Dynamic configuration and approximation capabilities of extreme learning machines

Zhang, Rui

2012

Zhang, R. (2012). Dynamic configuration and approximation capabilities of extreme learning machines. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/50729>

<https://doi.org/10.32657/10356/50729>

Dynamic Configuration and Approximation Capabilities of Extreme Learning Machines

ZHANG RUI

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University

in fulfillment of the requirements for the degree of

Doctor of Philosophy

2012

Statement of Originality

I hereby certify that the content of this thesis is the result of work done by me and has not been submitted for a higher degree to any other University or Institution.

.....

Date

.....

ZHANG RUI

Acknowledgments

I would like to express my gratitude to all those who helped me during the writing of this thesis.

First of all, I would like to acknowledge my supervisor, Assoc. Prof. Huang Guangbin, for his expert guidance and constant encouragement. He has walked me through all the stages of the writing of this thesis. Without his consistent and illuminating instruction, this thesis could not have reached its present form. I am deeply grateful of all the efforts he has made to help me complete the thesis.

Second, I would like to express my heartfelt gratitude to Dr. Lan Yuan for helping me in the simulations in one way or another.

My special thanks also go to the technical staff, Mr. Tan Peng Chye from Machine Learning Research Laboratory, in the School of Electrical and Electronic Engineering for his technical support and logistic assistance.

Furthermore, I owe my sincere gratitude to my friends for many fruitful discussions and generous helps.

Last but not least, my thanks would go to my parents, my husband and my daughter for their loving considerations and great confidence in me all through these years.

Table of Contents

Acknowledgments	i
Summary	v
List of Figures	viii
List of Tables	x
1 Introduction	1
1.1 Research Background	1
1.2 Motivation and Objectives	3
1.3 Major Contributions	6
1.4 Organization	7
2 Brief of Fundamental Extreme Learning Machines	9
2.1 Overview of Neural Networks	11
2.1.1 M-P Model of a neuron	11
2.1.2 Network connection	12
2.1.3 Learning process	14
2.2 Generalized Single-hidden Layer Feedforward Networks (GSLFNs) . .	15

2.2.1	Architecture and mathematical formulation of GSLFNs	16
2.2.2	Approximation capabilities of GSLFNs	18
2.3	Extreme Learning Machines (ELMs)	20
2.3.1	Mathematical preliminaries	20
2.3.2	Brief on basic ELM	22
2.3.3	ELMs with different learning schemes	26
2.3.4	Variants of ELMs	28
2.4	Summary	34
3	Review of Extreme Learning Machines with Adjustable Architectures	35
3.1	Architecture Design Methods for Neural Networks	36
3.1.1	Pruning algorithms	36
3.1.2	Constructive algorithms	37
3.1.3	Comparison between pruning algorithms and constructive algorithms	38
3.2	Prunning ELMs	40
3.3	Constructive ELMs	41
3.3.1	Incremental ELMs (I-ELMs)	41
3.3.2	Error minimized ELM (EM-ELM)	48
3.4	Other Architecture-Adjustable ELMs	50
3.5	Summary	52
4	Incremental Extreme Learning Machine with Optimal Incremental Coefficients (OCI-ELM)	53
4.1	The Proposed OCI-ELM and Its Approximation Capability	54

4.2	Experimental Verification	59
4.3	Summary	62
5	Extreme Learning Machine with Adaptive Growth of Hidden Nodes (AG-ELM)	64
5.1	Preliminary Mathematical Aspects	65
5.2	The Proposed AG-ELM and Its Approximation Capability	74
5.3	Experimental Verification	78
5.4	Summary	80
6	Dynamic Extreme Learning Machine (D-ELM)	82
6.1	The Proposed D-ELM and Its Approximation Capability	83
6.2	Experimental Verification	89
6.2.1	Comparison between D-ELM and other constructive ELMs with the same expected accuracy	91
6.2.2	Comparison between D-ELM and other constructive ELMs with the same step	92
6.3	Summary	93
7	Conclusions and Future Work	102
7.1	Conclusions	102
7.2	Future work	103
	Author's Publications	106
	Bibliography	108

Summary

Computational intelligence techniques have been extensively explored in wide applications in the past three decades. Out of numerous computational intelligence techniques, neural networks have been playing the dominant roles. However, it is known that neural networks usually face some challenging issues such as local minima, slow learning rate, and trivial human intervene.

Extreme Learning Machines (ELMs) as an emergent technology, which overcome some challenges faced by other techniques, study a much wider type of “generalized” single-hidden layer feedforward networks (GSLFNs). The essence of ELMs is that (1) the hidden layer of ELMs need not be iteratively tuned so that the training time required is dramatically reduced and the human intervene is eliminated; (2) the hidden layer feature mapping need to satisfy the universal approximation condition so that the approximation capability of ELMs can be guaranteed; (3) both of the training error and the norm of output weights need to be minimized so that better generalization performance can be obtained. Although a plenty of simulation results have revealed that the performance of ELMs are relatively stable in a wide range of the number of hidden nodes, it may be hard to tell what the exact and proper range of the number of hidden nodes should be for a specific task. In addition, based on achieving the equal performance, it is apparent that the less the number of hidden nodes is, the more efficient and less computationally expensive ELMs will be, and hence the convergence rate of ELMs raises. Therefore, finding an appropriate and parsimonious network which can be best suited to the given problem, and further exploring its corresponding approximation capability, are still the important issues in the study of ELMs.

The original incremental extreme learning machine (I-ELM) was proposed where the hidden nodes are randomly added one by one and the output weights of the existing hidden nodes are frozen when a new hidden node is added. Convex incremental extreme learning machine (CI-ELM) adopts another incremental method and allows to properly adjust the output weights of the existing hidden nodes when a new hidden node is added. CI-ELM can achieve faster convergence rate than I-ELM while retaining the simplicity of I-ELM. However, such convex combination of the existing hidden nodes and the newly added hidden node may not be the optimal fashion by which the residual error can be reduced largest. On the basis of this observation, an incremental extreme learning machine with optimal incremental coefficients (OCI-ELM) has been proposed in this thesis. Such OCI-ELM can not only provide an optimal tradeoff between the existing hidden nodes and the newly added hidden nodes but also work as an universal approximator.

Although different architecture-adjustable ELMs by using either pruning method or constructive method are engaged in addressing the problem of network acquisition, they investigate only restricted topological subsets rather than the complete class of network architectures. Inspired by this cognition, an extreme learning machine with adaptive growth of hidden nodes (AG-ELM) where a novel approach to determining the network architecture in an adaptive manner has been introduced in this thesis. AG-ELM realizes the automatic design of networks in a search space of possible structures which are suitable to the problem. Furthermore, the obtained universal approximation theory of AG-ELM not only strengthens those existing results, but also provides a fundamental theory for studying the approximation capability of algorithms which use random mechanism for the parameter searching in neural networks.

With the exception of I-ELMs, EM-ELM is another efficient algorithm that allows to add random hidden nodes one by one or group by group (with varying size) and incrementally updates the output weights during the network growth. However, in the implementation of both I-ELMs and EM-ELM, the number of the hidden nodes always monotonically increases with the learning progress and the final number of the hidden

nodes is equivalent to the learning steps. Then large number of hidden nodes will be obtained eventually if there needs many iterative steps, while some of the hidden nodes may play a very minor role in the network output. On the other hand, although AG-ELM can automatically determine the network architecture, it is not so efficient as expected. Therefore, in order to choose the right size network automatically as well as improve the efficiency of ELMs, a dynamic extreme learning machine (D-ELM) performing dynamic growth of hidden nodes and incrementally updating of output weights has been proposed in this thesis. On the basis of the theory obtained in AG-ELM, it has been proved that the obtained D-ELM can approximate any Lebesgue p -integrable function as long as the hidden activation is Lebesgue p -integrable.

List of Figures

2.1	Model of a neuron	12
2.2	Feedforward neural network (FNN)	13
2.3	Recurrent neural network (RNN)	14
2.4	Generalized single-hidden layer feedforward network (GSLFN)	16
4.1	The testing RMSE updating curves of OCI-ELM, CI-ELM, EI-ELM and I-ELM for Abalone case	60
4.2	The updating curves of α_n and β_n in OCI-ELM	61
5.1	The procedure of obtaining the network sequence at the s th step of AG-ELM .	76
6.1	The testing RMSE updating curves of D-ELM and EM-ELM for Abalone case	93
6.2	The hidden-node number updating progress in 100 steps for Abalone case	94

List of Tables

4.1	Specification of benchmark datasets	59
4.2	Performance comparison of OCI-ELM, CI-ELM, EI-ELM and I-ELM with the same number of hidden nodes	63
5.1	Specification of benchmark datasets	79
5.2	Generalization performance comparison of AG-ELM and I-ELM with the same expected accuracy	80
6.1	Specification of Benchmark Datasets	90
6.2	Generalization Performance and Complexity of SVM	92
6.3	Network Complexity Comparison of of D-ELM, EM-ELM, EI-ELM and I-ELM (with sigmoid nodes) given the same expected accuracy . . .	96
6.4	Network Complexity Comparison of of D-ELM, EM-ELM, EI-ELM and I-ELM (with RBF nodes) given the same expected accuracy	97
6.5	Generalization Performance Comparison of D-ELM, EM-ELM, EI-ELM and I-ELM (with sigmoid nodes) given the same expected accuracy. The close generalization performance (testing RMSE) obtained by different algorithms is underlined and the best results among four algorithms are shown in boldface	98

6.6	Generalization Performance Comparison of D-ELM, EM-ELM, EI-ELM and I-ELM (with RBF nodes) given the same expected accuracy. The close generalization performance (testing RMSE) obtained by different algorithms is underlined and the best results among four algorithms are shown in boldface	99
6.7	Generalization Performance and Network Complexity Comparison of D-ELM, EM-ELM, EI-ELM and I-ELM (with sigmoid nodes) given the same expected step. The close results obtained by different algorithms is underlined and the best results among four algorithms are shown in boldface	100
6.8	Generalization Performance and Network Complexity Comparison of D-ELM, EM-ELM, EI-ELM and I-ELM (with RBF nodes) given the same expected step. The close results obtained by different algorithms is underlined and the best results among four algorithms are shown in boldface	101

Chapter 1

Introduction

1.1 Research Background

Artificial neural networks (ANNs), commonly referred to as “neural networks”, are one of the most powerful learning machines that are designed to model the way in which the brain performs a particular task. A neural networks employs a massive interconnection of simple computing cells referred to as “neurons” to carry out powerful computation through a process of learning. Due to these two information-processing capabilities, i.e., the massively parallel distributed structure and the ability to learn, a neural network is capable of solving complex and intractable problems. Therefore, research on neural networks mainly focuses on two aspects: construction of the network topology and design of the learning process.

Feedforward neural networks (FNNs), characterized as the layered neural networks, are one important class of neural networks which have been successfully applied in function approximation, pattern recognition, and clustering etc.

It is well known that FNNs with hidden layer are much powerful than FNNs without hidden layer. For example, an FNN with one sigmoidal hidden layer and linear output

nodes can approximate any continuous function, but an FNN without hidden layer does nothing to achieve it. Hence FNNs with hidden layer have always been the main object of study in various scientific research and applications. In FNNs, the number of input nodes and the number of output nodes can be determined according to the training patterns directly. Specifically, the number of input nodes corresponds to the dimensions of each input data, while the number of output nodes meets with the dimensions of each output. Therefore, two questions have emerged with regard to the construction of feedforward network topology: (1) how many hidden layers should be involved in the network? and (2) how many hidden nodes should be involved in each hidden layer? In the current research, most FNNs are designed with no more than two to three hidden layers.

The procedure to perform the learning process is called learning algorithm. A learning algorithm is a set of well-defined rules by which the synaptic weights of the network are modified in an orderly fashion so that a desired objective can be attained. One of the most highly popular algorithms in FNNs is known as error back-propagation (BP) algorithm. It is a first-order approximation of the steepest-descent technique in the sense that it depends on the gradient of the instantaneous error surface in weight space, and it is therefore stochastic in nature and suffers from a slow convergence property. Another peculiarity of the error surface that impacts the performance of BP algorithm is the presence of local minima in addition to global minima. Based on the above cognition, how to design efficient learning algorithms becomes another important issue in the study of FNNs.

Single-hidden layer feedforward networks (SLFNs) is a special case of the FNNs which only consist of one hidden layer. The capabilities of SLFNs to approximate complex nonlinear mappings directly from the input samples have been widely investigated [7, 32, 36, 82]. Hornik [45] proved that an SLFN can approximate any continuous function as long as the hidden activation function is nonconstant, bounded, and monotone-increasing continuous. This result is reputed as the *Universal Approximation Theorem*. Following from this theorem, lots of results on the approximation capability of SLFNs

have been shown in [2, 4, 44, 47, 70, 76, 89, 103]. Due to their universal approximation capabilities, SLFNs have been widely investigated and the accompanying problems can be then specified as: (1) how to determine the optimal number of hidden nodes; and (2) how to design the more efficient learning algorithms?

Unlike the conventional neural network theories where the parameters of SLFNs are well adjusted, Extreme Learning Machines (ELMs) is a collection of fresh learning algorithms which study a much wider type of “generalized” SLFNs (GSLFNs) where much wider type of formulations are available for the hidden nodes. As an emergent technology, ELMs have shown good performance in regression and classification applications with fast running speed and better generalization. This thesis will further study the dynamic configuration and approximation capabilities of ELMs.

1.2 Motivation and Objectives

Huang et al. [23, 49, 50, 51, 52, 53, 54, 55, 56, 59, 60, 61, 62, 63, 72, 73, 77, 78, 79, 90, 106, 110, 111, 134, 135] have recently proposed a series of learning algorithms, called extreme learning machines (ELMs), which were originally developed for neuron based SLFNs [51, 61, 62] and then extended to the GSLFNs [49, 50, 53, 59, 60, 134, 135]. Compared with those traditional computational intelligence techniques, ELMs provide better generalization performance at a much faster learning speed and with least human intervene.

Despite many advances, to explore the general applicability of ELMs in real-world problems, several questions must still be answered. One such open question involves determining the most appropriate network architecture for solving a specific task. ELMs with the batch learning scheme ([55, 56, 61, 62, 63, 78]) and with the online sequential learning scheme ([54, 72, 79, 110]) have been developed in an *a priori* fixed network size. A common approach to searching such fixed network size is by trial and error. This approach, although straightforward, not only is computationally expensive but also

does not guarantee that the selected network size will be close to optimal or will generalize well. From another point of view, research on a class of ELMs with adjustable architectures ([23, 49, 50, 51, 73, 74, 77, 90, 106, 111]) represents an effort toward the automatic design of network architectures.

In [51], an incremental extreme learning machine (I-ELM) was proposed where the hidden nodes are randomly added one by one and the output weights of the existing hidden nodes are freezed when a new hidden node is added. The universal approximation capability of I-ELM has also been proved in the same paper. Convex incremental extreme learning machine (CI-ELM) [49] adopts another incremental method originated from the Barron's convex optimization concept [2]. This new approach allows to properly adjust the output weights of the existing hidden nodes when a new hidden node is added. Under this situation, CI-ELM can achieve faster convergence rate than I-ELM while retaining the simplicity of I-ELM. During the learning procedure of both I-ELM and CI-ELM, some newly added hidden nodes only play a very minor role in the final output of the network, and hence, the network complexity may be increased. To avoid this issue and obtain a more compact network architecture, the enhanced incremental extreme learning machine (EI-ELM) was proposed in [50]. At each learning step of EI-ELM, several hidden nodes are randomly generated and only the one leading to the largest residual error reduction will be added to the existing network. Furthermore, EI-ELM also extends I-ELM to GSLFNs. Recently, Feng et al. [23] provided a new approach referred to as error minimized extreme learning machine (EM-ELM). EM-ELM allows to add random hidden nodes one by one or group by group (with varying size). Moreover, EM-ELM incrementally updates the output weights during the network growth, which significantly reduces the computational complexity. Different from the above mentioned ELMs which add the hidden nodes incrementally, Rong et al. [111] proposed a pruned extreme learning machine (P-ELM) which starts from an initial large number of hidden nodes and then removes those ones with no relevance or low relevance to the class labels by using statistical tests as a measure. Miche et al. [90] further developed an optimally-pruned extreme learning machine (OP-ELM) which uses

a leave-one-out criterion for selecting the optimal number of hidden nodes and applies to both classification and regression problems.

Although different architecture-adjustable ELMs by using either pruning method or constructive method are engaged in addressing the problem of network acquisition, they investigate only restricted topological subsets rather than the complete class of network architectures. Moreover, in the implementation of the constructive ELMs, the number of the hidden nodes monotonically increases with the learning progress and the final number of the hidden nodes is equivalent to the learning steps. Then large number of hidden nodes will be obtained eventually if there needs many iterative steps, while some of the hidden nodes may be of no account in the network output. On the other hand, the pruning ELMs are more computationally expensive because the majority of the training time is spent on networks larger than necessary, and it is difficult to specify an initial network because one does not know in practice how big the initial network should be. In addition, the pruning ELMs usually measure the change in error when a hidden node is removed. However, such changes can only be approximated for computational efficiency, and hence may introduce large errors, especially when many nodes are to be pruned. Therefore, how to choose the optimal number of hidden nodes while retaining the efficiency and the good performance of ELMs is one objective of this thesis.

With the selected appropriate network architectures, another main problem focuses on studying the approximation capabilities of the networks. Many research on the approximation capabilities of neural networks has been developed in past two decades. In the case where the parameters are well adjusted, it has been shown that any continuous function can be approximated by an SLFN as long as the hidden activation function is suitably chosen. For constructive ELMs [23, 49, 50, 51] where all the hidden node parameters are randomly generated and the output weights are analytically determined, it has been proved that such constructive ELMs can converge to any continuous function as long as the hidden activation functions are nonlinear piecewise continuous. However, the capabilities of some newly proposed ELMs which can obtain more compact network architectures have not been discussed. Therefore, how to further analyze in theory the

universal approximation capability of ELMs is another objective of this thesis.

1.3 Major Contributions

The research progress and corresponding contributions in this thesis are summarized as follows.

In the first work, an incremental extreme learning machine with optimal incremental coefficients (OCI-ELM) has been proposed. In OCI-ELM, a new incremental method where the linear coefficients of the existing hidden nodes and the newly added hidden node are adjusted according to the least-squares method so that the residual error can be minimized. As a consequence, OCI-ELM is a generalization of both I-ELM and CI-ELM. Then the approximation capability of the proposed algorithm OCI-ELM is analyzed. It has been proved that such OCI-ELM can approximate any continuous target function if the hidden node activation function is bounded nonconstant piecewise continuous and $\text{span}\{G(\mathbf{a}, b, \mathbf{x})\}$ is dense in L^2 . Compared with the existing I-ELM, EI-ELM and CI-ELM, OCI-ELM can obtain a better generalization performance at faster convergence speed and with more compact network architecture.

In the second work, an extreme learning machine with adaptive growth of hidden nodes (AG-ELM) has been proposed, where a novel approach to determining the network architecture is introduced. In AG-ELM, the hidden node number is determined in an adapt way in the sense that the existing networks may be replaced by newly generated networks which have less hidden node numbers and better generalization performance rather than always keeping those existing ones in other constructive or pruning ELMs. Then the universal approximation capability of such AG-ELM is shown in the following sense: (1) the obtained AG-ELM can approximate any Lebesgue p -integrable function, not merely restricted to continuous functions in most literatures; (2) the hidden activation functions used in AG-ELM can be Lebesgue p -integrable, which is a much weaker condition than those assumed before. The obtained result not only strengthens those ex-

isting results, but also provides a fundamental theory for studying the approximation capability of algorithms which use random mechanism for the parameter searching in neural networks.

In the third work, a dynamic extreme learning machine (D-ELM) performing dynamic growth of hidden nodes and incremental updating of output weights has been proposed. In D-ELM, the hidden nodes can be recruited or deleted dynamically according to their significance to network performance, so that not only the parameters can be adjusted but also the architecture can be self-adapted simultaneously. Then it has been proved that such D-ELM can approximate any Lebesgue p -integrable target function $f(\mathbf{x})$ on a compact input set X in the case where the hidden node activation function is Lebesgue p -integrable and $\text{span}\{G(\mathbf{a}, b, \mathbf{x}) : (\mathbf{a}, b) \in \mathbb{R}^d \times \mathbb{R}\}$ is dense in $L^p(X)$. The obtained result generalizes those existing function approximation results of ELMs. It validates not only for much general type of ELMs (say, the hidden activation functions can be selected from broader type of functions) but also for a very general family of target functions (say, the target functions can be any Lebesgue p -integrable function rather than continuous functions), which then underlines successful applications of ELMs.

1.4 Organization

The remaining part of the thesis is organized as follows.

Chapter 2 first gives a brief overview of neural networks, and introduces the architecture, mathematical formulation and the approximation capabilities of GSLFNs. Then some background knowledge of ELMs, including the mathematical preliminaries, the basic ELM, ELMs with different learning schemes and some existing improvements of ELMs are reviewed point for point.

Chapter 3 focuses on reviewing a specific class of ELMs — the ELMs with adjustable architectures. The general architecture design methods for neural networks are first

reviewed. Then the pruning ELMs, the constructive ELMs and other architecture-adjustable ELMs are presented systematically.

Chapter 4 introduces an incremental extreme learning machine with optimal incremental coefficients (OCI-ELM). Its approximation capability and experimental verification are shown respectively.

Chapter 5 presents an extreme learning machine with adaptive growth of hidden nodes (AG-ELM). Its approximation capability and experimental verification are shown respectively.

Chapter 6 develops a dynamic extreme learning machine (D-ELM). Its approximation capability and experimental verification are shown respectively.

Chapter 7 addresses conclusions and the potential future work.

Chapter 2

Brief of Fundamental Extreme Learning Machines

Neural networks have been motivated from the recognition that the human brain computes in an entirely different way from conventional computers [36]. In 1943, the neurophysiologist Warren McCulloch and the logician Walter Pitts established the first artificial neuron model, which is called M-P model [88]. This pioneering work inaugurated the new era of modern neural networks and a large amount of researchers have been engaged in the development of neural networks during the past several tens of years [7, 32, 37, 43, 67, 82, 93, 94, 95, 98, 112]. Indeed, neural networks have established themselves as an interdisciplinary subject with deep roots in the neurosciences, psychology, mathematics, the physical science and engineering [36].

A neural network is capable of solving complex problems due to its two information processing capabilities: (1) it possesses the massively parallel distributed structure which makes it powerful in computation; (2) it has the ability to learn and then generalize. Nowadays, neural networks have become useful tools for solving many types of problems, including function approximation (or regression analysis), classification, data processing, signal processing, expert system, clustering and economical prediction etc.

Feedforward neural networks (FNNs), which are an important class of neural networks, have been applied successfully to solve some difficult and diverse problems by training them in a supervised manner. FNNs have three distinctive characteristics: nonlinearity, high degree of connectivity and containing of one or more layers of hidden nodes. Indeed, the computing power of FNNs are derived through the combination of these characteristics together with the ability to learn from experiences, however, these same characteristics are also responsible for some deficiencies on the behavior of FNNs. One of the most noteworthy problems concerns about the learning issue.

According to conventional neural network theories, all the parameters of FNNs are tuned based on different rules during the learning process. However, tuning all the parameters may render learning inefficient and time-consuming since tuning based learning may easily converge to local minima and many iterative steps may be required. In some cases, it may take several days or more to train the networks, which cannot meet the requirements of some practical applications where a fast response to external events within an extremely short time is expected.

In order to avoid this issue, Baum [4] proved that one may fix the connections on one level and simply adjust the connections on the other level and no additional benefit is possible by using an algorithm that adjusts the weights on both levels simultaneously. It implied that for SLFNs, the input weights can be fixed and one may only adjust the output weights, but it did not discuss whether the hidden node bias should be tuned or not. Almost twenty years later, Huang et al. [62] further proved that both the input weights and hidden node biases of SLFNs can be randomly assigned and the output weight vector can be calculated analytically. This resulting novel algorithm is called extreme learning machine (ELM). Different from the traditional learning algorithms for neural networks, ELM applies random computational nodes in the hidden layer, which may be independent of the training data. It not only tends to reach the smallest training error but also the smallest norm of weights. Therefore, ELM provides better generalization performance at a much faster learning speed and with least human intervene.

ELM has recently attracted the attention from more and more researchers and variants of ELM have been proposed [11, 12, 21, 25, 26, 33, 80, 87, 92, 115, 120, 123, 124, 131, 138]. ELM and its variants have been applied to various types of applications including bank clients classification [22], sales forecasting in fashion retailing [121, 129], sales forecasting in industry retailing [15], bacterial protein subcellular localization prediction [71], protein secondary structure prediction [126], microarray gene expression cancer diagnosis [133], prediction of melting point of organic compounds [6], fault diagnosis on hydraulic tube tester [46], power utility non-technical loss analysis [99], land cover classification [101], terrain modelling [130], plant species identification [132], water treatment process [136], face recognition [96, 139], human action recognition [92], image deblurring with filters [127], transmission line protection [86], no-reference image quality assessment [122], reversible watermarking [24], testing correct model specification [17], XML document classification [137] and classification of music genres [81], etc.

2.1 Overview of Neural Networks

A neural network is a kind of computational module by resembling the structure and scheme of human brain in two aspects: (i) knowledge is acquired by the network through a learning process; (ii) interneuron connection strengths known as synaptic weights are used to store the knowledge. It has been developed to be one of standard information processing technologies. This section gives a brief overview of neural networks.

2.1.1 M-P Model of a neuron

Originated from the idea of *neurons* as structural constituents of the brain, an artificial neuron is introduced to be a fundamental information processing unit for the operation of a neural network. There include three basic elements of a M-P neuron model, which

can be illustrated as Fig. 2.1:

Figure 2.1: Model of a neuron

- (1) A set of *synapses*, whose element is characterized by a weight of its own. Specifically, a signal x_j at the input of the synapse j connected to the neuron k is multiplied by the synaptic weight w_{kj} ;
- (2) An *adder*, which is used for calculating the sum of input signals that are weighted by the respective synapses of the neuron;
- (3) An *activation function*, which aims to limit the amplitude of the signals from the adder and keeps the output of the neuron in some range.

In mathematical terms, the neuron k can be formulated as

$$y_k = g\left(\sum_{j=1}^m w_{kj}x_j + \theta_k\right)$$

where g is the activation function and θ_k , y_k are the threshold and the output of the neuron k .

2.1.2 Network connection

In accordance with different connecting ways, neural networks are generally divided into two fundamental classes: *feedforward neural networks (FNNs)* and *recurrent neural networks (RNNs)*.

Feedforward Neural Networks (FNNs)

FNNs can be characterized as the layered neural networks, which are networks of neurons in the form of layers: (i) the input layer, which is composed of the source nodes; (ii) the hidden layers, whose function is to intervene between the external inputs and the network output. There can be one or more hidden layers in a neural network; (iii) the output layer, which constitutes the overall response of the network.

Figure 2.2: Feedforward neural network (FNN)

Seen from Fig 2.2, the signals in FNNs can only transmit in one direction, that is to say, signals travel from input layer to the first hidden layer, and the output signals of the first hidden layer are used as the input of the second hidden layer, and so on.

Recurrent Neural Networks (RNNs)

RNNs distinguish from FNNs in that a recurrent network has at least one feedback loop. For example, a recurrent network may consist of a single layer of neurons with each neuron feeding its output signal back to the inputs of the other neurons. The presence of the feedback loops has a profound impact on the learning capability of the network, and on its performance [36].

Figure 2.3: Recurrent neural network (RNN)

2.1.3 Learning process

One of the interesting properties of a neural network, which is of primary significance, is its ability to learn from the environment and then to improve its performance through learning. In the context of neural networks, learning can be defined as *a process by which the free parameters (e.g., synaptic weights and thresholds) of a neural network are adapted through a continuing process of simulation by the environment in which the network is embedded* [36]. In general, the learning process can be mathematically formulated by

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n)$$

where $w_{kj}(n)$, $w_{kj}(n+1)$ denote the the synaptic weight w_{kj} at time n and $n+1$ respectively, and $\Delta w_{kj}(n)$ denotes the adjustment.

A learning process consists of two parts, *learning algorithm* and *learning paradigm*.

Learning Algorithm

A learning algorithm is a set of well-defined rules for the solution of a learning problem. There are four basic rules: *error-correction learning* (being rooted in optimum filtering); *Hebbian learning* and *competitive learning* (being inspired by neurobiological considerations); and *Boltzman learning* (being based on ideas borrowed from thermodynamics and information theory).

Learning Paradigm

A learning paradigm refers to a model of the environment in which the neural network operates. There are three basic learning paradigms: *supervised learning* (being performed under the supervision of an external “teacher”); *reinforcement learning* (involving the use of a “critic” that involves through a trial-and-error process); *unsupervised learning* (being performed in a self-organized manner in that no external teacher or critic is required to instruct synaptic changes in the network).

2.2 Generalized Single-hidden Layer Feedforward Networks (GSLFNs)

Single-hidden layer feedforward networks (SLFNs), just as the term implies, is a special case of the FNNs which only consist of one hidden layer. This thesis focuses on the “generalized” SLFNs (GSLFN), which is an extension of SLFNs where much wider type of formulations are available for the hidden nodes.

2.2.1 Architecture and mathematical formulation of GSLFNs

Without loss of generality, a GSLFN with d input nodes, L hidden nodes and m linear output nodes is introduced first, which is illustrated as in Fig. 2.4.

Figure 2.4: Generalized single-hidden layer feedforward network (GSLFN)

Seen from Fig. 2.4, there are d inputs corresponding to the dimension of each observation. The hidden layer consists of L hidden nodes, where $G(\mathbf{a}_i, b_i, \mathbf{x})$ denotes the output of the i th hidden node with the hidden node parameters $(\mathbf{a}_i, b_i) \in R^d \times R$ ($i = 1, 2, \dots, L$). Denote by $\beta_i \in R^m$ the weight vector connecting between the i th hidden node and the output nodes, then the output of the network can be represented in a general form by

$$f_L(\mathbf{x}) = \sum_{i=1}^L \beta_i G(\mathbf{a}_i, b_i, \mathbf{x}). \quad (2.1)$$

It should be noted that the hidden nodes in GSLFNs can be neuron alike, or nonneuron alike.

Neuron-alike hidden nodes

Neuron-alike hidden nodes indicate the ones which are formulated according to the conventional neural network theory. Two types of neuron-alike hidden nodes have been mainly used:

(1) *Additive hidden nodes*. For additive nodes with activation function g , the hidden node output is defined as

$$G(\mathbf{a}_i, b_i, \mathbf{x}) = g(\mathbf{a}_i \cdot \mathbf{x} + b_i), \mathbf{a}_i \in R^d, b_i \in R$$

where $\mathbf{a}_i$ is the weight vector connecting the input layer to the i th hidden node and b_i is the bias of the i th hidden node. Here, $\mathbf{a}_i \cdot \mathbf{x}$ denotes the inner product of the weight vector $\mathbf{a}_i$ and the input $\mathbf{x}$.

(2) *Radial basis function (RBF) hidden nodes*. The RBF network can be considered as a special SLFN which applies RBF nodes in its hidden layer. The output of each RBF node is a radially symmetric function of the distance between the input and the center. For RBF nodes, the hidden node output is defined as

$$G(\mathbf{a}_i, b_i, \mathbf{x}) = \exp(-b_i \|\mathbf{x} - \mathbf{a}_i\|^2), \mathbf{a}_i \in R^d, b_i \in R^+$$

where $\mathbf{a}_i$ and b_i are the center and impact factor of the i th RBF node. Here, R^+ indicates the set of all nonnegative real values.

Nonneuron-alike hidden nodes

Nonneuron-alike hidden nodes, as the name implies, represent those nodes beyond the neuron-alike class. Besides the commonly used neuron-alike hidden nodes, for

GSLFNs, there also include a wide variety of nonneuron-alike ones, including fuzzy rules [54], fully complex nodes [66, 78], hinging functions [9], high-order nodes [31], ridge polynomials [117], wavelets [19, 20], and Fourier terms [33], etc. For example, the multiquadratic nodes with the form $G(\mathbf{a}_i, b_i, \mathbf{x}) = (\|\mathbf{x} - \mathbf{a}_i\|^2 + b_i^2)^{\frac{1}{2}}$ [59, 60], the fully complex hidden nodes with the form $G(\mathbf{a}_i, b_i, \mathbf{x}) = \prod_{l=1}^{s_i} g(\mathbf{a}_{il} \cdot \mathbf{x} + b_i)$ where $(\mathbf{a}_{il}, b_i) \in \mathbb{C}^d \times \mathbb{C}$, $\mathbf{x} \in \mathbb{C}^d$ and s_i is an integer constant [53].

From the network architecture point of view, additive and RBF hidden nodes have been often used in applications.

2.2.2 Approximation capabilities of GSLFNs

The capabilities of SLFNs to approximate complex nonlinear mappings have been widely investigated in past decades due to their applications in various areas of scientific research and engineering [7, 32, 36, 82]. In accordance with conventional neural network theories, when the parameters of SLFNs are well adjusted, SLFNs can approximate any continuous function as long as the hidden activation function is suitably chosen. In 1989, Hornik [45] proved the following distinguished theorem :

Theorem 2.1 (Universal Approximation Theorem) [45] *Let $\phi(\cdot)$ be a nonconstant, bounded, and monotone-increasing continuous function. Let I_p denote the p -dimensional unit hypercube $[0, 1]^p$. The space of continuous functions on I_p is denoted by $C(I_p)$. Then given any function $f \in C(I_p)$ and $\varepsilon > 0$, there exist an integer M and sets of real constants α_i, θ_i , and w_{ij} , where $i = 1, \dots, M$ and $j = 1, \dots, p$ such that we may define*

$$F(x_1, x_2, \dots, x_p) = \sum_{i=1}^M \alpha_i \phi\left(\sum_{j=1}^p w_{ij} x_j - \theta_i\right)$$

as an approximate realization of the function $f(\cdot)$, that is,

$$|F(x_1, x_2, \dots, x_p) - f(x_1, x_2, \dots, x_p)| < \varepsilon$$

for all $(x_1, x_2, \dots, x_p) \in I_p$.

In fact, the universal approximation theorem states that an SLFN is sufficient for multi-layer perceptrons to compute a uniform ε approximation to a given input $(x_1, x_2, \dots, x_p)$ and a desired output $f(x_1, x_2, \dots, x_p)$ over compact input sets. On the basis of this result, Leshno [76] have further proved that SLFNs with additive hidden nodes and with a non-polynomial activation function can approximate any continuous target function. For RBF networks, Park and Sandberg [103] have shown that RBF networks with the same impact factor for suitably chosen kernels can approximate any continuous target function.

In real applications, neural networks are usually trained in finite training set. For function approximation in a set consisting of N training samples, Huang and Babri [48] proved that an SLFN with at most N hidden neurons and with almost any nonlinear activation function can learn N distinct observations with zero error. In [47], it has been shown that with the input weights chosen randomly, SLFNs with N additive hidden nodes can learn N distinct observations with arbitrarily small error. Moreover, Huang et al. [51] has proved theoretically that SLFNs with randomly generated hidden nodes are indeed universal approximators, and further extended the result to GSLFNs.

2.3 Extreme Learning Machines (ELMs)

2.3.1 Mathematical preliminaries

This subsection will briefly introduce the Moore-Penrose generalized inverse and the minimum norm least-squares solution of a general linear system, which are the preliminary mathematical knowledge for the implementation of ELMs.

Moore-Penrose generalized inverse

For a general linear system $\mathbf{Ax} = \mathbf{y}$, if the matrix $\mathbf{A}$ is nonsingular, then it is well known that the solution of the system is $\mathbf{x} = \mathbf{A}^{-1}\mathbf{y}$ where $\mathbf{A}^{-1}$ is the inverse of $\mathbf{A}$. However, in many applications, the matrix $\mathbf{A}$ may be singular and may even be a non-square matrix, where the inverse $\mathbf{A}^{-1}$ does not exist any more. In this case, the Moore-Penrose generalized inverse is one of the most commonly used resolutions [116].

Definition 2.1 [107, 116] *A matrix $\mathbf{G}$ of order $n \times m$ is the Moore-Penrose generalized inverse of matrix $\mathbf{A}$ of order $m \times n$, if*

$$\mathbf{AGA} = \mathbf{A}, \quad \mathbf{GAG} = \mathbf{G}, \quad (\mathbf{AG})^T = \mathbf{AG}, \quad (\mathbf{GA})^T = \mathbf{GA}$$

For the sake of convenience, the *Moore-Penrose generalized inverse* of matrix $\mathbf{A}$ will be denoted by $\mathbf{A}^\dagger$.

Different methods can be used to calculate Moore-Penrose generalized inverse of a matrix: orthogonal projection method, orthogonalization method, iterative method, and singular value decomposition (SVD) etc [107]. The orthogonal projection method can be used in two cases: $\mathbf{A}^\dagger = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T$ when $\mathbf{A}^T \mathbf{A}$ is nonsingular or $\mathbf{A}^\dagger = \mathbf{A}^T (\mathbf{A} \mathbf{A}^T)^{-1}$

when $\mathbf{A}\mathbf{A}^T$ is nonsingular.

Minimum norm least-squares solution of general linear systems

Definition 2.2 [107, 116] For a general linear system $\mathbf{A}\mathbf{x} = \mathbf{y}$, we say that $\hat{\mathbf{x}}$ is a least-squares solution if

$$\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{y}\| = \min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|$$

where $\|\cdot\|$ is a norm in Euclidean space.

Definition 2.3 [107, 116] For a general linear system $\mathbf{A}\mathbf{x} = \mathbf{y}$, we say that $\mathbf{x}_0 \in R^n$ is a minimum norm least-squares solution if for any $\mathbf{y} \in R^m$, there is

$$\|\mathbf{x}_0\| \leq \|\mathbf{x}\|, \forall \mathbf{x} \in \{\mathbf{x} : \|\mathbf{A}\mathbf{x} - \mathbf{y}\| \leq \|\mathbf{A}\mathbf{z} - \mathbf{y}\|, \forall \mathbf{z} \in R^n\}$$

From the above definition, it can be seen that $\mathbf{x}_0$ is a minimum norm least-squares solution of the linear system $\mathbf{A}\mathbf{x} = \mathbf{y}$ if and only if it has the smallest norm among all the least-squares solutions, that is, $\mathbf{x}_0$ is the solution of both optimal problems: $\min \|\mathbf{A}\mathbf{x} - \mathbf{y}\|$ and $\min \|\mathbf{x}\|$.

Theorem 2.2 [107, 116] Let there exist a matrix $\mathbf{G}$ such that $\mathbf{G}\mathbf{y}$ is a minimum norm least-squares solution of a linear system $\mathbf{A}\mathbf{x} = \mathbf{y}$. Then it is necessary and sufficient that $\mathbf{G} = \mathbf{A}^\dagger$.

Remark 2.1 From Theorem 2.2, the following properties can be obviously obtained:

- (1) The special solution $\mathbf{x}_0 = \mathbf{A}^\dagger \mathbf{y}$ is one of the least-squares solutions of the general linear system $\mathbf{A}\mathbf{x} = \mathbf{y}$, that is,

$$\|\mathbf{A}\mathbf{x}_0 - \mathbf{y}\| = \|\mathbf{A}\mathbf{A}^\dagger \mathbf{y} - \mathbf{y}\| = \min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{y}\|.$$

- (2) Further, the special solution $\mathbf{x}_0 = \mathbf{A}^\dagger \mathbf{y}$ is also the minimum norm least-squares solution of the general linear system $\mathbf{A}\mathbf{x} = \mathbf{y}$, that is,

$$\begin{aligned} \|\mathbf{x}_0\| &= \|\mathbf{A}^\dagger \mathbf{y}\| \leq \|\mathbf{x}\|, \\ \forall \mathbf{x} &\in \{\mathbf{x} : \|\mathbf{A}\mathbf{x} - \mathbf{y}\| \leq \|\mathbf{A}\mathbf{z} - \mathbf{y}\|, \forall \mathbf{z} \in R^n\} \end{aligned}$$

- (3) The minimum norm least-squares solution of the general linear system $\mathbf{A}\mathbf{x} = \mathbf{y}$ is unique, which is $\mathbf{x} = \mathbf{A}^\dagger \mathbf{y}$.

2.3.2 Brief on basic ELM

The essence of ELMs is that: different from the common understanding of learning, the hidden layer of SLFNs need not be tuned. This subsection briefly reviews the basic ELM, whose implementation is to apply random computational nodes in the hidden layer, which may be independent of the training data.

Given training examples $\{(\mathbf{x}_j, \mathbf{t}_j)\}_{j=1}^N \subset R^d \times R^m$, if the output of the network is equal to the target, then according to the equation (2.1), there are

$$f_L(\mathbf{x}_j) = \sum_{i=1}^L \beta_i G(\mathbf{a}_i, b_i, \mathbf{x}_j) = \mathbf{t}_j, \quad j = 1, \dots, N,$$

which can also be equivalently expressed in the matrix form

$$\mathbf{H}\beta = \mathbf{T}, \quad (2.2)$$

where

$$\mathbf{H} = \begin{pmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_1) & \dots & G(\mathbf{a}_L, b_L, \mathbf{x}_1) \\ G(\mathbf{a}_1, b_1, \mathbf{x}_2) & \dots & G(\mathbf{a}_L, b_L, \mathbf{x}_2) \\ \vdots & & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_N) & \dots & G(\mathbf{a}_L, b_L, \mathbf{x}_N) \end{pmatrix}_{N \times L}, \quad \beta = \begin{pmatrix} \beta_1^T \\ \beta_2^T \\ \vdots \\ \beta_L^T \end{pmatrix}_{L \times m}, \quad \mathbf{T} = \begin{pmatrix} \mathbf{t}_1^T \\ \mathbf{t}_2^T \\ \vdots \\ \mathbf{t}_N^T \end{pmatrix}_{N \times m}.$$

Here, $\mathbf{H}$ is called the hidden-layer output matrix of the network whose i th row represents the output vector of the hidden layer with respect to the i th input $\mathbf{x}_i$, and j th column denotes the j th hidden node's output vector with respect to the inputs $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$.

Unlike the traditional algorithms for training SLFNs (such as BP algorithm), ELM randomly generates the hidden-layer output matrix $\mathbf{H}$. Specifically, the parameters $(\mathbf{a}_i, b_i)$ of the hidden layer are generated randomly from $R^d \times R$ based on a continuous sampling distribution probability. In this way, given the training data, after the hidden nodes are randomly generated, the hidden-layer output matrix $\mathbf{H}$ is known and need not be tuned. Thus, training SLFNs simply amounts to getting the solution of the linear system (2.2) with respect to the output weight vector β . To solve this problem, the following two cases are considered.

Case I. The number of hidden nodes is equal to the number of training samples, i.e. $L = N$.

In this case, the hidden-layer output matrix $\mathbf{H}$ is square. It has been shown that $\mathbf{H}$ can be nonsingular as long as the hidden activation function is suitable chosen, and hence, the

given SLFN can approximate the training samples with zero error. Therefore we have

$$\hat{\beta} = \mathbf{H}^{-1}\mathbf{T}$$

and the following theorem holds:

Theorem 2.3 [62] *Given an SLFN with N hidden nodes and activation function $g(x)$, which is infinitely differentiable in any interval of $\mathbb{R}$. Then for N arbitrary distinct inputs $\{\mathbf{x}_i | \mathbf{x}_i \in \mathbb{R}^d, i = 1, \dots, N\}$ and $\{(\mathbf{a}_i, b_i)\}_{i=1}^N$ that is randomly generated from any intervals of $\mathbb{R}^d$ and $\mathbb{R}$ respectively according to any continuous probability distribution, the hidden layer output matrix $\mathbf{H}$ is invertible with probability one.*

Case II. The number of hidden nodes is much less than the number of training samples, i.e. $L \ll N$.

In this case, the hidden-layer output matrix $\mathbf{H}$ is no longer square and hence cannot be nonsingular. Therefore, the problem becomes to find a minimum norm least-squares solution of the system $\mathbf{H}\beta = \mathbf{T}$ such that not only the norm of the weight vector $\|\beta\|$ is the smallest but also the residual error $\|\mathbf{H}\beta - \mathbf{T}\|$ can be minimized. In accordance with Theorem 2.2, it is known that

$$\hat{\beta} = \mathbf{H}^\dagger \mathbf{T} \tag{2.3}$$

is such a solution and the following theorem holds:

Theorem 2.4 [62] *Given any small possible value $\varepsilon > 0$ and an activation function $g(x)$ that is infinitely differentiable in any interval, there exists $L < N$ such that for N arbitrary distinct inputs $\{\mathbf{x}_i | \mathbf{x}_i \in \mathbb{R}^d, i = 1, \dots, N\}$ and $\{(\mathbf{a}_i, b_i)\}_{i=1}^N$ that is randomly generated*

from any intervals of R^d and R respectively according to any continuous probability distribution, $\|\mathbf{H}\boldsymbol{\beta} - \mathbf{T}\| < \varepsilon$ with probability one.

Remark 2.2 As discussed in Section 2.3.1, the following important properties can be concluded:

(1) *Minimum training error*

$\hat{\boldsymbol{\beta}} = \mathbf{H}^\dagger \mathbf{T}$ is one of the least-squares solutions of the system $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$, meaning that the smallest training error can be reached:

$$\|\mathbf{H}\hat{\boldsymbol{\beta}} - \mathbf{T}\| = \|\mathbf{H}\mathbf{H}^\dagger \mathbf{T} - \mathbf{T}\| = \min_{\boldsymbol{\beta}} \|\mathbf{H}\boldsymbol{\beta} - \mathbf{T}\|.$$

(2) *Smallest norm of weights*

$\hat{\boldsymbol{\beta}} = \mathbf{H}^\dagger \mathbf{T}$ has the smallest norm among all the least-squares solutions of the system $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$, that is,

$$\begin{aligned} \|\hat{\boldsymbol{\beta}}\| &= \|\mathbf{H}^\dagger \mathbf{T}\| \leq \|\boldsymbol{\beta}\| \\ \forall \boldsymbol{\beta} &\in \{\boldsymbol{\beta} : \|\mathbf{H}\boldsymbol{\beta} - \mathbf{T}\| \leq \|\mathbf{H}\mathbf{z} - \mathbf{T}\|, \quad \forall \mathbf{z} \in R^{L \times m}\}. \end{aligned}$$

(3) $\hat{\boldsymbol{\beta}} = \mathbf{H}^\dagger \mathbf{T}$ is the unique minimum norm least-squares solution of the system $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$.

Bartlett's theory [3] states that for FNNs, the smaller training error and the smaller the norm of weights reach, the better is the generalization performance of the network. Therefore, according to the method mentioned above, the ELM tends to have good generalization performance.

The three-step learning algorithm ELM [62]¹ can be summarized as follows.

¹The source codes of ELM can be downloaded from <http://www.ntu.edu.sg/home/egbhuang/>

Algorithm ELM: Given a training set $\{(\mathbf{x}_j, \mathbf{t}_j)\}_{j=1}^N \subset \mathbb{R}^d \times \mathbb{R}^m$, the hidden node output function $G(\mathbf{a}, b, \mathbf{x})$ and the number L of hidden nodes.

- 1) Randomly assign hidden node parameter $(\mathbf{a}_i, b_i), i = 1, \dots, L$;
- 2) Calculate the hidden-layer output matrix $\mathbf{H}$;
- 3) Calculate the output weight vector $\hat{\beta}$ according to (2.3).

The basic ELM uses the orthogonal projection method to compute the Moore-Penrose generalized inverse of the matrix $\mathbf{H}$:

$$\begin{aligned}\hat{\beta} &= (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T \mathbf{T} \quad \text{as } \mathbf{H}^T \mathbf{H} \text{ is nonsingular;} \\ \hat{\beta} &= \mathbf{H}^T (\mathbf{H} \mathbf{H}^T)^{-1} \mathbf{T} \quad \text{as } \mathbf{H} \mathbf{H}^T \text{ is nonsingular.}\end{aligned}\tag{2.4}$$

It is worth noting that the basic ELM is a batch learning algorithm with a fixed network structure. The number of hidden nodes in the hidden layer needs to be determined by the user before the learning process. This parameter is normally obtained by a trial and error method, which may be tedious for some applications.

2.3.3 ELMs with different learning schemes

As the basic ELM was proposed in [62], many variants of ELM have been developed. In accordance with the different learning schemes, the collection of ELMs can be divided into the following two types.

ELMs with batch learning

The batch learning corresponds to the learning procedure which updates its model of the environment after all the experiences are processed. The basic ELM uses the batch

learning scheme, and based on it, a number of its variants have been developed from different points of view. Different from many popular learning algorithms that cannot be used to approximate networks with threshold networks directly, Huang et al. [62] stated that ELM can work with wide type of activation functions, including both differentiable and nondifferentiable functions. In order to improve the stability of ELM, a new computation method [60] to calculate the optimal output weight based on the ridge regression theory [41] was established. On the other hand, Huang et al. [52] built up another optimization based method to calculate the optimal output weights. In the case where the hidden layer feature mapping is unknown to users, Huang et al. [60] studied the kernel based ELM where a kernel matrix was defined in ELM. For complex-value networks, rather than using split-complex activation function, the fully complex ELM [53, 78] which can use fully complex activation function directly was proposed.

ELMs with online learning

Online learning is another style of learning procedure where the learner updates its model of the environment whenever a new experience arrives. With the online learning, all the training examples are sequentially (one-by-one) presented to the learning system and only one example is learned at each time. The learned example may be randomly or circularly selected from the given training examples, but should keep periodic in the training set. Online learning algorithms are preferred over batch learning algorithms in many industrial applications where training data may come one by one or chunk by chunk. However, online learning is difficult to be implemented for FNNs with additive (e.g. [75]) or RBF hidden nodes since the conventional online learning algorithms have several parameters for users to specify and it is very time-consuming to tune those parameters [57, 58, 65, 83, 84, 104, 109, 114].

Online Sequential ELM (OS-ELM) [79] is a simple and efficient online learning algorithm that can handle both additive and RBF nodes in a unified framework. OS-ELM can learn the training data not only one by one but also chunk by chunk (with fixed or varying

length) and discards the data for which the training has already been done. In order to further improve the performance of OS-ELM, Lan et al. [72] proposed the ensemble of online sequential ELM (EOS-ELM), where several OS-ELM networks are integrated. Rong et al. [110] proposed an online sequential fuzzy ELM (OS-Fuzzy-ELM) where the equivalence between Takagi-Sugeno-Kang (TSK) fuzzy inference system (FIS) and SLFNs is used to design the algorithm.

2.3.4 Variants of ELMs

This subsection lists some representative algorithms in the development of ELMs.

Random hidden layer feature mapping based ELM

According to the ridge regression theory [41], it was suggested that a positive value $\frac{1}{\lambda}$ is added to the diagonal of $\mathbf{H}^T \mathbf{H}$ or $\mathbf{H} \mathbf{H}^T$ in the calculation of the output weights so that the resultant solution is stabler and tends to have better generalization performance [60, 125]. That is, in order to improve the stability of ELM, $\hat{\beta}$ can be calculated by

$$\begin{aligned}\hat{\beta} &= \left(\frac{\mathbf{I}}{\lambda} + \mathbf{H}^T \mathbf{H}\right)^{-1} \mathbf{H}^T \mathbf{T} && \text{as the number of training samples is huge;} \\ \hat{\beta} &= \mathbf{H}^T \left(\frac{\mathbf{I}}{\lambda} + \mathbf{H} \mathbf{H}^T\right)^{-1} \mathbf{T} && \text{as the number of training samples is not huge.}\end{aligned}$$

It has been shown that the above equations are actually the solutions to the regularized problem $\min\{\|\mathbf{H}\beta - \mathbf{T}\|^2 + \lambda \|\beta\|^2\}$ where λ is called the regularization factor used to balance the two items $\|\mathbf{H}\beta - \mathbf{T}\|$ and $\|\beta\|$ [60].

Kernel based ELM

In [60], another algorithm referred to as the kernel based ELM was also proposed. In the case where the hidden layer matrix $\mathbf{H}$ is unknown, one can define an alternative kernel matrix as

$$\Omega_{ELM} = \mathbf{H}\mathbf{H}^T$$

where the element in the i th row and j th column of the kernel matrix Ω_{ELM} is defined by

$$\Omega_{ELM\ i,j} = \mathbf{h}(\mathbf{x}_i) \cdot \mathbf{h}(\mathbf{x}_j) = K(\mathbf{x}_i, \mathbf{x}_j)$$

Here, $K(\mathbf{u}, \mathbf{v})$ is called the kernel function (e.g. the Gaussian kernel $K(\mathbf{u}, \mathbf{v}) = \exp(-\gamma \|\mathbf{u} - \mathbf{v}\|^2)$) and

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}(\mathbf{x}_1) \\ \vdots \\ \mathbf{h}(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} h_1(\mathbf{x}_1) & \cdots & h_L(\mathbf{x}_1) \\ \vdots & \cdots & \vdots \\ h_1(\mathbf{x}_N) & \cdots & h_L(\mathbf{x}_N) \end{bmatrix}$$

where $\mathbf{h}(\mathbf{x}_i) = [h_1(\mathbf{x}_i) \cdots h_L(\mathbf{x}_i)]$ is defined as a row vector in order to keep the consistent notation and formula formats (note that the rest vectors are defined as column vectors in

the whole thesis unless explicitly specified). Hence, the output of the network becomes

$$f(\mathbf{x}) = \begin{bmatrix} K(\mathbf{x}, \mathbf{x}_1) \\ K(\mathbf{x}, \mathbf{x}_2) \\ \vdots \\ K(\mathbf{x}, \mathbf{x}_N) \end{bmatrix}^T \left(\frac{\mathbf{I}}{\lambda} + \Omega_{ELM}^{-1} \right) \mathbf{T}.$$

Online sequential ELM

In OS-ELM [79], the training observations are sequentially presented to the learning system, and the observation will be discarded and may not be used again as soon as the learning procedure for it has been completed. In OS-ELM, a recursive least-squares algorithm is applied to implement the solution of output weight vector shown as in (2.4).

Algorithm OS-ELM: Giving a training set $\mathfrak{X} = \{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in R^d, \mathbf{t}_i \in R^m, i = 1, 2, \dots\}$, the hidden node output function $G(\mathbf{a}_i, b_i, \mathbf{x})$, and the number L of hidden nodes.

Step I. Initialization Phase: Initialize the learning using a small chunk of initial training data $\mathfrak{X}_0 = \{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=1}^{N_0}$ ($N_0 \geq L$) from $\mathfrak{X}$.

- (a) Randomly generate the hidden node parameters $(\mathbf{a}_i, b_i), i = 1, \dots, L$;
- (b) Calculation of the initial hidden layer output matrix $\mathbf{H}_0$:

$$\mathbf{H}_0 = \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_1) & \cdots & G(\mathbf{a}_L, b_L, \mathbf{x}_1) \\ \vdots & \cdots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_{N_0}) & \cdots & G(\mathbf{a}_L, b_L, \mathbf{x}_{N_0}) \end{bmatrix}_{N_0 \times L};$$

- (c) Estimate the initial output weight $\beta^{(0)} = \mathbf{P}_0 \mathbf{H}_0^T \mathbf{T}_0$ where $\mathbf{P}_0 = (\mathbf{H}_0^T \mathbf{H}_0)^{-1}$ and $\mathbf{T}_0 = [\mathbf{t}_1, \mathbf{t}_2, \dots, \mathbf{t}_{N_0}]^T$;

(d) Set $k = 0$.

Step II. Sequential Learning Phase: Present the $(k + 1)$ th chunk of new observations $\mathfrak{X}_{k+1} = \{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=(\sum_{j=0}^k N_j)+1}^{\sum_{j=0}^{k+1} N_j}$ where N_{k+1} denotes the number of observations in the $(k + 1)$ th chunk.

(a) Calculate the partial hidden layer output matrix $\mathbf{H}_{k+1}$ for the $(k + 1)$ th chunk of data $\mathfrak{X}_{k+1}$:

$$\mathbf{H}_{k+1} = \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_{(\sum_{j=0}^k N_j)+1}) & \cdots & G(\mathbf{a}_L, b_L, \mathbf{x}_{(\sum_{j=0}^k N_j)+1}) \\ \vdots & \cdots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_{\sum_{j=0}^{k+1} N_j}) & \cdots & G(\mathbf{a}_L, b_L, \mathbf{x}_{\sum_{j=0}^{k+1} N_j}) \end{bmatrix}_{N_{k+1} \times L}.$$

Set $\mathbf{T}_{k+1} = [\mathbf{t}_{(\sum_{j=0}^k N_j)+1}, \dots, \mathbf{t}_{\sum_{j=0}^{k+1} N_j}]^T$;

(b) Calculate the output weight $\beta^{(k+1)}$:

$$\begin{aligned} \mathbf{P}_{k+1} &= \mathbf{P}_k - \mathbf{P}_k \mathbf{H}_{k+1}^T (\mathbf{I} + \mathbf{H}_{k+1} \mathbf{P}_k \mathbf{H}_{k+1}^T)^{-1} \mathbf{H}_{k+1} \mathbf{P}_k \\ \beta^{(k+1)} &= \beta^{(k)} + \mathbf{P}_{k+1} \mathbf{H}_{k+1}^T (\mathbf{T}_k - \mathbf{H}_{k+1} \beta^{(k)}); \end{aligned}$$

(c) Set $k = k + 1$ and go to *Step II(a)*.

Optimization based ELM for classification

As stated above, the objective of ELM for binary classification is to minimize the training error as well as the norm of the output weights:

$$\text{Minimize:} \quad \sum_{i=1}^N \| \mathbf{h}(\mathbf{x}_i) \beta - t_i \|$$

and

$$\text{Minimize:} \quad \| \beta \|$$

where $t_i = 1$ or $t_i = -1$. Since it is most possible that some testing data may be within the classification margin if zero training error is strictly obtained, one may wish to separate the training data with an acceptable minimal training error ξ_i instead of the zero training error so that the testing error can thus be minimized accordingly [52]:

$$\begin{aligned} \mathbf{h}(\mathbf{x}_i)\beta &\geq 1 - \xi_i, \text{ if } t_i = 1; \\ \mathbf{h}(\mathbf{x}_i)\beta &\leq -1 + \xi_i, \text{ if } t_i = -1. \end{aligned}$$

That is,

$$t_i \mathbf{h}(\mathbf{x}_i)\beta \geq 1 - \xi_i, \quad i = 1, \dots, N.$$

Therefore, from the standard optimization point of view, the above objective can be rewritten as

$$\text{Minimize:} \quad L_p = \frac{1}{2} \|\beta\|^2 + C \sum_{i=1}^N \xi_i$$

$$\text{Subject to:} \quad t_i \mathbf{h}(\mathbf{x}_i)\beta \geq 1 - \xi_i, \quad i = 1, \dots, N$$

$$\xi_i \geq 0, \quad i = 1, \dots, N$$

where C is a positive scalar.

According the tradition optimization method, it can be obtained that

$$\beta = \sum_{i=1}^N \alpha_i t_i \mathbf{h}(\mathbf{x}_i)$$

where α_i is the solution to the following optimization problem:

$$\text{Minimize:} \quad L_D = \sum_{i=1}^N \sum_{j=1}^N t_i t_j \alpha_i \alpha_j \mathbf{h}(\mathbf{x}_i) \cdot \mathbf{h}(\mathbf{x}_j) - \sum_{i=1}^N \alpha_i$$

Subject to: $0 \leq \alpha_i \leq C, i = 1, \dots, N$

The optimization based ELM presents the following important results: (1) the consistency between the SVM's maximal separating margin property and the ELM's minimal norm of output weights property is shown under the ELM learning framework. (2) from the standard optimization point of view, it is equivalent to SVM, while it has milder optimization constraints than SVM due to its special separability feature. (3) it can achieve better generalization performance than SVM, and is less sensitive to user specified parameters.

ELM ensemble

The idea of neural network ensemble was proposed by Hansen and Salamon [34], which shows that a single network's performance can be expected to be improved using an ensemble of neural networks. After that this technique has been spread widely. An integration of several ELMs was proposed by Sun et al. [121] to predict the future sales amount. Several ELM networks are connected in parallel and the average of the ELMs' outputs is used as the final predicted sales amount. The findings of the paper indicate that this extension of ELM has a smaller predicting error even though the fluctuation of the output of an individual ELM is large.

Chen et al. [16] proposed an ELM ensemble by applying product index based excluding ensemble approach on regression applications. The experimental results show that the proposed ELM ensemble effectively improves the generalization performance and the PIE method outperforms bagging and simple averaging approaches.

Heeswijk et al. [38] investigated the adaptive ensemble models of ELM on the application of one-step ahead prediction in (non-)stationary time series. It is verified that the method does work on stationary time series and the capability of the method on non-stationary time series is also tested. The empirical studies show that the adaptive ensemble model achieves an acceptable testing error with good adaptivity.

2.4 Summary

This chapter first gave a brief overview of neural networks, including some basic concepts of neural networks, such as the model of a neuron, networks connections and learning process. Then the architecture, mathematical formulation and the approximation capability of GSLFNs were introduced, for which the study on ELMs in this thesis has been made. Finally, the fundamental ELMs were reviewed from the aspects of the basic ELM, ELMs with different learning schemes and some essential variants of ELMs.

Generally speaking, the essence of ELMs is that the hidden layer of GSLFNs need not be tuned, which is different from the common understanding of learning. Compared with those traditional computational intelligence techniques, ELMs provide better generalization performance, which is evaluated by both the smallest training error and the smallest norm of output weights, as well as attain a much faster learning rate with least human intervene. As an emergent learning technique, ELMs have demonstrated good potentials to resolving regression and classification problems.

Chapter 3

Review of Extreme Learning Machines with Adjustable Architectures

It is well known that the network architecture is a main factor affecting the generalization performance of a learning algorithm for SLFNs because the architecture has significant impact on a network's information processing capabilities. For an SLFN with too few hidden nodes, it may lead to high training error and poor generalization performance due to underfitting and large statistic bias. On the other hand, for an SLFN with too many hidden nodes, although low training error can be attained, it may still lead to poor generalization performance due to overfitting and large variance. Therefore, how to find an appropriate and parsimonious network which can fit the data at the same time is the rule of thumb to obtain good generalization performance.

This chapter will first summarize the popular architecture design methods for neural networks, and then review a class of ELMs with adjustable architectures systematically, which provide the background knowledge for the study in this thesis.

3.1 Architecture Design Methods for Neural Networks

The problem of designing a suitable network architecture is a complex task. Basically, three architecture design methods are commonly applied: pruning algorithms, constructive algorithms and evolutionary computation. As one of them, evolutionary computation has been widely used in the late years to evolve neural network architectures in many applications. These works fall in two broad categories of evolutionary computation: genetic algorithms (GA) and evolutionary programming (EP). COVNET [29] is a recent paradigm which develops subnetworks instead of whole networks. These modules are combined forming groups that make up a network. Angeline et al. [1] proposed an evolutionary algorithm constructing RNNs that simultaneously acquires both network topology and weight values while making minimal architectural restrictions and avoiding structural hill climbing. In what follows, the pruning algorithms and the constructive algorithms, which are mostly pursued for solving the problem of choosing the right size network, are mainly reviewed.

3.1.1 Pruning algorithms

Pruning algorithms, also known as destructive methods, start with a larger than necessary network, that is able to learn but usually ends in overfitting, and then try to remove the hidden nodes if they are no longer actively used.

The brute-force pruning algorithm [108] is a direct approach, where the weights are set to zero and then the change in the error is evaluated. If it increases too much then restore the corresponding weight, otherwise remove it. This exhaustive method is rather slow especially for large networks, and hence lots of less direct methods have been proposed, which can be mainly divided into two groups.

One group corresponds to the methods that reduce the excess weights/nodes during the training process. *Penalty-term method* [14, 39, 64, 128], which modifies the cost

function by adding penalty terms and prunes the weights by driving them to zero during training. Some of the penalty-term methods [100, 105] introduced weight decay into the learning process, and the weights normally do not decay in a constant rate. *The gain competition technique* [68] describes a method in which the hidden nodes compete to survive. The degree of a node participating the network is measured by the magnitude of its weight vector, which is called gain. The node with the smaller gain will be driven to zero and then be removed.

Another group is the methods in which the training and pruning processes are carried out in completely separate phases. This idea has been proved to be quite effective in the analogous problem of determining the proper dimension of a classification tree, thereby improving its generalization performance [10]. Sietsma and Dow [118] described *an interactive pruning method* in which the hidden nodes are evaluated and removed manually by the designer, where several heuristics to identify the significance of each node were used. *Mozer and Snolensky's skeletonization technique* [97] estimates the functionality or relevance of each individual node and eliminates those that convey redundant information. The relevance of a node is determined by the difference of the error when the unit is presented and when it is removed from the network. *The optimal brain damage (OBD) algorithm* [18] defined “saliency” of the weights and reduced the size of a learning network by deleting the weights with small saliency, i.e. those have less effect on the training error. The saliency is measured using the second derivative of the cost function with respect to the weight. *The optimal brain surgeon (OBS) algorithm* [35] used information of all second order derivatives of the error function to perform network pruning. OBS represented a major improvement over other methods (such as OBD) due to the use of full off-diagonal information of the Hessian matrix.

3.1.2 Constructive algorithms

Unlike pruning algorithms, constructive algorithms vary the network structure by adding nodes to the hidden layer. These algorithms start with a small network (usually a single

node) and dynamically grow the network by adding and training nodes as needed until a satisfactory solution is found [28, 42]. Baum [5] proved that neural networks with learning algorithms that can add hidden nodes or weights to the networks are universal learners, which are capable of learning any learnable problems.

A variety of constructive learning algorithms have been proposed for solving the general function approximation problems. These algorithms typically use a *greedy strategy* wherein each newly added hidden node is trained to minimize the residual error as much as possible [69]. In order to avoid the case where an SLFN with simple activation functions cannot approximate the inherently complex target function, Geva et al. [30] proposed a constructive algorithm using Gaussian activation function while others such as *projection pursuit regression* [27] used a summation of several nonlinear activation functions. Parekh et al. [102] proposed a *pyramid algorithm* which constructs a layered network of threshold logic unit's (TLU's) by successfully placing each new TLU above the existing ones, and a *tilling algorithm* which constructs a strictly layered network of threshold neurons. In [40], the backpropagation algorithm was experimented with changing the number of hidden nodes dynamically.

3.1.3 Comparison between pruning algorithms and constructive algorithms

On one hand, constructive algorithms offer several significant advantages over pruning algorithms. First, it is straightforward to specify an initial network (the smallest possible network to start with no hidden node) for constructive algorithms, whereas for pruning algorithms, one does not know in practice how big the initial network should be selected. Second, constructive algorithms always search for small network solution first, and thus more computationally economical than pruning algorithms, in which the majority of the training time is spent on networks larger than necessary. Third, in some applications, a number of networks with different sizes could provide acceptable solutions to a specific task. In this situation, constructive algorithms are likely to find smaller network

solutions than pruning algorithms. Smaller networks are preferred due to their potential for more efficient hardware implementation and greater transparency in extracting the learned knowledge.

On the other hand, pruning algorithms also have a number of advantages over the constructive approach. First, the large initial size allows the network to learn reasonably quickly with less sensitivity to initial conditions while the reduced complexity of the trimmed system favors improved generalization. Second, the network can be more compact because pruning algorithms can eliminate excessive redundancy by replacing two or more nodes that encode the same information by a single node.

Although constructive algorithms and pruning algorithms address the problem of network acquisition, there still exist the common deficiency for both approaches, which stem from inadequate methods for assigning credit to structural components of a network. Generally, they do so in a highly constrained manner. Both methods investigate only restricted topological subsets rather than the complete class of network architectures, thus introduce constraints in the search space of possible structures that may not be suitable to the problem. In some of these methods, once an architecture has been explored and determined to be insufficient, a new architecture is adopted and the old one becomes topologically unreachable. Others use only a single predefined structural modification, such as adding a fully connected hidden node, to generate successive topologies.

The subsequent sections will introduce two classes of ELMs with adjustable architectures, *pruning ELMs* and *constructive ELMs*, which have been established according to different pruning methods and constructive methods respectively. The constructive ELMs will be emphasized and a more detailed description of constructive ELMs will be given.

3.2 Pruning ELMs

Rong et al. [111] proposed a *pruned ELM (P-ELM)* which is a systematic and automatic approach for ELM classifier network design. It starts from an initial large number of hidden nodes whose parameters are randomly generated. Then statistical criteria, which is identified based on the Akaike information criterion (AIC), is used to measure the relevance of each hidden node in contributing to the prediction accuracy of the classifier. The hidden nodes with no relevance or low relevance to the class labels will be removed so that a compact network architecture can be attained. P-ELM mainly focuses on pattern classification applications.

Another pruning algorithm referred to as *optimally-pruned ELM (OP-ELM)* [91] was proposed by Miche et al. The OP-ELM methodology has three steps: (1) build the initial network using the basic ELM algorithm; (2) rank the hidden nodes by applying multi-response sparse regression algorithm (MRSR) [119]; and (3) select the actual best number of hidden nodes through leave-one-out (LOO) validation. OP-ELM is applicable for both regression and classification applications.

Pouzols et al. [106] proposed the *evolving fuzzy OP-ELM (eF-OP-ELM)*, where an evolving approach to the identification of evolving TSK fuzzy inference systems based on the original offline OP-ELM and the OS-ELM was introduced. In the proposed eF-OP-ELM, a set of simple fuzzy rules is randomly generated, with random structure for the antecedents and random values for the parameters of input membership functions. Then a generalized LARS method is used to rank the fuzzy basis functions according to their accuracy evaluated on recent samples. Finally, the best number of fuzzy rules is selected by performing a fast computation of the leave-one-out validation error.

3.3 Constructive ELMs

This section focuses on the constructive ELMs, where incremental ELMs and error minimized ELM (EM-ELM) are reviewed in detail due to the fact that they provide the background knowledge for the proposed algorithms in this thesis, which will be presented in subsequent chapters.

3.3.1 Incremental ELMs (I-ELMs)

I-ELMs use the incremental learning. Different from the basic ELM, the hidden nodes of I-ELMs are added one by one with the progress of learning by using incremental methods. The input data set is given before learning. In the following, the I-ELM [51], CI-ELM [49] and EI-ELM [50] are introduced respectively.

I-ELM

A short brief on I-ELM is given first. In accordance with the conventional neural network theories, SLFNs with additive or RBF hidden nodes are universal approximators when all the parameters of the networks are allowed adjustable. On the basis of the essence of ELM, I-ELM proves in theory an incremental constructive method that in order to let SLFNs work as universal approximators, one may simply choose the random hidden nodes and then only need to adjust the output weights.

As described in chapter 2, an SLFN with d inputs, n hidden nodes and one linear output can be expressed by

$$f_n(\mathbf{x}) = \sum_{i=1}^n \beta_i G(\mathbf{a}_i, b_i, \mathbf{x}) = \sum_{i=1}^n \beta_i g_i(\mathbf{x}), \mathbf{x} \in R^d, \beta_i \in R.$$

Obviously, the above equation can be further expressed as the following incremental method:

$$f_n(\mathbf{x}) = f_{n-1}(\mathbf{x}) + \beta_n g_n(\mathbf{x}). \quad (3.1)$$

Here, $g_n(\mathbf{x})$ denotes the output of the newly added hidden node at the n th step. Given the target function $f(\mathbf{x})$, the objective of I-ELM is to find such a network sequence $\{f_n(\mathbf{x})\}$ that can approximate $f(\mathbf{x})$ with arbitrary small error. For simplicity, f, f_n and g_n are used to represent $f(\mathbf{x}), f_n(\mathbf{x})$ and $g_n(\mathbf{x})$ respectively in what follows. The notation $e_n = \|f - f_n\|$ is used to denote the residual error between f and f_n . The universal approximation capability of I-ELM has been proved.

Theorem 3.1 (Universal Approximation Theorem)[51] *Given any bounded nonconstant piecewise continuous function $g : R \rightarrow R$ for additive nodes and any integrable piecewise continuous function $g : R \rightarrow R$ with $\int_R g(x)dx \neq 0$ for RBF nodes, for any continuous target function f and any randomly generated function sequence $\{g_n\}$, $\lim \|f - f_n\| = 0$ with probability one if*

$$\beta_n = \frac{\langle e_{n-1}, g_n \rangle}{\|g_n\|^2}. \quad (3.2)$$

The above formula (3.2) gives an analytical expression of the output weight vector β_n . In practice, however, β_n could not be calculated directly according to (3.2) since the exact functional form of $f(\mathbf{x})$ is unavailable. According to the method in [70], given a set of training data $\{(\mathbf{x}_i, t_i)\}_{i=1}^N \subset R^d \times R$, a consistent estimate of β_n on the basis of the

training set can be obtained:

$$\beta_n = \frac{E \cdot H^T}{H \cdot H^T} = \frac{\sum_{p=1}^N e(p)h(p)}{\sum_{p=1}^N h(p)^2} \quad (3.3)$$

where $T = [t_1, \dots, t_p, \dots, t_N]^T$ is the given target vector, $H = [h(1), \dots, h(p), \dots, h(N)]^T$ is the activation output of the new hidden node, and $E = [e(1), \dots, e(p), \dots, e(N)]^T$ is the corresponding residual error before this new node is added. Thus, such an I-ELM can be summarized as follows:

Algorithm I-ELM: [51] Given a training set $\mathfrak{X} = \{(\mathbf{x}_i, t_i) : \mathbf{x}_i \in R^d, t_i \in R, i = 1, 2, \dots, N\}$, the activation function $g(x)$, the maximum hidden node number $L_{\max}$ and the expected training error ε .

Step I. Initialization: Let $L = 0$ and the residual error $E = T$;

Step II. Learning step:

while $L < L_{\max}$ and $\|E\| > \varepsilon$.

(a) Increase by 1 the number of hidden nodes L : $L = L + 1$.

(b) Assign random hidden node parameters $(\mathbf{a}_L, b_L)$ for the new hidden node.

(c) Calculate the output weights β_L for the newly added hidden node:

$$\beta_L = \frac{E \cdot H_L^T}{H_L \cdot H_L^T}.$$

(d) Calculate the residual error after adding the new hidden node:

$$E = E - \beta_L \cdot H_L.$$

endwhile.

CI-ELM

Unlike I-ELM, CI-ELM adopts another incremental method, which is originated from the Barron's convex optimization concept [2], to construct the network sequence $\{f_n(\mathbf{x})\}$. Specifically, the network generated at the n th step is defined as

$$f_n(\mathbf{x}) = (1 - \beta_n)f_{n-1}(\mathbf{x}) + \beta_n g_n(\mathbf{x}). \quad (3.4)$$

Comparing with (3.1) and (3.4), it can be observed that CI-ELM recalculates the output weights of the existing nodes when a new hidden node is added. Obviously, this new method is more efficient since the network output combines the former information and the new information instead of only depending on the new information as in I-ELM. Thus CI-ELM can achieve faster convergence rate and more compact network architecture than I-ELM while retaining the simplicity of I-ELM.

Theorem 3.2 [49] *Given any nonconstant piecewise continuous function $g : R \rightarrow R$, if $\text{span}\{G(\mathbf{a}, b, \mathbf{x}) : (\mathbf{a}, b) \in R^d \times R\}$ is dense in L^2 , then for any continuous target function f and any randomly generated sequence $\{g_n = G(\mathbf{a}_n, b_n, \mathbf{x})\}$, $\lim \|f - f_n\| = 0$ holds with probability one if*

$$\beta_n = \frac{\langle e_{n-1}, g_n - f_{n-1} \rangle}{\|g_n - f_{n-1}\|^2}. \quad (3.5)$$

Theorem 3.2 guarantees that the network sequence $\{f_n(\mathbf{x})\}$ obtained by (3.4) and (3.5) can approximate any continuous target function if the hidden node activation function is bounded nonconstant piecewise continuous and $\text{span}\{G(\mathbf{a}, b, \mathbf{x})\}$ is dense in L^2 . Similar

to I-ELM, an estimate of β_n can be expressed by

$$\beta_n = \frac{E \cdot [E - (F - H)]^T}{[E - (F - H)] \cdot [E - (F - H)]^T} = \frac{\sum_{p=1}^N e(p)[e(p) - (f(p) - h(p))]}{\sum_{p=1}^N [e(p) - (f(p) - h(p))]^2}. \quad (3.6)$$

Hence, the algorithm CI-ELM can be summarized as follows.

Algorithm CI-ELM: [49] Given a training set $\mathfrak{X} = \{(\mathbf{x}_i, t_i) : \mathbf{x}_i \in R^d, t_i \in R, i = 1, 2, \dots, N\}$, the activation function $g(x)$, the maximum hidden node number $L_{\max}$ and the expected training error ε .

Step I. Initialization: Let $L = 0$ and the residual error $E = T$;

Step II. Learning step:

while $L < L_{\max}$ and $\|E\| > \varepsilon$.

(a) Increase by 1 the number of hidden nodes L : $L = L + 1$.

(b) Assign random hidden node parameters $(\mathbf{a}_L, b_L)$ for the new hidden node.

(c) Calculate the output weights β_L for the newly added hidden node:

$$\beta_L = \frac{E \cdot [E - (F - H_L)]^T}{[E - (F - H_L)] \cdot [E - (F - H_L)]^T}.$$

(d) Recalculate the output weight vectors of all existing hidden nodes if $L > 1$:

$$\beta_i = (1 - \beta_L)\beta_i.$$

(e) Calculate the residual error after adding the new hidden node:

$$E = (1 - \beta_L)E - \beta_L(F - H_L).$$

endwhile.

EI-ELM

During the learning procedure of I-ELM and CI-ELM, some newly added hidden nodes may make the residual error reduce less, and on the other hand, some newly added hidden nodes may make the residual error reduce more. This means that some hidden nodes only play a very minor role in the final output of the network. Under this situation, the network complexity may be increased eventually. To avoid this issue and to obtain a more compact network architecture, EI-ELM was proposed in [50]. In addition, EI-ELM also extends I-ELM for GSLFNs.

In EI-ELM, at each learning step, several hidden nodes are randomly generated (instead of only one hidden node being randomly added in I-ELM), and among them the best one, i.e., the one with the greatest residual error, is added to the network. By doing so, those hidden nodes with less residual error may not be added eventually into the network, and hence, EI-ELM can achieve much more compact network architectures. The network sequence $\{f_n(\mathbf{x})\}$ obtained in this manner can also work as an approximator. In fact, we have

Theorem 3.3 [50] *Given an SLFN with any nonconstant piecewise continuous hidden nodes $G(\mathbf{a}, b, \mathbf{x})$, if $\text{span}\{G(\mathbf{a}, b, \mathbf{x}) : (\mathbf{a}, b) \in C^d \times C\}$ is dense in L^2 , for any continuous target function f , any randomly generated sequence $\{g_n = G(\mathbf{a}_n, b_n, \mathbf{x})\}$ and any*

positive integer k , $\lim \|f - f_n^*\| = 0$ holds with probability one if

$$\beta_n^* = \frac{\langle e_{n-1}^*, g_n^* \rangle}{\|g_n^*\|^2} \quad (3.7)$$

where $f_n^* = \sum_{i=1}^n \beta_i^* g_i^*$, $e_n^* = f - f_n^*$, $g_n^* = \{g_i | \min_{(n-1)k \leq i \leq nk} \|(f - f_{n-1}^*) - \beta_i g_i\|\}$.

The implementation of EI-ELM can be summarized as follows.

Algorithm EI-ELM: [50] Given a training set $\mathfrak{X} = \{(\mathbf{x}_i, t_i) : \mathbf{x}_i \in R^d, t_i \in R, i = 1, 2, \dots, N\}$, the hidden node output function $G(\mathbf{a}, b, \mathbf{x})$, the maximum hidden node number $L_{\max}$, the maximum number k of trials of assigning random hidden nodes at each step, and the expected training error ε .

Step I. Initialization: Let $L = 0$ and the residual error $E = T$;

Step II. Learning step:

while $L < L_{\max}$ and $\|E\| > \varepsilon$.

(a) Increase by 1 the number of hidden nodes L : $L = L + 1$.

(b) for $i = 1 : k$,

(i) assign random hidden node parameters $(\mathbf{a}_{(i)}, b_{(i)})$ for the new added hidden node.

(ii) calculate the output weight $\beta_{(i)}$ for the new hidden node:

$$\beta_{(i)} = \frac{E \cdot H_{(i)}^T}{H_{(i)} \cdot H_{(i)}^T}.$$

(iii) calculate the residual error after adding the new hidden node: $E_{(i)} = E - \beta_{(i)} \cdot H_{(i)}$.

endfor

(c) Let $i^* = \operatorname{argmin}_i \|E_{(i)}\|$ and set $E = E_{(i^*)}$, $\mathbf{a}_L = \mathbf{a}_{(i^*)}$, $b_L = b_{(i^*)}$, $\beta_L = \beta_{(i^*)}$.

endwhile.

3.3.2 Error minimized ELM (EM-ELM)

This section briefly describes the EM-ELM. EM-ELM is a simple and efficient approach which can randomly add hidden nodes one by one or group by group. During the growth of the networks, the output weights are updated incrementally based on the error minimization method.

In EM-ELM, given target error $\varepsilon > 0$, an SLFN $f_{L_0}(\mathbf{x}) = \sum_{i=1}^{L_0} \beta_i G(\mathbf{a}_i, b_i, \mathbf{x})$ with L_0 hidden nodes is first initialized. Denote by $\mathbf{H}_1$ the hidden-layer output matrix of this network. Then according to (2.3), the output weight matrix can be calculated by $\hat{\beta}_1 = \mathbf{H}_1^\dagger \mathbf{T}$.

If the network output error $\|\mathbf{H}_1 \hat{\beta}_1 - \mathbf{T}\| > \varepsilon$, then new hidden nodes $\delta L_0 = L_1 - L_0$ are added to the existing SLFN and the new hidden-layer output matrix becomes $\mathbf{H}_2 = [\mathbf{H}_1, \delta \mathbf{H}_1]$. In this case, different from ELM where $\hat{\beta}_2 = \mathbf{H}_2^\dagger \mathbf{T}$ is obtained by calculating the Moore-Penrose inverse $\mathbf{H}_2^\dagger$ directly, EM-ELM proposes a fast output weight updating method to calculate $\mathbf{H}_2^\dagger$ in the following way:

$$\mathbf{H}_2^\dagger = \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{D}_1 \end{bmatrix} = \begin{bmatrix} ((\mathbf{I} - \mathbf{H}_1 \mathbf{H}_1^\dagger) \delta \mathbf{H}_1)^\dagger \\ \mathbf{H}_1^\dagger - \mathbf{H}_1^\dagger \delta \mathbf{H}_1 \mathbf{D}_1 \end{bmatrix}$$

Like the same way, the output weights are updated incrementally as

$$\hat{\beta}^{(k+1)} = \mathbf{H}_{k+1}^\dagger \mathbf{T} = \begin{bmatrix} \mathbf{U}_k \\ \mathbf{D}_k \end{bmatrix} \mathbf{T}$$

$$\begin{aligned}
\mathbf{D}_k &= ((\mathbf{I} - \mathbf{H}_k \mathbf{H}_k^\dagger) \delta \mathbf{H}_k)^\dagger \\
\mathbf{U}_k &= \mathbf{H}_k^\dagger - \mathbf{H}_k^\dagger \delta \mathbf{H}_k \mathbf{D}_k
\end{aligned} \tag{3.8}$$

On the basis of the above updating method, EM-ELM can be summarized as the following two phases.

EM-ELM Algorithm: Given a training set $\mathfrak{N} = \{(\mathbf{x}_i, t_i) : \mathbf{x}_i \in R^d, t_i \in R, i = 1, 2, \dots, N\}$, the maximum hidden node number L_{max} , and the expected training error ε .

Phase I — Initialization Phase:

- (1) Randomly generate a small group of hidden nodes $\{(\mathbf{a}_i, b_i)\}_{i=1}^{L_0}$, where L_0 is a small positive integer given by users.
- (2) Calculate the hidden layer output matrix:

$$\mathbf{H}_1 = \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_1) & \cdots & G(\mathbf{a}_{L_0}, b_{L_0}, \mathbf{x}_1) \\ \vdots & \cdots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_N) & \cdots & G(\mathbf{a}_{L_0}, b_{L_0}, \mathbf{x}_N) \end{bmatrix}_{N \times L_0}.$$

- (3) Calculate the corresponding output error $E(\mathbf{H}_1) = \|\mathbf{H}_1 \mathbf{H}_1^\dagger \mathbf{T} - \mathbf{T}\|$.
- (4) Let $k = 0$.

Phase II — Recursively Growing Phase

while $L_k < L_{max}$ and $E(\mathbf{H}_k) > \varepsilon$.

- (1) Randomly add δL_k hidden nodes to the existing SLFN. The number of hidden nodes in the network now becomes $L_{k+1} = L_k + \delta L_k$ and the corresponding hidden

layer output matrix becomes $\mathbf{H}_{k+1} = [\mathbf{H}_k, \delta\mathbf{H}_k]$, where $\delta\mathbf{H}_k$ is given by

$$\delta\mathbf{H}_k = \begin{bmatrix} G(\mathbf{a}_{L_k+1}, b_{L_k+1}, \mathbf{x}_1) & \cdots & G(\mathbf{a}_{L_k+1}, b_{L_k+1}, \mathbf{x}_1) \\ \vdots & \cdots & \vdots \\ G(\mathbf{a}_{L_k+1}, b_{L_k+1}, \mathbf{x}_N) & \cdots & G(\mathbf{a}_{L_k+1}, b_{L_k+1}, \mathbf{x}_N) \end{bmatrix}_{N \times \delta L_k}.$$

(2) Update the output weight β according to (3.8).

(3) $k = k + 1$.

The key improvement of EM-ELM is that there is no need to recompute the whole pseudo-inverse matrix so that the computational speed gets to greatly raise. In addition, different from I-ELMs which add hidden nodes one by one, EM-ELM can add the hidden nodes one by one or group by group. Obviously, the learning speed can also be raised in this way.

3.4 Other Architecture-Adjustable ELMs

With the exception of the pruning ELMs and the constructive ELMs, there have been developed some other architecture-adjustable ELMs that aim to find the most appropriate network architectures. This section gives a brief introduction on several representative algorithms.

Stepwise forward selection based constructive ELM for regression

Unlike I-ELMs and EM-ELM where the leaning stops when the network reaches the maximum number of hidden nodes or the expected training rate so that they cannot lead to the optimal structure automatically, *a stepwise forward selection based constructive ELM* was proposed in [73], where a large number of hidden nodes is first randomly

generated as the candidate reservoir, and then the hidden nodes are selected one by one according to a stepwise forward selection method. At each step, the hidden node whose output that has the highest correlation with the current residual is selected. The key advantage of the algorithm is that it can explicitly identify the network contribution of the newly added node without solving the whole least-squares problem, which significantly reduces the computational complexity.

Two-stage ELM for regression

Lan et al. [74] proposed *two-stage ELM (TS-ELM)* which uses the combination of both pruning method and the constructive method. In the first stage, a large number of hidden nodes is randomly generated as the candidate reservoir and the hidden nodes are selected and added to the network by forward recursive algorithm according to the final prediction error (FPE) criterion. While in the second stage, the significance of each selected node is measured and then the insignificant ones are removed from the network. Simulation results show that TS-ELM reduces the network complexity.

Structure-adjustable online ELM (SAO-ELM)

Li et al. [77] proposed a structure-adjustable online ELM (SAO-ELM) whose number of hidden nodes can be adjusted in OS-ELM. In online learning algorithms, the main challenge of adding hidden nodes is that the output of the newly added nodes on the old data, which have been discarded after having learned, is unknown. SAO-ELM introduces a solution to this challenge, where a spherical strategy was developed and an iterative equation to update the output matrix was established when the new hidden node is added. Numerical results show that SAO-ELM has significant advantages over that of the typical online learning algorithms on generalization performance while retaining the fast learning characteristic of ELM.

3.5 Summary

It is well known that the generalization ability of a learning system can be improved by reducing the system's complexity. Therefore, how to obtain a suitable and parsimonious network architecture has always been an important issue in the theoretical research and successful applications of ELMs.

The first part reviewed and summarized the pruning algorithms and the constructive algorithms, which are two popular approaches in the determination of the network architectures. Then the advantages and disadvantages of pruning algorithms and constructive algorithms were stated respectively, and also their performances were compared. Being the background knowledge, they provide different ways to the network architecture design of ELMs.

The second part presented the ELMs with adjustable architectures systematically, including pruning ELMs, constructive ELMs, and several other architecture-adjustable ELMs which have been proposed according to distinct methods. Three pruning algorithms P-ELM, OP-ELM and eF-OP-ELM were introduced, which have been proposed by applying different pruning strategies for various given tasks. Then two essential types of constructive ELMs were reviewed in detail, i.e., I-ELMs and EM-ELM, which have laid a foundation in the development of constructive ELMs. They provide the background knowledge for our study in this thesis. In addition, several other representative ELMs with adjustable architectures were listed. It should be noted that all the algorithms presented in this chapter contribute to the improvement of ELMs.

Chapter 4

Incremental Extreme Learning Machine with Optimal Incremental Coefficients (OCI-ELM)

As reviewed in the last chapter, unlike the conventional neural network theories, Huang et al. [51] recently proposed a new theory to show that in order to let SLFNs work as universal approximators, one may simply randomly choose hidden nodes (with random values for all the parameters of the hidden layer) and then only need to analytically calculate (instead of tuning) the output weights. The resulting method is referred to as I-ELM and it has been proved that such I-ELM can converge to any continuous function as long as the hidden activation functions are nonlinear piecewise continuous. Based on the idea of I-ELM, CI-ELM [49] adopts another incremental method originated from the Barron's convex optimization concept [2]. This new approach allows to adjust the output weights of the existing hidden nodes when a new hidden node is added. Under this situation, CI-ELM can achieve faster convergence rate than I-ELM while retaining the simplicity of I-ELM. However, such convex combination of the existing hidden nodes

The main ideas of this chapter can be found in "Incremental Extreme Learning Machine with Optimal Incremental Coefficients," submitted IEEE Transactions on Neural Network and Learning Systems.

and the newly added hidden node in (3.4) may not be the optimal fashion by which the residual error can be reduced largest.

On the basis of this observation, this chapter proposes an incremental extreme learning machine with optimal incremental coefficients (OCI-ELM) where a new incremental method is introduced. Different from CI-ELM where the sum of the coefficients of the existing hidden nodes and the newly added hidden node must always be one, in OCI-ELM, the new hidden nodes can be added into the network at each step in a flexible manner taking into account the interactions with the existing hidden nodes. OCI-ELM provides an optimal tradeoff between the existing hidden nodes and the newly added hidden nodes. As a result the proposed algorithm OCI-ELM becomes a generalization of I-ELM and CI-ELM. Note that the strategy of EI-ELM can also be incorporated into OCI-ELM in order to obtain more compact network architecture.

4.1 The Proposed OCI-ELM and Its Approximation Capability

Given a target function $f(\mathbf{x}) \in L^2(X)$ (say, $(\int_X |f(\mathbf{x})|^2 d\mathbf{x})^{\frac{1}{2}} < +\infty$) on a compact set $X \subset \mathbb{R}^d$, denote by $f_n(\mathbf{x})$ the network obtained at the n th step and by $e_n(\mathbf{x}) = f_n(\mathbf{x}) - f(\mathbf{x})$ the residual error of $f_n(\mathbf{x})$. In the following, we assume that the network has only one linear output node. Extension to multiple output nodes is straightforward.

In this new method, the network sequence is constructed incrementally by

$$f_n(\mathbf{x}) = \alpha_n f_{n-1}(\mathbf{x}) + \beta_n g_n(\mathbf{x}) \quad (4.1)$$

where the coefficients α_n and β_n are determined as the optimal ones leading to the resid-

ual error least, that is,

$$(\alpha_n, \beta_n) = \operatorname{argmin}\{\|e_n\|^2\} = \operatorname{argmin}\{\|f_n - f\|^2\}. \quad (4.2)$$

In this algorithm, the least-squares method is used to solve for α_n and β_n in (4.2). Let $F(\alpha_n, \beta_n) = \|e_n\|^2$, then

$$\begin{aligned} F(\alpha_n, \beta_n) &= \|\alpha_n f_{n-1} + \beta_n g_n - f\|^2 \\ &= \alpha_n^2 \|f_{n-1}\|^2 + \beta_n^2 \|g_n\|^2 + \|f\|^2 \\ &\quad + 2\alpha_n \beta_n \langle f_{n-1}, g_n \rangle - 2\alpha_n \langle f_{n-1}, f \rangle - 2\beta_n \langle g_n, f \rangle. \end{aligned}$$

According to the least-squares method, there are

$$\begin{aligned} \frac{\partial F(\alpha_n, \beta_n)}{\partial \alpha_n} &= 2\alpha_n \|f_{n-1}\|^2 + 2\beta_n \langle f_{n-1}, g_n \rangle - 2\langle f_{n-1}, f \rangle = 0, \\ \frac{\partial F(\alpha_n, \beta_n)}{\partial \beta_n} &= 2\beta_n \|g_n\|^2 + 2\alpha_n \langle f_{n-1}, g_n \rangle - 2\langle g_n, f \rangle = 0. \end{aligned}$$

Solving the above two equations for α_n and β_n , it can be obtained that

$$\begin{aligned} \alpha_n^* &= \frac{\langle f_{n-1}, f \rangle \|g_n\|^2 - \langle f_{n-1}, g_n \rangle \langle g_n, f \rangle}{\|f_{n-1}\|^2 \|g_n\|^2 - \langle f_{n-1}, g_n \rangle^2}, \\ \beta_n^* &= \frac{\langle g_n, f \rangle \|f_{n-1}\|^2 - \langle f_{n-1}, g_n \rangle \langle f_{n-1}, f \rangle}{\|f_{n-1}\|^2 \|g_n\|^2 - \langle f_{n-1}, g_n \rangle^2}. \end{aligned} \quad (4.3)$$

In this new incremental method (4.1), the hidden node is added one by one and the addition of a new hidden node proceeds in the following way: randomly generate a

new hidden node g_n and find the coefficients α_n^* , β_n^* according to (4.3) such that the resultant linear combination of g_n with the current network, i.e., $\alpha_n^* f_{n-1} + \beta_n^* g_n$ gives the minimum residual error $\min \|e_n\|$. The network sequence $\{f_n(\mathbf{x})\}$ obtained in this manner can also work as an approximator.

Theorem 4.1 *Given any bounded nonconstant piecewise continuous function $g : R \rightarrow R$, if $\text{span}\{G(\mathbf{a}, b, \mathbf{x})\}$ is dense in L^2 , then for any continuous target function f and any randomly generated function sequence $\{g_n(\mathbf{x}) = G(\mathbf{a}_n, b_n, \mathbf{x})\}$, $\lim \|f - \alpha_n^* f_{n-1} - \beta_n^* g_n\| = 0$ with probability one if α_n^* and β_n^* are calculated by (4.3).*

Proof: It is known that α_n^* and β_n^* in (4.3) are the least-squares solutions to the optimization problem $\min \|f - \alpha_n f_{n-1} - \beta_n g_n\|$, therefore for any $\alpha_n, \beta_n \in R$, there must have

$$\|f - \alpha_n^* f_{n-1} - \beta_n^* g_n\| \leq \|f - \alpha_n f_{n-1} - \beta_n g_n\|.$$

Let $\tilde{\alpha}_n = 1$ and $\tilde{\beta}_n$ be calculated by (3.2), then the following inequality

$$\|f - \alpha_n^* f_{n-1} - \beta_n^* g_n\| \leq \|f - f_{n-1} - \tilde{\beta}_n g_n\|$$

obviously holds. Therefore according to Theorem 3.1, it can be immediately derived that $\lim_{n \rightarrow \infty} \|f - \alpha_n^* f_{n-1} - \beta_n^* g_n\| = 0$ with probability one. Theorem 4.1 is thus proved.

Theorem 4.1 guarantees that the network sequence $\{f_n(\mathbf{x})\}$ obtained by (4.1) and (4.3) can approximate any continuous target function if the hidden node activation function is bounded nonconstant piecewise continuous and $\text{span}\{G(\mathbf{a}, b, \mathbf{x})\}$ is dense in L^2 . However, in practice, α_n^* and β_n^* could not be calculated directly according to (4.3) since the exact functional form of $f(\mathbf{x})$ is unavailable. Similar to I-ELM which adopts the method

in [70], given a set of training data $\{(\mathbf{x}_i, t_i)\}_{i=1}^N \subset R^d \times R$, a consistent estimate of α_n^* and β_n^* based on the training set is

$$\begin{aligned}
 \alpha_n^* &= \frac{T[T-E]^T H H^T - [T-E] H^T H T^T}{[T-E][T-E]^T H H^T - [T-E] H^T H [T-E]^T} \\
 &= \frac{\sum_{k=1}^N t_k [t_k - e(k)] \cdot \sum_{k=1}^N h(k)^2 - \sum_{k=1}^N h(k) [t_k - e(k)] \cdot \sum_{k=1}^N h(k) t_k}{\sum_{k=1}^N [t_k - e(k)]^2 \cdot \sum_{k=1}^N h(k)^2 - (\sum_{k=1}^N h(k) [t_k - e(k)])^2}, \\
 \beta_n^* &= \frac{H T^T [T-E][T-E]^T - [T-E] H^T [T-E] T^T}{[T-E][T-E]^T H H^T - [T-E] H^T H [T-E]^T} \\
 &= \frac{\sum_{k=1}^N h(k) t_k \cdot \sum_{k=1}^N [t_k - e(k)]^2 - \sum_{k=1}^N h(k) [t_k - e(k)] \cdot \sum_{k=1}^N t_k [t_k - e(k)]}{\sum_{k=1}^N [t_k - e(k)]^2 \cdot \sum_{k=1}^N h(k)^2 - (\sum_{k=1}^N h(k) [t_k - e(k)])^2}.
 \end{aligned} \tag{4.4}$$

Here, the symbols T, H, E have the same meanings as those indicated in chapter 3.

In real applications, one may not really wish to get zero approximation error by adding an infinite number of nodes to the network, so a maximum number of hidden nodes is usually given. Thus the proposed OCI-ELM can be summarized as follows.

OCI-ELM Algorithm: Given a set of training data $\{(\mathbf{x}_i, t_i)\}_{i=1}^N \subset R^d \times R$, the activation function g , the maximum hidden node number $L_{\max}$ and the expected training error ε .

Step I. Initialization: Let $L = 0$ and the residual error $E = T$;

Step II. Learning step:

while $L < L_{\max}$ and $\|E\| > \varepsilon$.

(a) Increase by 1 the number of hidden nodes L : $L = L + 1$.

(b) Randomly assign hidden node parameters $(\mathbf{a}_L, b_L)$ for the new hidden node and calculate H .

(c) Calculate the coefficients α_L^* and β_L^* according to (4.4) for the new hidden node.

(d) Calculate the residual error after adding the new hidden node:

$$E = (1 - \alpha_L^*)T + \alpha_L^*E - \beta_L^*H.$$

end while.

Remark 4.1 *OCI-ELM is a generalization of both I-ELM and CI-ELM. In order for the comparison, the three incremental methods are listed respectively as follows:*

$$I\text{-ELM: } f_n(\mathbf{x}) = f_{n-1}(\mathbf{x}) + \beta_n g_n(\mathbf{x}),$$

$$CI\text{-ELM: } f_n(\mathbf{x}) = (1 - \beta_n)f_{n-1}(\mathbf{x}) + \beta_n g_n(\mathbf{x}),$$

$$OCI\text{-ELM: } f_n(\mathbf{x}) = \alpha_n f_{n-1}(\mathbf{x}) + \beta_n g_n(\mathbf{x}).$$

Obviously, I-ELM is a special case of OCI-ELM when $\alpha_n = 1$, and CI-ELM is a special case of OCI-ELM when α_n, β_n satisfy $\alpha_n + \beta_n = 1$. It also should be noted that the formula (3.2) for calculating β_n in I-ELM and the formula (3.5) for calculating β_n in CI-ELM can also be obtained based on the least-squares method. Without loss of generality, we take I-ELM as an example for the verification. Under this situation, let $F(\beta_n) = \|e_n\|^2$, then

$$\begin{aligned} F(\beta_n) &= \|f_{n-1} + \beta_n g_n - f\|^2 \\ &= \|f - f_{n-1}\|^2 + \beta_n^2 \|g_n\|^2 - 2\beta_n \langle f - f_{n-1}, g_n \rangle. \end{aligned}$$

According to the least-squares method, there is

$$\frac{\partial F(\beta_n)}{\partial \beta_n} = 2\beta_n \|g_n\|^2 - 2\beta_n \langle f - f_{n-1}, g_n \rangle = 0.$$

Solving for β_n , it can be then obtained that

$$\beta_n^* = \frac{\langle f - f_{n-1}, g_n \rangle}{\|g_n\|^2}$$

which is completely consistent with (3.2).

4.2 Experimental Verification

In this section, the performance of the proposed algorithm OCI-ELM is compared with I-ELM [51], EI-ELM [50] and CI-ELM [23] on six real-world function approximation problems from UCI database [8]. The specification of these benchmark problems are shown in Table 4.1.

Table 4.1: Specification of benchmark datasets

Problems	Attributes	Cases	Training Data	Testing Data
Abalone	8	4177	2000	2177
Boston housing	13	506	250	256
California housing	8	20640	8000	12640
Census(House8L)	8	22784	10000	12784
Delta ailerons	5	7129	3000	4129
Delta elevators	6	9517	4000	5517

In the experiments, all the inputs (attributes) have been normalized into the range $[-1, 1]$, while the outputs (targets) have been normalized into $[0, 1]$. Neural networks with sigmoid type of additive hidden nodes $G(\mathbf{a}, b, \mathbf{x}) = 1/(1 + \exp(-(\mathbf{a} \cdot \mathbf{x} + b)))$ are tested in all the algorithms. The hidden node parameters $(\mathbf{a}, b)$ are randomly chosen from the range $[-1, 1]^d \times [-1, 1]$ based on a uniform sampling distribution probability. All the simulations for I-ELM, EI-ELM, CI-ELM and the new algorithm OCI-ELM are carried out in MATLAB 7.4.0 environment and the same PC with Pentium 4 2.66GHZ CPU and 3GB RAM. For each problem, the average results of 20 trials are obtained for all algorithms. The apparent better results are shown in boldface.

With the same number of hidden nodes, the generalization performance of OCI-ELM, CI-ELM, EI-ELM and I-ELM are compared. The maximum number of hidden nodes enabled was 50, and no more hidden nodes were added when the whole training set was learned. Results shown in Table 4.2 are the average performance (testing root mean square error (RMSE), the standard deviation (Dev), and the training time) of different algorithms. Seen from Table 4.2, the testing RMSE of OCI-ELM is generally much smaller than that of CI-ELM and I-ELM for each case, and it remains smaller than that of EI-ELM only except the Boston Housing case. It further demonstrates that OCI-ELM may achieve much faster convergence rate than CI-ELM, EI-ELM and I-ELM with the same nodes. This fact can also be verified from Fig. 4.1 which shows the average testing RMSE decreasing trends of the four algorithms for Abalone case respectively. It can be seen that in order to obtain the same testing RMSE 0.057, OCI-ELM only needs 7 nodes while CI-ELM needs 22 nodes, EI-ELM and I-ELM need beyond 50 nodes. Therefore, OCI-ELM obtains the most parsimonious network architecture among all algorithms. Similar curves could be found for other regression cases.

Figure 4.1: The testing RMSE updating curves of OCI-ELM, CI-ELM, EI-ELM and I-ELM for Abalone case

Next, how the coefficients α_n and β_n vary as the new hidden nodes are added incrementally in OCI-ELM is investigated. Fig. 4.2 shows the updating curves of α_n and β_n respectively.

Figure 4.2: The updating curves of α_n and β_n in OCI-ELM

Seen from Fig. 4.2, with the addition of hidden nodes, the coefficient α_n of the existing hidden nodes approaches to 1, while the coefficient β_n of the newly added hidden node goes to 0. Then combining with the incremental formula $f_n(\mathbf{x}) = \alpha_n f_{n-1}(\mathbf{x}) + \beta_n g_n(\mathbf{x})$ of OCI-ELM, it can be obviously observed that the network sequence $\{f_n(\mathbf{x})\}$ will be unchanged as the hidden nodes are added up to a certain level. This observation, to some extent, verifies the universal approximation capability of the proposed OCI-ELM from another point of view.

4.3 Summary

Motivated by the latest theory [49] and [51], this chapter has proposed the OCI-ELM where an optimal combination between the existing hidden nodes and the newly added hidden node can be obtained. Different from the incremental method $f_n = f_{n-1} + \beta_n g_n$ in I-ELM and the convex incremental method $f_n = (1 - \beta_n)f_{n-1} + \beta_n g_n$ in CI-ELM, OCI-ELM presents a new incremental method $f_n = \alpha_n f_{n-1} + \beta_n g_n$ whose linear coefficients α_n and β_n are adjusted according to the least-squares method so that the residual error can be minimized. As a consequence, OCI-ELM is a generalization of both I-ELM and CI-ELM. On the basis of the theory obtained in [51], it has also been proved that such OCI-ELM can approximate any continuous target function if the hidden node activation function is bounded nonconstant piecewise continuous and $\text{span}\{G(\mathbf{a}, b, \mathbf{x})\}$ is dense in L^2 . Compared with the existing algorithms I-ELM, EI-ELM and CI-ELM, OCI-ELM can obtain a better generalization performance at faster convergence speed and with more compact network architecture.

Table 4.2: Performance comparison of OCI-ELM, CI-ELM, EI-ELM and I-ELM with the same number of hidden nodes

Problems	I-ELM			EI-ELM			CI-ELM			OCI-ELM		
	Testing RMSE		Training Times(s)	Testing RMSE		Training Times(s)	Testing RMSE		Training Times(s)	Testing RMSE		Training Times(s)
	Mean	Dev		Mean	Dev		Mean	Dev		Mean	Dev	
Abalone	0.0998	0.0071	0.0125	0.0885	0.0037	0.1521	0.0922	0.0029	0.0156	0.0856	0.0025	0.0226
Boston Housing	0.16	0.0179	0.0062	0.1151	0.0075	0.071	0.1351	0.0081	0.007	0.122	0.0064	0.0086
California Housing	0.2166	0.0146	0.0406	0.1605	0.0045	0.4547	0.197	0.0044	0.0452	0.1594	0.0065	0.0671
Census(House8L)	0.1003	0.003	0.0421	0.0901	0.0017	0.5405	0.0934	0.002	0.0507	0.0891	0.0023	0.0811
Delta Ailerons	0.0772	0.0172	0.014	0.0448	0.0036	0.1794	0.0421	0.0012	0.0133	0.0404	0.0005	0.032
Delta Elevators	0.0892	0.0183	0.0203	0.0575	0.0029	0.2114	0.0563	0.0011	0.014	0.0541	0.0005	0.0281

Chapter 5

Extreme Learning Machine with Adaptive Growth of Hidden Nodes (AG-ELM)

As stated in the proceeding chapters, finding a suitable network architecture for ELMs is recognized to be crucial. Two different approaches, constructive algorithms and pruning algorithms, are mostly pursued for solving this problem. Although both approaches address the problem of network acquisition, they investigate only restricted topological subsets rather than the complete class of network architectures [29]. Therefore, exploring an approach for the automatic design of networks in a search space of possible structures which are suitable to the problem, and meanwhile, showing its universal approximation capability, are the main objectives of this chapter.

This chapter first proposes an extreme learning machine with adaptive growth of hidden nodes (AG-ELM) where a novel approach to determining the network architecture is introduced. Then the universal approximation capability of such AG-ELM is

The main ideas of this chapter can be found in “Universal approximation of extreme learning machine with adaptive growth of hidden nodes,” IEEE Transactions on Neural Networks and Learning Systems, vol. 23, no. 2, pp. 365 -371, 2012.

shown in the following sense: (1) the obtained AG-ELM can approximate any Lebesgue p -integrable function, not merely restricted to continuous functions in most literatures; (2) the hidden activation functions used in AG-ELM can be Lebesgue p -integrable, which is a much weaker condition than those assumed before.

5.1 Preliminary Mathematical Aspects

This section focuses on the continuity of the function $\beta G(\mathbf{a}, b, \mathbf{x}) = \beta g(h(\mathbf{a}, b, \mathbf{x}))$ with respect to the parameters $(\mathbf{a}, b, \beta)$ in the case where the hidden activation function g is Lebesgue p -integrable (say, $(\int_X |g(\mathbf{x})|^p d\mathbf{x})^{\frac{1}{p}} < +\infty$, denoted by $g \in L^p(X)$) on a compact set $X \subset R^d$. This fact is a necessary preliminary for the implementation and theoretical analysis of the proposed AG-ELM. Here, the function $h : R^{d+2} \rightarrow R$ expresses a relationship among the hidden node parameters $\mathbf{a}$, b and the input $\mathbf{x}$, which can be neuron alike (e.g. additive nodes $h(\mathbf{a}, b, \mathbf{x}) = \mathbf{a} \cdot \mathbf{x} + b$ or RBF nodes $h(\mathbf{a}, b, \mathbf{x}) = b \parallel \mathbf{x} - \mathbf{a} \parallel$ [51]) or non-neuron alike (e.g. fuzzy rules [54]). The one-dimensional case where $x \in R$ is first considered.

Lemma 5.1 [113] *Let j be a measurable function defined on an interval $[a, b]$, and assume that j takes the values $\pm\infty$ only on a set of measure zero. Then given $\varepsilon > 0$, we can find a continuous function k such that*

$$|j - k| < \varepsilon$$

except on a set of measure less than ε , i.e., $m\{x : |j(x) - k(x)| \geq \varepsilon\} < \varepsilon$. If in addition $m < j < M$, then we may choose the function k so that $m < k < M$.

Theorem 5.1 *Suppose that $g : R \rightarrow R$ is Lebesgue p -integrable. Then the function*

$\beta G(a, b, x)$ is continuous with respect to the parameters (a, b, β) in L^p metric, that is,

$$\lim_{(a,b,\beta) \rightarrow (a_0,b_0,\beta_0)} \|\beta G(a, b, x) - \beta_0 G(a_0, b_0, x)\|_p = 0$$

for any $x \in R$, $(a_0, b_0, \beta_0) \in R \times R \times R$ and $h : R^3 \rightarrow R$.

Proof: In accordance with the properties of Lebesgue integral, given any $\varepsilon_1 > 0$, there exist large enough positive numbers M and N such that

$$\|g(u) - g_1(u)\|_p < \frac{\varepsilon_1}{12} \quad (5.1)$$

where $\|\cdot\|_p = (\int_R |\cdot|^p dx)^{\frac{1}{p}}$ denotes the p -norm in Lebesgue metric and

$$g_1(u) = \begin{cases} \min\{g(u), N\}, & u \in [-M, M] \text{ and } g(u) \geq 0 \\ \max\{g(u), -N\}, & u \in [-M, M] \text{ and } g(u) < 0 \\ 0, & u > M \text{ or } u < -M. \end{cases}$$

By Lemma 5.1, for any positive number $\varepsilon_2 > 0$, there exists a continuous function $k_1(u)$ defined on $[-M, M]$ such that

$$\|g_1(u) - k_1(u)\|_p < \frac{\varepsilon_2}{12} \quad \text{and} \quad \|k_1(u)\|_p \leq N$$

except on a set of measure with small enough value. Let

$$k(u) = \begin{cases} k_1(u), & u \in [-M, M] \\ l_1(u), & u \in [-M', -M] \\ l_2(u), & u \in [M, M'] \\ 0, & u \notin [-M', M'] \end{cases}$$

where $l_1(u)$, $l_2(u)$ are two straight lines connecting the points $(-M, k_1(-M))$, $(-M', 0)$ as well as the points $(M, k_1(M))$, $(M', 0)$ respectively. The scalar M' is close enough to M with $M' > M$. Then $k(u)$ is continuous on R and satisfies

$$\|g_1(u) - k(u)\|_p < \frac{\varepsilon_3}{12}. \quad (5.2)$$

Here ε_3 is another positive number.

Let $\varepsilon = \max\{\varepsilon_1, \varepsilon_3\}$, connecting with the relationships (5.1) and (5.2), then there is

$$\|g(u) - k(u)\|_p < \frac{\varepsilon}{6}$$

for any $u \in R$. Denote by $u = h(a, b, x) \in R$, it can be obtained that

$$\|g(h(a_0, b, x)) - k(h(a_0, b, x))\|_p < \frac{\varepsilon}{6}$$

and

$$\|g(h(a_0, b_0, x)) - k(h(a_0, b_0, x))\|_p < \frac{\varepsilon}{6}$$

hold for all $h(a_0, b, x), h(a_0, b_0, x) \in R$. Furthermore, due to the uniform continuity of $k(u)$ on the closed interval $[-M', M']$ and the fact that $k(u) \equiv 0$ when $u \notin [-M', M']$, it is known that for $\varepsilon > 0$, there exists a positive number $\delta_1 > 0$ such that for all b and b_0

with $|b - b_0| < \delta_1$, there is

$$\begin{aligned} & \|k(h(a_0, b, x)) - k(h(a_0, b_0, x))\|_p \\ &= \left(\int_R |k(h(a_0, b, x)) - k(h(a_0, b_0, x))|^p dx \right)^{\frac{1}{p}} < \frac{\varepsilon}{6}. \end{aligned}$$

By combining the proceeding three relationships and Minkovski inequality [85], it can be obtained that

$$\begin{aligned} & \|G(a_0, b, x) - G(a_0, b_0, x)\|_p \\ &= \|g(h(a_0, b, x)) - g(h(a_0, b_0, x))\|_p \\ &\leq \|g(h(a_0, b, x)) - k(h(a_0, b, x))\|_p \\ &\quad + \|g(h(a_0, b_0, x)) - k(h(a_0, b_0, x))\|_p \\ &\quad + \|k(h(a_0, b, x)) - k(h(a_0, b_0, x))\|_p \\ &< \frac{\varepsilon}{6} + \frac{\varepsilon}{6} + \frac{\varepsilon}{6} = \frac{\varepsilon}{2} \end{aligned} \tag{5.3}$$

which shows the continuity of the function $G(a, b, x)$ with respect to the parameter b .

By using the same method, it can be proved that there is a positive number $\delta_2 > 0$ such that for all a and a_0 with $|a - a_0| < \delta_2$, there holds

$$\|G(a, b, x) - G(a_0, b, x)\|_p < \frac{\varepsilon}{2}, \tag{5.4}$$

in other words, the function $G(a, b, x)$ is continuous with respect to the parameter a .

Finally, the function $\beta G(a, b, x)$ is also proved to be continuous with respect to the parameter β . In accordance with the continuity of $G(a, b, x)$ proved above, it is known that $G(a, b, x)$ is bounded in the sense that $\|G(a, b, x)\|_p \leq C$ where C is a positive scalar.

Taking $\delta_3 = \frac{\varepsilon}{C}$, then for all $\beta \in R$ with $|\beta - \beta_0| < \delta_3$, there is

$$\|\beta G(a, b, x) - \beta_0 G(a, b, x)\|_p \leq |\beta - \beta_0| \|G(a, b, x)\|_p < \varepsilon. \quad (5.5)$$

In general, let $\delta = \min\{\delta_1, \delta_2, \delta_3\}$, then combining with the inequalities (5.3), (5.4) and (5.5), it can be concluded that for all $(a, b, \beta) \in R \times R \times R$ with $|(a, b, \beta) - (a_0, b_0, \beta_0)| < \sqrt{3}\delta$, we have

$$\begin{aligned} & \|\beta G(a, b, x) - \beta_0 G(a_0, b_0, x)\|_p \\ & \leq \|\beta G(a, b, x) - \beta_0 G(a, b, x)\|_p + \|\beta_0 G(a, b, x) - \beta_0 G(a_0, b, x)\|_p \\ & \quad + \|\beta_0 G(a_0, b, x) - \beta_0 G(a_0, b_0, x)\|_p \\ & < \varepsilon + |\beta_0| \left(\frac{\varepsilon}{2} + \frac{\varepsilon}{2}\right) = (1 + |\beta_0|)\varepsilon. \end{aligned}$$

This completes the proof of Theorem 5.1.

It will be next shown that in the multi-dimensional case where $\mathbf{x} \in X \subset R^d$, the function $\beta G(\mathbf{a}, b, \mathbf{x})$ is continuous with respect to the parameters $(\mathbf{a}, b, \beta)$ when X is compact.

Lemma 5.2 [113] *Let $k(x, y)$ is an integrable function in $X \times Y$. Then the functions $\int_Y k(x, y) d\nu(y)$ and $\int_X k(x, y) d\mu(x)$ are both integrable functions on X and Y for almost all x and y respectively, and*

$$\int_{X \times Y} k(x, y) d(\mu \times \nu) = \int_X \left(\int_Y k(x, y) d\nu \right) d\mu = \int_Y \left(\int_X k(x, y) d\mu \right) d\nu.$$

Theorem 5.2 *Suppose that $g : R \rightarrow R$ is Lebesgue p -integrable and $X \subset R^d$ is compact, then $\beta G(\mathbf{a}, b, \mathbf{x})$ is continuous with respect to the parameters $(\mathbf{a}, b, \beta)$ in L^p metric, that*

is,

$$\lim_{(\mathbf{a}, b, \beta) \rightarrow (\mathbf{a}_0, b_0, \beta_0)} \|\beta G(\mathbf{a}, b, \mathbf{x}) - \beta_0 G(\mathbf{a}_0, b_0, \mathbf{x})\|_p = 0$$

for any $\mathbf{x} \in X$, $(\mathbf{a}, b, \beta) \in R^d \times R \times R$ and $h : R^{d+2} \rightarrow R$.

Proof: The case where $d = 2$ is first considered.

Suppose that $\mathbf{x} = (x_1, x_2)^T \in X_1 \times X_2 \subset R^2$, $\mathbf{a} = (a_1, a_2)^T$ and $\mathbf{a}_0 = (a_{10}, a_{20})^T$ where X_1, X_2 are two compact sets. Then according to Lemma 5.2, there is

$$\begin{aligned} & \|\beta G(\mathbf{a}, b, \mathbf{x}) - \beta_0 G(\mathbf{a}_0, b_0, \mathbf{x})\|_p \\ &= \|\beta g(h(\mathbf{a}, b, \mathbf{x})) - \beta_0 g(h(\mathbf{a}_0, b_0, \mathbf{x}))\|_p \\ &= \|\beta g(h'(a_1, a_2, b, x_1, x_2)) - \beta_0 g(h'(a_{10}, a_{20}, b_0, x_1, x_2))\|_p \\ &= \left(\int_{X_1 \times X_2} |\beta g(h'(a_1, a_2, b, x_1, x_2)) - \beta_0 g(h'(a_{10}, a_{20}, b_0, x_1, x_2))|^p dx_1 dx_2 \right)^{\frac{1}{p}} \\ &= \left(\int_{X_1} \left(\int_{X_2} |\beta g(h'(a_1, a_2, b, x_1, x_2)) - \beta_0 g(h'(a_{10}, a_{20}, b_0, x_1, x_2))|^p dx_2 \right) dx_1 \right)^{\frac{1}{p}} \end{aligned}$$

where $h'(a_1, a_2, b, x_1, x_2)$ is the expansion of $h(\mathbf{a}, b, \mathbf{x})$ in two-dimensional case. By using Theorem 5.1 and the compactness of X_2 , given $\tilde{\varepsilon} > 0$, there is a positive number $\delta > 0$, for any fixed point $x_1 \in X_1$, there holds

$$\int_{X_2} |\beta g(h'(a_1, a_2, b, x_1, x_2)) - \beta_0 g(h'(a_{10}, a_{20}, b_0, x_1, x_2))|^p dx_2 < \tilde{\varepsilon}^p$$

for all $(\mathbf{a}, b, \beta)$ and $(\mathbf{a}_0, b_0, \beta_0)$ with $|(\mathbf{a}, b, \beta) - (\mathbf{a}_0, b_0, \beta_0)| < \delta$. Furthermore, due to the compactness of X_1 , it can be then obtained that

$$\begin{aligned} & \left(\int_X |\beta G(\mathbf{a}, b, \mathbf{x}) - \beta_0 G(\mathbf{a}_0, b_0, \mathbf{x})|^p d\mathbf{x} \right)^{\frac{1}{p}} \\ &= \left(\int_{X_1} \left(\int_{X_2} |\beta g(h'(a_1, a_2, b, x_1, x_2)) - \beta_0 g(h'(a_{10}, a_{20}, b_0, x_1, x_2))|^p dx_2 \right) dx_1 \right)^{\frac{1}{p}} < \varepsilon \end{aligned}$$

holds for all $(\mathbf{a}, b, \beta)$ and $(\mathbf{a}_0, b_0, \beta_0)$ with $|(\mathbf{a}, b, \beta) - (\mathbf{a}_0, b_0, \beta_0)| < \delta$. Note that $\varepsilon = \left(\int_{X_1} \tilde{\varepsilon}^p dx_1 \right)^{\frac{1}{p}}$.

For the case where $d > 2$, the conclusion can be proved immediately by using the mathematical induction method.

(i) Theorem 5.2 does hold when $d = 2$ as proved above.

(ii) Suppose that Theorem 5.2 holds when $d = m$, that is,

$$\lim_{(\mathbf{a}^{(m)}, b, \mathbf{x}^{(m)}) \rightarrow \beta_0 G(\mathbf{a}_0^{(m)}, b_0, \mathbf{x}^{(m)})} \|\beta G(\mathbf{a}^{(m)}, b, \mathbf{x}^{(m)}) - \beta_0 G(\mathbf{a}_0^{(m)}, b_0, \mathbf{x}^{(m)})\|_p = 0$$

where $\mathbf{x}^{(m)} = (x_1, x_2, \dots, x_m)^T \in \mathbf{X}^{(m)}$, $\mathbf{a}^{(m)} = (a_1, a_2, \dots, a_m)^T$ and $\mathbf{a}_0^{(m)} = (a_{10}, a_{20}, \dots, a_{m0})^T$. Here, $\mathbf{X}^{(m)} = X_1 \times X_2 \cdots \times X_m \subset R^m$ is compact. Note that the superscript (m) denotes the m -dimensions of vectors or space.

For $d = m + 1$, let $\mathbf{x}^{(m+1)} = (\mathbf{x}^{(m)T}, x_{m+1})^T \in \mathbf{X}^{(m)} \times X_{m+1} \subset R^{m+1}$, $\mathbf{a}^{(m+1)} = (\mathbf{a}^{(m)T}, a_{m+1})^T$ and $\mathbf{a}_0^{(m+1)} = (\mathbf{a}_0^{(m)T}, a_{m+1,0})^T$. Then

$$\begin{aligned} & \|\beta G(\mathbf{a}^{(m+1)}, b, \mathbf{x}^{(m+1)}) - \beta_0 G(\mathbf{a}_0^{(m+1)}, b_0, \mathbf{x}^{(m+1)})\|_p \\ &= \left(\iint_{\mathbf{X}^{(m)} \times X_{m+1}} |\beta g(h'(\mathbf{a}^{(m)T}, a_{m+1}, b, \mathbf{x}^{(m)T}, x_{m+1})) \right. \\ & \quad \left. - \beta_0 g(h'(\mathbf{a}_0^{(m)T}, a_{m+1,0}, b_0, \mathbf{x}^{(m)T}, x_{m+1}))|^p d\mathbf{x}^{(m)} dx_{m+1} \right)^{\frac{1}{p}} \end{aligned}$$

where $h'(\mathbf{a}^{(m)T}, a_{m+1}, b, \mathbf{x}^{(m)T}, x_{m+1})$ is the expansion of $h(\mathbf{a}, b, \mathbf{x})$ in $(m+1)$ -dimensional case. According to the assumption for the case of $d = m$, given $\bar{\varepsilon} > 0$, there is a positive number $\bar{\delta} > 0$, for any fixed point $x_{m+1} \in X_{m+1}$, there holds

$$\begin{aligned} & \left(\int_{\mathbf{X}^{(m)}} | \beta g(h'(\mathbf{a}^{(m)}, a_{m+1}, b, \mathbf{x}^{(m)}, x_{m+1})) \right. \\ & \quad \left. - \beta_0 g(h'(\mathbf{a}_0^{(m)}, a_{m+1,0}, b_0, \mathbf{x}^{(m)}, x_{m+1})) |^p dX_m \right)^{\frac{1}{p}} \\ & < \bar{\varepsilon} \end{aligned}$$

for all $(\mathbf{a}^{(m)}, b, \beta)$ and $(\mathbf{a}_0^{(m)}, b_0, \beta_0)$ with $|(\mathbf{a}^{(m)}, b, \beta) - (\mathbf{a}_0^{(m)}, b_0, \beta_0)| < \bar{\delta}$. Then similar as the proof for the case where $d = 2$, according to Lemma 5.2, we have

$$\begin{aligned} & \| \beta G(\mathbf{a}^{(m+1)}, b, \mathbf{x}^{(m+1)}) - \beta_0 G(\mathbf{a}_0^{(m+1)}, b_0, \mathbf{x}^{(m+1)}) \|_p \\ &= \left(\int_{X_{m+1}} \left(\int_{\mathbf{X}^{(m)}} | \beta g(h'(\mathbf{a}^{(m)T}, a_{m+1}, b, \mathbf{x}^{(m)T}, x_{m+1})) \right. \right. \\ & \quad \left. \left. - \beta_0 g(h'(\mathbf{a}_0^{(m)T}, a_{m+1,0}, b_0, \mathbf{x}^{(m)T}, x_{m+1})) |^p d\mathbf{x}^{(m)} \right) dx_{m+1} \right)^{\frac{1}{p}} \\ &\leq \left(\int_{X_{m+1}} \bar{\varepsilon}^p dx_{m+1} \right)^{\frac{1}{p}} = \varepsilon \end{aligned}$$

holds for all $(\mathbf{a}^{(m)}, b, \beta)$ and $(\mathbf{a}_0^{(m+1)}, b_0, \beta_0)$ with $|(\mathbf{a}^{(m+1)}, b, \beta) - (\mathbf{a}_0^{(m+1)}, b_0, \beta_0)| < \bar{\delta}$.

From (i) and (ii), Theorem 5.2 does hold for all integer d by induction on m .

The following notations are used in the subsequent analysis:

$$\begin{aligned} \theta &= (\mathbf{a}, b, \beta), \quad \Theta^n = (\theta_1, \theta_2, \dots, \theta_n), \\ \mathcal{X} &= R^d \times R \times R, \quad \mathcal{X}^n = \mathcal{X} \times \mathcal{X} \times \dots \times \mathcal{X} \end{aligned}$$

where $\theta_i = (\mathbf{a}_i, b_i, \beta_i), i = 1, 2, \dots, n$.

Definition 5.1 Let $\theta = (\mathbf{a}, b, \beta)$ be a random variable on $\mathcal{X}$ and $P(x)$ denote the probability density of θ . Then the random variable θ is said to be distributed pseudo-uniformly if $P(x) \neq 0$ for every x on $\mathcal{X}$.

Equivalently, if θ is distributed pseudo-uniformly, then there must have $P\{(\mathbf{a}, b, \beta) \in \Xi\} > 0$ for any open set Ξ on $\mathcal{X}$. On the basis of this fact, it can be stated that if a random variable is distributed based on a continuous probability, then it must be distributed pseudo-uniformly.

Remark 5.1 With the above notations, denote by $\Theta^n \in R^{n(d+2)}$ all the parameters of an SLFN with n additive hidden nodes $f_n(\mathbf{x}) = \sum_{i=1}^n \beta_i G(\mathbf{a}_i, b_i, \mathbf{x})$. Given parameters Θ_0^n , denote the corresponding SLFN by $f_n^0(\mathbf{x})$. Theorem 5.1 and Theorem 5.2 reveal that for any positive number $\varepsilon > 0$, there exists an open set $\Xi_0 \subset R^{n(d+2)}$ including Θ_0^n such that $\|f_n(\mathbf{x}) - f_n^0(\mathbf{x})\| < \varepsilon$ for all $\mathbf{x} \in X \subset R^d$ when Θ^n varies in Ξ_0 and X is compact. This fact is a necessary mathematical aspect to guarantee that the parameters of SLFNs can be randomly generated based on the pseudo-uniform distribution.

Remark 5.2 Note that Lemma II.5 in [51] showed the continuity of the function $G(\mathbf{a}, b, \mathbf{x})$ with the additive hidden nodes $G(\mathbf{a}, b, \mathbf{x}) = g(\mathbf{a} \cdot \mathbf{x} + b)$ and RBF nodes $G(\mathbf{a}, b, \mathbf{x}) = g(\frac{\|\mathbf{x} - \mathbf{a}\|}{b})$ with respect to the parameters $(\mathbf{a}, b)$ under the assumption that the activation functions g is bounded nonconstant piecewise continuous. From a mathematical point of view, it is known that a bounded nonconstant piecewise continuous function must be Riemann integrable. However, this chapter assumes that g is Lebesgue p -integrable, which is obviously a much weaker condition than that given in [51]. Thus more functions can be used as the activation functions of ELMs. Moreover, the results obtained are tenable for the GSLFNs rather than the conventional SLFNs with additive/RBF hidden nodes as stated in [51].

5.2 The Proposed AG-ELM and Its Approximation Capability

This section will first introduce the AG-ELM. Then it will be proved in theory that such AG-ELM with Lebesgue p -integrable hidden activation functions can approximate any Lebesgue p -integrable target function $f(\mathbf{x})$ on a compact set X with arbitrarily small error as long as $\text{span}\{G(\mathbf{a}, b, \mathbf{x}) : (\mathbf{a}, b) \in R^d \times R\}$ is dense in L^p .

The details of AG-ELM are presented as follows.

First, randomly generate a group of parameters $\theta_1^{(1)} = (\mathbf{a}_1^{(1)}, b_1^{(1)}, \beta_1^{(1)})$ from $\mathcal{X}$ based on a pseudo-uniform distribution. Let

$$\varphi_1^{(1)} = \beta_1^{(1)} G(\mathbf{a}_1^{(1)}, b_1^{(1)}, \mathbf{x}).$$

Second, randomly generate two groups of parameters $(\theta_1^{(2)}, \theta_2^{(2)}) = ((\mathbf{a}_1^{(2)}, b_1^{(2)}, \beta_1^{(2)}), (\mathbf{a}_2^{(2)}, b_2^{(2)}, \beta_2^{(2)}))$ simultaneously from $\mathcal{X}^2$ based on a pseudo-uniform distribution. Write $\tilde{\varphi}_1^{(2)} = \beta_1^{(2)} G(\mathbf{a}_1^{(2)}, b_1^{(2)}, \mathbf{x})$ and $\tilde{\varphi}_2^{(2)} = \beta_1^{(2)} G(\mathbf{a}_1^{(2)}, b_1^{(2)}, \mathbf{x}) + \beta_2^{(2)} G(\mathbf{a}_2^{(2)}, b_2^{(2)}, \mathbf{x})$. Let

$$\varphi_1^{(2)} = \begin{cases} \varphi_1^{(1)}, & \text{if } \|\varphi_1^{(1)} - f\|_p \leq \|\tilde{\varphi}_1^{(2)} - f\|_p \\ \tilde{\varphi}_1^{(2)}, & \text{otherwise} \end{cases}$$

and

$$\varphi_2^{(2)} = \begin{cases} \varphi_1^{(2)}, & \text{if } \|\varphi_1^{(2)} - f\|_p \leq \|\tilde{\varphi}_2^{(2)} - f\|_p \\ \tilde{\varphi}_2^{(2)}, & \text{otherwise.} \end{cases}$$

Similarly, at the s th step, s functions are obtained:

$$\tilde{\varphi}_1^{(s)} = \beta_1^{(s)} G(\mathbf{a}_1^{(s)}, b_1^{(s)}, \mathbf{x}), \tilde{\varphi}_2^{(s)} = \sum_{i=1}^2 \beta_i^{(s)} G(\mathbf{a}_i^{(s)}, b_i^{(s)}, \mathbf{x}), \dots, \tilde{\varphi}_s^{(s)} = \sum_{i=1}^s \beta_i^{(s)} G(\mathbf{a}_i^{(s)}, b_i^{(s)}, \mathbf{x})$$

by randomly generating the parameters $(\theta_1^{(s)}, \theta_2^{(s)}, \dots, \theta_s^{(s)})$ from $\mathcal{X}^s$ based on a pseudo-uniform distribution. Let

$$\varphi_1^{(s)} = \begin{cases} \varphi_{s-1}^{(s-1)}, & \text{if } \|\varphi_{s-1}^{(s-1)} - f\|_p \leq \|\tilde{\varphi}_1^{(s)} - f\|_p \\ \tilde{\varphi}_1^{(s)}, & \text{otherwise} \end{cases}$$

...

$$\varphi_s^{(s)} = \begin{cases} \varphi_{s-1}^{(s)}, & \text{if } \|\varphi_{s-1}^{(s)} - f\|_p \leq \|\tilde{\varphi}_s^{(s)} - f\|_p \\ \tilde{\varphi}_s^{(s)}, & \text{otherwise.} \end{cases}$$

Continuing the above processes, a function sequence $\{\varphi_1^{(1)}, \varphi_1^{(2)}, \varphi_2^{(2)}, \dots, \varphi_1^{(s)}, \varphi_2^{(s)}, \dots, \varphi_s^{(s)}, \dots, \varphi_l^{(m)}, \dots\}$ is then obtained. Write $\Phi_n = \varphi_l^m$ where $n = \frac{1}{2}(m-1)(m-2) + l$. Then $\{\Phi_n(\mathbf{x})\}$ is the AG-ELM we are looking for and satisfies $\|\Phi_n - f\| \leq \|\Phi_{n-1} - f\|$ for every n .

The above processes of obtaining the network sequence by AG-ELM can be illustrated as Figure 5.1.

Remark 5.3 *Different from the constructive or pruning algorithms, AG-ELM starts with a small network and changes the hidden node number in an adaptive way. From the approach to constructing AG-ELM described above, it can be observed that the networks in AG-ELM are generated in group at each step. Specifically, a candidate reservoir is first generated, which is composed of the last network obtained at the previous step and a group of newly generated networks with different numbers of randomly hidden nodes. Then, guided by the problem being solved, the performance of the all the networks in the candidate reservoir are evaluated and compared one by one. In each comparison, the one leading to the best generalization performance will be chosen. An advantage of doing so is that the existing networks may be replaced by newly generated networks which have less hidden node number and better generalization performance. That is to*

Figure 5.1: The procedure of obtaining the network sequence at the s th step of AG-ELM

say, at each step, one or more new hidden nodes may be added, and at the same time, one or more existing hidden nodes may be removed. The removed nodes may be more or less than, or equal to the added nodes. Therefore, the hidden node numbers of the networks obtained by AG-ELM change dynamically. Another key contribution of AG-ELM is that all the parameters of AG-ELM are randomly generated in the whole parameter space $R^d \times R \times R$ without any restriction so that it provides an unlimited search space of the available network architectures in some way.

Theorem 5.3 *Let $g : R \rightarrow R$ be a Lebesgue p -integrable hidden activation function defined on the compact set $X \subset R^d$. If $\text{span}\{G(\mathbf{a}, b, \mathbf{x}) : (\mathbf{a}, b) \in R^d \times R\}$ is dense in $L^p(X)$, then for any Lebesgue p -integrable function $f(\mathbf{x})$ and any AG-ELM $\{\Phi_n(\mathbf{x})\}$, $\lim_{n \rightarrow \infty} \|\Phi_n - f\| = 0$ holds with probability one.*

Proof. According to the assumption that $\text{span}\{G(\mathbf{a}, b, \mathbf{x}) : (\mathbf{a}, b) \in R^d \times R\}$ is dense in $L^p(X)$ and $f \in L^p(X)$, given any positive number ε , there exists a generalized SLFN $f_{n(\varepsilon)}^0 = \sum_{i=1}^{n(\varepsilon)} \beta_i^0 G(\mathbf{a}_i^0, b_i^0, \mathbf{x})$ such that

$$\|f_{n(\varepsilon)}^0 - f\| < \frac{\varepsilon}{2} \quad (5.6)$$

where $n(\varepsilon)$ is a positive integer. Write the parameters of $f_{n(\varepsilon)}^0$ as $\Theta_0^{n(\varepsilon)}$, then by using Theorem 5.2, there exists an open set $\Xi_\varepsilon \subset \mathcal{X}^{n(\varepsilon)}$ including $\Theta_0^{n(\varepsilon)}$ such that

$$\|f_{n(\varepsilon)} - f_{n(\varepsilon)}^0\| < \frac{\varepsilon}{2} \quad (5.7)$$

as $\Theta^{n(\varepsilon)}$ vary in Ξ_ε where $\Theta^{n(\varepsilon)}$ denote the parameters of the generalized SLFN $f_{n(\varepsilon)}$ with $n(\varepsilon)$ hidden nodes. Combining with the inequalities (5.6) and (5.7) and using

Minkovski inequality, we know that

$$\|f_{n(\varepsilon)} - f\| < \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon$$

holds for any $f_{n(\varepsilon)}$ as its parameters $\Theta^{n(\varepsilon)}$ vary in Ξ_ε .

Moreover, in accordance with the method to construct AG-ELM $\{\Phi_n\}$, it is known that there have infinite number of networks with $n(\varepsilon)$ hidden nodes in $\{\Phi_n\}$ and there must exist at least one such network whose parameters belong in Ξ_ε because AG-ELM is randomly generated from $\mathcal{X}^{n(\varepsilon)}$ based on the pseudo-uniform distribution. This fact obviously shows that

$$\lim_{n \rightarrow \infty} \|\Phi_n - f\| = \sup_n \inf_{k > n} \|\Phi_k - f\| < \varepsilon$$

holds with probability one. Since AG-ELM $\{\Phi_n\}$ is decreasing in the sense that $\|\Phi_n - f\| \leq \|\Phi_{n-1} - f\|$, it is then proved that

$$\lim_{n \rightarrow \infty} \|\Phi_n - f\| = 0$$

holds with probability one.

5.3 Experimental Verification

This section investigates the performance of the proposed algorithm AG-ELM in some benchmark applications. The simulations are running in MATLAB 7.6 environment and the same PC with Intel(R) Core(TM)2 2.20GHZ CPU and 1G RAM. The proposed

AG-ELM is compared with the original I-ELM on benchmark regression problems from UCI database [8]. The specification of these benchmark problems are shown in Table 5.1.

Table 5.1: Specification of benchmark datasets

Problems	Attributes	Cases	Training Data	Testing Data
Abalone	8	4177	2000	2177
Boston Housing	13	506	250	256
California Housing	8	20640	8000	12640
Census(House8L)	8	22784	10000	12784
Delta Ailerons	5	7129	3000	4129
Delta Elevators	6	9517	4000	5517
Machine CPU	6	209	100	109

In the experiments, all the inputs (attributes) have been normalized into the range $[-1, 1]$ while the outputs (targets) have been normalized into $[0, 1]$. Neural networks with sigmoid type of additive hidden nodes $G(\mathbf{a}, b, \mathbf{x}) = \frac{1}{1 + e^{-(\mathbf{a} \cdot \mathbf{x} + b)}}$ are tested in both AG-ELM and I-ELM. In I-ELM, the hidden node parameters $\mathbf{a}$ and b are randomly chosen from the range $[-1, 1]$ based on a uniform sampling distribution probability and the output weights β is calculated analytically according to (3.2). In AG-ELM, all the parameters $\mathbf{a}, b$ and β are randomly chosen from the range $[-1, 1]$. For each problem, the average results over 20 trials are obtained for AG-ELM and I-ELM. The apparent better results are shown in boldface and the close results are underlined.

Given the expected accuracy (or the same maximum step), the network complexity and the generalization performance of AG-ELM and I-ELM are compared. The stopping RMSE are presented in Table 5.2 and the maximum step is set to 200 for all the seven regression data sets. Seen from Table 5.2, for each case, the average hidden node number of AG-ELM is quite smaller than that of I-ELM. On the other hand, the hidden node numbers of I-ELM obtained in different trials are very unstable since the deviations of hidden node number are fairly large. Differently, the less value of the hidden node number deviation of AG-ELM illustrates that the proposed AG-ELM is robust in the aspect of determining the network architecture. Meanwhile, the testing root mean square errors (RMSEs) of AG-ELM are smaller or comparable with those of I-ELM for

each case. It also should be noted that the iterations required of AG-ELM is smaller than those of I-ELM in three cases (i.e., California Housing, Delta Ailerons, Delta Elevators), while in the other four cases, they are larger than those of I-ELM. Besides this, at each iteration, AG-ELM needs to first generate a candidate reservoir, and then evaluate and compare the performance of all the networks in the candidate reservoir so as to find the most parsimonious network architectures. Hence, AG-ELM spend more time than I-ELM in all the simulations. However, from network architecture point of view, the network architecture of AG-ELM is more compact than that of I-ELM. Therefore, AG-ELM actually provides tradeoff solutions if users wish to have more compact network architecture in their ELM applications while keeping the better or comparable generalization performance.

Table 5.2: Generalization performance comparison of AG-ELM and I-ELM with the same expected accuracy

Datasets	Stop RMSE	Max Steps	Algorithms	Testing RMSE	Testing Dev	Training Time (s)	#Nodes	NodeDev	Iterations
Abalone	0.12	200	I-ELM	0.1177	0.004	0.0102	12.8000	10.0974	12.8000
			AG-ELM	0.1126	0.0050	1.2367	2.4500	1.6376	13.3000
Boston Housing	0.20	200	I-ELM	0.1931	0.0142	0.0023	11.5500	11.8609	11.5500
			AG-ELM	0.1891	0.0121	0.3908	2.6000	2.5215	20.05
California Housing	0.25	200	I-ELM	0.2405	0.0121	0.0187	10.1500	12.8811	10.1500
			AG-ELM	0.2377	0.0088	0.2883	3.1000	3.1271	6.5500
Census (House8L)	0.11	200	I-ELM	0.1073	0.0057	0.0039	6.5400	8.6904	6.5400
			AG-ELM	0.1054	0.0043	2.7909	1.9000	1.4105	15.6500
Delta Ailerons	0.09	200	I-ELM	0.0835	0.0093	0.0109	29.8500	38.3519	29.8500
			AG-ELM	0.0723	0.0120	0.3674	5.1500	3.2489	12.4000
Delta Elevators	0.11	200	I-ELM	0.1021	0.0115	0.0164	24.9000	29.7426	24.9000
			AG-ELM	0.0957	0.0098	0.3750	3.7500	2.7314	10.3000
Machine CPU	0.10	200	I-ELM	0.0946	0.0215	0.0008	18.5500	9.4171	18.5500
			AG-ELM	<u>0.0945</u>	0.0210	3.8953	4.6500	4.4874	38.1500

5.4 Summary

This chapter has proposed the AG-ELM where the hidden node number is determined in an adapt way in the sense that the existing networks may be replaced by newly generated networks which have less hidden node number and better generalization performance rather than always keeping those existing ones in other constructive ELMs. The obtained AG-ELM provides tradeoff solutions if users wish to have more compact network

architecture in their ELM applications while keeping the better or comparable generalization performance. Then the universal approximation theory has been established in the sense that such AG-ELM can approximate any Lebesgue p -integrable target function $f(\mathbf{x})$ on a compact input set X if the hidden node activation function is Lebesgue p -integrable and $\text{span}\{G(\mathbf{a}, b, \mathbf{x}) : (\mathbf{a}, b) \in R^d \times R\}$ is dense in $L^p(X)$. The obtained result not only strengthens those existing results, but also provides a fundamental theory for studying the approximation capability of algorithms which use random mechanism for the parameter searching in neural networks.

Chapter 6

Dynamic Extreme Learning Machine (D-ELM)

I-ELMs can work as an universal approximator by adding the random hidden nodes one by one. Different from I-ELM, EM-ELM allows to add random hidden nodes one by one or group by group (with varying size). Moreover, EM-ELM incrementally updates the output weights during the network growth, which significantly reduces the computational complexity. However, in the implementation of all these existing constructive ELMs, the number of the hidden nodes always monotonically increases with the learning progress and the final number of the hidden nodes is equivalent to the learning steps. Then large number of hidden nodes will be obtained eventually if there needs many iterative steps, while some of the hidden nodes may play a very minor role in the network output. On the other hand, although AG-ELM provides a new approach to automatically determine the network architecture in an adaptive way, it is not so efficient as expected. Therefore, how to design an efficient approach which can choose the optimal network architecture automatically as well as possess the universal approximation capability is the main objective of this chapter.

The main ideas of this chapter can be found in “Dynamic extreme learning machine and its approximation capability,” submitted IEEE Transactions on System, Man and Cybernetics - Part B.

This chapter first proposes a dynamic extreme learning machine (D-ELM) performing dynamic growth of hidden nodes and incremental updating of output weights (some preliminary results on D-ELM have been published in [134]). Then the approximation capability of such D-ELM is analyzed on the basis of the theory obtained in [135]. It will be verified that the obtained D-ELM can approximate any Lebesgue p -integrable function as long as the hidden activation is Lebesgue p -integrable .

6.1 The Proposed D-ELM and Its Approximation Capability

In this section, the proposed dynamic extreme learning machine (D-ELM) will be first introduced. In the following, the GSLFN in the matrix form is expressed as

$$f_n(\mathbf{x}) = \sum_{i=1}^n \beta_i G(\mathbf{a}_i, b_i, \mathbf{x}) = \beta_n \cdot \mathbf{G}_n(\mathbf{x})$$

where $\beta_n = [\beta_1, \beta_2, \dots, \beta_n]^T$, $\mathbf{G}_n(\mathbf{x}) = [G(\mathbf{a}_1, b_1, \mathbf{x}), G(\mathbf{a}_2, b_2, \mathbf{x}), \dots, G(\mathbf{a}_n, b_n, \mathbf{x})]^T$ and $\beta_n \cdot \mathbf{G}_n(\mathbf{x})$ denotes the inner product of β_n and $\mathbf{G}_n(\mathbf{x})$.

Let $\Psi_n(\mathbf{x})$ be the network obtained by D-ELM at the n th step, E_n and L_n be the residual error and the hidden node number of Ψ_n respectively. The details of the proposed D-ELM are presented as follows.

D-ELM Algorithm: Given a set of training data $\{(\mathbf{x}_i, t_i)\}_{i=1}^N \subset R^d \times R$, the maximum hidden node number $L_{\max}$, and the expected learning accuracy ε .

Phase I — Initialization Phase:

Let $n = 1$.

1) Randomly generate the hidden node parameter $(\mathbf{a}_1, b_1) \in R^d \times R$ based on a pseudo-

uniform distribution;

2) Calculate the hidden-layer output $\mathbf{G}_1(\mathbf{x}) = [G(\mathbf{a}_1, b_1, \mathbf{x}_1), \dots, G(\mathbf{a}_1, b_1, \mathbf{x}_N)]^T$;

3) Calculate the optimal output weight $\hat{\beta}_1 = [\mathbf{G}_1(\mathbf{x})]^\dagger \mathbf{T}$.

Therefore, the approximated function Ψ_1 and its corresponding error E_1 are obtained:

$$\Psi_1(\mathbf{x}) = \hat{\beta}_1 \cdot \mathbf{G}_1(\mathbf{x}) \text{ and } E_1 = \|\Psi_1 - f\|.$$

Phase II — Recursively Growing Phase: while $L_n < L_{\max}$ and $E_n > \varepsilon$:

Let $n = 2$.

1) *Building a temporary network ψ_n :*

for $k = 1 : L_{n-1}$,

- (a) randomly generate $(\tilde{\mathbf{a}}_n^k, \tilde{b}_n^k) \in R^d \times R$ based on a pseudo-uniform distribution;
- (b) calculate $\tilde{\psi}_n^k(\mathbf{x}) = \sum_{i=1}^k \tilde{\beta}_n^i G(\tilde{\mathbf{a}}_n^i, \tilde{b}_n^i, \mathbf{x}) = \tilde{\beta}_k \cdot \tilde{\mathbf{G}}_k(\mathbf{x})$ with k hidden nodes and its corresponding error $\tilde{e}_n^k = \|\tilde{\psi}_n^k - f\|$ where $\tilde{\beta}_k$ is calculated by (3.8);
- (c) compare Ψ_{n-1} with $\tilde{\psi}_n^k(\mathbf{x})$: if $\tilde{e}_n^k \leq E_{n-1}$, then stop; otherwise, let $k = k + 1$ and go to step (a).

endfor

Denote by ψ_n the final obtained network in step 1) of *Phase II*:

$$\psi_n(\mathbf{x}) = \begin{cases} \tilde{\psi}_n^k(\mathbf{x}), & \text{if } k < L_{n-1}; \\ \tilde{\psi}_n^{L_{n-1}}(\mathbf{x}), & \text{if } k = L_{n-1} \text{ and } \tilde{e}_n^{L_{n-1}} < E_{n-1}; \\ \Psi_{n-1}(\mathbf{x}), & \text{if } k = L_{n-1} \text{ and } \tilde{e}_n^{L_{n-1}} \geq E_{n-1} \end{cases}$$

and by $\mathbf{G}_{\mathbf{nb}}(\mathbf{x})$ the hidden-layer output of $\psi_n(\mathbf{x})$.

2) *Incremental step:* randomly generate $(\mathbf{a}_n, b_n) \in R^d \times R$ based on a pseudo-uniform

distribution. Adding a hidden node on ψ_n in terms of the parameters $(\mathbf{a}_n, b_n)$ yields a new network with the hidden-layer output $\mathbf{G}_n(\mathbf{x}) = [\mathbf{G}_{n-1}(\mathbf{x})^T, G(\mathbf{a}_n, b_n, \mathbf{x})]^T$. Compute the output weight $\hat{\beta}_n$ according to (3.8).

Therefore, the approximated function Ψ_n and its corresponding error E_n are obtained:

$$\Psi_n(\mathbf{x}) = \hat{\beta}_n \cdot \mathbf{G}_n(\mathbf{x}) \quad \text{and} \quad E_n = \|\Psi_n - f\|.$$

3) *Recursive step:* Let $n = n + 1$ and go to step 1) of *phase II*.

endwhile

As stated above, D-ELM includes two phases: the initialization phase and the recursively growing phase.

The first phase is an initialization process. A network $\Psi_1(\mathbf{x}) = \hat{\beta}_1 \cdot \mathbf{G}_1(\mathbf{x})$ with one hidden node is initialized, where the hidden-layer output $\mathbf{G}_1(\mathbf{x})$ are randomly generated and the output weight is calculated by $\hat{\beta}_1 = [\mathbf{G}_1(\mathbf{x})]^\dagger \mathbf{T}$.

The recursively growing phase includes three steps. At the n th iteration, a temporary network ψ_n is first built up in the following way. We start from generating a network $\tilde{\psi}_n^1$ with one hidden node and compare it with the network Ψ_{n-1} according to the residual errors $\tilde{e}_n^1$ and E_{n-1} , and then retain the one whose error is less. If the retained network is $\tilde{\psi}_n^1$, then this network is just the temporary one we are looking for at this step. Otherwise, we keep generating a network $\tilde{\psi}_n^2$ with two hidden nodes and compare it with Ψ_{n-1} . This procedure continues until one of these networks performs better than Ψ_{n-1} , or all the networks $\tilde{\psi}_n^1, \dots, \tilde{\psi}_n^{L_{n-1}}$ have been compared with Ψ_{n-1} . Denote by ψ_n the network obtained at this step eventually. Obviously, ψ_n can be viewed as the “best” one since it has the most compact network architecture and the best generalization performance so far. We then add one more hidden node into ψ_n to be the network Ψ_n at the current iteration. And finally, we judge the stop criterion and go to the next iteration if needed. At each step of this phase, the output weights are updated incrementally according to

(3.8).

Remark 6.1 *Different from the constructive ELMs or pruning ELMs, in D-ELM, the hidden node number changes dynamically guided by the given task. At each iteration, among all the possible candidates $\{\tilde{\Psi}_n^1(\mathbf{x}), \tilde{\Psi}_n^2(\mathbf{x}), \dots, \tilde{\Psi}_n^{L_{n-1}}(\mathbf{x}), \Psi_{n-1}\}$, the generalization performance of them will be compared, and the one whose number of hidden nodes is the least among those having the best generalization, will be selected as the temporary network Ψ_n . Note that the hidden node number of Ψ_n may be any number of $\{1, 2, \dots, L_{n-1}\}$ where L_{n-1} satisfies $2 \leq L_{n-1} \leq n-1$. Thus for the network Ψ_n obtained at the current iteration, its hidden node number L_n must be any one of $\{2, \dots, L_{n-1} + 1\}$. Therefore, there may hold any one of the following cases: $L_n < L_{n-1}$, $L_n = L_{n-1}$ and $L_n > L_{n-1}$. This fact reveals that the hidden nodes can be recruited or deleted dynamically according to their significance to network performance. The dynamics of such architecture design allows the proposed D-ELM to determine an appropriate network architecture eventually.*

Remark 6.2 *In D-ELM, the approach to building the temporary network Ψ_n is originated from AG-ELM. From the standpoint of network architecture design, D-ELM can be viewed as an improvement of AG-ELM in the following sense: (i) the output weights in D-ELM are updated incrementally by the error-minimization-based method rather than randomly generated in AG-ELM, which obviously leads to better performance for the network output; (ii) in D-ELM, after the temporary network Ψ_n , which can be viewed as the “best” one so far, is selected, one more random hidden node is further added. This guarantees that the error may reduce further; (iii) A key common point of AG-ELM and D-ELM is that a group of networks are generated at each step. However, it is different for them in the aspect that how many networks are generated in group at each step. In AG-ELM, the same number of networks as the learning step are generated. That is to say, at the n th learning step, n networks are generated in group. Differently, in D-ELM, only L_{n-1} networks are generated in group where $1 \leq L_{n-1} \leq n-1$. This fact also supports why D-ELM can achieve a more compact network architecture than AG-ELM.*

The approximation capability of D-ELM is clarified in the following. Two lemmas need to be given in order for the proof of the main universal approximation theorem.

As mentioned in the proceeding chapters, the GSLFN $f_n(\mathbf{x}) = \beta_n \cdot \mathbf{G}_n(\mathbf{x}) = \sum_{i=1}^n \beta_i G(\mathbf{a}_i, b_i, \mathbf{x})$ is a linear system in the case where the hidden-layer output $\mathbf{G}_n(\mathbf{x})$ is randomly generated. The notation $e_n = \|f_n - f\|$ is used to denote the error between f_n and the target function f . The following lemma reveals that adding more hidden nodes into an existing GSLFN will lead to less residual error.

Lemma 6.1 *Given a GSLFN $f_{n_0}(\mathbf{x}) = \beta_{n_0} \cdot \mathbf{G}_{n_0}(\mathbf{x})$ with n_0 hidden nodes. If a new network $f_{n_1}(\mathbf{x})$ is generated by adding $(n_1 - n_0)$ new hidden nodes into $f_{n_0}(\mathbf{x})$, then the errors corresponding to $f_{n_0}(\mathbf{x})$ and $f_{n_1}(\mathbf{x})$ satisfy*

$$e_{n_1} \leq e_{n_0}$$

in the case where the output weight vector β_n is a minimum norm least-squares solution of the linear system $f_n(\mathbf{x}) = \beta_n \cdot \mathbf{G}_n(\mathbf{x})$.

Proof: Denote the new network by $f_{n_1}(\mathbf{x}) = \beta_{n_1} \cdot \mathbf{G}_{n_1}(\mathbf{x})$. Since f_{n_1} is obtained by adding extra $(n_1 - n_0)$ hidden nodes into f_{n_0} , we have $\mathbf{G}_{n_1}(\mathbf{x}) = [\mathbf{G}_{n_0}(\mathbf{x})^T, \delta \mathbf{G}_{n_0}^T]^T$ where $\delta \mathbf{G}_{n_0} = [G(\mathbf{a}_{n_0+1}, b_{n_0+1}, \mathbf{x}), \dots, G(\mathbf{a}_{n_1}, b_{n_1}, \mathbf{x})]^T$. Under the assumption that β_n is a minimum norm least-squares solution of the linear system $f_n(\mathbf{x}) = \beta_n \cdot \mathbf{G}_n(\mathbf{x})$, it can be then obtained that

$$\begin{aligned} e_{n_1} &= \|f_{n_1} - f\| \\ &= \|\beta_{n_1} \cdot \mathbf{G}_{n_1}(\mathbf{x}) - f\| \\ &= \|\beta_{n_1} \cdot [\mathbf{G}_{n_0}(\mathbf{x})^T, \delta \mathbf{G}_{n_0}^T]^T - f\| \\ &\leq \|[\beta_{n_0}, 0][\mathbf{G}_{n_0}(\mathbf{x})^T, \delta \mathbf{G}_{n_0}^T]^T - f\| \\ &= \|\beta_{n_0} \cdot \mathbf{G}_{n_0}(\mathbf{x}) - f\| = e_{n_0}. \end{aligned}$$

Lemma 6.1 is thus proved.

In view of Theorem 5.3, Theorem 2.2 and Lemma 6.1, the following universal approximation result for D-ELM can be proved.

Theorem 6.1 *Let $g : R \rightarrow R$ be a Lebesgue p -integrable hidden activation function defined on the compact set $X \subset R^d$. If $\text{span}\{G(\mathbf{a}, b, \mathbf{x}) : (\mathbf{a}, b) \in R^d \times R\}$ is dense in $L^p(X)$, then for any Lebesgue p -integrable function $f(\mathbf{x})$ and any D-ELM $\{f_n(\mathbf{x})\}$, $\lim_{n \rightarrow \infty} \|f_n - f\| = 0$ holds with probability one.*

Proof: In order to be more readable, the symbols and meanings of different network sequences that will be used in the following proof need to be presented systematically here. Specifically, $\{\Phi_n(\mathbf{x})\}$ is the network sequence generated by AG-ELM, $\{\psi_n(\mathbf{x})\}$ is the temporary network sequence generated in D-ELM and $\{\Psi_n(\mathbf{x})\}$ is the final network sequence generated by D-ELM.

The generalization performances between $\{\Phi_n(\mathbf{x})\}$ and $\{\psi_n(\mathbf{x})\}$ is first compared. As stated in Remark 6.2, $\{\Phi_n(\mathbf{x})\}$ in AG-ELM and $\{\psi_n(\mathbf{x})\}$ in D-ELM are generated in the similar manner. However, they are completely different in obtaining the output weights. The output weights of $\Phi_n(\mathbf{x})$ are randomly generated based on a pseudo-uniform distribution, while the output weights of $\{\psi_n(\mathbf{x})\}$ are calculated by (3.8). By Theorem 2.2, it is known that the output weights calculated by (3.8) must be minimum norm least-squares solutions. Therefore, we obviously have

$$\|\psi_n - f\| \leq \|\Phi_n - f\|. \quad (6.1)$$

Next the generalization performances between $\{\psi_n(\mathbf{x})\}$ and $\{\Psi_n(\mathbf{x})\}$ is analyzed, which are both generated in D-ELM. According to the algorithm D-ELM, $\Psi_n(\mathbf{x})$ is obtained

by adding one more random hidden nodes in $\psi_n(\mathbf{x})$. Then by Lemma 6.1, there is

$$\|\Psi_n - f\| \leq \|\psi_n - f\|. \quad (6.2)$$

Combining (6.1) with (6.2), it can be then obtained that

$$\|\Psi_n - f\| \leq \|\Phi_n - f\|. \quad (6.3)$$

In the light of Theorem 5.3, for any Lebesgue p -integrable function $f(\mathbf{x})$ and any AG-ELM $\{\Phi_n(\mathbf{x})\}$, $\lim_{n \rightarrow \infty} \|\Phi_n - f\| = 0$ holds with probability one, provided that $g : R \rightarrow R$ is a Lebesgue p -integrable on the compact set $X \subset R^d$ and $\text{span}\{G(\mathbf{a}, b, \mathbf{x}) : (\mathbf{a}, b) \in R^d \times R\}$ is dense in $L^p(X)$. Therefore, according to (6.3), it is immediately proved that under the same assumptions as those in Theorem 5.3,

$$\lim_{n \rightarrow \infty} \|\Psi_n - f\| = 0$$

holds with probability one. The proof of Theorem 6.1 is thus completed.

6.2 Experimental Verification

In this section, the proposed algorithm D-ELM is compared with other constructive ELMs and support vector machine (SVM) on ten real-world function approximation problems from UCI database [8]. The constructive ELMs to be compared include I-ELM, EI-ELM and EM-ELM. The SVM is performed using the SVM toolbox with the default settings [13]. The epsilon-SVR has been used for regression, with radial basis function (RBF) kernel. The specification of these benchmark problems are shown in

Table 6.1: Specification of Benchmark Datasets

Problems	Attributes	Cases	Training Data	Testing Data
Abalone	8	4177	2000	2177
Boston Housing	13	506	250	256
California Housing	8	20640	8000	12640
Census(House8L)	8	22784	10000	12784
delta ailerons	5	7129	3000	4129
delta elevators	6	9517	4000	5517
Japanese Vowels	12	9961	4275	5686
Parkinsons Telemonitoring	22	5875	2500	3375
Synthetic Control Chart Time Series	60	600	300	300
Connectionist Bench	14	990	450	540

Table 6.1.

In the experiments, all the inputs (attributes) have been normalized into the range $[-1, 1]$ while the outputs (targets) have been normalized into $[0, 1]$. Neural networks with sigmoid type of additive hidden nodes $G(\mathbf{a}, b, \mathbf{x}) = 1/(1 + \exp(-(\mathbf{a} \cdot \mathbf{x} + b)))$ and RBF nodes $G(\mathbf{a}_i, b_i, \mathbf{x}) = \exp(-b_i \|\mathbf{x} - \mathbf{a}_i\|^2)$ are tested in all the algorithms respectively. The hidden node parameters $(\mathbf{a}, b)$ are randomly chosen from the range $[-1, 1]^d \times [-1, 1]$ based on a uniform sampling distribution probability. For each trial of simulations, the data set of the application was divided into training and testing data sets with the number of samples indicated in Table 6.1. All the simulations for I-ELM, EI-ELM, EM-ELM, SVM and the new algorithm D-ELM are carried out in MATLAB 7.4.0 environment and the same PC with Intel Core i3 2.27GHZ CPU and 2GB RAM. One hundred trials have been conducted for each problem, with training and testing data sets randomly generated for each trial. Simulation results including the average testing root mean square error (RMSE) and the corresponding standard deviation (Dev), the training time, the number of hidden nodes (# Nodes) and the corresponding node deviation (NodeDev), the number of support vectors (# SVs) and the corresponding deviation (SVDev) are provided in this section. The apparent better results are shown in boldface and the close results are underlined.

6.2.1 Comparison between D-ELM and other constructive ELMs with the same expected accuracy

In this subsection, given the expected accuracy (or the same maximum step), the network complexity and the generalization performance of the proposed D-ELM are compared with other constructive extreme learning machines such as I-ELM, EI-ELM and EM-ELM, as well as SVM. The stopping RMSE are presented in the tables and the maximum step is set to 200 for all the ten regression data sets.

Table 6.2 shows the average performance (testing RMSE, the corresponding Dev, and the training time) and complexity (# SVs and the corresponding SVDev) of SVM. Table 6.3 and Table 6.4 show the network complexity (# Nodes and the corresponding NodeDev) of four ELM-algorithms with sigmoid nodes and RBF nodes respectively, Table 6.5 and Table 6.6 show the average performance (testing RMSE, the corresponding Dev, and the training time) of them. It can be seen that the proposed D-ELM with different activation functions always obtains much lower network complexity as well as better generalization performance than I-ELM and EI-ELM for all ten data sets. Compared to the results of EM-ELM, the number of hidden nodes obtained by D-ELM are always less than those of EM-ELM. Meanwhile, the generalization performance of D-ELM is better than or comparable to that of EM-ELM except for the Census(House8L) case (with both activation functions) and the Delta Elevators case (only with sigmoid nodes).

Compared to the results of SVM, D-ELM can always obtain better generalization performance than SVM. Particularly, the much less deviations of D-ELM also show that the performance of D-ELM is much stabler than SVM. Moreover, the number of support vectors used in SVM is generally several tens times of the number of hidden nodes used in D-ELM, especially, it is more than 200 times for the California Housing case. D-ELM spends less training time than SVM in six data sets.

Therefore, through this simulation it is found that to achieve the same residual error, D-ELM obtains the most parsimonious network architectures among all algorithms as

Table 6.2: Generalization Performance and Complexity of SVM

Problems	Testing RMSE		Support Vectors		Training Times(s)
	Mean	Dev	# SVs	SVDev	
Abalone	0.0902	0.0101	349.2	29.8842	0.1438
Boston Housing	0.1352	0.0085	72	4.9889	0.0282
California Housing	0.1731	0.0012	3770.2	36.9859	5.5719
Census(House8L)	0.1072	0.0033	1361.7	37.9182	6.3125
Delta Ailerons	0.0768	0.0077	291.7	124.2346	0.3481
Delta Elevators	0.0726	0.009	470.2	113.1958	0.3563
Japanese Vowels	0.1162	0.0023	495.1	59.439	0.3969
Parkinsons Telemonitoring	0.0897	0.0018	495	26.0811	0.3234
Synthetic Control Chart Time Series	0.2338	0.0068	189.9	2.2336	0.0469
Connectionist Bench	0.2464	0.0007	257.7	10.7502	0.0625

well as preserving excellent generalization performance.

6.2.2 Comparison between D-ELM and other constructive ELMs with the same step

The proposed algorithm D-ELM is also compared with other constructive ELMs given the expected step. In I-ELM, EI-ELM and EM-ELM, the hidden nodes are added one by one. Table 6.7 and Table 6.8 show the network complexity and generalization performance of different algorithms with sigmoid nodes and RBF nodes under the case where the learning steps are fixed to 100 respectively. Seen from Table 6.7 and Table 6.8, the proposed D-ELM always has lower network complexity than three other constructive ELMs for all ten problems. Since the hidden nodes are added one by one in I-ELM, EI-ELM and EM-ELM, 100 hidden nodes are attained eventually after 100 learning steps (only 100 hidden nodes obtained by EM-ELM are shown in Table 6.7 and Table 6.8 since they are same for I-ELM and EI-ELM). Differently, the network architectures of D-ELM are automatically determined through their own internal growing and pruning mechanism, thus more compact network architectures are obtained. Meanwhile, in each case, the generalization performance of D-ELM is better than I-ELM and EI-ELM, and is generally better or comparable to EM-ELM. These facts can also be verified from

Fig. 6.1 and Fig. 6.2 which show the average testing RMSE decreasing trends and the updating progress of hidden-node number of D-ELM and EM-ELM for Abalone case respectively. The similar learning curves could also be found for the rest cases as well.

It should be noted that the proposed D-ELM spends more training time than other constructive ELMs. Actually, this fact is reasonable because D-ELM needs more time to seek for the most appropriate network architecture from all the possible candidates at each step. Another fact is that D-ELM with sigmoid nodes performs better than that with RBF nodes in most cases.

Figure 6.1: The testing RMSE updating curves of D-ELM and EM-ELM for Abalone case

6.3 Summary

This chapter first proposes the D-ELM where the hidden nodes can be recruited or deleted dynamically according to their significance to network performance, so that not

Figure 6.2: The hidden-node number updating progress in 100 steps for Abalone case

only can the parameters be adjusted but also the architecture can be self-adapted simultaneously. The key idea of D-ELM is the following. The network starts with one random hidden node. Then at each subsequent step, several temporary networks with different numbers of hidden nodes are first generated. Among all these temporary networks and the network obtained at the previous step, the “best” one, which has the least hidden node number as well as the best generalization, will be selected. Adding one more random hidden node to such selected network then results in the network at the current step. In D-ELM, the hidden nodes are randomly generated and the output weights are updated incrementally by using the error-minimization-based method proposed in [23]. Next this chapter proves in theory that such D-ELM are actually the universal approximator. It has been proved that such D-ELM can approximate any Lebesgue p -integrable target function $f(\mathbf{x})$ on a compact input set X if the hidden node activation function is Lebesgue p -integrable and $\text{span}\{G(\mathbf{a}, b, \mathbf{x}) : (\mathbf{a}, b) \in R^d \times R\}$ is dense in $L^p(X)$. The obtained result generalizes those existing function approximation results using ELMs.

It is valid not only for much general type of ELMs (say, the hidden activation functions can be selected from broader type of functions) but also for a very general family of target functions (say, the target functions can be any Lebesgue p -integrable function rather than continuous functions), which then underlines successful applications of function approximation using ELMs. The network complexity and the generalization performance of D-ELM in solving ten real-world function approximation problems are compared with I-ELM, EI-ELM, EM-ELM and SVM. The results of the experimental comparison show that the proposed D-ELM does a good job reducing the network size efficiently while preserving excellent generalization performance.

Table 6.3: Network Complexity Comparison of D-ELM, EM-ELM, EI-ELM and I-ELM (with sigmoid nodes) given the same expected accuracy

Problems	Stop RMSE	I-ELM (Sigmoid)		EI-ELM (Sigmoid)		EM-ELM (Sigmoid)		D-ELM (Sigmoid)	
		# Nodes	NodeDev	# Nodes	NodeDev	# Nodes	NodeDev	# Nodes	NodeDev
Abalone	0.09	174.75	59.7635	61.1	44.1801	5.7	1.2607	4.65	0.8751
Boston Housing	0.12	151.28	46.1241	30.71	13.9054	12.09	3.098	8.84	1.9494
California Housing	0.14	200	0	200	0	20.26	4.0167	16.08	2.7768
Census(House8L)	0.07	200	0	200	0	47	8.4829	38.56	6.9141
Delta Ailerons	0.05	171.18	40.1314	21.06	15.5081	4.58	1.2648	3.71	0.9134
Delta Elevators	0.06	191.6	21.0536	33.81	13.4079	6.16	1.48	4.92	1.2034
Japanese Vowels	0.12	186	22.7758	28.95	7.99	9.35	2.4339	7.15	1.5985
Parkinsons Telemonitoring	0.09	115.04	50.9075	15.7	5.6165	10.07	2.5515	7.24	1.6704
Synthetic Control Chart Time Series	0.1	198.3	10.131	75.26	31.2732	12.46	4.1571	7.94	1.7628
Connectionist Bench	0.2	200	0	200	0	31.53	5.7357	24.16	3.5809

Table 6.4: Network Complexity Comparison of D-ELM, EM-ELM, EI-ELM and I-ELM (with RBF nodes) given the same expected accuracy

Problems	Stop RMSE	I-ELM (RBF)		EI-ELM (RBF)		EM-ELM (RBF)		D-ELM (RBF)	
		# Nodes	NodeDev	# Nodes	NodeDev	# Nodes	NodeDev	# Nodes	NodeDev
Abalone	0.09	147.25	63.3178	17.45	5.145	6.05	1.6694	5.25	1.0195
Boston Housing	0.12	200	0	98.82	24.625	14.76	3.2491	11	2.1973
California Housing	0.14	200	0	200	0	23.12	4.6434	17.36	3.569
Census(House8L)	0.07	200	0	199.97	0.3	44.83	5.9967	37.89	5.193
Delta Ailerons	0.05	188.21	43.8423	65.65	76.7663	4.85	1.5464	3.91	0.9857
Delta Elevators	0.06	189.81	35.7182	126.44	90.1825	7.62	1.6623	5.7	1.1326
Japanese Vowels	0.12	136.9	58.655	15.2	5.1667	9.45	2.8186	7.25	1.2927
Parkinsons Telemonitoring	0.09	126.56	67.5633	8.03	5.2444	6.59	2.6897	4.17	1.2796
Synthetic Control Chart Time Series	0.1	200	0	169.45	33.7592	41.89	13.9493	19.7	5.5021
Connectionist Bench	0.2	197.68	11.0104	60.02	15.6592	25.18	4.7998	19.29	3.1491

Table 6.5: Generalization Performance Comparison of D-ELM, EM-ELM, EI-ELM and I-ELM (with sigmoid nodes) given the same expected accuracy. The close generalization performance (testing RMSE) obtained by different algorithms is underlined and the best results among four algorithms are shown in boldface

Problems	I-ELM (Sigmoid)			EI-ELM (Sigmoid)			EM-ELM (Sigmoid)			D-ELM (Sigmoid)		
	Testing RMSE		Training Times(s)	Testing RMSE		Training Times(s)	Testing RMSE		Training Times(s)	Testing RMSE		Training Times(s)
	Mean	Dev		Mean	Dev		Mean	Dev		Mean	Dev	
Abalone	0.0963	0.005	0.0928	0.0904	0.0026	0.3744	0.0874	0.0033	0.0172	0.0868	0.0041	0.0554
Boston Housing	0.1312	0.0137	0.0184	0.1287	0.0135	0.0838	0.1182	0.0031	0.0163	0.1179	0.0048	0.073
California Housing	0.1682	0.0064	0.3084	0.1505	0.0016	3.4774	<u>0.1404</u>	0.0022	0.3538	<u>0.1404</u>	0.0023	2.5575
Census (House8L)	0.0924	0.0022	0.3814	0.0833	0.0015	4.1222	0.0697	0.0003	1.7622	0.0711	0.0022	32.0178
Delta Ailerons	0.0526	0.0044	0.1223	0.0497	0.0018	0.1601	<u>0.0456</u>	0.0033	0.022	<u>0.0455</u>	0.0032	0.0619
Delta Elevators	0.0668	0.0073	0.1535	0.0596	0.0012	0.3104	0.0566	0.002	0.0287	0.0574	0.0022	0.0953
Japanese Vowels	0.1257	0.0066	0.2039	0.1195	0.0017	0.2859	0.1158	0.0038	0.05	0.1142	0.0044	0.1773
Parkinsons Telemonitoring	0.092	0.0055	0.0841	0.0899	0.0025	0.1211	0.0875	0.0037	0.048	0.0871	0.0037	0.1849
Synthetic Control Chart Time Series	0.1335	0.0212	0.0615	0.111	0.0075	0.205	0.1018	0.0073	0.0256	0.1014	0.0062	0.1014
Connectionist Bench	0.2418	0.077	0.0578	0.2263	0.0065	0.3194	0.2121	0.0093	0.0616	0.2104	0.0095	0.6742

Table 6.6: Generalization Performance Comparison of D-ELM, EM-ELM, EI-ELM and I-ELM (with RBF nodes) given the same expected accuracy. The close generalization performance (testing RMSE) obtained by different algorithms is underlined and the best results among four algorithms are shown in boldface

Problems	I-ELM (RBF)			EI-ELM (RBF)			EM-ELM (RBF)			D-ELM (RBF)		
	Testing RMSE		Training Times(s)	Testing RMSE		Training Times(s)	Testing RMSE		Training Times(s)	Testing RMSE		Training Times(s)
	Mean	Dev		Mean	Dev		Mean	Dev		Mean	Dev	
Abalone	0.099	0.0104	0.1544	0.091	0.031	0.1849	0.089	0.0041	0.0328	0.0879	0.0027	0.1264
Boston Housing	0.205	0.0856	0.038	0.155	0.021	0.3334	0.1175	0.0034	0.0166	0.1171	0.0074	0.1103
California Housing	0.222	0.0992	0.6611	0.1541	0.0044	0.9801	0.1406	0.0026	0.5107	0.1402	0.0024	3.9545
Census (House8L)	0.0904	0.0064	0.9856	0.0817	0.0111	10.3789	0.0696	0.0005	1.9255	0.0712	0.0024	34.0106
Delta Ailerons	0.1547	0.1458	0.2293	0.1386	0.3673	0.8582	<u>0.0466</u>	0.0028	0.0197	<u>0.0465</u>	0.0025	0.0534
Delta Elevators	0.2119	0.2255	0.3246	0.1139	0.0911	2.2355	<u>0.0573</u>	0.0019	0.0454	<u>0.0575</u>	0.0018	0.1484
Japanese Vowels	0.1241	0.0119	0.3555	0.1199	0.0019	0.3703	0.1154	0.0054	0.0695	0.1148	0.0052	0.2797
Parkinsons Telemonitoring	0.095	0.0073	0.2065	0.0893	0.0022	0.1365	0.0888	0.0104	0.0441	0.088	0.0028	0.1052
Synthetic Control Chart Time Series	0.1903	0.0358	0.1084	0.117	0.013	0.9326	0.1162	0.0122	0.1452	0.1043	0.0081	1.0694
Connectionist Bench	0.2237	0.0127	0.088	<u>0.2083</u>	0.0102	0.1616	<u>0.2084</u>	0.0104	0.0419	<u>0.2082</u>	0.0101	0.4396

Table 6.7: Generalization Performance and Network Complexity Comparison of D-ELM, EM-ELM, EI-ELM and I-ELM (with sigmoid nodes) given the same expected step. The close results obtained by different algorithms is underlined and the best results among four algorithms are shown in boldface

Problems	I-ELM (Sigmoid)				EI-ELM (Sigmoid)				EM-ELM (Sigmoid)				D-ELM (Sigmoid)			
	Testing RMSE		Training Times(s)		Testing RMSE		Training Times(s)		Testing RMSE		Training Times(s)		Testing RMSE		Training Times(s)	
	Mean	Dev	Mean	Dev	Mean	Dev	Mean	Dev	Mean	Dev	Mean	Dev	Mean	Dev	Nodes	NodeDev
Abalone	0.0967	0.0055	0.0579	0.0034	0.088	0.0034	0.5746	0.0213	0.0918	0.0213	1.5088	0.0131	0.0851	0.0131	80.24	6.1973
Boston Housing	0.142	0.0163	0.0217	0.0068	0.1126	0.0068	0.3065	0.0253	0.1212	0.0253	0.2669	0.0151	0.106	0.0151	79.79	7.4972
California Housing	0.1886	0.0105	0.1557	0.0026	0.1539	0.0026	1.7059	0.0073	0.1324	0.0073	5.6405	0.007	0.1316	0.007	79.67	6.0904
Census (House8L)	0.0968	0.0025	0.2193	0.0016	0.0862	0.0016	2.2204	0.0012	0.0646	0.0012	7.2356	0.0016	0.0677	0.0016	80.8	7.4698
Delta Ailerons	0.062	0.0108	0.0474	0.0019	0.0421	0.0019	0.3902	0.0006	0.0364	0.0006	1.824	0.0008	0.0395	0.0008	79.17	7.074
Delta Elevators	0.0792	0.0111	0.0796	0.0009	0.0544	0.0009	0.8717	0.0005	<u>0.0535</u>	0.0005	2.8305	0.0005	<u>0.0534</u>	0.0005	80.75	6.4876
Japanese Vowels	0.1457	0.0147	0.0944	0.0024	0.1082	0.0024	1.0358	0.0029	0.0661	0.0029	3.2222	0.0017	0.0653	0.0017	78.7	8.115
Parkinsons Telemonitoring	0.0938	0.0092	0.071	0.0025	0.073	0.0025	0.7535	0.0022	0.0569	0.0022	1.9436	0.0017	0.0566	0.0017	80.72	7.2098
Synthetic Control Chart Time Series	0.1532	0.0226	0.0253	0.0089	0.1065	0.0089	0.27	0.0034	0.0747	0.0034	0.2952	0.0029	0.073	0.0029	81.76	7.1281
Connectionist Bench	0.2554	0.0103	0.0302	0.0067	0.2318	0.0067	0.158	0.0098	0.1717	0.0098	0.3191	0.0087	0.1701	0.0087	79.88	8.0205

Table 6.8: Generalization Performance and Network Complexity Comparison of D-ELM, EM-ELM, EI-ELM and I-ELM (with RBF nodes) given the same expected step. The close results obtained by different algorithms is underlined and the best results among four algorithms are shown in boldface

Problems	I-ELM (RBF)				EI-ELM (RBF)				EM-ELM (RBF)				D-ELM (RBF)			
	Testing RMSE		Training Times(s)	# Nodes	Testing RMSE		Training Times(s)	# Nodes	Testing RMSE		Training Times(s)	# Nodes	Testing RMSE		Training Times(s)	# Nodes
	Mean	Dev			Mean	Dev			Mean	Dev			Mean	Dev		
Abalone	0.1091	0.0208	0.0964		0.0925	0.0537	1.026	100	0.0958	0.032	1.5703	100	0.0873	0.022	43.5627	78.75
Boston Housing	0.1991	0.0641	0.0329		0.1583	0.0302	0.422	100	0.1293	0.0302	0.2825	100	0.1121	0.0212	10.982	79.69
California Housing	0.2106	0.0321	0.2883		0.162	0.0066	2.9472	100	0.142	0.0203	5.4252	100	0.135	0.0083	171.428	79.65
Census (House8L)	0.0955	0.0025	0.5343		0.0847	0.0017	5.6894	100	0.0706	0.0059	8.8133	100	0.0694	0.0032	195.955	79.69
Delta Ailerons	0.2567	0.0389	0.0925		0.1141	0.2216	0.883	100	0.0439	0.0122	2.2748	100	0.0396	0.0009	54.2347	78.97
Delta Elevators	0.2165	0.1447	0.161		0.1207	0.1616	1.7293	100	<u>0.0535</u>	0.0005	2.932	100	<u>0.0534</u>	0.0005	89.4325	80.08
Japanese Vowels	0.1396	0.027	0.1864		0.095	0.002	1.8907	100	0.0612	0.0009	3.2745	100	0.0619	0.0014	103.854	81.65
Parkinsons Telemonitoring	0.0973	0.0145	0.1702		0.0737	0.0022	1.7107	100	0.0675	0.0039	2.2243	100	0.0639	0.0071	66.7931	79.24
Synthetic Control Chart Time Series	0.2112	0.0305	0.0582		0.1303	0.0154	0.5831	100	0.106	0.0141	0.3664	100	0.0877	0.0075	12.9442	74.83
Connectionist Bench	0.2475	0.018	0.043		0.1958	0.007	0.2322	100	<u>0.1611</u>	0.0086	0.2998	100	<u>0.1613</u>	0.0087	8.9636	80.8

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis further studied the dynamic configuration and approximation capabilities of ELMs. The major research results covered in this thesis can be summarized as follows.

- **OCI-ELM and its universal approximation capability**

An incremental extreme learning machine with optimal incremental coefficients (OCI-ELM) has been proposed, where an optimal combination of existing hidden nodes and the newly added hidden nodes can be obtained by introducing a new incremental method. It has also been proved that such OCI-ELM can approximate any continuous target function in the case where the hidden node activation function is bounded nonconstant piecewise continuous and $\text{span}\{G(\mathbf{a}, b, \mathbf{x})\}$ is dense in L^2 . Simulation results over various regression problems verify that the proposed OCI-ELM can obtain a faster convergence rate and more compact network architecture than I-ELM, EI-ELM and CI-ELM.

- **AG-ELM and its universal approximation capability**

An extreme learning machine with adaptive growth of hidden nodes (AG-ELM)

where a novel approach to determining the network architecture in an adaptive way has been introduced. Then the universal approximation capability of such AG-ELM has been proved in the following sense: (1) the obtained AG-ELM can approximate any Lebesgue p -integrable function, but not merely restricted to continuous functions in most literatures; (2) the hidden activation functions used in AG-ELM can be Lebesgue p -integrable, which is a much weaker condition than those assumed before. The obtained result not only strengthens those existing results, but also provides a fundamental theory for studying the approximation capability of algorithms which use random mechanism for the parameter searching in neural networks. Simulation results demonstrate and verify that our new approach can achieve a more compact network architecture than I-ELM.

- **D-ELM and its universal approximation capability**

A dynamic extreme learning machine (D-ELM) has first been proposed, where the hidden nodes can be recruited or deleted dynamically according to their significance to network performance, so that not only the parameters can be adjusted but also the architecture can be self-adapted simultaneously. Then it has been proved that such D-ELM using Lebesgue p -integrable hidden activation functions can approximate any Lebesgue p -integrable function on a compact input set. Simulation results obtained over various test problems demonstrate and verify that the proposed D-ELM does a good job reducing the network size efficiently while preserving excellent generalization performance.

7.2 Future work

This thesis studies the dynamic configuration and approximation capabilities of extreme learning machines. As a result, three new algorithms have been proposed, which are engaged to find the optimal network architecture from different aspects of cognition. The obtained parsimonious network can obviously reduce the computational consump-

tion, and hence to improve the applicability of ELMs in real-world problems. Moreover, the approximation capabilities of three new algorithms have been explored in the thesis, which guarantee in theory the effectiveness of ELMs in real applications.

Based on the research work conducted in this thesis, the future study will focus on proposing some more efficient learning methods based on ELMs and further study ELMs from the theoretical point of view. Several promising directions which are to be investigated in my future research study are listed as follows:

- With regard to the study of ELMs in large applications, a new approach has been proposed, which can automatically determine the optimal network architecture as well as the weights connecting the hidden nodes and the output nodes simultaneously most recently. Some preliminary theoretical results to support the new approach has been obtained so far. However, in order to further verify the performance of the proposed algorithm, there still need a large amount of simulations with huge data sets, which is my ongoing work.
- In the recent development of ELMs, it has been revealed that the kernel based ELM provides a unified learning platform with widespread type of feature mappings and can be applied in regression and multi-class classification applications directly. In addition, simulation results showed that the kernel based ELM is more stable than other computational intelligence techniques, such as BP algorithm, support vector machine (SVM), and least square support vector machine (LS-SVM) etc. Therefore, how to prove it in theory and give a quantitative analysis of the stability of the kernel based ELM, and uncovering the necessary and sufficient conditions on the kernel based ELM so as to guarantee its universal approximation capability is the other problem.
- ELMs were originally developed for training the SLFNs and then extended to the GSLFNs, which include but are not limited to SVM, polynomial network and the conventional FNNs. It has been shown that the traditional neural networks with minimum training error and minimum norms of weights tend to have better generalization performance [2]. Therefore, the third problem is to verify that the above mentioned conclusion

is also true for the GSLFNs.

Publication list

Journal Papers:

1. **Rui Zhang**, Yuan Lan, Guang-Bin Huang, Zong-Ben Xu, “Universal approximation of extreme learning machine with adaptive growth of hidden nodes,” *IEEE Transactions on Neural Networks and Learning Systems*, vol.23, no.2, pp.365-371, February 2012.
2. **Rui Zhang**, Zong-Ben Xu, Guang-Bin Huang, Dianhui Wang “Global convergence of online BP training procedure with dynamic learning rate,” *IEEE Transactions on Neural Networks and Learning Systems*, vol.23, no.2, pp.330-341, February 2012.
3. Guang-Bin Huang, Hongming Zhou, Xiaojian Ding, **Rui Zhang**, “Extreme learning machine for regression and multi-class classification”, *IEEE Transactions on System, Man and Cybernetics-Part B*, vol.42, no.2, pp.513-529, April 2012.
4. **Rui Zhang**, Yuan Lan, Guang-Bin Huang, Zong-Ben Xu, “Dynamic extreme learning machine and its approximation capability,” *submitted to IEEE Transactions on System, Man and Cybernetics-Part B*, 2011.
5. **Rui Zhang**, Wei Wang, Yuan Lan, Guang-Bin Huang, Zong-Ben Xu, “Incremental extreme learning machine with optimal incremental coefficients,” *submitted to IEEE Transactions on Neural Networks and Learning Systems*, 2011.

Conference Papers:

1. **Rui Zhang**, Yuan Lan, Guang-Bin Huang, Yeng Chai Soh, “ Extreme learning machine with adaptive growth of hidden nodes and incremental updating of output weights,” *International Conference on Autonomous and Intelligence Systems (AIS 2011)*, Burnaby, Canada, June 22-24, 2011.
2. **Rui Zhang**, Le Yang, Wei Wang, “A new convergence property of online BP learning,” *International Conference on Information Science and Technology (ICIST 2011)*, Nanjing, China, March 26-28, 2011.
3. **Rui Zhang**, ZuoZhi Liu, “A general result on the convergence of online BP learning,” *2011 International Workshop on Future Communication and Networking (IWFCN 2011)*, Hong Kong, December 12-13, 2011.

Bibliography

- [1] P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *IEEE Transactions on Neural Networks*, 5(1):54–65, 1994.
- [2] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal functions. *IEEE Transactions on Information Theory*, 39(3):930–945, 1993.
- [3] P. L. Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory*, 44(2):525–536, 1998.
- [4] E. B. Baum. On the capabilities of multilayer perceptrons. *Journal of Complexity*, 4(3):193–215, 1988.
- [5] E. B. Baum. A proposal for more powerful learning algorithms. *Neural Computation*, 1(2):201–207, 1989.
- [6] A. U. Bhat, S. S. Merchant, and S. S. Bhagwat. Prediction of melting points of organic compounds using extreme learning machines. *Industrial and Engineering Chemistry Research*, 47(3):920–925, 2008.
- [7] C. M. Bishop. Neural networks for pattern recognition. NY: Oxford University Press, 1995.
- [8] C. L. Black and C. J. Merz. UCI repository of machine learning databases. in <http://www.ics.uci.edu/mllearn/MLRepository.html>, Department of Information and Computer Sciences, University of California, Irvine, USA, 1998.

- [9] L. Breiman. Hinging hyperplanes for regression, classification, and function approximation. *IEEE Transactions on Information Theory*, 39(3):999–1013, 1993.
- [10] L. Brieman, J. Friedman, C. J. Stone, and R. A. Olshen. Classification and regression trees. CA: Wadsworth, 1984.
- [11] J. Cao, Z. Lin, and G.-B. Huang. Composite function wavelet neural networks with extreme learning machine. *Neurocomputing*, 73:1405–1416, 2010.
- [12] J. Cao, Z. Lin, G.-B. Huang, and N. Liu. Voting based extreme learning machine. *Information Sciences*, 185:66–77, 2012.
- [13] C.-C. Chang and C.-J. Lin. LIBSVM: A library for Support Vector Machines, Available: <http://www.csie.ntu.edu.tw/~cjlin/libsvm>. 2001.
- [14] Y. Chauvin. A backpropagation algorithm with optimal use of hidden units. In *Advances in Neural Information Processing System 1*, pages 519–526. 1991.
- [15] F. L. Chen and T. Y. Ou. Sales forecasting system based on Gray extreme learning machine with Taguchi method in retail industry. *Experts Systems with Applications*, 38:1336–1345, 2011.
- [16] H. Chen, H.-H. Chen, X. Nian, and P. Lin. Ensembling extreme learning machine. In *Proceedings of International Symposium on Neural Networks*, pages 1069–1076, 2007.
- [17] J. S. Cho and H. White. Testing correct model specification using extreme learning machine. *Neurocomputing*, 74:2552–2565, 2011.
- [18] Y. L. Cun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in neural information processing systems 2*, pages 598–605, 1990.
- [19] I. Daubechies. Orthonormal bases of compatly supported wavelets. *Communications on Pure and Applied Mathematics*, 41(7):909–996, 1988.
- [20] I. Daubechies. The wavelets transform, time-frequency localization and signal analysis. *IEEE Transactions on Information Theorys*, 36(5):961–1005, 1990.

- [21] W.-Y. Deng, Q.-H. Zheng, S. Liang, L. Chen, and X. Wang. Ordinal extreme learning machine. *Neurocomputing*, 74:447–456, 2010.
- [22] G. Duan, Z. Huang, and J. Wang. Extreme learning machine for bank clients classification. In *Proceedings of International Conference on Information Management, Innovation Management and Industrial Engineering*, pages 496–499, 2009.
- [23] G. Feng, G.-B. Huang, Q. Lin, and R. Gay. Error minimized extreme learning machine with growth of hidden nodes and incremental learning. *IEEE Transactions on Neural Networks*, 20(8):1352–1357, 2009.
- [24] G. Feng, Z. Qian, and N. Dai. Reversible watermarking via extreme learning machine prediction. *Neurocomputing*, 82:62–68, 2012.
- [25] F. Fernandez-Navarro, C. Hervás-Martínez, J. Sánchez-Monedero, and P. A. Gutiérrez. A modified version of the extreme learning machine for generalized radial basis function neural networks. *Neurocomputing*, 74:2502–2510, 2011.
- [26] B. Frenay and M. Verleysen. Parameter-insensitive kernel in extreme learning machine for non-linear support vector regression. *Neurocomputing*, 74:2526–2531, 2011.
- [27] J. H. Friedman and W. Stuetzle. Projection pursuit regression. *Journal of American Statistical Association*, 76(376):817–823, 1981.
- [28] S. I. Gallant. Neural network learning and expert systems. MA: MIT Press, 1993.
- [29] N. García-Pedrajas, C. Hervás-Martínez, and J. Muñoz-Pérez. COVNET: a cooperative coevolutionary model for evolving artificial neural networks. *IEEE Transactions on Neural Networks*, 14(3):575–596, 2003.
- [30] S. Geva and J. Sitte. A constructive method for multivariate function approximation by multilayer perceptrons. *IEEE Transactions on Neural Networks*, 3(4):621–624, 1992.

- [31] C. L. Giles and T. Maxwell. Learning, invariance, and generalization in high-order neural networks. *Applied Optics*, 26(23):4972–4978, 1987.
- [32] M. T. Hagan, H. B. Demuth, and M. H. Beale. Neural network design. Boston: PWS Publishing Company, 1996.
- [33] F. Han and D.-S. Huang. Improved extreme learning machine for function approximation by encoding a priori information. *Neurocomputing*, 69:2369–2373, 2006.
- [34] L. K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.
- [35] B. Hassibi and D. G. Stork. Second order derivatives for network pruning: optimal brain surgeon. In *Advances in Neural Information Processing Systems 5*, pages 164–171, 1993.
- [36] S. Haykin. Neural networks: a comprehensive foundation. NY: Macmillan College Publishing Company, 1994.
- [37] D. O. Hebb. The organization of behavior. NY: Wiley, 1949.
- [38] M. V. Heeswijk, Y. Miche, T. Lindh-Knuutila, Peter-Hilbers, T. Honkela, E. Oja, and A. Lendasse. Adaptive ensemble models of extreme learning machines for time series prediction. *Lecture Notes in Computer Science*, 5769:305–314, 2009.
- [39] G. E. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40:185–234, 1989.
- [40] Y. Hirose, K. Yamashita, and S. Hijiya. Back-propagation algorithm which varies the number of hidden units. *Neural Networks*, 4:61–66, 1991.
- [41] A. E. Hoerl and R. W. Kennard. Ridge regression: biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.
- [42] V. G. Honavar and L. Uhr. Generative learning structures and processes for generalized connectionist networks. *Information Sciences*, 70:75–108, 1993.

- [43] J. J. Hopfield and D. W. Tank. Computing with neural circuits: a model. *Science*, 233(4764):625–633, 1986.
- [44] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.
- [45] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.
- [46] X.-F. Hu, Z. Zhao, S. Wang, F.-L. Wang, D.-K. He, and S.-K. Wu. Multi-stage extreme learning machine for fault diagnosis on hydraulic tube tester. *Neural Computing and Applications*, 17(4):399–403, 2008.
- [47] G.-B. Huang. Learning capability and storage capacity of two-hidden-layer feedforward networks. *IEEE Transactions on Neural Networks*, 14(2):274–281, 2003.
- [48] G.-B. Huang and H. A. Babri. Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions. *IEEE Transactions on Neural Networks*, 9(1):224–229, 1998.
- [49] G.-B. Huang and L. Chen. Convex incremental extreme learning machine. *Neurocomputing*, 70:3056–3062, 2007.
- [50] G.-B. Huang and L. Chen. Enhanced random search based incremental extreme learning machine. *Neurocomputing*, 71:3460–3468, 2008.
- [51] G.-B. Huang, L. Chen, and C.-K. Siew. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Transactions on Neural Networks*, 17(4):879–892, 2006.
- [52] G.-B. Huang, X. Ding, and H. Zhou. Optimization method based extreme learning machine for classification. *Neurocomputing*, 74:155–163, 2010.
- [53] G.-B. Huang, M.-B. Li, L. Chen, and C.-K. Siew. Incremental extreme learning machine with fully complex hidden nodes. *Neurocomputing*, 71:576–583, 2008.

- [54] G.-B. Huang, N.-Y. Liang, H.-J. Rong, P. Saratchandran, and N. Sundararajan. On-line sequential extreme learning machine. In *Proceedings of Computational Intelligence*, pages 232 – 237, 2005.
- [55] G.-B. Huang and C.-K. Siew. Extreme learning machine: RBF network case. In *Proceedings of 8th International Conference Control, Automation, Robotics and Vision*, pages 1029–1036, 2004.
- [56] G.-B. Huang and C.-K. Siew. Extreme learning machine with randomly assigned RBF kernels. *International Journal of Information Technology*, 11(1):16–24, 2005.
- [57] G.-B. Huang, N. Sundararajan, and P. Saratchandran. An efficient sequential learning algorithm for growing and pruning RBF (GAP-RBF) networks. *IEEE Transactions on System, Man and Cybernetics, Part B: Cybernetics*, 34(6):2284–2292, 2004.
- [58] G.-B. Huang and P. S. N. Sundararajan. A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation. *IEEE Transactions on Neural Networks*, 16(1):57–67, 2005.
- [59] G.-B. Huang, D. H. Wang, and L. Yuan. Extreme learning machines: a survey. *International Journal of Machine Learning and Cybernetics*, 2(2):107–122, 2011.
- [60] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang. Extreme learning machines for regression and multi-class classification. *IEEE Transactions on System, Man and Cybernetics, Part B: Cybernetics*, 42(2):513–529, 2012.
- [61] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: A new learning scheme of feedforward neural networks. In *Proceedings of International Joint Conference on Neural Networks*, pages 985–990, 2004.
- [62] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70:489–501, 2006.

- [63] G.-B. Huang, Q.-Y. Zhu, M.-K. Zhi, C.-K. Siew, P. Saratchandran, and N. Sundararajan. Can threshold networks be trained directly? *IEEE Transactions on Circuits and Systems II*, 53(3):187–191, 2006.
- [64] C. Ji, R. R. Snapp, and D. Psaltis. Generalizing smoothness constraints from discrete samples. *Neural Computation*, 2(2):188–197, 1990.
- [65] V. Kadirkamanathan and M. Niranjan. A function estimation approach to sequential learning with neural networks. *Neural Computation*, 5(6):954–975, 1993.
- [66] T. Kim and T. Adal. Approximation by fully complex multilayer perceptrons. *Neural Computation*, 15(7):1641–1666, 2003.
- [67] T. Kohonen. Self-organization and associative memory. Berlin: Springer-VerlagNeural, 1989.
- [68] J. K. Kruschke. Improving generalization in backpropagation with distributed bottlenecks. In *Proceedings of International Joint Conference on Neural Networks*, pages 443–447, 1989.
- [69] T.-Y. Kwok and D.-Y. Yeung. Constructive algorithms for structure learning in feedforward neural networks for regression problems. *IEEE Transactions on Neural Networks*, 8(3):630–645, 1997.
- [70] T.-Y. Kwok and D.-Y. Yeung. Objective functions for training new hidden units in constructive neural networks. *IEEE Transactions on Neural Networks*, 8(5):1131–1148, 1997.
- [71] Y. Lan, Y. C. Soh, and G.-B. Huang. Extreme learning machine based bacterial protein subcellular localization prediction. In *Proceedings of International Joint Conference on Neural Networks*, pages 1859–1863, 2008.
- [72] Y. Lan, Y. C. Soh, and G.-B. Huang. Ensemble of online sequential extreme learning machine. *Neurocomputing*, 72:3391–3395, 2009.

- [73] Y. Lan, Y. C. Soh, and G.-B. Huang. Constructive hidden nodes selection of extreme learning machine for regression. *Neurocomputing*, 73:3191–3199, 2010.
- [74] Y. Lan, Y. C. Soh, and G.-B. Huang. Two-stage extreme learning machine for regression. *Neurocomputing*, 73:3028–3038, 2010.
- [75] Y. Lecun, L. Bottou, G. B. Orr, and K.-R. Muller. Efficient backprop. *Lecture Notes in Computer Science*, 1524:5–50, 1998.
- [76] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, 6:861–867, 1993.
- [77] G. Li, M. Liu, and M. Dong. A new online learning algorithm for structure-adjustable extreme learning machine. *Computers and Mathematics with Applications*, 60(3):377–389, 2010.
- [78] M.-B. Li, G.-B. Huang, P. Saratchandran, and N. Sundararajan. Fully complex extreme learning machine. *Neurocomputing*, 68:306–314, 2005.
- [79] N.-Y. Liang, G.-B. Huang, S. P. Saratchandran, and N. Sundararajan. A fast and accurate online sequential learning algorithm for feedforward networks. *IEEE Transactions on Neural Networks*, 17(6):1411–1423, 2006.
- [80] J. Lim. Recursive DLS solution for extreme learning machine-based channel equalizer. *Neurocomputing*, 71:592–599, 2008.
- [81] Q.-J. B. Loh and S. Emmanuel. ELM for the classification of music genres. In *Proceedings of International Conference on Control, Automation, Robotics and Vision*, pages 1–6, 2006.
- [82] C. G. Looney. Pattern recognition using neural networks: theory and algorithms for engineers and scientists. NY: Oxford University Press, 1997.

- [83] Y. Lu, N. Sundararajan, and P. Saratchandran. A sequential learning scheme for function approximation using minimal radial basis function neural networks. *Neural Computation*, 9(2):461–478, 1997.
- [84] Y. Lu, N. Sundararajan, and P. Saratchandran. Performance evaluation of a sequential minimal radial basis function (RBF) neural network learning algorithm. *IEEE Transactions on Neural Networks*, 9(2):308–318, 1998.
- [85] L. A. Lusternik and V. J. Sobolev. Elements of functional analysis. Delhi: Hindustan Publishing Corporation, 1961.
- [86] V. Malathi, N. S. Marimuthu, and S. Baskar. Intelligent approaches using support vector machine and extreme learning machine for transmission line protection. *Neurocomputing*, 73:2160–2167, 2010.
- [87] J. M. Martinez-Martinez, P. Escandell-Montero, E. Soria-Olivas, J. D. Martin-Guerrero, R. Magdalena-Benedito, and J. Gomez-Sanchis. Regularized extreme learning machine for regression problems. *Neurocomputing*, 74:3716–3721, 2011.
- [88] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.
- [89] R. Meir and V. E. Maiorov. On the optimality of neural-network approximation using incremental algorithms. *IEEE Transactions on Neural Networks*, 11(2):323–337, 2000.
- [90] Y. Miche, A. Sorjamaa, and A. Lendasse. In *Proceedings of International Conference on Artificial Neural Networks*, pages 145–154, 2008.
- [91] Y. Miche, A. Sorjamaa, and A. Lendasse. OP-ELM: theory, experiments and a toolbox. *Lecture Notes in Computer Science*, 5163:145–154, 2008.
- [92] R. Minhas, A. A. Mohammed, and Q. M. J. Wu. A fast recognition framework based extreme learning machine using hybrid object information. *Neurocomputing*, 73:1831–1839, 2010.

- [93] M. L. Minsky. Theory of neural-analog reinforcement systems and its applicaion to the brain-model problem. Ph.D Thesis, Princeton University, 1954.
- [94] M. L. Minsky. Steps towards artificial intelligence. *Proceedings of the Institute of Radio Engineers*, 49(1):8–30, 1961.
- [95] M. L. Minsky and S. A. Papert. Perceptrons. MA: MIT Press, 1969.
- [96] A. A. Mohammed, R. Minhas, Q. M. J. Wu, and M. A. Sid-Ahmed. Human face recognition based on multidimensional PCA and extreme learning machine. *Pattern Recognition*, 44:2588–2597, 2011.
- [97] M. C. Mozer and P. Smolensky. Skeletonization: A technique for trimming the fat from a network via relevance assessment. In *Advances in neural information processing systems 1*, pages 107–115, 1989.
- [98] N. J. Nilsson. Learning machines: foundations of trainable pattern-classifying systems. NY: McGraw-Hill, 1965.
- [99] A. H. Nizar, Z. Dong, and Y. Wang. Power utility nontechnical loss analysis with extreme learning machine method. *IEEE Transaction on Power Systems*, 23(3):946–955, 2008.
- [100] S. J. Nowlan and G. E. Hinton. Simplifying neural networks by soft weight-sharing. *Neural Computation*, 4(4):473–493, 1992.
- [101] M. Pal. Extreme-learning-machine-based land cover classification. *Internatinal Journal of Remote Sensing*, 30(14):3835–3841, 2009.
- [102] R. Parekh, J. Yang, and V. Honavar. Constructive neural-network learning algorithms for patter classification. *IEEE Transactions on Neural Networks*, 11(2):436–451, 2000.
- [103] J. Park and I. W. Sandberg. Universal approximation using radial-basis-function networks. *Neural Computation*, 3(2):246–257, 1991.

- [104] J. Platt. A resource-allocating network for function interpolation. *Neural Computation*, 3(2):213–225, 1991.
- [105] D. C. Plaut, S. J. Nowlan, and G. E. Hinton. Experiments on learning by back propagation. Technical Report CMU-CS-86-126, Carnegie-Mellon University, 1986.
- [106] F. M. Pouzols and A. Lendasse. Evolving fuzzy optimally pruned extreme learning machine: a comparative analysis. In *Proceedings of International Conference on Fuzzy Systems*, pages 1–8, 2010.
- [107] C. R. Rao and S. K. Mitra. Generalized inverse of matrices and its applications. NY: Wiley, 1971.
- [108] R. Reed. Pruning algorithms - a survey. *IEEE Transaction on Neural Networks*, 4(5):740–747, 1993.
- [109] I. Rojas, H. Pomares, J. L. Bernier, J. Ortega, B. Pino, F. J. Pelayo, and A. Prieto. Time series analysis using normalized PG-RBF network with regression weights. *Neurocomputing*, 42:267–285, 2002.
- [110] H.-J. Rong, G.-B. Huang, N. Sundararajan, and P. Saratchandran. Online sequential fuzzy learning machine for function approximation and classification problem. *IEEE Transactions on System, Man and Cybernetics, Part B: Cybernetics*, 39(4):1067–1072, 2009.
- [111] H.-J. Rong, Y.-S. Ong, A.-H. Tan, and Z. Zhu. A fast pruned-extreme learning machine for classification problem. *Neurocomputing*, 72:359–366, 2008.
- [112] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychology Review*, 65(6):386–408, 1958.
- [113] H. L. Royden. Real analysis. NY: Macmillan College Publishing Company, 1968.
- [114] M. Salmeron, J. Ortega, C. G. Puntonet, and A. Prieto. Improved RAN sequential prediction using orthogonal techniques. *Neurocomputing*, 41:153–172, 2001.

- [115] R. Savitha, S. Suresh, and N. Sundararajan. Fast learning circular complex-valued extreme learning machine (CC-ELM) for real-valued classification problems. *Information Sciences*, 187:227–290, 2012.
- [116] D. Serre. *Matrices: theory and applications*. NY: Springer-Verlag, 2002.
- [117] Y. Shin and J. Ghosh. Approximation of multivariate functions using ridge polynomial networks. In *Proceedings of the International Joint Conference on Neural Networks*, pages 380–385, 1992.
- [118] J. Sietsma and R. J. F. Dow. Creating artificial neural networks that generalize. *Neural Networks*, 4(1):67–69, 1991.
- [119] T. Simila and J. Tikka. Multirespinse sparse regression with application to multi-dimensional scaling. *Lecture Notes in Computer Science*, 3697:97–102, 2005.
- [120] E. Soria-Olivas, J. Gomez-Sanchis, J. D. Martin, J. Vila-Frances, M. Martinez, J. R. Magdalena, and A. J. Serrano. BELM: bayesian extreme learning machine. *IEEE Transactions on Neural Networks*, 22(3):505–509, 2011.
- [121] Z.-L. Sun, T.-M. Choi, K.-F. Au, and Y. Yu. Sales forecasting using extreme learning machine with applications in fashion retailing. *Decision Support Systems*, 46(1):411–419, 2008.
- [122] S. Suresh, R. V. Babu, and H. J. Kim. No-reference image quality assessment using modified extreme learning machine classifier. *Applied Soft Computing*, 9:541–552, 2009.
- [123] S. Suresh, S. Saraswathi, and N. Sundararajan. Performance enhancement of extreme learning machine for multi-category sparse data classification problems. *Engineering Applications of Artificial Intelligence*, 23:1149–1157, 2010.
- [124] X. Tang and M. Han. Partial lanczos extreme learning machine for single-output regression problems. *Neurocomputing*, 72:3066–3076, 2009.

- [125] K.-A. Toh. Deterministic neural classification. *Neural Computation*, 20(6):1565–1595, 2008.
- [126] G. Wang, Y. Zhao, and D. Wang. A protein secondary structure prediction framework based on the extreme learning machine. *Neurocomputing*, 72:262 –268, 2008.
- [127] L. Wang, Y. Huang, X. Luo, Z. Wang, and S. Luo. Image deblurring with filters learned by extreme learning machine. *Neurocomputing*, 74:2464–2474, 2011.
- [128] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by weight-elimination with application to forecasting. In *Advances in Neural Information Processing System 3*, pages 875–882, 1991.
- [129] W. K. Wong and Z. X. Guo. A hybrid intelligence model for medium-term sales forecasting in fashion retail supply chains using extreme learning machine and harmony search algorithm. *International Journal of Production Economics*, 128:614–624, 2010.
- [130] C.-W. T. Yeu, M.-H. Lim, and G.-B. Huang. Terrain modelling using machine learning methods. In *Proceedings of International Conference on Control, Automation, Robotics and Vision*, pages 1–4, 2006.
- [131] J. Yin, F. Dong, and N. Wang. Modified gram-schmidt algorithm for extreme learning machine. In *Proceedings of International Symposium on Computational Intelligence and Design*, pages 517–520, 2009.
- [132] C.-M. Zhai and J.-X. Du. Applying extreme learning machine to plant species identification. In *Proceedings of International Conference on Information and Automation*, pages 879–884, 2008.
- [133] R. Zhang, G.-B. Huang, N. Sundararajan, and P. Saratchandran. Multicategory classification using an extreme learning machine for microarray gene expression cancer diagnosis. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 4(3):485–495, 2007.

- [134] R. Zhang, Y. Lan, G.-B. Huang, and Y. C. Soh. Extreme learning machine with adaptive growth of hidden nodes and incremental updating of output weights. *Lecture Notes in Artificial Intelligence*, 6752:253–262, 2011.
- [135] R. Zhang, Y. Lan, G.-B. Huang, and Z.-B. Xu. Universal approximation of extreme learning machine with adaptive growth of hidden nodes. *IEEE Transactions on Neural Networks and Learning Systems*, 23(2):365–371, 2012.
- [136] L. J. Zhao, X. K. Diao, D. C. Yuan, and W. Tang. Enhanced classification based on probabilistic extreme learning machine in wastewater treatment process. *Procedia Engineering*, 15:5563–5567, 2011.
- [137] X.-G. Zhao, G. Wang, X. Bi, P. Gong, and Y. Zhao. XML documents classification based on ELM. *Neurocomputing*, 74:2444–2451, 2011.
- [138] Q.-Y. Zhu, A. K. Qin, P. N. Suganthan, and G.-B. Huang. Evolutionary extreme learning machine. *Pattern Recognition*, 38:1759–1763, 2005.
- [139] W. Zong and G.-B. Huang. Face recognition based on extreme learning machine. *Neurocomputing*, 74:2541–2551, 2011.