, T. T. Brown, F. Haist, and T. L. Jernigan, “Brain and Cognitive Development,” *Handb. Child Psychol. Dev. Sci.*, 2015.
[148] M. A. Curtis, M. Kam, and R. L. Faull, “Neurogenesis in humans,” *Eur. J. Neurosci.*, vol. 33, no. 6, pp. 1170–1174, 2011.
[149] M. Götz and W. B. Huttner, “The cell biology of neurogenesis,” *Nat. Rev. Mol. Cell Biol.*, vol. 6, no. 10, pp. 777–788, 2005.
[150] J. Altman, “Are new neurons formed in the brains of adult mammals,” *Science*, vol. 135, no. 3509, pp. 1127–1128, 1962.
[151] D. Purves *et al.*, “Neuronal Migration,” 2001.
[152] B. Nadarajah, P. Alifragis, R. O. L. Wong, and J. G. Parnavelas, “Neuronal Migration in the Developing Cerebral Cortex: Observations Based on Real-time Imaging,” *Cereb. Cortex*, vol. 13, no. 6, pp. 607–611, Jun. 2003.
[153] C. L. de Rouvroit and A. M. Goffinet, “Neuronal migration,” *Mech. Dev.*, vol. 105, no. 1, pp. 47–56, 2001.

[154] R. C. Malenka, “Synaptic plasticity in the hippocampus: LTP and LTD,” *Cell*, vol. 78, no. 4, pp. 535–538, 1994.
[155] “kolb_07.pdf.” .
[156] J. Shaffer, “Neuroplasticity and positive psychology in clinical practice: A review for combined benefits,” *Psychology*, vol. 3, no. 12, p. 1110, 2012.
[157] R. Marciniak, K. Sheardova, P. Čermáková, D. Hudeček, R. Šumec, and J. Hort, “Effect of meditation on cognitive functions in context of aging and neurodegenerative diseases,” *Front. Behav. Neurosci.*, vol. 8, p. 17, 2014.
[158] P. R. Huttenlocher and A. S. Dabholkar, “Regional differences in synaptogenesis in human cerebral cortex,” *J. Comp. Neurol.*, vol. 387, no. 2, pp. 167–178, 1997.
[159] F. I. Craik and E. Bialystok, “Cognition through the lifespan: mechanisms of change,” *Trends Cogn. Sci.*, vol. 10, no. 3, pp. 131–138, 2006.
[160] “Children with Autism Have Extra Synapses in Brain,” *Columbia University Medical Center*, 21-Aug-2014. [Online]. Available: http://newsroom.cumc.columbia.edu/blog/2014/08/21/children-autism-extra-synapses-brain/. [Accessed: 10-Aug-2017].
[161] N. C. Andreasen, P. Nopoulos, V. Magnotta, R. Pierson, S. Ziebell, and B.-C. Ho, “Progressive brain change in schizophrenia: a prospective longitudinal study of first-episode schizophrenia,” *Biol. Psychiatry*, vol. 70, no. 7, pp. 672–679, 2011.
[162] P. R. Huttenlocher and others, “Synaptic density in human frontal cortex-developmental changes and effects of aging,” *Brain Res*, vol. 163, no. 2, pp. 195–205, 1979.
[163] Z. Petanjek *et al.*, “Extraordinary neoteny of synaptic spines in the human prefrontal cortex,” *Proc. Natl. Acad. Sci.*, vol. 108, no. 32, pp. 13281–13286, 2011.
[164] J. Y. Hua and S. J. Smith, “Neural activity and the dynamics of central nervous system development,” *Nat. Neurosci.*, vol. 7, no. 4, p. 327, 2004.
[165] P. Boksa, “Abnormal synaptic pruning in schizophrenia: Urban myth or reality?,” *J. Psychiatry Neurosci. JPN*, vol. 37, no. 2, p. 75, 2012.
[166] D. Das, A. Kole, S. Mukhopadhyay, and P. Chakrabarti, “Empirical Analysis of Binary Search Worst Case on Two Personal Computers Using Curve Estimation Technique,” *Int. J. Eng. Manag. Res.*, vol. 5, no. 5, pp. 304–311, 2015.
[167] A. L. Jacobson, “Auto-threshold peak detection in physiological signals,” in *Engineering in Medicine and Biology Society, 2001. Proceedings of the 23rd Annual International Conference of the IEEE*, 2001, vol. 3, pp. 2194–2195.
[168] N. M. Wagarachchi and A. S. Karunananda, “A Novel Technique for Optimizing the Hidden Layer Architecture in Artificial Neural Networks,” *Am. Int. J. Res. Sci. Technol. Eng. Math.*, vol. 4, no. 1, pp. 01–06, 2013.
[169] “UCI Machine Learning Repository.” [Online]. Available: https://archive.ics.uci.edu/ml/index.php. [Accessed: 11-Apr-2018].
[170] O. L. Mangasarian, R. Setiono, and W. H. Wolberg, “Pattern recognition via linear programming: Theory and application to medical diagnosis,” *Large-Scale Numer. Optim.*, pp. 22–31, 1990.

[171] K. P. Bennett and O. L. Mangasarian, “Robust linear programming discrimination of two linearly inseparable sets,” *Optim. Methods Softw.*, vol. 1, no. 1, pp. 23–34, 1992.
[172] W. H. Wolberg and O. L. Mangasarian, “Multisurface method of pattern separation for medical diagnosis applied to breast cytology.,” *Proc. Natl. Acad. Sci.*, vol. 87, no. 23, pp. 9193–9196, 1990.
[173] L. Prechelt, *PROBEN 1: a set of benchmarks and benchmarking rules for neural network training algorithms*. Univ., Fak. für Informatik, 1994.
[174] “UCI Machine Learning Repository: Credit Approval Data Set.” [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Credit+Approval. [Accessed: 10-Feb-2018].
[175] “UCI Machine Learning Repository: Pima Indians Diabetes Data Set.” [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Pima+Indians+Diabetes. [Accessed: 10-Feb-2018].
[176] “UCI Machine Learning Repository: ser Knowledge Modeling Data (Students’ Knowledge Levels on DC Electrical Machines) Data Set.” [Online]. Available: http://archive.ics.uci.edu/ml/datasets/ser+Knowledge+Modeling+Data+%28Students%27+Knowledge+Levels+on+DC+Electrical+Machines%29. [Accessed: 10-Feb-2018].
[177] D. Mo, “A survey on deep learning: one small step toward AI,” *Dept Comput. Sci. Univ N. M. USA*, 2012.
[178] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
[179] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, “A survey of deep neural network architectures and their applications,” *Neurocomputing*, vol. 234, pp. 11–26, 2017.
[180] Y. Bengio, “Learning deep architectures for AI,” *Found. Trends® Mach. Learn.*, vol. 2, no. 1, pp. 1–127, 2009.
[181] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
[182] J. Schmidhuber, “Deep learning in neural networks: An overview,” *Neural Netw.*, vol. 61, pp. 85–117, 2015.
[183] S. Singaravel, J. Suykens, and P. Geyer, “Deep-learning neural-network architectures and methods: Using component-based models in building-design energy prediction,” *Adv. Eng. Inform.*, vol. 38, pp. 81–90, 2018.
[184] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, “Convolutional neural networks: an overview and application in radiology,” *Insights Imaging*, vol. 9, no. 4, pp. 611–629, 2018.
[185] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
[186] K. O’Shea and R. Nash, “An introduction to convolutional neural networks,” *ArXiv Prepr. ArXiv151108458*, 2015.
[187] J. Wu, “Introduction to Convolutional Neural Networks,” p. 31, May 2017.

[188] A. Khan, A. Sohail, U. Zahoora, and A. S. Qureshi, “A Survey of the Recent Architectures of Deep Convolutional Neural Networks,” p. 62.
[189] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, “Activation Functions: Comparison of trends in Practice and Research for Deep Learning,” *ArXiv Prepr. ArXiv181103378*, 2018.
[190] G. E. Hinton, “Deep belief networks.” Scholarpedia, 4(5):5947, 2009.
[191] A. Khan, A. Zameer, T. Jamal, and A. Raza, “Deep Belief Networks Based Feature Generation and Regression for Predicting Wind Power,” *ArXiv Prepr. ArXiv180711682*, 2018.
[192] T. K. Gupta and K. Raza, “Optimizing Deep Neural Network Architecture: A Tabu Search Based Approach,” p. 15, 2018.
[193] A. Aly, D. Weikersdorfer, and C. Delaunay, “Optimizing Deep Neural Networks with Multiple Search Neuroevolution,” *ArXiv190105988 Cs*, Jan. 2019.
[194] “uci machine learning repository - Google Search.” [Online]. Available: https://www.google.lk/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=uci%20machine%20learning%20repository. [Accessed: 21-Jul-2016].
[195] “Banknote aurhentication Data set.” [Online]. Available: https://archive.ics.uci.edu/ml/datasets/banknote+authentication#. [Accessed: 23-Jan-2018].
[196] “UCI Machine Learning Repository: Cardiotocography Data Set.” [Online]. Available: https://archive.ics.uci.edu/ml/datasets/cardiotocography. [Accessed: 25-Feb-2018].
[197] “UCI Machine Learning Repository: Climate Model Simulation Crashes Data Set.” [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Climate+Model+Simulation+Crashes. [Accessed: 10-Feb-2018].
[198] “UCI Machine Learning Repository: Glass Identification Data Set.” [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Glass+Identification. [Accessed: 07-Feb-2018].
[199] “https://archive.ics.uci.edu/ml/machine-learning-databases/haberman/haberman.names.” [Online]. Available: https://archive.ics.uci.edu/ml/machine-learning-databases/haberman/haberman.names. [Accessed: 23-Jan-2018].
[200] R. A. Fisher, “The use of multiple measurements in taxonomic problems.” Annual Eugenics 7, 1936.
[201] R. O. Duda, P. E. Hart, and D. G. Stork, “Pattern classification and scene analysis 2nd ed,” *Ed Wiley Intersci.*, 1995.
[202] S. B. Thrun *et al.*, “The MONK’s Problems A Performance Comparison of Different Learning Algorithms,” 1991.
[203] I. King, *Neural information processing*. Springer Science & Business Media, 2006.
[204] M. Charytanowicz, J. Niewczas, P. Kulczycki, P.A. Kowalski, S. Lukasik, and S. Zak, “A Complete Gradient Clustering Algorithm for Features Analysis

of X-ray Images," *Inf. Technol. Biomed. Ewa Pietka Jacek Kawa Eds Springer-Verl. Berl.-Heidelb.*, pp. 15–24, 2010.
[205] "UCI Machine Learning Repository: Heart Disease Data Set." [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Heart+Disease. [Accessed: 10-Feb-2018].
[206] "UCI Machine Learning Repository: Statlog (Heart) Data Set." [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Statlog+%28Heart%29. [Accessed: 10-Feb-2018].
[207] J. E. Da Silva, J. M. De Sá, and J. Jossinet, "Classification of breast tissue by electrical impedance spectroscopy," *Med. Biol. Eng. Comput.*, vol. 38, no. 1, pp. 26–30, 2000.
[208] J. Jossinet, "Variability of impedivity in normal and pathological breast tissue," *Med. Biol. Eng. Comput.*, vol. 34, no. 5, pp. 346–350, 1996.
[209] "UCI Machine Learning Repository: Yeast Data Set." [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Yeast. [Accessed: 10-Feb-2018].

Appendix A

DATA SETS

To assess the effectiveness of the proposed method in determining the hidden layer architecture, the number of simulations carried out in different fields. In order to justify the theory, 33 data sets were selected from different 19 domains. All the samples chosen are real world classification problems, available in the Machine learning repository, University of California, Irvine [194], [173]. All the experiments were done for fully connected feedforward neural networks on supervised learning.

The different data sets from different domains were chosen to discuss the various steps of the approach of the modeling the hidden layer architecture in ANNs. Some data sets were tested by changing the size of input/output training patterns. The details of all chosen data sets are discussed here.

1. Banknote Authentication Data Set (Banknote)

The data set describes the information of 1372 banknotes, where 610 are genuine and 762 are counterfeit. By using digitalization, industrial camera 400x400 pixel images were taken and wavelet transformation tools were used to extract 4 features, Variance, Skewness, Kurtosis and Entropy. True banknotes have identified and labeled as 1 and others labeled as 0 [195].

2. Breast Cancer Wisconsin (Cancer)

The data set was introduced by Dr. William H. Wolberg of the University of Wisconsin Hospital, Madison to diagnose the breast cancer and classify that a tumor as either benign or malignant level [170],[171],[172]. The decision makes based on the information gathered by microscopic examination of 9 features.

The data set contains 699 continuous examples, where 65.5% are in benign stage [173]. To examine the performance of the proposed method different 4 types of data sets, namely Cancer I, Cancer II, Cancer III and Cancer IV considered with distinct

testing and training sets. The first group contains 75% of data in the training set while other 25% uses for testing purpose. The 2nd set considered with 50% data in training and the 3rd set contains 25% as training data. Finally, a very small group of data (20 sets) trained and tested the performance.

3. Credit Card Approval Data Set (Card)

A database to predict the approval or rejection of credit card to the applicants is presented here. Each example represents the details supplied by a real applicant and output shows whether the corresponding organization granted a credit card to the client or not. The decision makes based on 51 inputs with continuous values and 690 examples. Out of 690 applicants 44.5% show positive output [174]. Four different networks architectures Card I, Card II, Card III and Card IV were designed from this data.

4. Cardiotocography Data Set (Cardio)

This is a large data set, which consists of measurements of 2126 samples of fetal heart rate (FHR) and uterine contraction (UC) features on cardiotocograms. Each sample contains 21 attributes to classify the fetal class defined as Normal (0), Suspect (+1) or Pathologic (-1). From the whole set 77.8% and 13.9% were classified as normal and suspect respectively, while the rest are recognized as pathologic [196].

5. Climate Model Simulation Crashes Data Set (Climate)

This data set uses to predict the simulation outcome, such as success or failure on given Latin hypercube samples of 18 climate model input parameter values. There are 540 samples in the data set and 494 are success and 46 are fail [197].

6. Pima Indians Diabetes Data Set (Diabetes)

This data was originally created by the National Institute of Diabetes and Digestive and Kidney Diseases to binary classification on whether a patient has diabetes. There are records of 768 patients and out of 500 (≈ 65.1%) shown positive for diabetes. All the patients here are females of above 21 years old of Pima Indian

heritage. Four different networks, namely Diabetes I, II, III and IV carry 75%, 50%, 25%, and 2% samples respectively created for testing [175].

7. Solar Flare Data Set (Flare)

This database has been created to predict the solar flare which will occur in next 24 hours by using the information on past 24 hour period. In the set there are 1389 attributes and results are classified in three different classes, common flare, moderate flare and severe flare [173]. There are 10 attributes in the input set. First 3 inputs are given as the alphabetical characters whilst rest are integers. Before the training process, alphabetic characters converted to integers.

8. Glass Identification Data Set (Glass)

This is a classification dataset to identify the types of glass was motivated by criminological investigation. It suggests that, if it correctly identified, at the scene of the crime, the glass left can be used as evidence. The data set contains 10 attributes including Id number. In the classification glasses are distributed 7 different classes [198].

9. Heberman's Survival Data Set (Heberman)

The survival of patients who had undergone surgery for breast cancer is interpreted by this data set. The survey was conducted in 306 patients of the University of Chicago's billing hospital in between 1958 and 1970. Two survival classes were defined. 1) The patient survived 5 years or longer. 2) The patient died within 5 years. It was reported that 225 patients belonged to the first class 81 in the second class [199].

10. Iris Plant Data Set (Iris)

The Iris plant data sets [200], [201] classify 150 iris flowers on the basis of four of their independent features namely sepal length, sepal width, petal length and petal width. The output was desired to one the 3 classes Setosa, Versicolour or Virginica. Each class contains 50 instances.

11. User Knowledge Modeling Data Set (Knowledge)

The dataset is about the users' learning activities and knowledge levels on subjects of DC Electrical Machines. Information of 403 users with 5 attributes including the study time and exam performance considered for analysis. According to the information, uses knowledge was classified into four classes, very low, low, middle and high [176].

12. MONK's Problems Data Set (Monks)

The Monk's problem which was generated by Sebastian Thrun [202], contains a discrete data set which was created to classify the appearance of a robot. The appearance of each robot was described by 6 attributes, namely robot's head shape, body shape, is smiling (yes/no), holding item (sward/balloon/flag), Jacket colour and is a wearing a tie (yes/no). By analyzing these data it decides whether or not the given robot belongs to a one of the two classes.

Three different data sets from the same domain, but with different features were created. For example; in the Monk's 1 data set, head and body shapes are equal or jacket colour is red. In the third data set jacket colour is green and in addition, it was added 5% classification noise [203].

13. Seeds Data Set (Seeds)

Seven geometric parameters of the kernel of three different varieties of wheat; Kama, Rosa and Canadian were given in this data set. There are 70 elements in each category. The high-quality visualization of the internal kernel structure was detected using a soft X-ray technique [204].

14. Statlog (Heart) Data Set (Statlog)

This dataset is a heart disease database similar to a database present in the UCI repository (Heart Disease databases) [205] but in a slightly different form. The main

objective of this data is to predict whether or the heart disease is present. 13 inputs including age, sex are there in this data set. Among 270 patients 150 show absence in heart disease while other 120 show present [206].

15. Thyroid disease data set

There are several data sets to determine whether a patient referred to the clinic is hypothyroid. Patients are classified in to three classes: normal (not hypothyroid), hyper function and subnormal functioning. In this test we chose the data set includes 215 instances with 5 attributes.

16. Breast Tissue (Tissue)

The outcome of the application of electrical impedance spectroscopy in classification of breast tissue to detect cancer was described in the data set. 10 features were measured in 106 instances and classified them to 6 classes [207], [208].

17. Yeast Data Set (Yeast)

The main objective of the data is to use 8 attributes to predict the localizations (called cellular components) of proteins in a yeast's cell where each protein must be classified into one of nine different cellular components. The output is given as non-numeric variable and converted them to a numeric. Altogether there are 1484 instances [209].

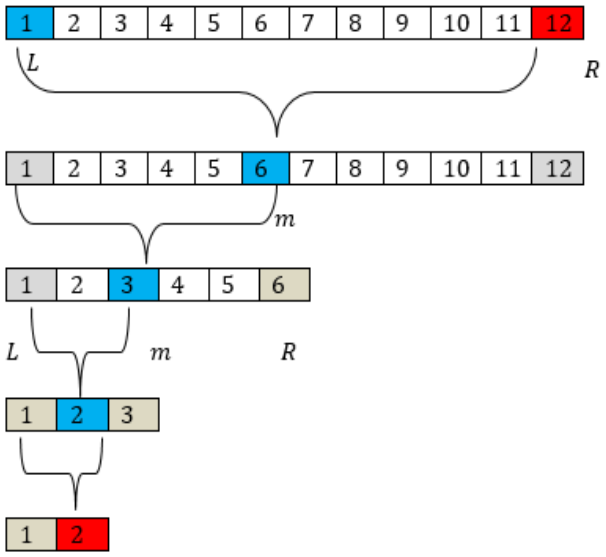
Appendix B

DETERMINING THE NUMBER OF HIDDEN LAYERS

The illustration of determining peak value of each network is described here. The table followed by the each figure verified the results.

B.1 Banknote Data Set

The input network of Banknote data set contains 12 hidden layers and 1014 hidden neurons. Neurons are divided in ascending order as shown in the Table B. 1. The initial network showed 100% accuracy. The single hidden layer network with 13 hidden neurons gives 0.0% performance. Hence, the peak search algorithm applied and process ended with 2 hidden layer network as shown in the Figure B. 1. The experimental results are shown in the Table B. 1. which agrees with the results obtained by the PSA.



$g_{12} = 100,\ g_1 = 0.0 \neq 100$
Start with $L = 1$ and $R = 12$
Compute $m = \left\lfloor \frac{1+12}{2} \right\rfloor = 6$
$g_6 = 100$

Step 1: $R - L > 1$
$g_1 < g_6 = g_{12}$
Replace $R = 6$
Compute $m = \left\lfloor \frac{1+6}{2} \right\rfloor = 3$
$g_3 = 100$

Step 2: $g_1 < g_3 = g_6$
Replace $R = 3$
Compute $m = \left\lfloor \frac{1+3}{2} \right\rfloor = 2$
$g_2 = 100$

Step 3: $g_1 < g_2 = g_3$
Replace $R = 2$
$R - L = 1$
Peak value = $\{h; g_h = \max\{g_1, g_2\}\}$
$= 2$

Figure B. 1: Illustration of Banknote problem

Table B. 1: Generalization of Banknote Problem

Data Set	No. of Hidden Layers										
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**12**
Generalization	0.0	**100**	100	100	100	100	100	100	100	100	100
No. of hidden neurons	13	26	39	52	65	78	91	104	117	130	152

B.2 Cancer Data Sets

Four different datasets were designed from Cancer data set which was described in the section 6.3.1. The Figure 6.8 illustrates the procedure of determining the number of layers in the Cancer I problem. Cancer II and III problems also have the same pattern with lesser generalization. All these sets start with 20 hidden layers and achieve their best performance at 4 hidden layers (Table B.2). The Cancer IV, which has only 20 sets in the training set always shows poor generalization comparatively other 3 sets. Figure B. 2 shows how it achieves the number of hidden layers in the Cancer IV problem. The Table B.3 verifies the results

Table B. 2: Generalization of Cancer I-III problems

Data Set	*P*	No. of Hidden Layers											
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**12**	**20**
Cancer I	525	97.7	97.7	98.9	**99.4**	99.4	96.0	94.3	95.4	62.6	62.6	62.6	62.6
Cancer II	350	97.1	97.4	97.7	**98.6**	97.7	97.7	65.6	65.6	65.6	65.6	65.6	65.6
Cancer III	175	94.7	94.7	94.7	**95.0**	94.7	65.8	65.8	65.8	65.8	65.8	65.8	65.8

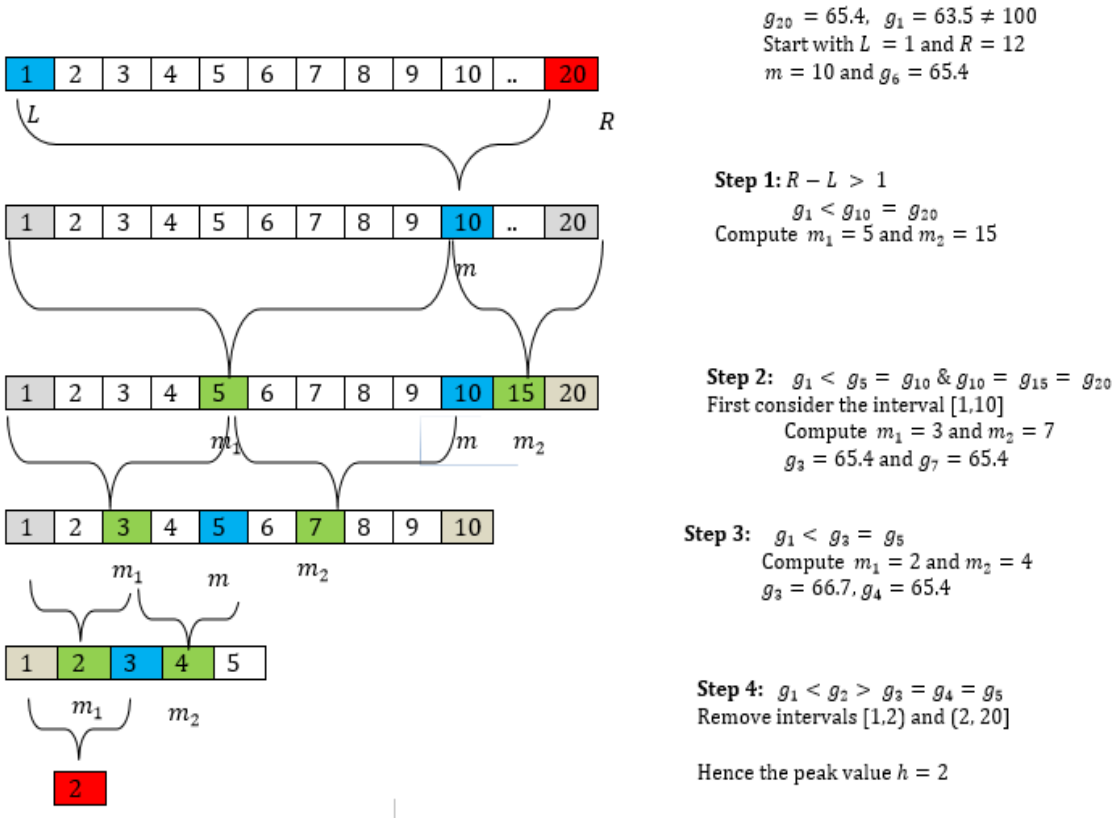


Figure B. 2: Illustration of Cancer IV problem

Table B. 3: Generalization of Cancer IV problem

Data Set	***P***	**No. of Hidden Layers**											
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**12**	**20**
Cancer IV	20	63.5	**66.7**	65.4	65.4	65.4	65.4	65.4	65.4	65.4	65.4	65.4	65.4

B.2 Card Data Sets

Each of the card problem has 51 attributes. Relative to the Cancer problems, they show poor generalization. The maximum generalization (88.1%) obtained for Card I problem with 6 hidden layers. However, like Cancer IV, Card IV which is having only one hidden neuron in each hidden layer show poor performance as the they do not have sufficient neurons to learn data. The following Figure B. 3 illustrate the method of obtaining the most suitable number of hidden layers in Card I problem. The performance of Card II is lesser than Card I, but it also has the same pattern (Table B. **4**)

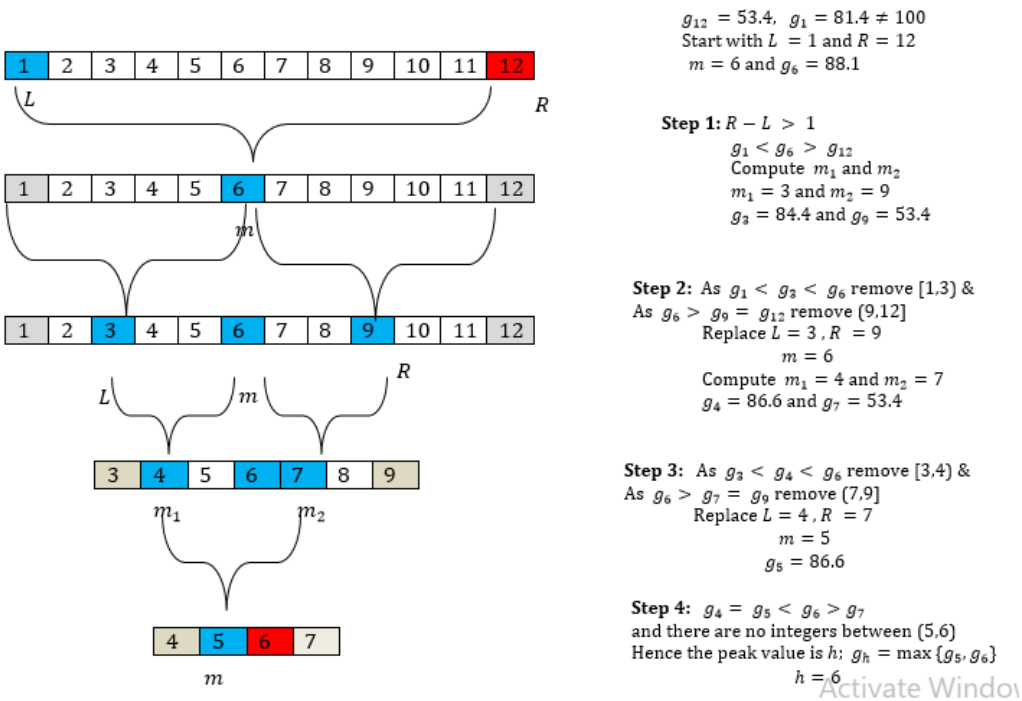


Figure B. 3: Illustration of Card I problem

Table B. 4: Generalization of Card I – II problem

Data Set	*P*	No. of Hidden Layers											
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Card I	518	81.4	81.4	84.9	86.6	86.6	**88.1**	53.4	53.4	53.4	53.4	53.4	53.4
Card II	345	75.1	76.5	84.6	84.6	85.8	**87.2**	55.1	55.1	55.1	55.1	55.1	55.1

The illustration of achieving number of hidden layers of the most appropriate architecture of Card III problem is described in the below Figure B. 4. This process also starts with 12 hidden layer network trained by the backpropagation algorithm. Initially, network shows 54.8% generalization. The network achieves its best performance with 5 hidden layers and generalization 84.8%. The Table B. 5 confirms this result.

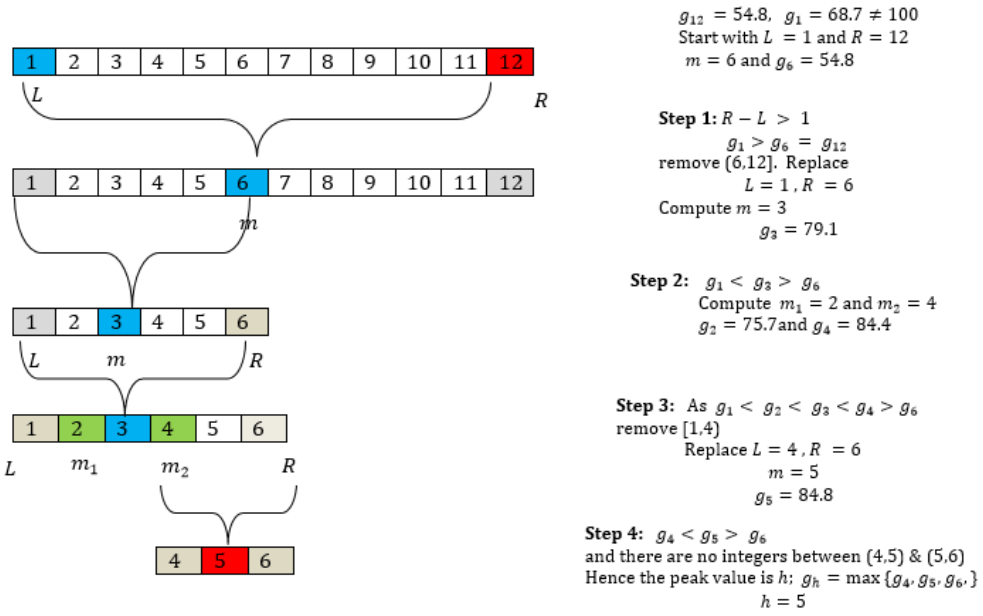


Figure B. 4: Illustration of Card III problem

Table B. 5: Generalization of Card III problem

Data Set	***P***	**No. of Hidden Layers**											
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Card III	172	68.7	75.7	79.1	84.4	**84.8**	54.8	55.1	54.8	54.8	54.8	54.8	54.8

Similar to the Cancer IV, Card IV also show poor performance. The highest generalization (65.7%) gives the 3 layer network (Figure B. 5, Table B. 6).

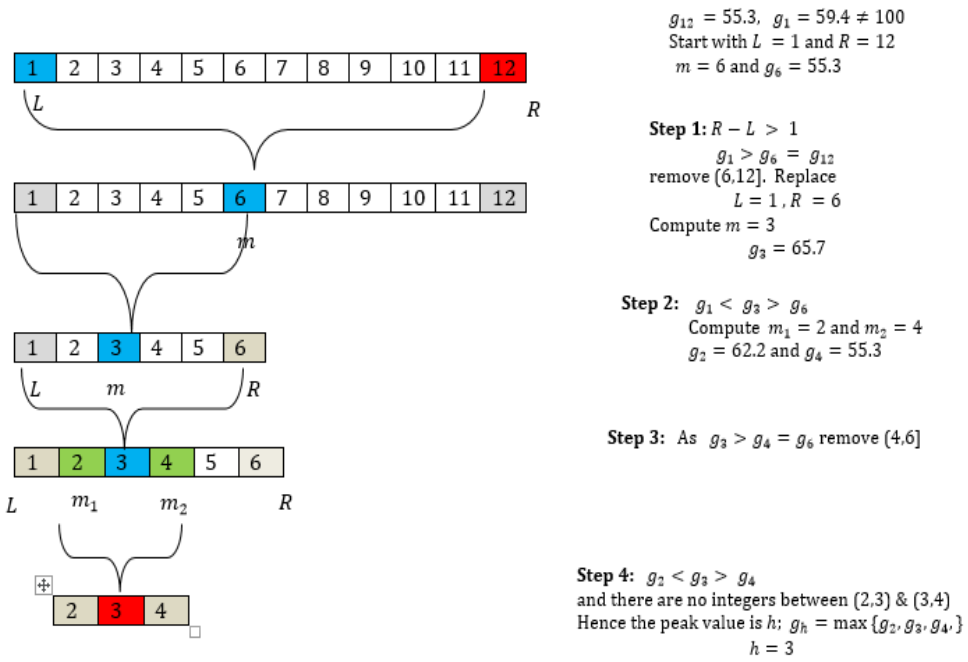


Figure B. 5: Illustration of Card IV problem

Table B. 6: Generalization of Card IV problem

Data Set	***P***	**No. of Hidden Layers**										
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Card IV	20	59.4	62.2	**65.7**	55.3	55.3	55.3	55.3	55.3	55.3	55.3	55.3	55.3

B.4 Climate Data Set

The initial network architecture of this set contained 10 hidden layers and 400 hidden neurons which are equally distributed among the hidden layers. The generalization of the input network was 92.6%. The single hidden layer network showed 95.6% accuracy. At the it was reduced to 2 hidden layer with generalization 97.8%. The process of obtaining optimal architecture is shown in the Figure B. 6.

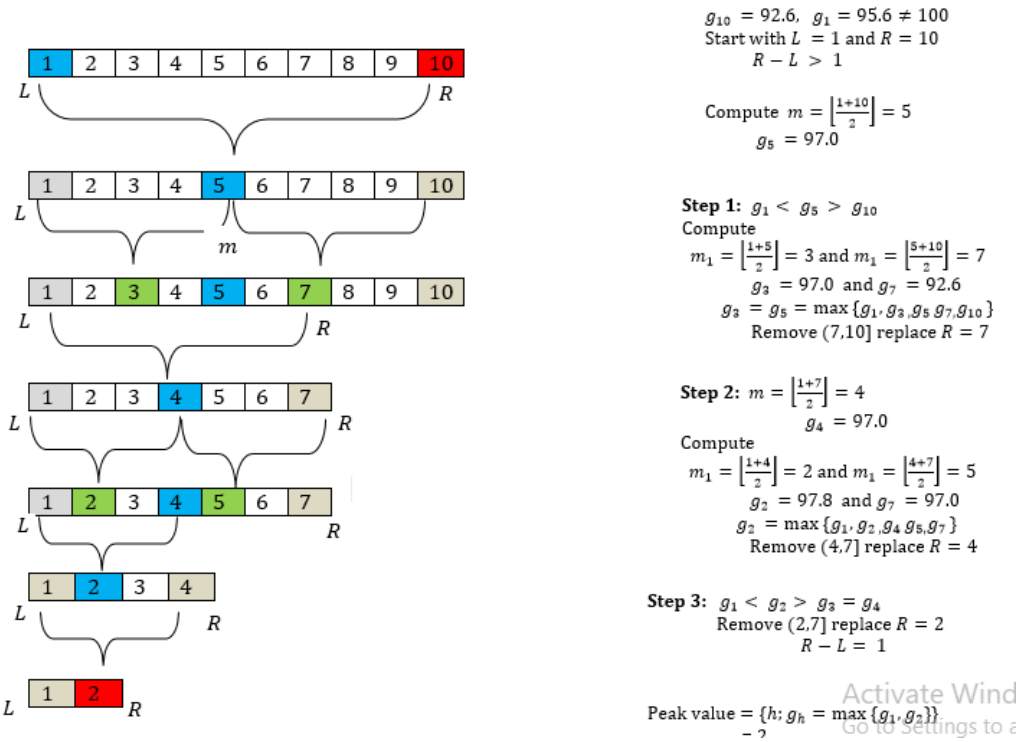


Figure B. 6: Illustration of Climate problem

The generalization of all the networks with hidden layers 1-10 were measured and the results are given in the table. These results verifies the above peak value.

Table B. 7: Generalization of Climate problem

Data Set	***P***	**No. of Hidden Layers**									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Climate	400	95.6	**97.7**	97.0	97.0	97.0	92.6	92.6	92.6	92.6	92.6

B.5 Flare Data Sets

The procedure of obtaining Flare I is described in the section 6.5.2. Unlike Cancer and Card problems, Flare set show its highest performance when there are 50% neurons in the training set. However, Flare I, II and III show there best performance at 7 hidden layer ANN architecture while highest performance of Flare IV gives it at 6 hidden layers. Process of Flare data sets start with 12 hidden layer networks and they show that $g_1 < g_6 < g_{12}$. Hence, the interval [1,6) remove and continue the procedure. However, from 7 layers all the networks give same performance. Therefore, it needs to compute generalization of all the networks which are having hidden layers 7 – 12. The Figure 6.9, and

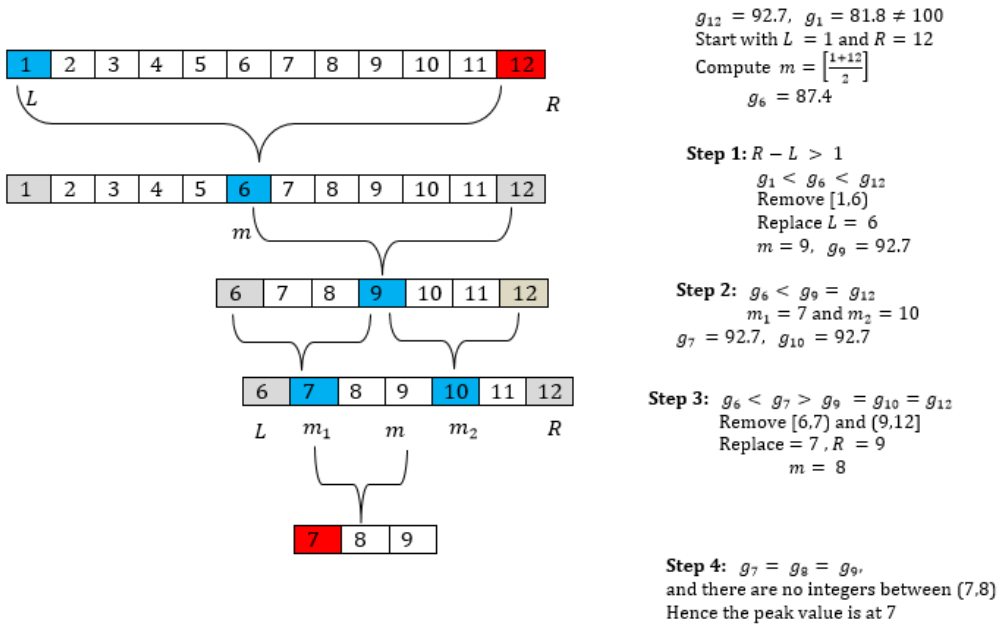


Figure B. 7 illustrate the achieving of hidden layers of Flare I and II.

The data of Table B. 8 depicts that Flare III and IV also have the same pattern.

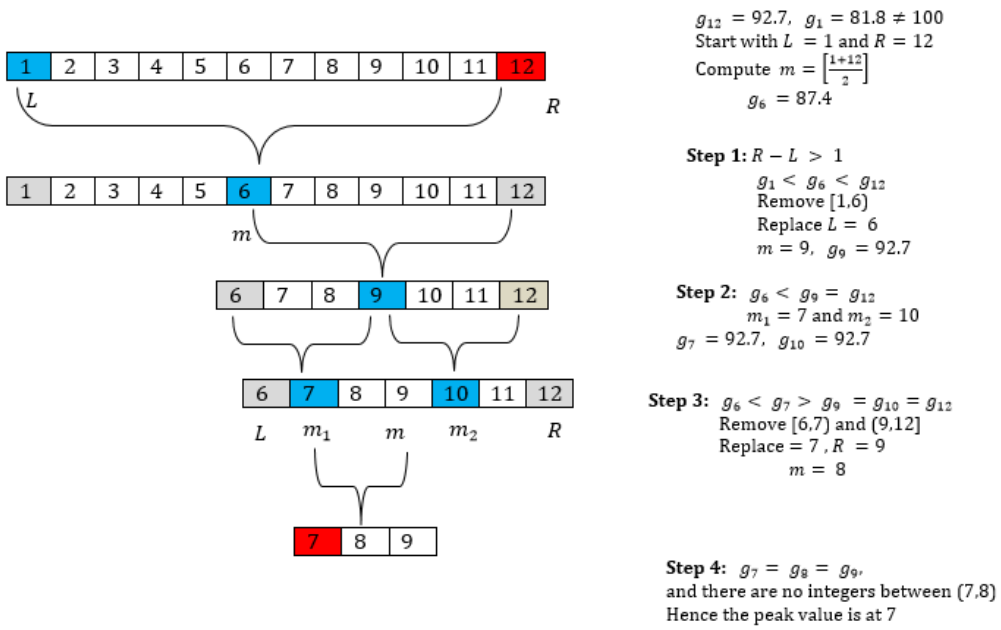


Figure B. 7: Illustration of Flare II problem

Table B. 8: Generalization of Flare I – IV problems

Data Set	*P*	No. of Hidden Layers											
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Flare I	800	71.1	74.4	78.2	82.0	80.1	80.1	**89.4**	89.4	89.4	89.4	89.4	89.4
Flare II	533	81.8	79.2	80.1	83.0	83.5	87.4	**92.7**	92.7	92.7	92.7	92.7	92.7
Flare III	266	82.5	79.1	83.2	82.8	88.9	82.8	**92.1**	92.1	92.1	92.1	92.1	92.1
Flare IV	50	68.4	83.1	80.2	81.6	80.7	**91.1**	91.1	91.1	91.1	91.1	91.1	91.1

B.6 Monk's Problems

In the Monk's problem there are 3 different data sets Monk's 1, Monk's 2 and Monk's 3 which have described in the Appendix 1. The two sets Monk's 1 and Monk's 3 perform in a similar pattern. The initial networks of both of them show 100% generalization. The Monk's 1 problem started with 10 hidden layers while Monks's 3 started with 12 hidden layers. The total number hidden neurons in both the sets were 120 which were distributed equally among the hidden layers. At the end they could reduce to networks with 7 and 6 hidden layers respectively. The

process of determining the hidden layers of these two sets are same as that of the Flare I data set.

However, Monk's 2 set shows different behavior. It starts with the network with12 hidden layer which contains 169 hidden neurons distributed arbitrary as shown in the Table B.. The generalization of the initial network was 67.1%. The data set shows its highest performance 86.8% with 4 hidden layers. The process of achieving this is shows in the Figure. The Table B.9 confirms this result

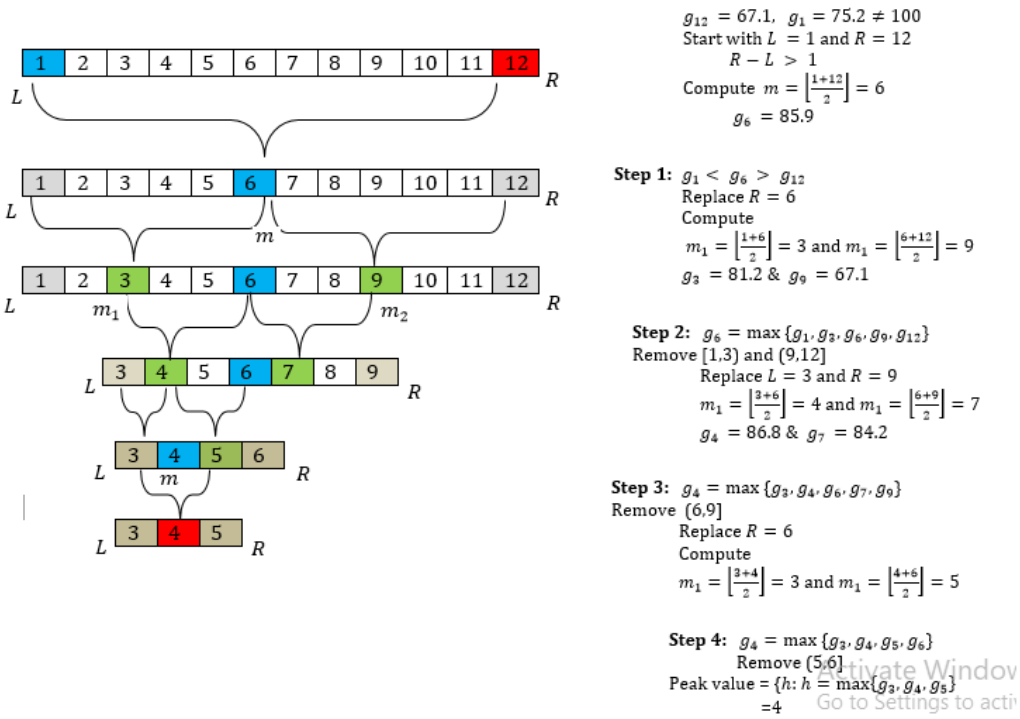


Figure B. 8: Illustration of Monk's 2 problem

Table B. 9: Generalization of Monk's 2 problems

Data Set	No. of Hidden Layers											
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Generalization	75.2	81.2	81.2	**86.8**	85.4	85.9	84.3	67.1	67.1	67.1	67.1	67.1
No. of hidden neurons	22	20	18	18	17	15	15	14	13	10	5	2

B.7 Seeds Data Set

The seeds problem started with 130 hidden neurons distributed with 10 hidden layers. The generalization of the initial network was 0.0 present. The network which shows the highest generalization has 7 hidden layers and 52 hidden neurons. The procedure of obtaining this is shown in the Figure B. 9. The values of the Table B. 10 verifies the results. It shows that until 7 layers generalization increases and then from the 8 layer onwards it becomes zero.

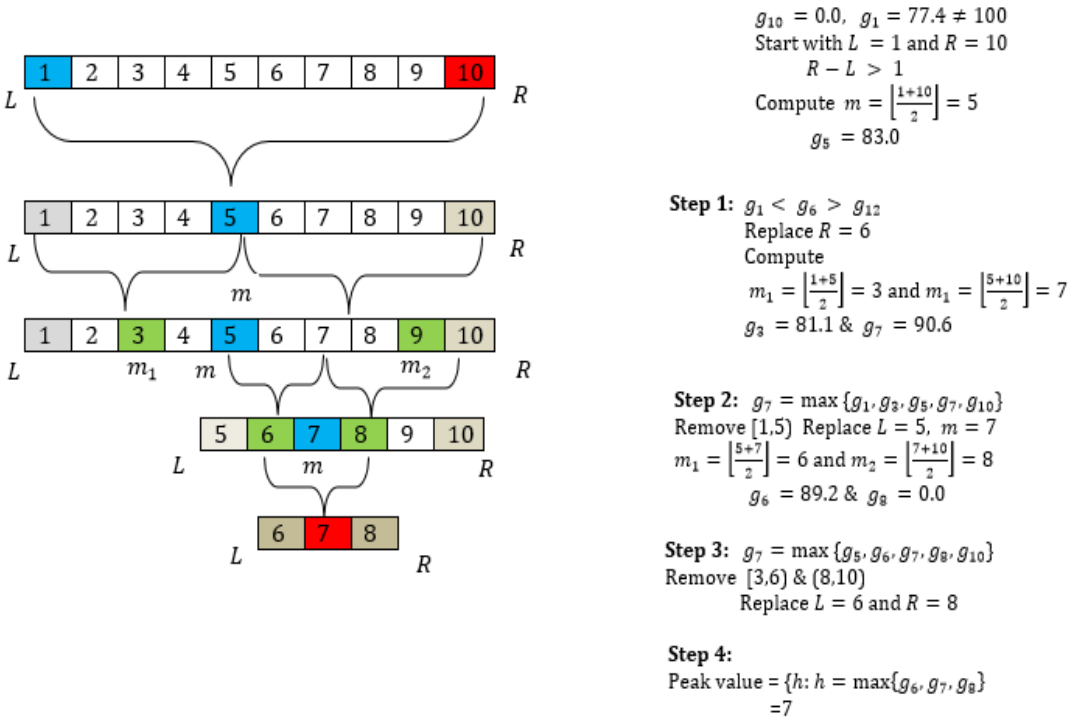


Figure B. 9: Illustration of Seeds problem

Table B. 10: Generalization of Seeds problem

Data Set	*P*	No. of Hidden Layers									
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Seeds	130	77.4	79.2	81.1	79.2	83.0	89.2	**90.6**	0.0	0.0	0.0

B.8 Yeast Data Set

The initial network of this data set contained 12 hidden layers and 1092 hidden neurons distributed in ascending order. However, at the first step of PSA it showed that the performance of the single hidden layer network was 100%. Hence, the process stopped at this stage. Therefore, the best architecture of this problem is network with 1 hidden layer and 14 hidden neurons [207] .

Appendix C

SELECTED CODES

C1. Introduction

This appendix presents the implementation of system while highlighting the most important functions. The system is developed with MATLAB programming. The Excel files are used to store the initial data.

C2. Import the data

The raw data are stored in the excel file as shown in the Figure C. 1 below.

B	C	D	E	F	G	H
Iris Data set						
Input				Output		
6.4	3.2	4.5	1.5	0	1	0
6.1	2.8	4	1.3	0	1	0
6.4	2.8	5.6	2.1	0	0	1
6.3	3.4	5.6	2.4	0	0	1
5.6	3	4.1	1.3	0	1	0
6.7	3.3	5.7	2.5	0	0	1
5.1	3.7	1.5	0.4	1	0	0
6.9	3.1	4.9	1.5	0	1	0
5.4	3.4	1.7	0.2	1	0	0
6.7	3	5	1.7	0	1	0
6.6	2.9	4.6	1.3	0	1	0
5.9	3.2	4.8	1.8	0	1	0

Figure C. 1. Raw data in excel worksheet

Firstly import the raw data to Matlab (Figure C. 2) and normalized them (Figure C. 3). However, if the inputs are 0 and 1, it proceeds without normalization.

```
train_inp = xlsread('worksheet_iris_unedited.xlsx', 1, 'B2:E151');
train_out = xlsread('worksheet_iris_unedited.xlsx', 1, 'F2:H151');
```

Figure C. 2: Import raw data from excel

```
%inputs
mu_inp = mean(train_inp);
sigma_inp = std(train_inp);
train_inp = (train_inp - repmat(mu_inp,[patterns 1])) ./ repmat(sigma_inp,[patterns 1]

%outputs
train_out = train_out';
mu_out = mean(train_out);
sigma_out = std(train_out);
train_out = (train_out(:,:) - mu_out(:,1)) / sigma_out(:,1);
train_out = train_out';
```

Figure C. 3: Normalization of inputs and outputs

After deciding number of layers n , initial weights are randomly generated as depicted in the Figure C. 4 between layer h_i layer and layer h_{i+1} layer. where $i = 1, 2, \dots, n$

```
for i=1:n-1

  [V]= (randn(h(i+1),h(i)) - mu)/sigma;
xlswrite('weights.xlsx', [V],i);

end
```

Figure C. 4: Generate random weights

C3. Backpropagation Algorithm

Now network has created and ready to train by the backpropagation algorithm. Thus, feed the inputs and obtain the output of each neuron using the codes given in the Figure C. 5 and compute the output error. Next compute the delta of each neuron and update the weights (Figure C. 6).

```
I = train_inp(p,:);%input vector=this pattern
T = train_out(p,:); %output = act
%calculate the current error for this pattern
H=I';
for i=1:n-2,
    V=xlsread('W.xlsx', i);
    H = logsig(V*H);
    xlswrite('H.xlsx', [H],i);
end
V=xlsread('W.xlsx', n-1);
         pred = xlsread('W.xlsx', n-1)*H;
         E = T'-pred;%error
```

Figure C. 5: Calculating error of a training cycle

```
% adjust weight hidden - output
D=E;%delta value of the output layer=Delta_out
V=xlsread('W.xlsx', n-1); %kk=2
    H=xlsread('H.xlsx', n-2);
    D=(H.*(1-H)).*(D'*V)';
    DW=eta*(E*H');

    V=V+DW;

    V1=V;
   xlswrite('W.xlsx', [V1],n-1);

for k=1:n-3
    kk=n-2-k;

       H=xlsread('H.xlsx', kk);
       V=xlsread('W.xlsx', kk+1);
        V1=V;
       DW=eta*(H*D');
       V=V+DW';
       D=(H.*(1-H)).*(D'*V1)';
    end

  DW= (eta*(I'*D'))';
  V = xlsread('W.xlsx', 1) + DW;
```

Figure C. 6: Updating weights using delta values

C4. Removing Hidden Neurons

While removing the neurons first it identifies the removable neurons by using the correlation coefficient of sum of delta value and output error. If there is no considerable correlation, it assumes correlation is 0. Then neurons which have infinitesimal delta values are identified as removable neurons which describes in the Figure C. 7.

```
if (CORR_1==0),
      M=0;
      A=abs(delta_H1)';
           B=sort(A);
     for i=1:sigma_1;
                    if (numel(B)>2)
                            mm=min(B);
                            m=find(A==mm);
                            M=[M,m];
                            B(1)=[];
                            size(B);
                    end

     end
```

Figure C. 7: Identify removal neuron when delta is zero

When there is considerable positive correlation (i.e. CORR_1=1) between summation of delta values and the output error, neurons with positive delta values which are very closed to zero are identified as removable neurons Figure C. 8. Similarly, when correlation is negative, neurons with negative delta values which are very close to zero are identified as removable neurons.

```
else if (CORR_1==1),
        M=0;
        A=delta_H1;
              for ii=1:(pp_1-1),
                  mm=min(deltaH1_posit);
                  m=find(A==mm);
                  mb=find(deltaH1_posit==m)
                  M=[M,m];
                  deltaH1_posit(mb)=[];
              end
```

Figure C. 8: Identify removal neuron when delta is a positive value

Then remove the neurons from each layer while merging the similar neurons as shown in the figures Figure C. 9 and Figure C. 10.

```
INweight_input_hidden1=weight_input_hidden1;
weight_input_hidden1(REMOVE_1,:)=[];
ADweight_input_hidden1=weight_input_hidden1;
NEW_weight_input_hidden1=weight_input_hidden1;
product_11=0;
for ii=1:numel(REMOVE_1)
   product=0;
    ii;
    v=REMOVE_1(ii);
    for jj=1:size(ADweight_input_hidden1,1);
        jj;

        product_11(jj)=sum(INweight_input_hidden1(v,:).*ADweight_input_hidden1(jj,:));
    end
 kk=max(product_11);
 m=find(product_11==kk);
 replace=NEW_weight_input_hidden1(m,:)+INweight_input_hidden1(v,:);
 NEW_weight_input_hidden1(m,:)=replace;
end
```

Figure C. 9: Removing unimportant neurons

```
       INweight_hidden1_hidden2= weight_hidden1_hidden2;
  weight_hidden1_hidden2(:,REMOVE_1)=[];
 ADweight_hidden1_hidden2 =weight_hidden1_hidden2;
 NEW_weight_hidden1_hidden2=weight_hidden1_hidden2;
 product_21=0;
 for iii=1:numel(REMOVE_1)
    product_21=0;
     iii;
     vv=REMOVE_1(iii);
     for jjj=1:size(ADweight_hidden1_hidden2,2);
         jjj;

         product_21(jjj)=sum(INweight_hidden1_hidden2(:,vv).*ADweight_hidden1_hidden2(:,jjj));
     end

 kkk=max(product_21);
 mm=find(product_21==kkk);
 replace2=NEW_weight_hidden1_hidden2(:,mm)+INweight_hidden1_hidden2(:,vv);
 NEW_weight_hidden1_hidden2(:,mm)=replace2;
end

  weight_input_hidden1 = NEW_weight_input_hidden1;
                   weight_hidden1_hidden2=NEW_weight_hidden1_hidden2;
  H_1=size(weight_input_hidden1,1);
```

Figure C. 10: Merging similar neurons

Appendix D

PUBLICATIONS

1. N. M. Wagarachchi and A. S. Karunananda, “Mathematical Modelling of Hidden Layer Architecture in Artificial Neural Networks", 3rd International Conference on Information Security and Artificial Intelligence (ISAI 2012) DOI: 10.7763/IPCSIT.2012.V56.28 IPCSIT vol. 56 IACSIT Press, Singapore, pp. 154-159, 2012

2. N. M. Wagarachchi and A. S. Karunananda, “Optimization of multi-layer artificial neural networks using delta values of hidden layers,” in Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB), 2013 IEEE Symposium on, 2013, pp. 80–86.

3. N. M. Wagarachchi and A.S. Karunananda, “ A Novel Technique for Optimizing the Hidden Layer Architecture in Artificial Neural Networks”, American International Journal of Research in Science, Technology Engineering and Mathematics, Issue 4, vol. 1, pp 1-6, November 2013.

4. N. M. Wagarachchi and A.S. Karunananda, “A Theoretical Basis for the Optimization of Hidden Layer Architecture in Artificial Neural Networks, HETC symposium 2014, July 2014. (Abstract only)

5. Mihirini Wagarachchi and Asoka Karunananda,” Towards a Theoretical Basis for Modelling Hidden Layer Architecture in Artificial Neural Networks”, 2nd International Conference on Advances Computing, Electronics and Communication, Switzerland, 2014.

6. N. M. Wagarachchi and A.S. Karunananda, “Optimization of Artificial Neural Network Architecture Using Neuroplasticity,” Int. Journal of Artificial Intelligence. vol. 15, no. 1, pp. 112–125, 2017.

7. Mihirini Wagarachchi and Asoka Karunanda “Modelling Modeling of Hidden layer Architecture in Multilayer Artificial Neural Networks” SLAAI International Conference on Artificial Intelligence (SLAAI – ICAI – 2018), University of Moratuwa , Sri Lanka.