# Biologically Inspired Modular Neural Networks

Farooq Azam

Dissertation submitted to the Faculty of the Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy
in
Electrical and Computer Engineering

May, 2000
Blacksburg, Virginia

Keywords: Biologically inspired neural networks, modular neural networks, principle of divide and conquer, a priori expert knowledge, artificial neural networks, robustness, accuracy, and generalization.

# Biologically Inspired Modular Neural Networks

Farooq Azam

(ABSTRACT)

This dissertation explores the modular learning in artificial neural networks that mainly driven by the inspiration from the neurobiological basis of the human learning. The presented modularization approaches to the neural network design and learning are inspired by the engineering, complexity, psychological and neurobiological aspects. The main theme of this dissertation is to explore the organization and functioning of the brain to discover new structural and learning inspirations that can be subsequently utilized to design artificial neural network.

The artificial neural networks are touted to be a neurobiologicaly inspired paradigm that emulate the functioning of the vertebrate brain. The brain is a highly structured entity with localized regions of neurons specialized in performing specific tasks. On the other hand, the mainstream monolithic feed-forward neural networks are generally unstructured black boxes which is their major performance limiting characteristic. The non explicit structure and monolithic nature of the current mainstream artificial neural networks results in lack of the capability of systematic incorporation of functional or task-specific a priori knowledge in the artificial neural network design process. The problem caused by these limitations are discussed in detail in this dissertation and remedial solutions are presented that are driven by the functioning of the brain and its structural organization.

Also, this dissertation presents an in depth study of the currently available modular neural network architectures along with highlighting their shortcomings and investigates new modular artificial neural network models in order to overcome pointed out shortcomings. The resulting proposed modular neural network models have greater accuracy, generalization, comprehensible simplified neural structure, ease of training and more user confidence. These benefits are readily obvious for certain problems, depending upon availability and usage of available a priori knowledge about the problems.

The modular neural network models presented in this dissertation exploit the capabilities of the principle of divide and conquer in the design and learning of the modular artificial neural networks. The strategy of divide and conquer solves a complex computational problem by dividing it into simpler sub-problems and then combining the individual solutions to the sub-problems into a solution to the original problem. The divisions of a task considered in this dissertation are the automatic decomposition of the mappings to be learned, decompositions of the artificial neural networks to minimize harmful interaction during the learning process, and explicit decomposition of the application task into sub-tasks that are learned separately.

The versatility and capabilities of the new proposed modular neural networks are demonstrated by the experimental results. A comparison of the current modular neural network design techniques with the ones introduced in this dissertation, is also presented for reference. The results presented in this dissertation lay a solid foundation for design and learning of the artificial neural networks that have sound neurobiological basis that leads to superior design techniques. Areas of the future research are also presented.

# Acknowledgments

This document would never have been complete without the support, encouragement, guidance of my mentor and Ph.D. committee chair, Professor Hugh F. VanLandingham. He was always the voice of reason and inspiration; and knew just what to say to make me believe in myself. I would like to express my deep gratitude and thanks to him for the freedom he provided me throughout the course of my Ph.D. studies to explore different areas of my academic interest. He was a source of guidance, support and help throughout my stay at Virginia Tech. He was the one who introduced me to the area of soft computing. His encouragement and advise always provided me with new prospectives in the field of my research. The expertise that he shared me will remain a tremendous source of professional growth to me.

I am thankful to all of the members of my Ph.D. committee: Dr. John S. Bay, Dr. William T. Baumann, Dr. Peter M. Athanas and Dr. William R. Saunders. Their expertise and assistance played an important role in the progress of my research. Specially, I would like to thank to Dr. Bay, Dr. Athanas and Dr. Baumann, with whom I had invaluable discussions on academic matters or otherwise. Their advice and guidance has tremendous influence on my professional and personal growth.

Foremost, my parents have always encouraged me to follow my dreams and deserve special thanks. I wish to thank them who always provided me with invaluable guidance and without whose tangible and intangible support my dream of Ph.D. degree would not have come true and my academic endeavors would not have been as fulfilling. I dedicate this work to them.

Last, but not the least, I want to thank both the DuPont Office of Education and the Office of Naval Research (Grant No. N00014-98-1-0779) for partial support during the progress of my work.

# Contents

# List of Figures

# List of Algorithms

# Chapter 1

# Introduction

Artificial intelligence is the study of intelligent behavior and how computer programs can be made to exhibit such behavior. There are two categories of artificial intelligence from the computational point of view. One is based on *symbolism*, and the other is based on *connectionism*. In the former approach intelligence is modeled using symbols, while the latter models intelligence using network connections and associated weights. Although these approaches evolved via different routes, both have been successfully applied to many practical problems. In contrast to the symbolic approach, the connectionist approach adopts the brain metaphor which suggests that intelligence emerges through a large number of interconnected processing elements in which any individual processing element performs a simple computational task. The weights of the connections between processing elements encode the long term knowledge of a network. The most popular and widely used connectionist networks are multi-layered perceptrons. The artificial neural network approach is synonymous with the connectionist approach. In the connectionist approach there is no separation between knowledge and inference mechanism, in contrast to the symbolic approach in which knowledge acquisition is separate from the inference mechanism. In recent years the artificial neural network approach has posed a serious challenge to the domination of the symbolic approach in the field artificial intelligence due to the ease of knowledge acquisition and manipulation mechanisms. Motivation for neural network research is multi-faceted. Some of these inspirations are biological and others are due to human ambition of understanding the principles of intelligent behavior for constructing artificially intelligent systems. The goal of artificial neural network research is to study how various types of intelligent behavior can be implemented in systems made up of neuron-like processing elements and brain-like structures.

## 1.1 Brief History of Artificial Neural Networks

The origins of the concept of artificial neural networks can be traced back more than a century as a consequence of man's desire for understanding the brain and emulating its behavior. A century old observation by William James that the *amount of activity at any given point in the brain cortex is the sum of the tendencies of all other points to discharge into it*, has been reflected subsequently in the work of many researchers exploring the field of artificial neural networks. About half a century ago the advances in neurobiology allowed researchers to build mathematical models of the neurons in order to simulate the working model of the brain. This idea enabled scientists to formulate one of the first abstract models of a biological neuron, reported in 1943. The formal foundation for the field of artificial neural networks was laid down by McCoulloch and Pitts [1]. In their paper the authors proposed a computational model based on a simple neuron-like logical element. Later, a learning rule for incrementally adapting the connection strength of these artificial neurons was proposed by Donald Hebb [2]. The Hebb rule became the basis for many artificial neural network research models.

The main factor responsible for the subsequent decline of the field of artificial neural networks was the exaggeration of the claims made about the capabilities of early models of artificial neural networks which cast doubt on the validity of the entire field. This factor is frequently attributed to a well known book by Minsky and Papert [3] who reported that perceptrons were not capable of learning moderately difficult problems like the XOR problem. However, theoretical analyses of that book were not alone responsible for the decline of research in the field of artificial neural networks. This fact, amplified by the frustration of researchers, when their high expectations regarding artificial neural network based artificial intelligent systems were not met, contributed to the decline and discrediting of this new research field. Also, at the same time logic based intelligent systems which were mostly based on symbolic logic reached a high degree of sophistication. Logic based systems were able to capture certain features of human reasoning, like default logic, temporal logic and reasoning about beliefs. All these factors lead to the decline of research in the field of artificial neural network to a point where it was almost abandoned.

Since the early 1980s there has been a renewed interest in the field of artificial neural networks that can be attributed to several factors. The shortcomings of the early simpler neuron network models were overcome by the introduction of more sophisticated artificial neural network models along with new training techniques. Availability of high speed desktop digital computers made the simulation of complex artificial neural network models more convenient. During the same time frame significant research efforts of several scientists helped in restoring the lost confidence in this field of research. Hopfield [4] with his excellent research efforts is responsible for revitalization of ANN field. The term connectionist was made popular by Feldman and Ballard [5]. Confidence in the artificial neural network field was greatly enhanced by the research efforts of Rumelhert, Hinton and Williams [6] who developed a generalization of Widrow's delta rule [7] which would make multi-layered artificial neural network learning possible. This was followed by demonstrations of artificial neural network learning of difficult tasks in the fields of speech recognition, language development, control systems and pattern recognition, and others.

Since the mid 1980s research in the area of artificial neural networks has experienced an extremely rapid growth for different reasons which is evident by its interdisciplinary nature. These disciplines range from physics and engineering to physiology and psychology. In recent years there has been a lot of progress in the development of new learning algorithms, network structures and VLSI implementations for artificial neural networks.

## 1.2 Artificial Neural Networks

Artificial neural networks, sometimes in context referred to only as neural networks, are information processing systems that have certain computational properties analogous to those which have been postulated for biological neural networks. Artificial neural networks generalize the mathematical models of the human cognition and neural biology. The emulation of the principles governing the organization of human brain forms the basis for their structural design and learning algorithms. Artificial neural networks exhibit the ability to learn from the environment in an interactive fashion and show remarkable abilities of learning, recall, generalization and adaptation in the wake of changing operating environments.

Biological neural systems are an organized and structured assembly of billions of biological neurons. A simple biological neuron consists of a cell body which has a number of branched protrusions, called *dendrites*, and a single branch called the *axon* as shown in Figure 1.1. Neurons receive signals through the dendrites from neighboring connected neurons. When these combined excitations exceed a certain threshold, the neuron fires an impulse which travels via an axon to the other connected neurons. Branches at the end of the axons form the synapses which are connected to the dendrites of other neurons. The synapses act as the contact between neurons and can be excitatory or inhibitory. An excitatory synapse adds to the total of signals reaching a neuron and an inhibitory synapse subtracts from this total. Although this description is very simple, it outlines all those features which are relevant to the modeling of biological neural systems using artificial neural networks. Generally, artificial neuron models ignore detailed emulation of biological neurons and can be considered as a unit which receives signals from other units and passes a signal to other units when its threshold is exceeded. Many of the key features of artificial neural network concepts have been borrowed from biological neural networks. These features include local processing of information, distributed memory, synaptic weight dynamics and synaptic weight modification by experience. An artificial neural network contains a large number of simple neuron-like processing units, called *neurons* or nodes along with their connections. Each connection generally “points” from one neuron to another and has an associated set

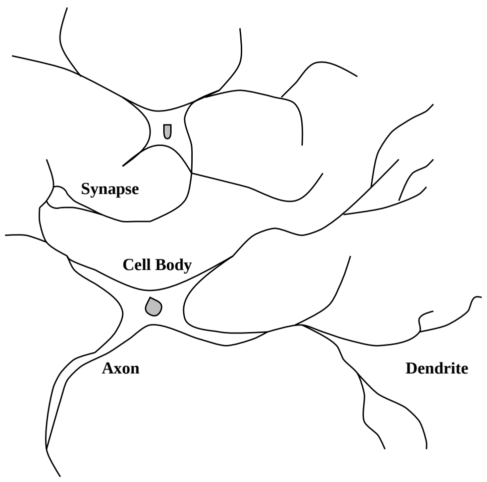


Figure 1.1: Biological processing element (neuron).

of weights. These weighted connections between neurons encode the knowledge of the network. Figure 1.2 illustrates a mathematical model of an artificial neuron corresponding to the biological neuron of Figure 1.1. The simplest neural network model is called a *perceptron* which can

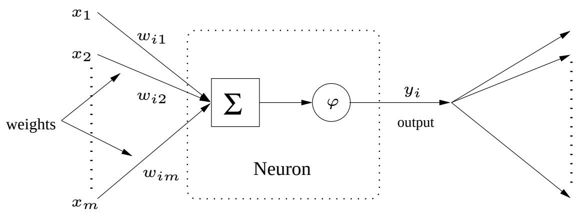


Figure 1.2: An artificial neuron.

be trained to discriminate between linearly separable classes using any of the sigmoidal nonlinearities as an output function. However, for more complex tasks of classification or regression, models with multiple layers of neurons are necessary. This artificial neural network model is generally referred to as a multi-layer perceptron (MLP). This architecture consists mainly of three types of neuron layers, namely input layer, hidden layer(s) and an output layer. The nodes in an input layer are called input neurons or nodes; they encode the data presented to the network for processing. These nodes do not process the information, but simply distribute the information to other nodes in the next layer. The nodes in the middle layers, not directly visible from input or output, are called hidden nodes. These neurons provide the nonlinearities for the network and compute an internal representation of the data. The nodes in the output layer are referred to as output neurons: they encode possible desired values assigned to the input data. A typical multi-layer perceptron neural network model is shown in Figure 1.3. $i^{th}$ node output is determined

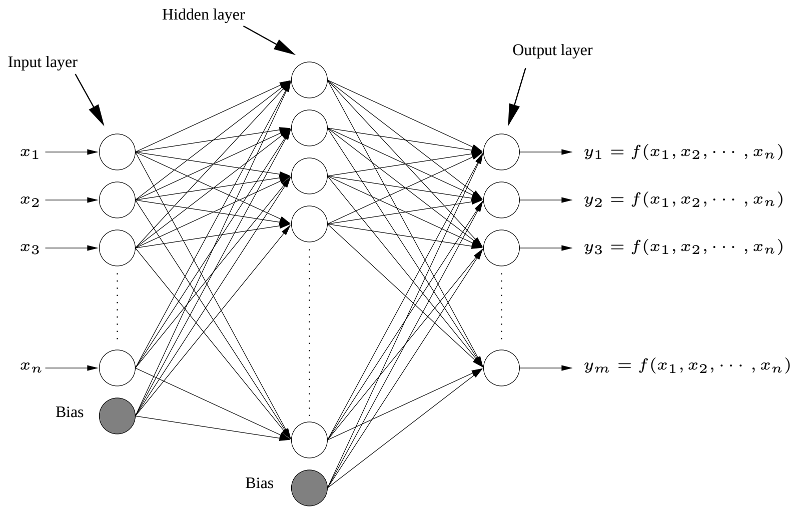

Hidden layer
Input layer
Output layer
$x_1$
$x_2$
$x_3$
$x_n$
Bias
Bias
$y_1 = f(x_1, x_2, \cdots, x_n)$
$y_2 = f(x_1, x_2, \cdots, x_n)$
$y_3 = f(x_1, x_2, \cdots, x_n)$
$y_m = f(x_1, x_2, \cdots, x_n)$

Figure 1.3: A typical multi layered perceptron model.

by

$$y_i(t+1) = \varphi(\sum_{j=1}^{m} w_{ij}x_j(t) - \theta_i)$$

where $x_j$ is the $j^{th}$ component of the input vector to the neuron. It consists of an activation function $z = \psi(x_1, \cdots, x_n) : \mathfrak{R}^n \rightarrow \mathfrak{R}$ and a possibly non-linear output or transfer function $\varphi(z) : \mathfrak{R} \rightarrow \mathfrak{R}$. The most commonly used activation function is $\psi(x_1, \cdots, x_n) = \sum_{i=1}^{N} w_i x_i$. Output functions which are widely used in the neural networks community are *linear*, *sigmoidal* and *tansigmoidal*.

Associated with each neuron is a weight vector $w_{ij}$ which represents the strength of the synapse connecting neuron $j$ to neuron $i$. A positive synaptic weight corresponds to an excitatory synapse; a negative weight corresponds to an inhibitory synapse and a zero-valued synaptic weight indicates no connection between the two neurons. Sometimes an additional constant is used as a part of the weight vector, this weight is called a *bias* with a fixed input value of $1$ or $-1$ in order to extend the model from linear to an affine transformation. Learning algorithms estimate these weights iteratively by minimizing a given cost function, defined in terms of the error between the desired and neural network model outputs.

An artificial neural network model is determined by three basic attributes; namely, types of neurons, types of synaptic interconnections and structure, and the type of learning rule used for updating neural network weights.

The types of neurons determine the appropriateness of an artificial neural network model for a given task. Neurons performs two major tasks, i.e., processing of inputs to a neuron and calculation of the output. Input processing is basically the combination of the inputs to form the neural network input. This combination could be linear, quadratic or spherical in nature. The second part of a neuron is responsible for generation of the activation level for the neuron as a linear or nonlinear output function of its net input. Commonly used output functions are step, hard limiting, unipolar sigmoidal and bipolar sigmoidal.

An artificial neural network is comprised of a set of highly interconnected nodes. The connections between nodes, called weights, carry activation levels from one neuron to another or to itself. According to the interconnection scheme used for a neural network, it can be categorized as a feed forward or a recurrent neural network. In the feed forward model all the interconnecting weights point in only one direction, i.e., strictly from input layer to the output layer. The multi-layered feed forward neural networks and self-organizing networks, competitive learning networks, Kohonen networks, principal component analysis networks, adaptive resonance networks and reinforcement learning networks are examples of this large class of networks. On the other hand in recurrent neural network models, there are feedback connections. Hopfield networks and Boltzmann machines are major networks belonging to this category of artificial neural networks. Also, the layout of the neural network weights determine whether the weights of a given neural networks are symmetrical or asymmetrical. In symmetrical configurations, the interconnecting weights between the two neurons are the same and in asymmetrical configuration these weights are not necessarily the same. The sparsity of interconnection weight arrays determines whether a neural network is a fully connected neural network, or not.

Learning in artificial neural networks refers to a search process which aims to find an optimal network topology and an appropriate set of neural network weights to accomplish a given task. After a successful search process, the final topology, along with the associated weights constitutes a network of the desired utility for a given task. The process of learning is equivalent to a search for an appropriate candidate neural network model from a search space comprising all topologies and corresponding weights. The process of searching for a sets of network weights for a given topology is typically far simpler than searching for a topology. For this reason, it is common for trial topologies to be chosen either randomly or using a rule-of-thumb. The more conventional learning process in artificial neural networks can be distinctly divided into supervised, reinforcement and unsupervised learning methods. Moreover, the use of evolutionary computation in neural network optimization is steadily gaining popularity because of the global search capabilities of evolutionary computation.

In the supervised learning paradigm, the desired output values are known for each input pattern. At each instant of time, when an input pattern is applied to an artificial neural network, the parameters of the neural network are adapted according to the difference between the desired value and the neural network output. The most commonly used supervised training algorithm for the update of weights of a multi-layer perceptron model is an iterative algorithm called error back-propagation which is an error correction rule. This algorithm is based on the chain rule for the derivative of continuous functions. The algorithm consists of a forward pass, in which training examples are presented to the network and activations of output neurons are computed. This is followed by a backward error propagation pass in which weights of neurons are updated using the gradient of a cost function, such as the sum-squared error between network outputs and desired target outputs.

Unsupervised learning, sometimes referred to as self-organized learning, is used to determine regularities, correlations or categories inherent in the input data set. In this learning scheme there is no feedback from the environment indicating what the outputs should be or whether the output of the neural network is a correct, or not.

Reinforcement learning is a combination of supervised and unsupervised learning schemes. In reinforcement learning, after the presentation of each pattern to the artificial neural network, the neural network model is provided with information about its performance in a supervised manner, but this information is very restricted in form. It consists of a qualitative evaluation of the neural network response and indicates whether the performance of the neural network to a particular pattern was “good” or “bad”. The learning algorithm has to make the best of this information during the search process for an optimal set of weights, typically by simply making good associations more probable.

Artificial neural networks have already found applications in a wide variety of areas. They are used in automatic speech recognition [8], handwritten character recognition [9], optimization [4, 10], robotics [11, 12], financial expert systems [13], system identification and control [14], statistical modeling [15], and other artificial intelligence problems.

## 1.3 Properties of Artificial Neural Networks

Some of the important inherent properties of artificial neural network models which make them an attractive tool for artificial intelligence and other interdisciplinary tasks are briefly described below.

1. **Generalization:** For real world data some type of continuous representation and/or inference systems are required such that for similar inputs or situations, outputs or inferences are also similar. This is an inherent property of neural networks with continuous activation functions. Statistically speaking discrete neural networks with a sufficient number of units

can exhibit the same behavior. This neural network property assures a smooth generalization from already learned cases to new ones.

2. **Graceful degradation:** The problems associated with generalization become worse if the data are incomplete or inaccurate and is usually refers to the failure of the weights. Due to the existence of the *similarity concept* in neural networks, if a small portion of the input data is missing or distorted, performance deteriorates only slightly. Performance degradation is proportion to the extent of data inaccuracy or incompleteness.

3. **Adaptation and learning:** Learning and adaptation in neural networks inherently exist because knowledge is indirectly encoded into most neural network models by the data that are specific to a particular situation under study; and, neural network models try to maintain that knowledge under changing operating conditions.

4. **Parallelism:** Virtually all of the neural network algorithms exhibit inherent parallelism. In most cases connecting weights associated with all neurons, or at least large groups of them can be updated simultaneously. This is an important property of neural networks from an implementation point of view. Since it is difficult to speed up single processing units, the alternative is to distribute computationally expensive tasks to large numbers of such units working in parallel.

## 1.4 Motivation and Scope

The main goal of research in the field of artificial neural networks is to understand and emulate the working principles of biological neural systems. Biological neural systems consist of billions of biological neurons interconnected via sets of individual synaptic weights. Recent advances in neurobiological sciences have given more insight into the structure and the workings of the brain. Research has also substantiated the fact that the brain is modular in nature with at least three hierarchical levels. At the fundamental level of the hierarchy are the individual neurons, the next level in ascending order is the micro-structural level and the top level is the macrostructure of the brain. Also, researchers believe that the brain is a system of interacting modules at the macro-structural level. Each module at the macro-structural level has its own micro-structure of various cell types and connectivity [16]. A sub-division of complex tasks into simpler tasks is also evident in human and animal brains [17].

Although the most commonly used monolithic architectures of artificial neural networks show inherent modularity at the microlevel with synaptic weights, neurons and layers of neurons forming the modular hierarchy, respectively, these architectures negate the fundamental structural organization of animal or human brain by not exhibiting modularity at the macro level. The task decomposition property is also non-existent in the prevalent commonly used global or monolithic artificial neural network models, even though this property is considered to be a fundamental property of biological neuronal systems.

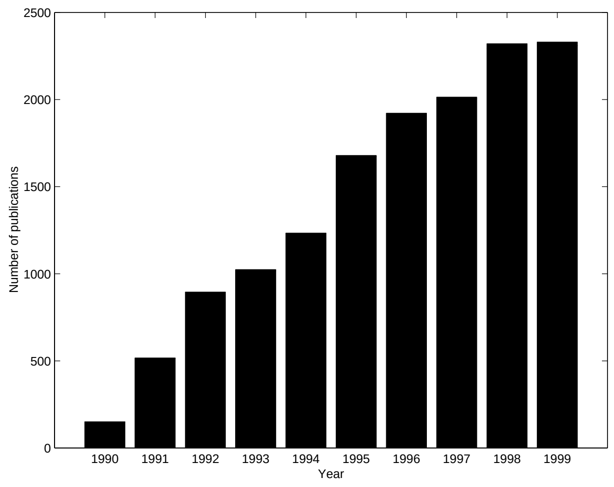
The inherent modularity in the human and animal brains enables them to acquire new knowledge without forgetting previously acquired knowledge. This “stability/plasticity” property is missing in the currently used monolithic artificial neural networks and is well documented and understood [18]. The modularity in brains makes knowledge retention possible in an environment where some of the environmental characteristics are changing by modifying only the neural pathways in a module that is responsible for responding to the changing environmental characteristics. Modularity in the brain is also responsible for optimization of the information capacity in the brain neural pathways. Also, researchers have indicated that the brain consists of a number of repeating modules. This fact is also not utilized in the design and learning of present day existing monolithic neural networks.

Artificial neural networks also suffer from the credit assignment problem. This problem arises when the size of an artificial neural network is large and the appropriate learning information is not made available effectively to the synaptic weights due to limitations of the existing learning algorithms. The credit assignment problem can be avoided by modularizing the artificial neural networks, thereby achieving individual neural networks which are simpler and of smaller size.

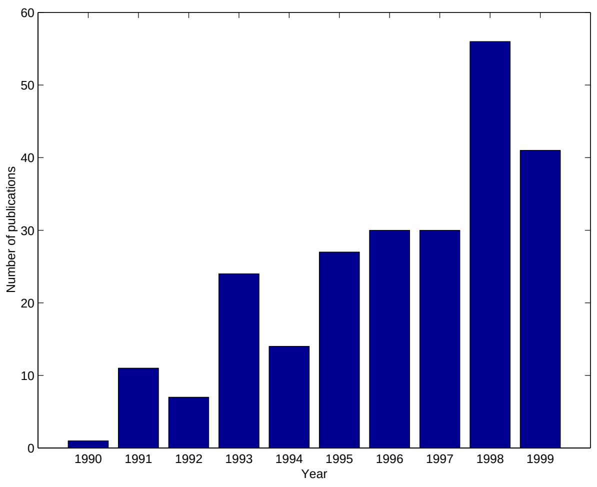
A monolithic artificial neural network can be viewed as an unstructured black box in which the only states which are observable are the input and output states. This makes an artificial neural network model void of any interpretability and generally renders it inappropriate for incorporating any functional *a priori* knowledge.

Considering the shortcomings of monolithic artificial neural networks mentioned above, modularization of artificial neural network design and learning seems to be an attractive alternative to the existing artificial neural network design and learning algorithms. The modularization of artificial neural networks helps to overcome certain problems mentioned in the previous paragraphs. Modular neural network architectures are a natural way of introducing a structure to the otherwise unstructured learning of neural networks. The incorporation of *a priori* knowledge is another major advantage of modular neural networks over monolithic artificial neural networks, thus facilitating design of efficient artificial neural networks which are functionally interpretable. Also, the reuse of already acquired knowledge leads modular neural networks to continual adaptation, thus economizing the re-learning process. Modular artificial neural network architecture tends to have smaller learning time due to the fact that a complex task is decomposed into simpler subtasks and each module has to learn only a part of the whole task. Also, this generally leads to relatively smaller modules and reduces the overall artificial neural network complexity.

Unfortunately, modular neural networks have been relatively neglected by researchers working in the field of artificial neural networks. All the books related to artificial neural networks seem to mention modular neural networks only in a very brief fashion. Although there has been a steady growth in research publications regarding modular artificial neural networks, these publications form only a minor percentage of the overall publications covering the field of artificial neural networks. To support this fact, a search was carried out to locate keywords from the published article titles, keywords, or abstracts and the publication data was generated using keywords *neural network*, *cooperative neural*, *competitive neural*, *hierarchical neural*, *structured neural*, and *modular neural*. The data was gathered from the Science Citation Index Expanded citation database provided by the Institute for Scientific Information (ISI), which contains information about 17 million published documents. This data base indexes more than 5,700 major journals across 164 scientific disciplines. The search data is shown graphically in Figure 1.4.



(a)



(b)

Figure 1.4: (a) Comparison of publications related to modular, relative to other artificial neural networks (white area indicates the publications related to modular artificial neural networks) (b) Publication related only to the modular artificial neural networks.

The numbers plotted in Figures 1.4(a) and (b) include publications regarding the keyword "modular" which may, or may not, be modular in the present context.

In light of the preceding discussion, there is a need to carry out research to explore the applicability of modularity in the field of artificial neural networks. The research needs to cover exploration for new modular artificial neural network architectures, along with their corresponding learning algorithms. This dissertation is an attempt in that direction. In this dissertation an effort is made to highlight the importance of modularity in the design and learning of artificial neural networks. Also, along with suggesting some improvements in existing design and learning techniques for modular artificial neural networks, new adaptive techniques for design and learning of modular neural networks are introduced.

## 1.5 Dissertation Organization

This dissertation addresses the issue of the modularity in the design and learning of the artificial neural networks. The remainder of the dissertation is organized as follows. Chapter 2 outlines shortcomings of the existing mainstream monolithic neural networks and justifies the need for modularity when designing artificial neural networks. Also, the advantages of using the modular approach to neural network design and learning over the exiting monolithic neural network models is discussed. A review of the available research literature about the modular neural networks is also presented in Chapter 2. In Chapter 3, a biologically plausible monolithic network model is presented that is inspired by the working of the brain and incorporates lateral inhibitory connections to introduce a sense of structure in a monolithic neural network. A modified version of a popular modular neural network model, the hierarchical mixture of experts model, is presented in Chapter 4 that draws its design and learning inspiration from the working principles of the successful knowledge creating companies, Bayesian methods and flow of the information in the brain. The shortcomings of the original hierarchical mixture of experts model and its variants are discussed. The proposed modified hierarchical mixture of experts model overcomes the pointed out shortcomings of the hierarchical mixture of experts model and is more inline with concepts well understood by the researchers in the field of artificial neural networks. Model selection problem is discussed in Chapter 5 and self-scaling version of the modular neural networks are described. The self-scaling models iteratively scale their structure to match the complexity of the problem at hand in order to solve it successfully. The conclusions and summary of the contributions of this dissertation are presented in Chapter 6.

# Chapter 2

# Modular Artificial Neural Networks

This chapter introduces and describes the concept of modularity and its applicability to the design and learning of artificial neural networks. The inspiration for modular design of neural networks is mainly due to biological and psychological reasons, namely, the functioning of the human and/or animal brain. Recent advances in neurobiological research have proven that the modularity is key to the efficient and intelligent working of human and animal brains. Economy of engineering design, psychological aspects and complexity issues in artificial neural network learning are the other motivational factors for modular design of neural networks. Since biological structure and functioning of animal and human brains form the basis of the artificial neural network design and learning, a brief description of the structure and working of the human and animal brains is presented. The motivations for the modularly designed artificial neural networks are listed. Also, the literature review of existing modular artificial neural network architectures is presented along with a discussion of pertinent issues related to the design and learning of modular neural networks.

## 2.1 The Brain

Biological brains are essentially general purpose, complex and efficient problem solving systems that provide living organisms with the trait of intelligence. The brain is generally viewed as a black box that receives an input in the form of a certain stimulus from the environment and produces a corresponding appropriate output or response. The environmental information is coded as neural signals in the brain. The brain uses this stored neural information in the form of patterns of neural firing to generate the desired actions that are most likely to accomplish the goal at hand. This type of behavior of the brain generally leads to a wrong conclusion that the brain is working as a whole [19]. The evidence of modularity in brain function can be attributed to two sources, neuropsychology and neurobiology. The experiments in neuropsychology indicate that a circumscribed brain lesion could cause a specific disorder of language while leaving other cognitive functions of brain intact. The neurobiologists on the other hand have long believed, and appreciated the fact, that the regions of animal and human brains are organized into specialist and functionally segregated modules [20, 21, 22, 23]. Recent advances in neurobiology indicate that animal brains exhibit a modular structure. The research indicates that animal and human brains are divided into major parts at the coarse-grain level. The human and animal brains are not comprised of monolithic homogeneous biological neural networks within these major parts, but instead, are comprised of specialized modules performing individual specialized tasks.

Neuroscience research also makes a strong case for the brain as a distributed, massively parallel and self-organizing modular system. This research indicates that biological brains are a combination of highly specialized, relatively independently working and hierarchically organized modules. Coherent functioning of massively distributed brain function is achieved as a result of the interconnecting reentrant signals which integrate different functional modules and different hierarchical levels [24, 23].

The functional segregation of the brain into independent functional and anatomical modules can be argued on the basis of a number of empirical evidences such as evolutionary process, economy and complexity [25].

The first and the most important of these arguments is the evolutionary process argument according to which the brain was developed into a complex modular system as a result of small and random changes through the process of the natural evolution. If the brain was a single complex and massively interconnected system of neurons, then a small change in a part of the this system would have lead to a need for some corresponding necessary changes in the interconnections through out the system to avoid any adverse effects of this change. This arrangement would not have led to an improvement of the brain through small changes during the process of evolution. On the other hand, if the brain was a modular system comprised of different independently working modules, then any change in the brain would be a local change and would keep the unaffected specialist modules intact. Since nature works in an efficient and simplistic fashion, modular evolution of brain approach seems to have occurred as compared to evolution of the brain as a single monolithic complex system [26, 27]. Also, the problem of catastrophic forgetting, i.e., drastic forgetting of old acquired knowledge while acquiring new knowledge, is not prevalent in higher mammalian brains because of their development of a hippocampal-neocortical separation. It is suggested that this was a result of evolution that two separate areas emerged in the brain, the hippocampus and the neocortex. The assumption about the brain being modular can be justified because of the following reasoning as well. Incremental acquisition of new knowledge is not consistent with the gradual discovery of new structures in brain and can lead to catastrophic interference with what has previously been learned. In view of this fact, it is concluded that the neocortex may be optimized for the gradual discovery of the shared structure of events and experiences, and that the hippocampal system is there to provide a mechanism for rapid acquisition of new information without interfering with the previously discovered regularities. After this initial acquisition, the hippocampal system serves as a teacher to the neocortex [28].

The second argument is economically driven. The different knowledge representations in the brain are stored in different specialist brain modules comprised of neurons with the same type of neuronal structures and interconnecting weights. This type of arrangement provides the brain with a capability of information storage where learned environmental knowledge can only be effectively stored and managed if the same types of neuron with similar interconnections are grouped together based on their functionality [29].

The complexity argument also favors an integrated modular structure of the brain. Simulations have shown that the behavioral complexity of globally integrated independent functional modules in a brain is more comparable to the brain models in which either specialized modules are either totally functionally independent or a brain model that has a homogenized set of interconnections and neurons. This emphasizes the fact that, along with modularity of the brain, it is equally important and necessity to have an effective integration mechanism among the specialized brain modules. Functionally similar modules are bound together through synchronization, feedback and lateral connections [30].

The observed modularity in brains is of two types. Structural modularity which is evident from sparse connections between strongly connected neuronal groups (with the trivial example of the two hemispheres of the brain) and/or functional modularity, which is indicated by the fact that neural modules have different neural response patterns, are grouped together.

The modular behavior of the brain suggests that individual brain modules are domain specific, autonomous, specific stimulus driven, unaware of the central cognitive goal to be achieved and of fixed functional neural architecture. These modules also exhibit a property of knowledge encapsulation, i.e., other modules cannot influence the internal working of an individual module. The only information about a module available to other modules is its output [31].

Along with the brain having a modular structure, it also exhibits a functional and structural hierarchy. Information in the brain is processed in a hierarchical fashion. First, the information is processed by a set of transducers which transform the information into the formats that each specialist modules can process. Specialist modules after processing the information, produce the information which is suitable for central or domain general processing. The hierarchical representation of the information is evident in the cortical visual areas where specialized modules perform individual tasks to accomplish highly complex visual tasks. For example, in the visual cortex of the macque monkey, there are over 30 specialized areas with each having some 300 interconnections [32]. Also, it is worth noting that the number of cortical areas increase as a function of level in the animal hierarchy [33]. For example, mice have on the average 3 to 5 visual areas, while hand human beings have visual areas on the order of 100. Hence, it can be deduced that an increase in brain size does not necessarily increase the sophistication or behavioral diversity, unless accompanied by a corresponding increase in specialized brain modules [34].

The functioning of the brain can be summarized as the cohesive co-existence of functional segregation and functional integration with a specialized integration among and within functionally segregated areas mediated by a specific functional connectivity.

## 2.2 The Concept of Modularity

In general, a computational system can be considered to have a modular architecture if it can be split into two or more subsystems in which each individual subsystem evaluates either distinct inputs or the same inputs without communicating with other subsystems. The overall output of the modular system depends on an integration unit which accepts outputs of the individual subsystems as its inputs and combines them in a predefined fashion to produce the overall output of the system. In a broader sense modularity implies that there is a considerable and visible functional or structural division among the different modules of a computational system. The modular system design approach has some obvious advantages, like simplicity and economy of design, computational efficiency, fault tolerance and better extendibility. The most important advantage of a computational modular system is its close biological analogy. Recent advances in the neurobiological sciences have strengthened the belief in the existence of modularity at the both functional and structural levels in the brain - the ultimate biological learning and computational system.

The concept of modularity is an extension of the principle of *divide and conquer*. This principle has no formal definition but is an intuitive way by which a complex computational task can be subdivided into simpler subtasks. The simpler subtasks are then accomplished by a number of the specialized local computational systems or models. Each local computational model performs an explicit, interpretable and relevant function according to the mechanics of the problem involved. The solution to the overall complex task is achieved by combining the individual results of specialized local computational systems in some task dependent optimal fashion. The overall task decomposition into simpler subtasks can be either a *soft-subdivision* or *hard-subdivision*. In the former case, the subtask can be simultaneously assigned to two or more local computational systems, whereas in the later case only one local computational model is responsible for each of the subdivided tasks.

As described earlier, a modular system in general is comprised of a number of specialist subsystems or modules. In general, these modules exhibit the following characteristics.

1. The modules are domain specific and have specialized computational architectures to recognize and respond to certain subsets of the overall task.

2. Each module is typically independent of other modules in its functioning and does not influence or become influenced by other modules.

3. The modules generally have a simpler architecture as compared to the system as a whole. Thus a module can respond to given input faster than a complex monolithic system.

4. The responses of the individual modules are simple and have to be combined by some integrating mechanism in order to generate the complex overall system response.

For example, the primate visual systems show a strong modularity and different modules are responsible for tasks, such as motion detection, shape and color evaluation. The central nervous system, upon receiving responses of the individual modules, develops a complete realization of the object being processed by the visual system [32].

To summarize, the main advantages of a modular computational system design approach are extensibility, engineering economy (which includes economy of implementation and maintenance), re-usability and enhanced operational performance.

## 2.3 Modular Artificial Neural Networks

The obvious advantages of modularity in learning systems, particularly as seen in the existence of the functional and architectural modularity in the brain, has made it a main stream theme in cognitive neuroscience research areas. Specifically, in the field of artificial neural network research, which derives its inspiration from the functioning and structure of the brain, modular design techniques are gaining popularity. The use of modular neural networks for the purpose of regression and classification can be considered as a competitor to conventional monolithic artificial neural networks, but with more advantages. Two of the most important advantages are a close neurobiological basis and greater flexibility in design and implementation. Another motivation for modular neural networks is to extend and exploit the capabilities and basic architectures of the more commonly used artificial neural networks that are inherently modular in nature. Monolithic artificial neural networks exhibit a special sort of modularity and can be considered as hierarchically organized systems in which synapses interconnecting the neurons can be considered to be the fundamental level. This level is followed by neurons which subsequently form the layers of neurons of a multi layered neural network. The next natural step to extend the existing level of hierarchical organization of an artificial neural network is to construct an ensemble of neural networks arranged in some modular fashion in which an artificial neural network comprised of multiple layers is considered as a fundamental component. This rationale along with the advances in neurobiological sciences have provided researchers a justification to explore the paradigm of modularity in design and training of neural network architectures.

A formal prevalent definition of a modular neural network is as follows [1].

**Definition 2.1.** *A neural network is said to be modular if the computation performed by the network can be decomposed into two or more modules (subsystems) that operate on distinct inputs without communicating with each other. The outputs of the modules are mediated by an integrating unit that is not permitted to feed information back to the modules. In particular, the integrating unit decided both (1) how the modules are combined to form the final output of the system, and (2) which modules should learn which training patterns.*

Modular artificial neural networks are especially efficient for certain classes of regression and classification problems, as compared to the conventional monolithic artificial neural networks. These classes of problems include problems that have distinctly different characteristics in different operating regimes. For example, in the case of function approximation, piecewise continuous functions cannot in general be accurately modeled by monolithic artificial neural networks. But on the other hand, modular neural networks have proven to be very effective and accurate when used for approximating these types of functions [38]. Some of the main advantages of learning modular systems are extendibility, incremental learning, continual adaptation, economy of learning and re-learning, and computational efficiency.

The modular neural networks are comprised of modules which can be categorized on the basis of both distinct structure and functionality which are integrated together via an integrating unit. With functional categorization, each module is a neural network which carries out a distinct identifiable subtask. Also, using this approach different types of learning algorithms can be combined in a seamless fashion. These algorithms can be neural network related, or otherwise. This leads to an improvement in artificial neural network learning because of the integration of the best suited learning algorithms for a given task (when different algorithms are available). On the other hand, structural modularization can be viewed as an approach that deviates from the conventional thinking about neural networks as non-parametric models, learning from a given data set. In structural modularization *a priori* knowledge about a task can be introduced into the structure of a neural network which gives it a meaningful structural representation. Generally, the functional and structural modularization approaches are used in conjunction with each other in order to achieve an optimal combination of modular network structure and learning algorithm.

[1]This definition has been adopted from [35, 36, 37, 38]

## 2.4 Motivations for Modular Artificial Neural Networks

The following subsections highlight some of the important motivations which make the modular neural network design approach more attractive than a conventional monolithic global neural network design approach.

### 2.4.1 Model Complexity Reduction

The model complexity of global monolithic neural networks drastically increases with an increase in the task size or difficulty. The rise in the number of weights is quadratic with respect to the increase in neural network models size [39]. Modular neural networks on the other hand, can circumvent the complexity issue, as the specialized modules have to learn only simpler and smaller tasks in spite of the fact that the overall task is complex and difficult [40, 41].

### 2.4.2 Robustness

The homogeneous connectivity in monolithic neural networks may result in a lack of stability of representation and is susceptible to interference. Modular design of neural network adds additional robustness and fault tolerance capabilities to the neural network model. This is evident from the design of the visual cortex system which is highly modular in design and is comprised of communicating functionally independent modules. Damage to a part of visual cortex system can result in a loss of some of the abilities of the visual system, but, as a whole the system can still function partially [42].

### 2.4.3 Scalability

Scalability is one of the most important characteristics of modular neural networks are which sets them apart form the conventional monolithic neural networks. In global or unitary neural networks there is no provision for incremental learning, i.e., if any additional incremental information is to be stored in a neural network, it has to be retrained using the data for which it was trained initially along with the new data set to be learned. On the other hand, modular neural networks present an architecture which is suitable for incremental addition of modules that can store any incremental addition to the already exiting learned knowledge of the modular neural network structure without having to retrain all of the modules.

### 2.4.4 Learning

Modular neural networks present a framework of integration capable of both supervised and unsupervised learning paradigms. Modules can be pre-trained individually for specific subtasks and then integrated via an integration unit or can be trained along with an integrating unit. In the later situation, there is no indication in the training data as to which module should perform which subtask and during training individual modules compete, or cooperate to accomplish the desired overall task. This learning scheme is a combined function of both supervised as well as unsupervised learning paradigms.

### 2.4.5 Computational Efficiency

If the processing can be divided into separate, smaller and possibly parallel subtasks, then the computational effort will in general be greatly reduced [43]. A modular neural network can learn a set of functional mappings faster than a corresponding global monolithic neural network because each individual module in a modular neural network has to learn a, possibly simpler, part of the overall functional mapping. Also, modular networks have an inherent capability of decomposing the decomposable tasks into a set of simpler tasks, thus enhancing the learn-ability and reducing the learning time.

### 2.4.6 Learning Capacity

Embedding modularity into neural network structures leads to many advantages compared to a single global neural network. For example, introduction of integrated local computational models of neural networks increases the learning capacity of a modular neural network model, and thus permits their use for more complex large-scale problems which ordinarily cannot be handled by global neural network models. Also, a complex behavior may require different types of knowledge and processing techniques to be integrated together which is not possible without any structural or functional modularity.

### 2.4.7 Economy of Learning

To enable continued survival of biological systems, new functionalities are integrated into already existing systems along with continued learning and adaptation to changing environments [44]. Using the same analogy, modularity enables learning economy in a way that if the operating conditions change, then only those parts of the modular neural network need to be modified that do not conform to the new environment, rather than the entire system. In addition, it is also possible to reuse some of the existing specialist modules for different task of the same nature instead of learning again the parts common to the two tasks.

### 2.4.8 Knowledge Integration

Modularity is a way of embedding *a priori* knowledge in a neural network architecture that is important to improve the neural network learning. The motivation for integration of *a priori* knowledge about the task at hand is that it might be the optimal way to design an appropriate neural network system for the available training data. This may include possibility to hybridize the neural network architecture. In a modular neural network architecture it is possible to use and integrate different neural functions, different neural structures or different kind of learning algorithms, depending on the task at hand.

### 2.4.9 Immunity to Crosstalk

Unitary monolithic neural networks suffer from the phenomenon of interference or catastrophic forgetting [18, 45] that does no affect modular neural networks. This interference can be divided into two categories which are briefly described below.

- **Temporal Crosstalk:** The phenomenon of temporal crosstalk is basically the loss of already learned knowledge by a neural network about a task when it is retrained to perform another task of a different type, or when two or more tasks have to be learned by a single global neural network consecutively [35].
- **Spatial Crosstalk:** This phenomenon occurs in global monolithic neural networks when a it has to learn two or more different tasks simultaneously [35].

The crosstalk phenomenon is attributed to the fact that the same set of weights of a neural network has to learn different mappings, whether simultaneously or consecutively. Both temporal and spatial crosstalk can be avoided by the use of modular structures of neural networks in which tasks can be subdivided; and, each module, with separate set of interconnecting weights, is responsible for its individual tasks.

### 2.4.10 Insight into Neural Network Models

Modular neural networks can achieve a significant performance improvement because knowledge about a task can be used to introduce a structure and a meaningful representation into their design. By virtue of the fact that different modules perform different distinct tasks within a modular neural network, and a mediating unit regulates their behavior, it is easy to obtain insight into the workings of the modular neural network just by separately analyzing the output behavior of individual modules and the mediating unit. This feature is non-existent and perhaps, is not even possible in global monolithic neural networks.

### 2.4.11 Biological Analogy

The concept of modularity can be justified on neurobiological grounds. Vertebrate nervous systems operate on the principle of modularity; and, the nervous system is comprised of different modules dedicated to different subtasks working together to accomplish complex nervous system tasks. For example, highly complex visual tasks are broken down into small subtasks, so that the visual tasks for different situations can be optimized [32]. Moreover, the same structures are replicated many times, giving the visual cortex the much desired property of robustness.

## 2.5 Literature Review

In the early stages of neural networks research, it was believed that the monolithic or unitary neural networks can solve difficult problems of varying difficulty when applied to classification and regression problems. Continued research in the field of artificial neural networks indicated that there are certain problems which cannot be effectively solved by the global monolithic neural networks. This led to the conception of modular neural networks. Modular neural networks present a new biologically inspired trend in artificial neural network design and learning. In order to improve the performance of the unitary neural networks to solve complex problems, two independent approaches were adopted, namely, ensemble-based and modular [46, 38].

The ensemble-based approach, which is sometimes referred to as the committee framework as well, deals with determination of an optimal combination of already trained artificial neural networks. Each member neural network in the ensemble is trained to learn the same task; and, the outputs of each member neural network are combined in an optimal fashion to improve the performance, compared to each member individually. In this approach, the underlying assumption is that an optimal combination of the ensemble output is better than the output of a single neural network. The aim of the ensemble-based approach is to obtain a more reliable and accurate estimate of a given functional mapping by combining individual neural networks estimates rather than just selecting one of the best performing neural networks out of the ensemble [47]. The use of the ensemble-based approach to formulate a neural network design and learning strategy is becoming more popular and has found application in the fields of classification, function approximation, econometrics and machine learning [48, 49, 50, 51].

The main theme of the ensemble-based approach is the same as the modular approach, i.e., to somehow improve the performance of monolithic neural networks. There are two main issues of concern. Firstly, the issue of creation of a set of diverse neural networks, and secondly, the determination of optimal methods to integrate the outputs of the individual neural networks in an ensemble. The first issue is handled in a variety of ways. The neural networks in an ensemble can be made different from each other by creating neural networks with randomly varying set of initial weights, varying topology, employing different learning algorithms for the individual members of the ensemble and/or different or disjoint the training data sets [46, 47]. Also, there are some methods based on “boosting” algorithms which have proven to be quite effective in training an ensemble of neural networks [52, 53].

After having obtained a trained ensemble of neural networks, the issue of finding effective ways to combine the individual neural network outputs needs to be resolved. The primary factors which need to be considered while choosing an ensemble combination method are bias and variance in the functional mapping estimates. A number of ways have been reported in the literature to handle this important issue [54, 55, 56, 57, 58, 59, 60, 61, 62]. The linear opinion polls or the linear combination of the outputs of the individual neural networks in an ensemble is one of the most widely used methods, with only one constraint, that the result of the combination of the individual neural network distributions is itself a distribution [54]. Also, simple averaging or weighted average of the outputs of the members of the ensemble to create the final output are also commonly used [57, 56]. Nonlinear combination methods include Dempster-Shafer belief based algorithms [60], rank based information [59] and a variety of voting schemes [58, 63, 64]. The probability based combination method of Bayesian with linear combination was proposed in [54]. The methods of stacked generalization and regression are some of the other ways to combine the members of an ensemble of neural networks [61, 62]. In these approaches the classifier neural networks are stacked in a hierarchical fashion and the output of one level is fed as the input to the higher level neural network which learns how to combine its inputs to generate an optimal combination of the outputs of the ensemble of the neural networks at the lower level of the hierarchy.

On the other hand, in contrast to the ensemble based approach, the underlying motivation of the modular approach is to decompose a complex task into simpler subtasks using the principle of *divide* and *conquer*. This decomposition can be explicitly based on *a priori* knowledge about the task or can be automatically determined during the learning phase of a modular neural network. The modular approach can lead to modular neural systems in which integration of specialist modules can result in solving some problems which otherwise would not have been possible using monolithic neural networks [46, 47]. The central issues of concern in this approach are the same as those of the ensemble-based approach, i.e., how the task is to be decomposed using an appropriate number of specialist modules and how to combine the individual outputs of each expert module to accomplish the desired goal at hand.

There is a wide variety of literature available on modular neural networks in the broader sense. Modular neural networks falling into broader category do not necessarily conform to the true sense of modular neural networks, i.e., having distinct modules operating specifically on different sub-regimes of the overall domain of operation. Modular neural network architectures that are modular in the true sense of modularity, i.e., follow Definition 2.1 are very few, and are described in the following paragraphs.

*Decoupled modules* architecture uses both unsupervised and supervised learning in two sequential stages [65]. In the first stage of the decoupled modules architecture, an adaptive resonant theory (ART) network, as proposed in [66], is used for decomposing the input data into its inherent clusters in an unsupervised fashion. After classification of the input data into its inherent classes, each class is assigned to an individual module for learning. These modules are then trained in parallel using a supervised learning paradigm; and, there is no communication between modules during the training. The final classification is obtained using the absolute maximum of the activation of all the individual modules. Another modular architecture which is similar to the decoupled modules architecture in structure, proposed in [67], is called *other-output* model. The only difference between this architecture and the decoupled modules architecture is that each module along with its classification output, has a binary output. The binary output indicates whether the current input sample belongs to the sub-region of the input space that this module was intended to learn and classify. A similar modular architecture proposed in the literature along the lines of the decoupled modules architecture and other-output architecture, is called the *ART-BP network* [68]. In this model, an ART network is used in an unsupervised fashion to classify the input data into its respective clusters. The same number of modules as that of the number of clusters determined by ART network are implemented. These modules are comprised of neural networks trained by a supervised learning paradigm. The trained ART network has two functions in this architecture. Firstly, to induce a sense of competition among the supervised modules; and, secondly, to direct the input samples for learning and classification purposes to the supervised modules. The outputs of the supervised neural networks is the final output of this type of architecture.

Another modular neural network architecture, called *The hierarchical network*, has strong structural similarities to the previously discussed neural network models [69, 70]. This neural network model has a two level hierarchy of neural network modules, trained in supervisory fashion. The base level module acts as a supervisory module and performs a coarse partitioning of the input space. The specialist modules at the higher level of hierarchy perform a fine-tuned learning and classification of the already partitioned input space by the supervisory module. Another extension of the already discussed neural models is the *hierarchical competitive modular neural network* [71]. Instead of having one level of unsupervised ART modules, it has two levels of ART modules with increasing vigilance factor for partitioning of the input data space. The first level of ART modules passes the coarse grain partitioned input space to the second level of ART modules for fine tuned partitioning. The second level of ART modules feeds the partitioned input space for learning and classification purposes to the neural network modules trained by a supervised learning paradigm. The supervised modules learn and classify the individually assigned regions of the input space without communicating with each other. Another extension of the hierarchical competitive modular neural network in the literature is called the *cooperative modular neural network* [72]. This model is has the same learning concept as that of a hierarchical competitive modular neural network, but has a three levels of ART neural networks with increasing vigilance factor in a hierarchical manner, instead of two levels, as is the case in hierarchical competitive modular neural network. A majority voting scheme is employed to determine the subclass of an input sample. Then supervised modules are used learn and classify the individual partitions of the input space partitioned by the 3-level ART hierarchy. The *Merge-and-glue network* is another modular neural model reported in the literature for classification and regression purposes [73, 74]. The training of this neural network model is conducted in two phases. In the first phase, a decomposition of the input data space is performed heuristically according the expert human knowledge available about the target problem. The decomposed subclasses of the input space are assigned to the individual neural networks which are trained using one of the available supervised learning algorithms. Training of the individual neural network modules is carried out till a satisfactory learning performance has been achieved and then the second phase of this neural network model begins. In the second phase, a global neural network is formed by merging the already trained individual neural network modules together, keeping their topology and set of weights intact. Also, some new hidden layer neurons are then "glued" into the global neural network. The new global neural network is trained again and the newly glued neurons try to compensate for the misclassification of any of the data samples in the input data space.

Adaptive mixture of local experts is by far the most widely used modular neural network architecture [75, 76] along with its variants [77, 78, 79]. This architecture is a widely accepted modular neural model because of its close biological analogy and intuitive structure. This architecture has origins in Gaussian mixtures and generalized local learning paradigms. Local learning algorithms attempt to locally adjust the capacity of the training system to the properties of the training set in each area of the input space [80]. There has been some amount of research carried out to prove the convergence properties of this architecture [81, 82] and improve its performance [83]. In this modular neural network model, an automatic nested partitioning of the input space is carried out; and, a hierarchy of specialist modules, also called expert neural networks, tries to specialize in learning and classifying each partition of the nested partitions in a competitive manner. The overall output of this modular neural network is a weighted sum of the outputs of the nested expert neural networks, weighted by the corresponding gating networks outputs at each level of the hierarchy. The resulting adaptive mixture of experts is a tree-like structure, referred to as a hierarchical mixture of experts [84, 35]. The primary training algorithm used for training a hierarchical mixture of experts is the Expectation Maximization (EM) algorithm [85]. The most commonly used training algorithms for hierarchical mixture of experts models are described in Appendix A. This modular neural model suffers from a major disadvantage of being fixed and rigid in a sense that its hierarchical structure must be decided *a priori* before starting the training phase. There has been surprisingly little research effort in this respect to adapt the static and balanced hierarchical structure during the training of the neural model [86]. This hierarchical modular structure has a wide variety of applications reported in research literature literature. These applications are mainly of two types, i.e., classification and regression [87, 88, 89, 90, 41].

## 2.6 Hierarchical Mixture of Experts

The Hierarchical Mixture of Experts (HME) architecture consisting of modular and hierarchically stacked neural networks was presented by Jordan and Jacobs [84]. The hierarchical mixture of experts architecture is a direct competitor to other widely used global non-modular monolithic neural network architectures for the purposes of classification and regression, such as feed-forward multi-layered perceptrons or the radial basis function networks. This architecture derives its functional basis from the popular (and similar) hierarchically structured *divide and conquer* models in the field of statistics. These models include “Classification and Regression Trees” (CART) [91], “Multivariate Adaptive Regression Splines” (MARS) [92], and “Inductive Decision Trees” (ID3) [93]. These algorithms fit surfaces to data by explicitly dividing the input space into a nested sequence of regions, and fitting simple surfaces within these regions. Convergence times of these algorithms are often orders of magnitude faster than the gradient based neural network algorithms. However, these algorithms solve function approximation or classification problems by explicitly *hard splitting* the input space into sub-regions, such that only one single “expert” is contributing to the overall output of the model. These “hard splits” of the input space make CART, MARS and ID3 algorithms to be variance increasing, especially in the case of higher dimensional input spaces where data is very sparsely distributed. In contrast, the hierarchical mixtures of experts architecture uses a *soft splitting* approach to partition the input space, instead of *hard splitting* as is the case in statistical models, allowing the input data to be present simultaneously in multiple sub regions. In this case, many experts may contribute to the overall output of the network which has a variance decreasing effect.

The hierarchical mixture of experts architecture consists of comparatively simple experts or specialists neural and gating networks, organized in a tree like structure as shown in Figure 2.1. The basic functional principle behind this structure is the well known technique called *divide and conquer*. Architectures of this type solve complex problems by dividing them into simpler problems for which solutions can be obtained easily. These partial solutions are then integrated to yield an overall solution to the whole problem. In the hierarchical mixture of experts architecture, the leaves of the tree represent expert networks, which act as simple local problem solvers. Their output is hierarchically combined by so called gating networks at the internal nodes of the tree to form the overall solution. Consider the case of functional mapping learning of the type $\vec{Y} = f(\vec{X})$ based on training data set $\mathcal{T} = (x^{(t)}, y^{(t)}), t = 0, \cdots, n$ with $\vec{X} = \{x_1, x_2, \cdots, x_n\}$ and a corresponding desired response $\vec{Y} = \{y_1, y_2, \cdots, y_n\}$. All of the networks, both expert and gating, receive the same input vector at the $t^{th}$ time instant, $x^{(t)}$, with the only difference being that the gating networks use this input to compute confidence level values for the outputs of the connected expert networks whereas the expert networks use the input to generate an estimate of the desired output value. The outputs of the gating networks are scalar values and are a partition of unity at each point in the input space, i.e., a probability set. Consider the two-layered binary

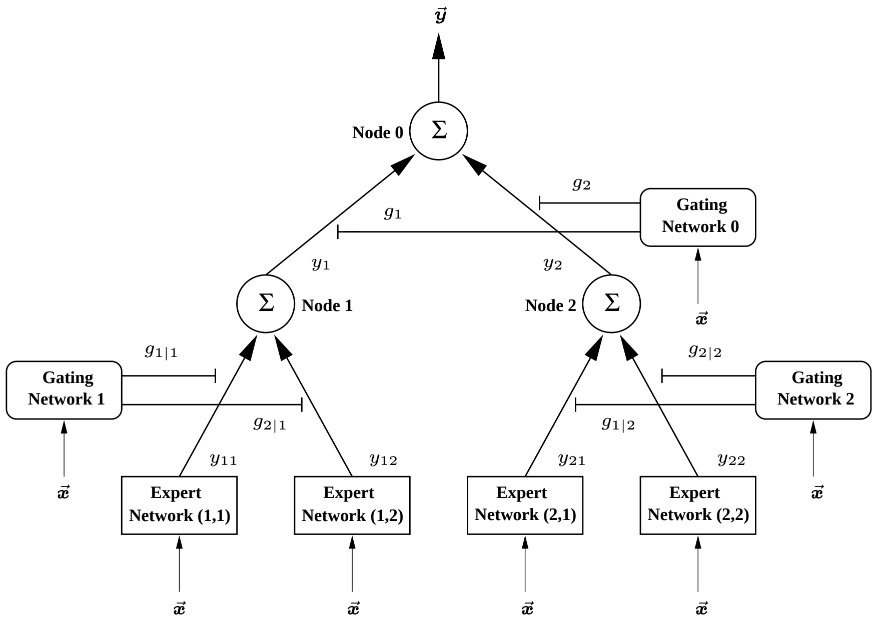


Figure 2.1: Hierarchical mixture of experts network.

branching HME as shown in Figure 2.1. Each of the expert neural networks $(i, j)$ produces an outputs $y_{ij}$ from the input vector $x^{(t)}$ according to the relationship

$$y_{ij} = f(x^{(t)}, \vec{W}_{ij})$$

where $f$ is a neural network mapping using input $x^{(t)}$ and its corresponding weight matrix $\vec{W}_{ij}$. The input vector $x^{(t)}$ is considered to have an additional constant value to allow for network bias.

The gating networks are generally linear. Since they perform multiway classification among the expert networks, the output nonlinearity is chosen to be a "softmax" which short for soft maximum. The outputs of the gating network $g_i$ at the top level are computed according to

$$g_i = \frac{e^{\zeta_i}}{\sum_k e^{\zeta_k}} \quad with \quad \zeta_i = \vec{V}_i^T x^{(t)}$$

where $\vec{V}_i$ is the weight matrix associated with gating network $g_i$. Due to the special form of the softmax nonlinearly, the $g_i$'s are positive and sum up to one for each input vector $x^{(t)}$. They can be interpreted as the local conditional probability, that an input vector $x^{(t)}$ lies in the affiliated partitioned sub-region of the associated expert network. The lower level gating networks compute their output activations similar to the top level gating network according to the following expression

$$g_{j|i} = \frac{e^{\zeta_{ij}}}{\sum_k e^{\zeta_{ik}}} \quad with \quad \zeta_{ij} = \vec{V}_{ij}^T x^{(t)}$$

The output activations of the expert networks are weighted by the gating networks output activations as they proceed up the tree to form the overall output vector. Specifically, the output of the $i^{th}$ internal node in the second layer of the tree is

$$y_i = \sum_j g_{j|i} y_{ij}$$

and the output at the top level node is

$$y^{(t)} = \sum_i g_i y_i$$

Since both the expert and the gating networks compute their activations as function of the input $\vec{X}$, the overall output of the architecture is a nonlinear function of the input.

The fundamental concept behind the probabilistic interpretation of this network is that a paralinguistic mapping of input vectors $x^{(t)}$ to output vectors $y^{(t)}$ in the data set can be subdivided into sequence of nested decisions. The architecture can be considered as generating a probabilistic tree. For a particular input vector $x^{(t)}$, values generated by the gating networks are assumed to be multinomial probabilities selecting one of the connected expert networks. A sequence of decisions starts from the top node influenced by the probability distributions of the intermediate gating networks. The process eventually ends at a specific terminal expert network.

## 2.7 Modular Neural Network Design Issues

The most important issue in designing modular neural networks is the definition of the modular neural architecture at the abstract level such that it does not lose its relevancy to the application at hand, biological/cognitive adherence and theoretical analysis.

The task of designing a modular network can essentially be broken down into two major subtasks; firstly, the determination of the optimal number and form of the operating regimes for task subdivision and secondly, the implementation of a methodology by which the individual local modules are integrated to produce an optimal neural global neural network structure. The advantages of a modular structure of a neural network cannot achieved to the fullest if either of these two steps is not implemented optimally.

Task decomposition is the most important step while implementing a modular neural network design. This is because of a major drawback of monolithic connectionist neural networks known as *catastrophic forgetting/remembering or interference* [94, 95, 96] which is sometimes referred to as crosstalk, as well [18, 35]. This is the loss of previously learned information when an attempt is made to train a previously trained neural network for a new task, or a neural network is forced to learn more than one task simultaneously.

In order to better understand this problem, consider the case of learning a relationship between two variables $\vec{X} \in \mathfrak{R}^r$ and $\vec{Y} \in \mathfrak{R}^p$. Let these variables be represented by $n$ observations, $\vec{X} = \{x_1, x_2, \cdots, x_n\}$ and a corresponding desired response $\vec{Y} = \{y_1, y_2, \cdots, y_n\}$. These observations represent numerical values of some underlying phenomenon of interest. Assume that these variables are related by an exact functional relationship which embodies all available information describing the relationship between $x^{(t)}$ and $y^{(t)}$ as follows;

$$y^{(t)} = g(x^{(t)})$$

for some mapping $g : \mathfrak{R}^r \rightarrow \mathfrak{R}^p$ where $x^{(t)}$ and $y^{(t)}$ represent numerical values of the variables $\vec{X}$ and $\vec{Y}$ at the $t^{th}$ time instant, respectively.

Let $\mu$ be the "environmental" probability law describing the manner in which numerical representations of the two variables were generated. Define a joint probability law $\nu$ that takes in account the environment $\mu$ as well as the probabilistic relationship between $\vec{Y}$ given $\vec{X}$. In neural network learning, the conditional probability law $\nu$ is of interest in an abstract sense, but the main objective is to find a neural network architecture that performs acceptably for a given performance-based objective measure of interest. Let the goodness of relationship between $x^{(t)}$ and $y^{(t)}$ be measured using performance function $\pi : \mathfrak{R}^p \times \mathfrak{R}^p \rightarrow \mathfrak{R}$. The neural network performing a functional mapping from input $\vec{X}$ to output $\vec{Y}$ can be expressed as $f : \mathfrak{R}^r \times \boldsymbol{w} \rightarrow \mathfrak{R}^p$, where $\boldsymbol{w}$ is the weight space appropriate to the neural network. Given the input $x^{(t)}$ and weights $w$, the neural network output is given by $o^{(t)} = f(x^{(t)}, w)$. Then given a target value $y^{(t)}$ and the neural network model output $o^{(t)}$, the performance function $\pi(y^{(t)}, o^{(t)}) = \pi(y^{(t)}, f(x^{(t)}, w))$ gives a numerical measure indicating how well the neural network performed on average for given different values of input $\vec{X}$. Average performance can be mathematically expressed as the expectation of the random quantity $\pi(\vec{Y}, f(\vec{X}, w))$ described as in Equations 2.1 and 2.2

$$\begin{aligned} \lambda(w) &\equiv \int \pi(y^{(t)}, f(x^{(t)}, w))\nu(dy^{(t)}, dx^{(t)}) && (2.1) \\ &\equiv E[\pi(\vec{Y}, f(\vec{X}, w))] \quad w \in \boldsymbol{w} && (2.2) \end{aligned}$$

The expected performance function of Equations 2.1 and 2.2 depends only on the weights $w$, and not on particular realizations of $\vec{Y}$ and $\vec{X}$ which can be considered averaged out. In the context of neural networks, the objective is to find the best possible set of weights that can deliver the best average performance and can be specified as the solution to the problem

$$\underset{w \in \boldsymbol{w}}{min} \lambda(w)$$

The solution to this problem is an optimal set of weights, that is not necessarily unique, denoted by $w^*$.

Now, consider a specific case of artificial neural network learning using the most frequently encountered performance measure, called the squared error, $\pi(y^{(t)}, o^{(t)}) = (y^{(t)} - o^{(t)})^2$, using backpropagation training algorithm. Then

$$\lambda(w) = E([\vec{Y} - f(\vec{X}, w)]^2) \tag{2.3}$$

Using $g(\vec{X}) = E(\vec{Y}|\vec{X})$, Equation 2.3, after some mathematical manipulations, can be reduced to the following form

$$\lambda(w) = E([\vec{Y} - g(\vec{X})]^2) + E([g(\vec{X}) - f(\vec{X}, w)]^2) \tag{2.4}$$

It follows that the optimal set of weights $w^*$ which minimizes $\lambda(w)$ of Equation 2.4, but also minimizes

$$E([g(\vec{X}) - f(\vec{X}, w)]^2) = \int [g(x^{(t)}) - f(x^{(t)}, w)]^2 \mu(dx^{(t)}) \tag{2.5}$$

In other words, $f(., w^*)$ with optimal weight vector $w^*$ is a mean-squared approximation to the conditional approximation function $g$.

The optimal weight vector $w^*$ that minimizes Equation 2.5 is highly dependent upon the environment $\mu$ which plays an important role in the determination of the optimal weights $w^*$ for a neural network. The neural network with $w^*$ weights on the average produces small approximation errors for the values of $\vec{X}$ that are likely to occur and large errors for the values which are not likely to occur. This leads to the conclusion that for an operating environment $\hat{\mu} \neq \mu$, the neural network with an optimal set of weights $w^*$ will not perform optimally. A neural network $f(\vec{X}, w^*)$ with weights $w^*$, will perform optimally by construction as long as the environment probability law $\nu$ governs the generation of $\vec{X}$ and $\vec{Y}$. On the other hand, if realizations of $\vec{X}$ and $\vec{Y}$ are generated by a different environment probability law $\hat{\nu}$, that is not the same environment probability law $\nu$ for which optimal weights $w^*$ were determined, the neural network performance will degrade drastically because it is encountering unseen observations that were not present during network training [97]. These assumptions are supported by results reported in the literature through a series of experiments conducted on various error backpropagation trained neural networks models of various types and sizes, for a variety of similar tasks and training vectors of different sizes [98, 99]. It was observed that previously learned information can be catastrophically forgotten by a trained neural network while attempting to learn a new set of patterns. The conclusions drawn from these experiments were that the underlying reason for this catastrophic forgetting was the single set of shared weights, and, that this behavior was a radical manifestation of so called ”stability-plasticity” dilemma [100].

In order to elaborate on the catastrophic forgetting behavior of monolithic neural networks, an insight into the weight space is essential. When a unitary neural network has been trained to perform optimally in recognizing an initial set of patterns, it corresponds to finding a point in weight space $w_{init} \in \boldsymbol{w}$. If the same trained neural network now learns a new set of patterns, irrespective of the size of the new set of patterns, the neural network will move to a new solution point in weight space, $w_{new} \in \boldsymbol{w}$. The set of weights $w_{new}$ enables the neural network to recognize new set of patterns. Catastrophic forgetting occurs when the new weight vector is completely inappropriate as a solution for the originally learned patterns. The same can be concluded for the case where a neural network is forced to learn two tasks simultaneously and is not able to find a suitable point in the weight space at which an optimal solution to both learning tasks can be found simultaneously. The very existence of catastrophic forgetting suggested the presence of so called “weight cliffs”, i.e., areas where moving even small distances over the weight landscape would radically disrupt prior learning [101]. The problem of catastrophic interference has long been known to exist in neural networks by the connectionist community. However, this interference is sometimes described as if it were mild and/or can readily avoided and, perhaps, for this reason, the interference phenomenon has received surprisingly little attention [98]. The problem of catastrophic forgetting is not common in higher mammalian brains because of the development of a hippocampal-neocortical separation which indicates that common weight sharing is not an ideal way to increase the learning capacity in neural network models [28].

In the light of the facts discussed in the previous paragraphs, the important issues related to the design and implementation of modular neural networks can be broadly categorized in the following subsections.

### 2.7.1 Task Decomposition

As mentioned earlier, task decomposition is the most important step in designing modular neural networks. This step is necessary and crucial for many reasons, most importantly to determine if the training set at hand is functionally or structurally divisible into a number distinct subsets. Task decomposition is also imperative because it directly dictates the structure of the modular neural network itself by deducing whether the problem at hand can be partitioned once, or must be partitioned recursively. If the task decomposition is recursive, then the modular neural network has to be recursive in nature too, otherwise a modular neural network with one level of structural hierarchy of specialist modules will suffice to solve the given problem. Task decomposition plays another important role in the subsequent training of the modular neural network by decomposing the task, whenever possible, such that there is minimum overlap of the specialized modules in the weight space.

Task decomposition can be explicitly defined using an *a priori* knowledge about the task at hand or can be learned automatically during the modular neural network learning process. Also, levels of hierarchy, if applicable, need to be determined *a priori* or automatically, in order to effectively exploit the modularity of the neural network structure. To summarize, if the right task decomposition is not employed with the global task in mind, all the advantageous characteristics of the modular networks cannot be utilized optimally.

### 2.7.2 Structural Modularization

Structural modularization deals with determination of the number and size of individual specialist modules within a modular neural network. The relative size of a specialized neural network dictates the performance of a modular neural network. If one of the expert neural networks is of relatively larger size, it may start dominating the other specialist modules, and the training of the modular neural network might face the same problems as a monolithic neural network. The number of expert modules plays an important role in efficient learning and operation of a modular neural network because too many individual modules will result in longer training times. On other hand, a modular neural network with too few specialist modules may not be able accomplish the learning task. It is also in this step that the functionality of each expert module is assigned according to the task at hand, i.e., the selecting architectures of individual specialist modules and their corresponding function in a modular neural network framework.

### 2.7.3 Training Algorithm

The implementation of a training algorithm also plays an equally important role in design and implementation of modular neural networks. The training algorithm should be comprehensive in a sense that it should setup local goals for individual specialized modules so that the overall goal of global training is accomplished satisfactorily.

The training algorithm should be the right algorithm for the right problem. Whenever it is necessary, a training algorithm should be able to decompose the learning data set automatically. It should be able to recognize, via expert knowledge or automatically, whether individual specialized modules of a modular neural network have to be trained independently of each other, in competitive fashion or in a cooperative manner in order to accomplish the assigned local tasks. Also, another important factor that a training algorithm should cater for is the decision of incremental or sequential learning based on the task decomposition information. Intra- and inter-module communication during the training phase should also be an integral part of a training algorithm in order to dynamically preserve the separation of new learning from old learning. Adaptation, structural or functional, is another desirable characteristic of a training algorithm.

### 2.7.4 Combination of Specialist Modules

Combining the outputs of the individual expert modules in a modular neural network is another crucial decision. Combination of expert modules can be competitive, cooperative or totally decoupled among the individual expert neural networks in a given modular neural network. In a decoupled approach, individual specialist modules have no information about other modules in the network and the output of the best performing special neural network is picked to be overall output of the modular neural network according to some pre-specified criterion. Competitive combination on the other hand, favors a communication and competition among the individual expert neural networks. The output of the winner expert neural network among the individual the expert modules is chosen to be the final output of the modular neural network. In cooperative combination, outputs of the multiple individual expert modules are combined in proportion to their degree of performance to form the final output of the modular neural network instead of just choosing the best performing specialist neural network. The combination scheme can also be a combination of cooperative and competitive schemes.

The steps and techniques outlined in the preceding paragraphs are by no means exhaustive, but are meant to highlight some of the points that need to be considered while designing a modular neural network. The subsequent chapters of this dissertation describe some of the modular neural architectures, demonstrating the use of the above mentioned techniques for regression and classification problems.

# Chapter 3

# Laterally Connected Neural Network Models

The previous chapters described in detail the importance of modularity in design and training of artificial neural networks. The modularity described earlier primarily dealt with the design and implementation of a number of neural networks that were grouped together in some modular fashion with non-existing modularity within an individual member neural network.

Many of the currently used feed-forward or recurrent neural network models have little, if any, structurally constrained architecture which is contrary to the notion that the neural network models draw their inspiration from the structural and functional mechanism of the brain. The brain is a highly structured entity in which a structured organizational and functional architecture is induced among the different neurons and groups of neurons by a combination of both excitatory and inhibitory connections. In light of these facts, it seems appropriate that before investigating the implementation and implications of modularity in a group of neural networks connected together to accomplish a task, an effort should be made in order to instill some sort of modularity and structure in a monolithic neural network using inhibitory and excitatory connections. This chapter describes a new neural network architecture and learning algorithm towards incorporating structure in an otherwise unstructured monolithic neural network. An Illustrative example is presented to prove the effectiveness of the proposed neural network architecture and its learning algorithm.

## 3.1 Motivation

Many of the popular neural network models, such as the well known multi layered perceptron model, are feed-forward networks without any feedback or inhibitory connections. For simplicity or other reasons, the flow of information through the neural network model is considered unidirectional such that outputs are calculated by feeding information only forward through the network model. However, there is much evidence that neural networks in biological sensory and nervous systems have lateral inhibition mechanisms. Using such connections, artificial neural network models can be designed with very interesting and improved behaviors.

Anatomical and physiological evidence suggests that lateral connections constitute an important part of local cortical circuitry [102]. The biological basis for these lateral connection is evident in the central nervous system. For example, consider the general architecture of the neocortical minicolumns modules. These modules extend through the entire human cortex. The modules are $0.2 - 0.3$ mm in width and and $2.5 - 3$ mm deep. These modules consist of pyramidal cells that are excitatory in nature and constitute about $60\%$ of a neocotrical minicolumn module. The rest of the neocortical module consists of various types of inhibitory inter-neurons such as basket cells [103].

The pyramidal cells in the upper layers of the cortex mainly are connected to the other cortical regions in the same hemisphere. Deeper layers of the cortex have pyramidal cells that are connected with the sub-cortical centers. The middle layers of the cortex have an abundance of inter-neurons with short range inhibitory connections. A similar type of functional similarity exists in the cerebellar cortex. Thus, given such a neurobiological model, the principle that governs intermodule organization seems to be excitation and the principle governing intra-modular organization seems to be inhibition. The architectural principle of intra-modular inhibition implies that the main learning process within a neocortical module is competitive in nature [104, 105, 106].

The inference drawn from the description of neocortical minicolumns leads to the conclusion that effective and neurobiologicaly plausible neural network models should have a mechanism of elaborate inhibition embedded into their architectures. The next section outlines one such neural network model which has the neocortical minicolumn architecture as the neural network design inspiration. The learning algorithm for such a neural neural network model is also described in the following sections.

## 3.2 Laterally Connected Neural Network Model

Fixed topology neural networks suffer from the *moving target* problem which leads to a *herd effect*. The moving target problem arises from the fact that each hidden layer node is trying to become a feature detector to contribute to the overall functioning of the neural network [107]. This fact becomes more complicated as all the hidden layer nodes are trying to evolve at the same time without communicating with other nodes in the hidden layer. The only information these hidden layer nodes have is their input and the error backpropagated from the output layer nodes. This information is constantly changing throughout the training process of an artificial neural network. The herd effect, which is a manifestation of the moving target problem, is the evolution of the weights of an artificial neural network model to reduce the source of largest training error in the training data. The herd effect becomes more pronounced in cases where each training pattern does not contribute the same amount of error. This effect can be eliminated if the hidden layer neurons receive different inputs and (backpropagated) error information. This can be accomplished by the introduction of lateral inhibitory connections among the hidden layer neurons of a neural network model. One such configuration is as shown in the Figure 3.1 and is referred to as a laterally connected neural network (LCNN) model in this chapter. The neural

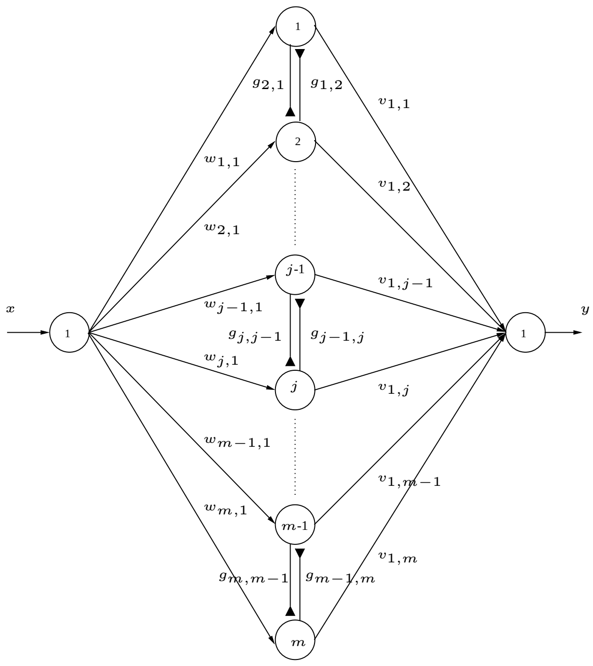


Figure 3.1: Laterally connected neural network model.

network bias connections are not shown in Figure 3.1 for the sake of simplicity, but are assumed to be present in the LCNN model architecture like any other neural network model. The idea of using lateral connections has not been explored by researchers in detail and as a consequence the literature on this topic is very limited. Some references in this respect are [108, 109, 110]. A more plausible concept on usage of lateral connections is discussed in [111]. In [111], the authors use lateral connections to establish a communication mechanism among the nodes in the hidden layer of a neural network. These lateral connections used in [111] are not, however, intended to be inhibitory in nature. The research work presented in this chapter is a major improvement of the concept presented in [111]. The idea presented and implemented in this chapter has sound neurobiological basis as mentioned in the previous section. Also, this architecture derives its inspiration from autonomous, unsupervised and self-organizing learning principles in that each hidden layer neuron is learning in a competitive fashion by inhibiting the outputs of the other hidden layer neurons which is the functionality of unsupervised and self-organizing learning [112].

The LCNN model is essentially a feed-forward neural network model with additional lateral connections among the hidden layer neurons. In the forward pass of the information through the LCNN model, every hidden layer node inhibits the output activity of the other neurons in the hidden layer. This process induces a sense of competition and communication amongst the hidden layer nodes. In the backward pass of error during the training phase, the neural network model output error is backpropagated through the lateral connections. The inhibitory effect is more pronounced among the adjacent hidden layer neurons, and a hidden layer neuron activation has a only remote effect on the neurons which are farther from it.

To summarize, the LCNN model combines seamlessly the supervised and unsupervised learning schemes in that the overall training scheme of the LCNN model is a supervised one, but the hidden layer nodes also implement a competitive learning mechanism.

## 3.3 LCNN Model Learning Algorithm

As standard popular artificial neural network models do not include lateral connections in the hidden layer, modification has to be made in the weight adaptation rules for gradient based learning algorithms such as the standard backpropagation learning algorithm. The reason for this is that the information flows through the lateral connections in the forward pass, and error propagates back via the same lateral connections during the weight update phase. Also, new learning rules have to be derived for updates of lateral inhibitory connections.

Consider the neural network model as shown in Figure 3.1. Let the net input of the $j$th hidden layer neuron be $h_j$ and its corresponding output be $z_j = \phi_j(h_j)$. Then

$$h_j = \begin{cases} \sum_{k=1}^{n} w_{j,k}x_k - g_{1,2}z_2 & j = 1 \\ \sum_{k=1}^{n} w_{j,k}x_k - g_{j,j-1}z_{j-1} - g_{j-1,j}z_j & 1 < j < m \\ \sum_{k=1}^{n} w_{j,k}x_k - g_{m,m-1}z_{m-1} & j = m \end{cases} \tag{3.1}$$

where $x_k$ is the $k$th component of the input vector $\vec{x}$. The $i$th neuron in the output layer receives a net input $s_i = \sum_{j=1}^{m} v_{i,j}z_j$ and produces an output $\hat{y}_i = \varphi_i(s_i)$. $\phi$ and $\varphi$, respectively, are the hidden and output layers neurons squashing functions.

The weights of the neural network model are adjusted by minimizing the mean sum of squared error (MSSE) objective function of Equation 3.2.

$$MSSE = J = \frac{1}{2N}\frac{1}{2}\sum_{k=1}^{q}(y_k - \hat{y}_k)^2 \tag{3.2}$$

where $q$ is the number of outputs and $y_i$ is the desired output of the neural network model. The weight update rules for the weights of the neural network models are obtained via the gradient descent algorithm to minimize the objective function $J$.

For the weights connecting the hidden to the output layer of the neural network model, the update rule is straight forward. For the weight connecting the $i$th node in the output layer to the $j$th node in the hidden layer, the weight update rules is given by the following relationship:

$$\begin{aligned} \Delta v_{ij} &= -\eta \frac{\partial J}{\partial v_{i,j}} \\ &= -\eta \frac{\partial J}{\partial \hat{y}_i}\frac{\partial y_i}{\partial s_i}\frac{\partial s_i}{\partial v_{ij}} \\ &= \eta \delta_i^o z_j \end{aligned} \tag{3.3}$$

where $\delta_i^o = \varphi'(s_i)(y_i - \hat{y}_i)$ is the $\delta$ (local gradient) for the $i$th neuron in the output layer.

For a lateral connection $g_{j,j-1}$ that connects the ($j$-1)th node to the $j$th node in the hidden layer, the weight update equations can be derived using the standard gradient descent algorithm and the chain rule of derivatives. Also, note the fact that connection $g_{j,j-1}$ was responsible for the information flow in the forward pass through ($j$-1)th through $m$th hidden layer nodes, and hence while backpropagating the output error, all of the $\delta$s corresponding to the neurons $j$ through $m$ have to be taken into consideration. The derivation of the weight update equations is as described in the following.

$$\begin{aligned} \Delta g_{j,j-1} &= -\eta_g \frac{\partial J}{\partial g_{j,j-1}} \\ &= -\eta_g \left[ \left\{ \left( \sum_{i=1}^{q} \frac{\partial J}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial s_i}\frac{\partial s_i}{\partial z_m}\frac{\partial z_m}{\partial h_m} \right) \frac{\partial h_m}{\partial h_{m-1}} \cdots \frac{\partial h_{j+1}}{\partial h_j}\frac{\partial h_j}{\partial g_{j,j-1}} \right\} \right. \\ &\quad + \left\{ \left( \sum_{i=1}^{q} \frac{\partial J}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial s_i}\frac{\partial s_i}{\partial z_{m-1}}\frac{\partial z_{m-1}}{\partial h_{m-1}} \right) \frac{\partial h_{m-1}}{\partial h_{m-2}} \cdots \frac{\partial h_{j+1}}{\partial h_j}\frac{\partial h_j}{\partial g_{j,j-1}} \right\} \\ &\quad + \\ &\quad \vdots \\ &\quad + \\ &\quad + \left\{ \left( \sum_{i=1}^{q} \frac{\partial J}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial s_i}\frac{\partial s_i}{\partial z_{j+1}}\frac{\partial z_{j+1}}{\partial h_{j+1}} \right) \frac{\partial h_{j+1}}{\partial h_j}\frac{\partial h_j}{\partial g_{j,j-1}} \right\} \end{aligned}$$

$$+ \left\{ \left( \sum_{i=1}^{q} \frac{\partial J}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial z_j} \frac{\partial z_j}{\partial h_j} \right) \frac{\partial h_j}{\partial g_{j,j-1}} \right\} \Bigg]$$

The above equations reduce to the following simplified form.

$$\begin{aligned}
\Delta g_{j,j-1} &= \eta_g \Bigg[ \Bigg\{ \left( \sum_{i=1}^{q} \delta_i^o v_{i,m} \phi'(h_m) \right) (-g_{m,m-1}\phi'(h_{m-1}))(-g_{m-1,m-2}\phi'(h_{m-2})) \cdots \\
&\qquad (-g_{j+1,j}\phi'(h_j)) z_{j-1} \Bigg\} \\
&\quad + \Bigg\{ \left( \sum_{i=1}^{q} \delta_i^o v_{i,m-1} \phi'(h_{m-1}) \right) (-g_{m-1,m-2}\phi'(h_{m-2}))(-g_{m-2,m-3}\phi'(h_{m-3})) \cdots \\
&\qquad (-g_{j+1,j}\phi'(h_j)) z_{j-1} \Bigg\} \\
&\quad + \\
&\quad \vdots \\
&\quad + \\
&\quad + \Bigg\{ \left( \sum_{i=1}^{q} \delta_i^o v_{i,j+1} \phi'(h_{j+1}) \right) (-g_{j+1,j}\phi'(h_j)) z_{j-1} \Bigg\} \\
&\quad + \Bigg\{ \left( \sum_{i=1}^{q} \delta_i^o v_{i,j} \phi'(h_j) \right) z_{j-1} \Bigg\} \Bigg]
\end{aligned}$$

Further simplification of the above equation results in

$$\begin{aligned}
\Delta g_{j,j-1} &= \eta_g \left[ \left( \sum_{i=0}^{q} \delta_i^o v_{i,j} \phi'(h_j) \right) + \left\{ \sum_{\beta=j+1}^{m} \left( \sum_{i=1}^{q} \delta_\beta^o v_{i,j} \phi'(h_j) \right) \prod_{\alpha=j+1}^{\beta} -g_{\alpha,\alpha-1}\phi'(h_{\alpha-1}) \right\} \right] z_{j-1} \\
&= \eta_g \left[ \delta_j^h + \left( \sum_{\beta=j+1}^{m} \delta_\beta^h \prod_{\alpha=j+1}^{\beta} -g_{\alpha,\alpha-1}\phi'(h_{\alpha-1}) \right\} \right] z_{j-1} \qquad (3.4)
\end{aligned}$$

where $\delta_j^h$ is the usual $\delta$ for the $j$th hidden layer neuron that is used in the standard backpropagation learning algorithm.

For a lateral connection $g_{j-1,j}$ that connects the $j$th node to the ($j$-1)th node in the hidden layer, the weight update equations can be derived in a similar fashion as that for a lateral connection

$g_{j,j-1}$ taking into account that this connection is responsible for feeding the information of $j$th through 1st hidden layer node.

$$\begin{aligned}
\Delta g_{j-1,j} &= -\eta_g \frac{\partial J}{\partial g_{j-1,j}} \\
&= -\eta_g \left[ \left\{ \left( \sum_{i=1}^{q} \frac{\partial J}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial z_1} \frac{\partial z_1}{\partial h_1} \right) \frac{\partial h_1}{\partial h_2} \cdots \frac{\partial h_{j-2}}{\partial h_{j-1}} \frac{\partial h_{j-1}}{\partial g_{j-1,j}} \right\} \right. \\
&\quad + \left\{ \left( \sum_{i=1}^{q} \frac{\partial J}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial z_2} \frac{\partial z_2}{\partial h_2} \right) \frac{\partial h_2}{\partial h_3} \cdots \frac{\partial h_{j-2}}{\partial h_{j-1}} \frac{\partial h_{j-1}}{\partial g_{j-1,j}} \right\} \\
&\quad + \\
&\quad \vdots \\
&\quad + \\
&\quad + \left\{ \left( \sum_{i=1}^{q} \frac{\partial J}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial z_{j-2}} \frac{\partial z_{j-2}}{\partial h_{j-2}} \right) \frac{\partial h_{j-2}}{\partial h_{j-1}} \frac{\partial h_{j-1}}{\partial g_{j-1,j}} \right\} \\
&\quad \left. + \left\{ \left( \sum_{i=1}^{q} \frac{\partial J}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial z_{j-1}} \frac{\partial z_{j-1}}{\partial h_{j-1}} \right) \frac{\partial h_{j-1}}{\partial g_{j-1,j}} \right\} \right]
\end{aligned}$$

The above equations reduce to the following simplified form.

$$\begin{aligned}
\Delta g_{j-1,j} &= \eta_g \left[ \left\{ \left( \sum_{i=1}^{q} \delta_i^o v_{i,1} \phi'(h_1) \right) (-g_{1,2}\phi'(h_1))(-g_{2,3}\phi'(h_2)) \cdots (-g_{j-2,j-1}\phi'(h_{j-2})) z_j \right\} \right. \\
&\quad + \left\{ \left( \sum_{i=1}^{q} \delta_i^o v_{i,2} \phi'(h_2) \right) (-g_{2,3}\phi'(h_2))(-g_{3,4}\phi'(h_3)) \cdots (-g_{j-2,j-1}\phi'(h_{j-2})) z_j \right\} \\
&\quad + \\
&\quad \vdots \\
&\quad + \\
&\quad + \left\{ \left( \sum_{i=1}^{q} \delta_i^o v_{i,j-2} \phi'(h_j) \right) (-g_{j-2,j-1}\phi'(h_{j-2})) z_j \right\} \\
&\quad \left. + \left\{ \left( \sum_{i=1}^{q} \delta_i^o v_{i,j-1} \phi'(h_{j-1}) \right) z_j \right\} \right]
\end{aligned}$$

Further simplification of the above equation results in

$$\Delta g_{j-1,j} = \eta_g \left[ \left( \sum_{i=0}^{q} \delta_i^o v_{i,j-1} \phi'(h_{j-1}) \right) + \left\{ \sum_{\beta=1}^{m} \left( \sum_{j-2}^{q} \delta_\beta^o v_{i,j-1} \phi'(h_j) \right) \prod_{\alpha=1}^{j-2} -g_{\alpha-1,\alpha} \phi'(h_\alpha) \right\} \right] z_j$$

$$= \eta_g \left[ \delta^h_{j-1} + \left\{ \sum_{\beta=1}^{j-2} \delta^h_\beta \prod_{\alpha=1}^{\beta} -g_{\alpha-1,\alpha}\phi'(h_\alpha) \right\} \right] z_j \tag{3.5}$$

For the lateral connections $g_{1,2}$ and $g_{m,m-1}$, the update equations are as given by Equation 3.6 and 3.7, respectively.

$$\Delta g_{1,2} = \eta_g \delta^h_1 z_2 \tag{3.6}$$

$$\Delta g_{m,m-1} = \eta_g \delta^h_m z_{m-1} \tag{3.7}$$

Similarly, the weights connecting the input layer neurons to the hidden layer neurons can be updated using the update rules derived. Consider a weight $w_{j,k}$ that connects the $j$th neuron in the input layer to the $k$th neuron in the hidden layer, then the update rules are derived as follows.

$$\begin{aligned}
\Delta w_{j,k} &= -\eta \frac{\partial J}{\partial w_{j,k}} \\
&= -\eta \Bigg[ \left\{ \left( \sum_{i=1}^{q} \frac{\partial J}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial z_m} \frac{\partial z_m}{\partial h_m} \right) \frac{\partial h_m}{\partial h_{m-1}} \frac{\partial h_{m-1}}{\partial h_{m-2}} \cdots \frac{\partial h_{j+1}}{\partial h_j} \frac{\partial h_j}{\partial w_{j,k}} \right\} \\
&\quad + \left\{ \left( \sum_{i=1}^{q} \frac{\partial J}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial z_{m-1}} \frac{\partial z_{m-1}}{\partial h_{m-1}} \right) \frac{\partial h_{m-1}}{\partial h_{m-2}} \frac{\partial h_{m-2}}{\partial h_{m-3}} \cdots \frac{\partial h_{j+1}}{\partial h_j} \frac{\partial h_j}{\partial w_{j,k}} \right\} \\
&\quad + \\
&\quad \vdots \\
&\quad + \\
&\quad + \left\{ \left( \sum_{i=1}^{q} \frac{\partial J}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial z_{j+1}} \frac{\partial z_{j+1}}{\partial h_{j+1}} \right) \frac{\partial h_{j+1}}{\partial h_j} \frac{\partial h_j}{\partial w_{j,k}} \right\} \\
&\quad + \left\{ \left( \sum_{i=1}^{q} \frac{\partial J}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial z_j} \frac{\partial z_j}{\partial h_j} \right) \frac{\partial h_j}{\partial w_{j,k}} \right\} \\
&\quad + \left\{ \left( \sum_{i=1}^{q} \frac{\partial J}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial z_1} \frac{\partial z_1}{\partial h_1} \right) \frac{\partial h_1}{\partial h_2} \cdots \frac{\partial h_{j-2}}{\partial h_{j-1}} \frac{\partial h_{j-1}}{\partial g_{j-1,j}} \right\} \\
&\quad + \left\{ \left( \sum_{i=1}^{q} \frac{\partial J}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial s_i} \frac{\partial s_i}{\partial z_2} \frac{\partial z_2}{\partial h_2} \right) \frac{\partial h_2}{\partial h_3} \cdots \frac{\partial h_{j-2}}{\partial h_{j-1}} \frac{\partial h_{j-1}}{\partial g_{j-1,j}} \right\} \\
&\quad + \\
&\quad \vdots \\
&\quad +
\end{aligned}$$

$$+\left\{\left(\sum_{i=1}^{q}\frac{\partial J}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial s_i}\frac{\partial s_i}{\partial z_{j-2}}\frac{\partial z_{j-2}}{\partial h_{j-2}}\right)\frac{\partial h_{j-2}}{\partial h_{j-1}}\frac{\partial h_{j-1}}{\partial g_{j-1,j}}\right\}$$
$$+\left\{\left(\sum_{i=1}^{q}\frac{\partial J}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial s_i}\frac{\partial s_i}{\partial z_{j-1}}\frac{\partial z_{j-1}}{\partial h_{j-1}}\right)\frac{\partial h_{j-1}}{\partial g_{j-1,j}}\right\}\Bigg]$$

Finally, the above equation reduces to the following update rule:

$$\begin{aligned}
\Delta w_{j,k} &= \eta\left[\left(\sum_{i=0}^{q}\delta_i^o v_{i,j}\phi'(h_j)\right)+\left\{\sum_{\beta=j+1}^{m}\left(\sum_{i=1}^{q}\delta_\beta^o v_{i,j}\phi'(h_j)\right)\prod_{\alpha=j+1}^{\beta}-g_{\alpha,\alpha-1}\phi'(h_{\alpha-1})\right\}\right.\\
&\quad\left.+\left\{\sum_{\beta=1}^{m}\left(\sum_{j-2}^{q}\delta_\beta^o v_{i,j-1}\phi'(h_j)\right)\prod_{\alpha=1}^{j-2}-g_{\alpha-1,\alpha}\phi'(h_\alpha)\right\}\right]x_k\\
&= \eta\left[\delta_j^h+\left\{\sum_{\beta=j+1}^{m}\delta_\beta^h\prod_{\alpha=j+1}^{\beta}-g_{\alpha,\alpha-1}\phi'(h_{\alpha-1})\right\}\right.\\
&\quad\left.+\left\{\sum_{\beta=1}^{j-2}\delta_\beta^h\prod_{\alpha=1}^{\beta}-g_{\alpha-1,\alpha}\phi'(h_\alpha)\right\}\right]x_k
\end{aligned} \tag{3.8}$$

The learning algorithm for the proposed LCNN model is summarized below as Algorithm 3.1 that takes the number of hidden layer nodes $m$ and the error goal $\epsilon$ as the predefined parameters along with the training data set as its input parameters.

## 3.4 An Illustrative Example

The effectiveness of the proposed algorithm for a class of function approximation problems is demonstrated by an illustrative example. Consider the case of approximating a deceptively simple function given by Equation 3.9.

$$f(x) = 2e^{(-0.2x)}|sin(x)| \tag{3.9}$$

with $x$ ranging from $0$ to $2\pi$. Figure 3.2 shows the plot of the function given in Equation 3.9. This function is difficult to be approximated by a monolithic neural network because during the training process the first "*hump*" of this function acts as the dominant error source over that of the second smaller "*hump*". All the hidden layer nodes try to compensate for the larger error source and thus do not learn the smaller hump properly. A simulation was carried out by training a monolithic neural network for $1000$ epochs with $10$ hidden layer nodes to approximate the function of Equation 3.9.

**Algorithm 3.1** Laterally connected neural network model learning algorithm

```
Requires: Training data set $T = \{x_i, y_i\}$ for $i = 1, \cdots, N$. $\epsilon, m$
  • Initialize the neural network with one hidden layer comprised of $m$ nodes with associated
    lateral connections.
  $t \Leftarrow 0$
  while $e \geq \epsilon$ do
    ◇ Present the desired inputs to the neural networks and calculate the corresponding
      outputs using Equation 3.1 and propagating information in the forward direction
      through the network.
    ◇ Calculate the value of the MSSE objective function , $J$, using Equation 3.2.
    ◇ Update the weights connecting the hidden layers to the output layer using Equation 3.3.
    ◇ Update the later connections using Equations 3.4, 3.5, 3.6, and 3.7.
    ◇ Update the weights connecting the input layer neurons to the hidden layer neurons using
      Equation 3.8.
    $e \Leftarrow J^T J$
  end while
```

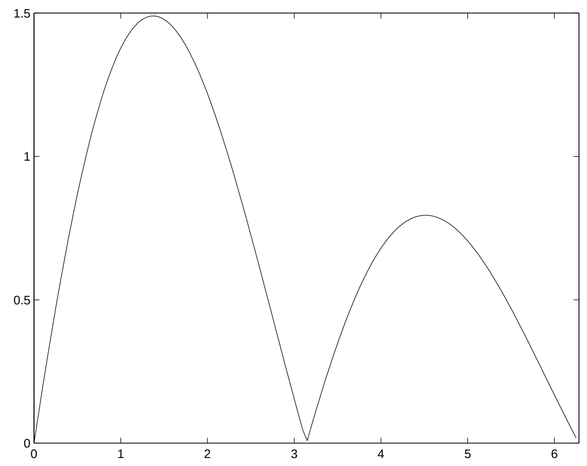


Figure 3.2: Plot of the function to be approximated.

The training algorithm used for adapting the weights of the neural network model is a recursive prediction error algorithm which belongs to a class of recursive Gauss-Newton methods. Recursive least squares (RLS) type algorithms have been used for recursive parameter estimation in the fields of system identification and adaptive filter design. This class of algorithms is characterized by two updating equations; one for parameter update and the other for covariance update. The stability and convergence of recursive least squares algorithms are heavily dependent upon the way in which the covariance matrix is updated. There have been many modifications to the basic recursive least squares algorithm to have bounded covariance matrix updates that include a constant forgetting factor, constant trace adjustment, covariance resetting and covariance modification. Considering these issues, a recursive prediction error method with exponential resetting and forgetting factor is used to train the LCNN model [113]. The EFRA algorithm is outlined below as Algorithm 3.2. In the EFRA algorithm, $\beta, \alpha, \lambda, \delta$ are the constant parameters and their choice affects the performance of the algorithm. $P_0$ and $\theta_0$ are the initial covariance matrix and initial weight vector of the neural network model respectively. $\epsilon$ is the desired error goal for the training error. It is very easy to implement this algorithm to train a neural network with some

**Algorithm 3.2** RPEM algorithm incorporating exponential resetting and forgetting

```
Requires: $T = \{\xi_i, y_i\}$ for $i = 1, \cdots, n.$ $\beta, \alpha, \lambda, \delta, P_0, \theta_0, \epsilon$
  while $e \geq \epsilon$ do
    $e \Leftarrow 0$
    for $k = 1$ to $n$ do
      $e_k \Leftarrow y_k - x_k^T \hat{\theta}_{k-1}$
      $\hat{\theta}_k \Leftarrow \hat{\theta}_{k-1} + \frac{\alpha P_{k-1} x_k}{1 + x_k^T P_{k-1} x_k} e_k$
      $P_k \Leftarrow \frac{1}{\lambda} P_{k-1} - \frac{\alpha P_{k-1} x_k x_k^T P_{k-1}}{1 + x_k^T P_{k-1} x_k} + \beta I - \delta P_{k-1}^2$
      $e \Leftarrow e + e_k^T e_k$
    end for
  end while
```

necessary computational modifications. The results have been very encouraging when using this algorithm for neural network training. The results indicate that by using the EFRA algorithm instead of the standard backpropagation algorithm, training time was reduced considerably with improved neural network generalization.

The squashing function used for the hidden layer neurons and its derivative, respectively, are given by Equations 3.10 and 3.11. This function will be referred to as the *alternate sigmoidal* squashing function with parameter $\beta$ controlling the slope of the squashing function. The output of this function lies in the range $[-1, 1]$ [114].

$$\phi(h) = \frac{\beta h}{(1 + |\beta h|)} \tag{3.10}$$

$$\phi'(h) = \frac{\beta}{(1+|h|)^2} = \beta(1 - |\phi(h)|)^2 \tag{3.11}$$

The comparative plots of commonly used tansigmoidal and alternate sigmoidal squashing functions and their derivatives shown in Figures 3.3(a) and (b), respectively. This function has

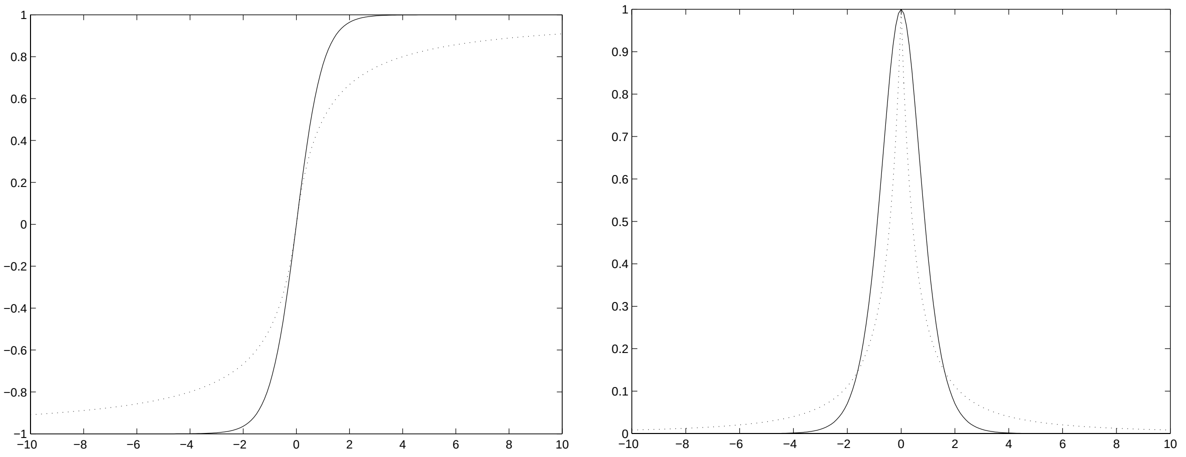
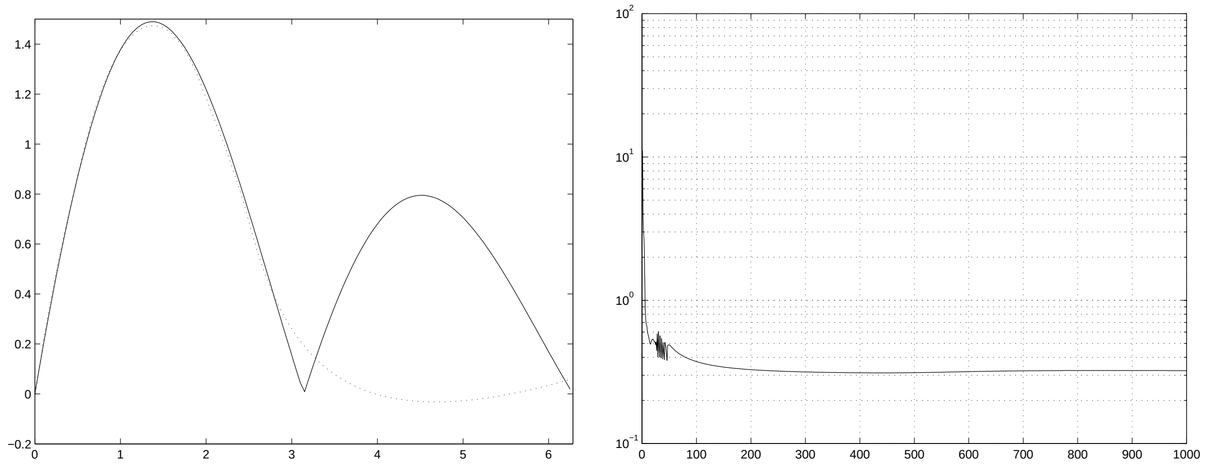


Figure 3.3: (a) Plot of the tansigmoidal and the alternate sigmoidal squashing functions. (b) Plot of the derivatives of the tansigmoidal and the alternate sigmoidal squashing functions. (solid and dotted line plots are for tansigmoidal and alternate sigmoidal functions respectively.)

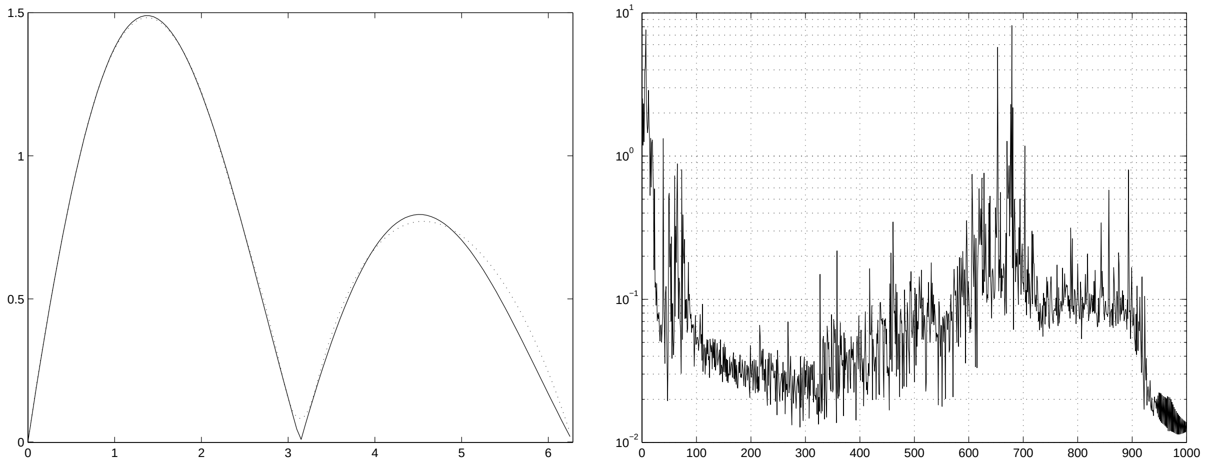
advantages over commonly used sigmoidal functions which are transcendental functions and are time consuming to calculate on some computers. The alternate sigmoidal function has slower asymptotic convergence compared to sigmoidal functions which is highly desirable in order to avoid paralysis of the neural network model during training (due the saturation of sigmoidal type squashing functions) [114]. Also, the alternate sigmoidal function takes half as many flops to evaluate than the conventional sigmoidal squashing functions.

The training was carried out for $500$ epochs. The weights were updated in an online fashion, i.e., the weight update was carried out after each training data pair presentation to the neural network. After the training phase, the function approximation test results are as shown in Figure 3.4(a). Figure 3.4(b) shows the sum squared error plot during the training phase. As is evident from the plot of Figure 3.4(a), the neural network did not learn the function properly. To further investigate the problem, the number of hidden layer nodes was increased with increments of $2$. Simulation results essentially had the same results as that of the neural network model with $10$ hidden layer nodes but with slight improvements. The last of the simulation was carried out with a neural network with $20$ hidden layer nodes. The simulation results are as shown in Figure 3.5(a). The sum squared error plot is provided in Figure 3.5(b). The results point out that increasing the



(a) (b)

Figure 3.4: (a) Function approximation by conventional monolithic neural network with 10 hidden layer neurons (solid line plot is the actual function and dotted line indicates the approximation by neural network ). (b) Plot of the sum squared training error.



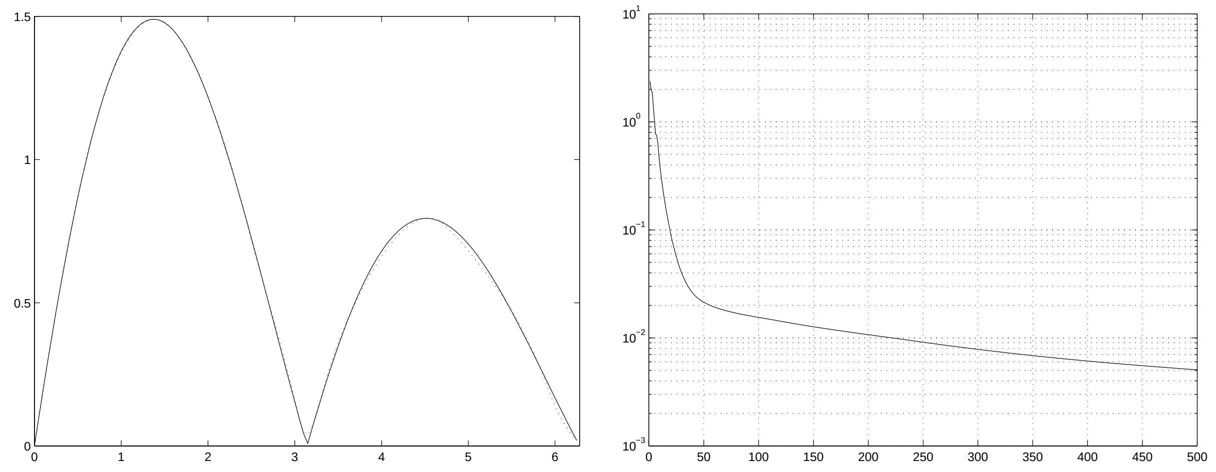
(a) (b)

Figure 3.5: (a) Function approximation by conventional monolithic neural network with 20 hidden layer neurons (solid line plot is the actual function and dotted line indicates the approximation by neural network ). (b) Plot of the sum squared training error.

number of hidden layer nodes does help in approximating the given function, but the sum of squared error plot of training error does not smoothly decrease with an increase in the training epochs. This is indicative of the fact that although the neural network was able to approximate the function adequately, it was still difficult to adjust the weights so that the error history is smooth.

To verify the effectiveness of the proposed laterally connected neural network model, the same function of Equation 3.9 was used as a test case. A series of simulations was carried out with different topologies with varying number of hidden layer neurons in the LCNN architecture. Simulations were carried out, starting with a LCNN model with $5$ hidden layer neurons and appropriate lateral connections, and sequentially increasing the hidden layer nodes by $1$ after each simulation run. A simulation run consisted of ten individual simulations. Consistent good performance was achieved with a LCNN model with $8$ hidden layer nodes. In order to report the training performance, the details of the simulation with $8$ hidden layer nodes are presented in this section.

A neural network with $8$ hidden layer nodes with lateral connections was used for this purpose. The weights connecting input layer nodes to the hidden layer nodes and connecting hidden layer nodes to the output layer nodes were chosen randomly from the range $[-1, 1]$ and lateral connection weights were chosen to be all positive from the range $[0, 1]$. In the forward pass, inhibitory information was only fed to the appropriate nodes after the outputs of the hidden layer nodes had settled down. This precaution was taken to avoid any recurrent behavior in the outputs of the hidden layer neurons. Equations 3.3, 3.4, 3.5, 3.6, 3.7, 3.8 were used to update the neural network weights during the training phase. The training was carried out for $500$ epochs. The weights were updated in an online fashion, i.e., the weight update was carried out after each training data pair presentation to the neural network. The training and test simulation results are as depicted in Figure 3.6. Figure 3.6(a) shows the test approximation results and Figure 3.6(b) shows the sum of squared training error plot. This plot indicates that the training error was below $10^{-2}$ after about $200$ iterations of training algorithm. These simulation results prove the effectiveness of the proposed laterally connected neural network model which was able to approximate a difficult function with less number of hidden layer nodes than a monolithic conventional neural network. Although, there were some extra lateral weights which needed to trained, but still the training sum squared error plot indicates that the simulation reached an acceptable error limit in less than $0.2$ of the training time of a monolithic neural network without lateral inhibitory connections with $20$ hidden layer nodes. The total number of weights to be trained in the monolithic neural network model with $20$ hidden layer nodes is $61$ whereas for LCNN model with $8$ hidden layer neurons, the total number of adjustable weights is $39$.



(a) (b)

Figure 3.6: (a) Function approximation by LCNN model (solid line plot is the actual function and dotted line indicates the approximation by LCNN model). (b) Plot of the sum squared training error for LCNN model.

## 3.5 Conclusions

In this chapter a laterally connected neural network model was introduced which has lateral inhibitory connections among the hidden layer neurons. The LCNN model is biologically inspired and the model of neocortical regions of central nervous system was used to derive its architecture. Also, a gradient descent based learning algorithm was presented for the proposed LCNN model. The proposed LCNN model is proven to have superior performance for function approximation problems compared to monolithic neural networks without lateral connections. The proposed LCNN model was able to learn a difficult function approximation problem with a smaller number of weights in a reduced learning time, and hence less computational effort.

The proposed LCNN model has consistent good performance when the network topology is chosen optimally. However, the proposed LCNN model has a drawback at this point of time. The lateral inhibitory connections, although initialized to positive values at the beginning of the training phase, can change their sign during the course of training as a consequence of the weight updates. This problem can be resolved by an introduction of a more robust learning algorithm which ensures that the lateral inhibitory weights do not change sign during the training phase. One of the possibilities could be to replace the gradient descent learning algorithm with an exponentiated gradient descent based training algorithm [115, 116, 117].

# Chapter 4

# Evidence Maximization Framework for Modular Neural Networks

As stated in the previous chapters, modularity in neural network design is an emerging and a revitalized concept. Modular decomposition of neural networks can be undertaken for the purposes of simplifying a complex learning task or improving performance. In the latter case a task could be accomplished with a monolithic neural network, but a better performance is achieved when the neural network is broken down into a number of specialist modules. In addition to performance improvement, there are other reasons for decomposing a problem. For example, it might not be possible to accomplish a task in question unless the problem is first simplified by decomposing it into simpler subtasks. The principle of divide and conquer, whereby a task is divided into a number of subproblems, generally forms the basis for the modular neural network design and can be used to extend the capabilities of a monolithic neural network. Each of the subproblems of a complex overall learning task obtained by the application of the principle of divide and conquer, could then be solved by a set of different specialist neural networks, which might employ different or similar architectures or learning algorithms, making it possible to exploit their specialist capabilities. Modular neural network architecture lends itself to a very important characteristic in that each specialist component in a modular neural system can take the form of an artificial neural network or any other non-neural network computing technique.

The objective of this chapter is to present a new hierarchically structured modular neural network model and discuss a new associated learning algorithm. The new learning scheme is called *middle-up-down* that draws its inspiration from the organizational models of the successful Japanese companies and combines the features of biologically inspired *bottom-up* and *top-down* neural network learning paradigms. This method of learning is also reminiscent of Bayesian neural network learning in which learning takes place using message passing in a tree structured architecture consisting of nodes representing different events. In the proposed learning scheme the evidence maximization framework is adopted for training of the expert and the integrating or gating neural network modules of the proposed modular model in which the performance of an individual expert neural network module is treated as an *evidence*. Better performing individual expert neural networks are rewarded according to their performance in regard to the task at hand by maximizing their individual evidences. The same concept is followed for the training of the gating neural network module as well. Considering the output of the gating neural network module as the *belief* in the performance of an expert neural network module, the outputs of the integrating unit corresponding to the better performing expert neural network modules are strengthened in order to increase belief in the better performing expert neural network modules.

The performance of the proposed modular neural network architecture and the associated learning scheme is illustrated by its application to a number of system identification and classification problems. The experimental results are encouraging and are included in the chapter for reference.

## 4.1 Introduction

Hierarchically structured modular neural networks are a class of neural networks in which a task to be performed by neural networks is decomposed into two or more overlapping or distinct decomposed subtasks. Each of the subtasks is then assigned to an individual expert neural network which tries to learn its assigned subtask without communicating with other expert neural network modules. The outputs of the expert neural network modules are mediated by an integrating or a gating unit that is not permitted to feed any information directly back to the expert neural network modules. In particular, the integrating unit decides how to combine the outputs of the individual expert neural networks to form the overall output of the system and it also decides which expert neural network module should learn which subset of the training data set patterns [46].

The possible motivations for adopting a modular approach to solve a problem using neural networks are numerous and have been outlined in the previous chapters. For example, potential advantages of adopting a modular neural network approach are that of reducing model complexity; and making the overall system easier to understand, modify, and extend which is a common engineering design principle. Training times for the neural networks can be reduced as a result of modular decomposition, and a priori knowledge task at hand can be incorporated in terms of devising an appropriate task decomposition strategy.

The design process for the modular neural networks is comprised of two important phases, namely the task decomposition and the combination of the partial representations of the overall task generated by the individual expert neural network modules. The decomposition of a problem into modular components may be accomplished automatically or explicitly [118]. Explicit decomposition of a task relies on a strong understanding of the problem at hand. If the division of a task into subtasks can be achieved based on a priori knowledge, an improved learning and subsequent neural network performance can result. Similarly, specialist modules can be developed specifically for particular subtasks. An alternative approach is one in which automatic decomposition of the task is undertaken [84]. This approach is characterized by a simultaneous application of a data partitioning technique and a learning scheme for the specialist neural network modules. Automatic task decomposition is more likely to be carried out with a view to improve performance; while explicit decomposition might be undertaken with the aim of improving performance, or that of accomplishing tasks which either could not be accomplished using a monolithic neural network, or could not be accomplished either as easily or as naturally.

Another similar approach to problem decomposition could be to use disjoint or mutually exclusive training sets, the there is no overlap between the data used to train different expert neural networks. The problem with this approach could be that the size of the training set is reduced which may result in a deteriorated generalization and subsequent individual neural network expert modules performance. Another method for selection of the data sets on which expert neural networks are trained is to use data from different input sources. This is possible under circumstances in which, for instance, more than one sensor is used. This approach is particularly applicable where the sensors are designed to generate different kinds of information. The data on which neural networks are trained can also be generated by using different input preprocessing methods. For example, different signal processing methods might be applied to the input data set, or different feature sets can be extracted from the input data set for neural network training.

The second phase while designing modular neural networks is the combination of the outputs or the partial representations of the overall complex task generated by expert neural network modules in a modular neural network model. Combination of specialist neural networks in a modular neural network architecture can be broadly categorized into two categories, namely cooperative and competitive. The main difference between cooperative and competitive combinations is that in cooperative combination it is assumed that all of the expert neural networks to be combined will make some contribution to solve the problem at hand; whereas in competitive combination, it is assumed that the most appropriate specialist neural network module will be selected depending on its performance which can either depend on its input or output. There are two main approaches for accomplishing this selection. In the first approach, the modular neural network model learns to allocate examples to the most appropriate module, like for example hierarchical mixture of experts architecture [84] and the second approach uses a predefined switching of specialist modules that is accomplished by means of a more explicit mechanism [119, 120].

## 4.2 Motivation

There are a variety of modular neural networks reported in the literature but by far the hierarchical mixture of experts (HME) model is the most widely accepted and used model to design modular neural networks. This model was initially proposed in [84], which has been described earlier in Section 2.6, and is an extension of generalized linear models approach. The HME model uses linear expert neural network modules and a linear gating network. The gating network tries to decompose the input space into mutually exclusive hyper-planes or sub-spaces and each sub-space is assigned to one or more expert neural networks. The learning algorithms for HME model in order to minimize or maximize the objective function are mainly of two types. The first approach consists of using standard gradient descent learning algorithm and has been applied with some success to train HME architecture. The second approach is an instance of the expectation maximization (EM) algorithm [85], which is often applied to unconditional mixture models and has also been formulated for and applied to conditional mixtures of experts [84]. The advantage of the EM algorithm as compared to the gradient descent approach lies in the fact that the EM algorithm nicely decouples the parameter estimation process for the different individual components of the HME model.

A potential drawback of using linear gating network in the HME model is that for complex functional mapping tasks, it is very difficult for the linear gating network to devise an appropriate partitioning of the complex input space to be assigned to the linear expert neural network modules to accomplish the complex functional mapping learning tasks.

Also, the proposed learning algorithms for the HME architecture in their original formulation, have to be changed according to a priori probability distribution assumption on the training data set and essentially resulting in different formulations for different learning situations. This also results in assuming a priori parameter values associated with the assumed probability distributions. For example, while training HME model for regression problems, Gaussian probability distribution assumption is imposed on the training data set and in case of binary classification problems, the probability distribution is assumed to be of Bernoulli type which results in two different learning algorithms formulations for the two cases.

Also, the Gaussian assumption on the training data sets for the regression problems is not appropriate in most of the cases. It has been shown that in the limit of an infinite training data set, the posterior distributions does, in fact, become Gaussian but with a finite number of patterns in a training data set, however, this assumption breaks down [121].

The original HME architecture suffers from credit assignment problems as well. The log likelihood objective function which is maximized is a function of the overall output of the HME model and the desired training output and does not take into account the performance of the individual expert neural network modules in respect to their individual outputs. Thus, even if an expert neural network module has performed well for certain assigned subset of the training data set, there is no guarantee that the parameters of that expert module will be updated accordingly.

There have been only few worth mentioning modifications to the originally proposed HME model reported in the neural network literature since its initial introduction. The use of nonlinear neural network based gating and expert neural network modules was reported in [122]. A fuzzy self-organizing feature map based gating network to replace the originally proposed linear gating network along with nonlinear neural network expert modules was proposed in [120]. The proposed modification was based on the observation that the role of the gating network in the hierarchal mixture of experts model is essentially the clustering of the input space into the soft hyper-ellipsoids [120]. The same observation was made in [123] and a modification to the gating network of the original HME architecture was proposed. The proposed modified HME model used linear expert neural network modules along with a modified gating network. The modified gating network used localized normalized Gaussian kernels as the basis functions that divided the input space into soft hyper-ellipsoids.

Assuming that the training data set is distributed according to the Gaussian probability distribution, Equation 4.1 is used to obtain the $j$th output $g_j$ for the modified gating network proposed in [123].

$$g_j(x, v) = \frac{\alpha_j P_j(x|v_j)}{\sum_i \alpha_i P_i(x|v_i)} \tag{4.1}$$

where

$$P_j(x|v_j) = (2\pi)^{-\frac{n}{2}} |\Sigma_j|^{-\frac{1}{2}} e^{-\frac{1}{2}(x-\mu_j)^T \Sigma_j^{-1}(x-\mu_j)}$$

is the $j$th probability density function and

$$\alpha_i \geq 0 \qquad \text{and} \qquad \sum_i \alpha_i = 1 \qquad \forall i$$

Also, the outputs of the modified gating network sum to 1 and are non-negative. The use of localized gating network ensures that the $j$th expert's influence is localized to a region around $\mu_j$. The gating network outputs are considered a priori probabilities for the expert neural network modules.

The hierarchical mixture of experts model proposed in [123] has some serious problems. In practice, special care has to be taken while training the hierarchical mixture of experts model with localized gating network. Considering the expectation maximization formulation for estimating the parameters of the proposed localized gating network, Equation 4.2 gives relationship for the posterior probability $h_j^k(y^t|x^t)$ at the $k$th iteration for the $j$th output of the localized gating network as a consequence of the expectation step.

$$h_j^k(y^t|x^t) = \frac{g_j^k(x^t, v_j)e^{\{\frac{1}{2}(y-\hat{y}_j)^T(y-\hat{y}_j)\}}}{\sum_i g_i^k(x^t, v_i)e^{\{\frac{1}{2}(y-\hat{y}_i)^T(y-\hat{y}_i)\}}} \tag{4.2}$$

where $x^t$, $y$ and $\hat{y}_j$ are the input vector at the $t$th time instant, desired output and output of the $j$th expert neural network module respectively.

Then, Equations 4.3 through 4.5 outline the maximization step at the $k$th iteration for the localized gating network parameters:

$$\alpha_j^{k+1} = \frac{1}{N}\sum_t h_j^k(y^t|x^t) \tag{4.3}$$

$$\mu_j^{k+1} = \frac{1}{\sum_t h_j^k(y^t|x^t)}\sum_t h_j^k(y^t|x^t)x^t \tag{4.4}$$

$$\Sigma_j^{k+1} = \frac{1}{\sum_t h_j^k(y^t|x^t)}\sum_t h_j^k(y^t|x^t)[x^t - \mu_j^t][x^t - \mu_j^t]^T \tag{4.5}$$

During the EM algorithm iterations for HME model with localized gating network, the expectation step according to Equation 4.1 for the localized gating network requires inversion of the covariance matrices $\Sigma_j$s obtained in the maximization step. Covariance matrices that are obtained during the maximization step can turn out to be close to singular for some of the inputs. There is no mention of this problem in [123] and hence no preventive remedial action is proposed. This problem can be avoided at the cost of additional computational time. For example, one of the solutions could be to consider only the diagonal elements of the covariance matrices after adding some small positive number and make the off diagonal elements zeros. This means that only variance terms of the different inputs are considered as the cross terms are being ignored. Although, this approach might not affect the learning capability of the hierarchical mixture of experts network, but might require more experts neural networks to accomplish the same learning task. Also, this problem can be avoided by imposing the lower bound on the diagonal elements of the covariance matrices which prevents them from becoming too small. The lower bound threshold for each entry can be selected in such a way that it reflects the variance of the individual inputs. When the computed element values drop below the assigned threshold value at any time, it may be replaced by the constant predefined threshold value.

Also, the expectation maximization algorithm which is used to train both of the HME architectures proposed in [84, 123], is sensitive to initial random initializations of both gating and expert neural networks. In the HME architecture of [84], there is no way of incorporating any prior knowledge about the task at hand for initialization of gating or expert networks. In the localized gating network model of [123], the means associated with each expert neural network can be initialized using a priori knowledge about the task at hand but the proposed training algorithm for the model does not take advantage of this fact.

In the light of above mentioned facts, it is desirable that prevalent hierarchical mixture of experts models be modified to overcome the aforementioned drawbacks. Hence, a modified hierarchical mixture of experts model is proposed in the Section 4.3 which addresses some of the issues discussed earlier in this section.

## 4.3 Modified Hierarchical Mixture of Experts Model

The proposed modified hierarchical mixture of experts (MHME) model with two level hierarchy is as shown in Figure 4.1. The proposed MHME model has a binary tree like structure and the expert neural network modules constitute the last level of the hierarchy of the proposed model. The proposed MHME model is more in line with the concepts well understood in the neural network community, i.e., its formulation and learning algorithm is intuitive and is an extension of commonly used monolithic neural network models. The MHME model does not impose any a priori assumptions on the probability distribution of the training data set which results in no need to alter the learning algorithm when switching from regression to classification problems and vise versa. Also, in the MHME model the credit assignment is explicit, i.e., the expert neural network module is reward for its better performance explicitly that results in an improved modular neural network model. The proposed MHME model is essentially the combination of the original HME model [84] and the alternate HME model [123] but is more robust, has better initializing methodology and extends the idea proposed in [123] by introducing a mathematically robust gating network. The MHME model also has two novel features, namely the input gates and the input switching capability for the gating network.

The input gating mechanism is required in situations where a sound a priori knowledge about the task at hand is available that can be utilized for the decomposition of the task at hand into simpler sub-tasks. After having obtained the sub-tasks as the consequence of the process of task decomposition using the a priori knowledge about the task, each expert neural network module is assigned to a specific sub-task out of the set of simplified sub-tasks and the expert neural network modules are only responsible for learning the assigned sub-tasks. The input gates facilitate the assignment of the decomposed sub-tasks only to the specific expert neural networks by opening the input gate for the appropriate expert neural network module when the assigned sub-task is presented to the overall MHME model. The input gates are a function of the desired input to the MHME model. These gates could simply be switches opening only for the specific expert neural network modules to assign the desired inputs or could be a complex functions that can be used for preprocessing the specific desired inputs for the specific expert neural network modules of the proposed MHME model.

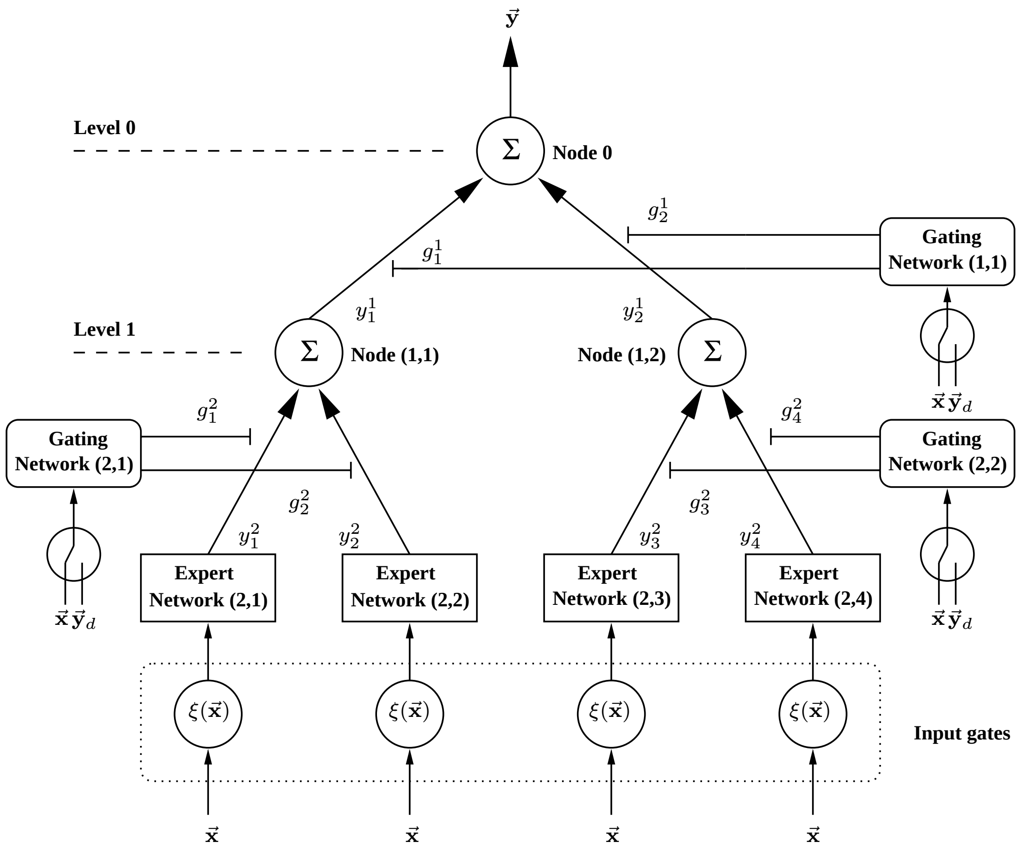


Figure 4.1: Proposed modified hierarchical mixture of experts model.

The input to the gating network in the proposed MHME model could either be the desired input or the output for the learning task at hand. This is to facilitate in accomplishing two types of partitioning of the input or the output space for the task at hand, referred to as in this dissertation as the horizontal and the vertical decomposition. This analogy is drawn from the way the x-y plots are generally generated for the functions. The input variables are generally plotted in the horizontal direction on the x-axis and the function output is plotted against input values on the y-axis or the vertical direction. So whenever the input to the gating network is the desired input, it is referred to as a horizontal decomposition and the vertical decomposition is achieved when the desired output for the task at hand is fed as an input to the gating network. The reason for introducing this functionality to switch between desired inputs or outputs is the rationale that the gating network is merely a clustering network. Clustering networks and algorithms perform well if the dimension of the space to be clustered is smaller. Also, if the dimension of the input space for a clustering neural network is smaller, it translates into lesser number of parameters to optimize during the learning phase. For the learning tasks accomplished by neural networks, generally the dimension of the output space is less than that of the input space. As the desired inputs and outputs are paired together during learning phase, if the gating network can recognize and learn the clustering in either of the two spaces, it has accomplished its task. So to facilitate in the training phase of the MHME model, in cases where the dimension of the output space is smaller than that of the desired input space, the input to the gating network can be the desired output otherwise the desired input can be fed to the gating network as its input.

The architecture of the gating network introduced in the MHME model is novel in itself due to which it is referred to as an alternate gating network. The alternate gating network is as shown in Figure 4.2. It is a radial basis function neural network that has the basis functions comprised of the composite sigmoidal functions that are commonly used in neural networks as squashing functions. The use of the composite sigmoidal functions to create a localized function has been reported in the literature where in a localized basis function is formed by subtracting two sigmoidal functions [124]. The shape and the width of these localized functions can be controlled by a suitable selection of certain strictly positive parameters. The parameters that control the width and shape of the resultant localized function have to be strictly positive for the localized function to maintain its characteristics as a localized radial basis function. The idea of localized functions presented in [124] was used in [125] to develop a robust radial basis function neural network. It is worth noting that the derivation of the parameter update equations is not relatively simple when using subtraction of two sigmoidal functions to generate a localized basis function. The robust radial basis function network has a potential. The learning algorithm used in [125] to optimize the basis function parameters like shapes, widths, and centers of the localized basis functions is an unconstrained gradient-based algorithm. This can lead to some serious potential problems, i.e., the parameter values that should remain positive during the training phase can become negative and resulting localized function is not at all what it is expected to be. In order to overcome the problems associated with localized gating network of HME model [85], an *alternate radial basis* function (ARBF) neural network is proposed in this dissertation that is used as the gating network for the proposed MHME mode. The ARBF neural network based gating network is an extension of the robust radial basis function network but eliminates the drawbacks associated with the robust radial basis function network. The proposed ARBF neural network uses a composite product of sigmoidal functions to form localized radial basis functions that are used as the hidden layer nodes. Output $z_i(x)$ for the $i$th hidden layer node of the proposed ARBF neural network for a one dimensional input case is as given by the following relationship and is plotted in Figure 4.3(b).

$$z_i(x) = \left( \frac{1}{1 + e^{-\beta^i[(x-\mu^i)+\theta^i]}} \right) \left( \frac{1}{1 + e^{\beta^i[(x-\mu^i)-\theta^i]}} \right) \tag{4.6}$$

where $\beta^i > 0$ controls the shape of the ARBF, $x$ is the real valued input, $\mu^i$ is the center of the ARBF, and $\theta^i$ is the bandwidth vector of the $i$th node. Similarly the multidimensional ARBF is obtained by simply multiplying the individual one dimensional ARBFs in each dimension.

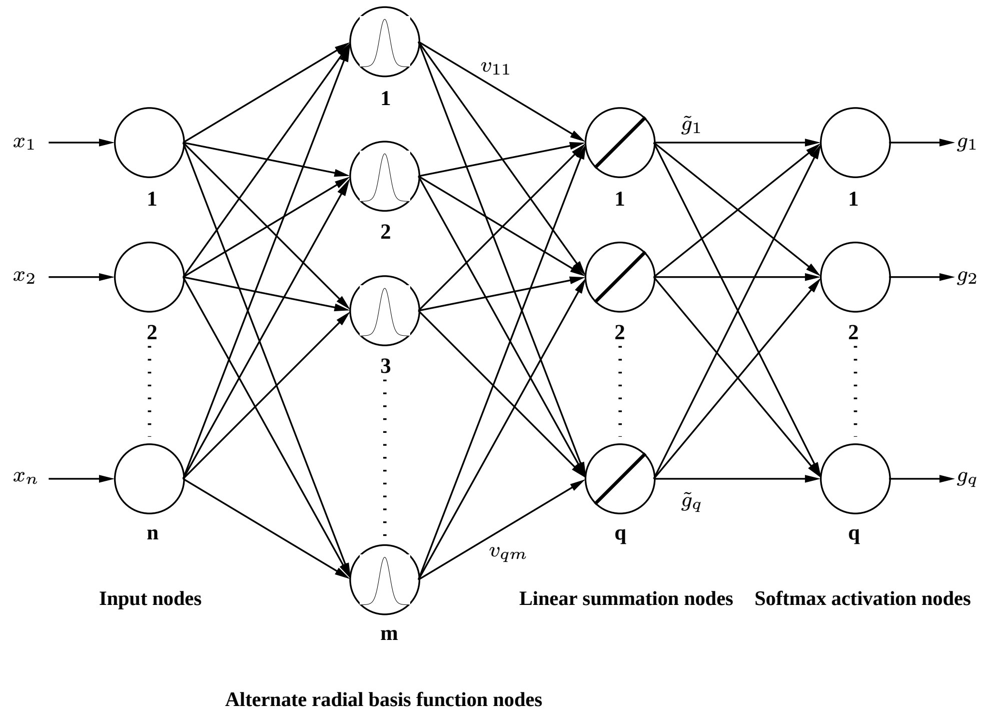


Figure 4.2: Proposed alternate gating network.

Equation 4.7 represents the mathematical relationship for the output $z_i(\vec{x})$ of the $i$th hidden layer node for a multidimensional input case in an ARBF neural network model and is as shown in Figure 4.4.

$$z_i(\vec{x}) = \prod_{k=1}^{n} \left\{ \left( \frac{1}{1 + e^{-\beta_k^i[(x_k - \mu_k^i) + \theta_k^i]}} \right) \left( \frac{1}{1 + e^{\beta_k^i[(x_k - \mu_k^i) - \theta_k^i]}} \right) \right\} \tag{4.7}$$

The function $z_i(\vec{x})$ is multidimensional ARBF activation function for the $i$th hidden layer node with real valued $\vec{x}^{\,i} = [x_1^i, x_2^i, \cdots, x_n^i]^T$, $\vec{\beta}^{\,i} = [\beta_1^i, \beta_2^i, \cdots, \beta_n^i]^T$, $\vec{\mu}^{\,i} = [\mu_1^i, \mu_2^i, \cdots, \mu_n^i]^T$, and $\vec{\theta}^{\,i} = [\theta_1^i, \theta_2^i, \cdots, \theta_n^i]^T$ being the input, the shape, the center, and the bandwidth vectors respectively.

As indicated by the plots of Figures 4.3 and 4.4, the proposed localized alternate radial basis function has all the desirable fundamental properties of a radial basis function commonly used in the neural networks field, i.e., a unique maximum at $\mu$, radial symmetry, and a local support property.

The nodes in the output layer of the proposed gating network use a special squashing function called softmax that is short for “soft maximum”. The softmax squashing function normalizes all

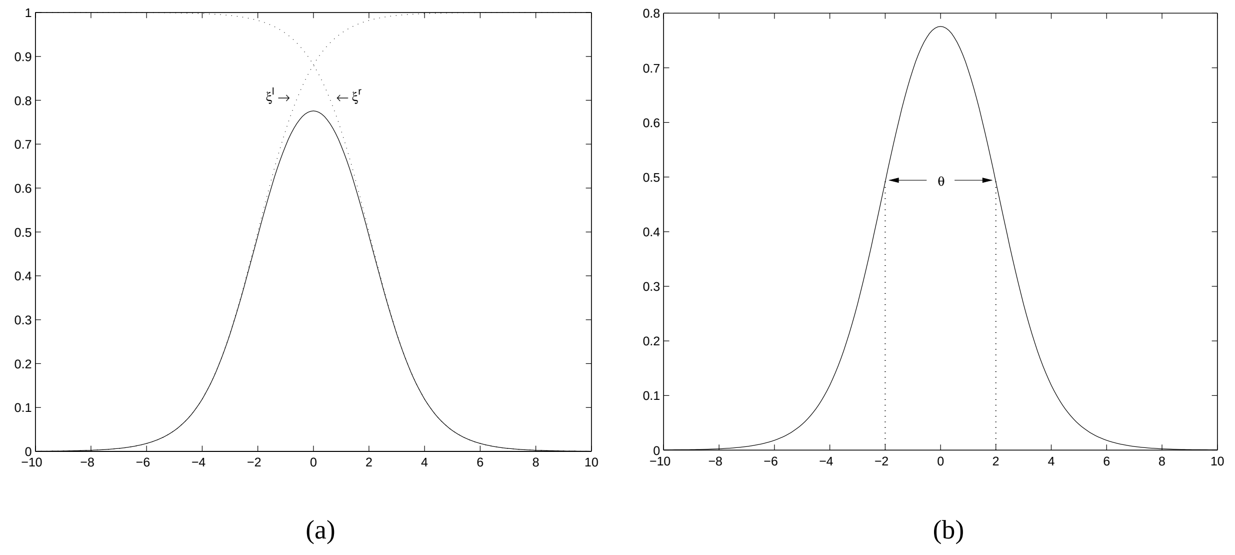


Figure 4.3: (a) Plot of two the sigmoidal functions $\xi^r$ and $\xi^l$ and their product (sigmoidal functions are represented by the dotted lines and solid line depicts their product). (b) Plot of a one dimensional ARBF with $\mu = 0$, $\theta = 2$, and $\beta = 1$.

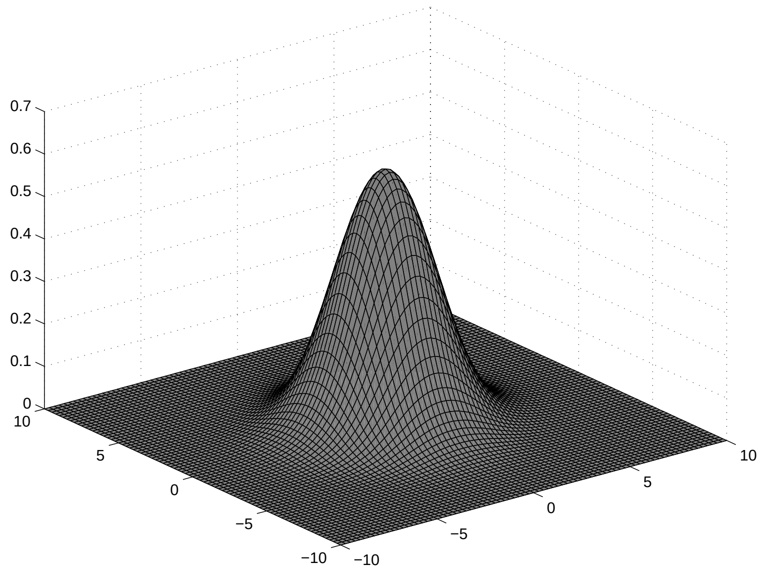


Figure 4.4: Plot of a two dimensional ARBF with $\mu = [0\ 0]^T$, $\theta = [2\ 2]^T$ and $\beta = [1\ 1]^T$.

of the outputs of the gating network in such a way that they are a partition of unity and sum to 1 with maximum portion of the unity assigned to the largest input value. Then the $j$th output of the alternate gating network $g_j$ is

$$g_j = \frac{e^{\alpha \tilde{g}_j}}{\sum_{k=1}^{q_g} e^{\alpha \tilde{g}_k}} \tag{4.8}$$

with

$$\tilde{g}_j = \sum_{i=1}^{m} v_{ji} z_i \tag{4.9}$$

where $v_{ji}$ is the weight connecting the $i$th alternate radial basis function node in the hidden layer to the $j$th summation node, $m$ is the number of the outputs of the alternate gating network, and $z_i$ is the output of the $i$th alternate radial basis function hidden layer node. The number of the outputs of an alternate gating network are the same as the number of expert neural network modules at the bottom level of the hierarchy and the higher level alternate gating networks have two outputs each. Parameter $\alpha$ controls the stiffness of the softmax squashing function. Softmax squashing function approaches the maximum operator as $\alpha \rightarrow \infty$ and approaches to a softer fuzzy maximum when $\alpha \rightarrow \epsilon$ where $\epsilon > 0$ is a small number.

The overall output $\vec{\hat{y}}$ of the proposed MHME model is governed by the the following relationship:

$$\vec{\hat{y}} = \sum_{m_1} g^1_{m_1} \sum_{m_2=2m_1-1}^{2m_1} g^2_{m_2} \cdots \sum_{m_l=2m_{l-1}-1}^{2m_{l-1}} g^l_{m_l} \vec{\hat{y}}^{\,l}_{m_l} \qquad m_1 = 1, 2 \tag{4.10}$$

where $g^l_{m_l}$ is the output of the gating network at the $l$th level corresponding $m_l$th node at the same level. $\vec{\hat{y}}^{\,l}_{m_l}$ is the output of the $m_l$th expert neural network at the $l$th level in the hierarchy. The output $\vec{\hat{y}}^{\,l}_{m_l}$ for the $m_l$th expert neural network model at the $l$th level is obtained by the procedure as described in the following.

Assuming that the $m_l$th expert neural network model is an $n$ layered feed-forward neural network. The activation input to the $i$th node in the $(k+1)$th layer is

$$a_i^{k+1} = \sum_{j=1}^{s_k} w_{ij}^{k+1} o_j^k + b_i^{k+1} \tag{4.11}$$

with $w_{ij}^{k+1}$,$o_j^k$, and $b_i^{k+1}$ being interconnecting weights from $jth$ node in $k$th layer to $i$th node in $(k+1)$th layer, output of $j$th node in $k$th layer and the bias of the $i$th node in the $(k+1)$th layer, respectively. $s_k$ denotes the number of nodes in $k$th layer of the neural network model. The output of unit $i$ in the $(k+1)$th layer is

$$o_i^{k+1} = \varphi_i^{k+1}(a_i^{k+1}) \tag{4.12}$$

where $\varphi_i^{(k+1)}$ is the squashing or transfer function for the $i$th node in the $(k+1)$th layer. For the overall network, the Equations 4.11 and 4.12 can be combined and rewritten in matrix notation as

$$\begin{aligned} \vec{o}^{\,0} &= \vec{x} && (4.13)\\ \vec{o}^{\,k+1} &= \vec{\varphi}^{\,k+1}\left(W^{k+1}\vec{o}^{\,k} + \vec{b}^{\,k+1}\right) && (4.14) \end{aligned}$$

for $k = 0, 1, \cdots, n-1$ with $\vec{x}$ is the input to the neural network model and $\vec{\hat{y}}^{\,l}_{m_l} = \vec{o}^{\,n}$.

## 4.4 MHME Model Learning Algorithm

The new learning scheme for the proposed modified hierarchical mixture of experts model is called *evidence maximization* and is inspired by the top-down and bottom-up working model of the central nervous system [100], a middle-up-down organizational model of the successful Japanese companies [126, 127] and borrows some concepts and terminology from the Bayesian learning paradigm.

Top-down and bottom-up metaphors are two common ways to explain the information processing in the hierarchical learning systems that have lower and higher levels of information processing. In such a hierarchy, lower levels of hierarchy are connected to the stimulus and are concerned mainly with recognizing the invariances in the presented stimulus and constructing multiple partial representation of the environment. Higher levels of processing are involved with integrating, comprehending and constructing the meaning of partial representations of the environment generated as a result of the processing at the lower levels. Bottom-up models view information processing proceeding linearly from the isolated units in the lower levels to the higher levels of hierarchy. Top-down models stress the influence of the higher levels on information processing carried out in the lower levels of the hierarchy.

From a neurobiological point of view, top-down and bottom-up metaphors are two different ways to explain the working of the cortex model. In the working model of the cortical, data is taken in at the bottom through primary sensory vortices and invariances are abstracted at successively higher levels. Thus, the world model at the lower levels is constructed and activated in a bottom up fashion and is constrained mostly by data from the real world. At the higher levels, the abstracted invariances are transformed into objects, concepts, motor sequences and thoughts. The nature of these invariances, i.e., the component lower order features, determines the behavior of an individual. Thus, there must be a selection process based on behavioral relevance. In the brain this selection process is instantiated by the basal ganglia. The basal ganglia selects the high level plan or prediction that satisfies or will satisfy some behavioral goal, and imposes this decision on the cortex. The decision is broadcast to the lower levels through cortico-cortical feedback connections as a prediction state. This top-down feedback determines to a high degree the activity of higher level cortex, and also the activity of lower levels, to a lesser degree. Thus, the bottom-up processes influence top-down processing, and vice versa in a loop that determines an individual's interaction with the world.

Knowledge creating model of an intelligent organization, called middle-up-down model, proposed in [126, 127], explains how have Japanese companies become the world leaders in automotive and electronics industries and what their secret is for this success. In the middle-up-down model of an organization, top management creates a vision for the organization, while middle management develops more concrete concepts that front-line employees can understand and implement. Middle managers try to solve the contradiction between what top management hopes to create and what actually exists in the real world. The authors emphasize that ”Simply put, knowledge is created by middle managers, who are often leaders of a team or task force, through a spiral conversion process involving both the top management and the front line employees. The process puts middle managers at the intersection of the vertical and horizontal flows of information within the company.” Also, the authors argue that the front line employees, the very bottom of the organization tree, are mostly busy in carrying out the day-to-day operational aspects of the various organizational processes and often may not be aware of the overall goal of the organization. The authors refer the front-line employees as the knowledge practitioners. These employees generally have no means to communicate the importance of their insights even if those are relevant to the improving the operation of the organization as a whole. In this situation, middle managers, also some times referred to as knowledge engineers, convert this unstructured insights into a structured organizationally purposeful knowledge by providing the subordinates or front-line employees with a conceptual framework that makes sense of their own experiences. Knowledge officers form the top level in the organizational hierarchy and are responsible for managing the overall organizational knowledge creation process.

The authors point out that there are two types of knowledge in an organization, namely explicit and tacit. The explicit knowledge is contained in the manuals and organizational procedures; and the tacit knowledge is acquired only by experience and is communicated only indirectly. The secret to the success of Japanese companies is their focus on the tacit knowledge and their success in having learned how to transform the tacit knowledge into the explicit knowledge. Socialization, externalization, combination and internalization lie at the heart of the new knowledge creation. Socialization is a way to share the tacit knowledge and the only effective way people can exchange the tacit knowledge is by spending time together. In the externalization process, people express and evaluate ideas of other members in a group, thus, translating tacit knowledge into an understandable and usable explicit knowledge. The combination phase facilitates the collection and organization of the newly generated explicit knowledge and disseminating it throughout the organization. This explicit knowledge is then internalized through action and training. Internalization enhances an individual's knowledge base of tacit knowledge which then can be utilized for a new round of knowledge creation cycle. Conversion from tacit to explicit knowledge is clearly essential to the knowledge creating organization as it takes knowledge out of the private space of the employees minds and into the organization.

Bayesian methods denote a decision strategy that minimizes the expectation of an objective function and a way to incorporate a priori knowledge into an inference and decision process. Bayesian methods were conceived and introduced in [128] and were described later in greater detail in [129]. The logical basis for utilizing Bayesian probabilities as measures of plausibility was subsequently established in [130]. Initially the emphasis of Bayesian probability theory was to formally utilize the prior knowledge. However, lately the area of Bayesian model comparison has been gaining popularity which does not involve an emphasis on prior information but rather emphasizes acquiring maximum information form the data. Bayesian probability theory provides a framework for inductive inference which essentially is the common sense translated and reduced to a corresponding mathematical formulation. The fundamental concept of the Bayesian analysis is that the plausibilities of alternative hypothesis are represented by probabilities, and inference is performed by evaluating those probabilities.

Bayes rule provides a methodology to update beliefs in the hypothesis of models in the light of the predictions made by different models given the same input data set. This concept has been formalized and is referred to as the Bayesian networks [131]. Such networks consist of several nodes whose parameters characterize a behavior and these parameters can be viewed as a transformation of a continuous flow of information into a behavior. This transformation produces decisions or actions based on the stimulus from the environment. The parameter update mechanism is an instant of the belief revision theory. In a Bayesian network, evidence based on the sensory or posteriori data flows from low-level behaviors to the high-level behaviors, and expectation, or a priori data, flows in the other direction. The belief in the veracity of a particular parameter value for each behavior is updated during every iteration. When each behavior is optimized with respect to a global criterion, the desired global performance is achieved.

The middle-up-down model is well suited for deriving a training methodology for the modified hierarchical mixture of experts model as a one to one correspondence can be drawn between the different hierarchical levels of the MHME model and the middle-up-down organizational model and is shown in Figure 4.5. The arrows with the solid lines in Figure 4.5 indicate the flow of information during the normal operation of the MHME model whereas the flow of information during the MHME model training phase is shown using arrows with the dashed lines. At the lowest level of the hierarchy of the MHME model are the expert neural network models which correspond to the front-line employees of the middle-up-down model. Like front-line employees, these expert neural network models at the bottom of the MHME hierarchy act as the knowledge specialists to gather information about the environment to create tacit knowledge about the task at hand. The partial representations generated by expert neural network modules for the task at hand are managed by the gating networks at the successive higher levels to generate the overall representation of the task at hand by the MHME model. So it can said that the overall MHME architecture is managed by the individual gating networks at different levels of the hierarchy. The the gating networks can be considered as the knowledge managers in the MHME model. Also, like in top-down learning model, the gating networks impose a structure on the MHME architecture which determines its behavior. The performance of the individual expert neural network models is evaluated by comparing the their outputs with the desired output of the task at hand and expert neural network models are ranked according to their individual performances. The performance of the individual expert neural networks is considered an evidence about the stimulus presented as the desired input. The a priori belief about the performance of each expert neural network models are updated based on their performances or the evidences. The principle feature of this evidence framework is an elaborate strategy for the selection of better performing expert neural network module. The parameters of the corresponding gating networks are updated according to the evaluated performances or evidences corresponding to the each of the expert neural network modules. The parameters of the each individual expert neural network are update in a way to minimize the localized individual generalization errors; in other words the individual evidences are maximized. In other words, the best performing expert neural network is rewarded by an increased belief in its evidence that is a feature of the Bayesian networks and the bottom-up learning. In bottom-up learning, as stated earlier, low level abstractions change the decision making abilities of the top levels in the hierarchy. This also corresponds to the middle management level in the middle-up-down organizational model and utilizes the features of socialization amongst the expert neural network modules to gather more know how about their hidden tacit knowledge. In this level of hierarchy in the MHME model, the tacit knowledge about the performances of the expert neural network modules is converted into an explicit knowledge, i.e., the error measures, which can then be consequently used by the MHME model to update its parameters. The Bayesian evidence framework and middle-up-down organizational model provide a unified methodology for a learning algorithm for the MHME architecture. The behavioral properties of the MHME model depend upon learned top-down expectations, matching bottom-up data with these expectations and a mismatch-driven search for new representations. The main advantage of using EM algorithm for the HME model is that it nicely decouples the parameters of the HME model for optimization purposes during the learning phase. The evidence maximization framework is also capable of doing so with the assumption that the parameter updates of the gating and expert neural networks are independent of each other. The following paragraphs describe the mathematical formulation of the learning algorithm for the MHME model.

Given a training data set $\{x^i, y^i\}_{i=1}^{N}$ comprised of $N$ desired input and output patterns with $x^i$ and $y^i$ being the desired input and output at the $i$th time instant respectively. Then the global

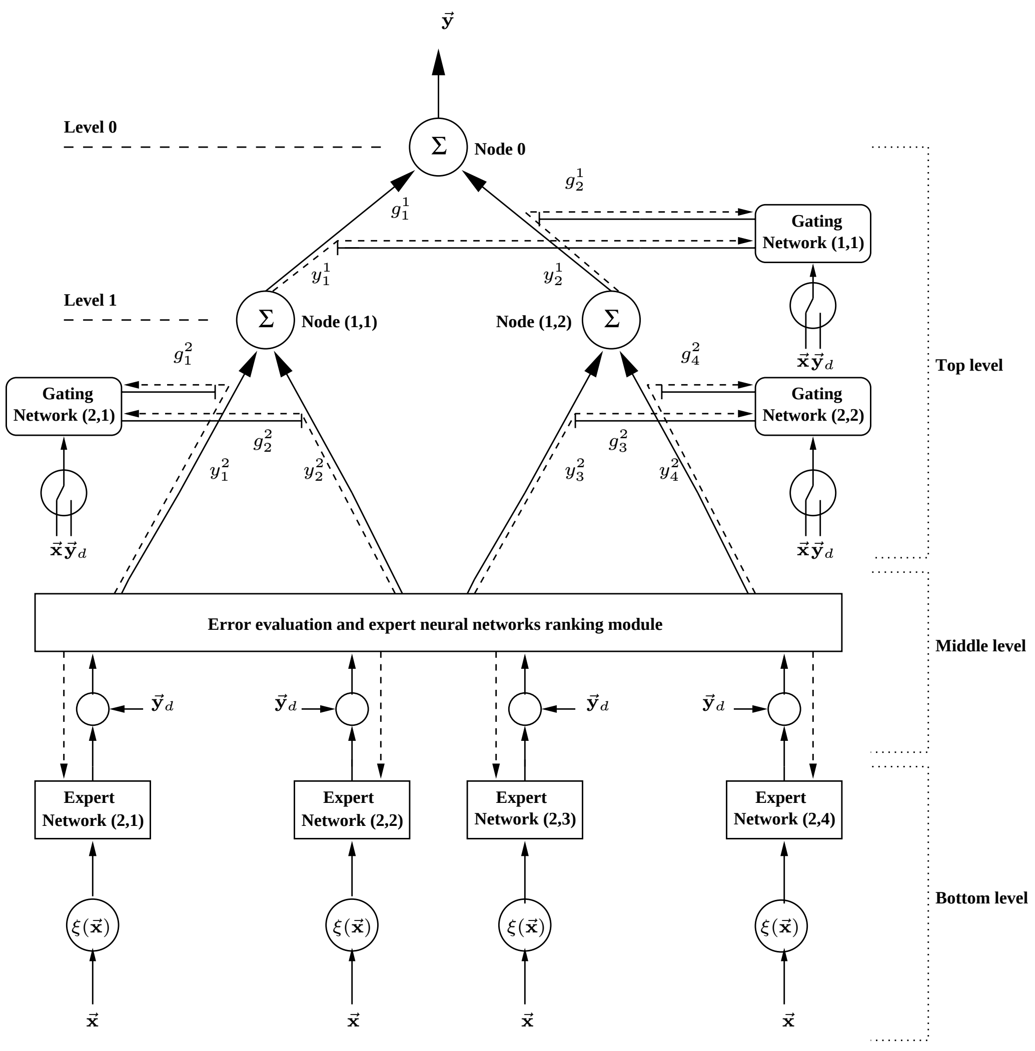


Figure 4.5: The learning scheme for the proposed modified hierarchical mixture of experts model.

objective function for the MHME model can be defined as

$$\mathcal{E} = \frac{1}{2N} \sum_{t=1}^{N} (\vec{y}^{\,(t)} - \vec{\hat{y}}^{\,(t)})^T (\vec{y}^{\,(t)} - \vec{\hat{y}}^{\,(t)}) \tag{4.15}$$

where $\vec{\hat{y}}^{\,(t)}$ is the overall MHME model output at the $t$th time instant. In the proposed MHME model, the overall objective function $\mathcal{E}$ is not minimized directly, instead local objective functions corresponding to each of the expert neural network models are minimized. The associated implicit local objective function to be minimized for the $m_l$th expert neural network module is

$$\tilde{\mathcal{E}}^{\,l}_{m_l} = \frac{1}{2N}\sum_{t=1}^{N}(\vec{y}^{\,(t)} - \vec{\hat{y}}^{\,(t)}_{m_l})^T(\vec{y}^{\,(t)} - \vec{\hat{y}}^{\,(t)}_{m_l}) \tag{4.16}$$

where $\vec{\hat{y}}_{m_l}^{\,(t)}$ is the output of the $m_l$th expert neural network module in the $l$th level of hierarchy at the $t$th time instant. Intuitively, minimizing the local objective functions of the type, as given by Equation 4.16, is equivalent to the minimization of the global objective function of Equation 4.15. Noting the fact that the overall probability mass of the MHME model sums to 1, that is,

$$\sum_{m_1} g^1_{m_1} \sum_{m_2=2m_1-1}^{2m_1} g^2_{m_2} \cdots \sum_{m_l=2m_{l-1}-1}^{2m_{l-1}} g^l_{m_l} = 1 \qquad m_1 = 1, 2 \tag{4.17}$$

then, the Equation 4.15 for the scaler desired output case can be rewritten as

$$\begin{aligned}\mathcal{E} \;=\; & \frac{1}{2N}\sum_{t=1}^{N}\left(\sum_{m_1} g^1_{m_1} \sum_{m_2=2m_1-1}^{2m_1} g^2_{m_2} \cdots \sum_{m_l=2m_{l-1}-1}^{2m_{l-1}} g^l_{m_l}\vec{y}^{\,(t)}\right. \\ & \left. \qquad - \sum_{m_1} g^1_{m_1} \sum_{m_2=2m_1-1}^{2m_1} g^2_{m_2} \cdots \sum_{m_l=2m_{l-1}-1}^{2m_{l-1}} g^l_{m_l}\vec{\hat{y}}^{\,l}_{m_l}\right)^2\end{aligned}$$

Simplifying the above equation results in

$$\begin{aligned}\mathcal{E} &= \frac{1}{2N}\sum_{t=1}^{N}\left\{\sum_{m_1} g^1_{m_1} \sum_{m_2=2m_1-1}^{2m_1} g^2_{m_2} \cdots \sum_{m_l=2m_{l-1}-1}^{2m_{l-1}} g^l_{m_l}\left(\vec{y}^{\,(t)} - \vec{\hat{y}}^{\,l}_{m_l}\right)\right\} \\ &= \frac{1}{2N}\sum_{t=1}^{N}\left\{\sum_{m_1} g^1_{m_1} \sum_{m_2=2m_1-1}^{2m_1} g^2_{m_2} \cdots \sum_{m_l=2m_{l-1}-1}^{2m_{l-1}} g^l_{m_l}\tilde{\mathcal{E}}_{m_l}\right\}\end{aligned} \tag{4.18}$$

As $g^l_i \geq 0 \;\; \forall i, l$ and the optimum minimum for each of the local implicit objective functions $\tilde{\mathcal{E}}_{m_l}$ is a arbitrary small value $\epsilon_m > 0$ for $m = 1, \cdots, 2^l$. Then if the minimization of the localized objective functions for each of the expert neural network modules at the $l$th level is successful; it amounts to the minimization of the global objective function $\mathcal{E}$ and results in improvement in the overall MHME model learning performance.

Looking at the minimization of the global objective function $\mathcal{E}$ from an alternate point of view, consider the task of learning functional mapping which maps inputs $\vec{x} \in \Re^n$ to outputs $\vec{y} \in \Re^m$

using the MHME model. The similarity measure $s^l_{m_l}$ of $m_l$th expert neural network at the $l$th level for an input $\vec{x}$ can then be defined as

$$s^l_m = (\vec{\hat{y}}^{\,l}_{m_l} - \hat{y})^T (\vec{\hat{y}}^{\,l}_{m_l} - \hat{y}) \tag{4.19}$$

where $\vec{\hat{y}}^{\,l}_{m_l}$ and $\vec{\hat{y}}$ are the outputs of the $m_l$th expert neural network at the $l$th level and overall output of the MHME model respectively. Similarity $s^l_m$ of the individual expert neural network models quantifies the disagreement among the individual expert neural networks in regard to the overall output of the MHME neural network model. The overall similarity of the MHME model then can be expressed as

$$\mathcal{S} = \sum_{m_1} g^1_{m_1} \sum_{m_2=2m_1-1}^{2m_1} g^2_{m_2} \cdots \sum_{m_l=2m_{l-1}-1}^{2m_{l-1}} s^l_m \qquad m_1 = 1, 2 \tag{4.20}$$

which can be considered as the variance of the outputs of the individual expert neural networks. The objective functions for a single input data pattern for the overall MHME neural network model and an individual $m_l$th expert neural network at the $l$th level are by can be defined as

$$\mathcal{E} \;=\; (\vec{y} - \vec{\hat{y}})^2 \tag{4.21}$$

$$\mathcal{E}^l_{m_l} \;=\; (\vec{\hat{y}}^{\,l}_{m_l} - \vec{\hat{y}})^2 \tag{4.22}$$

respectively. Adding and subtracting $\vec{y}$ in Equation 4.20 and subsequent mathematical manipulation and using the fact of the Equation 4.17, yields

$$\mathcal{E} = \bar{\mathcal{E}} - \mathcal{S} \tag{4.23}$$

with

$$\bar{\mathcal{E}} = \sum_{m_1} g^1_{m_1} \sum_{m_2=2m_1-1}^{2m_1} g^2_{m_2} \cdots \sum_{m_l=2m_{l-1}-1}^{2m_{l-1}} \tilde{\mathcal{E}}^l_{m_l}$$

The first term in Equation 4.23 is the sum of the weighted generalization errors of the individual expert neural networks while the second term is the overall similarity of the MHME model.

The relationship of Equation 4.23 nicely decomposes the overall generalization error of the MHME model for analysis purposes into two terms. First term that only depends on the generalization errors of the individual expert neural networks and the second term that takes into account the correlations between the individual expert neural network models. The relationship of Equation 4.23 in a way expresses the tradeoff between bias and variance of the MHME model. If the MHME model is strongly biased, i.e., each expert neural network has specialized in a specific region of the input space, then the overall generalization error will only be the weighted sum of the the generalization errors of individual expert neural network modules. Thus, if the training process is successful in reducing the individual generalization errors, then the overall generalization error for the MHME model will also reduce. Also, the Equation 4.23 leads to an observation that the generalization error of the overall MHME architecture is always smaller than the weighted sum of individual generalization errors of the expert neural network models, i.e., $\mathcal{E} \leq \bar{\mathcal{E}}$.

The relationship of the Equation 4.23 also indicates that in order for the MHME model to be more effective while learning a task, it should be employed in a competitive fashion, i.e., individual expert neural networks should be more diverse and compete among each other to learn the task at hand in order to reduce the overall similarity measure of the model that effectively reduces the similarity among the expert neural network models and encourages diversity.

The implicit performance index for the $m_l$th expert neural network model as described before in Equation 4.16 and is as given below

$$\tilde{\mathcal{E}}^{\,l}_{m_l} = \frac{1}{2N}\sum_{t=1}^{N}(\vec{y}^{\,(t)} - \hat{\vec{y}}^{\,(t)}_{m_l})^T(\vec{y}^{\,(t)} - \hat{\vec{y}}^{\,(t)}_{m_l})$$

The above relationship gives the evidence about the performance of the $m_l$th expert neural network and is a form of the tacit knowledge. This tacit knowledge about the performance of the individual performances of the individual neural networks need need to be converted into the explicit knowledge or the posterior beliefs. As the posterior beliefs lie in the range $[0, 1]$ and sum to 1, the desired conversion can be accomplished by using the softmin operator on the set of posterior evidences.

Softmin operator represents a smooth version of minimum operator. As a result of the softmin operation, the input with the smallest value has the output as the largest portion of the unity and largest input value has the smallest portion of the unity; and all of the outputs sum to 1. Mathematically, softmin operation for conversion of tacit knowledge into an explicit knowledge can be represented as follows

$$h^l_{m_l} = \frac{e^{-\alpha\tilde{\mathcal{E}}^l_{m_l}}}{\sum_k e^{-\alpha\tilde{\mathcal{E}}^l_k}} \tag{4.24}$$

where $k$ is the total number of expert neural network modules in the lowest $l$th level in the hierarchy of the MHME model and the parameter $\alpha$ controls the stiffness of the softmin operator. If $\alpha \to \infty$, the softmin operator approaches the minimum operation and as $\alpha \to \epsilon$ for a small positive $\epsilon > 0$, softmin operator becomes even softer fuzzy minimum operator.

The Equation 4.24 is the equivalent of the middle management in the middle-up-down organizational model. The explicit knowledge or he posterior beliefs are created at this stage of the learning process are propagated in the upward and downward directions in the hierarchy. In the downward direction, this information is used to update expert neural network parameters and in the other direction, the posterior beliefs are a source to update the appropriate gating network parameters at the higher levels of the hierarchy.

Parameters of the individual expert neural network modules are updates by choosing the explicit performance index for the $m_l$th expert neural network module as

$$\begin{aligned}\mathcal{E}^l_{m_l} &= \frac{1}{2N} h_{m_l}{}^{l\,(t)} (\vec{y}^{\,(t)} - \vec{\hat{y}}^{\,(t)}_{m_l})^T (\vec{y}^{\,(t)} - \vec{\hat{y}}^{\,(t)}_{m_l}) \\ &= \frac{1}{2N} \sum_{t=1}^{N} h^l_{m_l} \vec{e}_e^{\,T} \vec{e}_e \end{aligned} \tag{4.25}$$

where $h^{l\,(t)}_{m_l}$ is posterior belief in the $m_l$th expert and $e_e$ is the deviation of the output of the $m_l$th expert neural network from the desired output values.

The update rules for the weights in the output and input layer of the network are derived using the standard backpropagation algorithm by minimizing an approximation of Equation 4.25 by an instantaneous performance measure by the following equation.

$$\mathcal{E}^l_{m_l} = \frac{1}{2} h^l_{m_l} \vec{e}_e^{\,T} \vec{e}_e \tag{4.26}$$

where the total sum over all the training patterns is replaced by the sum squared of instantaneous errors. Then the update rule for weight $w^k_{ij}$ which connects the $j$th node in the $(k-1)$th layer to the $i$th node in the $k$th layer is

$$\Delta w^k_{ij} = -\mu_n \frac{\partial \mathcal{E}^l_{m_l}}{\partial w^k_{ij}} \tag{4.27}$$

and the update rule for the bias term for the $i$th node in the $k$th layer is

$$\Delta b^k_i = -\mu_n \frac{\partial \mathcal{E}^l_{m_l}}{\partial b^k_i} \tag{4.28}$$

with $\mu_n$ being the learning rate. Defining

$$\delta^k_i \equiv \frac{\partial \mathcal{E}^l_{m_l}}{\partial a^k_i} \tag{4.29}$$

as the sensitivity of the performance index to changes in the network activation input of the node $i$ in layer $k$. Using Equations 4.11, 4.26 and 4.29, it can be shown that

$$\frac{\partial \mathcal{E}^l_{m_l}}{\partial w^k_{ij}} = \frac{\partial \mathcal{E}^l_{m_l}}{\partial a^k_i} \frac{\partial a^k_i}{\partial w^k_{ij}} = \delta^k_i o^{k-1}_j \tag{4.30}$$

and

$$\frac{\partial \mathcal{E}^l_{m_l}}{\partial b^k_i} = \frac{\partial \mathcal{E}^l_{m_l}}{\partial a^k_i}\frac{\partial a^k_i}{\partial b^k_i} = \delta^k_i \tag{4.31}$$

Also, sensitivities satisfy the following recurrence relation

$$\vec{\delta}^{\,k} = \dot{F}^k(\vec{a}^k){W^{k+1}}^T\vec{\delta}^{\,k+1} \tag{4.32}$$

where

$$\dot{F}^k(\vec{a}^k) = \begin{bmatrix} \dot{\varphi}^k(a^k_1) & 0 & \cdots & 0 \\ 0 & \dot{\varphi}^k(a^k_2) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \dot{\varphi}^k(a^k_{s_k}) \end{bmatrix} \tag{4.33}$$

and

$$\dot{\varphi}^k(z) = \frac{d\varphi^k(a)}{da} \tag{4.34}$$

This recurrence relationship is initialized at the output layer by the following relationship

$$\vec{\delta}^{\,n} = -h^l_{m_l}\dot{F}^n(\vec{a}^{\,n})(\vec{y} - \vec{\hat{y}}^{\,l}_{m_l}) \tag{4.35}$$

To overall summary of backpropagation algorithm is as follows; inputs are propagated in the forward direction, sensitivities which depend upon desired and network outputs are propagated back and lastly weights and biases are updated depending backpropagated sensitivities.

Defining the performance index for the $j$th output of gating network in the $i$th level of the hierarchy as

$$\mathcal{G}^i_j = \frac{1}{2N}\sum_{t=1}^{N}(h^{i,\,(t)}_j - g^{i,\,(t)}_j)^T(h^{i,\,(t)}_j - g^{i,\,(t)}_j) \tag{4.36}$$

with

$$h^i_j = \sum_{m_1} g_m {m_1}^{i+1} \sum_{m2=2m_1-1}^{2m_1} g^{i+2}_{m_2} \cdots \sum_{m_l=2m_{l-1}-1}^{2m_{l-1}} g^l_{m_l} \qquad m_1 = 1, 2$$

The gating network parameters are updated to minimize the objective function of the Equation 4.36. The weights connecting the ARBF basis functions to the output summation nodes are updated according to following relationship:

$$\Delta v_{mi} = -\eta_v \frac{\partial \mathcal{G}^i_j}{\partial v_{mi}} \tag{4.37}$$

where $\eta_v$ is the learning rate, defining the step width for each of the iteration of the learning algorithm on the error surface. The calculation of $\Delta v_{mi}$ can be calculated as presented in the following:

$$\begin{aligned}
\Delta v_{mi} &= \eta_v \frac{\partial \mathcal{G}_j^i}{\partial v_{mi}} \\
&= \eta_v \frac{\partial \mathcal{G}_j^i}{\partial \tilde{g}_m^i} \frac{\partial \tilde{g}_m^i}{\partial v_{mi}} \\
&= \eta_v \left( \sum_{c=1}^{q_g} \frac{\partial \mathcal{G}_j^i}{\partial g_c^i} \frac{\partial g_c^i}{\partial \tilde{g}_m^i} \right) \frac{\partial \tilde{g}_m^i}{\partial v_{mi}} \\
&= \eta_v \left\{ \sum_{c=1}^{q_g} (h_c^i - g_c^i) \frac{\partial g_c^i}{\partial \tilde{g}_m^i} \right\} z_i \qquad (4.38) \\
&= \eta_v \left\{ \sum_{c=1}^{q_g} e_c \left( \delta_{mc} g_m^i - g_m^i g_c^i \right) \right\} z_i \\
&= \eta_v \delta_m^o z_i \qquad (4.39)
\end{aligned}$$

with

$$\frac{\partial g_c}{\partial \tilde{g}_m^i} = \delta_{mc} g_m^i - g_m^i g_c^i$$

and

$$\delta_m^o = \sum_{c=1}^{q_g} e_c \left( \delta_{mc} g_m^i - g_m^i g_c^i \right)$$

where $\delta_{mc}$ is the Kronekar delta defined as

$$\delta_{mc} = \begin{cases} 1 & m = c \\ 0 & m \neq c \end{cases}$$

and the $\delta_m^o$ is the local gradient at $m$th output unit situated before output the nodes.

Before deriving the update rules for the parameters of the ARBF basis functions, for simplifying the notation, let

$$\begin{aligned}
\zeta^{i,l} &= [(x - \mu^i) + \theta^i] \\
\zeta_k^{i,r} &= [(x - \mu^i) + \theta^i] \\
\xi^{i,l} &= \frac{1}{1 + e^{-\beta^i[(x-\mu^i)+\theta^i]}} = \frac{1}{1 + e^{-\beta^i \zeta^{i,l}}} \\
\xi^{i,r} &= \frac{1}{1 + e^{\beta^i[(x-\mu^i)-\theta^i]}} = \frac{1}{1 + e^{\beta^i \zeta^{i,r}}}
\end{aligned}$$

Then, $z_i(x)$, for one dimensional input case, the ARBF can be written as

$$\begin{aligned}
z_i(x) &= \left(\frac{1}{1+e^{-\beta^i[(x-\mu^i)+\theta^i]}}\right)\left(\frac{1}{1+e^{\beta^i[(x-\mu^i)-\theta^i]}}\right)\\
z_i(x) &= \left(\frac{1}{1+e^{-\beta^i\zeta^{i,l}}}\right)\left(\frac{1}{1+e^{\beta^i\zeta^{i,r}}}\right)\\
z_i(x) &= \xi^{i,l}\xi^{i,r}
\end{aligned}$$

Similarly, a multidimensional ARBF $z_i(\vec{x})$ becomes

$$\begin{aligned}
z_i(\vec{x}) &= \prod_{k=1}^{n}\left\{\left(\frac{1}{1+e^{-\beta_k^i[(x_k-\mu_k^i)+\theta_k^i]}}\right)\left(\frac{1}{1+e^{\beta_k^i[(x_k-\mu_k^i)-\theta_k^i]}}\right)\right\}\\
z_i(x) &= \prod_{k=1}^{n}\left\{\left(\frac{1}{1+e^{-\beta_k^i\zeta_k^{i,l}}}\right)\left(\frac{1}{1+e^{\beta_k^i\zeta_k^{i,r}}}\right)\right\}\\
z_i(\vec{x}) &= \prod_{k=1}^{n}\xi_k^{i,l}\xi_k^{i,r}
\end{aligned}$$

where $n$ is the dimension of the input space to the ARBF neural network. The parameters $\beta_j^i$ and $\theta_j^i$, which control the shape and bandwidth of the ARBF and for an ARBF to maintain its characteristics of a radial basis function, these parameters should not become negative during the training phase of the MHME architecture. Since $\beta_j^i$ and $\theta_j^i$ should remain positive throughout the learning phase, exponentiated gradient-based update rules are used for these two parameters.

Exponentiated gradient descent method was initially proposed as an alternative to gradient descent algorithms for linear predictors [115]. Exponentiated gradient replaces the usual additive updates of the parameters by multiplicative updates. Exponentiated gradient based algorithms have been used for parameters optimization in regression and reinforcement learning problems and results have indicate that exponentiated gradient based parameter optimization sometimes outperforms gradient based parameter optimization along with some distinctive similarities in their parameter updates [116, 117]. The exponentiated gradient method takes the same update steps as that of gradient descent methods but in the space of the logarithm of the parameters that yields multiplicative instead of additive updates in parameter space. These updates ensure that the parameters being optimized always remain positive during the update process when initialized with positive values. The updates for $\beta_j^i$ and $\theta_j^i$ as indicated in Equations 4.44 and 4.45 give the ARBF learning algorithm much desired mathematical robustness during the learning phase.

The update rule for the $\theta_j^i$ can be derived using the chain rule of derivatives coupled with standard error back propagating formulation and is presented in the following.

$$\begin{aligned}
\Delta\theta_j^i &= \eta_\theta \frac{\partial e_g}{\partial \theta_j^i} \\
&= \eta_\theta \left( \sum_{m=1}^{q_g} \frac{\partial e_g}{\partial \tilde{g}_m} \frac{\partial \tilde{g}_m}{\partial z_i} \right) \frac{\partial z_i}{\partial \theta_j^i} \\
&= \eta_\theta \left\{ \sum_{m=1}^{q_g} \left( \sum_{q=1}^{q_g} \frac{\partial e_g}{\partial g_q} \frac{\partial g_q}{\partial \tilde{g}_m} \right) \frac{\partial \tilde{g}_m}{\partial z_i} \right\} \left\{ \left( \prod_{\substack{k=1 \\ k\neq j}}^{n} \xi_k^{i,l} \xi_k^{i,r} \right) \frac{\partial}{\partial \theta_j^i} \left( \xi_j^{i,l} \xi_j^{i,l} \right) \right\} \\
&= \eta_\theta \left[ \sum_{m=1}^{q_g} \left\{ \sum_{q=1}^{q_g} (h_q - g_q) \frac{\partial g_q}{\partial \tilde{g}_m} \right\} v_{mi} \right] \beta_j^i \left[ \left\{ \prod_{\substack{k=1 \\ k\neq j}}^{n} \xi_k^{i,l} \xi_k^{i,r} \right\} \left\{ \xi_j^{i,l} \xi_j^{i,r} (1 - \xi_k^{i,r}) + \xi_j^{i,r} \xi_j^{i,l} (1 - \xi_j^{i,l}) \right\} \right] \\
&= \eta_g \beta_j^i \left( \sum_{m=1}^{q_g} \delta_m^o v_{mi} \right) \left[ \left\{ \prod_{\substack{k=1 \\ k\neq j}}^{n} \xi_k^{i,l} \xi_k^{i,r} \right\} \xi_j^{i,l} \xi_j^{i,r} \left\{ (1 - \xi_j^{i,r}) + (1 - \xi_j^{i,l}) \right\} \right] \\
&= \eta_g \beta_j^i \left( \sum_{m=1}^{q_g} \delta_m^o v_{mi} \right) \left[ \left\{ \prod_{k=1}^{n} \xi_k^{i,l} \xi_k^{i,r} \right\} \xi_j^{i,l} \xi_j^{i,r} \left\{ \frac{(1 - \xi_j^{i,r}) + (1 - \xi_j^{i,l})}{\xi_j^{i,l} \xi_j^{i,r}} \right\} \right] \\
&= \eta_g \beta_j^i \left( \sum_{m=1}^{q_g} \delta_m^o v_{mi} \right) \left( \prod_{k=1}^{n} \xi_k^{i,l} \xi_k^{i,r} \right) \left\{ (1 - \xi_j^{i,r}) + (1 - \xi_j^{i,l}) \right\} \qquad (4.40)
\end{aligned}$$

Following the same procedure as outlined above, the expressions for $\Delta\mu_j^i$ and $\Delta\beta_j^i$ can be derived and are given below as Equations 4.41 and 4.42 respectively.

$$\Delta\mu_j^i = \eta_\mu \beta_j^i \left( \sum_{m=1}^{q_g} \delta_m^o v_{mi} \right) \left( \prod_{k=1}^{n} \xi_k^{i,l} \xi_k^{i,r} \right) \left\{ (1 - \xi_j^{i,r}) - (1 - \xi_j^{i,l}) \right\} \qquad (4.41)$$

$$\Delta\beta_j^i = \eta_\beta \left( \sum_{m=1}^{q_g} \delta_m^o v_{mi} \right) \left( \prod_{k=1}^{n} \xi_k^{i,l} \xi_k^{i,r} \right) \left\{ \zeta_j^{i,l} (1 - \xi_j^{i,l}) - \zeta_j^{i,r} (1 - \xi_j^{i,r}) \right\} \qquad (4.42)$$

To summarize, the update rules for the alternate gating neural network parameters $v_{ji}$, $\theta_j^i$, $\beta_j^i$ and

$\mu_j^i$ are given by the following Equations 4.43, 4.44, 4.45 and 4.46 respectively.

$$\begin{aligned}
v_{ji}(n+1) &= v_{ji}(n) + \Delta v_{ji} && (4.43)\\
\ln\theta_j^i(n+1) &= \ln\theta_j^i(n) + \Delta\theta_j^i \\
\theta_j^i(n+1) &= \theta_j^i(n)e^{\Delta\theta_j^i} && (4.44)\\
\ln\beta_j^i(n+1) &= \ln\beta_j^i(n) + \Delta\beta_j^i \\
\beta_j^i(n+1) &= \beta_j^i(n)e^{\Delta\beta_j^i} && (4.45)\\
\mu_j^i(n+1) &= \mu_j^i(n) + \Delta\mu_j^i && (4.46)
\end{aligned}$$

The evidence maximization algorithm for the proposed MHME model is described is concise form as in Algorithm 4.1. Along with a training data set, it requires certain parameters to be specified a priori which are; the number of expert neural networks $n$, the error goal to be achieved $\epsilon$, and the maximum number of the iteration of the algorithm to be carried out $m$.

## 4.5 Illustrative Examples

In this section, the new features of the proposed MHME model and effectiveness of the new associated learning scheme are demonstrated to prove their usefulness. Also, the performance of the MHME model is compared with the conventionally used modular neural network HME architecture [84] for reference.

### 4.5.1 Example 1

The first example deals with modeling of a highly nonlinear plant model [132]. The plant model is of the form

$$y_p(k+1) = f\left(y_p(k), y_p(k-1), y_p(k-2), u(k), u(k-1)\right) \tag{4.47}$$

where the unknown function is of the type

$$f(x_1, x_2, x_3, x_4, x_5) = \frac{x_1x_2x_3x_5(x_3-1)+x_4}{1+x_3^2+x_2^2} \tag{4.48}$$

with

$$u(k) = \begin{cases} sin(\frac{2\pi k}{250}) & k \leq 500 \\ 0.8sin(\frac{2\pi k}{250}) + 0.2sin(\frac{2\pi k}{25}) & k > 500 \end{cases}$$

**Algorithm 4.1** Evidence maximization learning algorithm for the proposed MHME model

```
Requires: Training data set $T = \{x_i, y_i\}$ for $i = 1, \cdots, N$. $n, \epsilon, m$
  • Initialize $n$ expert neural networks with weights randomly chosen from the range $[-1, 1]$.
  • Initialize an alternate gating network with $n$ output units.
      ◇ Initialize the alternate gating network parameters using ad hoc partition
        algorithm of Section 4.5.1 for regression problems.
      ◇ Initialize the alternate gating network parameters using K-means clustering
        algorithm for classification problems.
  $t \Leftarrow 0$
  while $e \geq \epsilon$ and $t \leq m$ do
    ◇ Present desired inputs to the expert neural networks and calculate their respective outputs
      using the procedure of Section 4.3.
        ○ Implement the input gating scheme, if needed.
    ◇ Present inputs to the alternate gating network using appropriate input switching.
        ○ Implement the vertical decomposition if the dimension of the output space
          is considerably less than the input space dimensions.
        ○ Implement the horizontal decomposition if the task decomposition of the
          desired input space desired.
    ◇ Calculate the outputs of the alternate gating network using Equations 4.7 or 4.6, 4.9
      and 4.8 respectively.
    ◇ Calculate the overall output of the MHME model neural network using Equation 4.10.
    ◇ Calculate the overall error, $e$, of the MHME model using Equation 4.15.
    ◇ Calculate the evidences of individual expert neural networks using Equations 4.16.
    ◇ Convert the tacit knowledge or evidences to the explicit knowledge or usable error
      measures using Equation 4.24.
    ◇ Maximize the evidences of each individual expert neural networks by minimizing the error
      measures of Equations 4.26, 4.27 and 4.28 respectively.
    ◇ Maximize the beliefs in better performing expert neural networks by updating the
      parameters of the alternate gating networks using Equations 4.39, 4.43, 4.44, 4.45 and 4.46
      respectively.
    $t \Leftarrow t + 1$
  end while
```

A modular one level neural network based on the proposed MHME model was implemented with two expert neural networks. Each of the expert neural network model had a single hidden layer with $5$ hidden layer neurons. The output layer of the neural network had pure linear neurons and the nonlinear squashing function for the hidden layer neurons was a modified sigmoidal function which is more efficient to compute than the conventional transdental sigmoidal functions like hyperbolic tangent that is commonly used in the neural network literature. The modified sigmoidal function is used in all of the illustrative examples of this section is as defined as follows

$$\varphi(x) = 1 - \frac{2}{e^{2\beta x} + 1} \tag{4.49}$$

The gating network was an implementation of the ARBF neural network model proposed in Section 4.3. The gating or ARBF neural network had two hidden layer neurons and with the same number of outputs as that of the expert neural networks, i.e., $2$. The number of outputs of the gating network has to be the same as that of expert neural networks it is trying to mediate as its outputs are the corresponding gating probabilities or mixing coefficients, one for the each expert neural network modules. The centers and widths of the ARBF based gating network were automatically decided by partitioning of the input space in equal partitions by using an ad hoc partitioning algorithm. The ad hoc partitioning algorithm determines the maximum and minimum of the the input data set values and divides the obtained range into as many sub-partitions as that of the number of hidden nodes in the gating network. The centers for the gating network hidden layer nodes are obtained by determining the center or middle point of each of the partitions. Spreads for each of the hidden nodes of the gating network are obtained by calculating the spreads of the each obtained sub-partitions. This method works best while assigning a priori parameters to the gating network when MHME model is utilized to approximate a time series type of functions. For classification problems, more advanced clustering algorithms, like K-means or fuzzy K-means [133], can be utilized to better initialize the gating network. The expert neural network models were initialized by choosing their weight values randomly from the range $[-1, 1]$.

The HME model used for this example also had two expert neural networks and a gating network. Both expert and gating neural networks had a single hidden layer with $5$ nodes each. The modified sigmoidal function of Equation 4.49 was used as the nonlinear squashing function for the nodes in the hidden layers of the expert neural networks. The weights of the expert and gating neural networks were initialized randomly and were chosen from the range $[-1, 1]$. Two configurations of the gating network were tried, one with nonlinear squashing function for the hidden layer nodes as proposed in [122] and the other with purely linear squashing functions for the hidden layer nodes as used in the originally proposed HME model [84]. The choice of any one of the two configurations of the gating network did not had any noticeable effect on the performance of the HME model learning.

As the HME model uses expectation maximization algorithm which is a batch mode or off line training method, the training of the MHME model was also carried out in the batch mode, i.e., the parameters of the gating and expert neural networks were updated after the presentation of the entire training data set to the MHME model. The Levenberg-Marquardt (LM) learning algorithm was used to train the MHME model [134]. The parameters of gating and expert neural networks were updated after each iteration of the LM algorithm. The minimum error goal for the training phase was set to $1e^{-6}$ for both MHME and HME models and the maximum limit on the iterations of the training algorithm was set to $200$.

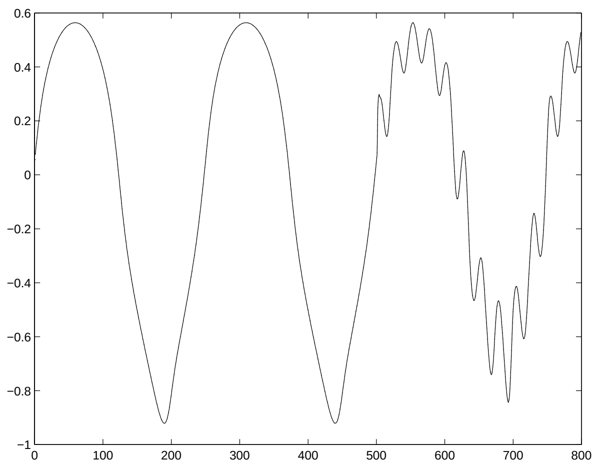
The performance results of the proposed MHME model on the test data set are shown in Figures 4.6(a), 4.7(a) and 4.7(b) respectively. The approximation and generalization capabilities of the proposed MHME model are evident from the plots of Figure 4.6(a). The MHME model approximation of the test data is indistinguishable from the desired test data set output. The results of the HME model are also of the same quality as ploted in Figure 4.6(b), its approximation is also indistinguishable from the actual test data desired output. The HME model training error and outputs of the corresponding gating network are ploted in Figures 4.8(a) and 4.8(b) respectively.

The training errors for both MHME and HME models are as ploted in Figures 4.7(a) and 4.8(a) respectively. From these plots it obvious that although the approximation and generalization performance of both the MHME and the HME models is equally good, but it took HME model with expectation maximization learning algorithm an additional $40$ iterations to reach the same error goal. Also, partitioning of the input space in the case of HME model is not smooth or fuzzy in nature. Transitions from one expert neural network to another are very sharp as is evident from the plot of Figure 4.8(b). On the other hand, gating network outputs for the MHME model as plotted in Figure 4.7(b), indicate a smooth transition from one input partition to another which is more in line with the spirit of the modular neural networks, i.e., sharing of information between expert neural network models whenever possible. As far as th training efforts are concerned, the both MHME and HME training algorithms took almost the same amount of computing effort when considered in terms of the floating point operations taken to reach the same training error goal. The training simulations were repeated $10$ times for both the HME and MHME models and the results of all the simulations were consistent with the results presented in this section.

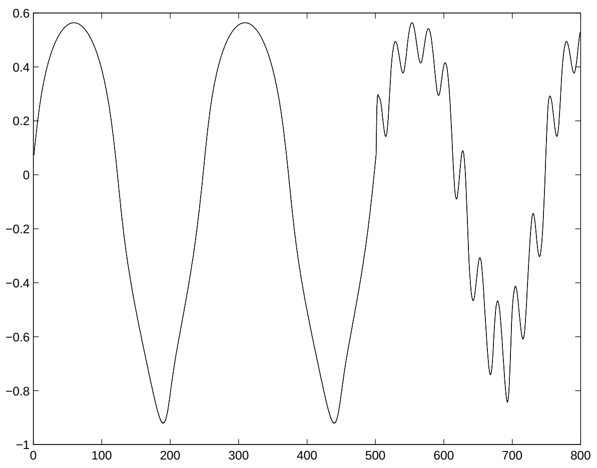
After having established that the proposed MHME model is better than the originally proposed HME model as far training time, training algorithm and performance of the gating network is concerned, the next few examples are used to demonstrate the additional features of the proposed MHME model which have not been explored in any of the previous implementation of the HME model.

### 4.5.2 Example 2

This example is used to demonstrate the usefulness of the input gating scheme which has been introduced in the proposed MHME model. Considering another function approximation problem

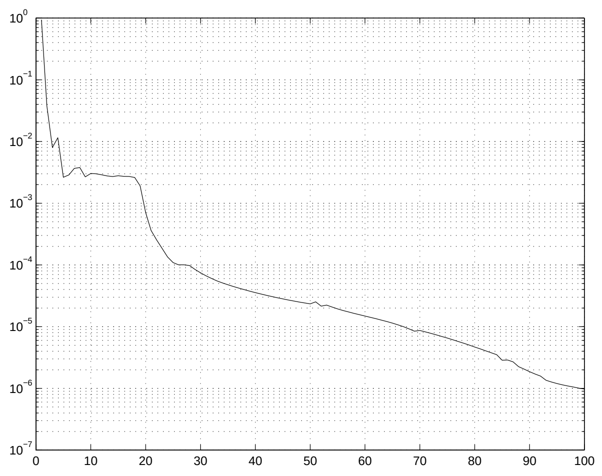


(a)

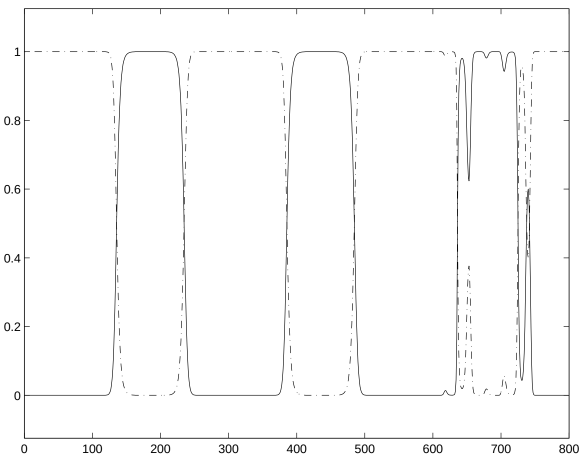


(b)

Figure 4.6: (a) Plots of the MHME model approximation and the actual test data. (b) Plots of the HME model approximation and the actual test data. (solid line plot is for the actual test data and -. line indicates the MHME and HME approximations of test data respectively).



(a)



(b)

Figure 4.7: (a) Plot of the sum of the squared errors for MHME training phase. (b) Plot of the outputs of the gating network.

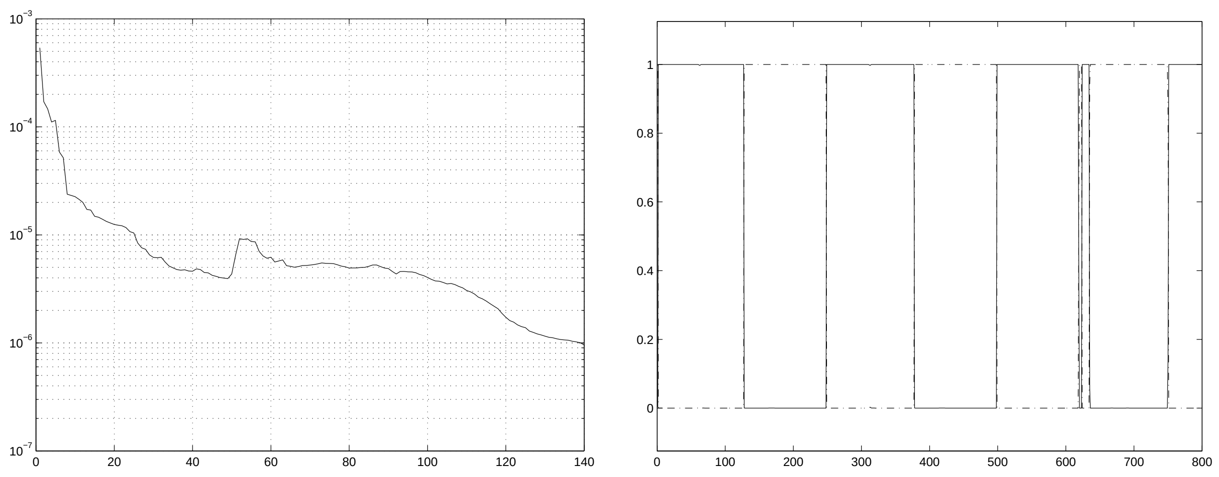


Figure 4.8: (a) Plot of the sum of the squared errors for HME training phase. (b) Plot of the outputs of the gating network.

for a function given by Equation 4.50 with its plot shown in Figure 4.9.

$$f(x) = |10e^{\frac{-x}{\sqrt{1+x}}} sin(0.4\pi x^{1.3})| \tag{4.50}$$

This function is similar in nature as that of function discussed in the previous chapter and has two distinct *humps*, one larger than the other. The larger hump acts as the dominant source of error as compared to the smaller hump when this function is presented to the neural networks for training and hence making it a difficult function to be learnt effectively by the neural networks. A one level MHME model with two expert neural networks and a gating network was implemented. The expert neural networks each had 3 hidden layer nodes and 2 nodes in the hidden layer of the gating network. The weights of the expert neural networks and gating network were initialized in the same fashion as for the example of the previous Section 4.5.1. Assuming that a priori knowledge about this function is available in that it is known that it has two humps and where they start and end. The input gates in the MHME model were designed in a way that the input gate for the first expert neural network was open only for the duration of the first hump and the second input gate was closed for the second expert neural network during the same time period. The same arrangement was made for the second hump as well. The input gating network was open for the second expert neural network for the duration of the second hump only and the first input gate was closed for the first expert neural network for the same time duration. The input to the gating network was the set of the desired input patterns from the training data set. The error goal for the training phase was set to $1e^{-6}$ and the maximum number of training algorithm iterations

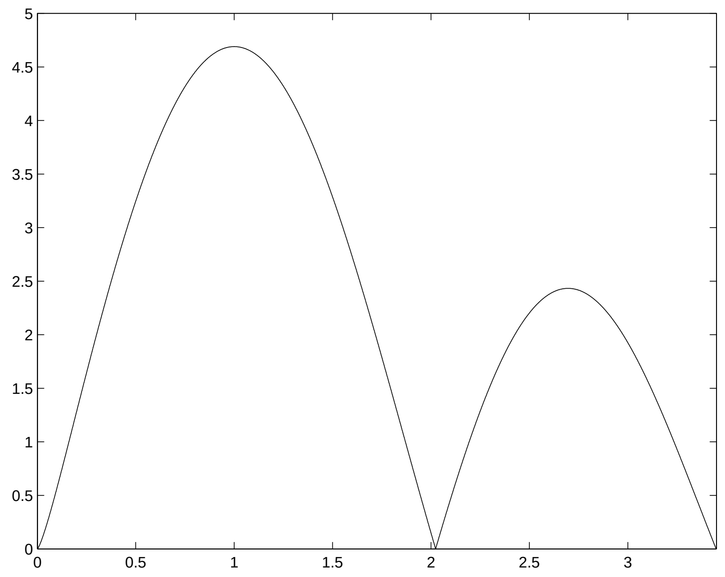


Figure 4.9: Plot of the nonlinear function of Equation 4.50.

was set to $200$. After the training phase, the approximation capabilities of the MHME model were tested on a test data set and the results are as shown in Figure 4.10(a). The approximation capabilities of the proposed MHME model with the input gating scheme were exceptional and the approximation of the test data set by MHME model was again indistinguishable from the original desired output values in the test data set.

The same function was presented to the HME model with two expert neural networks and a gating network. The expert neural networks had the same configuration as in MHME model used for this example, i.e., $3$ hidden layer neurons, and the gating network had $5$ hidden layer neurons. The learning phase did not prove to be effective and it could not reduce the training error to any acceptable limits. The number of hidden layer neurons was increased to $5$ for the expert neural networks and the experiment was repeated again. The approximation results of the HME model on the test data for the second experiment, after the end of training phase, are as shown in Figure 4.10(b). Comparing the plots of Figures 4.10(a) and 4.10(b), it is clear that the approximation capabilities of MHME model are superior than those of HME model for a class of functions and the difficulties faced by the HME model can be overcome by the introduction of an input gating scheme as is the case in the MHME model. Also, the training time for the MHME model was far less than that of HME model and is evident from the training error plots of MHME and HME models which are shown as Figures 4.11(a) and 4.12(a) respectively. The MHME model was able to achieve an accuracy in about $20$ iterations which was achieved by the HME model in $200$ iterations of the EM learning algorithm. The partitioning of the input space by the HME model was also inadequate and inaccurate to solve this problem and the outputs of the gating

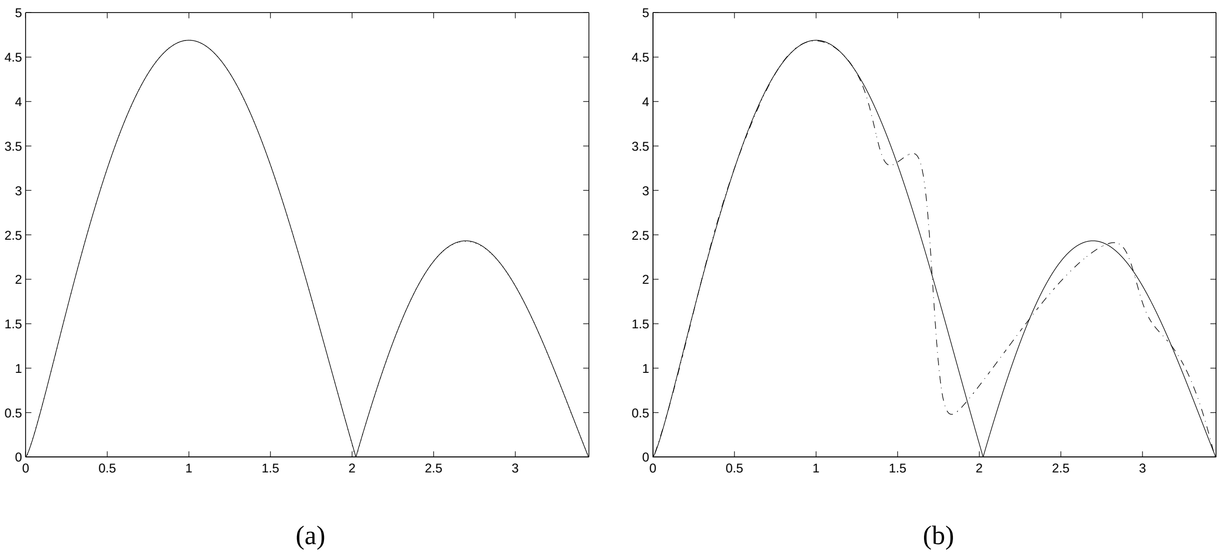


Figure 4.10: (a) Plots of the MHME model approximation and the actual test data. (b) Plots of the HME model approximation and the actual test data. (solid line plot is for the actual test data and -. line indicates the MHME and the HME approximation of the test data respectively).

network for HME model are plotted in Figure 4.12(b). The MHME model was able to partition the problem into two perfect partitions with each partition corresponding to each of the humps of the function to be approximated and is indicated by the plots of the gating network outputs as shown in Figure 4.11(b). The expert neural network sizes were increased by introducing $10$ hidden layer neurons for the HME model and the simulation was carried out again. There was only a slight improvement in the of the HME model performance and the partitioning of the input space still remained imperfect and hence resulted in degraded overall performance.

### 4.5.3 Example 3

Another feature that has been introduced in the proposed MHME model is the input switching to the gating network. As described in the previous sections that this feature will be helpful in the cases where the dimension of the input space is much larger than that of the output space. In this type of function approximation problems, it is better to use the desired output as the input to the gating network as the desired inputs and outputs are paired together in the training data set. The next example demonstrate the effectiveness of stated assumption by a function approximation problem. Consider again the problem of identifying a nonlinear system of Equations 4.47 and 4.48. It is a good example to demonstrate the input switching feature of the gating network as the output of this function is only one dimensional and the dimension of the input space is $5$. An MHME model was implemented with two expert neural networks and a gating network.

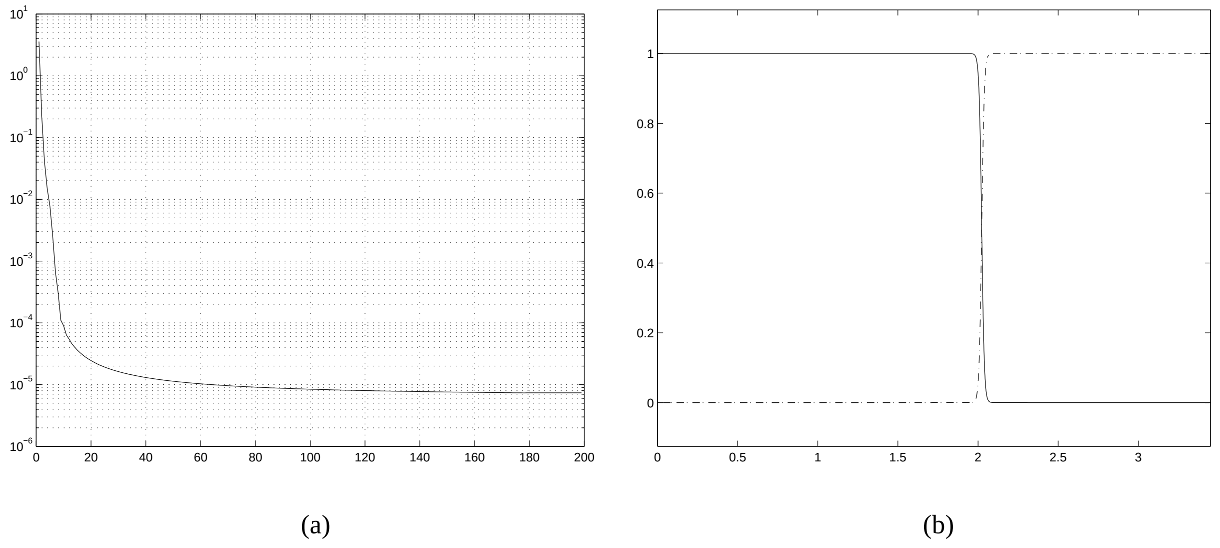


Figure 4.11: (a) Plot of the sum of the squared errors for the MHME training phase. (b) Plot of the outputs of the gating network.

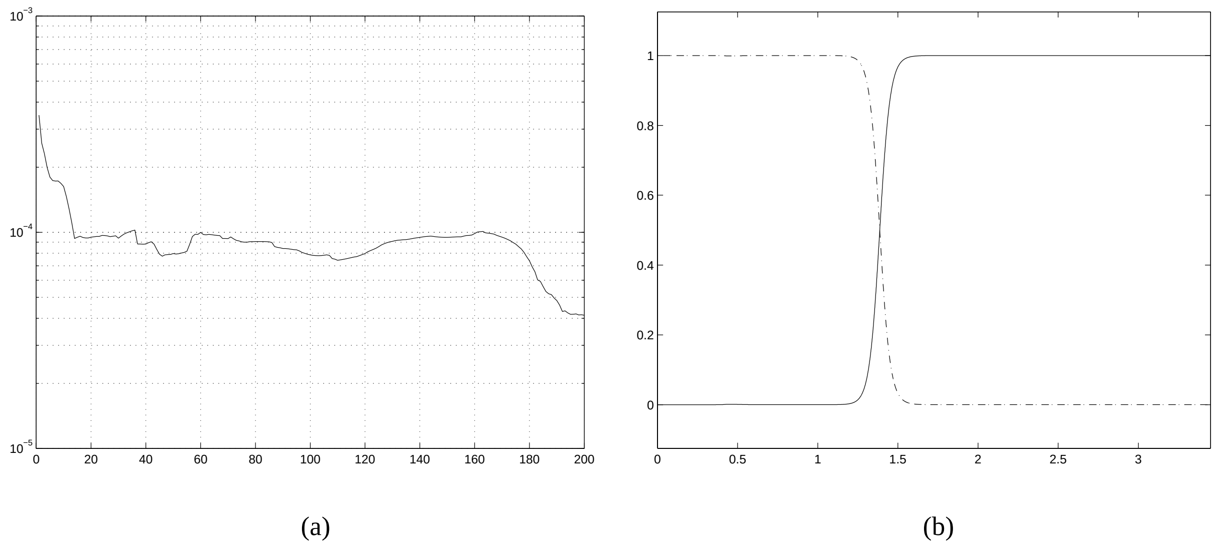


Figure 4.12: (a) Plot of sum of squared errors for HME training phase. (b) Plot of the outputs of the gating network.

The expert neural networks had $5$ hidden layer nodes and the gating network had $2$ hidden layer nodes. The input to the expert neural networks was the $5$ dimensional desired input patterns and the input to the gating network was the one dimensional desired output. The initialization of the MHME model was again carried out as descried earlier in the case of Example 1 of Section 4.5.1. As mentioned earlier that the parameter $\alpha$ used in the softmax squashing function and softmin operator represented by Equations 4.8 and 4.24 respectively, can be used to control the level competition or cooperation among the expert neural networks. As $\alpha \to \infty$, the MHME becomes a competitive MHME model and decreasing the value of $\alpha$ instills cooperation among the expert neural networks in the MHME model. In this example, a competitive MHME model was implemented with $\alpha = 30$. The error goal in this case was also set to $1e^{-6}$ and the maximum number of iterations for the training algorithm was set to $200$. The training of the MHME model was carried out in the similar fashion as before and the approximation results for the test data are as shown in Figure 4.13. Again, the results are similar to those that were achieved while using the desired inputs as the input to the gating network. The Figure 4.14(a) shows the training error. It is worth noticing that by reducing the dimensions of input to the gating network, the overall performance of the MHME model has improved. The competitive MHME model with desired output as an input to the gating network as able to approximate the given function in about $20$ less iterations as compared to the case when the higher dimensional desired input was fed to the gating network as in Section 4.5.1. The gating network outputs are shown in Figure 4.14(b). As the MHME model in this example was implemented in a competitive fashion, the partitioning of the desired output into different partitions is much sharper but still not as sharp as was the case in HME model results for Example 1 of Section 4.5.1.

### 4.5.4 Example 4

This example is used to demonstrate the performance of the MHME model in a cooperative environment in which all the expert neural networks are cooperating among each other instead competing to learn the task at hand. In this case the $\alpha$ was set to $0.5$ and the Example 3 of Sections 4.5.3 was repeated again. The approximation results on a test data are as shown in Figure 4.15 and the approximation results are again indistinguishable from the actual desired test data output. The plot of Figure 4.16(a) indicate that the training time for the cooperative MHME model is comparable with competitive MHME model as implemented in Section 4.5.3. The plots of the gating network outputs as shown in Figure 4.16(b) indicate that the gating network induces a competition among the expert neural networks which is evident from the partitioning boundaries of the input space. At every desired test data point, the overall output of the MHME model is a combination of the outputs of the two individual expert neural networks.

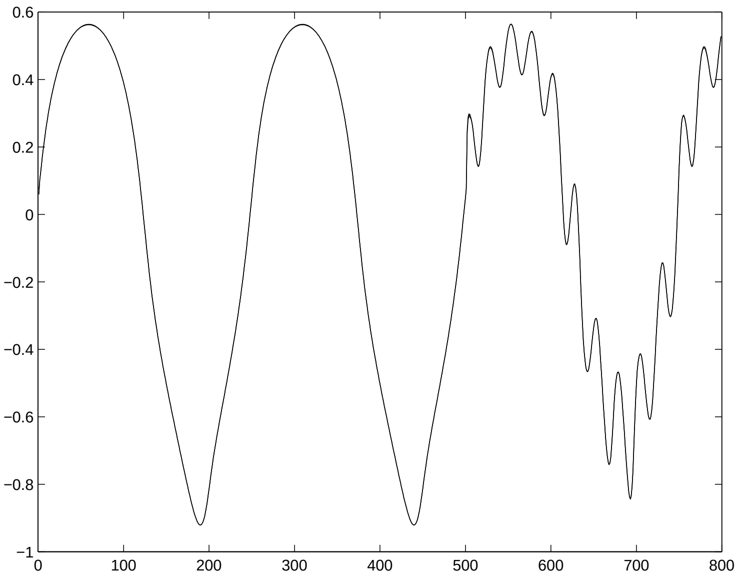
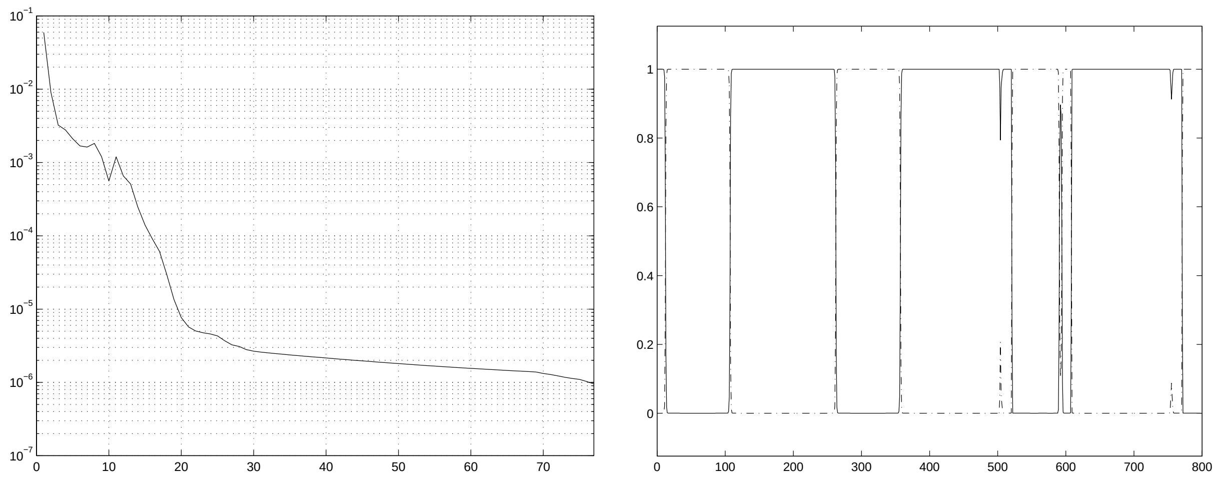


Figure 4.13: Plots of the MHME model approximation and the actual test data (solid line plot is for the actual test data and -. line indicates the MHME and HME approximation of test data respectively).



(a) (b)

Figure 4.14: (a) Plot of the sum of the squared errors for MHME training phase. (b) Plot of the outputs of the gating network.

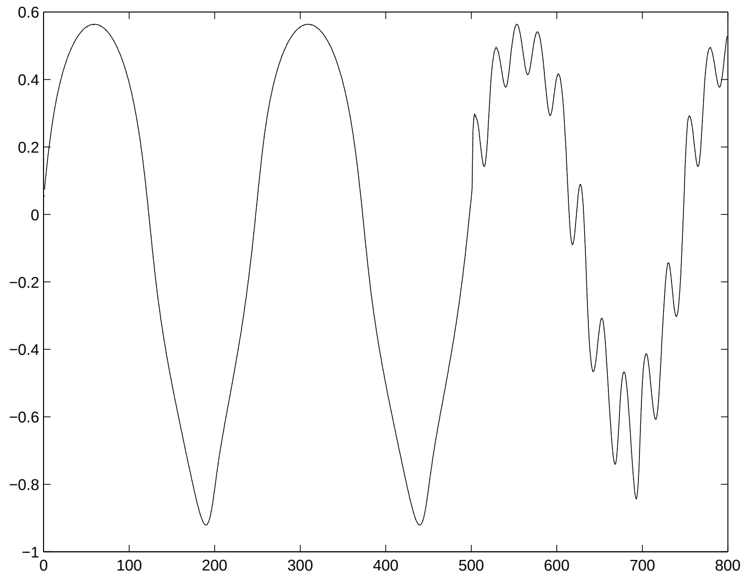
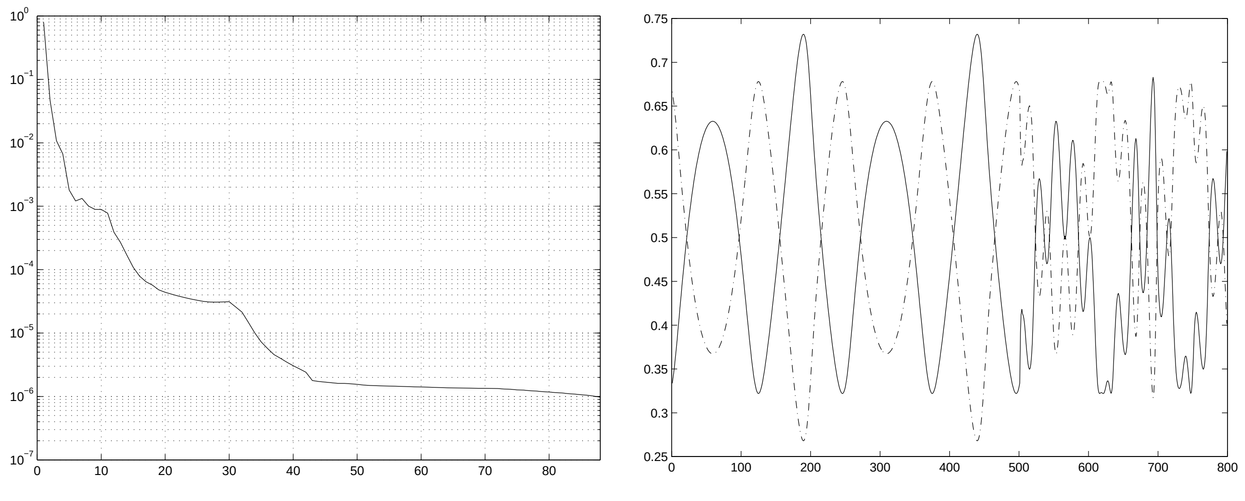


Figure 4.15: Plots of the MHME model approximation and the actual test data (solid line plot is for the actual test data and -. line indicates the MHME and HME approximation of test data respectively).



(a) (b)

Figure 4.16: (a) Plot of the sum of the squared errors for the MHME training phase. (b) Plot of the outputs of the gating network.

### 4.5.5 Example 5

This example is used to demonstrate the classification capabilities of the proposed MHME model using Iris plant database. Iris classic data is perhaps the best known database to be found in the pattern recognition literature [135]. The data set contains $3$ classes of $50$ instances each, where each class refers to a type of Iris plant. One class is linearly separable from the other two classes; the later two or not linearly separable as as shown in Figures 4.17(a) and (b) respectively. The classification prediction output is a class of Iris plant. The classification attributes are four numeric values and contain information about the sepal length, sepal width, petal length, and petal width. These values correspond to $4$ classes of Iris plants, namely Iris Setosa, Iris Versicolor, and Iris Virginica.

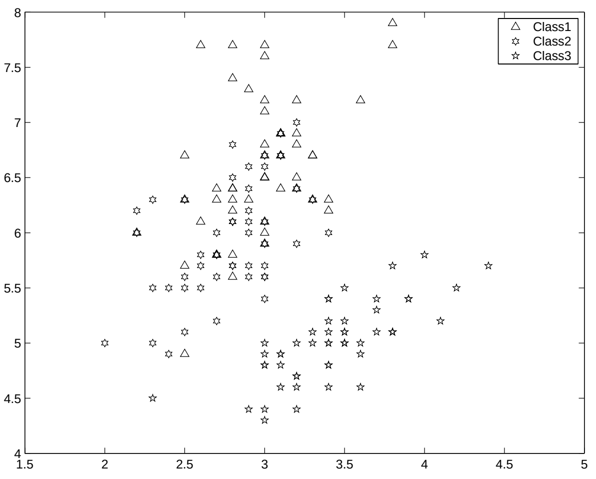
A one level MHME model was implemented with three expert neural network, one for each of the classes in the database. The gating network parameters were initialized using the results of the K-means algorithm. K-means algorithm was used to obtain the centers and spreads of the inherent classes in the Iris database and were consequently utilized as the spreads and centers of the alternate gating network. The output weights of the alternate gating network were initialized randomly to small values. The weights of the expert neural networks for chosen randomly from the range $[-1, 1]$. As there are three classes to be classified in this classification problem, the output units of the expert neural networks had a softmax squashing function because it is a better squashing function for multi-way classification than the commonly used sigmoidal functions.

The training error is plotted in the Figure 4.18(a) and the class decomposition accomplished by the gating network is as shown in Figure 4.18(b). The classification rate was $100\%$ and MHME model was able to classify all the patterns correctly.

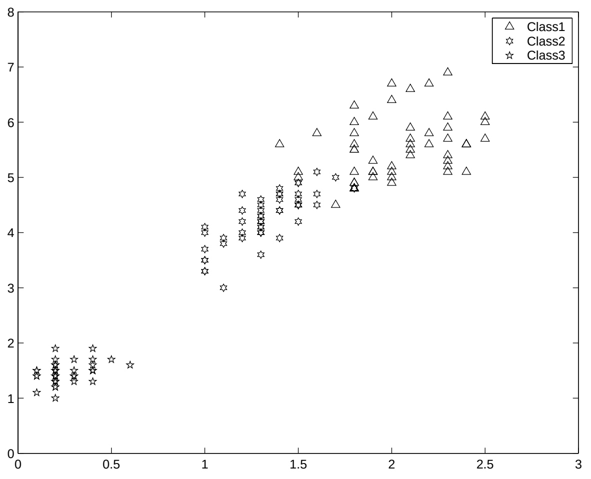
Also, this example demonstrate another feature of the modular neural networks, i.e., an insight into the problem. Assuming that it were not known that one of the classes is linearly separable. Then by evaluating the plots of the gating network outputs, it becomes evident that the two of the classes are not linearly separable as two of the expert neural networks have to work in a cooperative fashion to classify the patterns in those classes and one of the expert neural network was able to learn it all by itself as the third class is linearly separable from the other two.

### 4.5.6 Example 6

This example deals with approximation of the puma robot dynamics. The data set used for this example is generated through a realistic simulation of the dynamics of a Puma 560 robot arm. The task is to predict the angular acceleration of one of the links of the robot arm, given angular positions, velocities, torque and other dynamic parameters of the robot arm. The data set has about 5% noise added to the output, so there is some irreducible squared error present in the data

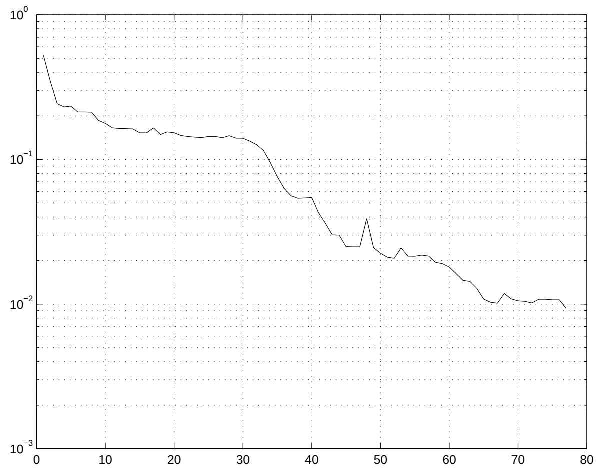


(a)

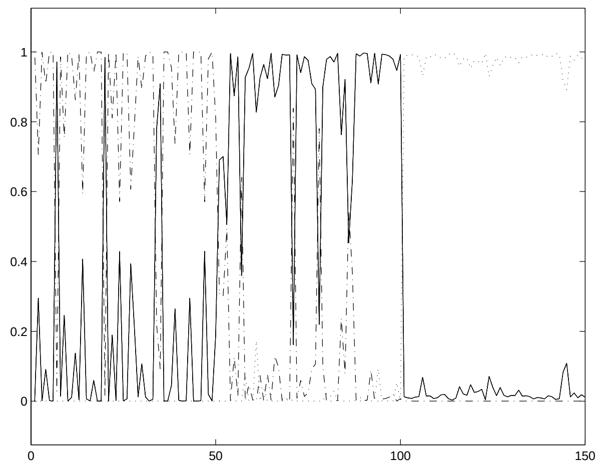


(b)

Figure 4.17: Plot of the features of the Iris data. The features are plotted two by two for better presentation. (a) Feature 1 vs feature 2 (b) Feature 3 vs feature 4



(a)



(b)

Figure 4.18: (a) Plot of the sum of the squares error. (b) Plot of the gating network outputs.

set. The data set has 8 inputs and one out, the acceleration of the joint.

Just like previous examples, a one level MHME model was constructed with two expert neural networks and a gating neural network. The weights of the expert neural networks and gating network were initialized in the same fashion as for the example of the previous Section 4.5.1. The training of the MHME model was carried out for $200$ iterations and the sum of the squared training error is plotted in Figure 4.20(a). The outputs of the gating networks are as shown in Figure 4.20(b). The MHME model output for a test data set is shown in Figure 4.19. It is evident the approximation from the plot that the approximation quality is not very good because of the inherent nonlinearities and the noise present in the data set.

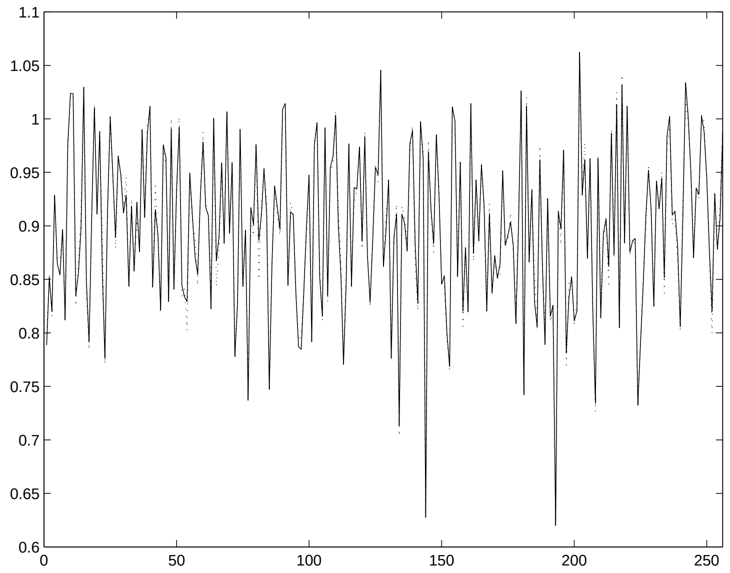


Figure 4.19: Plots of the MHME model approximation and the actual test data (solid line plot is for the actual test data and dotted line indicates the MHME model approximation).

## 4.6 Conclusions

In this chapter a modified hierarchical mixture of experts model is presented. This model, unlike prevalent modular neural network models, is an extension of the commonly understood artificial neural network concepts; is simple in nature and intuitive in its implementation. A novel and mathematically robust gating network was also introduced to enhance the capabilities of the proposed MHME model. The proposed model has proven to have very good initialization, learning, classification, and generalization abilities as has been demonstrated by a number of illustrative examples.

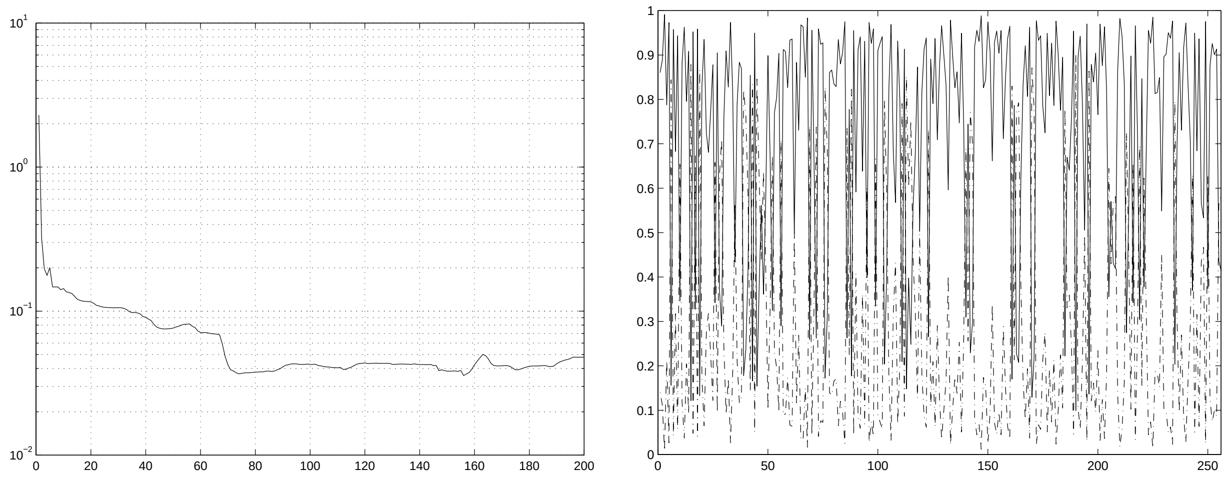


Figure 4.20: (a) Plot of the mean sum of the squares error. (b) Plot of the gating network outputs.

In this chapter the expert neural network models considered in the examples had the same configurations and topologies. Varying the topologies and configurations of the expert neural networks could result in the better performance because of the reason that the divided sub tasks are not same in nature so it makes sense to have a topologically different neural network for different tasks.

Also, another venue of further research could be employing different learning rules for the different specialist modules in the MHME architecture. Like for example, one of the specialist modules could be rules based system well suited for the portion of the overall task whereas rest of modules could be artificial neural network based.

One of the essential problems with the MHME and the HME architectures, as is the case with other neural network architectures, is the model selection. The MHME or the HME models, before they can be trained and applied to a classification or regression problem, require the choice of optimal configuration and topology, i.e., the selection of appropriate number of experts and depth of the hierarchy best suited to solve the problem at hand. This calls for a further investigation for a methodology by which either constructive or pruning approaches could be applied to the proposed modified hierarchical mixture of experts architecture to achieve optimal topology.

# Chapter 5

# Self-Scaling Modular Neural Network Models

A practical issue that arises when applying neural network models to solve any problem is the model selection. Model selection is a methodology for choosing the adequate size of a neural network model to learn the task, yet not compromising the neural network performance. One of the essential problems with the proposed modified hierarchical mixture of experts model presented in this dissertation and the original hierarchical mixture of experts model is the model selection. Applying HME or MHME models to classification or regression problems requires the choice of structural parameters such as number of experts and the depth of the hierarchical neural network model. These parameters are generally determined either by utilizing prior knowledge about the task at hand or by some other ad hoc heuristic methods.

This chapter outlines biologically and evolutionary plausible motivations for modular neural network architectures, called *self-scaling* neural network models. The self-scaling neural networks scale their topology in an incremental fashion so as to optimally match the complexity of the task. The self-scaling neural network architectures are categorized into two broader types, namely hierarchical and vertical, depending upon the way the growth of the neural network models proceeds. The learning algorithms for the proposed self-scaling neural network models also presented in this chapter. The proposed learning algorithms employ the principle of divide and conquer that is used in an iterative fashion and hence are referred to as *iterative divide and conquer* algorithms. Also, illustrative examples are presented to prove the effectiveness of the proposed neural network architectures and respective learning algorithms.

## 5.1 Motivation

In the recent past, artificial neural networks have emerged as powerful tools in the fields of classification and complex nonlinear functional approximation. However, before an artificial neural network can be used to accomplish any task, training or learning phase has to be carried out during which an artificial neural network model parameters or weights are adjusted in a way that the model performs the desired task as optimally as possible. Besides estimation of the neural network model parameters, an equally important issue is the model selection, i.e., to select a suitable neural network topology and configuration. As stated earlier, artificial neural networks are a biologically inspired paradigm and their functioning, to an extent, emulates the functioning of the vertebrate brain. Then in order to resolve the model selection issue, it makes sense to look closely again at the structure and functioning of the brain and draw some inspirations for the model selection methodology.

The basic organization of the human brain makes sense only if viewed in the context of evolution. Evolution tends to be conservative in a sense that it continues to make use of what has been developed already and abolishes only the obsolete and non-essential entities. As the brain evolved, additional structures developed that replaced or enhanced the older brain structures or regions. In some cases the older regions did not essentially disappear but may have assumed different roles [136].

In order to understand the functioning of the brain, it is important to understand the neuron, the basic functional unit of the brain. The brain is what it is because of its structural and functional organization, i.e., how the individual neurons are put together, how these neurons interact and function together. In the brain, neurons can be found in a large varieties of shapes and sizes. These neurons can be can be categorized into several basic categories like sensory neurons, principal neurons and intra-neurons. Although the brain has specialized local regions, the neurons within these specialized regions still differ from each other as far as their response to different stimuli are concerned [137, 138].

Advances in neurophysiology have confirmed that the neurons in the vertebrate brains typically have dendrites extending from neuron cell body that receive inputs form other neurons through connections called synapses. The synapses operate by chemical transmission and when a synaptic terminal receives an all or nothing action potential from the neuron of which it is a terminal, it releases a transmitter which crosses the synaptic cleft and produces either depolarization or hyper polarization in the post synaptic neuron, by opening a particular ionic channel. Summation of such depolarization or excitatory inputs within the time constant of the receiving neuron, this is typically $20 - 30$ ms, produces sufficient depolarization that the neuron fires an action potential. The firing rates of neurons need not to be the same in a localized region of brain and can be time dependent as well. The activation function commonly used in artificial neural networks relates to the output activity of a biological neuron and emulates its firing the rate [139].

The neural networks design inspirations that can be drawn from the preceding discussion are that an efficient model selection procedure should cater for the structural additions or deletions to an existing neural network model and the neurons in a neural network model need not to be of the same type and/or functionality. The neural network architectures presented in this chapter are primarily based on these neurobiologically motivated assumptions.

## 5.2 Model Selection in Neural Networks

Methods using back propagation of the output error to train an artificial neural network perform gradient descent only in the weight space of an artificial network with fixed topology. In general, this approach is useful and successful only when the network topology and configuration is chosen correctly. Too small a network cannot learn a problem well and on the other hand an oversized neural network will lead to an over fitting of the training data set and subsequent poor generalization performance.

The problem of model selection for the artificial neural networks has been tackled in a variety of ways. The simplest form of model selection for the neural networks is to train a number of neural network models with different configurations and topologies simultaneously. Then by evaluating the performance of each of the individually trained neural networks on an independent data set, the best performing neural network that generalizes well is selected and is assumed to be the optimal neural network, as far as the desired configuration and topology is concerned for a given learning task.

The model selection problem can be handled in a better fashion when the process of neural network parameter estimation is carried out simultaneously along with a neural network topology altering methodology. Recently, many researchers have investigated different approaches that alter the network architecture or topology as the learning of a neural network progresses.

One such approach involves using a larger than needed neural network and training it. The oversized architecture is evaluated to detect obsolete and ineffective weight connections or neurons in the neural network model which then are removed before further evolution of architecture is continued. This process is repeated iteratively until a predetermined stopping criterion is achieved. The algorithms following this approach are called *pruning* algorithms [140].

Another approach uses an opposite strategy to reach an optimal topology for a neural network model. This approach attempts to search for an appropriate neural network topology by starting with a small neural network model and successively adding hidden nodes or layers until reaching a satisfactory topology that can optimally perform the given learning task. The basic idea of all of the growing or constructive methods is to use some criterion to dynamically extend the existing neural network model to improve its overall performance. The training algorithms falling into this category are called *constructive* algorithms [141].

Constructive approaches to the neural network design and learning have some advantages over the neural network pruning approach. Firstly, it is very easy to specify the initial neural network configuration and topology, whereas while using a pruning algorithm based learning approach, it is very hard to guess how large initial neural network is needed to solve a given problem. Secondly, the constructive learning starts with a small neural network model and is thus more computationally plausible than pruning based learning in which majority of the training time is spent training an overly large neural network. Thirdly, constructive algorithms for neural network design, generally tend to find smaller neural networks than that of the pruning algorithms [141]. Based on the aforementioned observations, a constructive approach is followed in this chapter in order to design and implement the proposed self-scaling neural network models.

There are also some factors that need to be taken into consideration when following the constructive approach to the neural network design and training. For example, one of the most important factors is to determine when to stop further addition of the hidden nodes and layers or how many hidden nodes or layers to add after the performance of the network does not improve significantly. In addition, a decision has to be made about the selection of a training mechanism as well. For example, whether to train the whole neural network after the addition of hidden nodes and layers or only to train the newly added hidden nodes and layers.

## 5.3 Constructive Neural Network Methods

Effective constructive algorithms for monolithic neural network training and construction reported in the literature are few and are briefly described in this section for reference. The most widely used and referred to constructive method is cascade-correlation architecture [107] and its variants [142, 143]. The cascade-correlation algorithm constructs a neural network that has multiple hidden layers by additions of a single hidden node at a time. This structural arrangement allows the incremental development of powerful feature detectors. A major problem with the cascade-correlation algorithm is that as the number of hidden layers increases, the computational complexity increases as well. Also the network generalization capability decreases because some of the neural network weights might be irrelevant to the optimal operation of the neural network. As the neural network size increases, it give rise to propagation delays and ever increasing fan-in of the hidden layer nodes as more nodes and layers are added.

Group Method of Data Handling (GMDH) and its variants are another widely accepted constructive algorithm [144, 145, 146]. In GMDH type of algorithms, a neural network's hidden layer neurons accept a fixed number of connections. These connections come from a possible combination of network inputs or other existing hidden layers nodes in the network. Thus, when a new hidden layer neuron is added, it results in a number of candidate neural networks. Searching for one effective neural network out of all possible combinations is accomplished by applying one of the available learning algorithms for the neural networks and selecting a best performing candidate neural network. Convergence properties of these types of algorithms are not still very well established because of the ad hoc nature of these algorithms.

Dynamic node creation (DNC) category of algorithms [147, 148, 149] are variants of the originally proposed dynamic node creation algorithm [150]. In this approach, hidden layer neurons are added one at a time and are always added to the same hidden layer. The whole network is retrained after addition of each new hidden layer node. These algorithms have good convergence properties, but computational efforts increase with an increase in the network size when dealing with complex problems.

Projection pursuit regression algorithms are inspired from the statistical literature [151]. As with DNC, a number of hidden layer nodes are always added to the same hidden layer. Instead of retraining the entire neural network model and adding hidden layer nodes with sigmoidal squashing functions, these algorithms use hidden nodes with squashing functions of more complicated functional forms and train only the newly added hidden nodes.

As in the case of monolithic neural networks, the constructive methods for hierarchal mixture of experts reported in literature are very few and are briefly described in this section. The HME growing model presented in [152] is an extension of the adaptive growing tree concept used for classification and regression trees (CART) [91]. In the presented model, a HME tree is initialized with two experts and a gating network. During the training phase, at some point of time, all the terminal nodes or the experts neural networks of the HME tree are split into two. The corresponding gating networks are also split. The parameters of the new gating and expert neural networks are generated by an addition of small randomly generated values to the original gating and expert neural networks parameters before the splitting was carried out. Each of the splits are evaluated on the training data set. The best performing split is the one that can maximize the log likelihood of the overall HME architecture. Only the best performing split is incorporated into the HME architecture and the remaining splits are discarded. This splitting rule is similar to the splitting criterion used in CART model that uses the maximization of the entropy of a given node in the tree. The problem with this approach is that all of the experts are split without taking into consideration the performance of the individual expert neural network models. The best performing expert neural network will be split in the same fashion as that of the worst performing expert neural network. In addition, this approach can be susceptible to poor initialization of the expert neural networks that are added to the HME architecture as a result of the splits, and may be detrimental to the subsequent learning progress of the HME architecture. A similar constructive approach to the HME model is presented in [153]. The only difference is that a second order parameter optimization approach is followed, as opposed to the EM algorithm that is used in [152].

A growing and pruning methodology for the HME model is presented in [154] for binary classification problems. In the presented model, the gating network is used to generate hyperplanes to differentiate between different instances of the presented desired inputs.

Another pruning and growing approach for the HME model is presented in [86]. The pruning or growing of the HME model is performed in an expert-dependent likelihood fashion along with taking into consideration the data being processed, which in the presented case is the speech context modeling. The expert with a minimum likelihood is pruned or split and replaced. The model presented in this approach is a specialized modular network that is custom tailored for speech recognition tasks and the splitting criterion is the task dependent. One drawback of the presented growing and pruning framework is that it is not generalized enough that it can be applied to any other learning task than speech context learning.

An alternate approach to modifying the topology of the mixture of experts model adaptively is presented in [155] and follows similar approach presented for mixture of experts model in [156]. Both constructive and pruning methods for mixture of experts model are presented in [155]. The presented approach does not consider the HME model in the traditional sense; the mixture of experts model is trained for some iterations and if the existing model fails to reduce the mean squared error on the validation data set, the network parameters are saved. A new expert neural network is added to the region of influence of the worst performing expert neural network model with the same weights as those of the already existing expert neural network model for the same region of influence. The gating network used in the presented mixture of experts model is the localized version of the gating network as proposed in [123]. The additional parameters for the gating network corresponding to the newly added expert neural network model are chosen such that the values of the corresponding $\alpha$ parameter is reduced when compared with the $\alpha$ parameter value for an already existing expert neural network for the same region of influence. The corresponding variance $\Sigma$ of the new expert neural network model is the same as that of the old one for the same region of influence. As this approach does not consider the HME model, it is not discussed or considered further in this dissertation.

## 5.4 Self-Scaling Modular Neural Network Models

Artificial neural network architectures, monolithic or otherwise, suffer from the pre-determination of the topology and configuration problem before being trained to learn a specific task. To deal with this problem, an alternative approach to the neural network design is presented in this chapter, called self-scaling modular neural network models.

Self-scaling modular neural network models are a class of neural networks that scale their structure to match the complexity of the given learning task in an incremental fashion. Self-scaling is a function of the neural network architecture whose structure can be automatically generated by using an elaborate algorithm for a given task during the training phase. Self-scaling modular neural network models belong to the category of constructive neural network models because the structures of these neural network models grow incrementally according to a structure extension criterion until a satisfactory neural network training performance is achieved.

Self-scaling modular neural network models are generated through an iterated application of the divide and conquer principle. The individual components of a self-scaling modular neural network can be any of the most prevalent feed forward neural network models and any existing learning algorithm can be employed to train the individual component neural networks.

The self-scaling modular neural networks presented in this dissertation are further subdivided into two categories, vertically and hierarchically self-scaling modular neural networks. The names vertically and hierarchically self-scaling depict the way in which a neural network architecture will grow. For example, if the additional hidden layers or neural network modules are added in a fashion that the neural network structure grows in a vertical direction, then this type of neural networks are categorized as the vertically self-scaling modular neural networks. The hierarchically self-scaling modular neural networks, on the other hand, refer to the neural network architectures that expand their structure in an hierarchical fashion. These two types of self-scaling neural network models are described in detail in the following sections.

## 5.5 Hierarchically Self-Scaling Modular Neural Network Model

As stated earlier the modified hierarchical mixture of experts model presented in this dissertation suffers form the problem of the pre-determination of the structural parameters before it can be applied to a specified learning task. A hierarchically self-scaling modular neural network (HSMNN) model is presented in this section which overcomes the problem of the structure pre-determination by automatically self-scaling itself to reach the preset desired learning performance goals. The HSMNN model incorporates many desired improvements to the constructive HME models presented in [152, 86].

The major drawback of the growing HME model described in [86] is that it is a highly specialized neural network model designed only for the speech context learning. The growth criterion is learning domain dependent and the presented growing HME model cannot be utilized for any other task without incorporating any major changes in the presented learning algorithm. On the other hand, the growing HME model presented in [152] has the drawback that all of the expert neural networks are split into two, irrespective how well an individual expert neural network has been performing, if the HME model performance has not improved after certain training iterations. This can result into the loss of good expert neural network modules that have been performing well on the training data set and a degraded subsequent learning performance of the overall HME model.

The proposed HSMNN model overcomes the problems mentioned in the preceding paragraph by a novel structure evolving learning scheme. The HSMNN model starts with an initial MHME network with a gating network and two expert neural networks. The initial MHME network is trained by evidence maximization, as described in the previous chapter, for a number of iterations, related to a "patience" factor. When training iterations exceed the preset patience factor, the overall performance of the MHME model at that iteration is evaluated against the performance for the iteration at the start of the patience factor interval. If there has been a pre-determined percentage increase in the performance of the MHME model, then the training is continued with the same MHME structure.

On the other hand, if the training performance did not increase, or in other words there has not been pre-determined percentage decrease in the overall MSSE error of the MHME model, the belief in the performance of each of the expert neural network models is evaluated, called belief factors. This is accomplished by multiplying the corresponding gating network outputs, $g_j^i$s, by traversing up the tree structured MHME model, starting with the output of the gating network directly connected to the expert neural network model for which the overall belief is being calculated. The expert neural network with the lowest belief factor is split into two new expert networks leaving the rest of the expert neural network models intact.

For example, considering the initial MHME model as shown in Figure 5.1 and assuming that after carrying out the training of the initial MHME model using the evidence maximization algorithm for certain iterations, the patience factor, the expert neural network model $1$ at level $1$ is found to have the lower belief factor compared to the other expert neural network model. The expert neural network model $1$ at level $1$ is split into two expert neural networks and a new gating network is created to combine the outputs of the newly created expert neural networks; thus adding a new level of hierarchy to the existing MHME model. This is shown in Figure 5.2. As a result of this learning scheme and the training phase, the final resultant HSMNN model is an asymmetric hierarchical structure.

The newly created gating network to combine the new expert neural network models is a replica of the gating network connected to the expert neural network being split, with a very small perturbation of the output weights. The reason for this replication procedure is that the gating network has already learned the partitioning information of the input space in that region. The knowledge the gating network has, can be retained and utilized again by the new gating network instead of starting with a completely random set of weights and learning how to partition the input space all over again.

The new expert neural networks are thus said to be generated according to embryo-genetic principles. In the embryo-genetic process for generation of the new expert neural network models, the old expert neural network that has been earmarked to be split is cloned by a number of new expert neural networks. The weights of the newly generated expert neural networks are the same

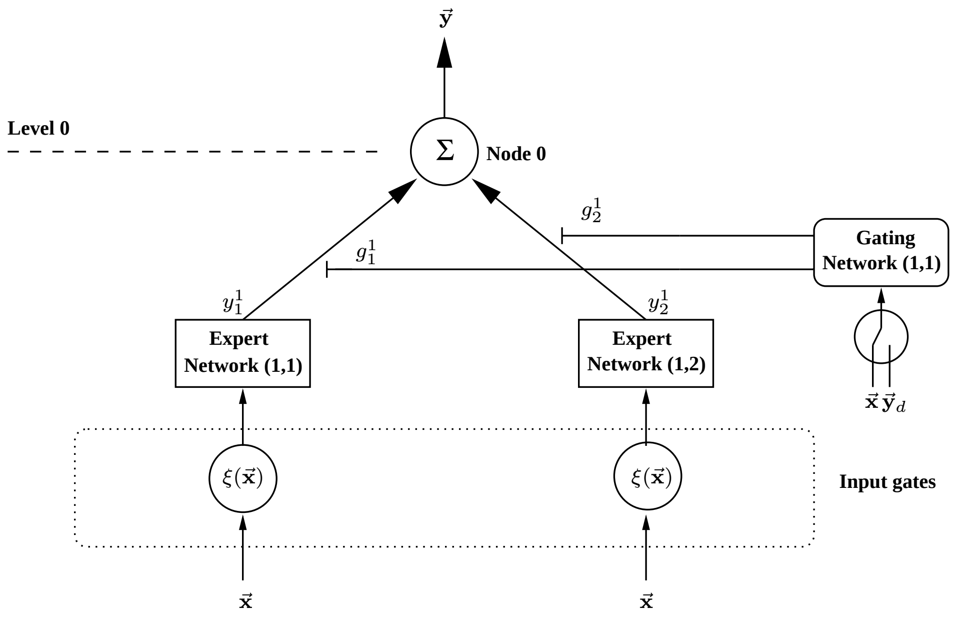


Figure 5.1: HSMNN model with one level hierarchy before an expert neural network model is split.

as the weights of the expert neural network model being split, but are perturbed with small random values, thus, emulating the genetic mutation process. The process of generating the weights for the new expert neural networks ensures that the knowledge possessed by the original expert neural network about the task is not lost and can be used again by the newly created expert neural networks. The newly generated expert neural networks are evaluated to determine an associated performance index for each of the newly generated expert neural network models. The performance index, $p_i$, of a newly created $i$th expert neural network model is a convex combination of the similarity and error indices and is given by the following relationship:

$$p_i = (1-\lambda)(1-e_i) + \lambda s_i \qquad \text{with} \quad 0 \leq \lambda \leq 1 \tag{5.1}$$

where $s_i$ and $e_i$ are the normalized similarity and the error indices of the $i$th newly created expert neural network, respectively, and range form $0$ to $1$. The normalized values of the similarity and error indices are obtained by dividing all of the individual error and similarities indices of the newly created expert neural networks by the largest values off the respective index. The value of $\lambda$ indicates the trade of between diversity and accuracy when evaluating the performance index of a new expert neural network.

The un-normalized similarity index for the $i$th newly generated expert neural network is given by

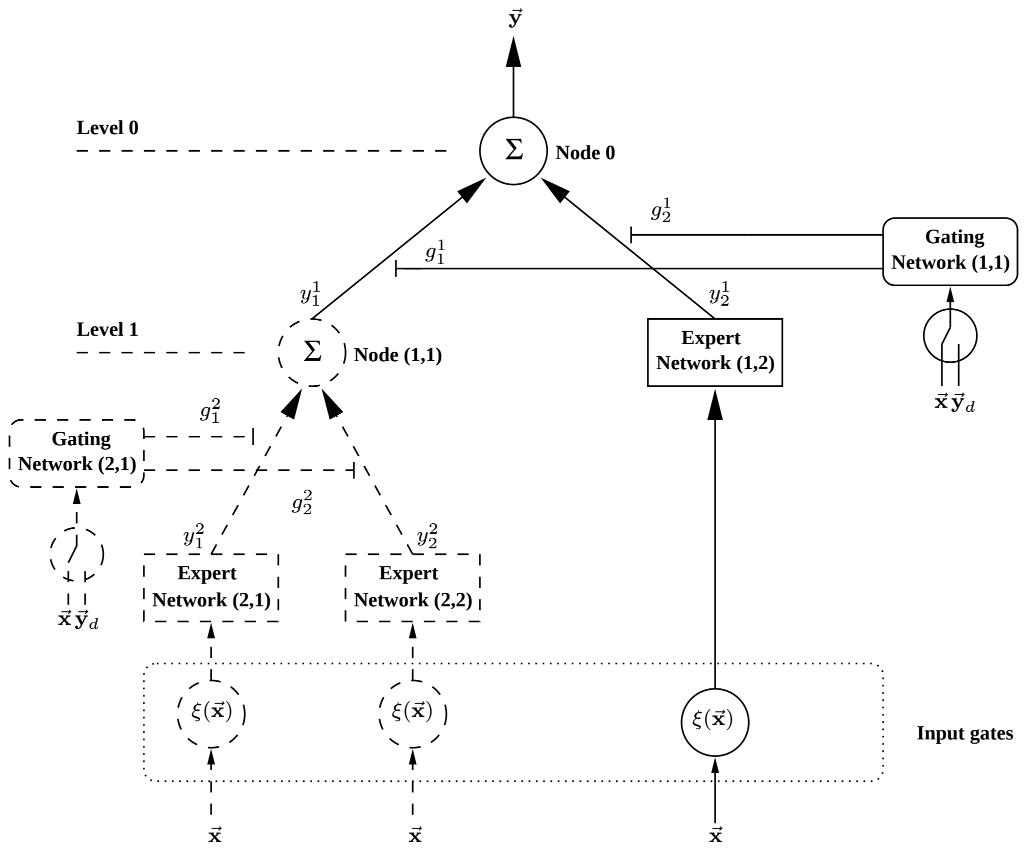


Figure 5.2: HSMNN model with two levels of hierarchy after an expert neural network model is split into two expert neural networks. A gating network is introduced in place of the old expert neural neural network to mediate the output of the newly added expert neural networks.

Equation 5.2.

$$\hat{s}_i = (\vec{\hat{y}} - \vec{y}_i)^T(\vec{\hat{y}} - \vec{y}_i) \tag{5.2}$$

where $\vec{\hat{y}}$ is the overall output of the HSMNN model before an expert neural network model was split and the $\vec{y}_i$ is the output of the $i$th newly created expert neural network model when presented with the training data set. The similarity index is a measure of the similarity between the output of the $i$th newly created expert neural network and the overall output of the HSMNN model before the split was carried out. As discussed in the previous chapter, the aim of designing the HSMNN model based modular neural networks is to generate as many diverse expert neural network models as possible, so the similarity index is made part of the selection criterion to select a pair of expert neural networks to be added to the existing HSMNN model structure.

On the other hand the un-normalized error index, $e_i$, measures the effect of the newly generated $i$th expert neural network model on the overall performance of the new HSMNN model after the inclusion of the new gating network and the newly generated $i$th expert neural networks; its formulation is given by Equation 5.3.

$$\hat{e}_i = (\vec{y} - \vec{\hat{y}}_i)^T (\vec{y} - \vec{\hat{y}}_i) \tag{5.3}$$

where $\vec{\hat{y}}_i$ is the output of $i$th newly created expert neural network. The inclusion of the error index in the performance index ensures that the performance of the $i$th newly created expert neural network model on the training data set is also taken into consideration when selecting a suitable expert neural network to be included in the existing HSMNN Model.

The overall performance index ensures that only eligible newly created expert neural networks are selected for inclusion into the existing HSMNN model by considering both their similarities and the error contributions. The parameter $\lambda$ in the performance index can be used to emphasize either of the two selection criteria. For example, if newly added expert neural networks are meant to be diverse, without caring for their contribution to the overall error of the HSMNN model, $\lambda$ can be set to $1$. On the other hand, setting $\lambda$ to $0$ would ignore diversity and will only concentrate on the error contributions of the newly created expert neural networks when added to the existing HSMNN model.

As every expert neural network earmarked for splitting and to be replaced by two new expert neural networks, that expert neural network with the highest performance index is selected from the pool of newly generated candidate neural networks and removed from the pool of the available expert neural networks. The remainder of the pool is searched again for another expert neural network with the highest performance index to be included into the HSMNN model.

To ensure that there is no degradation in the learning performance of the overall HSMNN model, if a suitable pair of the expert neural network models is not found in the first attempt, the embryogenetic process for creation of diverse and effective expert neural network models is carried out again. This process is repeated until a suitable pair of expert neural network models is found that when added to the existing HSMNN model will reduce the training error when HSMNN model training is continued again. The training algorithm for the HSMNN model is formally described and summarized in the following section.

## 5.6 Iterative Divide and Conquer Algorithm I

The training algorithm for the HSMNN model is based upon the iterative version of the principle of divide and conquer and is referred to as iterative divide and conquer algorithm I (IDCA-I). The main theme of the learning algorithm for the HSMNN model is inspired from cognitive psychology and is based on the principles of the selective attention and continued goal-directed learning.

Attention is defined as the taking possession of the mind, in clear and vivid form, of one of what seem simultaneously possible objects or trains of thought [157]. Alternatively, attention is the means by which a living organism focuses on a subset of the available information. Attention is the ability of the living biological organisms to focus on a task, to concentrate and often refers to the reallocation of available processing resources to accomplish a given task. Attention can categorized into different categories like selective attention, divided attention and automated attention. Selective attention is a process through which focus is placed on one of the aspects of the input stimuli.

Living organisms have powerful natural learning mechanisms that allow them to master their natural abilities to achieve their respective goals. Goal-directed learning uses the principles of decision theory to choose and alter learning tasks [158]. Goal directed learning models the learning behavior of an intelligent agent as the process that results naturally from the fact that the agent is situated in an environment where it must perform some tasks to satisfy some goals. Definition of goals is central to the learning process, determining not only what is to be learned but how the learned knowledge should be represented and how it may be used. Resource limitations are also taken into account: since the agent may not be able to satisfy its goals with the limited computational power and training data available to it, prioritization and tradeoffs are essential parts of goal-directed learning.

The HSMNN model incorporates the features of both selective attention and goal-directed learning. The selective attention feature used in the HSMNN model is the selection of the worst performing expert neural network that is subsequently split; this procedure allows the HSMNN model to selectively focus on the region of the input space where the expert neural network models are not able to learn properly. The HSMNN model utilizes the features of the goal-directed learning in the sense that the overall goal of learning process of the HSMNN model is to learn a specific task. This goal is accomplished by redefining the interim subgoals that are achieved iteratively by finding and splitting the worst performing expert neural network models and the subsequent recombination of the newly generated expert neural networks.

The HSMNN model is initialized with two expert neural networks and a gating network at the beginning of the training phase. The output, $\vec{\hat{y}}$, of the initial HSMNN model is generated by the following equation:

$$\vec{\hat{y}} = \sum_{m_1} g^1_{m_1} \vec{\hat{y}}^{\,1}_{m_1} \qquad m_1 = 1, 2 \tag{5.4}$$

The belief factors for each of the expert neural networks 1 and 2 in the initial HSMNN model are the same as the corresponding the gating network outputs $g^1_1$ and $g^1_2$, respectively.

The overall output $\vec{\hat{y}}$ of the proposed HSMNN model is governed by the following relationship:

$$\vec{\hat{y}} = \sum_{m_1} g^1_{m_1} \sum_{m_2=2m_1-1}^{2m_1} g^2_{m_2} \cdots \sum_{m_l=2m_{l-1}-1}^{2m_{l-1}} g^l_{m_l} \vec{\hat{y}}^{\,l}_{m_l} \qquad m_1 = 1, 2 \tag{5.5}$$

where $g^l_{m_l}$ is the output of the gating network at the $l$th level corresponding $m_l$th node at the same level. $\vec{\hat{y}}^{\,l}_{m_l}$ is the output of the $m_l$th expert neural network at the $l$th level in the hierarchy. The output $\vec{\hat{y}}^{\,l}_{m_l}$ for the $m_l$th expert neural network model at the $l$th level is obtained by the procedure described in the Section 4.3. As the training procedure results in an asymmetric hierarchical tree for which the overall output of the HSMNN model is not easy to describe in a nice mathematical formulation, so for the sake of mathematical simplicity, the relationship of the Equation 5.5 is followed to evaluate the overall output of the HSMNN model by assuming the outputs of the non-existent expert neural network models to be zero.

The overall performance of the HSMNN model is evaluated using the usual mean sum of squared error objective function as presented in the following:

$$\mathcal{E} = \frac{1}{2N} \sum_{t=1}^{N} (\vec{y}^{\,(t)} - \vec{\hat{y}}^{\,(t)})^T (\vec{y}^{\,(t)} - \vec{\hat{y}}^{\,(t)}) \tag{5.6}$$

where $\vec{\hat{y}}^{\,(t)}$ is the overall HSMNN model output for the $t$th training pattern.

After training the HSMNN Model for iterations equal to the patience factor, the performance of the overall HSMNN model is evaluated. If the improvement in the HSMNN model is more than a certain pre-set percentage value, then the training is continued with the same HSMNN structure. If on the other hand, the improvement is below the pre-determined percentage value, then the belief factor for each of the expert neural network is calculated. The belief factor for $m_l$th expert neural network at the $l$th level is calculated by following the relationship given by Equation 5.7.

$$\mathcal{B}^l_{m_l} = g^l_{m_l} g^{l-1}_{m_{l-1}} \cdots g^2_{m_2} g^1_{m_1} \tag{5.7}$$

where at the $l$th level $m_l = m_l$, at $(l-1)$the level $m_{l-1} = \texttt{floor}(\frac{m_l+1}{2})$, and at first level $m_1 = \texttt{floor}(\frac{m_2+1}{2})$; where the `floor` being the usual mathematical operator that rounds a number to the nearest integer value towards $-\infty$.

The expert neural network with lowest belief factor is earmarked to be split. A pool of candidate expert neural networks is created according to embryo-genetic principles. The performance of each newly created the expert neural network in the pool is evaluated utilizing individual normalized similarity and error indices according to the relationship of Equation 5.1. A pair of expert neural networks with high performance indices are chosen from the pool without replacement and are included into the HSMNN structure along with a new gating network. The training of the HSMNN model with the new structure is continued until a satisfactory training performance is achieved by repeated application of splitting and addition of newly created expert neural networks to the HSMNN model. The training of the HSMNN model is carried out using the evidence maximization framework as presented in the previous chapter.

The iterative divide and conquer algorithm I used for training of the HSMNNM model is summarized below as the Algorithm 5.1. The presented algorithm takes $\mu$, $m$, $n$, and $\epsilon$ as some predefined parameters that are the patience factor, the number of the hidden layer neurons of the gating network, the number of the hidden layer neurons of the expert neural network models to be added while extending the existing HSMNN model, and the desired error goal respectively.

## 5.7 Illustrative Example

A well known benchmark problem in neural network community is the so-called two-spiral problem. It consists of $194$ two-dimensional vectors lying into two interlocked spirals that represent two separate classes. The plot of the two spirals is shown in Figure 5.3. It is a difficult problem for simple monolithic feed-forward neural networks with simple structure. It has been reported that the problem could not be solved with a standard monolithic feed-forward neural networks and additional direct connections had to be used for achieving convergence [107]. Also, a special constructive learning architecture cascade-correlation was used to solve two-spiral problem successfully [107]. Another result report in literature used 2-5-5-5-1 feed-forward monolithic neural network with direct connections from the input to the output and hidden layers to successfully solve the two-spiral problem [159].

The same benchmark problem was used to evaluate the performance of the proposed HSMNN model. The initial HSMNN model comprised of the new proposed alternate gating network and two expert neural networks. The gating network and the expert neural networks had single hidden layers with $2$ and $4$ hidden layer neurons respectively. The squashing log-sigmoidal function was used as the transfer function for the neurons in the hidden and the output layers of the expert neural networks. The patience factor was set to $20$ iterations and the performance improvement factor was set to $10\%$. The gating network was initialized using K-means algorithm whereas the weights of the expert neural networks were initialized with random values from the range $[-1, 1]$. The value of $\lambda$ in the performance index was et to $0.5$ so that equal weight was given to both the diversity and the performance of the newly created expert neural network model.

The initially created HSMNN model architecture was trained using the iterative divide and conquer algorithm I (IDCA-I) as outlined in the Algorithm 5.1. The training phase was carried out for $2000$ iterations and the record of the MSSE is as shown in Figure 5.4(a). The decision boundaries generated by the final HSMNN model on a test data are as shown in Figure 5.4(b) that

**Algorithm 5.1** Iterative Divide and Conquer Algorithm I

```
Requires: $T = \{\vec{x}_i, \vec{y}_i\}$ for $i = 1, \cdots, N$.  $\mu, \epsilon, m, n$
 • Initialize the HSMNN model.
     ◇ An alternate gating network with one hidden layer comprised of $m$ neurons as described
       in Section 4.3 and two expert neural networks with one hidden layer each comprised of $n$
       neurons.
 $t \Leftarrow 0$
 while $e \geq \epsilon$ do
   if $t \leq \mu$ then
     $t \Leftarrow t + 1$
   else
     $t \Leftarrow 0$
     ◇ Evaluate the belief in each expert neural network using Equation 5.7.
     ◇ Select the expert neural network with the least belief factor and mark it to be split.
     $s \Leftarrow 0$
     while $s < 2$ do
       if FAILURE then
         ○ Generate a pool of candidate expert neural networks by cloning the expert neural
           network earmarked for split using embryo-genetic principles.
         ○ Evaluate the performance index of all the candidate neural networks in the pool.
       else
         ○ Evaluate the eligibility of the best performing expert neural network in the pool to
           be included into the existing HSMNN model.
         if ELIGIBLE then
           ○ Mark the expert neural network to be included in the HSMNN model and remove
             it form the candidate pool.
           $s \Leftarrow s + 1$
           ○ SUCCESS
         else
           ○ FAILURE
         end if
       end if
     end while
     ◇ Create a new alternate gating network based upon the parameters of the gating network
       that was associated with expert neural network that was split.
   end if
   • Add a new level of hierarchy and insert two selected expert neural networks and the new
     alternate gating network into the HSMNN structure.
   • Train the new HSMNN model with evidence maximization algorithm.
   • Calculate overall MSSE $J$ using Equation 5.9.
   $e \Leftarrow J_k^T J_k$
 end while
```

indicates that classification rate was $100\%$. There are no published results for this benchmark problem using constructive hierarchical mixture of experts, so no comparative remarks could be made about the performance of the proposed HSMNN model performance.

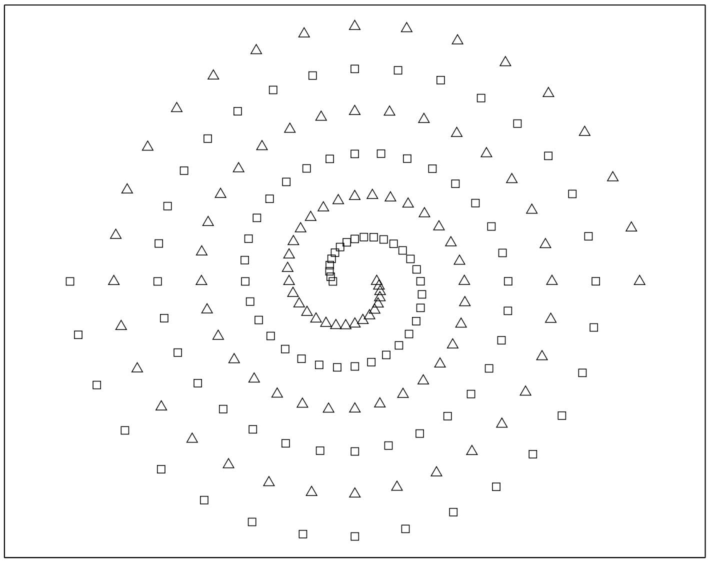
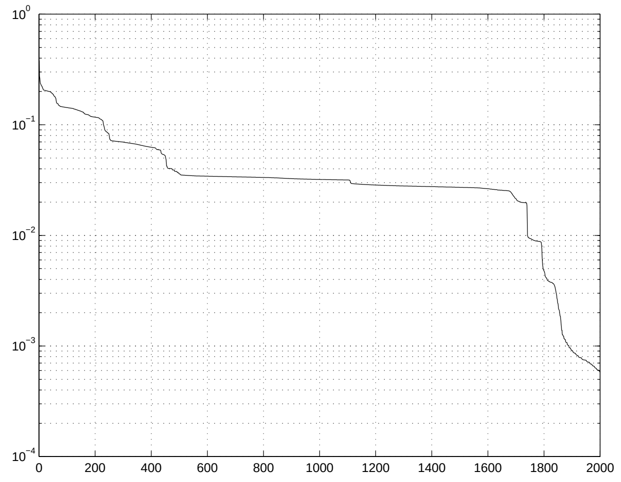


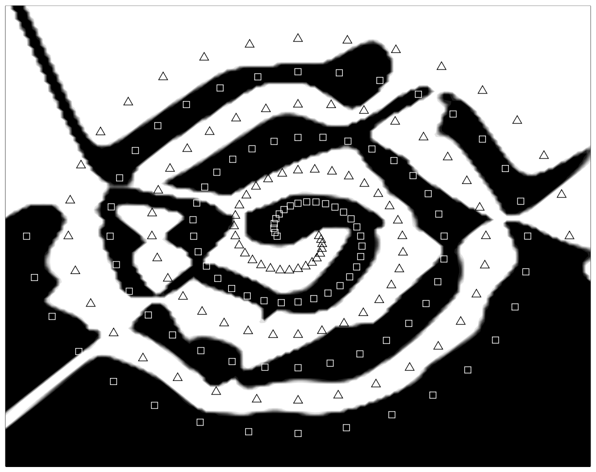
Figure 5.3: Plot of the two interlocked spirals. $\triangle$ and $\square$ represent spiral 1 and 2 respectively.

## 5.8 Vertically Self-Scaling Modular Neural Network Model

Vertically self-scaling modular neural network (VSMNN) model draws its inspiration from a popular constructive neural network model, the cascade-correlation learning model [107] and incorporates some conceptual and structural improvements to the constructive backpropagation (CBP) architecture presented in [160, 161]. The VSMNN model get its name from the fact that additions to the initial neural network structure are carried out in the vertical direction and the neural network structure grows in the downward vertical direction. The cascade-correlation type of learning algorithm has been found to be an efficient technique for its function approximation and classification capabilities [162, 163, 164, 165]. The constructive backpropagation model is another plausible constructive neural network model when dealing with complex learning tasks.

In the VSMNN model, an artificial neural network is constructed in a modular fashion by successive addition and training of multiple hidden layer nodes or neural networks as opposed to the cascade-correlation learning algorithm in which only one hidden layer node is added and trained at a time. A common feature to both the VSMNN and the cascade-correlation models is





(a) (b)

Figure 5.4: (a) Plot of sum squared training error for IDCA-I algorithm. (b) Classification boundaries generated by the HSMNN model that was trained using IDCA-I algorithm.

the freezing of the input and the output weights of the previously trained hidden layer nodes to achieve a better computational efficiency during the training phase.

However, there are two main differences that set the VSMNN and the cascade-correlation models apart. Firstly, a much simpler objective function is minimized during the training phase of the VSMNN model after an addition of the new multiple hidden layer neurons or new neural network modules. This objective function is the most commonly used mean sum squared error (MSSE) function as opposed to the cascade-correlation model that switches between two different objective functions during the training phase that calls for use of two different optimization routines, thus increasing the complexity and practical implementation. On the other hand, the VSMNN model uses only one objective function that makes the implementation of the constructive VSMNN model much simpler when trained using gradient-based learning methods like backpropagation.

The second major difference between the VSMNN model and both the cascade-correlation and the CBP models is the implementation of inhibitory lateral connections between neurons in the hidden layer of the initial VSMNN model topology. As mentioned earlier, the one reason monolithic neural networks do not learn difficult tasks properly is due to the herd effect that arises because of the fact that all of the hidden layer neurons evolve independently of each other and are trying to minimize the same source of error. The VSMNN model resolves the issue of the herd effect by introducing lateral inhibitory connections between the hidden layer neurons of the initial neural network model that makes these neurons dependent on each other and provides a way of communication among them; thus providing a good initial starting point for the VSMNN model.

One potential problem of the cascade-correlation model is due to the fact that it is restricted to the addition of only one new neuron to the hidden layer of the neural network model at a time. This leads to excessively large computational time when constructing a large neural network. A remedy to this problem is the possibility of training several candidate neurons simultaneously and picking the best trained neurons to be added to the neural network model. Moreover, training only one hidden unit at a time to reduce the output residual error may not be a very logical choice as the approximation capabilities of a single neuron are clearly limited. In other words, simultaneous addition and training of multiple hidden layer neurons is potentially a much more well suited solution to reduce the residual neural network model output error than many single independently trained hidden layer neurons, as is the case in the cascade-correlation learning algorithm. Thus both in terms of computational efficiency and modeling performance, it is desirable to add a batch of new hidden layer neurons instead of only one new hidden layer neuron at a time. This problem is also mentioned and addressed in the constructive backpropagation model as presented in [160, 161].

Also, the VSMNN architecture and the corresponding learning algorithm lends itself to ease of switching between online and batch mode learning types which is not the case with the cascade-correlation algorithm that is restricted to use batch mode learning only.

A major drawback of the constructive backpropagation algorithm lies in the fact that the new multiple hidden neurons added to the hidden layer of the existing neural network model are of the same type and are the same in number as that of the previously added and trained hidden layer nodes. This causes a serious problem while training a CBP algorithm based neural network model because if the previously added neurons of the same type and the same number could not reduce overall neural network model output error and could not learn the functional mapping or classification task, the newly added hidden layer nodes will also not be able to learn the desired functional mapping, and the overall neural network model output error will not be reduced. This hypothesis was tested and was proven correct. The VSMNN model based learning overcomes this problem by adding a hybrid neural network (HNN) model to the already existing neural network model. The hybrid neural network model is comprised of a two hidden layer neural network model with hidden layers that are comprised of a combination of linear or nonlinear neurons that are randomly initialized. The hybrid neural network model is shown in the Figure 5.5. The weights connecting neural network inputs to the first hidden layer are initialized at random and are not trained during the training phase. This training mechanism achieves the computational efficiency of a single hidden layer neural network and functional efficiency of a two hidden layered neural networks. Two hidden layered neural networks have proven to be more efficient in learning complex functional mappings and classification tasks than the neural networks with single hidden layer.

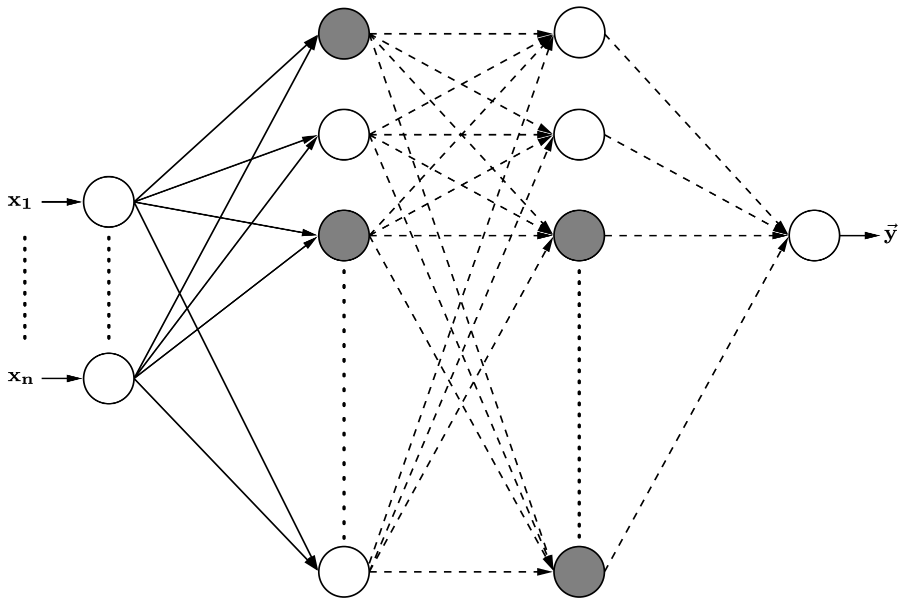


Figure 5.5: The proposed hybrid neural network architecture. The filled circles indicate neurons with nonlinear squashing functions.

Also, the VSMNN model has one more potential advantage over the cascade-correlation learning algorithm is that it lends itself to the applicability of the many well established faster and efficient neural network learning algorithms. In addition, the neural network pruning learning paradigms can be applied to the VSMNN architecture and as far as the cascade-correlation learning algorithm is concerned, this is not the case. The learning algorithm for the cascade-correlation model is primarily a growing algorithm. The learning algorithm for the VSMNN model is described and summarized in the following section.

## 5.9 Iterative Divide and Conquer Algorithm II

The learning algorithm for the VSMNN architecture is based on an iterative version of divide and conquer principle as it tries to divide the problem iteratively into subproblems which then can be conquered by the individual component neural networks. The VSMNN learning algorithm is referred to as iterative divide and conquer algorithm II (IDCA-II). The IDCA-II learning algorithm initially starts with a small neural network with inhibitory lateral connections among the hidden layer neurons.

The initial neural network model is trained for a certain number of iterations, called *patience*

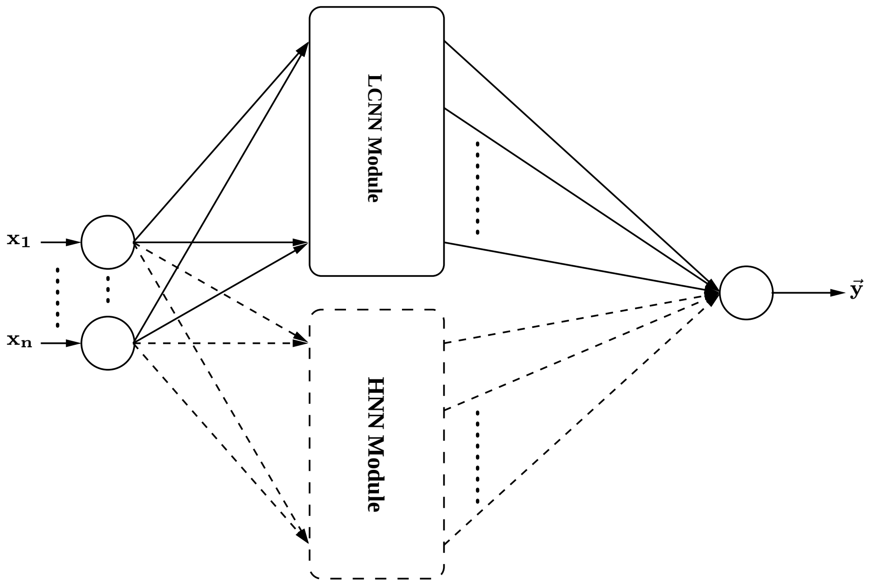


Figure 5.6: VSMNN model after addition of the first hybrid neural network module.

factor. The learning algorithm used to train the initial neural network model is as described in Section 3.3. If there is no substantial percentage drop in the modeling error of the VSMNN model, a new batch of hybrid hidden layer neurons or a hybrid neural network module is added to the exiting VSMNN model structure and the input and the output weights of the previously added neurons are frozen and are not adjusted any further during the course of the VSMNN model training phase, as shown in Figure 5.6. The solid lines indicate the frozen weight connections and the dashed lines depict the weight connections to be adapted during continued training process. The neurons in the newly added batch all have different transfer functions that have slopes greater than the slopes of the transfer function of the previously added batches of the hidden layer neurons. The increase in the slopes of the squashing functions is motivated by the fact that different neurons in the localized regions of the brain have different firing rates and time constants. The increase in slopes of the squashing functions also results in a more localized performance of the individual neurons.

The newly added batch of the hidden layer neurons are then trained to minimize the residual mean sum of squared errors (RMSSE) objective function. Specifically, considering the case when the $j$th HNN model is added to the existing VSMNN model, then the RMSSE objective function is:

$$RMSSE = J_r = \frac{1}{2N}\sum_{k=1}^{N}\left(\vec{y}_k - \sum_{i=1}^{j-1}\vec{o}_i^{\,k} - \vec{o}_j^{\,k}\right)^T\left(\vec{y}_k - \sum_{i=1}^{j-1}\vec{o}_i^{\,k} - \vec{o}_j^{\,k}\right) \qquad \text{for } j > 1 \quad (5.8)$$

where $\vec{y}_k$ is the desired output and $\vec{o}_j^k$ is the output of the $j$ batch of the hidden layer nodes for the $k$th training pattern. The residual error objective function can be minimized by the use of any of the existing gradient-based learning methods for the neural networks.

The overall performance the VSMNN model is measured by the usual MSSE error objective function:

$$MSSE = J = \frac{1}{2N} \sum_{k=1}^{N} \left(\vec{y}_k - \vec{\hat{y}}_k\right)^T \left(\vec{y}_k - \vec{\hat{y}}_k\right) \tag{5.9}$$

where $\vec{\hat{y}}_k$ is the overall output of the VSMNN model for the $k$th training pattern and is given the following relationship:

$$\vec{\hat{y}}_k = \sum_{n=1}^{j} \vec{o}_n^{\,k} \tag{5.10}$$

where $j$ is the number of individual component neural networks in the VSMNN model.

The learning of the VSMNN model is continued by iteratively adding HNN modules to the exiting neural network structure until a pre-determined error goal is achieved. This arrangement makes the IDCA-II algorithm easier to implement because it easier to identify the outputs of the individual batches of neurons and alleviates some of the computational issues.

Given a training data set $T = \{\vec{x}_i, \vec{y}_i\}$ for $i = 1, \cdots, N$, the summary of the IDC-II learning algorithm is presented as Algorithm 5.2, with $\mu$, $m$, and $\epsilon$ are the patience factor, the number of the hidden layer neurons of the HNN module to be added while extending the existing VSMNN model, and the desired error goal, respectively. The online version of the IDCA-II learning algorithm can be derived easily following the conceptual framework presented in Algorithm 5.2.

## 5.10 Illustrative Example

The performance of the proposed VSMNN model was evaluated by applying it to a function approximation problem. The function chosen to be approximated is the same as used for the evaluation of the LCNN model in Chapter 3 by Equation 3.9 and is given by the following relationship.

$$f(x) = 2e^{(-0.2x)}|sin(x)| \tag{5.11}$$

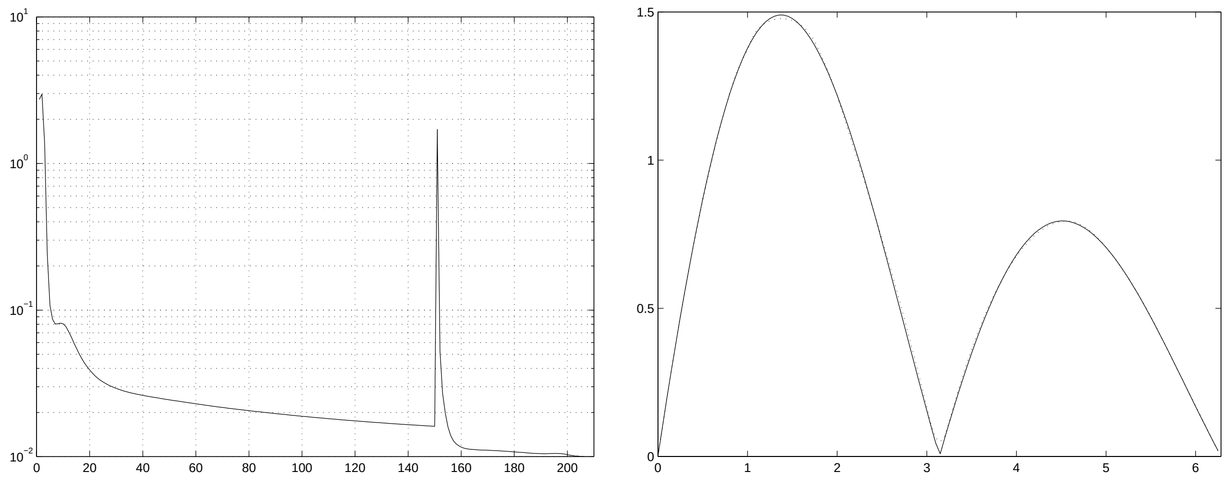
The initial VSMNN model was a LCNN model that had $4$ hidden layer neurons with associated inhibitory connection. The patience factor was set to $150$ iterations and the any subsequent hybrid

**Algorithm 5.2** Iterative Divide and Conquer Algorithm II

```
Requires: $T = \{\vec{x}_i, \vec{y}_i\}$ for $i = 1, \cdots, N$.  $\mu, \epsilon, m$
  • Initialize the neural network model with one hidden layer comprised of $m$ neurons with
    lateral inhibitory connections.
  $t \Leftarrow 0$
  $M \Leftarrow 1$
  while $e \geq \epsilon$ do
    if $t \leq \mu$ then
      $t \Leftarrow t + 1$
    else
      $t \Leftarrow 0$
      $M \Leftarrow M + 1$
      ◇ Freeze the existing batches of hidden layers neurons.
      ◇ Add a HNN model with $m$ hidden layer neurons to the existing VSMNN structure with
        incremented slope for squashing functions.
      ◇ Create input and output weights consisting of random values for the newly added HNN
        model.
    end if
    • Apply input $\vec{x}$ to the VSMNN model.
    • Compute outputs of the entire existing individual neural network modules in the VSMNN
      model.
    • Compute the overall output of the VSMNN model, $\vec{\hat{y}}_k$, using Equation 5.10.
    • Compute the residual RMSSE for the $M$th component module using Equation 5.8.
    if $M > 1$ then
      ◇ Adjust the input and the output weights of the newly added HNN module to minimize
        residual error objective function as given by Equation 5.8.
    else
      ◇ Update the lateral connection weights as described in Section 3.3.
      ◇ Update the input to hidden layer weights as described in Section 3.3.
    end if
    • Calculate overall MSSE $J$ using Equation 5.9.
    $e \Leftarrow J_k^T J_k$
  end while
```

neural network model to be added to the VSMNN model was forced to have $4$ hidden layer neurons as well.

The initial VSMNN model was trained using the iterative divide and conquer algorithm II (IDCA-II) that was outlined in the Algorithm 5.2. The training results are shown in Figure 5.7. Figure 5.7(a) shows the training error record of the IDCA-II learning algorithm and the function approximation results are plotted in Figure 5.7(b). The training error plot indicates that the VSMNN model was able to reach the same error limit in slightly fewer training iterations than was the case with the LCNN model.



(a) (b)

Figure 5.7: (a) Plot of sum squared training error for IDCA-II algorithm. (b) Function approximation by the HSMNN model that was trained using IDCA-II algorithm.

## 5.11 Conclusions

In this chapter two self-scaling neural network models were presented to overcome the problem of model selection. The proposed models are biologically motivated and the performance of the proposed self-scaling neural network models have been shown to be satisfactory.

A novel methodology was introduced to search for a correct and diverse expert neural network to be included in the HSMNN model is accomplished by collecting a set of candidate neural networks and selecting a neural network that best fits the performance index that measures both the accuracy of the network and the diversity among the candidate neural networks.

In this chapter, only a fixed value of $\lambda$ was used in the performance index. Adaptive settings of the value of $\lambda$ needs to be explored, that is, the value of $\lambda$ can change according to changing operating environments during the course of the training phase. Some ideas that relate to adapting the parameter $\lambda$ that have occurred to the author are described in the following. If the error of the HSMNN is decreasing, the value of $\lambda$ should not be changed. On the other hand, if the performance of the HSMNN model does not improve and overall diversity of the of the HSMNN model is decreasing, then value of $\lambda$ should be increased. Also, if the overall diversity of the HSMNN model is increasing, and still the overall performance of the HSMNN model is not improving, then the diversity seems to be overemphasized. In this case the value of the $\lambda$ may be decreased to improve overall performance of the HSMNN model.

# Chapter 6

# Conclusions

## 6.1 Summary

This dissertation has highlighted the potential of biologically motivated modularization of artificial neural network design and learning. The theoretical and experimental results of this dissertation provide evidence supporting the fact that modularization of neural network learning and design is an appropriate means of improving the learning performance, resulting in better generalization and improved performance.

The modularization approaches to the artificial neural network design allow automatic or explicit decomposition of a learning task into sub-tasks with localized performance evaluation criteria for each of the member components of a modular neural network.

In addition, the modularization approaches investigated in this dissertation provide a way of supporting supervised learning by means of the unsupervised learning that works on the principle of automatic feature discovery. In a way, modularization of artificial neural networks provides a means of seamlessly integrating the supervised and unsupervised learning paradigms.

Modularization of artificial neural networks minimizes the interactions between the weights of the component neural networks, thus, resulting in increased robustness and reliability. Also, the modularization approach helps in the modularization of the application task that can be automatic or knowledge-based. Knowledge-based determination of the structure of modular neural networks is another important feature of modularization by which it is possible to construct a neural network model that reflects the structural dependencies of the application domain.

## 6.2 Summary of Contributions

The aim of this dissertation was to develop new biologically motivated modular artificial neural network architectures and the associated learning algorithms. A review of the existing neural network models, both monolithic and modular, indicated a need for a fresh look at the functioning and the structural organization of the mammalian brain and other successful knowledge creating entities, in order to take steps toward design of more efficient and intuitive modular neural network models.

A laterally connected neural network model was presented in this dissertation that introduced the concept of modularity in a monolithic neural network by incorporating lateral inhibitory connections. The laterally connected neural network model was inspired by the findings in the field of neurobiology that indicates that the lateral inhibitory connections constitute an important part of the local cortical circuitry. The inference drawn from the local cortical circuitry leads to the conclusion that an effective and neurobiologically plausible neural network model should include an elaborate mechanism of embedded inhibitory connections. This hypothesis was proven to be true by the performance evaluation of the proposed laterally connected neural network model that was able to solve a class of difficult problems for which a solution by the mainstream feed-forward neural networks is not easily obtained.

A new modular neural network architecture was introduced that overcomes the shortcomings of the hierarchical mixture of experts architecture that is by far the most widely accepted modular neural network model. The new proposed modular neural network model is more in line with the concepts that are well understood by the neural network research community and has a very intuitive associated learning algorithm. The new modified hierarchical mixture of experts model draws its design and learning inspirations from different sources like top-down and bottom-up learning that is thought to be the basis of learning process in the brain, the model of successful knowledge creating companies and Bayesian neural networks. The proposed modified hierarchical mixture of experts model demonstrated a good performance when compared with the original hierarchical mixture of experts and its variants, but with a more simple and elegant learning mechanism.

Also, a new alternative radial basis function based gating network was introduced for the modified hierarchical mixture of experts model. The proposed alternate radial basis function neural network in itself is a novel neural network model. The proposed alternate radial basis function based gating network adds to the capabilities of the modified hierarchical mixture of experts model, allows a better initialization procedure, and introduces additional robustness to the modified hierarchical mixture of experts model.

The proposed self-scaling modular neural networks overcome the problem of model selection in a novel fashion. The self-scaling modular neural networks, as the name indicates, scale their architectures to solve a complex problem by adaptively scaling their respective structures to match the complexity of the problem at hand. The horizontally self-scaling modular neural network model uses a novel concept that is based upon embryo-genetic principles, to create new neural networks to extend its structure adaptively and provides a new original methodology to iteratively divide a complex problem into simpler problems to find an optimal solution. The vertically self-scaling modular neural networks iteratively make use of a new concept, the hybrid neural network, to solve complex problems.

## 6.3 Future Work

The performance of the proposed biologically inspired modular neural networks highlight the powerful potential of these approaches in the fields of classification and function approximation. However, there are still some aspects of these models that can be improved; some of these are outlined in the following as suggested future research work:

1. The use of different learning algorithms for the different specialist modules in the modified hierarchical mixture of experts architecture. One example would be in the case where a major portion of a task can be learned offline and due to some change in the operating environments, a continuous adaptation of the weights of a module is required to compensate for the change in the operating environment. Then, the offline training of the specialist modules can be carried out using efficient offline learning algorithms like Levenberg-Marquardt, whereas for the module for which a continuous adaptation is required, an online algorithm like standard backpropagation can be used.

2. Another topic for investigation is the use of modules with different topology and configuration to accomplish different decomposed sub-tasks. This approach can be especially advantageous in situations where the training data set might not be evenly distributed, so for the regions of the input space that are densely populated, specialist modules with more hidden layers and/or more hidden layer neurons may be used whereas in the regions where the training data is sparse, an expert neural network module with simpler topology might suffice to learn the desired mapping.

3. Another potential study is the use of more efficient combination/integration mechanism for expert neural network modules. For example, if a sound prior knowledge about the task at hand is available, then it might be appropriate to use a fuzzy logic based gating network to combine the partial representations of the overall task generated by the individual expert neural network modules that can eliminate the training time required to train the gating network.

4. The employment of different configurations should be considered for the different specialist modules in the modified hierarchical mixture of experts architecture. For example, some of the specialist modules could be fuzzy logic based systems that are well suited for that portion of the overall task, whereas the remaining expert modules could be artificial neural network based.

5. The modification of the existing learning algorithm for the laterally connected neural network model to incorporate much desired robustness against the change of sign of the lateral connections during the training phase of the neural network could be a valuable addition.

6. The introduction of the embryo-genetic principle could be made in the training phase to generate new efficient hybrid neural networks for the vertically self-scaling modular neural networks, as is already the case for the horizontally self-scaling modular neural networks.

7. It could be interesting to include the use of Bayesian weight regularization during the training phase of all of the presented neural network models. The Bayesian regularization minimizes a linear combination of squared errors and weights so that the resulting network has good generalization qualities.

# Appendix A

# Hierarchical Mixture of Experts Network Training Algorithms

The description of the training algorithms in this appendix follows closely the training algorithms as described in [166, 167]. Consider the hierarchical mixture of experts neural network as shown in Figure 2.1. The probabilities $g_j$ and $g_{j|i}$ are defined to be the prior probabilities, because they are computed, based only on the input $x^{(t)}$, without knowledge of the corresponding target output $y^{(t)}$. A posterior probability is defined once both the input and the target output are known. Using Bayes' rule, the posterior probabilities at the nodes of the tree are defined as as follows:

$$h_i = \frac{g_i \sum_j g_{j|i} P_{ij}(y)}{\sum_i g_i \sum_j g_{j|i} P_{ij}(y)} \tag{A.1}$$

and

$$h_{j|i} = \frac{g_{j|i} P_{ij}(y)}{\sum_j g_{j|i} P_{ij}(y)} \tag{A.2}$$

It is useful to define the joint posterior probability $h_{ij}$ as the product of $h_i$ and $h_{j|i}$:

$$h_{ij} = \frac{g_i g_{j|i} P_{ij}(y)}{\sum_i g_i \sum_j g_{j|i} P_{ij}(y)} \tag{A.3}$$

This quantity is the probability that expert network $(i, j)$ can be considered to have generated the data, based on knowledge of both the input and the output. All of these probabilities are conditional on the input at the $t^{th}$ time instant, $x^{(t)}$. In deeper trees, the posterior probability associated with an expert network is simply the product of the conditional posterior probabilities along the path from the root of the tree to that expert neural network.

## A.1 Gradient Descent Learning Algorithm

A gradient descent learning algorithm for the hierarchical mixture of experts architecture is presented in [166]. This algorithm is based on the earlier work described in [75] in which the problem of learning in mixture of experts architecture was treated as a maximum likelihood estimation problem.

Given the assumptions and definitions earlier in this Appendix and in Section 2.6, the total probability of generating $y^{(t)}$ from $x^{(t)}$ is the mixture of the probabilities of generating $y^{(t)}$ from each of the component densities, where the mixing proportions are multinomial probabilities, and can be expressed as

$$P(y^{(t)}|x^{(t)}, \boldsymbol{\theta}^0) = \sum_i g_i(x^{(t)}, \boldsymbol{V}_i^0) \sum_j g_{j|i}(x^{(t)}, \boldsymbol{V}_{ij}^0) P(y^{(t)}|x, \boldsymbol{\theta}_{ij}^0) \tag{A.4}$$

where $\boldsymbol{\theta}^0$ includes the expert neural network parameters $\boldsymbol{\theta}_{ij}^0$ as well the gating network parameters $\boldsymbol{V}_i^0$ and $\boldsymbol{V}_{ij}^0$. Note that the probabilities $g_i$ and $g_{j|i}$ are explicitly dependent on the input $x^{(t)}$ and the parameters.

The log likelihood of a data set $\mathcal{X} = \{(x^{(t)}, y^{(t)})\}_1^N$ is obtained by taking the log of the product of $N$ densities of the form of A.4, which yields the following log likelihood:

$$\boldsymbol{l}(\boldsymbol{\theta}; \boldsymbol{X}) = \sum_t \ln \sum_i g_i^{(t)} \sum_j g_{j|i}^{(t)} P_{ij}(y^{(t)}) \tag{A.5}$$

Let us assume that the probability density $P$ is Gaussian with an identity covariance matrix and that the link function is the identity. In this case, by differentiating $\boldsymbol{l}(\boldsymbol{\theta}; \boldsymbol{X})$ with respect to the parameters, the following gradient descent learning rule for the weight matrix $\boldsymbol{W}_{ij}$ is obtained:

$$\Delta \boldsymbol{W}_{ij} = \rho \sum_t h_i^{(t)} h_{j|i}^{(t)} (y^{(t)} - \mu^{(t)}) x^{(t)T} \tag{A.6}$$

where $\rho$ is the learning rate. The gradient descent learning rule for the $i^{th}$ weight vector in the top level gating network is given by:

$$\Delta \boldsymbol{V}_i = \rho \sum_t (h_i^{(t)} - g_i^{(t)}) x^{(t)} \tag{A.7}$$

and the gradient descent rule for the $j^{th}$ weight vector in the $i^{th}$ lower level gating network is given by:

$$\Delta \boldsymbol{V}_{ij} = \rho \sum_t h_i^{(t)} (h_{j|i}^{(t)} - g_{j|i}^{(t)}) x^{(t)} \tag{A.8}$$

Also, updates can be obtained for the covariance matrices as well [166].

The algorithm given by Equations A.6, A.7 and A.8 is a batch learning algorithm. The corresponding online algorithm can be obtained by simply dropping the summation sign and updating the parameters after each stimulus presentation. Thus, for example,

$$\boldsymbol{W}_{ij}^{(t+1)} = \boldsymbol{W}_{ij}^{(t)} + \rho_i^{(t)} h_{j|i}^{(t)} (y^{(t)} - \mu^{(t)}) x^{(t)T} \tag{A.9}$$

is the stochastic update rule for the weights in the $(i,j)^{th}$ expert network based on the $t^{th}$ stimulus pattern.

## A.2 The Expectation Maximization Algorithm

In this and the following section, development of a learning algorithm for the HME architecture is described which is based on the Expectation Maximization (EM) framework [85]. The EM algorithm for the HME architecture, which consists of an iterative solution of a coupled set of iteratively reweighted least squares problems, is discussed.

The EM algorithm is a general technique for maximum likelihood estimation. In practice EM has been applied almost exclusively to unsupervised learning problems. This is true of the neural network literature and machine learning literature, in which EM has appeared in the context of clustering [168] and density estimation [169], as well as the statistics literature, in which applications include missing data problems [170]. There is nothing in the EM framework that precludes its application to regression or classification problems; however, such applications have been few.

EM is an iterative approach to maximum likelihood estimation. Each iteration of an EM algorithm is composed of two steps: an Estimation (E) step and a Maximization (M) step. The M step involves the maximization of a likelihood function that is redefined at each iteration by the E step. If the algorithm simply increases the likelihood function during the M step, rather than maximizing the function, then the algorithm is referred to as a Generalized EM (GEM) algorithm. GEM algorithms are often significantly slower to converge than EM algorithms.

An application of EM generally begins with the observation that the optimization of the likelihood function $\mathcal{L}(\theta;\mathcal{X})$ would be simplified if only a set of additional variables, called "missing" or " hidden" variables, were known. In this context, the observable data $\mathcal{X}$ is referred to as the "incomplete data" and a "complete data" set $\mathcal{Y} = \{\mathcal{X},\mathcal{Z}\}$ is assumed that includes the missing variables $\mathcal{Z}$. Letting the probability model that links the missing variables to the actual data be $P(y,z|x,\theta)$. The logarithm of the density $P$ defines the "complete data likelihood", $\mathcal{L}_c(\theta;Y)$

and is given by

$$\mathcal{L}_c(\theta; Y) = \ln P(\mathcal{X}, \mathcal{Z}|\theta)$$

The original likelihood, $\mathcal{L}_c(\theta; X)$, is referred to in this context as the "incomplete data likelihood". It is the relationship between these two likelihood functions that motivates the EM algorithm. Note that the complete-data likelihood is a random variable, because the missing variables $Z$ are in fact unknown. An EM algorithm first finds the expected value of the complete-data likelihood, given the observed data and the current model. This is the E step:

$$Q(\theta, \theta^{(p)}) = E[\mathcal{L}_c(\theta; Y)|\mathcal{X}] \tag{A.10}$$

where $\theta^{(p)}$ is the value of the parameters at the $p^{th}$ iteration and the expectation is taken with respect to $\theta^{(p)}$. This step yields a deterministic function $Q$. The M step maximizes this function with respect to $\theta$ to find the new parameter estimates $\theta^{(p+1)}$:

$$\theta^{(p+1)} = \arg\max_{\theta} Q(\theta, \theta^{(p)}) \tag{A.11}$$

The E step is then repeated to yield an improved estimate of the complete likelihood and the process iterates.

An iterative step of EM chooses a parameter value that increases the value of $Q$, the expectation of the complete likelihood. It has been proven that an increase in expectation of the complete likelihood implies an increase in the incomplete likelihood [85].

$$\mathcal{L}(\theta^{(p+1)}; \mathcal{X}) \geq \mathcal{L}(\theta^{(p)}; \mathcal{X}) \tag{A.12}$$

Equality for Equation A.12 is obtained only at the stationary points of $\mathcal{L}$ [171]. Thus, the likelihood $\mathcal{L}$ increases monotonically along the sequence of parameter estimates generated by an EM algorithm. In practice this implies convergence to a local maximum.

## A.3 Applying EM to the HME Architecture

To develop an EM algorithm for the HME architecture, appropriate " missing data" has to be defined so as to simplify the likelihood function. Defining the indicator variables $z_i$ and $z_{j|i}$ to be equal to one, with only one of the $z_i$ is equal to one, and one and only one of the $z_{j|i}$ equal to one. These indicator variables have an interpretation as the labels that correspond to the decisions in the probability model. Also defining the indicator variable $z_{ij}$, which is the product of $z_i$ and $z_{j|i}$ and has an interpretation as the label that specifies the expert (the regressive process) in the probability model. If the labels $z_i$, $z_{j|i}$ and $z_{ij}$ were known, then the maximum likelihood problem would decouple into a separate set of regression problems for each expert network and a separate set of multi-way classification problems for the gating networks. These problems would be solved independently of each other, yielding a rapid one pass learning algorithm. Of course, the missing variables are not known, but we can specify a probability model that links them to the observable data. Their probability model can be written in terms of the $z_{ij}$ as follows:

$$\begin{aligned} P(y^{(t)}, z_{ij}^{(t)}|x^{(t)}, \theta) &= g_i^{(t)} g_{j|i}^{(t)} P_{ij}(y^{(t)}) &\quad (A.13)\\ &= \prod_i \prod_j \{g_i^{(t)} g_{j|i}^{(t)} P_{ij}(y^{(t)})^{z_{ij}^{(t)}}\} &\quad (A.14) \end{aligned}$$

using the fact that $z_{ij}^{(t)}$ is an indicator variable. Taking the logarithm of this probability model yields the following complete data likelihood:

$$\begin{aligned} \mathcal{L}_c(\theta; Y) &= \sum_t \sum_i \sum_j z_{ij}^{(t)} \ln\{g_i^{(t)} g_{j|i}^{(t)} P_{ij}(y^{(t)})\} &\quad (A.15)\\ &= \sum_t \sum_i \sum_j z_{ij}^{(t)} \{\ln g_i^{(t)} + \ln g_{j|i}^{(t)} + \ln P_{ij}(y^{(t)})\} &\quad (A.16) \end{aligned}$$

Note the relationship of the complete-data likelihood in Equation A.16 to the incomplete-data likelihood in Equation A.5. The use of the indicator variables $z_{ij}$ has allowed the logarithm to be brought inside the summation signs, substantially simplifying the maximization problem. We now define the E step of the EM algorithm by taking the expectation of the complete-data likelihood:

$$Q(\theta, \theta^{(p)}) = \sum_t \sum_i \sum_j h_{ij}^{(t)} \{\ln g_i^{(t)} + \ln g_{j|i}^{(t)} + \ln P_{ij}(y^{(t)})\} \quad (A.17)$$

where

$$\begin{aligned} E[z_{ij}^{(t)}|\mathcal{X}] &= P(z_{ij}^{(t)} = 1|y^{(t)}, x^{(t)}, \theta^{(p)}) &\quad (A.18)\\ &= \frac{P(y^{(t)}|z_{ij}^{(t)} = 1, x^{(t)}, \theta^{(p)}) P(z_{ij}^{(t)} = 1|x^{(t)}, \theta^{(p)})}{P(y^{(t)}|x^{(t)}, \theta^{(p)})} &\quad (A.19)\\ &= \frac{P(y^{(t)}|x^{(t)}, \theta_{ij}^{(p)}) g_i^{(t)} g_{j|i}^{(t)}}{\sum_i g_i^{(t)} \sum_j g_{j|i}^{(t)} P(y^{(t)}|x^{(t)}, \theta_{ij}^{(p)})} &\quad (A.20)\\ &= h_{ij}^{(t)} &\quad (A.21) \end{aligned}$$

(Note also that $E[z_i^{(t)}|\mathcal{X}] = h_i^{(t)}$ and $E[z_{j|i}^{(t)}|\mathcal{X}] = h_{j|i}^{(t)}$).

The M step requires maximizing $Q(\theta, \theta^{(p)})$ with respect to the expert network parameters and the gating network parameters. Examining Equation A.17, it can be deduced that the expert network parameters influence the $Q$ function only through the terms $h_{ij}^{(t)}\ln P_{ij}(y^{(t)})$ and the gating network parameters influence the $Q$ function only through the terms $h_{ij}^{(t)}\ln g_i^{(t)}$ and $h_{ij}^{(t)}\ln g_{j|i}(y^{(t)})$. Thus the M step reduces to the following separate maximization problems:

$$\theta_{ij}^{(p+1)} = \arg\max_{\theta_{ij}} \sum_t h_{ij}^{(t)} \ln P_{ij}(y^{(t)}) \tag{A.22}$$

$$V_i^{(p+1)} = \arg\max_{V_i} \sum_t \sum_k h_k^{(t)} \ln g_k^{(t)} \tag{A.23}$$

and

$$V_{ij}^{(p+1)} = \arg\max_{V_{ij}} \sum_t \sum_k h_k^{(t)} \sum_l h_{l|k}^{(t)} \ln g_{l|k}^{(t)} \tag{A.24}$$

Each of these maximization problems are themselves maximum likelihood problems. Equation A.22 is simply the general form of a weighted maximum likelihood problem in the probability density $P_{ij}$. Cross entropy maximization between posterior probabilities, $h_k^{(t)}$ and $g_k^{(t)}$ is carried out in Equation A.23. Equation A.24 solves a weighted maximum likelihood problem with output observations $h_{l|k}^{(t)}$ and observation weights $h_k^{(t)}$.

# Appendix B

# List of Publications

This chapter lists the publications by the author related to this dissertation.

1. F. Azam and H. F. VanLandingham. An efficient dynamic system identification technique using modular neural networks. *Artificial Neural Networks for Intelligent Engineering*, 7:225-230,1997.

2. F. Azam and H. F. VanLandingham. A modular neural network method for robust handwritten character recognition. *Artificial Neural Networks for Intelligent Engineering*, 8:503-508,1998.

3. F. Azam and H. F. VanLandingham. Generalized fuzzy adaptive control methodology. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, 3:2083-2088,1998.

4. F. Azam and H. F. VanLandingham. A new approach for modular neural network design and learning. *3rd International Conference on Computational Intelligence and Neurosciences*, 2:147-150,1998.

5. F. Azam and H. F. VanLandingham. Adaptive self-organizing feature map neuro-fuzzy technique for dynamic system identification. *Proceedings of the 1998 IEEE ISIC/CIRA/ISAS Joint conference*, 337-341,1998.

6. F. Azam and H. F. VanLandingham. Fuzzy adaptive genetic algorithms. *Advances in Systems, Signals, Control and Computers*, 2:231-235,1998.

7. F. Azam and H. F. VanLandingham. Dynamic system identification: a comparative study. *Proceedings of the 3rd Annual Genetic Programming Conference*, 2-5,1998.

8. F. Azam and H. F. VanLandingham. Dynamic system identification using genetic programming. *Proceedings of the 3rd Annual Genetic Programming Conference*, 1,1998.

9. H. F. VanLandingham and F. Azam. Soft computing applications in the electric power industry. *IEEE workshop on Soft Computing Methods in Industrial Applications SMCia/99*, 1-4,1999.

10. M. Abdallah, F. Azam and H. F. VanLandingham. A modular neural network approach to the speech recognition problem. *Artificial Neural Networks for Intelligent Engineering*, 9:1079-1083,1999.

11. F. Azam and H. F. VanLandingham. A laterally connected neural network model. *Accepted for publication in IASTED Neural Networks 2000 Conference*.2000.

12. F. Azam and H. F. VanLandingham. An evidence maximization framework for modular neural networks. *Accepted for publication in 4th World Multi-conference on Systems, Cybernetics and Informatics (SCI'2000)*.2000.

# Bibliography

[1] W. S. McCulloh and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[2] D. O. Hebb. *The Organization of of Behavior*. Wiley, New York, 1949.

[3] M. Minsky and S. Papert. *The Percptrons*. MIT Press, Cambridge, MA, 1968.

[4] J. J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of National Academy of Sciences*, 74:2554–2558, 1982.

[5] J.A. Feldman and D.H. Ballard. Connectionist models and their properties. *Cognitive Science*, 6(3), 1982.

[6] D. E. Rumelhert, G. E. Hinton, and R. Williams. Learning internal representation by error propagation. *Nature*, 323:533–536, 1986.

[7] B. Widrow and M. E. Hoff. Adaptive switching circuits. In *1960 IRE WESCON Convention Record*, pages 96–104, New York, 1960.

[8] R. Lippmann. Review of neural networks for speech recognition. *Neural Computation*, 1:1–38, 1989.

[9] Y LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1:541–551, 1989.

[10] C Peterson, S. Redfield, J. Keeler, and E. Hartmen. An optoelectronic architecture for multilayer learning in a single photorefrective crystal. *Neural Computation*, 2:25–34, 1990.

[11] D. Pomerleau. Alvinn:an autonomous land vehicle in an neural network. In D. Touretzky, editor, *Advances in Neural Information Processing systems 1*, pages 305–313, Denver, CO, 1989. Morgan Kaufmann.

[12] M. I. Jordan. Motor learning and the degrees of freedom problem. In M. Jeannerod, editor, *Attention and Performance XIII*, Hillsdale, NJ, 1990. Erlbaum.

[13] E. Gosh Collins and C. Scofield. An application of a multiple neural network learning system to emulation of mortgage underwriting judgments. Technical report, Nestor Inc., Providence, RI, 1989.

[14] K. S. Narendra and K. Parthasarathy. Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, 1:4–27, 1990.

[15] M. Smith. *Neural Networks for Statistical Modeling*. Van Nostrand Reinhold, NY, 1993.

[16] G. M. Shepherd. *The synaptic Organization of the Brain*. Oxford University Press, New York, 1974.

[17] F. W. Rexrodt. *Gehrin und Psyche*. Hippokrates, Stuttgart, Germany, 1981.

[18] J. W. Boers and H. Kupier. Biological metaphors and the design of modular artificial neural networks. Master's thesis, Leiden University, Leiden, August 1992.

[19] K. S. Lashley. *Brain Mechanisms and Intelligence*. University of Chicago Press, Chicago, 1929.

[20] D. H. Huble. *Eye, brian, and vision*. Scientific Ameircan Library, New York, 1988.

[21] V. B Montcastle. An organizing principle for cerebral function: The unit module and the distributed system. In G. M. Edelman and V. B Mountcatke, editors, *The mindful brain: Cortical organization and the group selective theory of higher brain function*, page 7. MIT Press, Cambridge, MA, 1978.

[22] J. C Eccles. The cerebral neocortex: Atheory of its operation. In E. G Jones and A. Peters, editors, *Cerebral Cortex: Functional Properties of Cortical Cells*, volume 2. Plenum Press, 1984.

[23] G. M. Edelman. *Neural Darwinism: Theory of Neural Group Selection*. Basic Books, 1987.

[24] G. M. Edelman. Group selection and phasic reentrant signaling: A theory of higher brain function. In F. O. Schmitt and F. G Worden, editors, *The neurosciences: Fourth study Program*. MIT Press, Cambridge, MA, 1979.

[25] R. S. J. Frackpwiak, K. J. Friston, C. D. Frith, R. J. Dolan, and J. C. Mazziotta. *Human Brain Function*. Academic Press, San Diego, 1997.

[26] H. A. Simson. *The Sciences of the Artificial*. MIT Press, Cambridge, MA, 1969.

[27] D. Marr. Early processing of visual information. *Philosophical transactions of the Royal Society of London, Series B*, 275:483–524, 1976.

[28] J. McClelland, B. McNaughton, and R. O'Reilly. Why there are complementary learning systems in the hippocampus and neocortex: Insights from the successes and failures of connectionist models of learning and memory. *Psychological Review*, 102:419–457, 1995.

[29] A. Cowey. Sensory and non-sensory disorders in man and monkey. *Philosophical transactions of the Royal Society of London, Series B*, 298:3–13, 1982.

[30] G. Tononi, O. Sporns, and G. Edelman. A measure for brain complexity: Relating functional segregation and integration in the nervous system. *Proceedings of the National Academy of Sciences*, 91:5033–5037, 1994.

[31] J. A. Fodor. *The Modularity of Mind*. MIT Press, Cambridge, MA, 1983.

[32] D. C. Van Essen, C. H. Anderson, and D. J. Fellman. Information processing in the primate visual system. *Science*, 255:419–423, 1992.

[33] J. H. Kaas. Why does the brain have so many visual areas? *Journal of Cognitive Neurosciences*, 1(2):121–135, 1989.

[34] T. Bossomaier and N. Snoad. Evolution and modularity in neural networks. In I. Pitas, editor, *Proc. IEEE Workshop on Non-Linear Signal and Image Processing*, pages 289–292, 1995.

[35] M. I. Jordan and R. A. Jacobs. A competitive modular connectionist architecture. In *Advances in Neural Information Processing Systems 3*, pages 767–773, San Maeto, CA, 1991. Morgan Kaufmann Publisher Inc.

[36] D. N. Osherson, S. Weinstein, and M. Stoli. Modular learning. *Computational Neuroscience*, pages 369–377, 1990.

[37] M. I. Jordan and R. A. Jacobs. Task decomposition through competition in a modular connectionist architecture: The what and where vision tasks. *Cognitive Science*, 15:219–250, 1991.

[38] Haykins Simon. *Neural Networks, A comprehensive Foundation*. Macmillan College Publishing Company, New York, NY, 1994.

[39] N. K. Perugini and W. E Engeler. Neural network learning time: Effects of network and training set size. *Proceedings of the International Joint conference on neural networks*, 2:395–401, 1989.

[40] H. Gomi and M. Kawato. Recognition of manipulated objects by motor learning with modular architecture networks. *Neural Networks*, 6:485–497, 1993.

[41] Farooq Azam and H. F. VanLandingham. A modular neural network method for robust handwritten character recognition. In *Artificial Neural Networks for Intelligent Engineering, ANNIE'98*, volume 8, pages 503–508, 1998.

[42] T. Lee. *Structure level adaptation for artificial neural networks.* Kluwer Academic Publishers, 1991.

[43] S. M. Kosslyn. *Image and Brain*. MIT Press, Massachusits, 1994.

[44] B. G. Stork, B. Jackson, and S. Walker. *Non-optimality via pre-adaptation in simple neural systems*, volume 3, pages 409–429. Addison-Wesley, Redwood City, CA, 1991.

[45] R. M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4):128–135, 1999.

[46] A. J. Sharkey. On combining artificial neural networks. *Connection Science*, 8(3 & 4):299–313, 1996.

[47] A. J. Sharkey. Modularity, combining and artificial nueral nets. *Connection Science*, 9(1):3–10, 1997.

[48] N. J. Nilsson. *Learning Machines: Foundations of Trainable Pattern-Classifying Systems.* McGraw Hill, New York, 1965.

[49] R. Clemen. Combining forecasts: A review and annotated bibliography. *International Journal of Forecasting*, 5:559–583, 1989.

[50] J. A. Barnett. Computational methods for a mathematical theory of evidence. In *Proceedings of IFCAI*, pages 868–875, 1981.

[51] C. W. J. Granger. Combining forecasts-twenty years later. *International Journal of Forecasting*, 8, 1989.

[52] R. E. Schapire. Strength of weak learners. *Machine Learnability*, 5:197–227, 1990.

[53] Y. Freund and R. Y. Schapire. Experiments with a new boosting algorithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148–156, 1996.

[54] R. A. Jacobs. Methods of combining experts' probability assessments. *Neural Computation*, 7:867–888, 1995.

[55] L. Xu, A. Krzyzak, and C. Y. Suen. Methods of combining multiple classifiers and their applications to handwriting recognition. *IEEE Transactions on Systems, Man and Cybernetics*, 22:418–435, 1992.

[56] S. Hashem and B. Schmeiser. Approximating a function and its derivatives using mse-optimal linear combination of trained neural networks. In *Proceedings of the World Congress on Neural Netwroks*, volume 1, pages 617–620, 1993.

[57] M. Perrone and L. N. Cooper. When networks disagree: Ensemble methods for hybrid neural networks. In *Neural Networks for Speech and Image Processing*. Chapman and Hall, London, 1993.

[58] L. K. Hansen and P. Salamon. Neural networks ensembles. *IEEE Transactions on Pattern Analysis ans Machine Intelligence*, 12:993–1000, 1990.

[59] K. Al-Ghoneim and V. Kumar. Learning ranks with neural networks. In *Applications and Science of Artificial Neural Networks, Proceedings of the SPIE*, volume 2492, pages 446–464, 1995.

[60] G Rogova. Combining results of several neural network classifiers. *Neural Networks*, 7:771–781, 1994.

[61] D. H. Wolpert. Stacked generalization. *Neural Networks*, 5:241–259, 1992.

[62] L. Breiman. Bagging predictors. *Machine Learning*, 24:123–140, 1996.

[63] E. Alpaydin. Multiple networks for function learning. In *International Conference on Neural Networks*, volume 1, pages 9–14, 1993.

[64] R. Battiti and A. Colla. Democracy in neural nets: Voting schemes for classification. *Neural Networks*, 7(4):691–707, 1994.

[65] G. Auda and M. Kamel. Modular neural networks classifiers: A comparative study. *Journal of Intelligent and Robotic Systems*, 21:117–129, 1998.

[66] G. Bartfei. Hierarchical clustering with art neural networks. In *World Congress on Computational Intelligence*, volume 2, pages 940–944, Florida, USA, 1994.

[67] M. De Bollivier, P. Gallinari, and S. Thiria. Cooperation of neural nets and task decomposition. In *International Joint conference on Neural Networks*, volume 2, pages 573–576, 1991.

[68] H. Tsai, H. Tai, and A. Reynolds. An art2-bp supervised neural net. In *World Congress on Neural Networks*, volume 3, pages 619–624, San Diego, USA, 1994.

[69] H. Raafat and M. Rashwan. A tree structured neural network. In *International Conference on Document Analysis and Recognition ICDAR93*, pages 939–942, 1993.

[70] E. Corwin, S. Greni, A. Logar, and K. Whitehead. A multi-stage neural network classifier. In *World Congress on Neural Networks*, volume 3, pages 198–203, San Diego, USA, 1994.

[71] G. Auda, M. Kamel, and H. Raafat. Modular neural network architecture for classification. In *IEEE International Conference on Neural Networks, ICNN'96*, volume 2, pages 1279–1284, Washington, DC, 1996.

[72] G. Auda, M. Kamel, and H. Raafat. Modular neural network architecture for classification. In *IEEE International Conference on Neural Networks, ICNN'95*, volume 3, pages 1240–1243, Perth, Australia, 1995.

[73] H. Hackbarth and J. Mantel. Modular connectionist structure for 100-word recognition. In *International Joint conference on Neural Networks*, volume 2, pages 845–849, 1991.

[74] A. Waibel. Modular construction of time-delay neural networks for speech recognition. *Neural Computation*, 1989:39–46, 1989.

[75] R.A. Jacobs, M.I. Jordan, S.J. Nowlan, and G.E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

[76] R.A. Jacobs, M.I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

[77] R.A. Jacobs, M.I. Jordan, and A. Barto. Task decomposition through competition in a modular coo nist architecture: the what and where vision tasks. *Neural Computation*, 3:79–87, 1991.

[78] S. Becker and G. E. Hinton. Learning mixture-models of spatial coherence. *Neural Computation*, 5(2):267–277, 1993.

[79] E. Alpaydin and M. I. Jordan. Local linear perceptrons for classification. *IEEE Transactions on Neural Networks*, 7(3):788–792, 1996.

[80] L. Bottou and V. Vapnik. Local learning algorithms. *Neural Computation*, 4(6):888–900, 1992.

[81] M.I. Jordan and L. Xu. Convergence results for the EM approach to mixtures of experts architectures. *Neural Networks*, 8(9):1409–1431, 1995.

[82] L. Xu and M. I. Jordan. On convergence properties of the EM algorithm for Gaussian mixtures. *Neural Computation*, 8(1):129–151, 1996.

[83] C. K. Tham. On-line learning using hierarchical mixtures of experts. In *IEE Conference on Artificial Neural Networks*, pages 347–351, 1995.

[84] M. I. Jordan and R. A. Jacobs. Hierarchal mixtures of experts and em algorithm. *Neural Computation*, 6:181–214, 1994.

[85] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39:1–38, June 1977.

[86] J. Fritsch, M. Finke, and A. Waibel. Adaptively growing hierarchical mixtures of experts. In *Advances in Neural Information Processing Systems 9*, pages 459–465, 1997.

[87] R. A. Jacobs and M. I. Jordan. A modular connectionist architecture for learning piecewise control strategies. *Proceedings of the American Control Conference*, 2.:1597–1602, 1991.

[88] A. Kehagias and V. Petridis. Predictive modular neural networks for time series classification. *Neural Networks*, 10(1):31–49, 1997.

[89] D. Miller and S. Uyar. A mixture of experts classifier with learning based on both labelled and unlabelled data. In *Advances in Neural Information Processing Systems 9*, pages 571–578, 1997.

[90] Farooq Azam and H. F. VanLandingham. An efficient dynamic systems identification technique using modular neural networks. In *Artificial Neural Networks for Intelligent Engineering, ANNIE'97*, volume 7, pages 503–508, 1997.

[91] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth and Brooks/Cole, Monterey, CA, 1984.

[92] J. H. Friedman. Multivariate adaptive regression splines. *Annals of Statistics*, 19(1):1–67, 1991.

[93] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[94] R. M. French. Semi-distributed representations and catastrophic forgetting in connectionistic networks. *Connection Science*, 4(3-4):365–377, 1992.

[95] R. M. French. Pseudo-recurrent connectionist networks: An approach to the sensitivity-stability dilemma. *Connection Science*, 9(4):353–379, 1997.

[96] N. E. Sharkey and A. J. Sharkey. An analysis of catastrophic interference. *Connection Science*, 7(3-4):301–329, 1995.

[97] H. White. *Artificial Neural Networks, Approximation and Learning Theory*. Blackwell Publishers, Cambridge, MA, 1992.

[98] M. McCloskey and N. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In G. H. Bower, editor, *The Psychology of Learning and Motivation*, volume 24, pages 109–164. Academic Press, New York, 1989.

[99] R. Ratcliff. Connectionist models of recognition memory: Constraints imposed by learning and forgetting functions. *Psychological Review*, 97:285–308, 1990.

[100] G.A. Carpenter and S. Grossberg. Art2: Self-organization of stable category recognition codes for analog input patterns. *Applied Optics*, 26:4919–4930, 1987.

[101] J. Kolen and J. Pollack. Backpropagation is sensitive to initial conditions. *Complex Systems*, 4:269–280, 1990.

[102] Y. Amir, M. Harel, and R. Malach. Cortical hierarchy reflected in the organization of intrinsic connections in macaque monkey visual cortex. *Journal of Comparative Neurobiology*, 334:19–46, 1993.

[103] R. Malach, Y. Amir, M. Harel, and A. Grinvald. Relationship between intrinsic connections and functional architecture,revealed by optical imaging and in vivo targeted biocytine injections in primate striate cortex. *Proceedings of the National Academy of Science USA*, 90:10469–10473, 1993.

[104] J Szentagothai. The module-concept in cerberal cortex architecture. *Brain Research*, 95:475–496, 1975.

[105] E. R. Kandal and J. H. Schwartz. *Priciples of Neural Science*. Elsevier, New York, 1985.

[106] J. D. Keeler. Comparision between kanerva's sdm and hopfield-type neural networks. *Cognitive Science*, 12:299–329, 1988.

[107] Scott E. Fahlman and Christian Lebiere. The Cascade-Correlation learning architecture. In *Advances in Neural Information Processing Systems 2*, pages 524–532, 1990.

[108] F Palmieri, C Catello, and G. D'Orio. Inhibitory synapses in neural networks with sigmoidal nonlinearities. *IEEE Transactions on Neural Networks*, 10(3):635–644, 1999.

[109] F Palmieri, J. Zhu, and J. Chang. Anti-hebbian learning in topologically constrained linear networks: A tutotrial. *IEEE Transactions of Neural Networks*, 4:748–761, 1993.

[110] A. Carlson. Anti-hebbian learning in nonlinear neural networks. *Biological Cybernetics*, 64:171–176, 1990.

[111] R. Kothari and K. Agyepong. On lateral connections in feed-forward neural networks. In *IEEE International Conference on Neural Networks Conference Proceedings*, volume 1, pages 13–18, 1996.

[112] H. Yin and N. M. Allinson. Bayesian learning for self-organising maps. *Electronics Letters*, 33(4):304–305, 1997.

[113] M. E. Salgado, G. C. Goodwin, and R. H. Middleton. Modified least squares algorithm incorporating exponential resetting and forgetting. *International Journal of Control*, 47:477–491, 1988.

[114] D. Elliott. A better activation function for artificial neural networks. Technical Report ISR TR 93-8, Institute for Systems Research, University of Maryland, College Park, MD, June 1993.

[115] J. Kivinen and M. K. Warmuth. Exponentiated gradient versus gradient descent for linear predictors. *Information and Computation*, 132(1):1–63, 1997.

[116] B. N. Cesa. Analysis of two gradient-based algorithms for on-line regression. *JournaL of Computer and System Sciences*, 59(3):392–411, 1999.

[117] D. Precup and R. S. Sutton. Exponentiated gradient methods for reinforcement learning. In *Proceedings of the 14th International Conference on Machine Learning*, pages 272–277. Morgan Kaufmann, 1997.

[118] A. Waibel, H. Sawai, and K. Shikano. Modularity and scaling in large phonemic neural networks. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 37(12):1888–97, December 1989.

[119] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2:14–23, 1986.

[120] F. Azam and H. VanLandingham. A new approach for modular neural network design and learning. In *3rd Internatioanl Conference on Computational Intelligence and Neurosciences*, pages 147–150, 1998.

[121] A. M. Wlaker. On the asymptotic behavior of posterior distributions. *Journal of the Royal Statistical Society B*, 31:80–88, 1969.

[122] A. S. Weigend, M. Mangeas, and A. N. Srivastava. Nonlinear gated experts for time series: discovering regimes and avioding overfitting. *International Journal of Neural Systems*, 6:373–399, 1995.

[123] Lei Xu, Geoff Hinton, and Michael I.Jordan. An alternative model for mixtures of experts. In *Advances in Neural Information Processing Systems 7*, pages 633–640, 1995.

[124] S. Geva and J. Sitte. A constructive method for multivariate function approximation by multilayer perceptrons. *IEEE Transactions on Neural Networks*, 3(4), 1992.

[125] C. C. Lee, P. C. Chung, J. R. Tsai, and C. I. Chang. Robust radial basis function neural networks. *IEEE Transactions on Systems Man and Cybernetics Part B-Cybernetics*, 29(6):674–685, 1999.

[126] I. Nonaka. The dynamic theory of organizational knowledge creation. *Organization Science*, 5(1):14–37, 1994.

[127] I. Nonaka and H. Takeuchi. *The Knowledge-Creating Company*. Oxford University Press, New York, NY, 1995.

[128] T. Bayes. An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of Royal Society of London*, 53:370–418, 1783.

[129] H. Jeffreys. *Theory of Probability*. Oxford Univerosty Press, London, 1939.

[130] R. T. Cox. Probability, frequency, and reasonable expectation. *American Journal of Physics*, 14:1–13, 1946.

[131] F. V. Jensen. *An introduction to Bayesian networks*. Springer-Verlag, New York, 1996.

[132] K. S. Narendra and K. Parthasarathy. Identification and control of dynamic systems using neural networks. *IEEE Transactions on Neural Networks*, 1(1):4–27, 1992.

[133] J. C. Bezdek and K. P. Sankar, editors. *Fuzzy models for pattern recognition : methods that search for structures in data*. Institute of Electrical and Electronics Engineers, New York, 1992.

[134] M. T. Hagan and M. B. Menhaj. Training feedforward networks with the marquardt algorithm. *IEEE Transactions on Neural Networks*, 5(6):989–993, 1994.

[135] C.L. Blake and C.J. Merz. UCI repository of machine learning databases, 1998.

[136] J. L. Davis, R. W. Newburg, and E. J. Wegman, editors. *Brain struture, learning, and Memory*. Westview Press, Inc., Boulder, CO, 1988.

[137] E. T. Rolls and A. Treves. *Neural Networks and Brain Function*. Oxford University Press, Inc., New York, New York, 1998.

[138] E. R. Kandel, J. H. Schwartz, and T. H. Jessel, editors. *Principles of Neural Science*. Elsevier, Amsterdam, Holland, 3rd edition, 1991.

[139] F. Richard Thompson. *The Brain, A neuroscience primer*. W. H. Freeman and Company, New York, 1993.

[140] R. Reed. Pruning algorithms - a survey. *IEEE Transactions on Neural Networks*, 4(5):740–747, 1993.

[141] Tin-Yau Kwok and Dit-Yan Yeung. Constructive algorithms for structure learning in feed-forward neural networks for regression problems. *IEEE Transactions on Neural Networks*, 8(3):630–645, 1997.

[142] Enno Littmann and Helge Ritter. Cascade network architectures. In *Proceedings of the International Joint Conference on Neural Networks, vol. 2*, pages 398–404, Baltimore, 1992.

[143] Natalio Simon, Henk Corporaal, and Eugene Kerckhoffs. Variations on the Cascade-Correlation learning architecture for fast convergence in robot control. In *Proc. Neuro-Nimes*, pages 455–464, Nimes, France, November 1992. EC2.

[144] A. G. Ivakhnenko. The group method of data handling - a rival of the method of stochastic approximation. *Soviet Automatic Control*, 13(3):43–55, 1968.

[145] S. J. Farlow, editor. *Self-Organizing Methods in Modeling : GMDH Type Algorithms*, volume 54. Marcel Dekker, Inc., New York, New York, 1984.

[146] M. F. Tenorio and T. W. Lee. Self-organizing network for optimum supervised learning. *IEEE Transactions on Neural Networks*, 1(1):100–110, 1990.

[147] John Moody. Prediction risk and architecture selection for neural networks. In V. Cherkassky, J.H. Friedman, and H. Wechsler, editors, *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*. Springer, NATO ASI Series F, 1994.

[148] Eric B. Bartlett. Dynamic node architecture learning: An information theoretic approach. *Neural Networks*, 7(1):129–140, 1994.

[149] Yoshito Hirose, Koichi Yamashita, and Shimpei Hijiya. Back-propagation algorithm which varies the number of hidden units. *Neural Networks*, 4(1):61–66, 1991.

[150] Timur Ash. Dynamic node creation in backpropagation networks. *Connection Science*, 1(4):365–375, 1989.

[151] J. H. Friedman and W. Stuetzle. Projection pursuit regression. *Journal of the American Statistical Association*, 76(376):817–823, 1981.

[152] S. R. Waterhouse and A. J. Robinson. Constructive algorithms for hierarchical mixtures of experts. In *Advances Inn Eural Information Processing Systems 8*, pages 584–590, 1996.

[153] K. Saito and R. Nakano. A constructive learning algorithm for an HME. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1268–1273, 1996.

[154] K. Chen, L.P. Yang, X. Yu, and H.S. Chi. A self-generating modular neural network architecture for supervised learning. *Neurocomputing*, 16(1):33–48, 1997.

[155] V. Ramamurti and J. Ghosh. Structurally adaptive modular networks for nonstationary environments. *IEEE Transactions on Neural Networks,* 10(1):152–160, 1999.

[156] J. L. Alba, L. Docio, D. Docampo, and O. W. Marquez. Growing gaussian mixtures network for classification applications. *Signal Processing,* 76(1):43–60, 1999.

[157] J. Reason. *Human error*. Cambridge University Press, New York, 1990.

[158] M. DesJardins. Goal-directed learning: A decision-theoretic model for deciding what to learn next. In *Goal-Driven Learning,* Cambridge, Massachusetts, 1995. MIT Press.

[159] K. Lang and M. Witbrock. Learning to tell two spirals apart. *Proceedings of the connectionist Models summer School,* pages 52–59, 1988.

[160] M. Lehtokangas. Modeling with constructive backpropagation. *Neural Networks*, 12(4):707–716, 1999.

[161] M. Lehtokangas. Constructive backpropagation learning algorithm. *Neural Network World*, 8(4):387–400, 1998.

[162] D. Phatak and I. Koren. Connectivity and performance tradeoffs in the cascade-correlationn learning architecture. *IEEE Transactions on Neural Networks,* 5(6):930–935, 1994.

[163] G. Drago and S. Ridella. Cascade-correlation: an incremental tool for function approximation. In *Proceedings of the International Conference on Artificial Neural Networks*, pages 750–754, 1993.

[164] S. Sjogaard. Generalization in cascade-correlation networks. In *Proceedings of the IEEE Signal Processing Workshop,* pages 59–68, 1992.

[165] N. Simon, H. Corporaal, and E. Kerckhoffs. Variants on the cascade-correlation learning architecture for fast convergence in robot control. In *Proceedings of the Fifth International Conference on Neural Networks and their Application*, pages 455–464, 1992.

[166] M. I. Jordan and R. A. Jacobs. Hierarchies of adaptive experts. In *Advances in Neural Information Processing Systems 4,* pages 985–992. Morgan Kaufmann, San Mateo, CA, 1992.

[167] M.I. Jordan and R.A. Jacobs. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6(2):181–214, 1994.

[168] P. Cheeseman, J. Stutz, J. Taylor, M. Self, and J. Kelley. Autoclass: A bayesian classification system. *Proceedings of the 5th International Conference on Machine Learning*, 1988.

[169] D. F. Specht. A general regression neural network. *IEEE Transactions on Neural Networks*, 2:568–576, 1991.

[170] R. J. A. Little and D. B. Rubin. *Statistical Analysis with Missing Data*. John Willey, New York, NY, 1987.

[171] C. F. J. Wu. On the convergence properties of the EM algorithm. *Annals of Statistics*, 11(1):95–103, 1983.

# Vita

Farooq Azam was born in Gujranwala City, the fourth largest city in Pakistan. He showed a special aptitude for the field of engineering at very young age, and decided to pursue an engineering career. He received his Bachelor of Engineering degree, with distinction, with specialization in Aviation Electronics (Avionics) in 1985 from the College of Aeronautical Engineering, Karachi. He stood first in his graduating class and was honored with two gold medals. One for being the best Avionics engineer of his class and the second one for being the best overall engineering student among the students who graduated in different engineering disciplines in that year. After finishing his bachelors degree, he worked for seven years as a research and evaluation engineer for the research and development wing of the Pakistan Air Force. In 1992, he came to the United States of America to pursue higher studies. He finished his Masters of Science in Electrical Engineering from University Of Southern California in 1993. He transferred to Virginia Tech to pursue his Ph.D. studies at Virginia Tech. He received 6 years of support from the Bradley Department of Electrical engineering at Virginia Tech as departmental workstation laboratory manager. His extracurricular interests include playing squash, listening to music and reading about philosophy. He has served as the president of Pakistani Students Association at Virginia Tech for past three years.