Network ensemble and constructive algorithms for model selection of extreme learning machine

Lan, Yuan

2011

Lan, Y. (2011). Network ensemble and constructive algorithms for model selection of extreme learning machine. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/44760>

<https://doi.org/10.32657/10356/44760>

Network Ensemble and Constructive Algorithms for Model Selection of Extreme Learning Machine

LAN YUAN

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

2011

Acknowledgments

I would like to express my sincere and deepest gratitude to my supervisor Prof. Soh Yeng Chai and co-supervisor Assoc. Prof. Huang Guangbin for their continual guidance and endless patience. I am grateful to them for their valuable advices.

I also wish to convey my deepest appreciation and gratitude to my seniors and my friends for many fruitful discussions and generous helps.

My sincere appreciation also goes to all the technical staff, Mr. Tan Peng Chye and Mrs. Tham Chuay Siong from the Instrumentation and System Laboratory and Mr. Chia Chiu Kiat and Mr. Neo Kok Chuan from the Intelligent Robotics Laboratory, in the School of Electrical and Electronic Engineering for their technical support and logistic assistance.

Furthermore, I wish to thank Dr. Liang Nanying for providing me with the source codes for generating the data of Mackey-Glass application.

Last but not least, I would like to thank my parents and my sister for their love and support.

Table of Contents

Acknowledgments	i
Summary	vii
List of Figures	xii
List of Tables	xiv
1 Introduction	1
1.1 Research background	1
1.2 Motivations	4
1.3 Objectives	6
1.4 Major contributions	7
1.5 Organization	8
2 Extreme Learning Machines	11
2.1 Review of extreme learning machine	13
2.1.1 Capabilities of single-hidden layer feedforward networks	13
2.1.2 Mathematical preliminaries	16
2.1.3 SLFNs with random hidden nodes	18

2.1.4	Minimum norm least-squares solution of SLFNs	21
2.1.5	ELM algorithm for SLFNs	22
2.2	Variants of extreme learning machine	23
2.2.1	Online sequential extreme learning machine (OS-ELM)	28
2.2.2	Error minimized extreme learning machine (EM-ELM)	33
2.3	Summary	37
3	Constructive Algorithm and Model Selection Criteria	39
3.1	Motivation	39
3.1.1	Pruning algorithms	41
3.1.2	Advantages of constructive algorithm as compared to pruning algorithm	43
3.2	Design of constructive algorithms	44
3.2.1	Constructive training strategies and constructive algorithms	46
3.2.2	Model selection criteria	49
3.3	Summary	52
4	OS-ELM Ensemble and Constructive OS-ELM	53
4.1	OS-ELM ensemble	53
4.1.1	Ensemble of OS-ELM (EOS-ELM)	54
4.1.2	Performance comparison between EOS-ELM and OS-ELM	56
4.1.3	Performance comparison between EOS-ELM and other sequen- tial algorithms	60
4.1.4	Summary	62
4.2	Constructive OS-ELM	63

4.2.1	A constructive enhancement for OS-ELM (COS-ELM)	64
4.2.2	Performance evaluation of CEOS-ELM	69
4.2.3	Summary	75
5	Random Search Enhancement of Error Minimized Extreme Learning Machine	77
5.1	Introduction	77
5.2	Enhancement of EM-ELM (EEM-ELM)	78
5.3	Performance evaluation of EEM-ELM	79
5.3.1	Comparison of EEM-ELM and EM-ELM with the same number of hidden nodes	80
5.3.2	Comparison of EEM-ELM and EM-ELM with the same expected accuracy	81
5.3.3	Effect of factor k	82
5.4	Summary	83
6	Stepwise Forward Selection based Constructive Extreme Learning Machine for Regression	89
6.1	Introduction	89
6.2	Subset model selection	91
6.3	Proposed CS-ELM	92
6.3.1	Candidate reservoir	92
6.3.2	Constructive hidden nodes selection	94
6.3.3	Updating $\bar{y}$ recursively	98
6.3.4	Stopping criterion	99
6.3.5	Summary of CS-ELM algorithm	99

6.4	Performance evaluation of CS-ELM	102
6.4.1	Parameter selection	103
6.4.2	Performance evaluation of CS-ELM	106
6.5	Summary	111
7	Two-Stage Extreme Learning Machine for Regression	115
7.1	Introduction	115
7.2	Proposed two-stage ELM for regression (TS-ELM)	117
7.2.1	Problem formulation	117
7.2.2	Stage one – forward model selection	118
7.2.3	Stage two – backward elimination	124
7.2.4	Comparison of TS-ELM and FCA	127
7.3	Performance evaluation of TS-ELM	128
7.3.1	Parameter selection	129
7.3.2	Performance evaluation of TS-ELM	132
7.4	Summary	139
7.5	Proof of equation (7.9)	140
8	Conclusions and Recommendations	143
8.1	Conclusions	143
8.2	Recommendations for future work	146
	Author's Publications	149
	Bibliography	151

Summary

Machine learning (ML) is a branch of computer science dealing with computational algorithms that have learning capabilities. The learning approaches such as data clustering, artificial neural networks, genetic programming, and support vector machines have found wide applications in the fields of engineering, business and science. Among the different approaches, the greatest advantage of artificial neural networks (ANNs) is their ability to approximate an arbitrary function through learning from observed data. The single hidden layer feedforward neural networks (SLFNs) are probably the most popular ANNs and have been studied extensively.

The extreme learning machine (ELM) introduced by Huang et al. is a learning algorithm designed based on the generalized SLFNs with a wide variety of hidden nodes. It randomly generates hidden node parameters and then determines the output weights analytically. The key advantage of ELM is that it needs no iteration when determining the hidden node parameters, which dramatically reduces the computational time required for training the model. In addition, ELM is very simple and it tends to obtain the smallest training error and the smallest norm of weights, which can lead to good generalization performance of networks. However, the good performance of ELM is valid only when the network architecture is chosen correctly. For any application, a small network cannot learn the problem well, but a large network may lead to overfitting. Both of these cases result in poor generalization performance. The network architecture design for the original ELM relies on a trial and error method that may be very tedious for certain applications. Therefore, how to choose an appropriate network structure remains an open problem for ELM.

An ELM ensemble is sometimes used to alleviate this problem. It encourages stable network outputs by averaging the outputs of several ELM networks when the fluctuation of individual ELM network is large. However, it cannot alter the topology for individual networks in the ensemble. A recently proposed idea is to have an integration of ELM that connects several ELM networks in parallel and the average output of these ELMs is considered as the final output of the integration. Inspired by this ensemble idea, we propose an ensemble of online sequential ELM (OS-ELM) that consists of several OS-ELM networks. The main advantage of OS-ELM ensemble as compared to the individual OS-ELM is that it is able to reduce the influence of an individual network that does not work well in one round of learning, and hence producing a much stabler network.

Another solution to tackle the problem of architecture design of ELM is to alter the network structure during the training process. There are normally two heuristic approaches to alter the network structure, including constructive (or growing) approaches and destructive (or pruning) approaches. Some improvements of ELM have already been proposed according to the two approaches. Incremental ELM (I-ELM) and error minimized ELM (EM-ELM) belong to the constructive methods; while pruned ELM (P-ELM) and optimally-pruned ELM (OP-ELM) are the destructive methods. Similar to the preliminary ELM, OS-ELM relies on a fixed network structure that is determined by a trial and error method and so it could be rather time consuming. Motivated by EM-ELM that starts with a small network and adds the random hidden nodes one-by-one or group-by-group, we propose a constructive improvement of OS-ELM, which we termed CEOS-ELM. It modifies the structure of OS-ELM by adding randomly generated hidden nodes to the hidden layer during the sequential learning process.

While researching on the constructive improvement of OS-ELM, we found that some hidden nodes in the EM-ELM network may become insignificant after some new hidden nodes are added to the network. Thus, we propose that a selection phase should be added to the original EM-ELM to alleviate the problem.

Although I-ELM and EM-ELM are good incremental learning algorithms that allow

the hidden nodes to be added to the network during the training process, there is a shortcoming in both algorithms in that we do not know when to stop the algorithms. In other words, both of them could not provide a solution to find the ideal network structure automatically for given applications. When investigating the pruning algorithms, we found that P-ELM is terminated by a criterion based on the Akaike information criterion (AIC); while OP-ELM determines the final number of hidden nodes by using leave-one-out cross validation. Encouraged by P-ELM and OP-ELM, we propose a constructive enhancement of ELM (referred to as CS-ELM) based on stepwise forward selection for regression applications. The proposed algorithm attempts to select the hidden node one-by-one from the candidate reservoir and the selection is stopped by the unbiased risk estimation criterion C_p .

Similar to many other constructive learning algorithms, in CS-ELM, the hidden nodes added earlier may become insignificant when new hidden nodes are added to the network. We solve the problem by applying a two-stage algorithm referred to as TS-ELM. In the first stage, hidden nodes can be selected one-by-one or group-by-group according to the net contribution and then the selected hidden nodes are added to the network. The second stage of the algorithm is the re-evaluation stage, in which the insignificant hidden nodes are removed from the network.

Extensive simulations have been conducted to evaluate the proposed algorithms. Empirical studies showed that the OS-ELM ensemble is much stabler than individual OS-ELM network in any round of learning. In addition, for individual OS-ELM network, CEOS-ELM is able to build the network constructively and it can achieve comparable performance as the original OS-ELM network. With the added selection phase, the enhanced EM-ELM obtains the same training accuracy with fewer hidden nodes in the hidden layer. Compared to EM-ELM, our proposed CS-ELM or TS-ELM could find an appropriate network structure for regression applications automatically. Simulation results showed that CS-ELM leads to a more compact or comparable network structure as compared to the traditional trial and error method. Furthermore, the parsimonious network structure can be obtained using TS-ELM by adding a refinement phase to the

constructive learning algorithm like CS-ELM, which is shown in the evaluations of CS-ELM and TS-ELM for the same benchmark applications.

List of Figures

2.1	Fully connected single-hidden layer feedforward network	14
4.1	Learning evolution for Auto-MPG case (sigmoid hidden nodes)	61
4.2	Learning evolution for Auto-MPG case (RBF hidden nodes)	62
5.1	The testing RMSE updating curves of EEM-ELM and EM-ELM	81
5.2	Performance comparison of EEM-ELM(10) and EM-ELM with sigmoidal additive hidden nodes on Abalone case	82
5.3	Performance comparison of EEM-ELM(10) and EM-ELM with sigmoidal additive hidden nodes on Auto-MPG case	83
5.4	Performance comparison of EEM-ELM(10) and EM-ELM with sigmoidal additive hidden nodes on Image Segmentation case	84
5.5	Performance comparison of EEM-ELM with different values of factor k with sigmoidal additive hidden nodes on Boston Housing case	87
6.1	Parameter selection for Delta elevators case, Ailerons case and Bank domains case - testing RMSE	104
6.2	Parameter selection for Delta elevators case, Ailerons case and Bank domains case - selected hidden nodes	105
6.3	Effect of C_p on Abalone case, number of hidden nodes selected $p \in [10, 50]$	107

6.4	Effect of C_P on Ailerons case, number of hidden nodes selected $p \in [10, 70]$	108
6.5	Effect of C_P on Delta elevators case, number of hidden nodes selected $p \in [5, 35]$	109
6.6	Average number of hidden nodes selected on Abalone case	110
6.7	Average number of hidden nodes selected on Ailerons case	111
6.8	Average number of hidden nodes selected on Delta elevators case	112
6.9	Learning evolution for Delta elevators case	112
7.1	Parameter selection for the Auto-MPG case	130
7.2	Parameter selection for the Abalone case	131
7.3	Parameter selection for the Ailerons case	131
7.4	Effect of FPE on Abalone case ($p^* = 11$)	133
7.5	Effect of FPE on Auto-MPG case ($p^* = 9$)	134
7.6	Effect of FPE on Delta elevators case ($p^* = 9$)	135
7.7	Effect of N_{Group} on Ailerons case: training time for stage one and stage two	137
7.8	Effect of N_{Group} on Ailerons case: number of hidden nodes selected for stage one and stage two	137
7.9	Effect of N_{Group} on Ailerons case: training and testing RMSE	138

List of Tables

4.1	Specification of benchmark datasets for evaluation of EOS-ELM	56
4.2	EOS-ELM model selection for New-thyroid case	58
4.3	Comparison between EOS-ELM and OS-ELM (sigmoid hidden nodes) .	59
4.4	Comparison between EOS-ELM and OS-ELM (RBF hidden nodes) . .	60
4.5	Comparison between EOS-ELM and NCL on classification applications	62
4.6	Specification of benchmark datasets for evaluation of CEOS-ELM . . .	70
4.7	Parameters setting for regression (Abalone and California Housing) and classification (Image Segmentation and Satellite Image) problems . . .	71
4.8	Learning mode comparison on regression cases (sigmoidal additive hid- den nodes, $L_0 = 1, \delta L_j = 1$)	72
4.9	Learning mode comparison on classification cases (sigmoidal additive hidden nodes, $L_0 = 1, \delta L_j = 1$)	73
4.10	Incremental mode comparison on regression cases (sigmoidal additive hidden nodes, $N_k = 20, L_0 = 1$)	74
4.11	Incremental mode comparison on classification cases (sigmoidal addi- tive hidden nodes, $N_k = 20$)	75
5.1	Specification of benchmark datasets for evaluation of EEM-ELM	80

5.2	Performance comparison between EEM-ELM and EM-ELM with sigmoidal additive hidden nodes	85
5.3	Performance comparison between EEM-ELM and EM-ELM with RBF hidden nodes	86
6.1	Specification of benchmark datasets for evaluation of CS-ELM	103
6.2	Parameters setting for benchmark regression cases	106
6.3	Performance comparison	113
7.1	Specification of benchmark datasets for evaluation of TS-ELM	129
7.2	Parameters setting for benchmark regression cases	132
7.3	Performance comparison: hidden nodes are added one-by-one	139
7.4	Performance comparison: hidden nodes are added group-by-group	140
8.1	Feature comparison among EEM-ELM, CS-ELM and TS-ELM	145

Chapter 1

Introduction

1.1 Research background

Machine learning (ML) [1] is the study of computational algorithms that have the capability to learn from previous experiences so as to improve their performance for certain tasks. It is closely related to pattern recognition and statistical inference. The learning approaches such as data clustering, artificial neural networks, genetic programming, and support vector machines have found wide applications in the fields of engineering, business and science. Although different scientific fields have their own versions of scientific processes, they share common stages in one cycle of the process: the procedure may start from observing, creating hypotheses, and testing by decisive experiments or observations, and then iteratively building up comprehensive testable models or theories [2].

Due to the fast improvements in the acquisition process, it is possible to obtain or observe large amounts of information about a specific phenomenon, which makes the available data for analysis more abundant in terms of number of variables and samples. Therefore, in the early stage of a scientific process, i.e. observing and creating hypotheses, one is increasingly faced with the situation of high data volume, high data

acquisition rates, and/or requirements for objective analysis that cannot be handled by human perception alone. Hence, much of current ML research focuses on visualization and exploration of high-dimensional data and many automatic pattern recognition approaches have been developed. However, high-dimensional feature vectors may not be suitable for subsequent recognition process in many aspects. Therefore, parsimonious pattern representations with salient features are desired to improve the efficiency and effectiveness of the recognizer. Dimensionality reduction, i.e. reducing the original many dimensions of the data to just a few dimensions in some new space, could be a feasible solution for the above problem, which is normally achieved by feature extraction and feature selection. Those dimensionality reduction techniques that capture the significant features can be helpful for building a less complex model.

Choosing a suitable model is central to all ML tasks with data. The desired model should be rich enough to explain essential relations in the data, easy to understand, and simple to use. Hence, many model selection methods come into play when we want to balance goodness of fit and simplicity. Given the data, statistical model selection is the task of selecting a statistical model from the set of candidate models, which is an integration of almost any data analysis. For model selection, there are many key general issues [3]:

- (1) Models are approximations: In most situations, it is difficult to obtain the true model that generates the collected data, because it might be extremely complex. Therefore, it is more practical to derive a simpler model that can adequately explain the relations in the data.
- (2) The bias-variance trade-off: Model selection techniques can be considered as estimators of some physical quantity, and the bias and variance are both important measures of the quality of this estimator. In the framework of model fitting and selection, if a simple estimator with few parameters is involved, it may lead to lower variability but may be associated with modelling bias; whereas a complex estimator with more parameters would mean a higher degree of variability but small modelling bias. Statistical model selection intends to seek a proper balance

between simplicity and complexity.

- (3) Parsimony: “The principle of parsimony” is employed in many areas, including philosophy, physics, arts, communications as well as statistics. For example, *Ockham’s Razor* [4] suggested that when the predictive powers of competing theories are equal, one should choose the theory that introduces the fewest assumptions. In statistical modelling, if many models fit the data equally well, a simpler model is preferred [5].
- (4) The context and focus: Any modelling problem is conducted for certain purpose within an appropriate scientific context. However, “the context” may not be precisely defined sometimes, and researchers might discover or learn different things from the same dataset. In addition, preferences when modelling and analyzing the data may vary from one school of science to another. Therefore, the model selection methods could vary in different scientific cultural background.
- (5) Model averaging: Most of the model selection methods work by comparing all the candidate models and selecting the one with highest performance. However, in some cases, there might not be a clear winner where several candidates do almost as well as the winner. In such cases, ensemble methods [6] that seek to combine several best models in an optimal way are considered.

Artificial neural network (ANN) or simply “neural network” is one of the powerful learning approaches of ML. The greatest advantage of ANNs is their ability to approximate an arbitrary function through learning from observed data [7, 8, 9]. However, many aspects need to be considered before employing ANNs:

- (1) Choice of models: Similar to almost all the learning approaches of ML, choosing a suitable model is essential for using ANNs, which depends on the data representations and applications. An overly complex model may lead to overfitting problem, while a simple model with too few parameters may underfit the data.

- (2) Learning algorithm: There are numerous algorithms available for training ANNs models. Selecting and tuning the algorithm for training unseen data may require numerous experiments.

Single hidden-layer feedforward neural networks (SLFNs) are probably the most popular ANNs due to their universal approximation capabilities and hence they have been widely exploited in various applications. There are a lot algorithms proposed based on SLFNs, and extreme learning machine (ELM) [10, 11, 12, 13, 14] is one of them. ELM is originally designed for neural networks with additive or RBF hidden nodes, it could be extended to generalized single hidden-layer feedforward networks with multiplicative nodes [10], such as fuzzy rules [15], fully complex nodes[16, 17], hinging functions [18], high-order nodes [19], ridge polynomials [20], wavelets [21, 22], and Fourier series [23], etc. In this thesis, the research work would focus on the network architecture design of ELM.

1.2 Motivations

Huang et al. [12] has shown that SLFNs with randomly assigned hidden nodes parameters and with almost any nonlinear piecewise continuous activation function are universally approximators for any target continuous functions on any compact datasets. Based on this finding, extreme learning machine (ELM) [10, 11, 12, 13, 14] is introduced on generalized SLFNs with a wide variety of hidden nodes, including those that may not be neuron like. ELM randomly generates hidden nodes parameters and then determines the output weights analytically by the minimum norm least-squares solutions of a general system of linear equations. The training speed of ELM can be of thousand times faster than the classical methods (such as back-propagation algorithm (BP) [24] and support vector machine (SVM) [25]). Additionally, ELM is very simple and it tends to obtain the smallest training error and the smallest norm of output weights, which can lead to good generalization performance of networks. However, the good performance

of ELM is only valid when the network architecture is chosen correctly. For any application, a smaller than desired network cannot learn the problem well, but a larger network may lead to overfitting [26]. Both of these cases result in poor generalization performance. For ELM, the only parameter of network architecture design is the number of hidden nodes in the hidden layer and it is usually determined by a trial and error method, in which many networks in various sizes are trained and the one with highest generalization performance is picked. The trial and error method is really inefficient because many networks have to be trained before an acceptable network can be found and the solution may not always be optimal. Therefore, how to choose an appropriate parsimonious network model that fits the data remains an open problem for ELM.

Neural network ensembles can be used to improve the networks' generalization ability [27]. In addition, network with regularization is sometimes used to prevent overfitting by introducing a penalty term to the cost function for large network structure. However, both network ensemble and network with regularization cannot alter the network topology.

Another solution to tackle the problem of architecture design of ELM is to alter the network structure during the training process. There are normally two heuristic approaches to alter the network structure: the constructive (or growing) approaches and the destructive (or pruning) approaches. Several improvements of ELM have been proposed according to these two approaches. Incremental ELM (I-ELM) [12] and error minimized ELM (EM-ELM) [28] belong to the constructive methods; while pruned ELM (P-ELM) [29] and optimally-pruned ELM (OP-ELM) [30] are the destructive methods. The network with pruning algorithm usually converges quickly and is less sensitive to local minima. However, there are also some disadvantages of pruning algorithms, such as the problem of determining the starting network structure to initiate pruning, and the issue of the high computational complexity as the network always starts at a larger size than necessary. Besides, I-ELM and EM-ELM are two constructive improvements of preliminary ELM, but they cannot determine the parsimonious network structure automatically. Therefore, in this thesis, our research would mainly focus on the constructive

network architecture design of ELM.

1.3 Objectives

To ensure a good generalization performance, the primary goal of this thesis is to devise algorithms that can find appropriate parsimonious network structure for ELM and its variants on various applications. More specifically, the objectives of this thesis can be summarized as follows:

- (1) To improve the stability of online sequential ELM (OS-ELM). A shortcoming of OS-ELM is that its solution usually fluctuates due to the randomly generated hidden nodes parameters in each round of simulation. Therefore, we attempt to improve the stability of OS-ELM and ensure good generalization performance in each round of simulation.
- (2) To develop a constructive algorithm for OS-ELM. Similar to the preliminary ELM, OS-ELM relies on a fixed network structure pre-determined by a trial and error method that could be rather time consuming. Besides, all the data must be available when applying the trial and error method, which contradicts sequential arrival of the data in online learning. Therefore, an effective algorithm to adjust the network structure during sequential learning process is desired.
- (3) To improve EM-ELM. It is found that some of the hidden nodes in EM-ELM network play a very minor role in the network output but they increase the network complexity. We attempt to address this problem.
- (4) To develop a constructive algorithm for preliminary ELM such that it can be terminated automatically. As mentioned earlier, both I-ELM and EM-ELM are terminated by the parameters given by the user. We intend to introduce some statistical criteria to determine the network structure rather than manually tuning the parameters.

- (5) To develop a constructive algorithm for preliminary ELM so as to provide an appropriate parsimonious network structure that can explain the data. We noted that using a constructive algorithm alone may not guarantee an parsimonious solution because the best solution could be missed by applying a greedy constructive algorithm. Therefore, a reinforcement phase is required to re-evaluate the selected hidden nodes.

1.4 Major contributions

The major contributions of this thesis can be summarized as follows:

- (1) **EOS-ELM**: An ensemble of OS-ELM (called EOS-ELM) is derived to improve the stability of OS-ELM. It is shown that the ensemble can improve the network's generalization ability. EOS-ELM encourages stable network outputs by averaging the outputs of several OS-ELM networks when the fluctuation of individual OS-ELM network is large. Empirical studies show that the performance of EOS-ELM is more reliable as compared to a single OS-ELM network.
- (2) **CEOS-ELM**: A constructive enhancement of OS-ELM (called CEOS-ELM) is derived based on EM-ELM, which can modify the network structure by adding randomly generated hidden nodes to the hidden layer during the sequential learning process. Simulation results show that the CEOS-ELM can provide comparable generalization performance with a compact network structure.
- (3) **EEM-ELM**: An enhancement of EM-ELM (called EEM-ELM) is proposed by applying a selection phase before the addition of hidden nodes, and it improves the convergence rate of the original EM-ELM and reduces the network complexity.
- (4) **CS-ELM**: A constructive improvement of ELM (called CS-ELM) is proposed based on stepwise forward selection for regression applications. The proposed

algorithm attempts to select the hidden node one-by-one from the candidate reservoir and the selection is stopped by the unbiased risk estimation criterion C_P .

- (5) **TS-ELM:** A two-stage algorithm (called TS-ELM) is developed. In the first stage, hidden nodes can be selected one-by-one or group-by-group according to the net contribution and then the selected hidden nodes are added to the network. The second stage of the algorithm is the re-evaluation stage, in which the insignificant hidden nodes are removed from the network.

1.5 Organization

The rest of the thesis is organized as follow:

Chapter 2 briefly reviews the preliminary ELM that is designed based on SLFNs to provide the background knowledge for the proposed algorithms in this thesis. Some existing improvements of ELM are also reviewed in this chapter.

Chapter 3 presents the motivations and the usage of constructive algorithms on ANN as well as some commonly used statistical criteria to terminate constructive algorithms.

Chapter 4 introduces an ensemble of OS-ELM that integrates many single OS-ELM networks. It also includes the proposal of a constructive OS-ELM based on EM-ELM that enables the network to adjust the hidden nodes during the sequential learning process.

In chapter 5, a selection phase is proposed to improve the performance of EM-ELM. One of several randomly generated hidden nodes is selected in each learning step.

Chapter 6 develops a constructive hidden node selection ELM based on stepwise forward selection for regression applications. A hidden nodes candidate reservoir is first established followed by a constructive hidden node selection that selects hidden node one-by-one from the reservoir. Mallows' C_P criterion is used to terminate the selection.

Chapter 7 develops a two-stage algorithm on the basis of preliminary ELM. A forward recursive algorithm is applied to select the hidden nodes from the candidate pool in the first stage. Then the selected hidden nodes are reviewed by leave-one-out cross validation in the second stage.

Chapter 8 presents the conclusions of this thesis and it also contains several recommendations for further research.

Chapter 2

Extreme Learning Machines

Artificial neural networks (or simply neural networks) that have been motivated by the way human brain calculates is quite different from that of conventional computers. The first artificial neuron was introduced by neurophysiologist Warren McCulloch and logician Walter Pits in 1943 [31], and this signaled the birth of neural networks as a discipline. The findings of McCulloch and Pits spread widely and many researchers were influenced by their work until 1960s. In 1969, a book written by Minsky and Papert [32] presented the limitations of single layer perceptrons as well as multilayered systems. The book and many other factors significantly slowed the development of neural networks. Resurgence of interest in neural networks only took place in 1980s when a firm theoretical foundation was established and powerful design of neural networks appeared. There are two information-processing capabilities that make neural networks capable of solving complex problems that are normally intractable. First, the massively parallel distributed structure of neural network makes it powerful in computation. Second, neural network has the ability to learn and therefore generalize. Currently, neural networks are useful tools for solving many types of problems, which can be grouped into broad categories of function approximation (or regression analysis), classification including pattern and sequence recognition etc, and data processing including filtering, clustering etc.

Among all the neural networks, feedforward neural networks have been the most extensively used in many fields because of their powerful nonlinear mapping abilities. They are capable of providing the model for a large class of natural and artificial phenomena, and they outperform the classical parametric techniques. Nevertheless, the learning algorithms for neural networks are usually far slower than desired. For some applications, it may take several days or more to train the networks by using traditional algorithms.

Conventionally, all the parameters of feedforward networks are tuned during the learning process and finally there is dependency between different layers of parameters (i.e. weights and biases). The gradient descent-based methods are part of the traditional learning methods, which have been widely used in various learning algorithms of feedforward neural networks. They give satisfactory responses once they have been properly trained. However, they could not be used to train the networks with threshold or digital nodes directly as the threshold functions are non-differentiable. Moreover, in some practical applications of neural networks, a fast response to external events within an extremely short time is expected. The gradient-descent based methods are probably not good choices when frequent retraining is needed, especially for complex (and large scale) applications and when higher learning accuracy and generalization performance are required.

One decade ago, Baum [33] 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. This implied that the input weights can be fixed and one may only adjust the output weights. Unlike the traditional methods where all the parameters have to be tuned during the learning process, it is proved by Huang [13] that the input weights and hidden layer biases of single-hidden layer feedforward neural networks can be randomly assigned.

Based on this concept, a new algorithm called extreme learning machine (ELM) [10, 11, 12, 13, 14] is proposed. ELM is different from other learning algorithms for feedforward neural networks in that it randomly selects all the hidden node parameters, after

which the network can be represented as a linear system and the output weights can be computed analytically. ELM is simple. Most importantly, it tends to obtain the smallest training error and the smallest norm of weights, which can lead to good generalization performance of networks that are proved theoretically. Moreover, some simulation results on artificial and real large applications have shown that ELM is extremely fast in learning and provides good generalization performance [13]. Huang et al. has also proved the universal approximation capability of ELM [10, 12].

ELM and its variants have been applied to various types of applications including palm-print recognition system [34], no-reference image quality assessment [35], bank clients classification [36], sales forecasting in fashion retailing [37], arrhythmia classification in ECG [38], bacterial protein subcellular localization prediction [39], microarray gene expression cancer diagnosis [40], prediction of melting point of organic compounds [41], fault diagnosis on hydraulic tube tester [42], power utility non-technical loss analysis [43], land cover classification [44], terrain modelling [45], plant species identification [46], and classification of music genres [47], etc.

2.1 Review of extreme learning machine

2.1.1 Capabilities of single-hidden layer feedforward networks

The ELM introduced by Huang et al. is originally designed based on the single-hidden layer feedforward networks (SLFNs). SLFNs have the ability to universally approximate continuous functions [7, 8, 9, 48, 49]. In past decades, they have been used in various areas, such as regression, classification, etc.

Figure 2.1: Fully connected single-hidden layer feedforward network

Model of an SLFN

An SLFN is presented in Fig. 2.1, which consists of three layers, namely input layer, hidden layer and output layer. There are n input nodes in the input layer that is the dimension of the observations. The hidden layer is composed of L hidden nodes. For SLFN, the activation functions of hidden nodes are usually nonlinear and defined by G . The output of the i th hidden node is denoted by $G(\mathbf{a}_i, b_i, \mathbf{x})$, where $\{(\mathbf{a}_i, b_i)\}_{i=1}^L$ are called the hidden node parameters. The aim of the hidden layer is to provide a more complicated network structure, which can manage higher-order statistics. The output layer has one output unit. The input signal travels from the input layer to the hidden layer, and finally the output layer. The output of the output layer reflects the overall response of the network to the applied input signals.

Consider N arbitrary distinct samples $(\mathbf{x}, \mathbf{t}) \in \mathbb{R}^n \times \mathbb{R}^m$. The output of SLFN with L

hidden nodes (additive or RBF nodes) can be represented by

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

where $\mathbf{a}_i$ and b_i are the learning parameters of the hidden nodes, β_i is the output weight, and $G(\mathbf{a}_i, b_i, \mathbf{x})$ denotes the output of the i th hidden node with respect to the input $\mathbf{x}$.

If the hidden node is an additive type with activation function $g(x)$, the mathematical representation is given by

$$G(\mathbf{a}_i, b_i, \mathbf{x}) = g(\mathbf{a}_i \cdot \mathbf{x} + b_i), \quad b_i \in \mathbb{R} \quad (2.2)$$

where $\mathbf{a}_i$ is the input weight vector, b_i is the bias of the i th hidden node, and $\mathbf{a}_i \cdot \mathbf{x}$ denotes the inner product of the two. The activation function $g(x)$ for additive nodes can be any bounded nonconstant piecewise continuous functions.

If the hidden node is a radial basis function (RBF) node with activation function $g(x)$, then the mathematical model is given by

$$G(\mathbf{a}_i, b_i, \mathbf{x}) = g(b_i \|\mathbf{x} - \mathbf{a}_i\|), \quad b_i \in \mathbb{R}^+, \quad (2.3)$$

where $\mathbf{a}_i$ and b_i are respectively the center and impact width of the i th RBF node, and $\mathbb{R}^+$ indicates the set of all positive real values. The activation function $g(x)$ for RBF nodes can be any integrable piecewise continuous functions.

Huang et al. [10] extended the preliminary ELM from SLFNs with additive or RBF hidden nodes to “generalized” SLFNs with a wide variety of hidden nodes. These hidden nodes can be any type of piecewise continuous hidden nodes including additive or RBF type of nodes, multiplicative nodes, fuzzy rules, fully complex nodes, hinging functions, high-order nodes, ridge polynomials, wavelets, and Fourier terms, etc.

Approximation capabilities of SLFNs

Research on the approximation capabilities of SLFNs has mainly focused on two aspects: universal approximation on compact input sets and approximation in a set consisting of a finite number of training samples.

According to conventional neural network theories, SLFNs with additive or radial basis function (RBF) hidden nodes are universal approximators when all the parameters of the networks can be adjusted. Hornik [8] reported that if the activation function is continuous, bounded, and not a constant, then continuous mappings can be approximated by SLFNs with additive hidden nodes over compact input sets. Leshno [50] worked on the basis of Hornik's results [8] and proved that SLFNs with additive hidden nodes and with a non-polynomial activation function can approximate any continuous target functions. And for RBF networks, Park and Sandberg [48] 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 [51] established that a SLFN with at most N hidden neurons and with almost any nonlinear activation function can learn N distinct observations with zero error. In [52], 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 [12] has proved theoretically that the SLFNs with randomly generated hidden nodes are indeed universal approximators.

2.1.2 Mathematical preliminaries

The Moore-Penrose generalized inverse and the minimum norm least-squares solution of a general linear system are introduced in this section as they are essential for the

implementation of ELM.

- Moore-Penrose generalized inverse

The resolution of a general linear system $\mathbf{Ax} = \mathbf{y}$, where $\mathbf{A}$ may be singular and may even be a non-square matrix, can be made very simple by the use of the Moore-Penrose generalized inverse [53].

Definition 2.1 [53, 54] 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

$$\mathbf{AGA} = \mathbf{A}, \quad \mathbf{GAG} = \mathbf{G}, \quad (\mathbf{AG})^T = \mathbf{AG}, \quad (\mathbf{GA})^T = \mathbf{GA} \quad (2.4)$$

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

- Minimum norm least-squares solution of general linear system

For a general linear system $\mathbf{Ax} = \mathbf{y}$, we say that $\hat{\mathbf{x}}$ is a least-squares solution if

$$\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{y}\| = \min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{y}\| \quad (2.5)$$

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

Definition 2.2 $\mathbf{x}_0 \in \mathbb{R}^n$ is said to be a minimum norm least-squares solution of a linear system $\mathbf{Ax} = \mathbf{y}$ if for any $\mathbf{y} \in \mathbb{R}^m$, $\|\mathbf{x}_0\| \leq \|\mathbf{x}\|$,

$$\forall \mathbf{x} \in \{\mathbf{x} : \|\mathbf{Ax} - \mathbf{y}\| \leq \|\mathbf{Az} - \mathbf{y}\|, \quad \forall \mathbf{z} \in \mathbb{R}^n\} \quad (2.6)$$

That means that a solution $\mathbf{x}_0$ is a minimum norm least-squares solution of a linear system $\mathbf{Ax} = \mathbf{y}$ if it has the smallest norm among all the least-squares solutions.

Theorem 2.1 (p. 147 of [53], p. 51 of [54]) Let there exist a matrix $\mathbf{G}$ such that $\mathbf{Gy}$ is a minimum norm least-squares solution of a linear system $\mathbf{Ax} = \mathbf{y}$. Then it

is necessary and sufficient that $\mathbf{G} = \mathbf{A}^\dagger$, the Moore-Penrose generalized inverse of matrix $\mathbf{A}$.

Remark 2.1 Form Theorem 2.1, we can have the following properties that are important for our proposed ELM learning algorithm:

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

$$\|\mathbf{Ax}_0 - \mathbf{y}\| = \|\mathbf{AA}^\dagger \mathbf{y} - \mathbf{y}\| = \min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{y}\| \quad (2.7)$$

- (2) Further, the special solution $\mathbf{x}_0 = \mathbf{A}^\dagger \mathbf{y}$ has the smallest norm among all the least-squares solutions of $\mathbf{Ax} = \mathbf{y}$:

$$\begin{aligned} \|\mathbf{x}_0\| &= \|\mathbf{A}^\dagger \mathbf{y}\| \leq \|\mathbf{x}\| \\ \forall \mathbf{x} &\in \{\mathbf{x} : \|\mathbf{Ax} - \mathbf{y}\| \leq \|\mathbf{Az} - \mathbf{y}\|, \quad \forall \mathbf{z} \in \mathbb{R}^n\} \end{aligned} \quad (2.8)$$

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

2.1.3 SLFNs with random hidden nodes

Consider N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i) \in \mathbb{R}^n \times \mathbb{R}^m$. If an SLFN with L hidden nodes can approximate these N samples with zero error, it then implies that there exist $\beta_i, \mathbf{a}_i$ and b_i such that

$$f_L(\mathbf{x}_j) = \sum_{i=1}^L \beta_i G(\mathbf{a}_i, b_i, \mathbf{x}_j) = \mathbf{t}_j, \quad j = 1, \dots, N. \quad (2.9)$$

Equation (2.9) can be written compactly as:

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{T} \quad (2.10)$$

where

$$\mathbf{H}(\mathbf{a}_1, \dots, \mathbf{a}_L, b_1, \dots, b_L, \mathbf{x}_1, \dots, \mathbf{x}_N) = \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_1) & \cdots & G(\mathbf{a}_L, b_L, \mathbf{x}_1) \\ \vdots & \cdots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_N) & \cdots & G(\mathbf{a}_L, b_L, \mathbf{x}_N) \end{bmatrix}_{N \times L} \quad (2.11)$$

$$\boldsymbol{\beta} = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_L^T \end{bmatrix}_{L \times m} \quad \text{and} \quad \mathbf{T} = \begin{bmatrix} \mathbf{t}_1^T \\ \vdots \\ \mathbf{t}_N^T \end{bmatrix}_{N \times m} \quad (2.12)$$

$\mathbf{H}$ is called the hidden layer output matrix of the network [52]; the i th column of $\mathbf{H}$ is the i th hidden node's output vector with respect to inputs $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$ and the j th row of $\mathbf{H}$ is the output vector of the hidden layer with respect to input $\mathbf{x}_j$.

Theorem 2.2 [13] *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 $\{\mathbf{x}_i | \mathbf{x}_i \in \mathbb{R}^n, i = 1, \dots, N\}$, and $\{(\mathbf{a}_i, b_i)\}_{i=1}^N$ that is randomly generated from any intervals of $\mathbb{R}^n$ and $\mathbb{R}$ respectively according to any continuous probability distribution, the hidden layer output matrix $\mathbf{H}$ is invertible with probability one.*

In other words, if the number of hidden nodes L is equal to the number of distinct training samples N , i.e. $L = N$, matrix $\mathbf{H}$ is square and invertible when the input parameters $\mathbf{a}_i, b_i$, are randomly chosen, and SLFNs can approximate these training samples with zero error.

However in most cases, the number of hidden nodes is much less than the number of distinct training samples, i.e. $L \ll N$, so $\mathbf{H}$ is a non-square matrix and there may not exist $\mathbf{a}_i$, b_i and β_i ($i = 1, \dots, L$), such that $\mathbf{H}\beta = \mathbf{T}$. Hence, Theorem 2.2 is no longer valid.

Theorem 2.3 [13] *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 $\{\mathbf{x}_i | \mathbf{x}_i \in \mathbb{R}^n, i = 1, \dots, N\}$, and $\{(\mathbf{a}_i, b_i)\}_{i=1}^N$ that is randomly generated from any intervals of $\mathbb{R}^n$ and $\mathbb{R}$ respectively according to any continuous probability distribution, $\|\mathbf{H}_{N \times L}\beta_{L \times N} - \mathbf{T}_{N \times m}\| < \varepsilon$ with probability one.*

When the number of hidden nodes is much less than the number of distinct training samples i.e. $L \ll N$, the equation (2.10) is replaced by

$$\mathbf{T} = \mathbf{H}\beta + \mathbf{E} \quad (2.13)$$

where $\mathbf{E} = [\mathbf{e}_1 \cdots \mathbf{e}_N]_{N \times m}^T$ is the system residual. Define the cost function as

$$\mathbf{J} = \sum_{j=1}^N \left(\sum_{i=1}^L \beta_i G(\mathbf{a}_i, b_i, \mathbf{x}_j) - \mathbf{t}_j \right)^2 \quad (2.14)$$

which can be reformulated as

$$\mathbf{J} = (\mathbf{H}\beta - \mathbf{T})^T (\mathbf{H}\beta - \mathbf{T}) \quad (2.15)$$

Then the problem becomes one of minimizing the cost function, which is equivalent to finding a least-squares solution $\hat{\beta}$ such that

$$\|\mathbf{H}\hat{\beta} - \mathbf{T}\|_2 = \min_{\beta} \|\mathbf{H}\beta - \mathbf{T}\|_2 \quad (2.16)$$

where $\|\cdot\|_2$ represents the 2-norm. The detailed proofs for Theorem 2.2 and Theorem 2.3 are presented in [13], and discussions on threshold networks are given in [55].

2.1.4 Minimum norm least-squares solution of SLFNs

As mentioned previously, contrary to the most common understanding that all the parameters of SLFNs need to be adjusted, the input weights and hidden layer biases of SLFNs are in fact not necessarily tuned and the hidden layer output matrix $\mathbf{H}$ can actually remain unchanged once arbitrary values have been assigned to these parameters in the beginning of learning.

With fixed input parameters $\mathbf{a}_i, b_i$, to train an SLFN, it is simply equivalent to finding a least-squares solution $\hat{\beta}$ such that:

$$\|\mathbf{H}(\mathbf{a}_1, \dots, \mathbf{a}_L, b_1, \dots, b_L)\hat{\beta} - \mathbf{T}\| = \min_{\beta} \|\mathbf{H}(\mathbf{a}_1, \dots, \mathbf{a}_L, b_1, \dots, b_L)\beta - \mathbf{T}\| \quad (2.17)$$

According to Theorem 2.1, the smallest norm least-squares solution of the above linear system is

$$\hat{\beta} = \mathbf{H}^{\dagger} \mathbf{T} \quad (2.18)$$

where $\mathbf{H}^{\dagger}$ is the Moore-Penrose generalized inverse of matrix $\mathbf{H}$ [54, 53].

Remark 2.2 *As discussed in Section 2.1.2, we have the following important properties:*

(1) *Minimum training error*

The special solution $\hat{\beta} = \mathbf{H}^{\dagger} \mathbf{T}$ is one of the least-squares solutions of a general linear system $\mathbf{H}\beta = \mathbf{T}$, meaning that the smallest training error can be reached by this special solution:

$$\|\mathbf{H}\hat{\beta} - \mathbf{T}\| = \|\mathbf{H}\mathbf{H}^{\dagger} \mathbf{T} - \mathbf{T}\| = \min_{\beta} \|\mathbf{H}\beta - \mathbf{T}\| \quad (2.19)$$

Although almost all learning algorithms aim to reach the minimum training error, however, most of them cannot reach it because of local minimum or infinite

training iteration is usually not allowed in applications.

(2) *Smallest norm of weights*

Further, the special solution $\hat{\beta} = \mathbf{H}^\dagger \mathbf{T}$ has the smallest norm among all the least-squares solutions of $\mathbf{H}\beta = \mathbf{T}$:

$$\begin{aligned} \|\hat{\beta}\| &= \|\mathbf{H}^\dagger \mathbf{T}\| \leq \|\beta\| \\ \forall \beta &\in \{\beta : \|\mathbf{H}\beta - \mathbf{T}\| \leq \|\mathbf{H}\mathbf{z} - \mathbf{T}\|, \quad \forall \mathbf{z} \in \mathbb{R}^{L \times N}\} \end{aligned} \quad (2.20)$$

(3) *The minimum norm least-squares solution of $\mathbf{H}\beta = \mathbf{T}$ is unique, which is $\hat{\beta} = \mathbf{H}^\dagger \mathbf{T}$.*

Bartlett's theory [56] on the generalization performance of feedforward neural networks states that for feedforward neural networks reaching smaller training error, the smaller the norm of weights, the better is the generalization performance of the network. Therefore, following the method proposed above, the learning algorithm tends to have good generalization performance for feedforward neural networks.

2.1.5 ELM algorithm for SLFNs

The ELM algorithm¹ which only consists of three steps can be summarized as follows:

ELM Algorithm: Given a training set $\mathfrak{X} = \{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbb{R}^n, \mathbf{t}_i \in \mathbb{R}^m, i = 1, \dots, N\}$, activation function $g(x)$, and hidden node number L ,

- (1) Assign random hidden nodes by randomly generating parameters $(\mathbf{a}_i, b_i)$, $i = 1, \dots, L$.

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

- (2) Calculate the hidden layer output matrix $\mathbf{H}$.
- (3) Calculate the output weight β : $\beta = \mathbf{H}^\dagger \mathbf{T}$

where $\mathbf{H}^\dagger$ is the Moore-Penrose generalized inverse [53] of the hidden layer output matrix $\mathbf{H}$. The singular value decomposition (SVD) method is used to calculate the Moore-Penrose generalized inverse $\mathbf{H}^\dagger$ in ELM and most of the implementations of ELM.

In addition, if $\mathbf{H}$ is nonsingular, then the solution can also be written in the form [57]:

$$\hat{\beta} = (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T \mathbf{T} \quad (2.21)$$

where $\mathbf{H}^\dagger = (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T$, which is called the left pseudoinverse of $\mathbf{H}$ due to $\mathbf{H}^\dagger \mathbf{H} = \mathbf{I}$. Hence, the ordinary least-squares (LS) estimate of the linear system response is presented as follows [57]

$$\mathbf{H} \hat{\beta} = \mathbf{H} (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T \mathbf{T} \quad (2.22)$$

It is worth noting that the preliminary 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 prediction process. This parameter is normally obtained by a trial and error method, which may be tedious for some applications.

2.2 Variants of extreme learning machine

Some essential improvements of the preliminary ELM algorithm are presented in this section. In addition, online sequential ELM (OS-ELM) and error minimized ELM (EM-ELM) are reviewed in detail. They provide the background knowledge for our proposed algorithms, which will be presented in subsequent chapters.

Fully complex ELM: The real domain ELM algorithm can be extended to the complex domain and is called fully complex extreme learning machine (**C-ELM**) [17]. The activation functions used in C-ELM are fully complex and previously introduced by Kim and Adali [58]. The simulation results show that C-ELM is fast with lower symbol error rate (SER), which is superior to other algorithms in the complex domain.

Evolutionary ELM: Although ELM has a lot of advantages, it may require a large number of hidden neurons due to the single hidden layer structure. Zhu et al. [59] proposed a hybrid-learning algorithm (called evolutionary extreme learning machine (**E-ELM**)) using differential evolutionary (DE) algorithm to solve the problem. The DE has been proved by Storn [60] to be an efficient tool to locate global optimum over other evolutionary algorithms. E-ELM uses a modified DE algorithm to search for optimal input weights and hidden biases. As a result, E-ELM is compact and could achieve better generalization performance than BP and the original ELM.

Sequential or incremental ELM: Liang et al. [61] proposed the online sequential extreme learning machine (**OS-ELM**), which could learn data one-by-one or chunk-by-chunk (a block of data) with fixed or varying chunk size. Compared to the original ELM, which uses batch learning involving many iterations through the training data, OS-ELM is faster than other sequential algorithms and also achieves good generalization performance. The special feature for OS-ELM, in addition to the original ELM, is that only the new arrival data is seen and learnt at any time rather than the entire past data, which accelerates the training speed. And a training observation will be discarded immediately after the learning process is finished for the observation and thus save memory space.

The online sequential fuzzy extreme learning machine (**OS-Fuzzy-ELM**) developed by Rong et al. [15] extended the OS-ELM to fuzzy inference systems (FIS). A neuro-fuzzy system is developed by combining the advantages of neural network (learning) and fuzzy inference system (approximate reasoning). In

[15], it has been shown that the generalized SLFNs and a FIS with any membership function is functionally equivalent. Therefore, the FIS with Gaussian membership functions can be seen as a special case of SLFNs. Applying the functional equivalence with OS-ELM, OS-Fuzzy-ELM can handle any bounded non-constant piecewise continuous membership function.

ELM ensemble: Chen [62] proposed an ELMs ensemble by applying product index based excluding ensemble (PIEx) approach on regression applications. The experimental results showed that the ensemble effectively improve the generalization performance of ELM network and the PIEx method outperformed bagging and simple averaging approaches.

An integration of ELMs is proposed by Sun et al. [37] 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 indicates that this extension of ELM has a smaller predicting error when the fluctuation of output of individual ELM network is large.

Heeswijk et al. [63] 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 tested. The empirical studies showed that the adaptive ensemble model achieved an acceptable testing error with good adaptivity.

Constructive or growing ELM: In [12], it is proved in theory that SLFNs with randomly generated hidden nodes are actually universal approximators. The proof itself presents a practical incremental constructive method. The resulting learning algorithm is called incremental extreme learning machine (**I-ELM**), which randomly adds node to the hidden layer one by one and freezes the output weights of the existing hidden nodes when a new hidden node is added. And the simulation results showed that the proposed method was efficient and simple with better generalization performance as compared to many popular learning algorithms.

On the basis of I-ELM [12] and Barron's convex optimization learning method [64], the convex I-ELM (**CI-ELM**) is proposed [10]. CI-ELM differs from I-ELM in that it recalculates the output weights of the existing hidden nodes after new hidden nodes are added. In fact, CI-ELM can achieve faster convergence rates and more compact network structure as compared to I-ELM. It is simple and efficient.

Inspired by I-ELM [12], an enhanced random search based incremental algorithm is introduced by Huang et al. [11]. The resultant learning algorithm is called enhanced I-ELM (**EI-ELM**). In each learning step of EI-ELM, several hidden nodes are randomly generated and the hidden nodes that yield the largest decrease in residual error will be selected and added to the existing network. The output weight of the network would be calculated in the same way as in I-ELM. It has been verified by simulation results that EI-ELM can obtain faster convergence rate and much more compact network structure as compared to I-ELM. EI-ELM could also be applied to CI-ELM.

Feng et al. [28] have proposed a constructive method called the error minimized extreme learning machine (**EM-ELM**), which can add random hidden nodes to SLFN one-by-one or group-by-group. Unlike I-ELM, EM-ELM updates all the output weights recursively when new hidden node (or nodes) is added into the network, which increases the convergence rate. The convergence of EM-ELM is proved in [28] as well. The key idea of this development is that one does not have to know beforehand the exact best number of hidden nodes required for ELM to perform well and there is no need to recompute the whole pseudo-inverse matrix.

Yin et al. [65] introduced an enhanced ELM (**EN-ELM**) that selects hidden nodes from the pool of random hidden nodes by applying the Modified Gram-Schmidt (MGS) method and the selection is terminated by the information criterion of Akaike's final prediction error (FPE).

Destructive or pruning ELM: Rong et al. [29] presented a pruned ELM (**P-ELM**) as a systematic and automated method for ELM classifier network design. It starts

with a large network and then eliminates the hidden nodes that have low relevance to the class labels by using statistical criteria, namely, the Chi-squared (χ^2) and information gain (IG) measures. P-ELM mainly focuses on pattern classification applications.

Another pruning algorithm called optimally-pruned ELM (**OP-ELM**) was proposed by Miche et al. [30]. The OP-ELM methodology has three steps: (1) Build the SLFN using the original ELM algorithm; (2) Rank the hidden nodes by applying multi-response sparse regression algorithm (MRSR) [66]; and (3) Select the hidden nodes through leave-one-out (LOO) validation. OP-ELM is applicable for both regression and classification applications.

Wavelet neural network with ELM: Cao et al. [67] introduced a network structure of wavelet neural network (WNN) with ELM, called composite function wavelet neural network with ELM (in short, **CFWNN-ELM**). CFWNN-ELM applies the composite functions at the hidden nodes that allows wavelet functions and a type of bounded non-constant piecewise continuous activation functions in series to manipulate the input information, and then the network is trained with preliminary ELM algorithm. CFWNN-ELM is better than SVMs in function approximation, real world data sets regression and chaotic signal prediction that have been investigated in empirical studies.

ELM for classification: Recently, Huang et al. [68] proposed an extension of ELM to a specific type of generalized SLFNs, support vector network, which further studies ELM for classification in the aspect of the standard optimization method. The paper presents many important results: (1) The consistence 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) ELM for classification is equivalent to SVM from the standard optimization point of view. In addition, ELM has milder optimization constraints due to its special separability feature. (3) It is proved that ELM for classification can achieve better generalization performance than traditional SVM, and it is less sensitive to user specified

parameters.

There are also improvements of ELM combining more than one features mentioned above. Tang et al. [69] proposed the ternary reversible extreme learning machine (**TRELM**), an sequential tri-training method for semi-supervised classification, that employed three reversible extreme learning machine (RELM) as its base learners. RELM is a sequential learning method based on OS-ELM, which outperforms most of sequential learning methods by removing the influence of previous mistaken samples without data retraining. The paper showed that TRELM significantly improves the learning speed of tri-training method with acceptable classification accuracy.

2.2.1 Online sequential extreme learning machine (OS-ELM)

This subsection briefly reviews the essentials of OS-ELM proposed by Liang et al. [61], upon which we will further develop our algorithms.

Assume we are given the type of network node (additive or RBF), the activation function $g(x)$, and the number of hidden nodes $\tilde{N}$. And the data $\mathfrak{X} = \{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbb{R}^n, \mathbf{t}_i \in \mathbb{R}^m, i = 1, \dots\}$ is presented to the network sequentially.

There are two phases in OS-ELM algorithm, an initialization phase and a sequential phase. In the initialization phase, it is required by the algorithm that $rank(\mathbf{H}_0) = L$. (where $\mathbf{H}_0$ denotes the hidden output matrix for initialization phase), which means the number of training data required in the initialization phase is equal to or greater than the number of hidden nodes L . If the first L training data is not distinct, more training data may be needed. The most significant improvement of OS-ELM is presented in the sequential phase. As a property of sequential learning system, the data used to train the OS-ELM network would be discarded once it is used and never retrieved.

Initialization Phase

A small chunk of training data is used to initialize the learning, $\mathfrak{X}_0 = \{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=1}^{N_0}$ from the given training set $\mathfrak{X} = \{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbb{R}^n, \mathbf{t}_i \in \mathbb{R}^m, i = 1, \dots\}$, $N_0 \geq L$

- (1) Randomly assign the input parameters: for additive hidden nodes, parameters are input weights $\mathbf{a}_i$ and bias b_i ; for RBF hidden nodes, parameters are center $\mathbf{a}_i$ and impact factor b_i ; $i = 1, \dots, L$.

- (2) Calculation of the initial hidden layer output matrix $\mathbf{H}_0$

$$\mathbf{H}_0 = \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_1) & \cdots & G(\mathbf{a}_L, b_L, \mathbf{x}_1) \\ \vdots & \cdots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_{N_0}) & \cdots & G(\mathbf{a}_L, b_L, \mathbf{x}_{N_0}) \end{bmatrix}_{N_0 \times L} \quad (2.23)$$

- (3) Estimate the initial output weight $\beta^{(0)}$

We have

$$\mathbf{T}_0 = \begin{bmatrix} \mathbf{t}_1^T \\ \vdots \\ \mathbf{t}_{N_0}^T \end{bmatrix}_{N_0 \times m} \quad (2.24)$$

Then, the problem is to minimize $\|\mathbf{H}_0\beta - \mathbf{T}_0\|$, and from [61], we know that $\mathbf{H}^\dagger = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T$. By Theorem 2.3, the solution to minimize $\|\mathbf{H}_0\beta - \mathbf{T}_0\|$ is given by $\beta^{(0)} = \mathbf{P}_0\mathbf{H}_0^T\mathbf{T}_0$, where $\mathbf{P}_0 = (\mathbf{H}_0^T\mathbf{H}_0)^{-1}$.

- (4) Set $k = 0$. (k : a parameter that indicates the number of chunks of data that is presented to the network.)

Sequential Learning Phase

Given the 1st chunk of new observations, i.e. $k = 1$, $\mathfrak{X}_1 = \{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=N_0+1}^{N_0+N_1}$, which is also from the given training set stated above. Let N_1 denote the number of observations in the 1st chunk.

- (1) Compute the partial hidden layer output matrix $\mathbf{H}_1$

$$\mathbf{H}_1 = \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_{N_0+1}) & \cdots & G(\mathbf{a}_L, b_L, \mathbf{x}_{N_0+1}) \\ \vdots & \cdots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_{N_0+N_1}) & \cdots & G(\mathbf{a}_L, b_L, \mathbf{x}_{N_0+N_1}) \end{bmatrix}_{N_1 \times L} \quad (2.25)$$

- (2) Calculate the output weight $\beta^{(1)}$

We have

$$\mathbf{T}_1 = \begin{bmatrix} \mathbf{t}_{N_0+1}^T \\ \vdots \\ \mathbf{t}_{N_0+N_1}^T \end{bmatrix}_{N_1 \times m} \quad (2.26)$$

And now the problem becomes minimizing $\left\| \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix} \beta - \begin{bmatrix} \mathbf{T}_0 \\ \mathbf{T}_1 \end{bmatrix} \right\|$. From the initialization phase (3), the output weight $\beta^{(1)}$ can be obtained by the equation

$$\beta^{(1)} = \left(\begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix} \right)^{-1} \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{T}_0 \\ \mathbf{T}_1 \end{bmatrix} \quad (2.27)$$

In the sequential learning, we need to express $\beta^{(1)}$ as a function of $\beta^{(0)}$, $\mathbf{H}_1$, and $\mathbf{T}_1$ instead of a function of parameters used for data set $\mathfrak{X}_0$.

Let

$$\mathbf{K}_0 = \mathbf{H}_0^T \mathbf{H}_0 = \mathbf{P}_0^{-1} \quad (2.28)$$

Then

$$\mathbf{K}_1 = \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix} = \begin{bmatrix} \mathbf{H}_0^T & \mathbf{H}_1^T \end{bmatrix} \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix} = \mathbf{K}_0 + \mathbf{H}_1^T \mathbf{H}_1 \quad (2.29)$$

and

$$\begin{aligned} \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{T}_0 \\ \mathbf{T}_1 \end{bmatrix} &= \mathbf{H}_0^T \mathbf{T}_0 + \mathbf{H}_1^T \mathbf{T}_1 = \mathbf{K}_0 \mathbf{K}_0^{-1} \mathbf{H}_0^T \mathbf{T}_0 + \mathbf{H}_1^T \mathbf{T}_1 \\ &= \mathbf{K}_0 \boldsymbol{\beta}^{(0)} + \mathbf{H}_1^T \mathbf{T}_1 = (\mathbf{K}_1 - \mathbf{H}_1^T \mathbf{H}_1) \boldsymbol{\beta}^{(0)} + \mathbf{H}_1^T \mathbf{T}_1 \\ &= \mathbf{K}_1 \boldsymbol{\beta}^{(0)} - \mathbf{H}_1^T \mathbf{H}_1 \boldsymbol{\beta}^{(0)} + \mathbf{H}_1^T \mathbf{T}_1 \end{aligned} \quad (2.30)$$

From (2.29) and (2.30), $\boldsymbol{\beta}^{(1)}$ is given by

$$\begin{aligned} \boldsymbol{\beta}^{(1)} &= \mathbf{K}_1^{-1} \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{T}_0 \\ \mathbf{T}_1 \end{bmatrix} = \mathbf{K}_1^{-1} (\mathbf{K}_1 \boldsymbol{\beta}^{(0)} - \mathbf{H}_1^T \mathbf{H}_1 \boldsymbol{\beta}^{(0)} + \mathbf{H}_1^T \mathbf{T}_1) \\ &= \boldsymbol{\beta}^{(0)} + \mathbf{K}_1^{-1} \mathbf{H}_1^T (\mathbf{T}_1 - \mathbf{H}_1 \boldsymbol{\beta}^{(0)}) \end{aligned} \quad (2.31)$$

The above arguments can be generalized, and we have a recursive algorithm for updating the least-squares solution $\boldsymbol{\beta}$, which is similar to the recursive least-squares algorithm [70].

Given the $(k+1)$ th chunk of new observations, $\mathfrak{X}_{k+1} = \{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=(\sum_{j=1}^k N_j)+1}^{\sum_{j=0}^{k+1} N_j}$, and let N_{k+1} denote the number of observations in the $(k+1)$ th chunk.

(1) Compute the partial hidden layer output matrix $\mathbf{H}_{k+1}$

$$\mathbf{H}_{k+1} = \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_{(\sum_{j=0}^k N_j)+1}) & \cdots & G(\mathbf{a}_L, b_L, \mathbf{x}_{(\sum_{j=0}^k N_j)+1}) \\ \vdots & \cdots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_{\sum_{j=0}^{k+1} N_j}) & \cdots & G(\mathbf{a}_L, b_L, \mathbf{x}_{\sum_{j=0}^{k+1} N_j}) \end{bmatrix}_{N_{k+1} \times L} \quad (2.32)$$

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

We have

$$\mathbf{T}_{k+1} = \begin{bmatrix} \mathbf{t}_{(\sum_{j=0}^k N_j)+1}^T \\ \vdots \\ \mathbf{t}_{\sum_{j=0}^{k+1} N_j}^{k+1} \end{bmatrix}_{N_{k+1} \times m} \quad (2.33)$$

Then

$$\mathbf{K}_{k+1} = \mathbf{K}_k + \mathbf{H}_{k+1}^T \mathbf{H}_{k+1} \quad (2.34)$$

$$\beta^{(k+1)} = \beta^{(k)} + \mathbf{K}_{k+1}^{-1} \mathbf{H}_{k+1}^T (\mathbf{T}_{k+1} - \mathbf{H}_{k+1} \beta^{(k)}) \quad (2.35)$$

From equation (2.35), we find that $\mathbf{K}_{k+1}^{-1}$ is used to compute $\beta^{(k+1)}$. In order to avoid calculating the inverse in the recursive process, the Woodbury formula [71] is applied to modify the equations.

$$\mathbf{K}_{k+1}^{-1} = (\mathbf{K}_k + \mathbf{H}_{k+1}^T \mathbf{H}_{k+1})^{-1} = \mathbf{K}_k^{-1} - \mathbf{K}_k^{-1} \mathbf{H}_{k+1}^T (\mathbf{I} + \mathbf{H}_{k+1} \mathbf{K}_k^{-1} \mathbf{H}_{k+1}^T)^{-1} \mathbf{H}_{k+1} \mathbf{K}_k^{-1} \quad (2.36)$$

Let $\mathbf{P}_{k+1} = \mathbf{K}_{k+1}^{-1}$, and we modify the equations (2.34) and (2.35) using (2.36):

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \mathbf{P}_k \mathbf{H}_{k+1}^T (\mathbf{I} + \mathbf{H}_{k+1} \mathbf{P}_k \mathbf{H}_{k+1}^T)^{-1} \mathbf{H}_{k+1} \mathbf{P}_k \quad (2.37)$$

$$\beta^{(k+1)} = \beta^{(k)} + \mathbf{P}_{k+1} \mathbf{H}_{k+1}^T (\mathbf{T}_{k+1} - \mathbf{H}_{k+1} \beta^{(k)}) \quad (2.38)$$

(3) Set $k = k + 1$.

Remark 2.3 Five remarks of OS-ELM in addition to the mathematical expressions are listed as follows.

(1) From equations (2.31), (2.35) and (2.38), as mentioned before, the sequential im-

plementation of least-squares solution is similar to recursive least-squares algorithm [70]. Therefore, all the convergence rules for recursive least-squares (RLS) can be applied for OS-ELM.

- (2) The detail derivation of OS-ELM can be found in [61]. From the derivation, it is clear that OS-ELM and ELM can achieve the same learning performance if $\text{rank}(\mathbf{H}_0) = L$. To satisfy this criteria, the number of initialization data N_0 should not be less than the number of hidden nodes L , which is already mentioned at the beginning of Section 2.2.1.
- (3) When the data presented to the network is in chunk-by-chunk mode, the chunk size could be varied. And if the training data is received one-by-one instead of chunk-by-chunk, equation (2.38) can be simplified as follows, which make use of the Sherman-Morrison formula [71].

$$\mathbf{P}_{k+1} = \mathbf{P} - \frac{\mathbf{P}_k \mathbf{h}_{k+1} \mathbf{h}_{k+1}^T \mathbf{P}_k}{1 + \mathbf{h}_{k+1}^T \mathbf{P}_k \mathbf{h}_{k+1}} \quad (2.39)$$

$$\beta^{(k+1)} = \beta^{(k)} + \mathbf{P}_{k+1} \mathbf{h}_{k+1} (\mathbf{t}_{k+1}^T - \mathbf{h}_{k+1}^T \beta^{(k)}) \quad (2.40)$$

where $\mathbf{h}_{k+1} = [G(\mathbf{a}_1, b_1, \mathbf{x}_{(k+1)}) \quad \cdots \quad G(\mathbf{a}_L, b_L, \mathbf{x}_{(k+1)})]$ [61].

- (4) $\beta^{(k)}$ and $\mathbf{P}_k$ are calculated using SVD to make OS-ELM more robust.
- (5) In case of $N_0 = N$, OS-ELM is the same as batch ELM. Hence, ELM can be seen as a special case of OS-ELM when all the data is presented in one iteration.

2.2.2 Error minimized extreme learning machine (EM-ELM)

In this subsection, a brief review about EM-ELM [28] is presented. EM-ELM is an error minimization based method where the number of hidden nodes can be grown one-by-one or group-by-group until it is optimal. The approach can significantly reduce the computational complexity and its convergence was proved as well.

Assume we have a set of training data $\{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=1}^N$, the maximum number of hidden nodes L_{max} , and the expected learning accuracy ε . There are two phases in the EM-ELM algorithm.

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:

$$\mathbf{H}_1 = \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_1) & \cdots & G(\mathbf{a}_{L_0}, b_{L_0}, \mathbf{x}_1) \\ \vdots & \cdots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_N) & \cdots & G(\mathbf{a}_{L_0}, b_{L_0}, \mathbf{x}_N) \end{bmatrix}_{N \times L_1} \quad (2.41)$$

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

(4) Let $k = 0$.

If the network output error $E(\mathbf{H}_1)$ is less than the expected value $\varepsilon > 0$, the learning is completed and the number of hidden nodes is optimal. Otherwise, a group of random hidden nodes, $\delta L_0 = L_1 - L_0$, can be added into the existing SLFN to yield the new hidden layer output matrix $\mathbf{H}_2 = [\mathbf{H}_1, \delta \mathbf{H}_1]$.

$$\begin{aligned} \mathbf{H}_2^\dagger &= (\mathbf{H}_2^T \mathbf{H}_2)^{-1} \mathbf{H}_2^T \\ &= \left(\begin{bmatrix} \mathbf{H}_1^T \\ \delta \mathbf{H}_1^T \end{bmatrix} \begin{bmatrix} \mathbf{H}_1 & \delta \mathbf{H}_1 \end{bmatrix} \right)^{-1} \begin{bmatrix} \mathbf{H}_1^T \\ \delta \mathbf{H}_1^T \end{bmatrix} \end{aligned} \quad (2.42)$$

Assume

$$\begin{aligned}
 \mathbf{A} &= \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix} = \left(\begin{bmatrix} \mathbf{H}_1^T \\ \delta \mathbf{H}_1^T \end{bmatrix} \begin{bmatrix} \mathbf{H}_1 & \delta \mathbf{H}_1 \end{bmatrix} \right)^{-1} \\
 &= \begin{bmatrix} \mathbf{H}_1^T \mathbf{H}_1 & \mathbf{H}_1^T \delta \mathbf{H}_1 \\ \delta \mathbf{H}_1^T \mathbf{H}_1 & \delta \mathbf{H}_1^T \delta \mathbf{H}_1 \end{bmatrix}^{-1}
 \end{aligned} \tag{2.43}$$

then,

$$\mathbf{H}_2^\dagger = \begin{bmatrix} \mathbf{U} \\ \mathbf{D} \end{bmatrix} = \begin{bmatrix} \mathbf{A}_{11} \mathbf{H}_1^T + \mathbf{A}_{12} \delta \mathbf{H}_1^T \\ \mathbf{A}_{21} \mathbf{H}_1^T + \mathbf{A}_{22} \delta \mathbf{H}_1^T \end{bmatrix} \tag{2.44}$$

Let $\mathbf{R} = \delta \mathbf{H}_1^T \delta \mathbf{H}_1 - \delta \mathbf{H}_1^T \mathbf{H}_1 (\mathbf{H}_1^T \mathbf{H}_1)^{-1} \mathbf{H}_1^T \delta \mathbf{H}_1$, which is the Schur complement of $\mathbf{H}_1^T \mathbf{H}_1$.

The matrix $\mathbf{A}$ is invertible if and only if $\mathbf{R}$ is invertible [72]. Based on the inversion of 2×2 block matrices, we have

$$\begin{aligned}
 \mathbf{A}_{11} &= (\mathbf{H}_1^T \mathbf{H}_1)^{-1} + (\mathbf{H}_1^T \mathbf{H}_1)^{-1} \mathbf{H}_1^T \delta \mathbf{H}_1 \mathbf{R}^{-1} \delta \mathbf{H}_1^T \mathbf{H}_1 (\mathbf{H}_1^T \mathbf{H}_1)^{-1} \\
 \mathbf{A}_{12} &= -(\mathbf{H}_1^T \mathbf{H}_1)^{-1} \mathbf{H}_1^T \delta \mathbf{H}_1 \mathbf{R}^{-1} \\
 \mathbf{A}_{21} &= -\mathbf{R}^{-1} \delta \mathbf{H}_1^T \mathbf{H}_1 (\mathbf{H}_1^T \mathbf{H}_1)^{-1} \\
 \mathbf{A}_{22} &= \mathbf{R}^{-1}
 \end{aligned} \tag{2.45}$$

We know $\mathbf{H}_1^\dagger = (\mathbf{H}_1^T \mathbf{H}_1)^{-1} \mathbf{H}_1^T$, then $\mathbf{R} = \delta \mathbf{H}_1^T \delta \mathbf{H}_1 - \delta \mathbf{H}_1^T \mathbf{H}_1 \mathbf{H}_1^\dagger \delta \mathbf{H}_1$. Hence we get

$$\begin{aligned}\mathbf{D} &= ((\mathbf{I} - \mathbf{H}_1 \mathbf{H}_1^\dagger) \delta \mathbf{H}_1)^\dagger \\ \mathbf{U} &= \mathbf{H}_1^\dagger - \mathbf{H}_1^\dagger \delta \mathbf{H}_1 \mathbf{D}\end{aligned}\tag{2.46}$$

Therefore, the formula (2.46) shows us the mechanism on how the hidden nodes will be added to the SLFNs. The procedure can be summarized in the following phase.

Recursively Growing Phase:

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

- (1) Randomly add δL_k hidden nodes to the existing SLFNs. 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

$$\delta \mathbf{H}_k = \begin{bmatrix} G(\mathbf{a}_{L_k+1}, b_{L_k+1}, \mathbf{x}_1) & \cdots & G(\mathbf{a}_{L_k+1}, b_{L_k+1}, \mathbf{x}_1) \\ \vdots & \cdots & \vdots \\ G(\mathbf{a}_{L_k+1}, b_{L_k+1}, \mathbf{x}_N) & \cdots & G(\mathbf{a}_{L_k+1}, b_{L_k+1}, \mathbf{x}_N) \end{bmatrix}_{N \times \delta L_k}\tag{2.47}$$

- (2) Update the output weight β

$$\begin{aligned}\mathbf{D}_k &= ((\mathbf{I} - \mathbf{H}_k \mathbf{H}_k^\dagger) \delta \mathbf{H}_k)^\dagger \\ \mathbf{U}_k &= \mathbf{H}_k^\dagger - \mathbf{H}_k^\dagger \delta \mathbf{H}_k \mathbf{D}_k \\ \beta^{(k+1)} &= \mathbf{H}_{k+1}^\dagger \mathbf{T} = \begin{bmatrix} \mathbf{U}_k \\ \mathbf{D}_k \end{bmatrix} \mathbf{T}\end{aligned}\tag{2.48}$$

(3) $k = k + 1$.

The key idea of this development is that one does not have to know beforehand the exact best number of hidden nodes required for ELM to perform well and there is no need to recompute the whole pseudo-inverse matrix.

2.3 Summary

In this chapter, ELM algorithm proposed by Huang et al. and its essential variants are introduced and reviewed. Generally speaking, an ELM network can be regarded as a special single-hidden layer network that is made linear in output parameters by randomly generating and fixing the hidden node parameters. Thus, the hidden layer of the ELM network maps the input space onto a new space (i.e. ELM feature space) by performing a fixed nonlinear transformation with no adjustable parameters. The output layer behaves like a linear combiner on the new space and the output weights are the only adjustable parameters in the system, which can be solved analytically using the linear least-squares (LS) method.

Theoretically, ELM can produce an exactly mapping when the number of hidden nodes is equal to the number of training samples. However, an approximate mapping is preferred in practice considering the computational complexity and generalization performance of the network in real world applications. Therefore, the number of hidden nodes becomes a free parameter that is selected a priori and in general is less than the number of training samples. This parameter is normally obtained by a trial and error method, which may probably lead to a large network that includes some insignificant units. In this thesis, we would like to address the problem of architecture design of ELM and some solutions will be presented in the subsequent chapters.

Chapter 3

Constructive Algorithm and Model Selection Criteria

In this chapter, a brief review of the existing constructive algorithms is given. Some statistical model selection criteria are introduced to terminate the search in constructive algorithms.

3.1 Motivation

An important issue of when a network is trained by given examples is how well it can generalize to patterns outside the training set, i.e. the generalization performance of the network. It is usually impossible to collect examples for every possible input especially for continuous domains and/or large discrete problems. Training the network to memorize all the input examples may fail when similar but slightly different inputs are presented. The network is supposed to generalize from the training examples to the underlying functions and give reasonable answers to novel inputs. To obtain a good generalization, a rule of thumb is to use an appropriate parsimonious network that will fit the data. For a network with too few hidden nodes, it may lead to high training error and

poor generalization performance due to underfitting and large statistical bias. While for a network with too many hidden nodes, it may lead to low training error but poor generalization performance due to overfitting and large variance [26]. Unfortunately, it is usually difficult to tell what size is the optimal for a new problem faced. The global optimum structure of the network may be obtained using exhaustive search over all possible architectures. However, it is impractical for large networks [73]. A common approach to search for a suitable network structure is by trial and error, in which many networks in various sizes are trained and the one with the highest generalization performance is picked. There are two disadvantages when applying the trial and error method: 1) Many networks may have to be trained before an acceptable one is found. Hence, the approach is rather inefficient although straightforward. 2) The approach cannot always guarantee an optimal solution. Hence, some researchers have advocated that the network structure should not be determined by a trial and error method but should instead be computed by the learning algorithm [74].

Neural network ensembles can be used to improve the networks' generalization ability [27]. In some cases, when comparing all the candidate single networks, there might not be a clear winner in the sense that several candidates perform almost as well as the "winner". In such case, a network ensemble could be a better choice. However, the ensemble method cannot alter the topology of the individual network in the ensemble.

Additionally, regularization is sometimes used to prevent overfitting by introducing additional information, which is usually in the form of a penalty for complexity. However, it cannot alter the network structure too. The use of regularization may allow the network to be larger than necessary, but an overly large network is not recommended in practice because the optimization of the regularized error function with respect to its network weights can be computationally more intensive and difficult. Moreover, there is a constant parameter in regularization method that controls the balance between the normal error term and the penalty term. This parameter is manually set in early attempts. Then, Mackay [75, 76] developed Bayesian regularization methods to automatically determine the constant parameter for the penalty term thus removing the need to set the

parameter manually. However, the penalty terms may create additional local minima, which increase the probability that the network may converge to local minima [77].

The alternative solution for tackling the structure design of the network is to alter the network structure during the training process. There are normally two heuristic approaches to modify the structure of SLFNs: constructive methods (or growing methods) [78, 79] and destructive methods (or pruning methods) [80]. A constructive method starts with a small network and then adds hidden nodes into the hidden layer in order to decrease the learning error, while a destructive method starts with a large network and eliminates the hidden nodes that do not contribute significantly to success.

3.1.1 Pruning algorithms

Generally speaking, a pruning algorithm trains the network that is larger than necessary and then removes the parts that are not needed. It starts with a large initial network, which allows the network to learn quickly with less sensitivity to initial conditions.

The direct approach of pruning algorithm is the so called brute-force method, which sets the weights to zero and investigates the changes in the error. If the error increases by too much, the weights are restored, otherwise they are removed. Sometimes, an even more conservative method is applied that evaluates the change in the error for each weight and pattern and then the weights or patterns with less effects are removed. The exhaustive methods mentioned above are rather slow especially for large networks. Thus, lots of methods that take a less direct approach are proposed.

The greedy approaches to pruning can be categorized into two broad groups: *sensitivity methods* and *penalty-term methods* [80]. The first group of methods estimates the sensitivity of the error functions, and the elements with the least effect may then be removed. While, the other group of methods add penalty terms to the objective function for selecting efficient solutions. Sometimes, the objective function could include the sensitivity term. Therefore, the two groups of approaches may have overlaps. The major difference

between the two groups of methods is that the sensitivity methods modify the network after the training process while the penalty-term methods modify the objective function in order to drive the unnecessary weights to zero during the training process.

Many methods have been introduced based on the calculation of sensitivities. Mozer and Snolensky [81] proposed an approach that estimates the functionality or relevance of individual units and eliminates the hidden units that convey redundant information. The relevance of a unit is determined by the difference of the error when the unit is presented and when it is removed from the network. Karnin [82] estimated the sensitivity of weights after training and pruned the weights with low sensitivity. A node in the network can be removed when all the input weights to a node are deleted, i.e. the output of node is constant and it can be seen as a bias term. Le Cun et al. [83] 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 methods mentioned above attempt to identify the insignificant weights or hidden units by calculating the sensitivity of the error. Besides, there are researches [84, 85, 86] that focused on modifying the objective functions by adding penalty-terms and pruning the weights by driving them to zero during training. Some of the penalty-term methods [87, 88] introduced weight decay into the learning process, and the weights normally do not decay in a constant rate.

In addition to sensitivity methods and penalty methods, Sietsma and Dow [89] described an interactive pruning method in which the hidden nodes are evaluated and removed manually by the designer. There are several heuristics to identify the significance of each node. Kruschke [90] proposed a method where the hidden nodes have to “compete” to survive. Furthermore, Whitley and Bogart [91] introduced the use of genetic algorithm to prune a trained network. Hagiwara [92] presented simple and intuitive pruning methods based on weight values and unit activation values. OP-ELM [30] mentioned in Section 2.2 pruned the hidden nodes using leave-one-out validation method.

There are many advantages for using pruning algorithms. Starting with a large network structure, the network when using a pruning algorithm can converge reasonably quickly with less sensitivity to local minima. In addition, the functional flexibility is better for large network structure as compared to a small network structure in most of the cases, which guarantees a better fit of the underlying function. However, there are some drawbacks for using pruning algorithms as compared to constructive algorithms and they are discussed in the next subsection.

3.1.2 Advantages of constructive algorithm as compared to pruning algorithm

A constructive algorithm aims to search for a good network structure in the opposite direction of a pruning algorithm. It usually starts with a small network (normally a single node) and grows the network by adding hidden nodes until an acceptable solution is obtained [93, 94]. Baum [95] 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 more detailed description of constructive algorithms will be given in the next section.

Constructive approaches are usually superior to destructive approaches in many aspects [78]. First, it is difficult to determine how big of a network is enough to start the pruning process. Second, at least most of the times, the pruning approach has to deal with a network that is larger than necessary, which increases the computational complexity and wastes computing time. Third, many networks that come in different sizes have the capabilities to achieve acceptable solutions. “The principle of parsimony” suggests that if many networks are capable to solve the problem with acceptable accuracy, a smaller network is preferred [5]. A constructive method tends to produce a smaller network structure for a given application as compared to a destructive one. In addition, with small network structures, the amount of training data required for good generalization performance may be reduced.

There are still some obstacles that the constructive algorithms have to overcome. Theoretically speaking, finding the “minimal” architecture of the network is usually NP-hard [96]. Many constructive algorithms apply greedy approaches to network construction, which may be suboptimal. Another common question is when to stop the addition of hidden nodes. A similar question also arose in pruning algorithms and that is when to stop the pruning of the hidden nodes.

3.2 Design of constructive algorithms

Unlike a pruning algorithm, a constructive algorithm varies the network structure by adding hidden nodes to the hidden layer. It starts with a small network (or network with no hidden nodes) and then grows the network by adding the hidden nodes and weights until a satisfactory solution is found [93, 94].

Three major problems need to be considered when designing constructive algorithms [97]:

- (1) Network architecture: How are the new hidden nodes to be connected to the existing network?
- (2) Training strategy: How are the parameters of the network determined? How can the parameters of the (existing and newly added) hidden nodes be updated after the addition of new hidden nodes?
- (3) Stopping condition: When can the addition of new hidden nodes be terminated? In other words, what is the optimal number of hidden nodes in the hidden layer for the particular application?

The first issue addresses the problem of network architecture, which is different from what has been mentioned and what will be presented in this thesis. The network architecture design for ELM is mainly to determine the number of hidden nodes in the

hidden layer and the number of hidden layer is fixed at one according to the SLFN network structure. On this issue, it is related to a broad concept that includes the number of hidden layers in the network and whether the network has feedback mechanism, etc. In fact, no single class of network architectures is optimal for all problems [98], but a comprehensive comparison of different architectures is still lacking. We shall focus on the feedforward neural networks in this thesis and recurrent networks that have at least one feedback loop is out of the scope of this thesis. Generally speaking, for linear and generalized linear models, the feedforward neural networks without a hidden layer are useful in a wide variety of applications [99]. Even for mildly nonlinear applications, it may be better to use a simple linear model than a complicated nonlinear model if the number of data provided is small and the data has high noise. In contrast, two hidden layers are required for full generality when the activation function used in the network is step (or threshold or heaviside) function in multi-layer feedforward neural networks [100]. In multi-layer feedforward neural networks with any continuous nonlinear hidden-layer activation functions, the networks with one hidden layer are universal approximators [7, 9]. In addition, Chester [101] claimed that two hidden layer feedforward networks can often yield an accurate approximation with fewer weights than a single hidden layer network. However, the networks with two hidden layers exacerbate the problem of local minima. Feedforward networks with more than two hidden layers can be useful in certain networks (like cascade correlation networks [102]) and some applications (such as two-spirals problem [103] and ZIP code recognition [104]). Since ELM [13] is designed for SLFNs, the first issue can be easily determined for the constructive algorithm of ELM.

We shall elaborate on the second issue, i.e. training strategy, in Subsection 3.2.1. The training strategy determines the parameters of the whole network and the update of the parameters of the old and newly added hidden units. In this thesis, we will concentrate on the development of constructive algorithms for regression applications. And we will review some typical constructive algorithms for each training strategy to serve as motivations and bases for our proposed results.

The third issue can be tackled using different techniques, such as evidence comparison in Bayesian framework [76], or information criteria for non-Bayesian framework [105, 106, 107, 108, 109], or data resampling methods [110, 111, 112]. The detailed stopping criteria for model selection are discussed in Subsection 3.2.2 where we will focus on the information criteria based on cross validation for non-Bayesian framework.

3.2.1 Constructive training strategies and constructive algorithms

Given the network architecture (meaning specifically the number of hidden layers and the full connection of hidden nodes), the determination of the parameter of the network can be simply achieved by training the whole network. For networks with fixed number of hidden units, the parameters are trained only once. However, for networks using a constructive or pruning algorithm, the training process must be repeated when the number of hidden nodes has been changed (eg. Dynamic Node Creation (DNC) [113]), which dramatically affects the computational efficiency especially for applications that requires a large network structure. There are four ways to alleviate the problem: by matrix decomposition (eg. Orthogonal Least-Squares (OLS) [114] and its variants), by recursively updating the parameters (eg. EM-ELM [28]), by training only the new hidden unit (eg. Projection Pursuit Learning (PPL) [115], I-ELM [12] and its variants – single hidden layer; Cascade-Correlation Learning Architecture (CCLA) [102] – multiple hidden layers) and by training with memorization (eg. Resource-Allocating Network (RAN) [116] and its variants).

Training the whole network

Dynamic Node Creation (DNC) network proposed by Ash [113] is a typical example of training the whole network. DNC sequentially adds the sigmoid hidden node one at a time to the hidden layer of Back Propagation (BP) network until the minimum topology that achieves the desired approximation accuracy is found. The algorithm

simply retrains the whole network after each hidden unit is added. However, this training algorithm is computational expensive when a large network is desired.

The Orthogonal Least-Squares (OLS) algorithm and its variants [114, 117, 118, 119] are efficient stepwise constructive algorithms, which is based on the modified Gram-Schmidt (MGS) orthogonalisation. OLS selects significant model terms one-by-one from a large set of candidates. At each step, the net decrease in the residual error is maximized. The key advantage of the algorithm is that it can explicitly identify the net contribution of the newly added node without solving the whole least-squares problem, which significantly reduces the computational complexity.

EM-ELM [28] mentioned in Section 2.2 is designed based on SLFN. The hidden nodes can be added one-by-one or group-by-group. The whole network is retrained when the new hidden nodes are added, but there is no need to recompute the whole pseudo-inverse matrix. Instead, the pseudo-inverse of the new matrix can be computed based on the pseudo-inverse of the hidden layer output matrix in the previous step.

Training only the new hidden unit

The Projection Pursuit Learning (PPL) suggested by Friedman and Stuetzle [115] is designed using a two-layered network for multivariate data analysis, which interprets high dimensional data through well-chosen lower dimensional projections. The mathematical form of PPL network is quite similar to SLFN. With all the training patterns presented, a PPL network adds hidden nodes one-by-one to the network and updates the parameters layer-by-layer. The input weights are estimated using Gauss-Newton nonlinear least-squares method; while the output weights are calculated by linear least-squares method [120]. PPL uses hidden nodes with more complicated functional forms and trains the newly added hidden node only, which is different from using sigmoid hidden nodes and retraining the whole network after the addition of each hidden node in DNC. In [121], the authors claimed that PPL networks are not universal approximators for any finite order and hence the network sequence produced by PPL cannot converge to the target

function. To overcome the problem, a bias term is added in each linear projection of the predictor variables [121] based on the results in [122], by which the PPL networks obtain the universal approximation and convergence properties.

I-ELM [12] and its variants [10, 11] have also been mentioned in Section 2.2. In these algorithms, all the parameters of the existing hidden nodes are fixed, and the new hidden node is trained to reduce as much residual error as possible.

Fahlman and Lebiere [102] has proposed a constructive architecture called Cascade-Correlation Learning Architecture (CCLA). The CCLA network has multiple hidden layers that allow it to establish more powerful approximation capabilities with simple hidden units. The way of connection of the hidden units in CCLA is quite different from other algorithms. For CCLA, there is no hidden units in the network initially and all the inputs and outputs are directly connected. The existing connections are trained to minimize the error. Then, hidden nodes are added incrementally to the network one-by-one until the desired training accuracy is met. More specifically, when a new hidden unit is installed in the network, it connects to all the inputs as well as hidden units previously added to the network. These new connections are trained to maximize the correlation between the output of newly added hidden node and the residual error at the network output. The input weights are frozen after training. Then, the output of newly added hidden node is connected to all the network output units and the new output weights are determined when the training error is minimized. In fact, adding one hidden node in CCLA is equivalent to adding one hidden layer to the network that leads to a deep cascade structure. The convergence property of the cascade-correlation algorithm is proved in [123] for sigmoid hidden units and uniform input environment measure. Although the algorithm is quite flexible and fast in training, the deep structure may cause a long propagation delay. As more hidden units are added, the structure may not be suitable for Very Large Scale Integration (VLSI) implementation. To solve the problem, Smotroff et al. [124] suggested using “iterative atrophy” to prune connections that are less important. However, the approach is computationally intensive and the depth of the network is not affected. Phatak and Koren [125] introduced a modification

of cascade-correlation algorithm that allows more than one hidden nodes in each layer and the newly added hidden node is connected to the previous hidden layer only. The fan-in of the hidden nodes is controlled in such way that if the number of hidden nodes in one layer exceeds a certain value, the hidden unit is added to a new layer of the network.

Training with memorization

Resource-Allocating Network (RAN) [116] proposed by Platt is designed based on the SLFN and adds hidden nodes to the hidden layer one at a time. Different from all the algorithms mentioned above, RAN and its variants [126, 127, 128, 129, 130] utilize the memorization of training patterns to reduce the computational complexity. The training data are presented pattern-by-pattern in RAN algorithm. If the network performs well on a presented pattern, the network parameters are adjusted using standard least mean square gradient descent method. Otherwise, if the pattern is novel (i.e. far away from existing hidden nodes), a new hidden unit is installed and memorized by the network. The activation function used in RAN and its variants is normally RBF, which responds to only a local region of input space. The advantage of RAN is that the addition of a new RBF hidden node can immediately improve the performance of the network in a local region near the added node. However, noisy data may lead to a large number of hidden nodes in RAN.

3.2.2 Model selection criteria

In this section, some model selection criteria based on cross validation are reviewed. Cross validation is the method for evaluating the generalization capabilities of network model based on “resampling” [111]. Model selection criteria based on cross validation can either be used for predicting the network’s generalization performance or choosing among the various models. In k -fold cross validation, the data is divided into k subsets of equal size without overlapping. The network is trained for k times, leaving one of the

subsets out for training at each time, and use the subset omitted in the training to evaluate the performance of the network. Leave-one-out (LOO) cross validation is applied when k is equal to the number of samples.

Supposing that we have a nonlinear model with a linear-in-the-parameters structure and N data samples $\{\mathbf{x}_i, t_i\}_{i=1}^N$ are given, the general model is given by

$$f_L(\mathbf{x}, \beta) = \sum_{i=1}^L \phi_i(\mathbf{x}_j) \beta_i, \quad j = 1, \dots, N \quad (3.1)$$

where $\phi_i(\mathbf{x})$ is a known nonlinear basis function mapping, such as sigmoid, RBF, etc, β_i is unknown parameter and L is the number of basis functions in the model (i.e. for SLFNs, L refers to the number of hidden nodes). The estimation of the unknown parameter is given by $\hat{\beta} = [\hat{\beta}_1, \hat{\beta}_2, \dots, \hat{\beta}_L]^T$. Let the residual error be $\mathbf{e}$, which follows the distribution $\mathbf{N}(0, \sigma^2)$. Let $\hat{\sigma}^2$ be the estimator of σ^2 . The residual sum of squares over the estimation dataset is given by $RSS(\hat{\beta}) = \sum_{i=1}^N [t_i - \sum_{i=1}^L \phi_i(\mathbf{x}) \hat{\beta}_i]^2$. Various model selection criteria have been introduced and these are briefly discussed below.

- Mallow's C_P criterion:

Mallow's C_P proposed by Mallows [109] is an estimate of the standardized total mean squared error of estimation for $\phi(\mathbf{x})$. Assume that L hidden nodes are presented in the current model. Denoting the standardized total mean squared error of estimation for $\phi(\mathbf{x})$ as Γ_P , then

$$\Gamma_P = \frac{1}{\sigma^2} \sum_{i=1}^N \text{MSE}(t_i) = \frac{E(RSS)}{\sigma^2} + 2L - N \quad (3.2)$$

Replacing $E(RSS)$ and σ^2 by appropriate estimates yields

$$C_P = \frac{RSS(\hat{\beta})}{\hat{\sigma}^2} + 2L - N \quad (3.3)$$

However, some subjective judgement is required for the determination of $\hat{\sigma}^2$ when applying the C_P criterion.

- Final prediction error (FPE) criterion:

After the C_p criterion was proposed, a new criterion based on controlling the mean-square error of the one-step-ahead prediction is introduced. The mean-square error is called the final prediction error (FPE) [105], which is obtained by using the least squares estimates of the coefficients. The FPE criterion is given as:

$$FPE(p) = \frac{RSS(\hat{\beta})}{N} \cdot \left(\frac{N+L}{N-L} \right) \quad (3.4)$$

For the FPE criterion, there is no subjective element. However, the choice of the upper bound of L (i.e. the maximum number of hidden nodes for selection) requires judgement.

- Akaike's information criterion (AIC):

The Akaike's information criterion (AIC) [106] is based on the maximum likelihood estimate (MLE). AIC criterion is given by

$$AIC = N \log \left(\frac{RSS(\hat{\beta})}{N} \right) + 2L \quad (3.5)$$

The definition of minimum AIC estimate (MAICE) provides a mathematical formulation of the principle of parsimony in model building. If two models have the same maximum likelihood, the compactness of networks can be determined using MAICE. In addition, AIC can be implemented without the aid of subjective judgement. Furthermore, Young [131] presented that AIC is equivalent to Mallows's C_p when σ^2 is known.

- Predicted residual error sum of squares (PRESS) statistic:

The predicted residual error sum of squares (PRESS) [132] is a popular variant of the mean squares of LOO errors, which replaces the denominator of the mean squares of LOO errors for all data sample by their average value. Thus, PRESS is

given by:

$$PRESS = \hat{\sigma}^2 \left(1 + \frac{2L}{N} \right) \quad (3.6)$$

Similar to Mallow's C_P criterion, the choice of $\hat{\sigma}^2$ may require some subjective judgement.

The results obtained by the set of model selection criteria may be different especially for small datasets, which can be revealed by the sensitivity of the model selection criteria to a unit increase in the number of model parameters. Nevertheless, the model selection criteria mentioned above are asymptotically equivalent under some general conditions [133]. Additionally, they are consistent with the basic principle of parsimony (*Ockham's Razor*) to choose the smallest possible model.

3.3 Summary

The generalization performance of networks is fundamentally tied to their topology. The structure and the free parameters of the network will determine the capacity and accuracy of a network mapping. Too small a network cannot accurately approximate the underlying function of the data given (thus giving rise to underfitting problem), while too large a network can have a poor performance due to the overfitting problem. Neural network ensembles and regularization have been used to alleviate the problem of incorrect topology, but they cannot alter the topology itself. There are normally two heuristics to modify the network topology, that is by constructive approach or destructive approach. There are some other algorithms that combine the constructive method with the pruning approach. Model selection criteria are normally used to stop both the constructive and the pruning algorithms. Those criteria are designed based on the statistical procedures, which ensure good performance of the network as well as choosing the smallest possible model following the principle of parsimony.

Chapter 4

OS-ELM Ensemble and Constructive OS-ELM

As described in Section 2.2, OS-ELM can learn the data one-by-one or chunk-by-chunk with fixed or varying chunk size. It has been shown in [61] that OS-ELM runs much faster and provides better generalization performance than other popular sequential learning algorithms, such as stochastic gradient descent back-propagation [134], resource allocation network (RAN)[116], RAN via extended Kalman filter [128], minimal RAN [129][130], growing and pruning RBF [126] and generalized growing and pruning RBF [127]. A brief review of OS-ELM was presented in Section 2.2.1. In this chapter, an OS-ELM ensemble and a constructive improvement of OS-ELM are presented.

4.1 OS-ELM ensemble

A common problem in ELM is that the solutions may fluctuate from one round of simulation to another because the hidden nodes parameters are randomly generated. Although OS-ELM has many advantages compared to those popular sequential algorithms, the same problem may occur when applying OS-ELM. The solutions obtained may dif-

fer from time to time because of the random generation of the hidden nodes parameters.

Neural network ensemble is probably one of good solutions to solve the problem. Early in 1990, the idea of neural network ensembles has already been proposed by Hansen and Salamon [27]. Their work showed that a network's performance can be expected to improve by using an ensemble of neural networks with a plurality consensus scheme. This technique has been widely used after that.

In this section, we propose an integrated network structure, which we called ensemble of online sequential extreme learning machine (EOS-ELM). EOS-ELM comprises several OS-ELM networks. The average value of the outputs of each OS-ELM in the ensemble is used as the final measurement of network performance. Simulation results prove that EOS-ELM is more stable than the original OS-ELM in each trial of simulation for most of problems. And we also conduct the comparison between EOS-ELM and Negative Correlation Learning (NCL) [135]. The results show that EOS-ELM is better than NCL on the applications selected.

4.1.1 Ensemble of OS-ELM (EOS-ELM)

EOS-ELM consists of many OS-ELM networks with the same number of hidden nodes and the same activation function for each hidden node. We constructed P OS-ELM networks to form our EOS-ELM. All of the P OS-ELMs are trained with new data in each incremental step. The input parameters for each OS-ELM network are randomly generated and the output weights are obtained analytically based on the sequentially arrival input data. Then we compute the average of the outputs of each OS-ELM network to provide the final output of the EOS-ELM. Assume that the output of each OS-ELM network is $f^{(j)}(\mathbf{x}_i)$, $j = 1, \dots, P$. Then, we have

$$f(\mathbf{x}_i) = \frac{1}{P} \sum_{j=1}^P f^{(j)}(\mathbf{x}_i) \quad (4.1)$$

where $f(\mathbf{x}_i)$ is the output of the whole system with the input of $\mathbf{x}_i$.

We expect that EOS-ELM works better than an individual OS-ELM network because the randomly generated parameters make each OS-ELM network in the ensemble distinct. Therefore, the OS-ELM networks that make up the ensemble may have different adaptive capacity to the new data. When the data come into the ensemble network sequentially, some of OS-ELM networks may adapt faster and better to the new data than others. However, with respect to different incoming data, different OS-ELM networks would have the good performance, so the EOS-ELM as an ensemble would have a better performance than individual OS-ELMs. The system could avoid the situations where the individual OS-ELM network could not adapt well to the new data, which make the final result of the individual OS-ELM network worse in that round of simulation as compared to other rounds. Statistically speaking, the population mean is always closer to the expectation than the elements in the population, which implies that the results obtained by EOS-ELM would always have a smaller fluctuation range as compared to the results obtained from a single OS-ELM network. When P becomes larger, the mean value $f^{(j)}(\mathbf{x}_i)$ is closer to the expectation of $f^{(j)}(\mathbf{x}_i)$. Therefore, we may conclude that EOS-ELM would be more stable when P is larger. However, when P increases, the computation time also increases and the network becomes more complex. Considering the trade-off between computation time, complexity and stability of the network, we decided to conduct the selection of parameter P from 5, 10, 15, 20, 25, and 30. The simulation results are all presented in the following section.

When all the samples are presented to the network as a whole rather than presented chunk-by-chunk sequentially, i.e. $N_0 = N$, EOS-ELM becomes an ensemble of batch ELM networks [37]. Therefore, the ensemble of ELM proposed in [37] can be seen as a special case of EOS-ELM when all the training data are available at one time.

4.1.2 Performance comparison between EOS-ELM and OS-ELM

In this section, we compare the performance of EOS-ELM with OS-ELM and other sequential algorithms in *one-by-one* learning mode (i.e. the network learns the data one-by-one) on the benchmark problems. The specifications of the benchmark datasets are presented in Table 4.1. All the evaluations were carried out in a Matlab 7.4.0 environment running in a desktop with CPU 2.66GHz and 3GB RAM.

Table 4.1: Specification of benchmark datasets for evaluation of EOS-ELM

Dataset	# Attibutes	# Classes	# Training Data	# Testing Data
Auto-MPG	7	-	320	72
Abalone	8	-	3000	1177
California Housing	8	-	8000	12640
Mackey Glass	4	-	4000	500
Zoo	17	7	71	30
Wine	13	3	120	58
New-thyroid	5	3	140	75
Monks-1	6	2	300	132
Image Segmentation	19	7	1500	810
Satellite Image	36	6	4435	2000
Letter	16	26	18000	2000
Vehicle	18	4	630	216
Adult	14	2	40699	4523
Optical Digits	64	10	1200	4420
Mushroom	21	2	7311	813

In regression and time series cases, the input and output attributes were normalized into the range $[0, 1]$; while in classification cases, the input attributes were normalized into the range $[-1, 1]$. Generally speaking, the data normalization scheme could be the same for classification and regression applications (i.e. the attributes could be normalized into the range $[-1,1]$ or $[0,1]$). Here, we follow the data normalization scheme of [61]. This is so that we can compare the results obtained by the original OS-ELM with the ones obtained by the proposed algorithms.

Two types of activation functions were used in the comparison: the sigmoidal additive activation function $G(\mathbf{a}, b, \mathbf{x}) = (1)/(1 + \exp(-(\mathbf{a} \cdot \mathbf{x} + b)))$, and Gaussian RBF activa-

tion function $G(\mathbf{a}, b, \mathbf{x}) = \exp(-b\|\mathbf{x} - \mathbf{a}\|^2)$. For both EOS-ELM and OS-ELM with additive hidden nodes, the input weights and biases were randomly generated from the range $[-1, 1]$. For both EOS-ELM and OS-ELM with RBF hidden nodes, the centers were randomly generated from the range $[-1, 1]$, and the impact widths b were chosen from the range $(0, 0.5)$ for all cases except the California Housing case. For EOS-ELM and OS-ELM in the California Housing case, the impact width b was generated from the range $(0, 1)$.

Further, the number of training data N_0 for initialization phase is set the same as [61] with $N_0 = L + 50$ for the regression cases and $N_0 = L + 100$ for the classification cases of Image Segment and Satellite Image; but different from [61], we set $N_0 = L + 2000$ for the time-series case, and $N_0 = L$ for the classification cases: Zoo, Wine, New-thyroid, Monks-1, Letter, Vehicle, Adult, Optical Digits and Mushroom, where L is the network size.

Parameter selection

For OS-ELM, there is only one parameter, the optimal number of hidden nodes, which needs to be determined. We make use of the 5-folder cross validation method for choosing this parameter. 50 trials of simulations are conducted for each problem from regression, classification or time series area. The optimal number of hidden nodes is chosen for a particular problem in such a way that the network provides the lowest validation error. An example of parameter selection of the optimal number of hidden nodes for OS-ELM with sigmoidal activation function has been shown in [61] for the Auto-MPG case.

For EOS-ELM, we set the number of hidden nodes used in each OS-ELM network that make up the EOS-ELM to be equal to the optimal number of hidden nodes for OS-ELM network that we want to compare with. And there is one more parameter that needs to be determined, which is the number of OS-ELM networks in the ensemble. We set this parameter to be 5, 10, 15, 20, 25 and 30 in our simulations, and select the one

based on the average RMSE or Testing Accuracy and Standard Deviation (SD) value. The optimal number of networks is chosen in such a way that the ensemble network provided the lowest SD value and the average RMSE or Testing Accuracy is better or comparable with the results obtained by OS-ELM for the same application. An example for selecting the number of OS-ELM networks (with sigmoidal activation function) in the ensemble for the New-thyroid case is shown below.

Table 4.2: EOS-ELM model selection for New-thyroid case

#hidden nodes=20	OS-ELM	EOS-ELM					
# OS-ELMs	1	5	10	15	20	25	30
Testing Accuracy	89.68%	90.18%	89.6%	89.6%	90.8%	90.33%	89.9%
Testing SD	0.1134	0.0377	0.0449	0.0386	0.0283	0.033	0.0297

As observed from Table 4.2, the lowest SD value is obtained when the number of OS-ELMs in the ensemble is 20, and the testing accuracy achieved by EOS-ELM is better than a single OS-ELM as well. Hence we have selected the number of OS-ELMs in the EOS-ELM as 20 in our experiments.

Evaluation methods

To measure and compare stability of the algorithm, we calculate the standard deviation of RMSE (for regression and time series) and Testing Accuracy (for classification) over 50 runs. Additionally, following the idea in [37], a new parameter is introduced, i.e. Coefficient of Variation (CV for short). CV is a dimensionless measure and can be used to compare the amount of variance between populations with different means. And CV is defined as the ratio between standard deviation (SD for short) and the mean, i.e. $CV = SD/mean$.

Furthermore, we conduct one round simulation to show the learning evolution of EOS-ELM and OS-ELM on the Auto-MPG application for both sigmoid and RBF hidden nodes cases. The simulation results (the number of samples for training vs. RMSE) are

presented in Fig. 4.1 and Fig. 4.2. The model specifications for both cases are stated in Table 4.3 and Table 4.4.

Performance comparison between EOS-ELM and OS-ELM

In this section, we evaluate the performance of EOS-ELM and OS-ELM in the *one-by-one* learning mode. For each problem, the simulation is conducted 50 times. And for each trial of simulation, the training and testing data are randomly selected from the given dataset. The simulation results are presented in Table 4.3 and Table 4.4.

Table 4.3: Comparison between EOS-ELM and OS-ELM (sigmoid hidden nodes)

Datasets	Algorithms	# Nodes	# Network	Training Time(s)	RMSE		SD		CV
					Training	Testing	Training	Testing	
Auto-MPG	OS-ELM	25		0.0169	0.069	0.0742	0.0019	0.0088	0.1186
	EOS-ELM	25	15	0.2428	0.0685	0.0762	0.0016	0.0068	0.0892
Abalone	OS-ELM	25		0.1672	0.0756	0.078	0.001	0.0047	0.0603
	EOS-ELM	25	5	0.8844	0.0755	0.0777	0.0009	0.0024	0.0309
California	OS-ELM	50		0.9669	0.1299	0.1331	0.002	0.0033	0.0248
Housing	EOS-ELM	50	5	4.3431	0.1302	0.133	0.0015	0.0017	0.0128
Mackey	OS-ELM	120		1.3741	0.0179	0.0184	0.0015	0.0016	0.087
Glass	EOS-ELM	120	5	6.8678	0.0179	0.0183	0.0006	0.0006	0.0328
Datasets	Algorithms	# Nodes	# Network	Training Time(s)	Accuracy		SD		CV
					Training	Testing	Training	Testing	
Zoo	OS-ELM	55		0.0063	100%	93.07%	0	0.0494	0.0531
	EOS-ELM	55	30	0.1856	100%	93.64%	0.0001	0.0378	0.0404
Wine	OS-ELM	25		0.0072	99.83%	97.24%	0.0034	0.0249	0.0256
	EOS-ELM	25	5	0.0356	99.78%	97.39%	0.0022	0.0121	0.0124
New-thyroid	OS-ELM	20		0.0066	93.17%	89.68%	0.1049	0.1134	0.1264
	EOS-ELM	20	20	0.135	94.27%	90.8%	0.0192	0.02838	0.0312
Monks-1	OS-ELM	70		0.0541	89.35%	78.74%	0.0179	0.0327	0.0415
	EOS-ELM	70	20	1.0872	88.58%	78.77%	0.0127	0.0189	0.024
Image Segmentation	OS-ELM	180		2.6472	97.06%	94.82%	0.0033	0.0073	0.0077
	EOS-ELM	180	25	74.1669	97.08%	94.79%	0.0023	0.0057	0.006
Satellite	OS-ELM	400		63.4181	92.82%	88.9%	0.0039	0.0057	0.0064
Image	EOS-ELM	400	25	1591.763	92.79%	89.01%	0.0024	0.0029	0.0033

We select three regression problems: Auto-MPG, Abalone, and California Housing and six classification problems: Zoo, Wine, New-thyroid, Monks-1, Image Segment, and Satellite Image from [136]. Additionally, the performance of EOS-ELM is tested on the time-series prediction application: chaotic Mackey-Glass case given by Mackey and Glass [137]. For the detail data generation for the Mackey-Glass case, please refer to [61].

Table 4.4: Comparison between EOS-ELM and OS-ELM (RBF hidden nodes)

Datasets	Algorithms	# Nodes	# Network	Training Time(s)	RMSE		SD		CV
					Training	Testing	Training	Testing	
Auto-MPG	OS-ELM	25		0.0422	0.0682	0.0762	0.0025	0.0079	0.1037
	EOS-ELM	25	20	0.8469	0.0686	0.0752	0.0017	0.0071	0.0944
Abalone	OS-ELM	25		0.4828	0.0752	0.0774	0.0009	0.0025	0.0323
	EOS-ELM	25	25	12.2753	0.0752	0.0776	0.0007	0.002	0.0258
California Housing	OS-ELM	50		2.6078	0.1295	0.1315	0.0017	0.0018	0.0137
	EOS-ELM	50	5	12.5772	0.1296	0.1315	0.0013	0.0011	0.0084
Mackey Glass	OS-ELM	120		2.43	0.0182	0.0184	0.0087	0.0085	0.462
	EOS-ELM	120	10	24.2913	0.0171	0.0174	0.0013	0.0012	0.069
Datasets	Algorithms	# Nodes	# Network	Training Time(s)	Accuracy		SD		CV
					Training	Testing	Training	Testing	
Zoo	OS-ELM	35		0.0109	99.89%	91.13%	0.0039	0.0502	0.0551
	EOS-ELM	35	5	0.055	99.81%	90.44%	0.0028	0.0436	0.0482
Wine	OS-ELM	30		0.0184	99.72%	97.07%	0.0043	0.0221	0.0228
	EOS-ELM	30	15	0.2863	99.74%	97.19%	0.0013	0.0134	0.0138
New-thyroid	OS-ELM	20		0.0172	93.4%	89.95%	0.0816	0.0709	0.0788
	EOS-ELM	20	5	0.0884	93.71%	89.89%	0.036	0.0436	0.0485
Monks-1	OS-ELM	80		0.1412	94.55%	87.18%	0.0875	0.0859	0.0985
	EOS-ELM	80	15	2.1303	93.91%	86.29%	0.0225	0.0298	0.0345
Image Segmentation	OS-ELM	180		3.8756	94.96%	91.8%	0.0266	0.0312	0.034
	EOS-ELM	180	20	72.0784	94.34%	91.23%	0.0132	0.0142	0.0156
Satellite Image	OS-ELM	400		68.3988	93.7%	89.52%	0.0037	0.0053	0.0059
	EOS-ELM	400	5	343.85	93.82%	89.28%	0.0027	0.0037	0.0041

Table 4.3 and Table 4.4 show that EOS-ELM takes more time in training which is reasonable. And the SD and CV values obtained by EOS-ELM for all the cases are always lower than OS-ELM, which means EOS-ELM is more stable and reliable. This property is also proved by the learning evolution curves, which is presented in Fig. 4.1 and Fig. 4.2. From both of the figures, we can see that the evolution curves for EOS-ELM are much smoother than OS-ELM, with which we could conclude that EOS-ELM is more stable in the incremental learning process. Comparing EOS-ELM for regression and classification applications, we found that the advantage of EOS-ELM is more obvious for classification applications.

4.1.3 Performance comparison between EOS-ELM and other sequential algorithms

In this section, the comparison between EOS-ELM and Negative Correlation Learning (NCL) [135] is presented on five classification applications, which are Letter, Vehicle,

Figure 4.1: Learning evolution for Auto-MPG case (sigmoid hidden nodes)

Adult, Optical Digits and Mushroom. The specifications of the applications are stated in Table 4.1.

There are two approaches for NCL used in incremental learning [135]: Fixed Size NCL and Growing (Size) NCL. The architecture of Fixed Size NCL is an ensemble of Multi-layer Perceptron Networks (MLP). “Fixed size” means that all the MLPs that compose the ensemble are trained with the new data in each incremental step. Whereas, in Growing NCL, there is only one MLP network in the ensemble initially. New MLP network is inserted in each incremental step for learning the new data. The previously inserted MLP networks in the ensemble do not receive any new data and the outputs to the new data are calculated analytically.

Table 4.5 shows the comparison results, and only the testing accuracy is presented here. The results for MLP, Fixed Size NCL and Growing NCL are taken from [135] directly. The activation function used in both OS-ELM and EOS-ELM is sigmoid. From Table 4.5, it shows that EOS-ELM is better than NCL on four applications, which are Letter,

Figure 4.2: Learning evolution for Auto-MPG case (RBF hidden nodes)

Vehicle, Optical Digits and Mushroom and worse on one application, which is Adult.

Table 4.5: Comparison between EOS-ELM and NCL on classification applications

Datasets	MLP	Fixed NCL	Growing NCL	OS-ELM	EOS-ELM
Letter	73.92%	81.617%	73.193%	77.05%	76.4%
Vehicle	74.63%	76.898%	77.269%	80.4%	80.51%
Adult	84.321%	84.294%	83.823%	80.74%	82.78%
Optical Digits	89.153%	90.464%	90.027%	92.6%	95.98%
Mushroom	99.938%	99.926%	99.377%	97.87%	99.51%

4.1.4 Summary

Learning data sequentially is badly required when the dataset cannot be obtained at once or the previous data needs to be discarded after learning. The OS-ELM developed by Liang et al. [61] is an efficient sequential learning algorithm which can handle the data one-by-one and chunk-by-chunk with fixed or varying chunk size. OS-ELM produces

better generalization performance with lower training time as compared to other well-known sequential learning algorithms. In this section, we have further improved the stability of OS-ELM by incorporating EOS-ELM. EOS-ELM can provide more stable and accurate results when compared to the single OS-ELM network. The comparison of EOS-ELM with NCL is also presented in this section and it has been shown that EOS-ELM performs better than NCL.

4.2 Constructive OS-ELM

As described in Chapter 3, the mapping capability of SLFNs depends on the correct determination of network structures, i.e. the number of hidden nodes in the hidden layer. Too small a network cannot learn the problem well; while too large a network may lead to overfitting and poor generalization performance. Similar to the preliminary ELM, OS-ELM relies on a fixed network structure determined by a trial and error method that could be rather time consuming.

A constructive improvement [138] has been proposed to handle the problem, which is named an enhanced online sequential extreme learning machine algorithm (EOS-ELM). EOS-ELM is designed for SLFNs with RBF hidden nodes based on minimal resource allocation network (MRAN) algorithm. Although a more compact network could be found using EOS-ELM, the approach is complicated and only suitable for RBF hidden nodes.

In this section, we propose a constructive enhancement for OS-ELM (i.e. CEOS-ELM), which can add random hidden nodes *one-by-one* or *group-by-group* based on EM-ELM algorithm [28]. CEOS-ELM can handle various type of hidden nodes, including additive, RBF hidden nodes, etc. The proposed algorithm is evaluated and compared with OS-ELM in *one-by-one* and *chunk-by-chunk* (fixed and varying chunk size) learning modes. Additionally, we analyze the performance of CEOS-ELM in different incremental modes. The results show that CEOS-ELM is fast and can automatically provide

the suitable number of hidden nodes according to user specified error with good generalization performance.

4.2.1 A constructive enhancement for OS-ELM (COS-ELM)

Adding new hidden nodes into the network

OS-ELM can learn data *one-by-one* or *chunk-by-chunk* sequentially. However, the network structure of OS-ELM needs to be determined beforehand and it is fixed in the sequential learning process. While in EM-ELM network, the EM-ELM algorithm searches for an appropriate network structure in the learning process. However, EM-ELM is a batch learning algorithm which requires the availability of all the data. Therefore, it is not suitable for sequential learning. Our work is to develop a new algorithm, which can explore the proper size of the network in the sequential learning process. The big challenge is how to add new hidden nodes into the existing SLFN in sequential learning.

From the background knowledge in Section 2.2.2, we know that when new hidden nodes are added into the network, the hidden layer output matrix $\mathbf{H}$ is expanded with more columns. Assume that the hidden layer output matrix in the initial step is $\mathbf{H}_0$, and we are given another chunk of data $\mathfrak{X}_1 = \{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=N_0+1}^{N_0+N_1}$, where N_0, N_1 denotes the number of samples in the initial chunk and new chunk respectively. Assume that the network cannot achieve the target accuracy and new hidden nodes need to be added. Therefore, the hidden layer output matrix, say $\mathbf{H}_1$, needs to be expanded.

According to OS-ELM, the original problem is to minimize $\left\| \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix} \beta^{(1)} - \begin{bmatrix} \mathbf{T}_0 \\ \mathbf{T}_1 \end{bmatrix} \right\|$, and now becomes one of minimizing

$$\left\| \begin{bmatrix} \mathbf{H}_0 & \mathbf{Z}_0 \\ \mathbf{H}_1 & \mathbf{Z}_1 \end{bmatrix} \hat{\beta}^{(1)} - \begin{bmatrix} \mathbf{T}_0 \\ \mathbf{T}_1 \end{bmatrix} \right\| \quad (4.2)$$

where $\mathbf{Z}_0$ is the output matrix of newly added hidden nodes corresponding to the previ-

ous chunk of data, and $\mathbf{Z}_1$ is the output matrix of newly added hidden nodes corresponding to data $\mathfrak{X}_1$.

Therefore, we have

$$\hat{\beta}^{(1)} = \hat{\mathbf{K}}_1^{-1} \begin{bmatrix} \mathbf{H}_0 & \mathbf{Z}_0 \\ \mathbf{H}_1 & \mathbf{Z}_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{T}_0 \\ \mathbf{T}_1 \end{bmatrix} \quad (4.3)$$

where

$$\hat{\mathbf{K}}_1 = \begin{bmatrix} \mathbf{H}_0 & \mathbf{Z}_0 \\ \mathbf{H}_1 & \mathbf{Z}_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{H}_0 & \mathbf{Z}_0 \\ \mathbf{H}_1 & \mathbf{Z}_1 \end{bmatrix} \quad (4.4)$$

From Section 2.2.1, we have $\mathbf{K}_0 = \mathbf{H}_0^T \mathbf{H}_0 = \mathbf{P}_0^{-1}$, $\beta^{(0)} = (\mathbf{H}_0^T \mathbf{H}_0)^{-1} \mathbf{H}_0^T \mathbf{T}_0 = \mathbf{P}_0 \mathbf{H}_0^T \mathbf{T}_0$, and $\mathbf{K}_1 = \mathbf{H}_0^T \mathbf{H}_0 + \mathbf{H}_1^T \mathbf{H}_1 = \mathbf{K}_0 + \mathbf{H}_1^T \mathbf{H}_1$. So $\hat{\mathbf{K}}_1$ can be written as

$$\hat{\mathbf{K}}_1 = \begin{bmatrix} \mathbf{H}_0^T & \mathbf{H}_1^T \\ \mathbf{Z}_0^T & \mathbf{Z}_1^T \end{bmatrix} \begin{bmatrix} \mathbf{H}_0 & \mathbf{Z}_0 \\ \mathbf{H}_1 & \mathbf{Z}_1 \end{bmatrix} = \begin{bmatrix} \mathbf{K}_1 & \mathbf{H}_0^T \mathbf{Z}_0 + \mathbf{H}_1^T \mathbf{Z}_1 \\ \mathbf{Z}_0^T \mathbf{H}_0 + \mathbf{Z}_1^T \mathbf{H}_1 & \mathbf{Z}_0^T \mathbf{Z}_0 + \mathbf{Z}_1^T \mathbf{Z}_1 \end{bmatrix} \quad (4.5)$$

and

$$\begin{aligned} & \begin{bmatrix} \mathbf{H}_0 & \mathbf{Z}_0 \\ \mathbf{H}_1 & \mathbf{Z}_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{T}_0 \\ \mathbf{T}_1 \end{bmatrix} = \begin{bmatrix} \mathbf{H}_0^T \mathbf{T}_0 + \mathbf{H}_1^T \mathbf{T}_1 \\ \mathbf{Z}_0^T \mathbf{T}_0 + \mathbf{Z}_1^T \mathbf{T}_1 \end{bmatrix} \\ & = \begin{bmatrix} \mathbf{K}_1 \beta^{(0)} - \mathbf{H}_1^T \mathbf{H}_1 \beta^{(0)} + \mathbf{H}_1^T \mathbf{T}_1 \\ \mathbf{Z}_0^T \mathbf{T}_0 + \mathbf{Z}_1^T \mathbf{T}_1 \end{bmatrix} = \begin{bmatrix} \mathbf{G}_1 \\ \mathbf{G}_2 \end{bmatrix} \end{aligned} \quad (4.6)$$

Assume

$$\hat{\mathbf{P}}_1 = \begin{bmatrix} \hat{\mathbf{P}}_{11} & \hat{\mathbf{P}}_{12} \\ \hat{\mathbf{P}}_{21} & \hat{\mathbf{P}}_{22} \end{bmatrix} = \hat{\mathbf{K}}_1^{-1} = \begin{bmatrix} \mathbf{K}_1 & \mathbf{H}_0^T \mathbf{Z}_0 + \mathbf{H}_1^T \mathbf{Z}_1 \\ \mathbf{Z}_0^T \mathbf{H}_0 + \mathbf{Z}_1^T \mathbf{H}_1 & \mathbf{Z}_0^T \mathbf{Z}_0 + \mathbf{Z}_1^T \mathbf{Z}_1 \end{bmatrix}^{-1} \quad (4.7)$$

then

$$\hat{\beta}^{(1)} = \begin{bmatrix} \hat{\mathbf{U}} \\ \hat{\mathbf{D}} \end{bmatrix} = \begin{bmatrix} \hat{\mathbf{P}}_{11} \mathbf{G}_1 + \hat{\mathbf{P}}_{12} \mathbf{G}_2 \\ \hat{\mathbf{P}}_{21} \mathbf{G}_1 + \hat{\mathbf{P}}_{22} \mathbf{G}_2 \end{bmatrix} \quad (4.8)$$

Let $\mathbf{S} = (\mathbf{Z}_0^T \mathbf{Z}_0 + \mathbf{Z}_1^T \mathbf{Z}_1) - (\mathbf{Z}_0^T \mathbf{H}_0 + \mathbf{Z}_1^T \mathbf{H}_1) \mathbf{P}_1 (\mathbf{H}_0^T \mathbf{Z}_0 + \mathbf{H}_1^T \mathbf{Z}_1)$, which is the Schur complement of $\mathbf{K}_1$, where $\mathbf{P}_1 = \mathbf{K}_1^{-1}$. The matrix $\hat{\mathbf{P}}_1$ is invertible if and only if $\mathbf{S}$ is invertible [72]. And based on the inversion of 2×2 block matrices, we have

$$\begin{aligned} \hat{\mathbf{P}}_{11} &= \mathbf{P}_1 + \mathbf{P}_1 (\mathbf{H}_0^T \mathbf{Z}_0 + \mathbf{H}_1^T \mathbf{Z}_1) \mathbf{S}^{-1} (\mathbf{Z}_0^T \mathbf{H}_0 + \mathbf{Z}_1^T \mathbf{H}_1) \mathbf{P}_1 \\ \hat{\mathbf{P}}_{12} &= -\mathbf{P}_1 (\mathbf{H}_0^T \mathbf{Z}_0 + \mathbf{H}_1^T \mathbf{Z}_1) \mathbf{S}^{-1} \\ \hat{\mathbf{P}}_{21} &= -\mathbf{S}^{-1} (\mathbf{Z}_0^T \mathbf{H}_0 + \mathbf{Z}_1^T \mathbf{H}_1) \mathbf{P}_1 \\ \hat{\mathbf{P}}_{22} &= \mathbf{S}^{-1} \end{aligned} \quad (4.9)$$

then

$$\begin{aligned} \hat{\mathbf{U}} &= (\mathbf{I} + \mathbf{P}_1 (\mathbf{H}_0^T \mathbf{Z}_0 + \mathbf{H}_1^T \mathbf{Z}_1) \mathbf{S}^{-1} (\mathbf{Z}_0^T \mathbf{H}_0 + \mathbf{Z}_1^T \mathbf{H}_1)) (\beta^{(0)} + \mathbf{P}_1 \mathbf{H}_1^T (\mathbf{T}_1 - \mathbf{H}_1 \beta^{(0)})) \\ &\quad - \mathbf{P}_1 (\mathbf{H}_0^T \mathbf{Z}_0 + \mathbf{H}_1^T \mathbf{Z}_1) \mathbf{S}^{-1} (\mathbf{Z}_0^T \mathbf{T}_0 + \mathbf{Z}_1^T \mathbf{T}_1) \\ \hat{\mathbf{D}} &= -\mathbf{S}^{-1} (\mathbf{Z}_0^T \mathbf{H}_0 + \mathbf{Z}_1^T \mathbf{H}_1) (\beta^{(0)} + \mathbf{P}_1 \mathbf{H}_1^T (\mathbf{T}_1 - \mathbf{H}_1 \beta^{(0)})) + \mathbf{S}^{-1} (\mathbf{Z}_0^T \mathbf{T}_0 + \mathbf{Z}_1^T \mathbf{T}_1) \end{aligned} \quad (4.10)$$

The formulae (4.7) and (4.8) show the method for updating the output weight β when the number of hidden nodes grows during the sequential learning process. Similar derivations can be found in [138].

Summary of CEOS-ELM algorithm

Given a set of training data $\{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=1}^N$, the number of samples used to initialize the training N_0 , the number of samples at each sequential learning step N_k , the initial number of hidden nodes L_0 , the number of hidden nodes used to add into the network in each incremental step δL_j , the maximum number of hidden nodes L_{max} , and the expected learning accuracy ε , the proposed algorithm is summarized as follows.

Step 1 Randomly generate L_0 hidden nodes and train the network using N_0 initial data.

Set $j = 0$ and $k = 0$. If the training algorithm can achieve the expected learning accuracy ε , go to Step 2; otherwise, go to Step 4 for growing.

Step 2 If $L_j < L_{max}$, train the network using another chunk of data that have N_{k+1} samples with L_j hidden nodes.

Step 3 If the training algorithm can achieve the expected learning accuracy ε , the output weight β can be updated as follows: If no new hidden nodes have been added, use equations (4.11) and (4.12):

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \mathbf{P}_k \mathbf{H}_{k+1}^T (\mathbf{I} + \mathbf{H}_{k+1} \mathbf{P}_k \mathbf{H}_{k+1}^T)^{-1} \mathbf{H}_{k+1} \mathbf{P}_k \quad (4.11)$$

$$\beta^{(k+1)} = \beta^{(k)} + \mathbf{P}_{k+1} \mathbf{H}_{k+1}^T (\mathbf{T}_{k+1} - \mathbf{H}_{k+1} \beta^{(k)}) \quad (4.12)$$

If new hidden nodes have been added, the output weight has to be denoted by $\hat{\beta}$ and the system parameter $\mathbf{P}$ has to be denoted by $\hat{\mathbf{P}}$ (The parameter $\mathbf{P}$ below is just an intermediate parameter). $\hat{\mathbf{P}}$ and $\hat{\beta}$ can be updated using equations (4.14)

and (4.18):

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \mathbf{P}_k \mathbf{H}_{k+1}^T (\mathbf{I} + \mathbf{H}_{k+1} \mathbf{P}_k \mathbf{H}_{k+1}^T)^{-1} \mathbf{H}_{k+1} \mathbf{P}_k \quad (4.13)$$

$$\hat{\mathbf{P}}_{k+1} = \begin{bmatrix} \hat{\mathbf{P}}_{11} & \hat{\mathbf{P}}_{12} \\ \hat{\mathbf{P}}_{21} & \hat{\mathbf{P}}_{22} \end{bmatrix} \quad (4.14)$$

where

$$\begin{aligned} \hat{\mathbf{P}}_{11} &= \mathbf{P}_{k+1} + \mathbf{P}_{k+1} (\mathbf{H}_k^T \mathbf{Z}_k + \mathbf{H}_{k+1}^T \mathbf{Z}_{k+1}) \mathbf{S}^{-1} (\mathbf{Z}_k^T \mathbf{H}_k + \mathbf{Z}_{k+1}^T \mathbf{H}_{k+1}) \mathbf{P}_{k+1} \\ \hat{\mathbf{P}}_{12} &= -\mathbf{P}_{k+1} (\mathbf{H}_k^T \mathbf{Z}_k + \mathbf{H}_{k+1}^T \mathbf{Z}_{k+1}) \mathbf{S}^{-1} \\ \hat{\mathbf{P}}_{21} &= -\mathbf{S}^{-1} (\mathbf{Z}_k^T \mathbf{H}_k + \mathbf{Z}_{k+1}^T \mathbf{H}_{k+1}) \mathbf{P}_{k+1} \\ \hat{\mathbf{P}}_{22} &= \mathbf{S}^{-1} \end{aligned} \quad (4.15)$$

and

$$\mathbf{S} = (\mathbf{Z}_k^T \mathbf{Z}_k + \mathbf{Z}_{k+1}^T \mathbf{Z}_{k+1}) - (\mathbf{Z}_k^T \mathbf{H}_k + \mathbf{Z}_{k+1}^T \mathbf{H}_{k+1}) \mathbf{P}_{k+1} (\mathbf{H}_k^T \mathbf{Z}_k + \mathbf{H}_{k+1}^T \mathbf{Z}_{k+1}) \quad (4.16)$$

then

$$\begin{aligned} \hat{\mathbf{U}} &= (\mathbf{I} + \mathbf{P}_{k+1} (\mathbf{H}_k^T \mathbf{Z}_k + \mathbf{H}_{k+1}^T \mathbf{Z}_{k+1}) \mathbf{S}^{-1} (\mathbf{Z}_k^T \mathbf{H}_k + \mathbf{Z}_{k+1}^T \mathbf{H}_{k+1})) \\ &\quad \times (\beta^{(k)} + \mathbf{P}_{k+1} \mathbf{H}_{k+1}^T (\mathbf{T}_{k+1} - \mathbf{H}_{k+1} \beta^{(k)})) \\ &\quad - \mathbf{P}_{k+1} (\mathbf{H}_k^T \mathbf{Z}_k + \mathbf{H}_{k+1}^T \mathbf{Z}_{k+1}) \mathbf{S}^{-1} (\mathbf{Z}_k^T \mathbf{T}_k + \mathbf{Z}_{k+1}^T \mathbf{T}_{k+1}) \\ \hat{\mathbf{D}} &= -\mathbf{S}^{-1} (\mathbf{Z}_k^T \mathbf{H}_k + \mathbf{Z}_{k+1}^T \mathbf{H}_{k+1}) (\beta^{(k)} + \mathbf{P}_{k+1} \mathbf{H}_{k+1}^T (\mathbf{T}_{k+1} - \mathbf{H}_{k+1} \beta^{(k)})) \\ &\quad + \mathbf{S}^{-1} (\mathbf{Z}_k^T \mathbf{T}_k + \mathbf{Z}_{k+1}^T \mathbf{T}_{k+1}) \end{aligned} \quad (4.17)$$

$$\hat{\beta}^{(k+1)} = \begin{bmatrix} \hat{\mathbf{U}} \\ \hat{\mathbf{D}} \end{bmatrix} \quad (4.18)$$

Set $k = k + 1$. Go to Step 2 for training.

If the training algorithm cannot achieve the expected learning accuracy ε , go to Step 4 for growing.

Step 4 Let $\mathbf{H}_j = \mathbf{H}_k$.

While $L_j < L_{max}$ and $E(\mathbf{H}_j) > \varepsilon$, randomly add δL_j hidden nodes into network in each incremental step. The number of hidden nodes in the network now becomes $L_{j+1} = L_j + \delta L_j$ and the corresponding hidden layer output matrix becomes $\mathbf{H}_{j+1} = [\mathbf{H}_j, \delta \mathbf{H}_j]$. Then the output weight β can be updated as follows:

$$\begin{aligned} \mathbf{D}_j &= ((\mathbf{I} - \mathbf{H}_j \mathbf{H}_j^\dagger) \delta \mathbf{H}_j)^\dagger \\ \mathbf{U}_j &= \mathbf{H}_j^\dagger - \mathbf{H}_j^\dagger \delta \mathbf{H}_j \mathbf{D}_j \\ \beta^{(j+1)} &= \mathbf{H}_{j+1}^\dagger \mathbf{T} = \begin{bmatrix} \mathbf{U}_j \\ \mathbf{D}_j \end{bmatrix} \mathbf{T} \end{aligned} \quad (4.19)$$

Set $j = j + 1$.

End while.

Set $\mathbf{H}_k = \mathbf{H}_j$ and $\beta^{(k)} = \beta^{(j)}$. Go to Step 2.

4.2.2 Performance evaluation of CEOS-ELM

The performance of CEOS-ELM is evaluated on the benchmark problems described in Table 4.6, which include four applications (Regression case: Abalone and California Housing; Classification cases: Image Segmentation and Satellite Image) [136]. And it has been compared with the original OS-ELM algorithm.

Table 4.6: Specification of benchmark datasets for evaluation of CEOS-ELM

Dataset	# Attributes	# Classes	# Training Data	# Testing Data
Abalone	8	-	3000	1177
California Housing	8	-	8000	12640
Image Segmentation	19	7	1500	810
Satellite Image	36	6	4435	2000

All the evaluations were carried out in the Matlab 7.4.0 environment running on a desktop with CPU 2.66GHz and 3GB RAM. Twenty trials of simulations were conducted for each applications with both OS-ELM and CEOS-ELM algorithms. For each trial of simulations, the dataset of the application was divided into training and testing datasets with the number of samples indicated in Table 4.6. The process is random based, which means, take Abalone for example, we randomly select 3000 samples out of 4177 to form the training dataset, and the rest of samples are in the testing dataset. The activation function used in the comparison is the sigmoidal additive activation function (Sig) $G(\mathbf{a}, b, \mathbf{x}) = (1)/(1 + \exp(-(\mathbf{a} \cdot \mathbf{x} + b)))$. In regression cases, the input and output attributes were normalized into the range $[0, 1]$; while in classification cases, the input attributes were normalized into the range $[-1, 1]$. For both CEOS-ELM and OS-ELM with additive hidden nodes, the input weights and biases were randomly generated from the range $[-1, 1]$.

Parameter selection

For the original OS-ELM, two parameters need to be considered: the optimal number of hidden nodes in the hidden layer, i.e. $\#nodes$, and the number of samples used to initialize the training N_0 . $\#nodes$ is selected following the trial and error method stated in the paper [61]. As mentioned in [61], N_0 is usually set to be equal to or greater than $\#nodes$ so as to make hidden layer output matrix full rank. Hence, in our simulations, N_0 has been set greater than $\#nodes$.

For CEOS-ELM, there is one parameter that needs to be determined by the user, and

this is the expected learning accuracy ε . The maximum number of hidden nodes L_{max} can be set to any large value. (In our simulation, $L_{max}=1000$.) The number of samples used to initialize the training N_0 is set to be the same as the values used in the OS-ELM algorithm. Additionally, CEOS-ELM introduces three parameters (the number of samples at each sequential learning step N_k , the initial number of hidden nodes L_0 , and the number of hidden nodes to be added into the network in each incremental step δL_j), which increase the adaptation capability of the algorithm to different applications. N_k usually depends on the application itself, i.e. how much data is available in each sequential learning process, whereas L_0 and δL_j could be set freely in principle. These parameters will be further discussed in the next section. The parameter settings for OS-ELM and CEOS-ELM are shown in Table 4.7.

Table 4.7: Parameters setting for regression (Abalone and California Housing) and classification (Image Segmentation and Satellite Image) problems

Dataset	Algorithm	# nodes	N_0	L_{max}	ε
Abalone	OS-ELM	25	75		
	CEOS-ELM		75	1000	0.06
California Housing	OS-ELM	50	200		
	CEOS-ELM		200	1000	0.11
Dataset	Algorithm	# nodes	N_0	L_{max}	ε
Image Segmentation	OS-ELM	200	500		
	CEOS-ELM		500	1000	98%
Satellite Image	OS-ELM	300	800		
	CEOS-ELM		800	1000	95%

Performance evaluation of CEOS-ELM: learning mode (N_k)

In this subsection, we evaluate and compare the performance of the proposed CEOS-ELM with OS-ELM in one-by-one and chunk-by-chunk (with fixed and varying chunk size) learning modes, i.e. $N_k = 1$, $N_k = AnyValue > 1$, and $N_k = [10, 30]$ (i.e. randomly pick an integer number from the range $[10, 30]$). Without loss of generality, we set $L_0 = 1$ and $\delta L_j = 1$.

As mentioned in the previous section, the number of samples presented to the network at each sequential learning step N_k usually depends on the application itself. The number of samples that are available for training may be one or more in each sequential learning step. Therefore, the simulations that we conducted cover almost all the situations that may happen in real world applications.

Table 4.8: Learning mode comparison on regression cases (sigmoidal additive hidden nodes, $L_0 = 1$, $\delta L_j = 1$)

Dataset	Algorithm	Training Time(s)	Training RMSE	Testing RMSE	# Nodes	N_k
Abalone	OS-ELM	0.1675	0.0757	0.0774	25	1
		0.0334	0.0754	0.0776	25	20
		0.0416	0.0755	0.0775	25	[10,30]
	CEOS-ELM	0.5000	0.0761	0.0769	25.95	1
		0.0789	0.0758	0.0791	24.95	20
		0.1656	0.0764	0.0781	25.3	[10,30]
California Housing	OS-ELM	0.8256	0.1301	0.1333	50	1
		0.1475	0.1302	0.1329	50	20
		0.1716	0.1298	0.1330	50	[10,30]
	CEOS-ELM	4.1430	0.1315	0.1347	46.15	1
		0.5305	0.1305	0.1336	49.85	20
		0.5383	0.1319	0.1360	48.15	[10,30]

Table 4.8 shows the results for regression cases with sigmoidal hidden nodes. For both CEOS-ELM and OS-ELM networks, we have found that the networks with one-by-one learning mode and chunk-by-chunk learning mode achieve similar performance. However, when use the one-by-one learning mode, the network requires much more time in training. In addition, there is not much difference in network performance between the fixed and varying chunk size cases. Comparing CEOS-ELM with the OS-ELM network, CEOS-ELM spends more time in training which is reasonable because the training time for OS-ELM recorded in the table is only for the fixed structure situation, which means the time for selecting the network structure is excluded and this process is actually time wasting. The optimal network structure obtained by CEOS-ELM for each regression application is similar to the network structure obtained by using trial and error method for OS-ELM, which indicates that for regression applications, the trial and error method

can achieve the optimal solution.

Table 4.9: Learning mode comparison on classification cases (sigmoidal additive hidden nodes, $L_0 = 1$, $\delta L_j = 1$)

Dataset	Algorithm	Training Time(s)	Training Accuracy	Testing Accuracy	# Nodes	N_k
Image Segmentation	OS-ELM	2.9031	97.34%	94.76%	200	1
		0.3934	97.34%	95.04%	200	20
		0.4088	97.39%	94.78%	200	[10,30]
	CEOS-ELM	1.8531	96.22%	94.51%	135.7	1
		0.8047	96.37%	94.51%	145.35	20
		0.9023	96.36%	94.55%	153.5	[10,30]
Satellite Image	OS-ELM	24.2203	91.67%	88.76%	300	1
		2.1516	91.78%	88.65%	300	20
		2.2072	91.67%	88.79%	300	[10,30]
	CEOS-ELM	13.107	90.75%	88.29%	224.2	1
		3.0258	90.84%	88.23%	221.3	20
		2.8984	90.58%	88.19%	215.75	[10,30]

The simulation results for two classification applications are presented in Table 4.9 with sigmoidal hidden nodes. The results obtained in one-by-one and chunk-by-chunk learning modes are comparable. Therefore, the network with the chunk-by-chunk learning mode is superior to one-by-one learning mode because it requires less training time. Furthermore, the trial and error method may not work well for classification cases, and CEOS-ELM provides a more compact network structure for both of the classification cases.

Performance evaluation of CEOS-ELM: incremental mode (δL_j)

In this subsection, the performance of CEOS-ELM is evaluated with different incremental modes (i.e. adding hidden nodes one-by-one ($\delta L_j = 1$) and group-by-group ($\delta L_j = AnyValue > 1$)) and compared with OS-ELM network. Without loss of generality, we set $N_k = 20$.

As mentioned previously, the parameter δL_j usually can be set freely. However, there are still some guidelines to follow. δL_j cannot be set too high. For example, we know that the optimal network structure for the Abalone case has approximately 25 hidden nodes in the hidden layer as determined by trial and error method, and so δL_j could not exceed 25 for CEOS-ELM. If we set δL_j to be 25 for the Abalone case, the simulation may be completed in one iteration, which is exactly the same as OS-ELM. In fact, it is better to set δL_j small in the beginning when the optimal number of hidden nodes is unknown. Additionally, we may alter the initial number of hidden nodes L_0 as well rather than set it to be 1 all the time. For a single round of simulation of CEOS-ELM, if $L_0 = 1$ and $\delta L_j = 5$, the final number of hidden nodes definitely could not be 25. (The algorithm may end up with 21 hidden nodes or 26 hidden nodes.) Hence we alter L_0 sometimes to control the possible final network structure in a single round of simulation.

Table 4.10: Incremental mode comparison on regression cases (sigmoidal additive hidden nodes, $N_k = 20, L_0 = 1$)

Dataset	Algorithm	Training Time(s)	Training RMSE	Testing RMSE	# Nodes	δL_j
Abalone	OS-ELM	0.0334	0.0754	0.0776	25	
	CEOS-ELM	0.0789	0.0758	0.0791	24.95	1
		0.0711	0.0762	0.0780	23.5	2
		0.0766	0.0758	0.0788	24.25	3
California Housing	OS-ELM	0.1475	0.1302	0.1329	50	
	CEOS-ELM	0.5305	0.1305	0.1336	49.85	1
		0.4609	0.1322	0.1369	44	2
		0.4836	0.1295	0.1345	50.8	4

For regression applications, the initial number of hidden nodes $L_0 = 1$ and the number of hidden nodes used to add into the network in each incremental step δL_j varies. Results are shown in Table 4.10 for sigmoidal hidden nodes. From the Tables, the performance of CEOS-ELM is stable in both one-by-one and group-by-group incremental modes. The resultant network structures for both of the incremental modes are comparable.

For two benchmark classification problems, the results are presented in Table 4.11 with varied L_0 and δL_j for sigmoidal hidden nodes. Comparing CEOS-ELM with OS-ELM,

Table 4.11: Incremental mode comparison on classification cases (sigmoidal additive hidden nodes, $N_k = 20$)

Dataset	Algorithm	Training Time(s)	Training Accuracy	Testing Accuracy	# Nodes	L_0	δL_j
Image Segmentation	OS-ELM	0.3934	97.34%	95.04%	200		
	CEOS-ELM	0.8047	96.37%	94.51%	145.35	1	1
		0.5117	96.15%	94.22%	139.7	1	2
		0.3695	96.20%	94.14%	143.25	1	5
		0.3727	96.51%	94.23%	155	1	10
Satellite Image	OS-ELM	2.1516	91.78%	88.65%	300		
	CEOS-ELM	3.0258	90.84%	88.23%	221.3	1	1
		1.6945	90.97%	88.30%	224.75	5	5
		1.5367	91.00%	88.32%	231.5	10	10
		1.5656	91.02%	88.29%	242	20	20

it is obvious that the networks with CEOS-ELM is more compact and CEOS-ELM is much faster than OS-ELM. (The training time stated in the table for OS-ELM does not include the time used for searching the optimal structure.)

4.2.3 Summary

One of the challenges in neural network research is how to determine network structure automatically for given application. In this section, we proposed a constructive enhancement for OS-ELM (CEOS-ELM) based on EM-ELM algorithm, which can automatically determine the network structure in SLFNs. CEOS-ELM can add random hidden nodes one-by-one or group-by-group with fixed or varying group size during the sequential learning process. We have compared the performance of CEOS-ELM with OS-ELM and the results showed that CEOS-ELM is fast and accurate with lower network complexity obtained.

Chapter 5

Random Search Enhancement of Error Minimized Extreme Learning Machine

5.1 Introduction

The good generalization performance of SLFNs relies on the correct selection of network structure. To determine the network structure, one of the useful methods is to allow the network to grow during training [78, 139]. Huang et al. proposed incremental learning algorithm, such as I-ELM, CI-ELM and EI-ELM mentioned in Section 2.2. Additionally, Feng et al [28] provided a new approach referred to as error minimized extreme learning machine (EM-ELM), which is also described in Section 2.2. However, we recently found that some of the hidden nodes in EM-ELM network may play a very minor role in the network output, but they increase the network complexity. And we realize that there is no selection of the hidden nodes before they are added to the EM-ELM network.

Inspired by EI-ELM, this chapter proposes an enhancement of EM-ELM (referred to as EEM-ELM), which introduces a selection phase based on the random search method. At each incremental learning step, several hidden nodes are randomly generated and

the hidden node that leads to the highest residual error reduction will be added into the network, and then the output weights are updated incrementally in the same way as the original EM-ELM. The empirical study shows that EEM-ELM leads to a more compact network structure.

5.2 Enhancement of EM-ELM (EEM-ELM)

In EEM-ELM, the hidden node is added to the network one-by-one. At each incremental learning step, k hidden nodes are randomly generated and the hidden node that leads to the highest residual error reduction will be added to the network, and then the output weights are updated incrementally in the same way as the original EM-ELM (refer to Section 2.2.2 for the details of EM-ELM).

Hence, the enhancement of EM-ELM algorithm can be summarized as follows:

EEM-ELM Algorithm: Given a training set $\mathfrak{X} = \{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbb{R}^n, \mathbf{t}_i \in \mathbb{R}^m, i = 1, \dots, N\}$, the activation function $g(x)$, the maximum number of hidden nodes L_{max} added to the network, number of hidden nodes k for selection in each step, and the expected learning accuracy ε .

Initialization step: Let the number of hidden nodes in the network $L = 0$, step $j = 0$ and the residual error is equal to the target $\mathbf{T}$.

Incremental learning step:

while $L_j < L_{max}$ and $E(\mathbf{H}_j) > \varepsilon$

- Let $L_{j+1} = L_j + 1$.
- **for** $i = 1 : k$

(1) Generate one random hidden node $(\mathbf{a}^i, b^i)$ and add it into the existing network. The number of hidden nodes in the network now is L_{j+1} , and the

corresponding hidden layer output matrix becomes $\mathbf{H}_{j+1}^i = [\mathbf{H}_j, \delta \mathbf{h}_j^i]$, where

$$\delta \mathbf{h}_j^i = \begin{bmatrix} G(\mathbf{a}_{L_j+1}^i, b_{L_j+1}^i, \mathbf{x}_1) & \cdots & G(\mathbf{a}_{L_j+1}^i, b_{L_j+1}^i, \mathbf{x}_N) \end{bmatrix}^T.$$

(2) Update the output weight β_i

$$\begin{aligned} \mathbf{D}_j &= \frac{\delta \mathbf{h}_j^{iT} (\mathbf{I} - \mathbf{H}_j \mathbf{H}_j^\dagger)}{\delta \mathbf{h}_j^{iT} (\mathbf{I} - \mathbf{H}_j \mathbf{H}_j^\dagger) \delta \mathbf{h}_j^i} \\ \mathbf{U}_j &= \mathbf{H}_j^\dagger - \mathbf{H}_j^\dagger \delta \mathbf{h}_j^i \mathbf{D}_j \\ \beta_i^{(j+1)} &= \mathbf{H}_{j+1}^{iT} \mathbf{T} = \begin{bmatrix} \mathbf{U}_j \\ \mathbf{D}_j \end{bmatrix} \mathbf{T} \end{aligned} \quad (5.1)$$

(3) Calculate the corresponding output error $E_i(\mathbf{H}_{j+1}^i) = \|\mathbf{H}_{j+1}^i \beta_i^{(j+1)} - \mathbf{T}\|$.

endfor

- Let $i^* = \{i | \min_{1 \leq i \leq k} \|E_i\|\}$. Hence, hidden node $(\mathbf{a}^{i^*}, b^{i^*})$ leads to the highest error reduction and is added into the network. Then set $E = E_{i^*}$, $\mathbf{a}_{L_j+1} = \mathbf{a}^{i^*}$, $b_{L_j+1} = b^{i^*}$, and $\beta^{(j+1)} = \beta_{i^*}$.
- $j = j + 1$.

endwhile

5.3 Performance evaluation of EEM-ELM

In this section, the performance of EEM-ELM is evaluated on the benchmark problems described in Table 5.1, which includes six applications [136] (Regression case: Abalone, Auto-MPG, Boston Housing, Machine CPU; Classification cases: Image Segmentation and Satellite Image). And we compare the results of EEM-ELM with the original EM-ELM algorithm.

All the evaluations were carried out in the Matlab 7.4.0 environment running on a desktop with CPU 2.66GHz and 3GB RAM. In our simulations, the input data were nor-

Table 5.1: Specification of benchmark datasets for evaluation of EEM-ELM

Datasets	# Attributes	#Classes	# Training Data	# Testing Data
Abalone	8	-	2000	2177
Auto-MPG	7	-	200	198
Boston Housing	13	-	250	256
Machine CPU	6	-	100	109
Image Segmentation	19	7	1210	1100
Satellite Image	36	6	3217	3218

malized into $[-1, 1]$, while the output data for regression cases were normalized into $[0, 1]$. Twenty trials of simulations were conducted for each applications with EEM-ELM and original EM-ELM algorithms. For each trial of simulations, the dataset of the application was divided into training and testing datasets with the number of samples indicated in Table 5.1. The activation functions used in the comparison were the sigmoidal additive activation function (Sig) $G(\mathbf{a}, b, \mathbf{x}) = (1)/(1 + \exp(-(\mathbf{a} \cdot \mathbf{x} + b)))$ and the Gaussian radian basis function (RBF) $G(\mathbf{a}, b, \mathbf{x}) = \exp(-b\|\mathbf{x} - \mathbf{a}\|^2)$. For both EEM-ELM and EM-ELM with additive hidden nodes, the input weights and biases were randomly generated from the range $[-1, 1]$. For both EEM-ELM and EM-ELM with RBF hidden nodes, the centers were randomly generated from the range $[-1, 1]$, and the impact width b was chosen from the range $(0, 0.5)$.

5.3.1 Comparison of EEM-ELM and EM-ELM with the same number of hidden nodes

In this subsection, we compare the generalization performance of EEM-ELM and EM-ELM with the same number of hidden nodes. Fig. 5.1 shows the testing root mean square error (RMSE) updating curves of EEM-ELM and EM-ELM for both Boston Housing and Auto-MPG cases. From the figure, it is obvious that EEM-ELM can

Figure 5.1: The testing RMSE updating curves of EEM-ELM and EM-ELM

achieve faster convergence rate than EM-ELM. And with the same number of hidden nodes added, EEM-ELM can obtain better generalization performance if the network is not overfitting.

5.3.2 Comparison of EEM-ELM and EM-ELM with the same expected accuracy

We compare the network structure obtained by both EEM-ELM and EM-ELM with the same stopping criterion (i.e. the same expected accuracy). Table 5.2 and Table 5.3 show the network structure and generalization performance of both EEM-ELM and EM-ELM networks for all six applications. The stopping RMSE for regression cases and the stopping accuracy for classification cases are presented in the tables, respectively. In this section, we only compare the performance between EM-ELM and EEM-ELM with $k = 10$ (denoted EEM-ELM(10)).

From two tables, they show that with comparable generalization performance, EEM-

ELM(10) always has a lower network complexity as compared to EM-ELM. In addition, the testing standard deviation (Std dev) of EEM-ELM(10) is better than or comparable with EM-ELM, which shows the stability of EEM-ELM. Fig. 5.2, Fig. 5.3 and Fig. 5.4 show the performance comparison of EEM-ELM(10) and EM-ELM with sigmoidal additive nodes on the Abalone, Auto-MPG, and Image Segmentation cases respectively. In Fig. 5.2, to achieve the same testing RMSE (0.08) on the Abalone case, EM-ELM needs 12 hidden nodes, while EEM-ELM(10) only needs 7 hidden nodes. In Fig. 5.3, to achieve the same testing RMSE (0.08) on the Auto-MPG case, EM-ELM needs 16 hidden nodes, while EEM-ELM(10) only needs 9 hidden nodes. In Fig. 5.4, to achieve the same testing accuracy (80%) on the Image Segmentation case, EM-ELM needs 10 hidden nodes, while EEM-ELM(10) only needs 7 hidden nodes.

Figure 5.2: Performance comparison of EEM-ELM(10) and EM-ELM with sigmoidal additive hidden nodes on Abalone case

5.3.3 Effect of factor k

In this subsection, the effect of the number of hidden nodes generated for selection k is studied. From Table 5.2 and Table 5.3, we compare the performance between EEM-

Figure 5.3: Performance comparison of EEM-ELM(10) and EM-ELM with sigmoidal additive hidden nodes on Auto-MPG case

ELM(10) and EEM-ELM with $k = 20$ (denoted EEM-ELM(20)). The tables show that EEM-ELM(20) achieves comparable generalization performance with even more compact network structure. And EEM-ELM(20) is stable according to the testing standard deviation. Fig. 5.5 shows the performance of EEM-ELM with different values of k with sigmoidal additive nodes on the Boston Housing case. Thus, the generalization performance of the network tends to be better when k increases. However, the training time increases with increasing k . Hence, there is a tradeoff between network performance and training time. Generally speaking, k can be set to any value greater than 3 (i.e. $k > 3$). For specific applications, k could also be selected by trial and error based on the expected network performance or the maximum training time.

5.4 Summary

In this chapter, we proposed an enhancement of EM-ELM (referred to as EEM-ELM) via the introduction of a selection phase based on a random search method. It is differ-

Figure 5.4: Performance comparison of EEM-ELM(10) and EM-ELM with sigmoidal additive hidden nodes on Image Segmentation case

ent from the original EM-ELM in that at each incremental learning step, several hidden nodes are randomly generated and the hidden node that leads to the highest reduction in residual error is added to the network. The output weights are then updated incrementally in the same way as the original EM-ELM. Indeed, EM-ELM can be seen as a special case of EEM-ELM with $k = 1$. Empirical studies show that EEM-ELM with a much smaller network can achieve better or similar performance as that of EM-ELM for the six cases we studied. EEM-ELM can be extended to the case when the hidden nodes are added group-by-group, which is more suitable for applications that require a large network structure.

Table 5.2: Performance comparison between EEM-ELM and EM-ELM with sigmoidal additive hidden nodes

Datasets (Stop RMSE)	Algorithms(k)	#Nodes	Training Time (s)	Testing RMSE	Testing Std dev
Abalone (0.08)	EM-ELM(1)	11.7	0.0148	0.0804	0.0028
	EEM-ELM(10)	7.1	0.0563	0.0802	0.0027
	EEM-ELM(20)	6.5	0.0992	0.0802	0.0023
Auto-MPG (0.07)	EM-ELM(1)	19.55	0.0109	0.0798	0.0055
	EEM-ELM(10)	14.65	0.057	0.0787	0.0056
	EEM-ELM(20)	13.1	0.0898	0.0784	0.0045
Boston Housing (0.07)	EM-ELM(1)	49.95	0.025	0.1058	0.016
	EEM-ELM(10)	30.7	0.1234	0.0978	0.0108
	EEM-ELM(20)	26.2	0.207	0.0973	0.0124
Machine CPU (0.03)	EM-ELM(1)	13.9	0.0031	0.0861	0.061
	EEM-ELM(10)	9.05	0.0313	0.0728	0.0301
	EEM-ELM(20)	8.55	0.057	0.0746	0.0438
Datasets(Stop Rate)	Algorithms(k)	#Nodes	Training Time (s)	Testing Rate	Testing Std dev
Image Segmentation (90%)	EM-ELM(1)	24.75	0.1508	88.83%	0.0121
	EEM-ELM(10)	15.45	0.8961	89.09%	0.0094
	EEM-ELM(20)	14.65	1.6742	89.25%	0.0105
Satellite Image (90%)	EM-ELM(1)	165	4.9172	87.64%	0.0065
	EEM-ELM(10)	126.6	30.4555	87.49%	0.0056
	EEM-ELM(20)	117.25	54.1523	87.23%	0.0046

Table 5.3: Performance comparison between EEM-ELM and EM-ELM with RBF hidden nodes

Datasets (Stop RMSE)	Algorithms(k)	#Nodes	Training Time (s)	Testing RMSE	Testing Std dev
Abalone (0.08)	EM-ELM(1)	13.95	0.0148	0.081	0.003
	EEM-ELM(10)	10.45	0.1008	0.0808	0.0025
	EEM-ELM(20)	9.7	0.1805	0.0799	0.0025
Auto-MPG (0.07)	EM-ELM(1)	23.85	0.0086	0.0807	0.0076
	EEM-ELM(10)	15.9	0.0625	0.0795	0.0067
	EEM-ELM(20)	14.25	0.1141	0.0794	0.0039
Boston Housing (0.07)	EM-ELM(1)	52	0.0281	0.1086	0.0126
	EEM-ELM(10)	36.35	0.1672	0.1079	0.0199
	EEM-ELM(20)	29.6	0.2695	0.1069	0.0119
Machine CPU (0.03)	EM-ELM(1)	13	0.0102	0.0788	0.0376
	EEM-ELM(10)	8.05	0.0273	0.0766	0.0498
	EEM-ELM(20)	6.2	0.0398	0.0741	0.0343
Datasets(Stop Rate)	Algorithms(k)	#Nodes	Training Time (s)	Testing Rate	Testing Std dev
Image Segmentation (90%)	EM-ELM(1)	34.6	0.2266	88.96%	0.0141
	EEM-ELM(10)	22	1.3359	88.69%	0.0129
	EEM-ELM(20)	21.35	2.5492	89.24%	0.0103
Satellite Image (90%)	EM-ELM(1)	121.5	3.468	87.83%	0.0056
	EEM-ELM(10)	87.15	19.7148	87.54%	0.0073
	EEM-ELM(20)	84.85	39.0906	87.34%	0.0077

Figure 5.5: Performance comparison of EEM-ELM with different values of factor k with sigmoidal additive hidden nodes on Boston Housing case

Chapter 6

Stepwise Forward Selection based Constructive Extreme Learning Machine for Regression

6.1 Introduction

The ELM [10, 11, 12, 13, 14] based on the generalized SLFNs with a wide variety of hidden nodes has been successfully applied in many applications. Although the network training utilizing ELM is faster than other algorithms and with improvement in generalization performance, there are still two major unresolved problems: (1) The only parameter that needs to be determined for ELM is the number of hidden nodes in the hidden layer. In former studies, this parameter is usually obtained by trial and error methods that may be tedious for some applications. How to choose the most suitable network structures for different applications is still unknown; and (2) ELM sometimes requires a large network structure (large number of hidden nodes in the hidden layer) due to the random process in the initial stage. The issue is whether the network complexity can be further reduced without affecting the generalization performance.

Pruning method is one of the useful heuristic approaches to address the problem of network architecture design. Some researchers have tried to obtain compact ELM networks based on the pruning method. P-ELM [29] and OP-ELM [30] mentioned in Section 2.2 are two examples of the pruning algorithms. Although pruning methods are straightforward and easy to understand, they are rather inefficient since most of time they are dealing with a network structure that is larger than necessary. And the situation may become worse for cases with a huge number of training samples.

Several researchers managed to handle the architecture design through incremental learning. I-ELM [12] and EM-ELM [28] are two examples of constructive improvement of ELM, which have also been described in Section 2.2. Theoretically speaking, the EM-ELM network is equivalent to the preliminary ELM with the same network structure. Therefore, they are sharing the same drawback which requires a large network structure for some applications. Besides, both I-ELM and EM-ELM are terminated when the network reaches the maximum number of hidden nodes or the expected training error given by user, but how to determine the appropriate expected training error for both of I-ELM and EM-ELM algorithms is still unknown. Nevertheless, the idea of incremental learning and recursively updating the output weights does motivate a forward hidden node selection algorithm.

In this chapter, a constructive hidden nodes selection method for ELM (CS-ELM) is proposed to handle the architecture design of ELM network. A large number of hidden nodes are firstly generated using the preliminary ELM to form the candidate reservoir. Then the hidden nodes are selected one-by-one by a stepwise forward selection method. At each step, the hidden node with an output that has the highest correlation with the current residual is selected. The hidden nodes selection stops automatically when the unbiased risk estimation criterion C_P reaches its minimum. The proposed CS-ELM is compared with I-ELM, EM-ELM and the preliminary ELM algorithm on real benchmark regression applications obtained from UCI machine learning repository [136]. The empirical study of CS-ELM shows that the proposed algorithm leads to a compact network structure automatically.

6.2 Subset model selection

The preliminary ELM network can be considered as a linear system after all the non-linearities in the hidden layer are fixed. Therefore, the selection of hidden nodes is the same as subset model selection in linear systems.

In the literature, many computational techniques have been introduced for selecting the appropriate subset model in linear systems [140, 132]. All possible regressions consider all the subset combinations, which is an exhaustive search method. However, if the input variables (i.e. the number of hidden nodes when considering ELM networks) are not small, evaluating all the possible combinations of variables (or hidden nodes) becomes computationally expensive. Hence, various methods have been proposed for evaluating only a small number of subsets, which is generally referred to as stepwise methods. Stepwise methods consists of two basic variations [132], which are called forward selection (FS) and backward elimination (BE). When a large variable pool has to be considered, FS is probably the only possible method to build the model.

Weisberg described the procedure of classic FS in [132]. In a linear system, a collection of possible variables are given. We first select the one having the largest absolute correlation with the response (may be denoted by $\mathbf{x}_1$), and perform a simple linear regression using LS estimate. Hence, the residual can be obtained by the difference between the target response and the LS estimate, which is orthogonal to $\mathbf{x}_1$. The remaining variable that has the highest absolute correlation with the current residual is selected as the second variable of the model. Then the LS estimate of the output with the current model and residual are updated. After k steps, there are k variables selected and presented in the model. Classic FS is a greedy algorithm that uses LS estimate to update the output response, which may miss some useful variables.

Another widely implemented technique used in FS is orthogonal least squares (OLS) algorithm and its variants [114, 117, 118, 119], which is based on the modified Gram-Schmidt (MGS) orthogonalisation. OLS selects a suitable set of variables to form the

subset model from a large set of candidates. At each step, the net decrease in the residual error is maximized. The key advantage of the algorithm is that it can explicitly identify the net contribution of the newly added node without solving the whole least-squares problem, which significantly reduces the computational complexity. However, OLS cannot guarantee an optimal solution because it is greedy and on the basis of a local optimization [141].

Unlike OLS, the proposed algorithm in this chapter is based on the classic FS method without requiring matrix decomposition and transformation. And it is a less greedy forward stepwise selection method.

6.3 Proposed CS-ELM

In this section, we propose a constructive method for hidden nodes selection for ELM, and we call it CS-ELM. The main idea of CS-ELM is to identify the significance of each hidden node and select the optimal subset of hidden nodes when the stopping criterion reaches its minimum. The hidden node is selected one-by-one from the candidate reservoir.

6.3.1 Candidate reservoir

The preliminary ELM is used to build the candidate reservoir with a relatively large number of hidden nodes (denoted by $L_{\max}$). Consider N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i) \in \mathbb{R}^n \times \mathbb{R}^m$. The output of SLFN with $L_{\max}$ hidden nodes can be represented by

$$f_{L_{\max}}(\mathbf{x}_j) = \sum_{i=1}^{L_{\max}} \beta_i G(\mathbf{a}_i, b_i, \mathbf{x}_j) \quad j = 1, \dots, N. \quad (6.1)$$

where $\mathbf{a}_i$ and b_i are the hidden node parameters that are randomly generated, β_i is the output weight, and $G(\mathbf{a}_i, b_i, \mathbf{x})$ denotes the output of the i th hidden node with respect to the input $\mathbf{x}$.

For real applications, in most cases, there are a large number of samples in the dataset. We assume $L_{\max} < N$, which means the network approximates the given N samples with a small training error [13]. After the nonlinearities in the SLFN (i.e. $\mathbf{a}_i$ and b_i) are fixed, the mathematical model can be represented by the following matrix form:

$$\mathbf{T} = \mathbf{H}\beta + \mathbf{E} \quad (6.2)$$

where

$$\mathbf{T} = \begin{bmatrix} \mathbf{t}_1 & \cdots & \mathbf{t}_N \end{bmatrix}_{N \times m}^T \quad (6.3)$$

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}_1 & \cdots & \mathbf{h}_{L_{\max}} \end{bmatrix}_{N \times L_{\max}} \quad (6.4)$$

$$\mathbf{h}_i = \begin{bmatrix} G(\mathbf{a}_i, b_i, \mathbf{x}_1) & \cdots & G(\mathbf{a}_i, b_i, \mathbf{x}_N) \end{bmatrix}^T, \quad 1 \leq i \leq L_{\max} \quad (6.5)$$

$$\beta = \begin{bmatrix} \beta_1 & \cdots & \beta_{L_{\max}} \end{bmatrix}_{L_{\max} \times m}^T \quad (6.6)$$

$$\mathbf{E} = \begin{bmatrix} \mathbf{e}_1 & \cdots & \mathbf{e}_N \end{bmatrix}_{N \times m}^T \quad (6.7)$$

$\mathbf{h}_i$ is the i -th hidden node's output vector with respect to inputs $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$, where $1 \leq i \leq L_{\max}$. Thus, $\mathbf{h}_i$ can be regarded as variable vector in the linear regression model. β is the least-squares solution of the linear system. And the residuals $\mathbf{e}_1 \dots \mathbf{e}_N$ are assumed to be independent random variables with mean zero and unknown common variance σ^2 . Without loss of generality, the single output case is considered in this chapter for notational simplicity, i.e. $m = 1$. Therefore, the equations (6.3), (6.6), and (6.7) become:

$$\mathbf{t} = \begin{bmatrix} t_1 & \dots & t_N \end{bmatrix}_{N \times 1}^T, \quad \beta = \begin{bmatrix} \beta_1 & \dots & \beta_{L_{\max}} \end{bmatrix}_{L_{\max} \times 1}^T, \quad \mathbf{e} = \begin{bmatrix} e_1 & \dots & e_N \end{bmatrix}_{N \times 1}^T \quad (6.8)$$

Hence, the hidden nodes selection is simply to measure the significance of each hidden node output vector $\mathbf{h}_i$, and selecting the $\mathbf{h}_i$ with high significance from reservoir $\mathbf{H}$.

6.3.2 Constructive hidden nodes selection

The selection of hidden nodes is regarded as a problem of subset model selection or variable selection in linear regression. In this section, the hidden nodes output vectors are ranked and selected one-by-one by a stepwise forward selection method.

Before hidden nodes selection, we normalize the hidden nodes output matrix $\mathbf{H}$ to a standard normal distribution (i.e. $\mathbf{N}(0, 1)$) and the target vector $\mathbf{t}$ to zero mean:

$$\mathbf{h}_{i,norm} = \frac{\mathbf{h}_i - \text{mean}(\mathbf{h}_i)}{\text{var}(\mathbf{h}_i)}, \quad i = 1, 2, \dots, L_{\max}, \quad \mathbf{t}_{norm} = \mathbf{t} - \text{mean}(\mathbf{t}) \quad (6.9)$$

The proposed constructive algorithm adds sequentially active hidden node output vector to the model

$$\mathbf{y}^{(k)} = \mathbf{H}_P \beta^{(k)} \quad (6.10)$$

where P is a set that consists of all the indices of selected hidden node output vectors by the first k steps and $\mathbf{H}_P$ contains hidden node output vectors selected from the candidate reservoir by the first k steps (i.e. $\mathbf{H}_P = \begin{bmatrix} \mathbf{h}_i & \cdots & \mathbf{h}_j \end{bmatrix}$, $i, \dots, j \in P$). In each step, a new hidden node output vector is added to the matrix $\mathbf{H}_P$, and a new row is added to β . Therefore, the number of elements in set P is equal to k .

When $k = 0$, the $\mathbf{y}^{(0)}$ and $\beta^{(0)}$ are initialized to zero. And P is initially $\emptyset$ and $\mathbf{H}_P = \mathbf{0}$. Then in the first step when $k = 1$, the cumulative correlation between the i th hidden node output vector $\mathbf{h}_{i,norm}$ and the current residual is calculated as

$$c_i^{(1)} = (\mathbf{t}_{norm} - \mathbf{y}^{(0)})^T \mathbf{h}_{i,norm}, \quad i = 1, 2, \dots, L_{\max} \quad (6.11)$$

The hidden node output vector that has the highest cumulative correlation with the current residual is selected and added into the matrix $\mathbf{H}_P$. The maximum correlation is denoted by $c_{\max}^{(1)}$ and the index of selected hidden node output vector is added into the set P .

$$c_{\max}^{(1)} = \max_i \{|c_i^{(1)}|\}, \quad P = \{i | c_i^{(1)} = c_{\max}^{(1)}\} \quad (6.12)$$

Hence, the LS estimate of output response (mentioned in Section 2.1.5 equation (2.22)) with one hidden node is given by

$$\bar{\mathbf{y}}^{(1)} = \mathbf{H}_P \beta = \mathbf{H}_P \mathbf{H}_P^\dagger \mathbf{t}_{norm} = \mathbf{H}_P (\mathbf{H}_P^T \mathbf{H}_P)^{-1} \mathbf{H}_P^T \mathbf{t}_{norm} \quad (6.13)$$

where $\mathbf{H}_P^\dagger = (\mathbf{H}_P^T \mathbf{H}_P)^{-1} \mathbf{H}_P^T$ is called the left pseudoinverse of $\mathbf{H}_P$ from the fact that $\mathbf{H}_P^\dagger \mathbf{H}_P = \mathbf{I}$ when $\mathbf{H}_P^T \mathbf{H}_P$ is nonsingular.

In the classic FS algorithm described in Section 6.2, the hidden node is selected based on equation (6.12) and the LS estimate is used to update $\mathbf{y}^{(1)}$:

$$\mathbf{y}^{(1)} = \bar{\mathbf{y}}^{(1)} \quad (6.14)$$

The classic FS can be greedy and misses some useful hidden nodes. Hence, in the proposed algorithm, a less greedy algorithm is applied by estimating $\mathbf{y}^{(1)}$ in the direction of $\bar{\mathbf{y}}^{(1)} - \mathbf{y}^{(0)}$ with a step size of $\gamma^{(1)}$ until some other hidden node vector has as much correlation as the current residual, i.e.

$$\mathbf{y}^{(1)} = \mathbf{y}^{(0)} + \gamma^{(1)}(\bar{\mathbf{y}}^{(1)} - \mathbf{y}^{(0)}) \quad (6.15)$$

where $\gamma^{(1)}$ is the step size to find a new hidden node that satisfies the maximum correlation in the direction of $\bar{\mathbf{y}}^{(1)} - \mathbf{y}^{(0)}$. Thus, the problem is transformed to one of solving for suitable $\gamma^{(1)}$.

From equation (6.13), we obtain

$$\mathbf{H}_P^T(\mathbf{t}_{norm} - \mathbf{y}^{(0)}) = \mathbf{H}_P^T(\bar{\mathbf{y}}^{(1)} - \mathbf{y}^{(0)}) \quad (6.16)$$

Or

$$(\mathbf{t}_{norm} - \mathbf{y}^{(0)})^T \mathbf{h}_{i,norm} = (\bar{\mathbf{y}}^{(1)} - \mathbf{y}^{(0)})^T \mathbf{h}_{i,norm}, \quad i \in P \quad (6.17)$$

With (6.11), (6.15) and (6.17), the cumulative correlations $c_i^{(2)}(\gamma)$ with $k = 2$ can be addressed by

$$\begin{aligned} c_i^{(2)}(\gamma) &= (\mathbf{t}_{norm} - \mathbf{y}^{(0)})^T \mathbf{h}_{i,norm} - \gamma^{(1)}(\bar{\mathbf{y}}^{(1)} - \mathbf{y}^{(0)})^T \mathbf{h}_{i,norm} \\ &= \begin{cases} |1 - \gamma^{(1)}| c_{max}^{(1)}, & i \in P \quad (a) \\ |c_i^{(1)} - \gamma^{(1)} a_i^{(1)}|, & i \in P^c \quad (b) \end{cases} \end{aligned} \quad (6.18)$$

where $a_i^{(1)} = (\bar{\mathbf{y}}^{(1)} - \mathbf{y}^{(0)})^T \mathbf{h}_{i,norm}$, and P^c means the complement of set P , which contains indices of hidden node vectors that have not been selected. When $k = 2$, a new hidden node with index $i \in P^c$ enters into the model when equations (a) and (b) in (6.18) are

equal. Hence the step size γ can be found by $\gamma^{(1)} = \min_{i \in P^c}^+ \left\{ \frac{c_{\max}^{(1)} - c_i^{(1)}}{c_{\max}^{(1)} - a_i^{(1)}} , \frac{c_{\max}^{(1)} + c_i^{(1)}}{c_{\max}^{(1)} + a_i^{(1)}} \right\}$, where " $\min_{i \in P^c}^+$ " refers to the minimum that is taken over the positive components within each possible i .

Generalizing the arguments above, the hidden node selection in the k th step is given as follows:

$$c_i^{(k)} = (\mathbf{t}_{\text{norm}} - \mathbf{y}^{(k-1)})^T \mathbf{h}_{i,\text{norm}} \quad (6.19)$$

$$c_{\max}^{(k)} = \max_i \{|c_i^{(k)}|\}, \quad P = \{i | c_i^{(k)} = c_{\max}^{(k)}\} \quad (6.20)$$

The cumulative correlation is calculated using equation (6.19), and the hidden node output vector most correlated to the current residual is identified and added into set P . The linear model response is updated as follows:

$$\mathbf{y}^{(k)} = \mathbf{y}^{(k-1)} + \gamma^{(k)} (\bar{\mathbf{y}}^{(k)} - \mathbf{y}^{(k-1)}) \quad (6.21)$$

where

$$\bar{\mathbf{y}}^{(k)} = \mathbf{H}_P (\mathbf{H}_P^T \mathbf{H}_P)^{-1} \mathbf{H}_P^T \mathbf{t}_{\text{norm}} \quad (6.22)$$

$$\text{and } \gamma^{(k)} = \min_{i \in P^c}^+ \left\{ \frac{c_{\max}^{(k)} - c_i^{(k)}}{c_{\max}^{(k)} - a_i^{(k)}} , \frac{c_{\max}^{(k)} + c_i^{(k)}}{c_{\max}^{(k)} + a_i^{(k)}} \right\}, \text{ with } a_i^{(k)} = (\bar{\mathbf{y}}^{(k)} - \mathbf{y}^{(k-1)})^T \mathbf{h}_{i,\text{norm}}.$$

The main idea of this stepwise forward hidden node selection method is that in each step, the hidden node output vector that has the largest correlation with the current residual is selected. Then the algorithm is moving in the direction that is equiangular among all selected vectors until a new hidden node output vector has as much correlation with the current residual. The purpose for moving along equiangular direction is to ensure a common correlation between the selected vectors and the current residual. The process continues until all the vectors have entered the model.

Remark 6.1 *The proposed hidden nodes selection method in nonlinear system is sim-*

ilar to the multi-response sparse regression (MRSR) algorithm in linear system introduced in [66]. MRSR is based on the alternative algorithm of least angle regression (LAR [142]) proposed in [143]. Hence, the validity of least angle regression algorithm can be applied here.

6.3.3 Updating $\bar{\mathbf{y}}$ recursively

From the constructive hidden nodes selection method described above, the hidden node is selected from the candidate reservoir one-by-one. Thus, in each iteration of selection, $\mathbf{H}_P$ (the collection of selected hidden node vectors) is enlarged by adding one more column. As the matrix $\mathbf{H}_P$ becomes bigger, the LS estimation $\bar{\mathbf{y}}$ is computationally more expensive. In order to reduce the computational complexity of calculating $\bar{\mathbf{y}}$, the method proposed in [28] is implemented in the proposed CS-ELM.

In step $(k-1)$, collect all the selected hidden node output vectors to form $\mathbf{H}_P^{(k-1)}$. And assume that $\mathbf{h}_{j,norm}$ satisfies the maximum correlation criterion and is added to the model in the k th step. Hence, $\mathbf{H}_P^{(k)} = [\mathbf{H}_P^{(k-1)} \quad \mathbf{h}_{j,norm}]$. Following the mathematical model described in [28], and $\mathbf{H}_P^\dagger = (\mathbf{H}_P^T \mathbf{H}_P)^{-1} \mathbf{H}_P^T$, we have

$$\mathbf{D}^{(k-1)} = \frac{\mathbf{h}_{j,norm}^T (\mathbf{I} - \mathbf{H}_P^{(k-1)} \mathbf{H}_P^{\dagger(k-1)})}{\mathbf{h}_{j,norm}^T (\mathbf{I} - \mathbf{H}_P^{(k-1)} \mathbf{H}_P^{\dagger(k-1)}) \mathbf{h}_{j,norm}} \quad (6.23)$$

$$\mathbf{U}^{(k-1)} = \mathbf{H}_P^{\dagger(k-1)} (\mathbf{I} - \mathbf{h}_{j,norm} \mathbf{D}^{(k-1)}) \quad (6.24)$$

$$\bar{\mathbf{y}}^{(k)} = \mathbf{H}_P^{(k)} \mathbf{H}_P^{\dagger(k)} \mathbf{t}_{norm} = \mathbf{H}_P^{(k)} \begin{bmatrix} \mathbf{U}^{(k-1)} \\ \mathbf{D}^{(k-1)} \end{bmatrix} \mathbf{t}_{norm} \quad (6.25)$$

Equations (6.23)-(6.25) provide a recursive method to calculate $\mathbf{H}_P^\dagger$, which significantly reduces the computational complexity of calculating $\bar{\mathbf{y}}$. Equation (6.25) is used to replace equation (6.22) in our implementation of CS-ELM.

6.3.4 Stopping criterion

To evaluate the performance of sub-model and to look for the most suitable model, a stopping criterion is introduced in the algorithm to terminate the hidden nodes selection by using the validation data that is drawn from the training data.

The C_p criterion proposed by Mallows [109] based on the unbiased risk estimation is used in CS-ELM. Assume that p hidden nodes are selected from the candidate reservoir in the current model. The C_p criterion is specified as

$$C_p = \frac{RSS_p}{\sigma^2} - n + 2p \quad (6.26)$$

where $RSS_p = \sum_{i=1}^n (t_i - y_i)^2$ stands for the residual sum of squares of the current model for validation data; σ^2 is the variance of $\mathbf{e}$ that was mentioned in the beginning of Section 6.3.1; n denotes the number of observations in the validation dataset. For real regression cases, σ^2 is unknown and has to be estimated. In CS-ELM, we estimate σ^2 by $MSE_{full} = mse(t_i - y_i^{(L_{max})})$ [144], which is the mean squared error from the full model that includes all the hidden nodes in the candidate reservoir. MSE_{full} is determined by using the data in the training dataset, which does not belong to the validation dataset. Hence, the C_p criterion becomes

$$C_p = \frac{RSS_p}{MSE_{full}} - n + 2p \quad (6.27)$$

Therefore, the optimal network structure is determined when C_p reaches its minimum.

6.3.5 Summary of CS-ELM algorithm

The CS-ELM algorithm can be summarized in three phases, namely an initialization phase, a hidden nodes selection phase (or simply selection phase) and a training and testing phase. In the initialization phase, the training set is split equally into two non-overlapping subsets for learning and validation. The SLFN is built using preliminary

ELM algorithm with the maximum number of hidden nodes that is defined by the users, and then it is trained by the learning set split from the training set. The hidden layer output matrix that consists of a large number of hidden nodes (i.e. the column vectors of hidden layer output matrix) can be regarded as a candidate reservoir. The selection of the useful hidden nodes is conducted in phase 2. In each step of selection phase, the hidden node output vector that has the largest correlation with the current residual is selected. Then the algorithm is moving in the direction that is equiangular among all selected vectors until a new hidden node output vector has as much correlation with the current residual. The process continues until all the hidden node output vectors have been ranked. Then, the SLFN is trained using the validation set with the number of hidden nodes from one to the maximum taken from the ordered hidden layer output matrix. C_P is calculated for all the subsets and the optimal number of hidden nodes is determined when C_P reaches its minimum value. In phase 3, the SLFN is rebuilt by the hidden nodes selected. And the network is retrained using the whole training set and evaluated using the testing set. The algorithm is described mathematically as follows:

CS-ELM Algorithm: Given a set of training data $\{(\mathbf{x}_i, t_i)\}_{i=1}^{2n}$, the number of hidden nodes forming the candidate reservoir for selection $L_{\max}$, and the activation function g .

Phase 1 Initialization phase:

Split the training set equally into two non-overlapping subsets for learning and validation. Build the SLFN using the preliminary ELM algorithm by randomly generating the hidden nodes $(\mathbf{a}_i, b_i)_{i=1}^{L_{\max}}$. Train the SLFN using the data for learning with the number of observations n . And the model in the matrix form is presented as $\mathbf{t} = \mathbf{H}\beta + \mathbf{e}$, where $\mathbf{H} = \begin{bmatrix} \mathbf{h}_1 & \cdots & \mathbf{h}_{L_{\max}} \end{bmatrix}$.

Phase 2 Selection phase:

When $k = 0$, set $L_0 = 0$, $\mathbf{y}^{(0)} = \mathbf{0}$, $\beta^{(0)} = \mathbf{0}$, $P = \emptyset$ and $\mathbf{H}_P = \mathbf{0}$. Normalize $\mathbf{H}$ to standard normal distribution and $\mathbf{t}$ to zero mean.

$$\mathbf{h}_{i,norm} = \frac{\mathbf{h}_i - \text{mean}(\mathbf{h}_i)}{\text{var}(\mathbf{h}_i)}, \quad i = 1, 2, \dots, L_{\max}, \quad \mathbf{t}_{norm} = \mathbf{t} - \text{mean}(\mathbf{t}) \quad (6.28)$$

while $L_k \leq L_{\max}$:

(1) $k = k + 1$.

(2) Compute $c_i^{(k)} = (\mathbf{t}_{norm} - \mathbf{y}^{(k-1)})^T \mathbf{h}_{i,norm}$ and set $c_{\max}^{(k)} = \max_i \{ |c_i^{(k)}| \}$.

(3) Let $P = \{i | c_i^{(k)} = c_{\max}^{(k)}\}$, and set $L_k = L_{k-1} + 1$.

(4) Set $\mathbf{H}_P^{(k)} = \begin{bmatrix} \dots & \mathbf{h}_{j,norm} & \dots \end{bmatrix}_{j \in P}$ for calculating $\bar{\mathbf{y}}^{(k)} = \mathbf{H}_P^{(k)} \begin{bmatrix} \mathbf{U}^{(k-1)} \\ \mathbf{D}^{(k-1)} \end{bmatrix} \mathbf{t}_{norm}$.

Let $\mathbf{h}_{j,norm}$ denote the hidden node selected in this iteration, then $\mathbf{D}^{(k-1)} = \frac{\mathbf{h}_{j,norm}^T (\mathbf{I} - \mathbf{H}_P^{(k-1)} \mathbf{H}_P^{\dagger(k-1)})}{\mathbf{h}_{j,norm}^T (\mathbf{I} - \mathbf{H}_P^{(k-1)} \mathbf{H}_P^{\dagger(k-1)}) \mathbf{h}_{j,norm}}$ and $\mathbf{U}^{(k-1)} = \mathbf{H}_P^{\dagger(k-1)} (\mathbf{I} - \mathbf{h}_{j,norm} \mathbf{D}^{(k-1)})$.

(5) Set

$$\mathbf{y}^{(k)} = \mathbf{y}^{(k-1)} + \gamma^{(k)} (\bar{\mathbf{y}}^{(k)} - \mathbf{y}^{(k-1)})$$

where, if P^c is not empty, $\gamma^{(k)} = \min_{i \in P^c}^+ \left\{ \frac{c_{\max}^{(k)} - c_i^{(k)}}{c_{\max}^{(k)} - a_i^{(k)}}, \frac{c_{\max}^{(k)} + c_i^{(k)}}{c_{\max}^{(k)} + a_i^{(k)}} \right\}$. Otherwise, set $\gamma^{(k)} = 1$. And $a_i^{(k)} = (\bar{\mathbf{y}}^{(k)} - \mathbf{y}^{(k-1)})^T \mathbf{h}_{i,norm}$.

endwhile

$$MSE_{full} = \text{mse}(\mathbf{t}_{norm} - \mathbf{y}^{(L_{\max})})$$

for $p = 1 : L_{\max}$, where p is the number of hidden nodes selected:

Train the SLFN using the validation data (normalize the hidden layer output matrix and target using the method stated above) and calculate $C_P(p) = \frac{RSS_p}{MSE_{full}} - n + 2p$, where n is the number of observations in the validation set.

endfor

Let $p^* = \{p | \min_{1 \leq p \leq L_{\max}} \|C_P\|\}$.

Phase 3 Training and testing phase:

After the selection phase, assume the number of hidden nodes determined by phase 2 is p^* . Build the SLFN with hidden nodes selected (with original unnormalized hidden node output vectors) and retrain the network with the whole training set including learning and validation data. And then evaluate the performance of the SLFN using the testing dataset. The model in the matrix form is presented as $\mathbf{t} = \mathbf{H}\boldsymbol{\beta} + \mathbf{e}$, where $\mathbf{H} = \begin{bmatrix} \mathbf{h}_1 & \cdots & \mathbf{h}_{p^*} \end{bmatrix}$.

Remark 6.2 *The fast construction algorithm (FCA) proposed in [145] is a constructive hidden node selection method for ELM based on OLS. Both FCA and the proposed CS-ELM are able to select the significant hidden nodes one-by-one from a pool of candidates. As mentioned in Section 6.2, OLS may not be efficient for the non-orthogonal basis [146]. Therefore, FCA may not achieve the most compact network structure. By modifying the classic FS algorithm, the proposed CS-ELM is less greedy and without any matrix decompositions.*

6.4 Performance evaluation of CS-ELM

The performance of CS-ELM is evaluated on the benchmark problems described in Table 6.1, which include ten regression applications (Servo, Auto-MPG, Boston housing, Abalone, Delta ailerons, Ailerons, Bank domains, Delta elevators, California Housing and House census(8L)) [136]. It is compared with the original ELM [13], I-ELM [12], and EM-ELM [28].

All the evaluations were carried out in the Matlab 7.4.0 environment running on a desktop with CPU 2.66GHz and 3GB RAM. For all the regression applications, the input data has been normalized into $[-1, 1]$, while the output data has been normalized into $[0, 1]$. Twenty trials of simulations were conducted for each applications with ELM, I-ELM, EM-ELM and CS-ELM algorithms. For each trial of simulations, the dataset

Table 6.1: Specification of benchmark datasets for evaluation of CS-ELM

Datasets	# Attributes	# Observations	# Training Data	# Testing Data
Servo	4	167	80	87
Auto-MPG	7	398	200	198
Boston housing	13	506	250	256
Abalone	8	4177	2000	2177
Delta ailerons	5	7129	3000	4129
Ailerons	40	7154	3000	4154
Bank domains	8	8192	4500	3692
Delta elevators	6	9517	4000	5517
California housing	8	20640	8000	12640
House census(8L)	8	22784	10000	12784

of the application was divided into training and testing datasets with the number of samples indicated in Table 6.1. The process was random based, which means, take Abalone for example, we randomly selected 2000 samples out of 4177 to form the training dataset, and the rest of the samples is in the testing dataset. The activation function used in the comparison was the sigmoidal additive activation function (Sig) $G(\mathbf{a}, b, \mathbf{x}) = (1)/(1 + \exp(-(\mathbf{a} \cdot \mathbf{x} + b)))$. For SLFNs with ELM, I-ELM, EM-ELM, and CS-ELM algorithms and additive hidden nodes, the input weights and biases were randomly generated from the range $[-1, 1]$.

6.4.1 Parameter selection

For the preliminary ELM, the number of hidden nodes in the hidden layer L was searched using the trial and error method with grid search technique. L was set to be 5, 10, 15, 20, ..., 400. Twenty trials of simulations were conducted for each network with L hidden nodes and the network structure with the highest generalization was selected. Values of L selected for each application is presented in Table 6.2.

Both the I-ELM and EM-ELM algorithms have two parameters that need to be considered: the maximum number of hidden nodes $L_{\max}$ and the expected training rate ϵ . In the evaluations, we aimed to obtain the network structure when the training rate reached

Figure 6.1: Parameter selection for Delta elevators case, Ailerons case and Bank domains case - testing RMSE

the specific ε without the constraint of $L_{\max}$. Therefore, in our experiments, the value of $L_{\max}$ was set to be a large value of 2000, and the value of ε was set according to the training rate obtained by the preliminary ELM for each regression case.

For the proposed CS-ELM, only the number of hidden nodes that forms the candidate reservoir $L_{\max}$ needs to be set by users. As mentioned in Section 6.3.4, the variance of the residual σ^2 can be estimated using MSE_{full} , the mean squared error from the full model that include all the hidden nodes in the candidate reservoir. Hence, intuitively, $L_{\max}$ could not be set randomly because different values of $L_{\max}$ form different full model, which create variations in estimating the σ^2 . We conducted experiments on the Delta elevators case, Ailerons case and Bank domains case to investigate the influence on the performance of CS-ELM by the selection of $L_{\max}$. $L_{\max}$ was evaluated for the values of 5, 10, 15, 20, ..., 100. For each value of $L_{\max}$, the simulation was repeated for twenty times.

The performance of CS-ELM with different values of $L_{\max}$ is presented in Fig. 6.1. If

Figure 6.2: Parameter selection for Delta elevators case, Ailerons case and Bank domains case - selected hidden nodes

$L_{\max}$ is set too small (less than 30), the performance of CS-ELM would not be good. This is probably due to two reasons: one is that the network is underfitting because there is not enough hidden nodes; another reason is that MSE_{full} cannot estimate the real variance of the residual properly. As the value of $L_{\max}$ increases, the performance of CS-ELM becomes stable. As observed from Fig. 6.1, the generalization of SLFN with CS-ELM is fluctuating by less than 0.005 when $L_{\max}$ is in the range of [30,100] for all three cases.

With different values of $L_{\max}$, the number of hidden nodes selected is presented in Fig. 6.2. For cases of Delta elevators and Ailerons, the number of hidden nodes selected by CS-ELM is stable when the performance of CS-ELM is stable. However, for the Bank domain case, the number of hidden nodes selected is increasing when the value of $L_{\max}$ is increasing. Because the generalization performance of CS-ELM for the Bank domain case is stable when parameter $L_{\max}$ is above some small value (say 50 as observed from Fig. 6.1), it is better to choose a small value of $L_{\max}$ that leads a compact network structure.

Therefore, it seems that $L_{\max}$ can be freely selected if it is greater than some threshold. In our simulations, $L_{\max}$ was set to be 50, 100, 200, 300 and 400. If the simulation results were comparable, we prefer to have a small $L_{\max}$ and it reduces the computational complexity in phase 2 of the proposed CS-ELM. The $L_{\max}$ selected for all the regression cases are presented in Table 6.2.

Table 6.2: Parameters setting for benchmark regression cases

Datasets	CS-ELM	EM-ELM		I-ELM		ELM
	$L_{\max}$	$L_{\max}$	ϵ	$L_{\max}$	ϵ	L
Servo	50	2000	0.07	2000	0.07	30
Auto-MPG	50	2000	0.07	2000	0.07	25
Boston housing	50	2000	0.07	2000	0.07	50
Abalone	50	2000	0.08	2000	0.08	25
Delta ailerons	50	2000	0.04	2000	0.04	45
Delta elevators	50	2000	0.053	2000	0.053	80
California housing	50	2000	0.13	2000	0.13	50
Ailerons	100	2000	0.05	2000	0.05	65
Bank domains	200	2000	0.045	2000	0.045	330
House census(8L)	300	2000	0.065	2000	0.065	160

6.4.2 Performance evaluation of CS-ELM

Effect of stopping criterion C_P in CS-ELM algorithm

The effect of stopping criterion C_P is illustrated in Fig. 6.3, Fig. 6.4 and Fig. 6.5 on the Abalone, Ailerons and Delta elevators cases respectively. The simulations were conducted for twenty times and the average values of C_P were recorded and shown in the figures. We concentrate on the range when the number of hidden nodes selected is from 10 to 50 for the Abalone case. From Fig. 6.3, it is obvious that when $p = 20$, the C_P has its minimum value. The C_P curve reflects the generalization capabilities of the model. And the suitable number of hidden nodes is determined when C_P reaches its minimum. Moreover, $p = 33$ for the Ailerons case from Fig. 6.4, and $p = 11$ for the Delta elevators case from Fig. 6.5.

Then we consider the number of hidden nodes selected by the criterion C_P . Twenty rounds of simulations were conducted and the average number of hidden nodes was recorded on the Abalone case. The evaluation was repeated for twenty times and the result is shown in Fig. 6.6. From Fig. 6.6, the number of hidden nodes selected fluctuates by less than 4 nodes, which implies that the proposed CS-ELM can achieve a stable network structure. Similar conclusions could be drawn from Fig. 6.7 for the Ailerons case and Fig. 6.8 for the Delta elevators case.

Figure 6.3: Effect of C_P on Abalone case, number of hidden nodes selected $p \in [10, 50]$

Performance comparison

In this section, we evaluate and compare the performance of the proposed CS-ELM with EM-ELM, I-ELM and the preliminary ELM. In phase 2 of CS-ELM, the hidden node was selected one-by-one. And the hidden node of I-ELM was also added to the SLFN one-by-one. For a fair comparison, the EM-ELM was implemented with the one-by-one incremental mode.

Simulation results for ten regression cases are presented in Table 6.3. Ten regression

Figure 6.4: Effect of C_P on Ailerons case, number of hidden nodes selected $p \in [10, 70]$

cases are separated into two groups according to the selection of $L_{\max}$ of the CS-ELM algorithm. Regression cases with threshold of $L_{\max}$ that is less than 50 are in Group one (including Servo, Auto-MPG, Boston housing, Abalone, Delta ailerons, Delta elevators and California housing), while the remaining ones are in Group two (including Ailerons, Bank domains and House census(8L)). CS-ELM tends to select the optimal subset of hidden nodes from the hidden nodes reservoir. However, all the hidden nodes in the reservoir are involved in the selection phase, which means that with a higher $L_{\max}$, a longer time is spent on training the network. Hence, the applications in Group two may require more time in training the network as compared to the ones in Group one. We compare the training time spent by the proposed CS-ELM and the rest of algorithms, it seems that CS-ELM may spend much more time especially for cases in Group two. However, the training time presented in the table cannot be compared directly. For the preliminary ELM, the training time indicated in the table is for a fixed network that does not include the selection process. While for EM-ELM and I-ELM, the expected accuracy is set with the reference given by the preliminary ELM. When we do not have any reference value for the expected accuracy, this parameter may be determined by trial

Figure 6.5: Effect of C_p on Delta elevators case, number of hidden nodes selected $p \in [5, 35]$

and error, which is time consuming.

From Table 6.3, it is clear that the proposed CS-ELM achieves comparable generalization performance as the preliminary ELM and CS-ELM is quite stable as indicated by the standard deviation of the generalization.

I-ELM achieves comparable performance with the largest network structure for all the regression cases. Comparing CS-ELM, EM-ELM with the preliminary ELM, the network obtained by using CS-ELM or EM-ELM is always more compact than the trial and error method except for the Ailerons case. The network with EM-ELM needs more nodes to achieve the expected training accuracy as compared to that of ELM. However, our CS-ELM network performs well with a network structure much smaller than the ELM network. And it is obvious that a trial and error method could not guarantee the optimal network structure.

The key difference between EM-ELM and CS-ELM is that the hidden node is added one-by-one without any selection in the EM-ELM algorithm, while in our CS-ELM the

Figure 6.6: Average number of hidden nodes selected on Abalone case

hidden node is selected based on its significance. From Table 6.3, CS-ELM performs much better than EM-ELM on applications like Servo, Auto-MPG, Boston housing, Delta elevators, California housing, Ailerons, and Bank domains. And, for the remaining cases, the performances of CS-ELM and EM-ELM are comparable.

For a final comparison of CS-ELM, EM-ELM and the preliminary ELM, an experiment has been conducted for the Delta elevators case. For the ELM, the number of hidden nodes L was set to be 5, 10, 15, ..., 100. For EM-ELM, the performance of network was evaluated each time a hidden node was added into the network as well as the proposed CS-ELM. The learning evolution for the Delta elevators case is presented in Fig. 6.9, which starts from $L = 5$. Simulations for ELM was conducted for twenty times. While for EM-ELM and CS-ELM, the simulation was running for one time, since the number of hidden nodes selected in network with EM-ELM and CS-ELM may be slightly different in different rounds of simulations. The network structures selected are 80, 39, and 10 for ELM, EM-ELM, and CS-ELM networks, respectively. From Fig. 6.9, CS-ELM converges faster than EM-ELM and ELM, which is reasonable because the hidden nodes were ranked by significance in the CS-ELM. Although EM-ELM provides a so-

Figure 6.7: Average number of hidden nodes selected on Ailerons case

lution of network structure, the structure selected depends on the expected training rate and it may not be optimal. The optimal structure seen from the Figure is 10, which can be obtained by using CS-ELM.

6.5 Summary

In this chapter, the CS-ELM algorithm is proposed to address the problem of network architecture design of ELM by applying a constructive hidden nodes selection method. A performance comparison of CS-ELM with EM-ELM, I-ELM and ELM is evaluated on ten real world benchmark regression applications. Specifically, the proposed algorithm achieves a compact network structure with competitive generalization performance when compared to the preliminary ELM. In addition, CS-ELM selects the hidden nodes based on the correlation between hidden nodes and the current residual, while EM-ELM adds the randomly generated hidden nodes without any selection. Empirical studies show that CS-ELM with a much smaller network can achieve better or

Figure 6.8: Average number of hidden nodes selected on Delta elevators case

Figure 6.9: Learning evolution for Delta elevators case

similar performance as that of EM-ELM for the ten cases we studied.

Table 6.3: Performance comparison

Datasets	Algorithms	# Hidden nodes	Training time (s)	Training RMSE	Testing RMSE	
					Mean	Dev.
Servo	ELM	30	0.0001	0.0696	0.1331	0.0454
	I-ELM	2000	0.1398	0.1317	0.1440	0.0124
	EM-ELM	28.6	0.0125	0.0665	0.1369	0.0220
	CS-ELM	16.8	0.0078	0.0863	0.1214	0.0177
Auto-MPG	ELM	25	0.0023	0.0681	0.0804	0.0044
	I-ELM	2000	0.1422	0.0789	0.0831	0.0053
	EM-ELM	21.7	0.0086	0.0692	0.0803	0.0056
	CS-ELM	13.5	0.0125	0.0725	0.0806	0.0056
Boston housing	ELM	50	0.0078	0.0692	0.0992	0.0111
	I-ELM	2000	0.1844	0.0922	0.1043	0.0077
	EM-ELM	46.05	0.0219	0.0730	0.1052	0.0101
	CS-ELM	22.75	0.0117	0.0883	0.1086	0.0113
Abalone	ELM	25	0.0102	0.0756	0.0776	0.0190
	I-ELM	1840.6	0.4820	0.0814	0.0822	0.0030
	EM-ELM	11.5	0.0117	0.0792	0.0794	0.0025
	CS-ELM	20.8	0.0375	0.0755	0.0771	0.0014
Delta ailerons	ELM	45	0.0352	0.0377	0.0391	0.0005
	I-ELM	1779.8	0.5758	0.0409	0.0409	0.0012
	EM-ELM	11.65	0.0195	0.0398	0.0398	0.0008
	CS-ELM	13	0.0742	0.0391	0.0394	0.0007
Delta elevators	ELM	80	0.1109	0.0521	0.0533	0.0005
	I-ELM	1926.2	0.7781	0.0546	0.0543	0.0014
	EM-ELM	28.35	0.1250	0.0530	0.0538	0.0009
	CS-ELM	11.65	0.1078	0.0533	0.0537	0.0005
California housing	ELM	50	0.1109	0.1306	0.1329	0.0020
	I-ELM	2000	1.6391	0.1489	0.1489	0.0011
	EM-ELM	46.65	0.5117	0.1304	0.1328	0.0014
	CS-ELM	38.25	0.3469	0.1318	0.1337	0.0019
Ailerons	ELM	65	0.0656	0.0505	0.0526	0.0013
	I-ELM	2000	0.8492	0.0625	0.0658	0.0126
	EM-ELM	79.8	0.5203	0.0499	0.0526	0.0013
	CS-ELM	31.9	0.3414	0.0510	0.0530	0.0008
Bank domains	ELM	330	1.2617	0.0383	0.0428	0.0007
	I-ELM	2000	1.1430	0.0595	0.0601	0.0018
	EM-ELM	90.9	1.0125	0.0449	0.0465	0.0011
	CS-ELM	85.55	2.5313	0.0424	0.0442	0.0009
House census(8L)	ELM	160	1.2961	0.0618	0.0697	0.0072
	I-ELM	2000	2.0906	0.0827	0.0819	0.0009
	EM-ELM	99.6	3.2664	0.0649	0.0681	0.0024
	CS-ELM	89.3	14.1922	0.0633	0.0685	0.0031

Chapter 7

Two-Stage Extreme Learning Machine for Regression

7.1 Introduction

According to Chapter 6, which is inspired by pruning improvements of ELM, we could randomly generate a large number of hidden nodes as the candidate reservoir. Instead of pruning the hidden nodes from a large network, a forward selection method can be designed to select the significant hidden nodes from the candidate reservoir to form the smallest model that can explain the data. There are two categories of methods available for forward selection – the best subset selection and the stepwise forward selection. Give a criterion such as Mallows's C_p [109], final prediction error (FPE)[105], or Akaike's information criteria (AIC) [106], the best subset selection considers all the possible subset combinations, which is too expensive to implement when there is a large number of terms in the reservoir. To reduce the computational burden, many stepwise forward selection algorithms are introduced.

A constructive hidden nodes selection method for ELM (named CS-ELM) [147] was proposed based on a stepwise forward selection method, which is described in Chapter

6. At each step, the hidden node with an output that has the highest correlation with the current residual is selected. The hidden nodes selection stops automatically when the unbiased risk estimation criterion C_P reaches its minimum. However, this method is greedy and we may miss a parsimonious solution due to the fact that previously selected hidden nodes may influence the selection of later hidden nodes. In other words, the previously selected hidden nodes could become insignificant when the later selected hidden nodes are added to the network.

Besides, one of the most popular stepwise forward algorithms is orthogonal least squares (OLS) [114, 117, 118, 119], which is based on the modified Gram–Schmidt (MGS) orthogonalisation. OLS has been mentioned in Section 6.2. Based on the OLS algorithm, Li et al. [145] has proposed a fast construction algorithm (FCA) for a class of SLFNs with fully random hidden nodes. An integrated framework has been developed with two sequential phases. Phase one selects the significant hidden node one-by-one from the candidate pool; while phase two reviews the hidden nodes selected in phase one, and replaces the selected hidden nodes with candidate nodes that achieve more contribution. The performance evaluation showed that FCA is faster and better than the preliminary ELM and OLS methods.

Inspired by the CS-ELM and the FCA algorithm, we propose a systematic two-stage algorithm (named TS-ELM) to improve the preliminary ELM. In the first stage, a forward recursive algorithm is applied to select the hidden nodes from the candidates randomly generated in each step and add them to the network until the stopping criterion achieving its minimum. The significance of each hidden node is measured by the net contribution when it is added to the network. Then, the selected hidden nodes are reviewed at the second stage and the insignificance ones are removed from the network, which drastically reduces the network complexity.

7.2 Proposed two-stage ELM for regression (TS-ELM)

In this section, a two-stage ELM, called TS-ELM, is proposed. TS-ELM consists of two stages, forward selection stage and backward elimination stage. Before introducing the two stages, the problem formulation is presented.

7.2.1 Problem formulation

Suppose N data samples $\{\mathbf{x}_i, t_i\}_{i=1}^N$ are given, and the hidden node parameters ($\mathbf{a}_i$ and b_i) in the ELM network are randomly generated. As mentioned in Section 2.1.3, the mathematical model can be represented in the matrix form $\mathbf{T} = \mathbf{H}\boldsymbol{\beta} + \mathbf{E}$. Without loss of generality, a single output is considered here to simplify the exposition, i.e. $m = 1$. Therefore, the matrix form can be rewritten as

$$\mathbf{t} = \mathbf{H}\boldsymbol{\beta} + \mathbf{e} \quad (7.1)$$

where

$$\mathbf{t} = \begin{bmatrix} t_1 & \cdots & t_N \end{bmatrix}_{N \times 1}^T, \quad \mathbf{e} = \begin{bmatrix} e_1 & \cdots & e_N \end{bmatrix}_{N \times 1}^T \quad (7.2)$$

From equation (2.21) and equation (2.15), the formulae for least-squares solution $\hat{\boldsymbol{\beta}}$ and cost function $\mathbf{J}$ can be modified as

$$\hat{\boldsymbol{\beta}} = (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T \mathbf{t} \quad (7.3)$$

$$\mathbf{J}(\hat{\boldsymbol{\beta}}) = \mathbf{t}^T \mathbf{t} - \hat{\boldsymbol{\beta}}^T \mathbf{H}^T \mathbf{t} \quad (7.4)$$

7.2.2 Stage one – forward model selection

In forward model selection, many groups of hidden nodes are randomly generated in each step, and the group of hidden nodes with the highest net contribution is selected and added to the network. The selection process is terminated when the stopping criterion of final prediction error (FPE) [105] achieves its minimum. In order to identify the significance of the group of hidden nodes, the net contribution when they are added to the network has to be calculated explicitly [114, 117]. In OLS and FCA, the net contribution is obtained using orthogonal transformation. Alternatively, it can be computed by solving the least-squares problem recursively [148, 149, 150]. In our proposed method, hidden nodes are added to the network group-by-group and the net contributions of the group of hidden nodes are computed recursively without matrix decomposition.

We define a recursive matrix $\mathbf{K}_k \in \mathbb{R}^{k \times k}$ as

$$\mathbf{K}_k \equiv \mathbf{H}_k^T \mathbf{H}_k \quad (7.5)$$

where $\mathbf{H}_k \in \mathbb{R}^{N \times k}$ contains k groups of hidden nodes selected by the first k steps. And the corresponding least-squares solution and cost function can be presented as

$$\hat{\beta}_k = \mathbf{K}_k^{-1} \mathbf{H}_k^T \mathbf{t} \quad (7.6)$$

$$\mathbf{J}_k = \mathbf{t}^T \mathbf{t} - \hat{\beta}_k^T \mathbf{H}_k^T \mathbf{t} \quad (7.7)$$

where $\hat{\beta}_k \in \mathbb{R}^k$, and $\mathbf{J}_k$ is the cost function when k groups of hidden nodes are selected and added to the network.

In order to calculate the net contribution of a group of hidden nodes, a matrix series has to be defined

$$\mathbf{R}_k \equiv \begin{cases} \mathbf{I} & , k = 0 \\ \mathbf{I} - \mathbf{H}_k \mathbf{K}_k^{-1} \mathbf{H}_k^T & , \text{otherwise} \end{cases} \quad (7.8)$$

where $\mathbf{H}_k$ is of full column rank.

In the $(k + 1)$ th step, a group of hidden nodes is added to the existing network and the hidden nodes output matrix for the newly added group of hidden nodes is denoted by $\delta\mathbf{H}_{k+1}$. Then, the recursive formula for $\mathbf{R}$ is given as

$$\mathbf{R}_{k+1} = \mathbf{R}_k - \mathbf{R}_k \delta\mathbf{H}_{k+1} (\delta\mathbf{H}_{k+1}^T \mathbf{R}_k \delta\mathbf{H}_{k+1})^{-1} \delta\mathbf{H}_{k+1}^T \mathbf{R}_k^T \quad (7.9)$$

The proof of equation (7.9) is presented in section 7.5 of this chapter. The matrix $\mathbf{R}_k$ is known as a residue matrix and its properties is discussed in [148]. In the group-by-group hidden nodes adding mode, the properties of $\mathbf{R}_k$ are still valid. Hence, we have

$$\mathbf{R}_k^T = \mathbf{R}_k, \quad \mathbf{R}_k \mathbf{R}_k = \mathbf{R}_k \quad (7.10)$$

With the definition of $\mathbf{R}_k$ in (7.8) and the cost function equation defined by (7.7), the calculation of the cost function can be modified as

$$\mathbf{J}_k = \mathbf{t}^T \mathbf{t} - \hat{\beta}_k^T \mathbf{H}_k^T \mathbf{t} = \mathbf{t}^T \mathbf{R}_k \mathbf{t} \quad (7.11)$$

And, with (7.9) and (7.11), it follows that

$$\begin{cases} \mathbf{J}_{k+1} = \mathbf{t}^T \mathbf{R}_{k+1} \mathbf{t} = \mathbf{J}_k - \mathbf{t}^T \mathbf{R}_k \delta\mathbf{H}_{k+1} (\delta\mathbf{H}_{k+1}^T \mathbf{R}_k \delta\mathbf{H}_{k+1})^{-1} \delta\mathbf{H}_{k+1}^T \mathbf{R}_k^T \mathbf{t} \\ \mathbf{J}_0 = \mathbf{t}^T \mathbf{t} \end{cases} \quad (7.12)$$

Hence, the net contribution of $\delta\mathbf{H}_{k+1}$ to the cost function is given by

$$\begin{aligned} \Delta\mathbf{J}_{k+1}(\delta\mathbf{H}_{k+1}) &= \mathbf{J}_k - \mathbf{J}_{k+1} \\ &= \mathbf{t}^T (\mathbf{R}_k - \mathbf{R}_{k+1}) \mathbf{t} \\ &= \mathbf{t}^T \mathbf{R}_k \delta\mathbf{H}_{k+1} (\delta\mathbf{H}_{k+1}^T \mathbf{R}_k \delta\mathbf{H}_{k+1})^{-1} \delta\mathbf{H}_{k+1}^T \mathbf{R}_k^T \mathbf{t} \end{aligned} \quad (7.13)$$

Usually, the number of groups generated for selection in each step is determined by the

user. And the net contribution of each group to the cost function is computed using equation (7.13). The group of hidden nodes that has the highest net contribution to the cost function reduction is selected and added to the network. Then the output weight of the system is updated accordingly.

From equation (7.6), the recursive formula for updating the output weight is

$$\hat{\beta}_{k+1} = \mathbf{K}_{k+1}^{-1} \mathbf{H}_{k+1}^T \mathbf{t} \quad (7.14)$$

where $\mathbf{H}_{k+1} = [\mathbf{H}_k \quad \delta \mathbf{H}_{k+1}]$, and

$$\mathbf{K}_{k+1}^{-1} = (\mathbf{H}_{k+1}^T \mathbf{H}_{k+1})^{-1} = \begin{bmatrix} \mathbf{H}_k^T \mathbf{H}_k & \mathbf{H}_k^T \delta \mathbf{H}_{k+1} \\ \delta \mathbf{H}_{k+1}^T \mathbf{H}_k & \delta \mathbf{H}_{k+1}^T \delta \mathbf{H}_{k+1} \end{bmatrix}^{-1} \quad (7.15)$$

Let $\mathbf{K}_{k+1}^{-1} = \mathbf{M}_{k+1}$, and partition it as

$$\mathbf{K}_{k+1}^{-1} = \mathbf{M}_{k+1} = \begin{bmatrix} \mathbf{M}_1 & \mathbf{M}_2 \\ \mathbf{M}_3 & \mathbf{M}_4 \end{bmatrix} \quad (7.16)$$

Let $\mathbf{S}_k = \delta \mathbf{H}_{k+1}^T \delta \mathbf{H}_{k+1} - \delta \mathbf{H}_{k+1}^T \mathbf{H}_k \mathbf{K}_k^{-1} \mathbf{H}_k^T \delta \mathbf{H}_{k+1}$, which is the Schur complement of $\mathbf{H}_k^T \mathbf{H}_k$ [72]. According to the theory of ELM, $\mathbf{H}_k^T \mathbf{H}_k$ is nonsingular, and so the matrix $\mathbf{M}_{k+1}$ is invertible if and only if $\mathbf{S}_k$ is invertible [72]. In addition, the matrix $\mathbf{M}_{k+1}$ can be solved as follows:

$$\begin{cases} \mathbf{M}_1 = \mathbf{M}_k + \mathbf{M}_k \mathbf{H}_k^T \delta \mathbf{H}_{k+1} \mathbf{S}_k^{-1} \delta \mathbf{H}_{k+1}^T \mathbf{H}_k \mathbf{M}_k \\ \mathbf{M}_2 = -\mathbf{M}_k \mathbf{H}_k^T \delta \mathbf{H}_{k+1} \mathbf{S}_k^{-1} \\ \mathbf{M}_3 = -\mathbf{S}_k^{-1} \delta \mathbf{H}_{k+1}^T \mathbf{H}_k \mathbf{M}_k \\ \mathbf{M}_4 = \mathbf{S}_k^{-1} \end{cases} \quad (7.17)$$

Then, the updating of output weight (equation (7.14)) can be rewritten as

$$\hat{\beta}_{k+1} = \begin{bmatrix} \hat{\beta}_k^{new} \\ \beta_{k+1} \end{bmatrix} = \begin{bmatrix} \hat{\beta}_k - \mathbf{M}_k \mathbf{H}_k^T \delta \mathbf{H}_{k+1} \mathbf{D}_k \\ \mathbf{D}_k \end{bmatrix} \quad (7.18)$$

where $\mathbf{D}_k = \mathbf{S}_k^{-1} \delta \mathbf{H}_{k+1}^T \mathbf{R}_k \mathbf{t}$ and $\mathbf{S}_k = \delta \mathbf{H}_{k+1}^T \mathbf{R}_k \delta \mathbf{H}_{k+1}$. In the $(k+1)$ th step, a new group of hidden nodes are added to the network. The output weight of the hidden nodes previously added is updated and denoted by $\hat{\beta}_k^{new}$; while the output weight of newly added hidden nodes is denoted by β_{k+1} .

The hidden nodes are added group-by-group to the network until the number of hidden nodes in the network reaches the value defined by the user. The optimal network size is determined by a stopping criterion that is applied to the validation data drawn from the training set.

The final prediction error (FPE) criterion [105, 151] is implemented in our algorithm, which is one of most widely used information theoretic criteria. The FPE criterion compromises the performance of the current model and the number of hidden nodes existing in the network, and it is defined as

$$FPE(p) = \frac{SSE_p}{n} \cdot \left(\frac{n+p}{n-p} \right) \quad (7.19)$$

where n is the number of samples in the validation dataset; p is the number of hidden nodes in the current model that is under investigation; and SSE_p stands for the sum of squared errors over the validation dataset, which is given by $SSE_p = \sum_i^n (\mathbf{t}_i - \mathbf{y}_i^{(p)})^2$. And $\mathbf{y}_i^{(p)}$ is the output of the model with p selected hidden nodes. Hence, the most suitable network size is given by the value of p when FPE reaches its minimum.

Summary of the forward model selection algorithm:

The key step of forward model selection is to effectively compute the net contribution of a hidden node (or a group of hidden nodes) to the cost function. The significant hidden node (or a group of hidden nodes) is selected based on the net contribution. The value of

the cost function is updated recursively by making use of residue matrix $\mathbf{R}$. Moreover, the output weights of the system are also updated recursively. Finally, the selection is terminated by the stopping criterion FPE.

Given a set of training data $\{(\mathbf{x}_i, t_i)\}_{i=1}^N$, the maximum number of hidden nodes that can be added into the network (denoted by $L_{\max}$), the number of groups generated in each step for selection (denoted by N_{Group}), the number of hidden nodes in each group (denoted by n_{Group}) and the hidden node output function G , we split the training set equally into two non-overlapping subsets for learning and validation.

Let $k = 0$, $L = 0$, $\mathbf{H}_0 = \mathbf{0}$, $\hat{\beta}_0 = \mathbf{0}$, $\mathbf{R}_0 = \mathbf{I}$ and $\mathbf{J}_0 = \mathbf{t}^T \mathbf{t}$.

while $L < L_{\max}$:

- Increase the number of hidden nodes by n_{Group} : $L = L + n_{\text{Group}}$
- **for** $i = 1 : N_{\text{Group}}$:
 - Randomly generate the parameters $\{(\mathbf{a}_j^{(i)}, b_j^{(i)})\}_{j=1}^{n_{\text{Group}}}$ for a group of hidden nodes according to any continuous probability distribution.
 - Train the group of hidden nodes using the data for learning with the number of samples n . And the corresponding hidden node output matrix of the newly added group of hidden nodes is denoted by $\delta \mathbf{H}_{k+1}^{(i)}$.
 - Compute the net contribution of the group of hidden nodes to the cost function $\Delta \mathbf{J}_{k+1}^{(i)}$:

$$\Delta \mathbf{J}_{k+1}^{(i)} = \mathbf{t}^T \mathbf{R}_k \delta \mathbf{H}_{k+1}^{(i)} (\delta \mathbf{H}_{k+1}^{(i),T} \mathbf{R}_k \delta \mathbf{H}_{k+1}^{(i)})^{-1} \delta \mathbf{H}_{k+1}^{(i),T} \mathbf{R}_k^T \mathbf{t}$$

endfor

- Let $i^* = \{i | \max_{1 \leq i \leq N_{\text{Group}}} \|\Delta \mathbf{J}_{k+1}^{(i)}\|\}$. Then set $\delta \mathbf{H}_{k+1} = \delta \mathbf{H}_{k+1}^{(i^*)}$ and $\Delta \mathbf{J}_{k+1} = \Delta \mathbf{J}_{k+1}^{(i^*)}$. Therefore, the value of the cost function and the output weights of the

system can be updated as

$$\mathbf{J}_{k+1} = \mathbf{J}_k - \Delta \mathbf{J}_{k+1}$$

$$\hat{\beta}_{k+1} = \begin{bmatrix} \hat{\beta}_k^{new} \\ \beta_{k+1} \end{bmatrix} = \begin{bmatrix} \hat{\beta}_k - \mathbf{M}_k \mathbf{H}_k^T \delta \mathbf{H}_{k+1} \mathbf{D}_k \\ \mathbf{D}_k \end{bmatrix}$$

where $\mathbf{M}_k = (\mathbf{H}_k^T \mathbf{H}_k)^{-1}$, $\mathbf{D}_k = \mathbf{S}_k^{-1} \delta \mathbf{H}_{k+1}^T \mathbf{R}_k \mathbf{t}$ and $\mathbf{S}_k = \delta \mathbf{H}_{k+1}^T \mathbf{R}_k \delta \mathbf{H}_{k+1}$.

- Update the intermediate parameters:

$$\mathbf{R}_{k+1} = \mathbf{R}_k - \mathbf{R}_k \delta \mathbf{H}_{k+1} (\delta \mathbf{H}_{k+1}^T \mathbf{R}_k \delta \mathbf{H}_{k+1})^{-1} \delta \mathbf{H}_{k+1}^T \mathbf{R}_k^T$$

$$\mathbf{M}_{k+1} = \begin{bmatrix} \mathbf{M}_k + \mathbf{M}_k \mathbf{H}_k^T \delta \mathbf{H}_{k+1} \mathbf{S}_k^{-1} \delta \mathbf{H}_{k+1}^T \mathbf{H}_k \mathbf{M}_k & -\mathbf{M}_k \mathbf{H}_k^T \delta \mathbf{H}_{k+1} \mathbf{S}_k^{-1} \\ -\mathbf{S}_k^{-1} \delta \mathbf{H}_{k+1}^T \mathbf{H}_k \mathbf{M}_k & \mathbf{S}_k^{-1} \end{bmatrix}$$

- $k = k + 1$.

endwhile

for $p = n_{\text{Group}} : n_{\text{Group}} : L_{\text{max}}$, where p is the number of hidden nodes selected:

Train the SLFN using the validation data and calculate $FPE(p) = \frac{SSE_p}{n}$.

$(\frac{n+p}{n-p})$, where n is the number of observations in the validation set and

$$SSE_p = \sum_i^n (\mathbf{t}_i - \mathbf{y}_i^{(p)})^2.$$

endfor

Let $p^* = \{p | \min_{1 \leq p \leq L_{\text{max}}} \|FPE\|\}$.

Build the SLFN with hidden nodes selected and retrain the network with the whole training set including learning and validation data. Then evaluate the performance of the SLFN using the testing dataset.

7.2.3 Stage two – backward elimination

According to the procedure of forward selection, the forward model selection is trying to maximize the cost function reduction in each step (i.e. maximize $\Delta\mathbf{J}$ in each step). However, the process cannot guarantee the most compact network structure due to the fact that previously selected hidden nodes may influence the selection of later hidden nodes [149, 152]. In other words, the forward selection may not be optimal from a global sense because previously selected hidden nodes may become insignificant when the later selected hidden nodes are added to the network. Nevertheless, if a conservative stopping point is used, all the hidden nodes of the minimal model should be contained in this non-minimal model [141], which means that the resultant network structure by forward selection may contain all the important hidden nodes of the optimal structure as well as some insignificant hidden nodes. This is because the network could not meet the expected performance until all the important hidden nodes are added. Therefore, one needs to remove the insignificant hidden nodes from the network in order to achieve a parsimonious solution. To improve the compactness of the network, a pruning algorithm using leave-one-out (LOO) cross validation [153, 154, 111] is applied to this non-minimal model to refine the final model structure.

The hidden node is reviewed one-by-one in this stage, and the insignificant hidden nodes will be removed from the network. There are two steps in the backward elimination stage, which is outlined below, and the detailed procedure will be given later. Assume the number of hidden node determined by stage one is p^* .

Step 1 Hidden layer output matrix reconstruction. The main idea is that if the hidden node under review is not the last hidden node added to the network, the output of this hidden node has to be exchanged with the output of the hidden node that is the last one to be added. For example, given the hidden layer output matrix $\mathbf{H} = [\mathbf{h}_1 \quad \mathbf{h}_2 \cdots \mathbf{h}_{p^*}]$ (where $\mathbf{h}_k$ is the output of k th hidden node, i.e. the k th column of $\mathbf{H}$, $k = 1, \dots, p^*$). Assume the first hidden node is under review and its output is $\mathbf{h}_1$. After the reconstruction, the hidden layer output matrix becomes $\mathbf{H}^{(R)} =$

$$[\mathbf{h}_{p^*} \quad \mathbf{h}_2 \cdots \mathbf{h}_1] = [\mathbf{h}_1^{(R)} \quad \mathbf{h}_2^{(R)} \cdots \mathbf{h}_{p^*}^{(R)}].$$

Step 2 Hidden nodes elimination. This step is to review the contribution of the hidden nodes under consideration. Because the hidden node (or the group of hidden nodes) added in the last step in the forward selection must be the significant node(s), the evaluation of the last added hidden node (or group of hidden nodes) can be omitted. If the LOO error of the network without any hidden node is smaller than the network with the hidden node or the change of LOO error is trivial, the hidden node can be removed from the network.

Step 1 – Hidden layer output matrix reconstruction

Suppose that one of the selected hidden nodes is reviewed and its output is $\mathbf{h}_k$, $k \in [1, \dots, p^* - 1]$. Then the task of matrix reconstruction is to exchange the $\mathbf{h}_k$ with $\mathbf{h}_{p^*}$. Given the hidden layer output matrix that is determined by stage one, $\mathbf{H} = [\mathbf{h}_1 \quad \mathbf{h}_2 \cdots \mathbf{h}_{p^*}]$, and the elementary matrix $\mathbf{Q}_{kp^*} = [\mathbf{q}_1 \quad \mathbf{q}_{p^*} \cdots \mathbf{q}_k]$ (where $\mathbf{q}_k(k) = 1$, and the rest of the elements in $\mathbf{q}_k$ are 0), we have

$$\mathbf{H}^{(R)} = \mathbf{H}\mathbf{Q} \tag{7.20}$$

Step 2 – Hidden nodes elimination

The hidden nodes elimination is based on the results of LOO cross validation [153, 154, 111]. The review of the hidden node starts from the anterior hidden node (i.e. $\mathbf{h}_1$, followed by $\mathbf{h}_2$, etc.). After the matrix reconstruction, we make use of the PREdiction Sum of Squares (PRESS) statistics [153, 154] to calculate the LOO error directly without actually splitting the training data set, i.e.

$$\varepsilon^{(-)}(i) = \frac{\varepsilon(i)}{1 - \tilde{\mathbf{h}}_i \mathbf{M} \tilde{\mathbf{h}}_i^T} \quad i = 1, \dots, N \tag{7.21}$$

where $\tilde{\mathbf{h}}_i$ is the i th row of $\mathbf{H}^{(R)}$ ($i = 1, \dots, N$), $\varepsilon(i) = \mathbf{y}_i - \tilde{\mathbf{h}}_i \beta_i$ and $\mathbf{M} = (\mathbf{H}^{(R)T} \mathbf{H}^{(R)})^{-1}$.

In addition, the mean of LOO errors over N data samples is given by

$$\mathbf{J}(\varepsilon) = \frac{1}{N} \sum_{i=1}^N (\varepsilon^{(-)}(i))^2 \quad (7.22)$$

If the value of $\mathbf{J}(\varepsilon)$ without the hidden node under consideration is smaller than the value of $\mathbf{J}(\varepsilon)$ with the hidden node under review, the hidden node will be removed from the network.

To simplify the calculation of the pruning process, a recursive method is introduced in [150], which computes $\mathbf{M}_k$ by using the matrix $\mathbf{M}_{k+1}$ and equations (7.16) and (7.17). Because the hidden node is considered one-by-one, the equations (7.16) and (7.17) can be modified by

$$\mathbf{M}_{k+1} = \begin{bmatrix} \mathbf{M}_1 & \mathbf{M}_2 \\ \mathbf{M}_3 & \mathbf{M}_4 \end{bmatrix} = \begin{bmatrix} \mathbf{M}_k + \mathbf{M}_k \mathbf{H}_k^T \delta \mathbf{h}_{k+1} \mathbf{S}_k^{-1} \delta \mathbf{h}_{k+1}^T \mathbf{H}_k \mathbf{M}_k & -\mathbf{M}_k \mathbf{H}_k^T \delta \mathbf{h}_{k+1} \mathbf{S}_k^{-1} \\ -\mathbf{S}_k^{-1} \delta \mathbf{h}_{k+1}^T \mathbf{H}_k \mathbf{M}_k & \mathbf{S}_k^{-1} \end{bmatrix} \quad (7.23)$$

where $\mathbf{S}_k = \delta \mathbf{h}_{k+1}^T \mathbf{R}_k \delta \mathbf{h}_{k+1}$ is a scalar, and $\mathbf{M}_1 = \mathbf{M}_k + \mathbf{M}_k \mathbf{H}_k^T \delta \mathbf{h}_{k+1} \mathbf{S}_k^{-1} \delta \mathbf{h}_{k+1}^T \mathbf{H}_k \mathbf{M}_k = \mathbf{M}_k + \mathbf{M}_2 \mathbf{M}_3 / \mathbf{M}_4$. Therefore, when deleting a hidden node with the output occupying the last column of $\mathbf{H}^{(R)}$, $\mathbf{M}_k$ could be formulated by

$$\mathbf{M}_k = \mathbf{M}_1 - \mathbf{M}_2 \mathbf{M}_3 / \mathbf{M}_4 \quad (7.24)$$

where $\mathbf{M}_1$, $\mathbf{M}_2$, $\mathbf{M}_3$ and $\mathbf{M}_4$ are sub-matrices of $\mathbf{M}_{k+1}$.

Summary of the backward elimination algorithm:

Given a SLFN with p^* hidden nodes in the hidden layer that is determined by stage one.

for $k = p^* : 1$

- Calculate the mean of LOO errors with k hidden nodes using equation (7.22), and

denote it by PRESS1.

- **for** $i = 1 : k$

Calculate the mean of LOO errors with $k - 1$ hidden nodes by eliminating the i th hidden node from the network using equations (7.20)-(7.24).

And the error is denoted by PRESS2.

If $\text{PRESS2} < \text{PRESS1}$, break.

endfor

- if $\text{PRESS2} > \text{PRESS1}$, break.
- $k = k - 1$.

endfor

If the LOO errors of the network with $p^* - 1$ hidden nodes are always larger than the network with p^* hidden nodes, the solution obtained in stage one is the optimal solution.

7.2.4 Comparison of TS-ELM and FCA

As mentioned in the introduction, FCA [145] proposed by Li et al. is a fast construction algorithm specially developed for an integrated analytic framework with two sequential phases. The functions of the two phases in FCA are quite similar to the two stages introduced in the proposed TS-ELM. However, there are many differences between the two algorithms.

FCA selects hidden node *one-by-one* from a candidate pool that contains a large number of hidden nodes randomly generated beforehand, while TS-ELM selects hidden nodes *group-by-group* from groups of hidden nodes randomly generated in each step. TS-ELM can also select hidden nodes one-by-one when the number of hidden node in each group is one. In the first phase of FCA, the forward selection method relies on the matrix

decomposition, while in the first stage of TS-ELM, an alternative method is introduced that solves the least-squares problem recursively. In FCA, the forward selection is terminated when the number of hidden nodes reaches a default number or the training error is less than or equal to the expected value, which cannot be considered as an automatic termination. Although other stopping criteria like AIC have also been suggested in [145], no empirical study has been done with other criteria. In our proposed TS-ELM, the forward selection is terminated by the stopping criterion FPE. The suitable number of hidden nodes is determined when FPE achieves its minimum value.

According to Section 7.2.3, the stepwise forward selection cannot guarantee an optimal solution because it is greedy and is not a global search method. In FCA, the network structure is determined in phase one. The objective of phase two is to re-evaluate the contribution of the selected hidden nodes and some of them may be replaced by the hidden nodes in the candidate pool if necessary, which indeed does not change the number of hidden nodes. However, in TS-ELM, the key idea of stage two is to look for smaller network structure that can achieve good performance. Hence, TS-ELM may be better than FCA in obtaining a more compact network structure.

7.3 Performance evaluation of TS-ELM

The performance of TS-ELM is evaluated on the benchmark problems described in Table 7.1, which include six regression applications (Auto-MPG, Abalone, Ailerons, Puma, Delta elevators, and House census(8L)) [136]. It is compared with the preliminary ELM [13] and EM-ELM [28].

All the evaluations were carried out in the Matlab 7.4.0 environment running on a desktop with CPU 2.66GHz and 3GB RAM. For all the regression applications, the input data have been normalized into $[-1, 1]$, while the output data have been normalized into $[0, 1]$. Twenty trials of simulations were conducted for each application with the preliminary ELM, EM-ELM and TS-ELM algorithms. For each trial of simula-

Table 7.1: Specification of benchmark datasets for evaluation of TS-ELM

Datasets	# Attributes	# Observations	# Training Data	# Testing Data
Auto-MPG	7	398	200	198
Abalone	8	4177	2000	2177
Ailerons	40	7154	3000	4154
Puma	8	8192	4500	3692
Delta elevators	6	9517	4000	5517
House census(8L)	8	22784	10000	12784

tions, the dataset of the application was divided into training and testing datasets with the number of samples indicated in Table 7.1. The process is random based, which means, take Abalone for example, we randomly select 2000 samples out of 4177 to form the training dataset, and the remaining samples in the testing dataset. The activation function used in the comparison is the sigmoidal additive activation function (Sig) $G(\mathbf{a}, b, \mathbf{x}) = (1)/(1 + \exp(-(\mathbf{a} \cdot \mathbf{x} + b)))$. For SLFNs with ELM, EM-ELM, and TS-ELM algorithms and additive hidden nodes, the input weights and biases were randomly generated from the range $[-1, 1]$.

7.3.1 Parameter selection

For the preliminary ELM, the number of hidden nodes in the hidden layer L is searched using a trial and error method. L is set to be 5, 10, 15, 20, ..., 400. Twenty trials of simulations are conducted for each network with L hidden nodes and the network structure with the highest generalization is selected. The value of L selected for each application is presented in Table 7.2.

The EM-ELM algorithm has two parameters that need to be considered: the maximum number of hidden nodes $L_{\max}$ and the expected training rate ε . In the evaluations, we aim to obtain the network structure when the training rate reaches the specified ε without the constraint of $L_{\max}$. Therefore, in our experiments, the value of $L_{\max}$ is set to be a large value of 2000, and the value of ε is set according to the training rate obtained by

the preliminary ELM for each regression case.

For TS-ELM, the maximum number of hidden nodes that can be added into the network ($L_{\max}$) is given by the user. Intuitively, this parameter may influence the model selection using the FPE criterion. Different values of $L_{\max}$ may lead to different set of models to be evaluated using FPE criterion. Hence, we conduct investigations on applications with different $L_{\max}$ values. $L_{\max}$ is evaluated on the range [5,400] with an interval of 5. For each value of $L_{\max}$, the simulation is repeated for twenty times. The performance of TS-ELM with different values of $L_{\max}$ is presented in Fig. 7.1, Fig. 7.2 and Fig. 7.3 on the Auto-MPG, Abalone and Ailerons cases respectively. As observed from Fig. 7.1, when $L_{\max}$ is greater than 20, the performance of the network is stable with fluctuation of less than 0.005. Therefore, practically speaking, $L_{\max}$ can be set at any value greater than 20 for the Auto-MPG case. The statement is also true for the Abalone and Ailerons cases, which is shown in Fig. 7.2 and Fig. 7.3. For all of the cases, the values of $L_{\max}$ are shown in Table 7.2. We will discuss the number of groups generated in each step for selection (N_{Group}) and the number of hidden nodes in each group (n_{Group}) in the next subsection.

Figure 7.1: Parameter selection for the Auto-MPG case

Figure 7.2: Parameter selection for the Abalone case

Figure 7.3: Parameter selection for the Ailerons case

Table 7.2: Parameters setting for benchmark regression cases

Datasets	TS-ELM	EM-ELM		ELM
	$L_{\max}$	$L_{\max}$	ε	L
Auto-MPG	100	2000	0.07	25
Abalone	100	2000	0.08	25
Ailerons	100	2000	0.05	65
Puma	200	2000	0.016	360
Delta elevators	100	2000	0.053	80
House census(8L)	100	2000	0.065	160

7.3.2 Performance evaluation of TS-ELM

Effect of stopping criterion FPE in TS-ELM algorithm

The effect of stopping criterion FPE on the Abalone case is illustrated in Fig. 7.4. The FPE curve reflects the generalization capabilities of the model. The suitable number of hidden node is determined when FPE reaches its minimum. As shown in Fig. 7.4(a), the FPE curve is a “U” shape and small FPE is achieved when the network size (the number of hidden nodes) is on the range of [5, 20]. As observed in Fig. 7.4(b), the global minimum FPE is obtained when the number of hidden nodes added is equal to 11, i.e. $p^* = 11$. Similar curves have been obtained for the Auto-MPG and Delta elevators cases as shown in Fig. 7.5 and Fig. 7.6 respectively.

Effect of N_{Group} in TS-ELM algorithm

As mentioned in section 7.2.2, N_{Group} groups of hidden nodes are generated for selection in each step. In this section, the effect of N_{Group} in the TS-ELM algorithm is investigated and the performance of TS-ELM is compared with preliminary ELM and EM-ELM. The number of hidden nodes in each group is set to be one, i.e. $n_{\text{Group}} = 1$. All the evaluations are repeated for twenty times and the average hidden nodes selected in stage one and stage two, the average training time in stage one and stage two, the av-

(a) Network size $p \in (0, 25)$ (b) Network size $p \in (4, 22)$ Figure 7.4: Effect of FPE on Abalone case ($p^* = 11$)

Figure 7.5: Effect of FPE on Auto-MPG case ($p^* = 9$)

(a) Network size $p \in (0, 25)$ (b) Network size $p \in (4, 22)$ Figure 7.6: Effect of FPE on Delta elevators case ($p^* = 9$)

erage training root mean squared error (RMSE), the average testing RMSE, the standard deviation of average testing RMSE, and N_{Group} are listed in Table 7.3.

The training times for ELM and EM-ELM are the average training time of the evaluations. For the original ELM, the trial and error method that is used to determine the number of hidden nodes may require a lot of time, which has not been included in the training time shown in the table. While for EM-ELM, the parameter of expected training rate is usually determined by manual tuning, which is also time consuming. And the time spent on tuning this parameter is not included in the training time of EM-ELM.

As can be seen from Table 7.3, TS-ELM gives the most compact network structure with good generalization performance as compared to the original ELM and EM-ELM except for the Abalone case. For the Abalone case, the results of EM-ELM and TS-ELM are comparable. A comparison between $N_{\text{Group}} = 5$ and $N_{\text{Group}} = 10$ for TS-ELM is presented in Table 7.3 as well. Generally speaking, when N_{Group} increases, the selection of hidden nodes is more effective in stage one, which leads to less time spent in stage two to seek for the optimal network structure.

More detailed investigation of N_{Group} is conducted for the Aileron case. Fig. 7.7 presents the training time spent for stage one and stage two when N_{Group} varies. When N_{Group} increases, more effort is made in stage one to search for the significant hidden nodes and less insignificant nodes are involved, which saves the computational time in stage two. However, the overall training time spent for TS-ELM increases as the N_{Group} becomes larger and larger because the increasing rate of the training time for stage one is larger than the decreasing rate of training time for stage two. Fig. 7.8 presents the compactness of network when N_{Group} varies. More significant nodes can be identified when N_{Group} increases. And the network structure tends to remain unchanged after N_{Group} reaches a certain value. The network performance of TS-ELM is shown in Fig. 7.9, and TS-ELM is stable throughout the variation of N_{Group} .

Figure 7.7: Effect of N_{Group} on Ailerons case: training time for stage one and stage two

Figure 7.8: Effect of N_{Group} on Ailerons case: number of hidden nodes selected for stage one and stage two

Effect of n_{Group} in TS-ELM algorithm

In this section, the effect of n_{Group} in TS-ELM algorithm is investigated and the performance of TS-ELM is compared with preliminary ELM and EM-ELM. All the eval-

Figure 7.9: Effect of N_{Group} on Ailerons case: training and testing RMSE

uations are repeated for twenty times and the average number of hidden nodes selected in stage one and stage two, the average training time in stage one and stage two, the average training RMSE, the average testing RMSE, the standard deviation of average testing RMSE, n_{Group} and N_{Group} are listed in Table 7.4. Similarly, TS-ELM achieves the most compact network structure which is observed on most of the applications.

For TS-ELM, the resultant network with the group-by-group adding mode is slightly larger than the one-by-one adding mode, but is no more than n_{Group} . As observed from Table 7.4, the group-by-group adding mode always spends less time in the training for EM-ELM. However, for TS-ELM, the situation becomes more complicated. When the hidden nodes are added group-by-group, the training time spent in stage one decreases. Whereas many insignificant hidden nodes may be included because the net contribution of the group is evaluated instead of the individual hidden node. Therefore, in stage two, it probably needs more time to check and eliminate the insignificant nodes one by one. In addition, the training time for stage two increases dramatically when n_{Group} becomes larger and larger. Hence, in practice, it is better to set n_{Group} to be equal to or less than 3.

Table 7.3: Performance comparison: hidden nodes are added one-by-one

Datasets	Algorithms	#Nodes (Stage2)	#Nodes (Stage1)	Training Time(s) (Stage1)	LOO Time(s) (Stage2)	Training RMSE	Testing RMSE		N_{Group}
							Mean	Dev.	
Auto-MPG	ELM		25	0.0023		0.0681	0.0804	0.0044	
	EM-ELM		21.7	0.0086		0.0692	0.0803	0.0056	
	TS-ELM	8.15	9.35	0.0984	0.0172	0.0767	0.0809	0.0060	5
		7.95	10.4	0.0813	0.0250	0.0744	0.0816	0.0052	10
Abalone	ELM		25	0.0102		0.0756	0.0776	0.0190	
	EM-ELM		11.5	0.0117		0.0792	0.0794	0.0025	
	TS-ELM	12.75	17.05	0.3422	0.5242	0.0758	0.0777	0.0021	5
		11.45	14.6	0.3211	0.3953	0.0759	0.0780	0.0011	10
Ailerons	ELM		65	0.0656		0.0505	0.0526	0.0013	
	EM-ELM		79.8	0.5203		0.0499	0.0526	0.0013	
	TS-ELM	20.35	30.3	0.8008	2.7055	0.0515	0.0526	0.0009	5
		21	29.05	1.3070	2.4312	0.0506	0.0530	0.0010	10
Puma	ELM		360	1.5156		0.1287	0.1404	0.0019	
	EM-ELM		209.15	5.5789		0.1398	0.1478	0.0023	
	TS-ELM	109.75	153.5	4.8422	263.1023	0.1349	0.1428	0.0024	5
		97.6	130.7	5.8578	150.4445	0.1353	0.1399	0.0024	10
Delta elevators	ELM		80	0.1109		0.0521	0.0533	0.0005	
	EM-ELM		28.35	0.1250		0.0530	0.0538	0.0009	
	TS-ELM	8.35	11.05	6.4094	0.5703	0.0534	0.0538	0.0005	5
		8.1	9.5	0.4234	0.3435	0.0533	0.0537	0.0006	10
House census(8L)	ELM		160	1.2961		0.0618	0.0697	0.0072	
	EM-ELM		99.6	3.2664		0.0649	0.0681	0.0024	
	TS-ELM	26.7	32.05	0.6	7.2336	0.0700	0.0720	0.0023	5
		27.9	33.55	0.7859	7.3703	0.0693	0.0703	0.0015	10

7.4 Summary

In this chapter, the TS-ELM algorithm is proposed to address the problem of network architecture design of ELM by applying a two-stage selection process. Stage one is a forward selection process that selects the hidden nodes group-by-group until the stopping criterion is met. Stage two is a backward refinement process that reviews the hidden nodes selected in stage one and eliminates the insignificant nodes. A performance comparison of TS-ELM with EM-ELM and ELM is evaluated on six real world benchmark regression applications. Specifically, the proposed algorithm achieves a compact network structure with competitive generalization performance when compared to the preliminary ELM. Empirical studies show that TS-ELM with a much smaller network can achieve better or similar performance as that of EM-ELM for the six cases we studied.

Table 7.4: Performance comparison: hidden nodes are added group-by-group

Datasets	Algorithms	#Nodes (Stage2)	#Nodes (Stage1)	Training Time(s) (Stage1)	LOO Time(s) (Stage2)	Training RMSE	Testing RMSE		n_{Group}	N_{Group}
							Mean	Dev.		
Auto-MPG	ELM		25	0.0023		0.0681	0.0804	0.0044		
	EM-ELM		13	0.0031		0.0779	0.0856	0.0091	1	
			12	0.0023		0.0762	0.0857	0.0056	2	
	TS-ELM	8.15	9.35	0.0984	0.0172	0.0767	0.0809	0.0060	1	5
		8.3	11	0.0211	0.0305	0.0751	0.0830	0.0069	2	5
Abalone	ELM		25	0.0102		0.0756	0.0776	0.0190		
	EM-ELM		11.5	0.0117		0.0792	0.0794	0.0025	1	
			12	0.0078		0.0785	0.0796	0.0029	3	
	TS-ELM	12.75	17.05	0.3422	0.5242	0.0758	0.0777	0.0021	1	5
		13.35	18.9	0.1203	0.6352	0.0755	0.0782	0.0023	3	5
Ailerons	ELM		65	0.0656		0.0505	0.0526	0.0013		
	EM-ELM		79.8	0.5203		0.0499	0.0526	0.0013	1	
			73.25	0.1383		0.0498	0.0528	0.0007	5	
	TS-ELM	20.35	30.3	0.8008	2.7055	0.0515	0.0526	0.0009	1	5
		23.4	37.25	0.3555	4.2086	0.0511	0.0532	0.0010	5	5
Puma	ELM		360	1.5156		0.1287	0.1404	0.0019		
	EM-ELM		209.15	5.5789		0.1398	0.1478	0.0023	1	
			212	0.8836		0.1392	0.1464	0.0023	10	
	TS-ELM	109.75	153.5	4.8422	263.1023	0.1349	0.1428	0.0024	1	5
		115.25	177	2.2273	415.9609	0.1371	0.1439	0.0029	10	5
Delta elevators	ELM		80	0.1109		0.0521	0.0533	0.0005		
	EM-ELM		28.35	0.1250		0.0530	0.0538	0.0009	1	
			32.8	0.0945		0.0529	0.0536	0.0005	2	
	TS-ELM	8.35	11.05	6.4094	0.5703	0.0534	0.0538	0.0005	1	5
		9	12.3	2.2078	0.6695	0.0532	0.0537	0.0004	2	5
House census(8L)	ELM		160	1.2961		0.0618	0.0697	0.0072		
	EM-ELM		99.6	3.2664		0.0649	0.0681	0.0024	1	
			92.25	0.7773		0.0646	0.0689	0.0027	5	
	TS-ELM	26.7	32.05	0.6000	7.2336	0.0700	0.0720	0.0023	1	5
		30.9	41.75	2.0203	15.3883	0.0693	0.0711	0.0019	5	5

7.5 Proof of equation (7.9)

From the equations (7.16) and (7.17), we have

$$\mathbf{K}_{k+1}^{-1} = \mathbf{M}_{k+1} = \begin{bmatrix} \mathbf{M}_k + \mathbf{M}_k \mathbf{H}_k^T \delta \mathbf{H}_{k+1} \mathbf{S}_k^{-1} \delta \mathbf{H}_{k+1}^T \mathbf{H}_k \mathbf{M}_k & -\mathbf{M}_k \mathbf{H}_k^T \delta \mathbf{H}_{k+1} \mathbf{S}_k^{-1} \\ -\mathbf{S}_k^{-1} \delta \mathbf{H}_{k+1}^T \mathbf{H}_k \mathbf{M}_k & \mathbf{S}_k^{-1} \end{bmatrix} \quad (7.25)$$

where $\mathbf{M}_k = \mathbf{K}_k^{-1}$, and $\mathbf{S}_k = \delta \mathbf{H}_{k+1}^T \mathbf{R}_k \delta \mathbf{H}_{k+1}$.

Based on the definition of $\mathbf{R}_k$ in (7.8), it gives

$$\mathbf{R}_{k+1} = \mathbf{I} - \mathbf{H}_{k+1} \mathbf{K}_{k+1}^{-1} \mathbf{H}_{k+1}^T = \mathbf{I} - \mathbf{H}_{k+1} \mathbf{M}_{k+1} \mathbf{H}_{k+1}^T \quad (7.26)$$

where $\mathbf{H}_{k+1} = [\mathbf{H}_k \quad \delta \mathbf{H}_{k+1}]$.

With $\mathbf{R}_k = \mathbf{I} - \mathbf{H}_k \mathbf{M}_k \mathbf{H}_k^T$, and substitute for $\mathbf{M}_{k+1}$ from (7.25), it produces

$$\begin{cases} \mathbf{R}_{k+1} = \mathbf{R}_k - \mathbf{R}_k \delta \mathbf{H}_{k+1} (\delta \mathbf{H}_{k+1}^T \mathbf{R}_k \delta \mathbf{H}_{k+1})^{-1} \delta \mathbf{H}_{k+1}^T \mathbf{R}_k^T \\ \mathbf{R}_0 = \mathbf{I} \end{cases} \quad (7.27)$$

Remark: If the hidden node is added one-by-one instead of group-by-group, the hidden node output matrix of newly added node becomes a column vector (denoted by $\delta \mathbf{h}_{k+1}$) and $\mathbf{H}_{k+1} = [\mathbf{H}_k \quad \delta \mathbf{h}_{k+1}]$. Then, equation (7.27) can be simplified by

$$\begin{cases} \mathbf{R}_{k+1} = \mathbf{R}_k - \frac{\mathbf{R}_k \delta \mathbf{h}_{k+1} \delta \mathbf{h}_{k+1}^T \mathbf{R}_k^T}{\delta \mathbf{h}_{k+1}^T \mathbf{R}_k \delta \mathbf{h}_{k+1}} \\ \mathbf{R}_0 = \mathbf{I} \end{cases} \quad (7.28)$$

which has been discussed by Li et al [148].

Chapter 8

Conclusions and Recommendations

8.1 Conclusions

This thesis investigated the problems of network architecture design and model selection of ELM. Essentially, we proposed the use of network ensemble to improve the generalization performance and then we focused on the novel constructive approaches to alter the network structure during the learning process in order to find the appropriate architecture. A parsimonious structure can be found by the constructive method with a backward refinement phase. The major research results covered in this thesis can be summarized as follows.

- Network ensemble

The solutions of OS-ELM network may vary due to the random generation of hidden nodes parameters. To solve the problem, an integration of OS-ELM networks is proposed to connect several OS-ELM networks in parallel and the average output of OS-ELMs is considered as the final output of the integration. OS-ELM ensemble is a system that has good adaptation to the slightly larger fluctuation caused by individual networks.

- Constructive method for ELM and its variants

OS-ELM is a fast and good online sequential learning machine. Similar to ELM, OS-ELM is based on a fixed network structure. The determination of the network structure relies on the entire data, which may contradict the fact that the data arrives sequentially. In our proposed CEOS-ELM, the network structure is allowed to vary during the sequential learning process. Inspired by EM-ELM that starts with a small network, hidden units can be added to OS-ELM if the network could not achieve the expected training accuracy.

While studying the enhancement of OS-ELM, we found that there may be redundant hidden units added when one applies constructive learning algorithms such as I-ELM and EM-ELM. In addition, the termination of the I-ELM and EM-ELM relies on the parameters set by the user, which could not guarantee a good generalization performance. To measure the significance of hidden nodes, and to automatically obtain a suitable network structure, three methods are introduced in the thesis:

(1) Selection method based on performance measure

Motivated by EI-ELM, a simple method to improve EM-ELM is to add a selection phase to measure the significance of randomly generated hidden nodes. In each step, the hidden node from the group of hidden nodes that leads to the highest residual error reduction is picked and added to the network. Based on this idea, we proposed EEM-ELM. However, it is still terminated by manually tuned parameters.

(2) Stepwise forward selection method based on correlation

Instead of using a simple selection method that randomly generates a group of hidden nodes in each step of the training process, we could randomly generate a large number of hidden nodes as the candidate reservoir and then pick the hidden node one-by-one via a stepwise forward selection method. In our proposed algorithm of CS-ELM, at each step, the hidden node with an output that has the highest correlation with the current residual is selected.

Furthermore, the unbiased risk estimation criterion C_P is applied to terminate the selection.

(3) Recursive forward selection method based on net contribution

Besides forward selection method based on correlation, the significance of hidden nodes can be measured by the net contribution when it is added to the network. Additionally, we found that the parsimonious network structure is probably missed by some greedy selection methods due to the fact that the hidden nodes added earlier may become insignificant when other hidden nodes are added to the network. Therefore, a two-stage algorithm is proposed and we called it TS-ELM. The first stage attempts to select hidden nodes by forward recursive algorithm and the selection is terminated by the FPE criterion; while the second stage is a backward refinement phase that removes the insignificant hidden nodes.

In conclusion, there are different features among the three proposed algorithms, EEM-ELM, CS-ELM and TS-ELM. These can be compared in terms of forward selection method, hidden node addition mode, stopping criterion, backward elimination and automatic termination, and the comparisons are summarized in Table 8.1.

Table 8.1: Feature comparison among EEM-ELM, CS-ELM and TS-ELM

	EEM-ELM	CS-ELM	TS-ELM
Forward selection method	In each step, pick the node (or nodes) with highest residual error reduction from the group of hidden node (or hidden nodes).	In each step, pick the node with an output that has the highest correlation with the current residual.	In each step, pick the nodes with highest net contribution.
Hidden node addition mode	One-by-one or group-by-group	One-by-one	One-by-one or group-by-group
Stopping criterion	Expected error or rate	C_P	FPE
Backward elimination	No	No	Yes, use LOO
Automatically terminated?	The expected error or rate need to be provided by user	Yes	Yes

8.2 Recommendations for future work

Based on the research work conducted in this thesis, several research directions that deserve further investigations are recommended below.

- In this thesis, we have constructed an ensemble of OS-ELM network, named EOS-ELM. EOS-ELM consists of many OS-ELM networks with the same number of hidden nodes and the same activation function for each hidden node. EOS-ELM may be expended in such way that given an initial training rate, the number of hidden nodes for each OS-ELM could vary during the sequential learning process according to CEOS-ELM. The initial training rate can be modified by the generalization performance of the ensemble. The process continues until a suitable training rate is found. Moreover, the activation function of the hidden node could be different in different OS-ELM networks or even in the same network. The choice of optimal activation function for specific application is still an open question.
- The proposed constructive algorithms in this thesis, either the stepwise forward selection method based on correlation or recursive forward selection based on net contribution, are designed for regression applications. Classification problems can be much different from the regression problem where the function's output is restricted to discrete values (or classes). In other words, the constructive algorithms for regression implements a real to real mapping; while the constructive algorithms for classification implements a real to digital mapping. The proposed constructive methods are not suitable for classification applications. Hence, we attempt to address the network architecture design of ELM for pattern classification in the future.
- In this thesis, we have proposed an improvement of OS-ELM based on EM-ELM, named as CEOS-ELM. However, it still relies on the manually tuned parameter, the expected training rate, to terminate the network structure in each sequential

learning step. The statistical stopping criteria cannot be applied directly in CEOS-ELM because the validation data is required to determine the stopping point and it is not possible to separate the sequential arriving data within the learning step. As one future work, we attempt to find a solution to automatically terminate the hidden nodes selection in online sequential learning.

Publication list

Journal Papers:

1. **Yuan Lan**, Yeng Chai Soh, and Guang-Bin Huang, “Ensemble of Online Sequential Extreme Learning Machine”, *Neurocomputing*, vol. 72, pp. 3391-3395, August 2009.
2. **Yuan Lan**, Yeng Chai Soh, and Guang-Bin Huang, “Two-Stage Extreme Learning Machine for Regression”, *Neurocomputing*, vol. 73, pp. 3028-3038, 2010.
3. **Yuan Lan**, Yeng Chai Soh, and Guang-Bin Huang, “Constructive Hidden Nodes Selection of Extreme Learning Machine for Regression”, *Neurocomputing*, vol. 73, pp. 3191-3199, 2010.
4. Guang-Bin Huang, Dianhui Wang, and **Yuan Lan**, “Extreme Learning Machines: a Survey”, *International Journal of Machine Learning and Cybernetics*(Accepted), 2011.
5. Rui zhang, **Yuan Lan**, Guang-Bin Huang and Zong-Ben Xu, “Universal Approximation of Extreme Learning Machine with Adaptive Growth of Hidden Nodes”, *IEEE Transactions on Neural Networks* (Submitted), 2011.

Conference Papers:

1. **Yuan Lan**, Yeng Chai Soh, Guang-Bin Huang, “Extreme Learning Machine based bacterial protein subcellular localization prediction”, *IEEE World Congress on Computational intelligence (WCCI’08)*, June 2008.
2. **Yuan Lan**, Yeng Chai Soh, Guang-Bin Huang, “A constructive enhancement for Online Sequential Extreme Learning Machine”, ”, *Proc. Int’l Joint Conf. Neural Networks (IJCNN’09)*, June 2009.
3. **Yuan Lan**, Yeng Chai Soh, Guang-Bin Huang, “Random Search Enhancement of Error Minimized Extreme Learning Machine”, *In European Symposium on Artificial Neural Networks (ESANN) 2010, Bruges, Belgium*, April 28-30th 2010.
4. Rui Zhang, **Yuan Lan**, Guang-Bin Huang, and Yeng Chai Soh, “Extreme Learning Machine with Adaptive Growth of Hidden Nodes and Incremental Updating of Output Weights”, *International Conference on Autonomous and Intelligent System (AIS 2011)*(Accepted), June 2011.

Bibliography

- [1] T. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.
- [2] E. Mjolsness and D. DeCoste, “Machine learning for science: State of the art and future prospects,” *Science*, vol. 293, no. 5537, pp. 2051–2055, 2001.
- [3] G. Claeskens and N. L. Hjort, *Model Selection and Model Averaging*. New York: Cambridge University Press, 2008.
- [4] R. Hoffmann, V. I. Minkin, and B. K. Carpenter, “Ockham’s razor and chemistry,” *HYLE International Journal for Philosophy of Chemistry*, vol. 3, pp. 3–28, 1997.
- [5] A. Zellner, H. A. Keuzenkamp, and M. M. eds., *Simplicity, Inference and Modelling: Keeping it Sophisticatedly Simple*. Cambridge: Cambridge University Press, 2001.
- [6] M. Sewell, *Ensemble Learning*. Department of Computer Science, University College London, London, 2007.
- [7] K. Hornik, “Multilayer feedforward networks are universal approximators,” *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.
- [8] —, “Approximation capabilities of multilayer feedforward networks,” *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991.
- [9] —, “Some new results on neural network approximation,” *Neural Networks*, vol. 6, no. 9, pp. 1069–1072, 1993.

- [10] G.-B. Huang and L. Chen, "Convex incremental extreme learning machine," *Neurocomputing*, vol. 70, pp. 3056–3062, 2007.
- [11] —, "Enhanced random search based incremental extreme learning machine," *Neurocomputing*, vol. 71, pp. 3060–3068, 2008.
- [12] 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*, vol. 17, no. 4, pp. 879–892, 2006.
- [13] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, pp. 489–501, 2006.
- [14] —, "Extreme learning machine: A new learning scheme of feedforward neural networks," in *Proceedings of International Joint Conference on Neural Networks (IJCNN'04)*, vol. 2, Budapest, Hungary, 25-29 July, 2004, pp. 985–990.
- [15] H.-J. Rong, G.-B. Huang, N. Sundararajan, and P. Saratchandran, "Online sequential fuzzy extreme learning machine for function approximation and classification problems," *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, vol. 39, no. 4, pp. 1067–1072, 2009.
- [16] T. Kim and T. Adali, "Approximation by fully complex multilayer perceptrons," *Neural Computation*, vol. 15, pp. 1641–1666, 2003.
- [17] M.-B. Li, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "Full complex extreme learning machine," *Neurocomputing*, vol. 68, pp. 306–314, 2005.
- [18] L. Breiman, "Hinging hyperplanes for regression, classification, and function approximation," *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 999–1013, 1993.
- [19] C. L. Giles and T. Maxwell, "Learning, invariance, and generalization in high-order neural networks," *Applied Optics*, vol. 26, no. 23, pp. 4972–4978, 1987.

- [20] Y. Shin and J. Ghosh, "Approximation of multivariate functions using ridge polynomial networks," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN 2002)*, Baltimore, USA, June, 2002, pp. 380–385.
- [21] I. Daubechies, "Orthonormal bases of compactly supported wavelets," *Communications on Pure and Applied Mathematics*, vol. 41, pp. 909–996, 1988.
- [22] —, "The wavelet transform, time-frequency localization and signal analysis," *IEEE Transactions on Information Theory*, vol. 36, no. 5, pp. 961–1005, 1990.
- [23] F. Han and D.-S. Huang, "Improved extreme learning machine for function approximation by encoding a priori information," *Neurocomputing*, vol. 69, pp. 2369–2373, 2006.
- [24] S. Haykin, *Neuron Networks: A Comprehensive Foundation*, second edition ed. Prentice Hall, 1998.
- [25] R. Collobert, S. Bengio, and Y. Bengio, "A parallel mixtures of svms for very large scale problems," *Neural Computation*, vol. 14, pp. 1105–1114, 2002.
- [26] S. Geman, E. Bienenstock, and R. Doursat, "Neural networks and the bias/variance dilemma," *Neural Computation*, vol. 4, pp. 1–58, 1992.
- [27] L. K. Hansen and P. Salamon, "Neural network ensemble," *IEEE Transaction Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 993–1001, 1990.
- [28] 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*, vol. 20, no. 8, pp. 1352–1357, 2009.
- [29] H.-J. Rong, Y.-S. Ong, A.-H. Tan, and Z. Zhu, "A fast pruned-extreme learning machine for classification problem," *Neurocomputing*, vol. 72, pp. 359–366, 2008.
- [30] Y. Miche, A. Sorjamaa, and A. Lendasse, "OP-ELM: Theory, experiments and a toolbox." in *ICANN (I)*, ser. Lecture Notes in Computer Science, V. Kurkov, R. Neruda, and J. Koutník, Eds., vol. 5163. Springer, 2008, pp. 145–154.

- [Online]. Available: <http://dblp.uni-trier.de/db/conf/icann/icann2008-1.html#MicheSL08>
- [31] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.
- [32] M. Minsky and S. Papert, *Perceptrons*. MIT Press, Cambridge, 1969.
- [33] E. B. Baum, "On the capability of multilayer perceptrons," *Journal of Complexity*, vol. 4, no. 3, pp. 193–215, 1988.
- [34] T.-Q. Wang, S.-J. Wang, and H.-X. Zhang, "Dynamic extreme learning machine: A learning algorithm for neural network with elastic output structure," in *Proceedings of 2009 International Symposium on Intelligent Information Systems and Applications*, 2009, pp. 271–275.
- [35] S. Suresh, R. V. Babu, and H. J. Kim, "No-reference image quality assessment using modified extreme learning machine classifier," *Applied Soft Computing*, vol. 9, no. 2, pp. 541–552, 2009.
- [36] G. Duan, Z. Huang, and J. Wang, "Extreme learning machine for bank clients classification," in *Proceedings of 2009 International Conference on Information Management, Innovation Management and Industrial Engineering*, 2009, pp. 496–499.
- [37] 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*, vol. 46, no. 1, pp. 411–419, 2008.
- [38] J. Kim, H. S. Shin, and K. Shin, "Robust algorithm for arrhythmia classification in ecg using extreme learning machine," *Biomedical Engineering Online*, vol. 8, pp. 31–42, 2009.
- [39] Y. Lan, Y. C. Soh, and G.-B. Huang, "Extreme learning machine based bacterial protein subcellular localization prediction," in *Proceedings of 2008 IEEE International Joint Conference on Neural Networks*, 2008, pp. 1859–1863.

- [40] R.-X. Zhang, G.-B. Huang, and N. Sundararajan, "Multicategory classification using an extreme learning machine for microarray gene expression cancer diagnosis," *IEEE-ACM Transactions on Computational Biology and Bioinformatics*, vol. 4, no. 3, pp. 485–495, 2007.
- [41] 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*, vol. 47, no. 3, pp. 920–925, 2008.
- [42] X.-F. Hu, Z. Zhao, and S. Wang, "Multi-stage extreme learning machine for fault diagnosis on hydraulic tube tester," *Neural Computing and Applications*, vol. 17, no. 4, pp. 399–403, 2008.
- [43] A. H. Nizar, Z. Dong, and Y. Wang, "Power utility nontechnical loss analysis with extreme learning machine method," *IEEE Transaction on Power Systems*, vol. 23, no. 3, pp. 946–955, 2008.
- [44] M. Pal, "Extreme-learning-machine-based land cover classification," *International Journal of Remote Sensing*, vol. 30, no. 14, pp. 3835–3841, 2009.
- [45] C. W. T. Yeu, M. H. Lim, and G.-B. Huang, "Terrain modelling using machine learning methods," in *Proceedings of 2006 9th International Conference on Control, Automation, Robotics and Vision, VOLS 1-5*, 2006, pp. 2299–2302.
- [46] C. Zhai and J. Du, "Applying extreme learning machine to plant species identification," in *Proceedings of 2008 International Conference on Information and Automation, VOLS 1-4*, 2008, pp. 879–884.
- [47] Q. J. B. Loh and S. Emmanuel, "Elm for the classification of music genres," in *Proceedings of 2006 9th International Conference on Control, Automation, Robotics and Vision, VOLS 1-5*, 2006, pp. 2284–2289.
- [48] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Computation*, vol. 3, no. 2, pp. 246–257, 1991.

- [49] ———, “Approximation and radial-basis-function networks,” *Neural Computation*, vol. 5, no. 2, pp. 305–316, 1993.
- [50] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, “Multilayer feedforward networks with a nonpolynomial activation function can approximate any function,” *Neural Networks*, vol. 6, pp. 861–867, 1993.
- [51] 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, Neural Networks*, vol. 9, no. 1, pp. 224–229, 1998.
- [52] G.-B. Huang, “Learning capability and storage capacity of two-hidden-layer feedforward networks,” *IEEE Transactions on Neural Networks*, vol. 14, no. 2, pp. 274–281, 2003.
- [53] D. Serre, *Matrices: Theory and Applications*. Springer-Verlag New York, Inc, 2002.
- [54] C. R. Rao and S. K. Mitra, *Generalized Inverse of Matrices and its Applications*. New York: Wiley, 1971.
- [55] G.-B. Huang, Q.-Y. Zhu, K.-Z. Mao, C.-K. Siew, P. Saratchandran, and N. Sundararajan, “Can threshold networks be trained directly?” *IEEE Transactions on Circuits and Systems II*, vol. 53, no. 3, pp. 187–191, 2006.
- [56] 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*, vol. 44, no. 2, pp. 525–536, 1998.
- [57] T. Hastie, R. Tibshirani, and J. Friedman, *The elements of statistical learning: data mining, inference and prediction*. Springer Verlag: New York, 2001.
- [58] T. Kim and T. Adali, “Approximation by fully complex multilayer perceptrons,” *Neural Computation*, vol. 15, no. 7, pp. 1641–1666, 2003.

- [59] Q. Zhu, A. K. Qin, P. N. Suganthan, and G.-B. Huang, "Evolutionary extreme learning machine," *Pattern Recognition*, vol. 38, pp. 1759–1763, 2005.
- [60] R. Storn and K. Price, "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.
- [61] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate online sequential learning algorithm for feedforward networks," *IEEE Transactions on Neural Networks*, vol. 17, no. 6, pp. 1411–1423, 2006.
- [62] H.-W. Chen, H.-H. Chen, and X. Nian, "Ensembling extreme learning machine," *Lecture Notes in Computer Science*, vol. 4491, pp. 1069–1076, 2007.
- [63] M. V. Heeswijk, Y. Miche, T. L. Knuutila, P. A. J. Hilbers, T. Honkela, E. Oja, and A. Lendasse, "Adaptive ensemble models of extreme learning machines for time series prediction," in *Lecture Notes in Computer Science*, C. Alippi, M. Polycarpou, C. Panayiotou, and G. Ellinas, Eds., 2009, vol. 5769, no. 305–314.
- [64] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 930–945, 1993.
- [65] J. Yin, F. Dong, and N. Wang, "Modified gram-schmidt algorithm for extreme learning machine," in *Proceedings of Second International Symposium on Computational Intelligence and Design*, 2009, pp. 517–520.
- [66] T. Simila and J. Tikka, "Multirespinse sparse regression with application to multi-dimensional scaling," in *Proceedings in Artificial Neural Networks: Fornal Models and Their Applications, ICANN 2005*, vol. 3697, 2005, pp. 97–102.
- [67] J. Cao, Z. Lin, and G. bin Huang, "Composite function wavelet neural networks with extreme learning machine," *Neurocomputing*, vol. 73, no. 7–9, pp. 1405–1416, 2010.

- [68] G.-B. Huang, X.-J. Ding, and H.-M. Zhou, "Optimization method based extreme learning machine for classification," *Neurocomputing*, vol. 74, pp. 155–163, 2010.
- [69] X.-L. Tang and M. Han, "Ternary reversible extreme learning machines: the incremental tri-training method for semi-supervised classification," *Knowledge and Information Systems*, vol. 23, no. 3, pp. 345–372, 2010.
- [70] E. K. P. Chong and S. H. Zak, *An introduction to optimization*. New York: Wiley, 2001.
- [71] G. H. Golub and C. F. V. Loan, *Matrix Computations, 3rd edition*. Baltimore, MD: The Johns Hopkins University Press, 1996.
- [72] S. Boyd, L. E. Ghaoui, E. Feron, and V. Balakrishnan, *Linear Matrix Inequalities in System and Control Theory*. Philadelphia, PA:SIAM, 1994.
- [73] J. Moody and P. J. Antsaklis, "The dependence identification neural network construction algorithm," *IEEE Transaction Neural Networks*, vol. 7, pp. 3–15, 1996.
- [74] E. Alpaydin, "Gal: Networks that grow when they learn and shrink when they forget," *International Computer Science Institute, TR 91-032, USA*, 1991.
- [75] D. J. C. Mackay, "The evidence framework applied to classification networks," *Neural Computation*, vol. 4, no. 5, pp. 720–736, 1992.
- [76] —, "A practical bayesian framework for backprop netowrks," *Neural Computation*, vol. 4, no. 5, pp. 448–472, 1992.
- [77] S. J. Hanson and L. Y. Pratt, "Comparing biases for minimal netowrk construction with backpropagation," in *Advances in neural information processing systems (1)*, D. S. Touretzky, Ed. Morgan Kaufmann Publishers Inc., 1989, pp. 177–185.
- [78] T.-Y. Kwok and D.-Y. Yeung, "Constructive algorithm for structure learning in feedforward neural networks for regression problems," *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 630–645, 1997.

- [79] R. Parekh, J. Yang, and V. Honavar, "Constructive neural-network learning algorithms for pattern classification," *IEEE Transactions on Neural Networks*, vol. 11, no. 2, pp. 436–451, 2000.
- [80] R. Reed, "Pruning algorithms - a survey," *IEEE Transaction on Neural Networks*, vol. 4, no. 5, pp. 740–747, 1993.
- [81] 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)*, D. S. Touretzky, Ed. Morgan Kaufmann Publishers Inc., 1989, pp. 107–115.
- [82] E. D. Karnin, "A simple procedure for pruning back-propagation trained neural networks," *IEEE Transaction on Neural Networks*, vol. 1, no. 2, pp. 239–242, 1990.
- [83] Y. L. Cun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in neural information processing systems (2)*, D. S. Touretzky, Ed. Morgan Kaufmann Publishers Inc., 1990, pp. 598–605.
- [84] Y. Chauvin, "A back-propagation algorithm with optimal use of hidden units," in *Advances in neural information processing systems (1)*, D. S. Touretzky, Ed. Morgan Kaufmann Publishers Inc., 1989, pp. 519–526.
- [85] C. Ji, R. R. Snapp, and D. Psaltis, "Generalizing smoothness constraints from discrete samples," *Neural Computation*, vol. 2, no. 2, pp. 188–197, 1990.
- [86] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman, "Generalization by weight-elimination with application to forecasting," in *Advances in neural information processing systems (3)*, R. Lippmann, J. Moody, and D. S. Touretzky, Eds. Morgan Kaufmann Publishers Inc., 1991, pp. 875–882.
- [87] S. J. Nowlan and G. E. Hinton, "Simplifying neural networks by soft weight-sharing," *Neural Computation*, vol. 4, no. 4, pp. 473–493, 1992.

- [88] D. C. Plaut, S. J. Nowlan, and G. E. Hinton, "Experiments on learning by back propagation," Carnegie-Mellon University, Tech. Rep. CMU-CS-86-126, 1986.
- [89] J. Sietsma and R. J. F. Dow, "Creating artificial neural networks that generalize," *Neural Networks*, vol. 4, no. 1, pp. 67–69, 1991.
- [90] J. K. Kruschke, "Creating local and distributed bottlenecks in hidden layers of back-propagation networks," in *Proceedings of 1988 Connectionist Models Summer School*, 1988, pp. 120–126.
- [91] D. Whitley and C. Bogart, "The evolution of connectivity: Pruning neural networks using genetic algorithms," in *Technical Report CS-89-113*, Colorado State University, Department of Computer Science, Fort Collins, 1989.
- [92] M. Hagiwara, "Removal of hidden units and weights for backpropagation network," in *Proceedings of 1993 International Joint conference of Neural Networks*, 1993, pp. 351–354.
- [93] S. Gallant, *Neural Network learning and expert systems*. Cambridge, MA: MIT Press, 1993.
- [94] V. G. Honavar and L. Uhr, "Generative learning structures and processes for generalized connectionist networks," *Information Sciences*, vol. 70, no. 1–2, pp. 75–108, 1993.
- [95] E. B. Baum, "A proposal for more powerful learning algorithms," *Neural Computation*, vol. 1, no. 2, pp. 201–207, 1989.
- [96] J.-H. Lin and J. S. Vitter, "Complexity results on learning by neural nets," *Machine Learning*, vol. 6, pp. 211–230, 1991.
- [97] T.-Y. Kwok and D.-Y. Yeung, "Objective function for training new hidden units in constructive neural networks," *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp. 1131–1148, 1997.

- [98] J. H. Friedman, "An overview of predictive learning and function approximation," in *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, J. H. Friedman and H. Wechsler, Eds. Berlin: Springer-Verlag, ASI Proc., Subseries F, 1994, pp. 1–61.
- [99] P. McCullagh and J. A. Nelder, *Generalized Linear Models*. London: Chapman and Hall, 2nd Edition, 1989.
- [100] E. D. Sontag, "Feedback stabilization using two-hidden-layer nets," *IEEE Transactions on Neural Networks*, vol. 3, pp. 981–990, 1992.
- [101] D. L. Chester, "Why two hidden layers are better than one," in *Proceedings of 1990 International Joint Conference on Neural Networks*, 1990, pp. 265–268.
- [102] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in neural information processing systems (2)*, D. S. Touretzky, Ed. Morgan Kaufmann Publishers Inc., 1990, pp. 524–532.
- [103] K. J. Lang and M. J. Witbrock, "Learning to tell two spirals appart," in *Proceedings of 1988 Connectionist Models Summer School, San Mateo, CA: Morgan Kaufmann*, 1988, pp. 52–59.
- [104] Y. L. Cun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, pp. 541–551, 1989.
- [105] H. Akaike, "Statistical predictor identification," *Annals of the Institute of Statistical Mathematics*, vol. 22, pp. 202–217, 1970.
- [106] —, "A new look at the statistical model identification," *IEEE Transactions on Automatic Control*, vol. AC-19, pp. 716–732, 1974.
- [107] A. R. Barron, "Predicted squared error: A criterion for automatic model selection," in *Self-Organizing Methods in Modeling*, S. Farlow, Ed. New York: Marcel Dekker, 1984.

- [108] P. Craven and G. Wahba, "Smoothing noisy data with spline functions: Estimating the correct degree of smoothing by the method of generalized cross-validation," *Numerische Mathematik*, vol. 31, no. 4, pp. 377–403, 2005.
- [109] C. L. Mallows, "Some comments on C_p ," *Technometrics*, vol. 42, no. 1, pp. 87–94, 2000.
- [110] J. Moody, "Prediction risk and architecture selection for neural networks," in *Statistics to Neural Networks - Theory and Pattern Recognition Applications*, ser. F, V. Cherkassky, J. H. Friedman, and H. Wechsler, Eds. Springer-Verlag, 1994, vol. 136 of NATA ASI, pp. 147–165.
- [111] M. Stone, "Cross validatory choice and assessment of statistical predictions," *Journal of the Royal Statistical Society, Series B*, vol. 36, no. 1, pp. 111–147, 1974.
- [112] A. S. Weigend and B. LeBaron, "Evaluating neural-network predictors by bootstrapping," in *Proceedings of 1994 International conference of Neural Information Processing*, 1994, pp. 1207–1212.
- [113] T. Ash, "Dynamic node creation in backpropagation networks," *Connection Science*, vol. 1, no. 4, pp. 365–375, 1989.
- [114] S. Chen, S. A. Billings, and W. Luo, "Orthogonal least squares methods and their application to non-linear system identification," *International Journal of Control*, vol. 50, no. 5, pp. 1873–1896, 1989.
- [115] J. H. Friedman and W. Stuetzle, "Projection pursuit regression," *Journal of the American Statistical Association*, vol. 76, no. 376, pp. 817–823, 1981.
- [116] J. Platt, "A resource-allocating network for function interpolation," *Neural Computation*, vol. 3, pp. 213–225, 1991.
- [117] S. Chen, C. F. N. Cowan, and P. M. Grant, "Orthogonal least squares learning algorithm for radial basis function networks," *IEEE Transactions on Neural Networks*, vol. 2, no. 2, pp. 302–309, 1991.

- [118] S. Chen and J. Wigger, "Fast orthogonal least squares algorithm for efficient subset model selection," *IEEE Transactions on Signal Processing*, vol. 43, no. 7, pp. 1713–1715, 1995.
- [119] Q. M. Zhu and S. A. Billings, "Fast orthogonal identification of nonlinear stochastic models and radial basis function neural networks," *International Journal of Control*, vol. 64, no. 5, pp. 871–886, 1996.
- [120] J. N. Hwang, S. R. Lay, and M. Maechler, "Regression modeling in backpropagation and projection pursuit learning," *IEEE Transactions on Neural Networks*, vol. 5, no. 3, pp. 342–353, 1994.
- [121] T.-Y. Kwok and D.-Y. Yeung, "Use of bias term in projection pursuit learning improves approximation and convergence properties," *IEEE Transaction on Neural Networks*, vol. 7, pp. 1168–1183, 1996.
- [122] L. K. Jones, "A simple lemma on greedy approximation in hilbert space and convergence rates for projection pursuit regression and neural network training," *The Annals of Statistics*, vol. 20, no. 1, pp. 608–613, 1992.
- [123] G. P. Drago and S. Ridella, "Convergence properties of cascade correlation in function approximation," *Neural Computing and Applications*, vol. 2, no. 3, pp. 142–147, 1994.
- [124] I. G. Smotroff, D. H. Friedman, and D. Connolly, "Large scale networks via self organizing hierarchical networks," in *Proceedings of SPIE Conference of Applied AI and Neural network*, April 1991, p. 544.
- [125] D. S. Phatak and I. Koren, "Connectivity and performance tradeoffs in the cascade correlation learning architecture," *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 930–935, 1994.
- [126] G.-B. Huang, P. Saratchandran, and N. Sundararajan, "An efficient sequential learning algorithm for growing and pruning RBF (GAP-RBF) networks," *IEEE Trans. Syst., Man, Cybern., B, Cybern.*, vol. 34, pp. 2284–2292, 2004.

- [127] —, “A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation,” *IEEE Transactions on Neural Networks*, vol. 16, pp. 57–67, 2005.
- [128] V. Kadirkamanathan and M. Niranjan, “A function estimation approach to sequential learning with neural networks,” *Neural Computation*, vol. 5, pp. 954–975, 1993.
- [129] Y.-W. Lu, N. Sundararajan, and P. Saratchandran, “A sequential learning scheme for function approximation using minimal radial basis function (RBF) neural networks,” *Neural Computation*, vol. 9, pp. 461–478, 1997.
- [130] —, “Performance evaluation of a sequential minimal radial basis function (RBF) neural network learning algorithm,” *IEEE Transactions on Neural Networks*, vol. 9, pp. 308–318, 1998.
- [131] A. S. Young, “The bivar criterion for selecting regressors,” *Technometrics*, vol. 24, pp. 181–189, 1982.
- [132] S. Weisberg, *Applied Linear Regression*. Wiley: New York, 1985.
- [133] R. Nishii, “Asumptotic properties of criteria for selection of variables in multiple regression,” *The Annals of Statistics*, vol. 12, no. 2, pp. 758–765, 1984.
- [134] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, “Efficient backprop,” *Lecture Notes Computational Science*, vol. 1524, pp. 9–50, 1998.
- [135] F. L. Minku, H. Inoue, and X. Yao, “Negative correlation in incremental learning,” *Natural Computing*, vol. 8, no. 2, pp. 289–320, 2009.
- [136] A. Asuncion and D. Newman, “UCI machine learning repository,” <http://www.ics.uci.edu/~mllearn/MLRepository.html>, 2007, University of California, Irvine, School of Information and Computer Sciences.
- [137] M. C. Mackey and L. Glass, “Oscillation and chaos in physiological control system,” *Science*, vol. 197, no. 4300, pp. 287–289, 1977.

- [138] Y. Jun and M. J. Er, “An enhanced online sequential extreme learning machine algorithm,” in *Proceedings of Control and Decision Conference (CCDC 2008)*, Yantai, Shandong, 2-4 July, 2008, pp. 2902–2907.
- [139] R. Parekh, J. Yang, and V. Honavar, “Constructive neural-network learning algorithms for pattern classification,” *IEEE Transactions on Neural Networks*, vol. 11, pp. 436–451, 2000.
- [140] R. R. Hocking, “The analysis and selection of variables in linear regression,” *Biometrics*, vol. 32, pp. 1–49, 1976.
- [141] K.-Z. Mao and S. A. Bilings, “Algorithms for minimal model structure detection in nonlinear dynamic system identification,” *International Journal of Control*, vol. 68, no. 2, pp. 311–330, 1997.
- [142] B. Efron, T. Hastie, I. Johnstone, and R. Tibshirani, “Least angle regression,” *Annals of statistics*, vol. 32, pp. 407–499, 2004.
- [143] B. A. Turlach, “Least angle regression: Discussion,” *Annals of statistics*, vol. 32, no. 2, pp. 481–490, 2004.
- [144] X. Yan and X. G. Su, *Linear Regression Analysis: Theory and Computing*. World scientific publishing Co. Pte. Ltd., 2009.
- [145] K. Li, G.-B. Huang, and S. S. Ge, “Fast construction of single hidden layer feed-forward networks,” in *Handbook of Natural Computing*, G. Rozenberg, T. H. W. Bäck, and J. N. Kok, Eds. Springer, to be published, 2010.
- [146] A. Sherstinsky and R. W. Picard, “On the efficiency of the orthogonal least squares training method for radial basis function networks,” *IEEE Transactions on Neural Networks*, vol. 7, no. 1, pp. 195–200, 1996.
- [147] Y. Lan, Y. C. Soh, and G.-B. Huang, “Constructive hidden nodes selection of extreme learning machine for regression,” *Accepted and to be published in Neurocomputing*, 2010.

- [148] K. Li, J.-S. Peng, and G. W. Irwin, "A fast nonlinear model identification method," *IEEE Transactions on Automatic Control*, vol. 50, no. 8, pp. 1211–1216, 2005.
- [149] K. Li, J.-X. Peng, and E.-W. Bai, "A two-stage algorithm for identification of nonlinear dynamic systems," *Automatica*, vol. 42, no. 7, pp. 1189–1197, 2006.
- [150] P.-K. Li, T.-T. Jin, and X.-X. Du, "Simple recursive algorithm for linear-in-the-parameters nonlinear model identification," *Science in China Series F: Information Sciences*, vol. 52, no. 10, pp. 1739–1745, 2009.
- [151] X. Hong, R. J. Mitchell, S. Chen, C. J. Harris, K. Li, and G. W. Irwin, "Model selection approaches for non-linear system identification: a review," *International Journal of Systems Science*, vol. 39, no. 10, pp. 925–946, 2008.
- [152] J. Peng, K. Li, and D. S. Huang, "A hybrid forward algorithm for RBF neural network construction," *IEEE Transactions on Neural Networks*, vol. 17, no. 11, pp. 1436–1451, 2006.
- [153] G. Bontempi, M. Birattari, and H. Bersini, "Recursive lazy learning for modeling and control," *European Conference on Machine Learning*, pp. 292–303, 1998.
- [154] R. Myers, *Classical and Modern Regression with Applications, 2nd edition*. Pacific Grove, CA, USA: Duxbury, 1990.