Incremental extreme learning machine

Chen, Lei

2007

Chen, L. (2007). Incremental extreme learning machine. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/3804>

<https://doi.org/10.32657/10356/3804>

Nanyang Technological University

Downloaded on 26 Apr 2023 16:52:48 SGT

Incremental Extreme Learning Machine

Chen Lei

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University

in fulfillment of the requirement for the degree of

Doctor of Philosophy

2006

Acknowledgements

I would like to express my gratitude to my supervisor Dr. Huang Guangbin. He is a kind and knowledgeable researcher and mentor. I was deeply impressed by his enthusiasm and perseverance on research. His stimulating suggestions and encouragement were tremendously helpful to my PhD research project during the last three years. I will always treat him as an important tutor in my life.

I am also grateful to all the staff and research students in the Division of ICIS, School of Electrical and Electronic Engineering, Nanyang Technological University, for their kind assistance and support.

Last but not least, I would also like to thank Associate Professor Siew Chee Kheong, Mr. Zhu Qinyu and Mr. Li Mingbin for the useful discussions and comments on my research work.

Contents

Acknowledgements	i
Summary	vii
List of Figures	xi
List of Tables	xv
1 Introduction	1
1.1 Motivation	1
1.2 Objectives	2
1.3 Major Contribution of the Thesis	2
1.4 Organization of the Thesis	5
2 Literature Review	7

2.1	Approximation Capability of Feedforward Neural Networks . .	8
2.2	Some Popular Learning Algorithms	14
2.2.1	Gradient-Descent Learning Algorithms	15
2.2.2	Support Vector Machine (SVM)	17
2.2.3	Extreme Learning Machine (ELM)	22
2.2.4	Ensemble of Extreme Learning Machines	27
2.3	Summary	34
3	Proposed New Universal Approximation Theory On Neural Networks	35
3.1	Introduction	36
3.2	Necessary Lemmas	39
3.3	Universal Approximation of SLFNs with Random Hidden Nodes (ELM)	46
3.4	Conclusions	55
4	Incremental Extreme Learning Machine (I-ELM)	57
4.1	Introduction	58

4.2	Proposed Incremental Extreme Learning Machine (I-ELM) . . .	61
4.3	Performance Evaluation	63
4.3.1	Comparison between I-ELM and other Popular Incremental Learning Algorithms	65
4.3.2	Comparison between I-ELM with SVR and Stochastic Gradient Descent BP	69
4.3.3	Performance Evaluation of I-ELM for Threshold Networks	73
4.4	Proposed Ensemble of Incremental Extreme Learning Machines with Random Additive Hidden Nodes	74
4.5	Discussion	83
4.6	Conclusions	84
5	Enhanced Incremental Extreme Learning Machine (EI-ELM)	87
5.1	Introduction	88
5.2	Proposed Enhanced Incremental Extreme Learning Machine (EI-ELM)	90
5.3	Performance Evaluation	95

5.3.1	Comparison between EI-ELM and I-ELM with the same number of hidden nodes	96
5.3.2	Comparison between EI-ELM and I-ELM with the dif- ferent number of hidden nodes	100
5.3.3	Effects of number of selecting trials k	104
5.4	Discussion	106
5.5	Conclusion	107
6	Convex Incremental Extreme Learning Machine (CI-ELM)	109
6.1	Introduction	110
6.2	Proposed Convex Incremental Extreme Learning Machine (CI- ELM)	112
6.3	Performance Evaluation	121
6.3.1	Benchmarking with Regression Problem	121
6.3.2	Function Approximation SinC Case	126
6.4	Discussion	127
6.5	Conclusion	129
7	Conclusions and Recommendations	130

7.1	Conclusions	130
7.2	Recommendations for Further Research	133
Author's Publications		135
Bibliography		136

Summary

Feedforward neural networks have attracted considerable attention in many fields mainly due to their approximation capability, i.e., neural networks can accurately approximate target functions with a finite number of neurons¹. According to conventional neural network theories, single-hidden layer feedforward networks (SLFNs) with additive or Radial Basis Function (RBF) hidden nodes are universal approximation when the parameters of the networks are allowed adjustable. However, as observed in most neural network implementations tuning the parameters of the networks may cause learning complicated and inefficient, and it may be difficult to train networks with non-differential activation functions such as threshold networks. Unlike conventional neural network theories, this thesis first proposes a new theory. This new theory shows that in order to let SLFNs work as universal approximators, one may simply randomly choose input-to-hidden nodes, and then we only need to adjust the output weights linking the hidden layer and the output layer. In such SLFNs implementations, the activation functions for additive nodes can be any bounded nonconstant piecewise continuous functions $g : R \rightarrow R$ or the activation functions for RBF nodes can be any integrable piecewise continuous functions $g : R \rightarrow R$ and $\int_R g(x)dx \neq 0$. Fur-

¹Neurons are also called nodes in the context of this thesis.

thermore, based on the new theory, we propose two incremental algorithms:

- 1) Incremental extreme learning machine (I-ELM): it freezes the parameters of hidden nodes once they have been tuned and need not further adjust them when a new node is added. Unlike conventional learning algorithms, I-ELM randomly generates hidden nodes, and then analytically determines the weights connecting with the outputs of the networks. In order to obtain a compact network, an ensemble of I-ELM is also proposed in this thesis which allows single I-ELM to be trained on each region of a dataset. The ensemble of these I-ELMs performs well for the dataset.
- 2) Convex I-ELM (CI-ELM): I-ELM does not recalculate the output weights of all the existing nodes when a new node is added, but CI-ELM shows that while retaining the same simplicity and efficiency, the convergence rate of I-ELM can be further improved by recalculating the output weights of the existing nodes based on a convex optimization method when a new hidden node is randomly added.

Some simulation results based on different kinds of real-world benchmark problems demonstrate that the above two new incremental methods can achieve good generalization performance at a fast speed.

List of Figures

2.1	Model of single-output TLFNs	28
2.2	Output of p -th quantizer	30
2.3	Model of multi-output TLFN	32
3.1	Bounded piecewise continuous functions $g(\mathbf{a} \cdot \mathbf{x} + b)$ and $g\left(\frac{\ \mathbf{x} - \mathbf{a}\ }{b}\right)$ can respectively be as close to $g(\mathbf{a}_0 \cdot \mathbf{x} + b_0)$ and $g\left(\frac{\ \mathbf{x} - \mathbf{a}_0\ }{b_0}\right)$ as possible by adjusting $(\mathbf{a}, b)$	44
3.2	Error residual function e_n is always orthogonal to g_n : $e_n \perp g_n$. The sequence $\{\ e_n\ \}$ are decreasing and bounded below by zero.	48

3.3	For the limit e^* of a subsequence of $\{e_n\}$, if $\ e^*\ = r \neq 0$ there exists a g^* which is not orthogonal to e^* . Then there exists e_{n_k+p+1} such that $\ e_{n_k+p+1}\ < \lim_{n \rightarrow +\infty} \ e_n\ = r$, which is contradictory to the fact that $\ e_{n_k+p+1}\ \geq r$. Thus r must be zero. For the sake of simplicity, $\theta_{(g^*, e^*)}$, $\theta_{(e_{n_k+p}, e^*)}$, $\theta_{(g_{n_k+p+1}, g^*)}$ and $\theta_{(e_{n_k+p}, g_{n_k+p+1})}$ are denoted by θ^* , $\theta^{(1)}$, $\theta^{(2)}$, and θ , respectively. $\theta \leq \theta^* + \theta^{(1)} + \theta^{(2)}$	50
4.1	Performance of the proposed SLFN learning algorithm with RBF nodes on some benchmark problems: (a) average testing RMSE, (b) training time (seconds).	67
4.2	Performance of the SLFN learning algorithm with 500 hardlimit additive nodes on some benchmark problems: average testing RMSE.	75
4.3	A compact TLFN constructed by L quantizers and L local learning machines which share common hidden nodes.	77
5.1	The testing error updating curves of EI-ELM and I-ELM . . .	97
5.2	Training time spent by EI-ELM is linearly increased with the number of hidden nodes to be added	100

5.3	Testing RMSE performance comparison between EI-ELM and I-ELM (with Sigmoid hidden nodes) for Abalone case	103
5.4	Effect of number of selecting trials k on the generalization performance of EI-ELM in Airplane case	104
5.5	Effect of number of selecting trials k on the stability of the generalization performance of EI-ELM in Airplane case	105
5.6	Testing RMSE updating progress with new hidden nodes added and different number of selecting trials k in Airplane case	105
6.1	Performance comparison (testing RMSE) between CI-ELM and I-ELM for Abalone case.	122
6.2	Performance of CI-ELM with Sigmoidal activation function for Census case and Gaussian activation function for Abalone case	126
6.3	Performance of the proposed SLFN learning algorithm with sig and sin nodes on the SinC problem: (a) Sigmoidal node, (b) Sin node.	128

List of Tables

2.1	Comparison on the approximation issues of neural networks	11
4.1	Specification of benchmark datasets.	64
4.2	Average testing RMSE of different algorithms: SLFN (with 200 nodes) with different activation functions (Sigmoid, Sin and RBF), and two popular sequential learning algorithms: RAN and MRAN.	68
4.3	The standard deviations (Dev) of average testing RMSE for different algorithms: SLFN (with 200 nodes) with different activation functions (Sigmoid, Sin and RBF), and two popular sequential learning algorithms: RAN and MRAN.	69

4.4	Training time (seconds) and network complexity comparison of different algorithms: SLFN with different activation functions (Sigmoid, Sin and RBF), and two popular sequential learning algorithms: RAN and MRAN. In all these case, 200 nodes are incrementally added to SLFNs.	70
4.5	Performance comparison of SLFN with incrementally added 500 nodes, stochastic gradient descent BP, and SVR. The close generalization performance (testing RMSE) obtained by different algorithms are underlined and the best results among three algorithms are shown in boldface.	71
4.6	Training time comparison of SLFN with incrementally added 500 nodes, stochastic gradient descent BP, and SVR.	72
4.7	Performance comparison between the approximated threshold network ($\lambda = 10$) trained by stochastic gradient descent BP and the true threshold networks trained by the proposed SLFN with 500 threshold nodes: $g(x) = -1_{x<0} + 1_{x\geq 0}$	73
4.8	Specification of benchmark datasets.	81

4.9	Performance comparison of TLFN with 100 Sin nodes and SLFN with 200 Sin nodes. The training times of TLFNs are the total time spent on separate SLFNs trained on different sub-regions and do not include the time spent on sorting training data.	82
5.1	Specification of 16 Benchmarking Regression Datasets	96
5.2	Performance comparison between EI-ELM and I-ELM (both with 200 Sigmoid hidden nodes)	98
5.3	Performance comparison between EI-ELM and I-ELM (both with 200 RBF hidden nodes)	99
5.4	Performance comparison between EI-ELM with 50 Sigmoid hidden nodes and I-ELM with 500 Sigmoid hidden nodes . . .	101
5.5	Performance comparison between EI-ELM with 50 RBF hidden nodes and I-ELM with 500 RBF hidden nodes	102
6.1	Specification of Benchmarking Datasets	121
6.2	Training time (seconds) comparison between CI-ELM and I-ELM with 200 hidden nodes.	123
6.3	Performance comparison (testing RMSE) between CI-ELM and I-ELM with 200 hidden nodes.	124

6.4	Number of neurons comparison between CI-ELM and I-ELM	
	with additive sigmoidal nodes	125
6.5	Number of neurons comparison between CI-ELM and I-ELM	
	with Gaussian nodes	125

Chapter 1

Introduction

1.1 Motivation

Neural networks have been successfully applied to some fields of research and application. Up to now, many different models of neural networks have been proposed. A major problem of neural networks and the learning methods is the computational burden involved in solving highly complex functions with a large number of variables in neural networks. However, fast response is often demanded in the practical applications. How to make neural networks learn real-time (online) is a challenging problem in the whole neural network development.

1.2 Objectives

In order to solve the real-time learning target, Huang, et al have proposed Extreme Learning Machine (ELM) [1]-[7], which can randomly choose the hidden node parameters and analytically determine the hidden-to-output weights of neural networks one time. Due to the random character, ELM has been proved as an effective method to make neural networks achieve a better generalization performance at a fast learning speed. Though some experimental results show that input weights and biases of additive hidden nodes or centers and impact factors of RBF hidden nodes need not be adjusted at all, so far there lacks a strict theoretical justification whether ELM with randomly generated hidden nodes can have universal approximation capability. In this thesis, we will answer the question, and rigorously prove that ELM can work as universal approximators. Furthermore, based on the constructive proof, this thesis also proposes several effective incremental learning methods.

1.3 Major Contribution of the Thesis

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

- 1) A new theory about the universal approximation capability of single-hidden layer feedforward neural networks (SLFNs) is proposed. According

to conventional neural network theories, SLFNs with additive or Radial Basis Function (RBF) hidden nodes are universal approximators when the parameters of the networks are allowed adjustable. However, as observed in some neural network implementations, tuning the parameters of the networks may cause learning complicated and inefficient, and it may be difficult to train networks with non-differential activation functions such as threshold networks. Unlike conventional neural network theories, this thesis provides an incremental constructive method. It demonstrates that in order to let SLFNs work as universal approximators, one may simply randomly choose hidden nodes and then only need to adjust the output weights linking the hidden layer and the output layer. In such SLFNs implementations, the activation functions for additive nodes can be any bounded nonconstant piecewise continuous functions $g : R \rightarrow R$ and the activation functions for RBF nodes can be any integrable piecewise continuous functions $g : R \rightarrow R$ and $\int_R g(x)dx \neq 0$. The proposed incremental method is not only efficient for SFLNs with continuous (including non-differentiable) activation functions but also for SLFNs with piecewise continuous (such as threshold) activation functions. Different from the traditional understanding of elementary learning elements - neurons, different type of computational hidden nodes other than neurons can be used in our work.

- 2) The proof of the new theory itself provides a practical incremental con-

- structive method, which actually shows an efficient way to construct an incremental feedforward network. This incremental method referred to as Incremental Extreme Learning Machine (I-ELM) randomly adds nodes to the hidden layer one by one, and meantime freezes the output weights of the existing hidden nodes when a new hidden node is added. Several simulation results based on different kinds of real-world benchmarking problems show that neural networks with I-ELM can provide good generalization performance at a fast learning speed. Moreover, based on I-ELM, an ensemble of I-ELM is proposed in this thesis. The new architecture allows a single I-ELM to be trained on each region of a dataset. An ensemble of these single I-ELMs will work on the dataset. While retaining the simplicity of I-ELM, it also reduces the number of hidden nodes.
- 3) In order to achieve more parsimonious network architecture than the original I-ELM, we provide a simple enhanced implementation of I-ELM (EI-ELM). EI-ELM randomly choose parameters several times for new nodes, from which we determine the best parameters leading to the smallest residual error, therefore EI-ELM can remove these useless nodes and improve the efficiency of neural networks. The experimental results on several benchmark real world regression problems will verify our conclusion.
 - 4) We propose an improved learning algorithm (CI-ELM) for I-ELM. We know that I-ELM randomly generates the hidden nodes and analytically calculates the output weights of SLFNs, but I-ELM does not recalculate

the output weights of all the existing nodes when a new node is added.

In order to achieve smaller residual error, while retaining the same simplicity and efficiency, we recalculate the output weights of the existing nodes based on a convex optimization method when a new node is added.

Further, we rigorously prove that CI-ELM can work as universal approximators too. Since it also adjusts the previous neurons, it can achieve faster convergence rate than I-ELM.

1.4 Organization of the Thesis

This thesis is organized as bellow:

Chapter 2 introduces the background of our research and reviews some typical learning algorithms for feedforward neural networks derived so far.

Chapter 3 proposes a new theory, which shows that neural networks with random hidden nodes can also work as function approximators.

Chapter 4 proposes the incremental extreme learning machine (I-ELM) and an ensemble of I-ELM, furthermore the corresponding generalization performance is studied thoroughly.

Chapter 5 proposes an enhanced learning algorithm (EI-ELM) for I-ELM, which chooses random hidden nodes after several trials.

1.4. Organization of the Thesis

6

Chapter 6 proposes an improved learning algorithm (CI-ELM) for I-ELM based on a convex iterative sequence.

Chapter 7 summarizes the thesis and gives some suggestions in my future research work.

Chapter 2

Literature Review

Neural networks have been successfully applied to several developments and applications, such as signal process, pattern recognition, medicine, speech production, speech recognition, business and function approximation, etc.

It is generally agreed that neural networks are the collection of models of computation based on biological motivations. Feedforward neural network, as an important model in the whole neural network research, has been studied.

I mainly focus on feedforward neural networks.

2.1. Approximation Capability of Feedforward Neural Networks 8

2.1 Approximation Capability of Feedforward Neural Networks

Neural networks have attracted considerable attention in many fields over the last few years. They have been successfully applied to an extraordinary range of practical areas, such as finance, medicine, engineering, geology, physics, prediction, classification or control, etc. The importance of neural networks is that they can offer a powerful and general framework for representing nonlinear mappings from input variables to output variables. From mathematical viewpoint, each neuron is an activation function. The approximation capability is always a critical issue in neural network study. In fact, the non-linear property of activation functions ensures the approximation capability of neural networks. So far, many researches about the approximation capability of neural networks have been investigated thoroughly[8]-[49].

Stone-Weierstrass theorem and Lusin theorem play important roles in the approximation capabilities of neural networks.

Stone-Weierstrass Theorem{P.212 of [50]} *Let $\mathbf{X}$ be a compact space and A an algebra of continuous real-valued functions on $\mathbf{X}$ that separates the points of $\mathbf{X}$ and contains the constant functions. Then given any continuous real-valued functions f on $\mathbf{X}$ and any $\varepsilon > 0$, there is a function g in A such that for all x in $\mathbf{X}$, we have $|g(x) - f(x)| < \varepsilon$. In other words, A is a dense*

2.1. Approximation Capability of Feedforward Neural Networks 9

subset of $C(\mathbf{X})$.

Stone-Weierstrass theorem explains that certain network architectures can approximate continuous target functions. Nevertheless, many popular functions, such as step functions or threshold functions are discontinuous. The feedforward neural networks with these functions also attract a lot of interest mainly due to their easy implementation in digital hardware. In order to cope with these discontinuous cases, Stone-Weierstrass theorem can be extended to bounded measurable functions by applying Lusin theorem.

Lusin Theorem{P.74 of [50]} *If f be a measurable real-valued function on an interval $[a, b]$. Then given $\delta > 0$, there is a continuous function φ on $[a, b]$ such that $m\{x : f(x) \neq \varphi(x)\} < \delta$.*

Lusin theorem extends the area of functions into the set of bounded measurable functions, which may include continuous functions and bounded functions that may have a countable number of discontinuous points.

It is well known that an input-to-output feedforward neural network (without hidden layer) is not capable of universal approximation[51]. On the other hand, four or more layer feedforward neural networks are rarely used in practice; furthermore, their universal approximation proof can be easily deduced from single-hidden-layer feedforward neural networks (SLFNs). Hence, almost all the work focus on the most challenging issue of the approximation capability of SLFNs. Based on different parameter combinations of

2.1. Approximation Capability of Feedforward Neural Networks¹⁰

activation functions, there are mainly two kinds of feedforward neural networks i.e., additive hidden nodes and RBF hidden nodes, respectively. Based on the two different models, there have been several studies concerning the approximation capabilities of feedforward neural networks, which show that certain classes of activation functions are dense in wide variety classes of target functions. Some classical results are summarized in Table 2.1.

For neural networks with additive hidden nodes, Cybenko[9] demonstrated that the sum of continuous discriminatory functions¹ is dense in the continuous space of d -dimensional unit cube I_d , $[0, 1]^d$. Hornik[8] rigorously proved that standard multilayer feedforward networks with squashing functions² are able to approximate any Borel measurable function. Hornik continued to extend the scope of activation functions in [12], where he showed that single-hidden layer feedforward neural networks (SLFNs) with bounded and nonconstant activation functions are dense in L^p . Moreover, if activation functions are continuous, bounded and nonconstant, SLFNs are dense in the space of continuous functions for all compact subsets X of $\mathbf{R}^d$. Leshno[18] improved the results of Hornik[12] and gave necessary and sufficient conditions for universal approximation, i.e., feedforward neural networks with

¹A function σ is discriminatory if for a measure $\mu \in M(I_d)$

$$\int_{I_d} \sigma(\mathbf{w}^T \mathbf{x} + b) d\mu(\mathbf{x}) = 0$$

for all $\mathbf{w} \in R^d$ and $b \in R$ implies that $\mu = 0$.

²A function $\varphi : R \rightarrow [0, 1]$ is a squashing function if it is non-decreasing,

$\lim_{x \rightarrow \infty} \varphi(x) = 1$, and $\lim_{x \rightarrow -\infty} \varphi(x) = 0$

2.1. Approximation Capability of Feedforward Neural Networks¹¹

Table 2.1: Comparison on the approximation issues of neural networks

Researchers	Target functions	Activation functions
Hornik, <i>et al.</i> [8]	Borel measurable	squashing functions
Cybenko[9]	$C([0, 1]^d)$	continuous discriminatory
Hornik[12]	L^p	bounded and nonconstant
Hornik[12]	$C(\mathbf{K})^I, \mathbf{K} \subset \mathbf{R}^d$	continuous, bounded and nonconstant
Park, <i>et al.</i> [13]	$C(\mathbf{R}^d)$	RBF with the same impact factor
Ito[14]	$C(\mathbf{R}^d)$	monotonic sigmoid
Hornik[17]	$C(\mathbf{K})$	Riemann integrable and nonpolynomial
Leshno, <i>et al.</i> [18]	$C(\mathbf{R}^d)$	nonpolynomial
Barron[19]	$f \in \text{conv}(G)^{II}, o(n^{-1/2})$	$\ g\ \leq b^{III}, g \in G$
Chen, <i>et al.</i> [21]	$C(\overline{\mathbf{R}}^d)$	generalized sigmoidal function
Igel'nik, <i>et al.</i> [22]	$C(\mathbf{K})$	absolutely integrable ^{IV}
Meir, <i>et al.</i> [38]	$f \in W_p^r(\mathbf{K})^V, o(n^{-\lambda})^{VI}$	expressed by wavelet functions ^{VII}

I: $C(\cdot)$: the set of continuous functions.

II: $\text{conv}(G)$: the closure of the convex hull of a set G , where G is a subset of a Hilbert space.

III: $b > 0$ is a constant value

IV: $\int_{\mathbf{R}} g^2(x) dx < \infty$

V: $W_p^r(K) = \left\{ f : \max_{0 \leq |\vec{k}| \leq r} \left\| \frac{\partial^{|\vec{k}|} f}{\partial x_1^{k_1} \dots \partial x_d^{k_d}} \right\|_{L^p(K)} < \infty \right\}$ where $|\vec{k}| = k_1 + \dots + k_d$ and r is any nonnegative integer.

VI: $\lambda > 0$ is a constant value

VII: cf. P.328 of [38]

a nonpolynomial activation function can approximate continuous functions.

Following their work, several authors have addressed further results[11], [21],

2.1. Approximation Capability of Feedforward Neural Networks¹²

[29], where activation functions are assumed to be monotonic sigmoid functions or generalized sigmoid functions. Ito[14] proved the uniform approximation capability of feedforward networks in $C(\overline{\mathbf{R}}^d)$, where activation functions are assumed to be the monotonic sigmoid function. Chen[21] proved the approximation capability of feedforward networks with generalized sigmoidal function³ in $C(\overline{\mathbf{R}}^d)$. Igelnik[22] further proved that any absolutely integrable activation functions are dense in compact spaces. For the convergence rates for the approximation error, several researchers have studied the issue for various classes of functions. Jones[16] first applied a convex iteration algorithm to neural networks, and proved the corresponding approximation property for linear combinations of sigmoidal functions. Barron[19] showed that neural networks constructed by the convex iteration can approximate smooth target functions within an integrated squared error of order $O(1/n)$. Further, Lee[26] extended this result to show that even though the target functions are not in the closure of the convex hull of the set of activation functions, neural networks still ensure the same approximation error rate. Following their work, under certain conditions, optimal convergent rates have also been demonstrated[42]. Meir[38], Romero[39], [41], Koiran[20], Lee[26], Donahue[27] and Kwok[28] also proposed some incremental algorithms and

³A function $\sigma : \mathbf{R} \rightarrow \mathbf{R}$ is called a generalized sigmoidal function, if the limits

$$\begin{cases} \lim_{x \rightarrow \infty} \sigma(x) = 1 \\ \lim_{x \rightarrow -\infty} \sigma(x) = 0 \end{cases}$$

2.1. Approximation Capability of Feedforward Neural Networks

illustrated the corresponding convergent rates.

For the neural networks with RBF hidden nodes, Park and Sandberg[13] rigorously proved that RBF networks having one hidden layer and the same smoothing factor in each kernel node are capable of universal approximation. Based on RBF networks, lots of researches are developed thoroughly[52]-[58]. Many optimization algorithms, such as gradient descent, orthogonal least squares learning algorithms [59]-[63] or others incremental RBF algorithms (RAN[64], MRAN[65]-[66], RANEKF[67], GAP-RBF[68], GGAP-RBF[69]), could be applied to training RBF networks. Now RBF networks have been successfully applied to many fields and demonstrated that RBF networks are an effective mechanism for approximating and estimating complex nonlinear target functions based on data.

In a word, for a finite of training samples, neural networks can obtain effective information to adjust their weights and bias so as to approximate target functions with any accuracy. Due to the robust approximation property, neural networks are able to be successfully applied to across an extraordinary range of practical areas, such as finance, medicine, engineering, geology, physics, prediction, classification or control, etc. It should be noted that the previous results on universal approximation of neural networks[8]-[69] are almost based on the conventional network model where the parameters of hidden nodes need to be tuned during training phase. Furthermore, some previous theoretical results may not be implementable either.

2.2 Some Popular Learning Algorithms

In fact, from the mathematical viewpoint, the nonlinear nature of activation functions ensures network approximation capability. However, the nonlinearity of neural networks brings some disadvantages. The cost of the nonlinearity is that the process of seeking the parameters in neural networks is a burden in nonlinear optimizations. The burden tends to be intensive and complicated computation. Moreover, the structural complexity of neural network would increase corresponding computation. Therefore, how to find an effective algorithm becomes a hot and challenging issue in neural network studies. The difficulty is to choose a model, its degree of freedom and the free parameters, while maintaining a good prediction performance on the test data, i.e. the outputs should match the unknown target values. This objective is also known as to generalize to unknown data (generalization performance). All the previous studies almost try to use every possible optimization techniques to seek the optimal parameters of neural networks such that neural networks can achieve a better generalization performance. In this section, we will introduce some classical algorithms applied in feedforward neural networks, which are used to compare with our algorithm in the later chapters.

2.2.1 Gradient-Descent Learning Algorithms

Many gradient-based learning algorithms were proposed in last two decades. Back-propagation (BP) algorithm [82] is one of the most popular learning algorithms. BP algorithm has been applied successfully to solving some difficult problems.

BP algorithm consists of two processes based on the error correction learning rule: a forward process and a backward process. In the forward process, inputs are applied to the input layer nodes of neural networks, and all the weights are pre-fixed. The effect of these inputs propagates through networks layer by layer. Finally, corresponding outputs are produced as the actual responses of networks. During the backward process, those weights are all adjusted in accordance with an error-correction rule. Those actual responses of networks are subtracted from the desired response of networks; therefore we can obtain error signals. All the error signals are then propagated backward through the whole networks. The name "back-propagation" is coming from this reason.

We sum all the squared error signals, and call the corresponding results as the cost function. Then we update the weights in term of the following formula:

$$\begin{aligned}\mathbf{W}(t+1) &= \mathbf{W}(t) + \eta \Delta \mathbf{E}(t) \\ &= \mathbf{W}(t) - \eta \frac{\partial \mathbf{E}(t)}{\partial \mathbf{W}}\end{aligned}\tag{2.1}$$

where the parameter η indicates learning rate, $\mathbf{W}(t)$ is the current weight vector at the t -th iteration, and $\Delta \mathbf{E}(t)$ denotes the current gradient of the cost function at the t -th iteration. When the stopping criteria are satisfied, the iteration stops.

Since original BP needs to compute the gradient of cost function repeatedly, it may be particularly slow for multilayer networks, especially when the cost function is a typically high order nonlinear function with many parameters. Moreover, the convergence of BP may be flat, or the convergence of BP does not even occur at all. Due to the complexity of nonlinear functions, BP would lead to local minima. Therefore, a number of tricks[83] -[91] are applied to BP so as to greatly improve the chances of finding a good solution or remarkably decrease the learning time. Next, we briefly introduce one variation of BP, stochastic BP[92], which will be used in the later chapters.

For stochastic BP, an estimate of the true gradient is computed based on errors. In another word, the weights may move down along the random gradient for every iteration. Therefore, it may result in better solutions due to avoiding the probability of local minimum. In my research, our algorithms also have the similar thinking, i.e., randomly choose hidden nodes. Stochastic BP and our fast learning algorithms are both stochastic learning method. The advantages of stochastic learning are:

1. stochastic learning is usually much faster than batch learning.

2. stochastic learning avoids local minimum solutions.
3. stochastic learning usually results in better solutions.

2.2.2 Support Vector Machine (SVM)

In this section, we briefly review another popular and successful learning machine, support vector machine (SVM)[93]. The main tasks in machine learning are learning and prediction. Learning means to find a proper model and optimal parameters in a so called training dataset with known target values such that the model can be applied to future, unknown test data. SVM was proposed by Vapnik[93] and originally designed for binary classification. Now it and its variants have been effectively extended to multiclass classification and regression problems. SVM can provide high and efficient performance[93]-[135]. The main idea of SVM is that in SVM learning the data is internally transformed into a higher dimensional feature space, and then we construct an optimal separating hyperplane in this space by solving a quadratic programming problem. A set of so called support vectors is selected from the training set. These support vectors define a hyperplane in the feature space which is used to calculate the target values. If a simple linear SVM is used, the feature space is equal to the data space. In most SVM algorithms the transformation into the feature space is done implicitly by a kernel function, which qualifies a relation between two examples in feature

space.

Support Vector Machine for Classification

We assume that $\mathbf{X}$ is a compact set in a subset of $\mathbb{R}^d$. Consider N training samples $\{(\mathbf{x}_i, y_i)\}$, $i = 1, \dots, N$, where $\mathbf{x}_i \in \mathbf{X} \subseteq \mathbb{R}^d$ is the input pattern for the i th example and $y_i \in \{-1, 1\}$ is the corresponding desired target output. Normally, the process finding a suitable hyperplane in an input space to separate these samples is too restrictive. A solution is mapping the input space into a higher dimension feature space by nonlinear functions. According to Cover theorem⁴{P.258 of [136]}, we can find the corresponding optimal hyperplane in this feature space. Let $\phi(\mathbf{x})$ denote feature mapping from input space $\mathbb{R}^d$ to a feature space $\mathbf{H}$, where $\phi(\mathbf{x}) = [\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \dots, \phi_{d_1}(\mathbf{x})]^T$. Normally, $d_1 = \dim(\mathbf{H}) \gg \dim(\mathbf{X}) = n$. Then we wish to find a hyperplane or decision function

$$f(\mathbf{w}, \mathbf{x}) = \mathbf{w} \cdot \phi(\mathbf{x}) + b$$

defined by the pair $(\mathbf{w}, b)$ such that we can separate the points $\mathbf{x}_i$, i.e., the inequalities

$$\begin{cases} \mathbf{w} \cdot \phi(\mathbf{x}_i) + b \geq +1, \text{ for } y_i = +1 \\ \mathbf{w} \cdot \phi(\mathbf{x}_i) + b \leq -1, \text{ for } y_i = -1 \end{cases}$$

⁴Cover Theorem: A complex pattern-classification problem case in a high-dimensional space nonlinearly is more likely to be linearly separable than in a low-dimensional space.

2.2. Some Popular Learning Algorithms

19

are valid for all elements of the training set. The above inequalities can also be wrote as another form:

$$y_i (\mathbf{w} \cdot \phi(\mathbf{x}_i) + b) \geq 1, i = 1, \dots, N$$

The above optimal hyperplane problem is then regarded as finding a decision function with a small $\mathbf{w}$ to maximize the margin, i.e.,

$$\min \frac{1}{2} \|\mathbf{w}\|^2 \quad (2.2)$$

subject to

$$y_i (\mathbf{w} \cdot \phi(\mathbf{x}_i) + b) \geq 1, i = 1, \dots, N$$

It is a QP problem, which can be solved by constructing Lagrangian multiplier to search the optimal hyperplane. Then for given the training sample $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, we find the Lagrange multipliers $\{\alpha_i\}_{i=1}^N$ that maximize the objective function

$$\max W(\alpha) = \sum_i^N \alpha_i - \frac{1}{2} \sum_i^N \sum_j^N \alpha_i \alpha_j y_i y_j \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$$

subject to

$$\begin{aligned} \sum_i^N \alpha_i y_i &= 0 \\ \alpha_i &\geq 0, i = 1, \dots, N \end{aligned}$$

However, we do not know any knowledge of ϕ . Hence, in order to be practical for this method, a key characteristic of SVM is that one can obtain $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle$ without computing $\phi(\mathbf{x}_i)$ and $\phi(\mathbf{x}_j)$. This is achieved by using

a kernel function $k(\cdot, \cdot)$ that can compute the dot product of the data points in feature space, i.e.,

$$k(\mathbf{x}_i, \mathbf{x}_j) = \langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle \quad (2.3)$$

which satisfies the Mercer's theorem[136].

Therefore, the corresponding decision function is

$$f_N(\mathbf{x}) = \sum_{i=1}^N \alpha_i y_i \langle \phi(\mathbf{x}), \phi(\mathbf{x}_i) \rangle = \sum_{i=1}^N \alpha_i y_i k(\mathbf{x}, \mathbf{x}_i)$$

Support Vector Machine for Regression

Sometimes, after being transformed into feature space, we still cannot find a decision function to linearly separate training data. We can consider this case with error tolerance, and then the corresponding problem can be regarded as a regression problem. Under this case, SVM is named as SVR. Next, we will briefly introduce SVR.

Given a set of training samples $\{(\mathbf{x}_i, y_i)\}$, $i = 1, \dots, N$, where $\mathbf{x}_i \in \mathbf{X} \subseteq \mathbb{R}^d$ and $y_i \in \mathbb{R}$, the previous analysis can be generalized by introducing some nonnegative variables $\xi_i \geq 0$. Then the optimization problem can be formulated as

$$\min \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N (\xi_i + \xi_i^*) \quad (2.4)$$

subject to

$$\begin{cases} y_i - f(x_i) \leq \epsilon + \xi_i \\ f(x_i) - y_i \leq \epsilon + \xi_i^*, i = 1, \dots, N \\ \xi_i, \xi_i^* \geq 0 \end{cases}$$

where $C > 0$ is a constant, $\epsilon > 0$ is a small positive number, and ξ_i, ξ_i^* are error.

Then the Lagrange multipliers $\{\alpha_i\}_{i=1}^N$ maximize the objective function

$$W(\alpha) = \sum_{i=1}^N (\alpha_i - \alpha_i^*) y_i - \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N (\alpha_i - \alpha_i^*) (\alpha_j - \alpha_j^*) \langle \phi(x_i), \phi(x_j) \rangle - \epsilon \sum_{i=1}^N (\alpha_i + \alpha_i^*) \quad (2.5)$$

subject to

$$\sum_{i=1}^N (\alpha_i - \alpha_i^*) = 0, \alpha_i, \alpha_i^* \in [0, C], i = 1, \dots, N$$

Next we can get the linear decision surface computed in the feature space

as:

$$f(\mathbf{x}) = \sum_i^N (\alpha_i - \alpha_i^*) \langle \phi(\mathbf{x}), \phi(\mathbf{x}_i) \rangle + b = \sum_i^N (\alpha_i - \alpha_i^*) k(\mathbf{x}, \mathbf{x}_i) + b$$

The Kuhn-Tucker theorem plays an important role in the theory of SVM.

According to this theorem, the solution $\bar{\alpha}_i$ and $\bar{\alpha}_i^*$ satisfy

$$\bar{\alpha}_i^* (\epsilon + \bar{\xi}_i^* + y_i - \bar{\mathbf{w}} \mathbf{z}_i - \bar{b}) = 0 \quad (2.6)$$

$$(C - \bar{\alpha}_i^*) \bar{\xi}_i^* = 0 \quad (2.7)$$

$$\bar{\alpha}_i (\epsilon + \bar{\xi}_i - y_i + \bar{\mathbf{w}} \mathbf{z}_i + \bar{b}) = 0 \quad (2.8)$$

$$(C - \bar{\alpha}_i)\bar{\xi}_i = 0 \quad (2.9)$$

The point x_i with the corresponding $\bar{\alpha}_i > 0$ or $\bar{\alpha}_i^* > 0$ is called a support vector. But there are also two types of support vectors in a nonseparable case. The one with corresponding $0 < \bar{\alpha}_i < C$ or $0 < \bar{\alpha}_i^* < C$ lies on the margin of the hyperplane. The one with corresponding $\bar{\alpha}_i = C$ or $\bar{\alpha}_i^* = C$ is misclassified, and we refer to such support vectors as errors. The points corresponding with $\bar{\alpha}_i = 0$ or $\bar{\alpha}_i^* = 0$ are classified correctly and clearly away from the decision margin.

2.2.3 Extreme Learning Machine (ELM)

In real applications, neural networks are usually trained in finite training datasets. For a finite training dataset, Huang and Babri[70] have shown that a single-hidden layer feedforward neural network (SLFN) with at most N hidden nodes and with almost any nonlinear activation function⁵ can exactly learn N distinct observations. It should be noted that the input weights (linking the input layer to the first hidden layer) and hidden layer biases need to be adjusted in all these previous theoretical research as well as in almost all practical learning algorithms of feedforward neural networks.

It has been shown [30], [137] that SLFNs (with N hidden nodes) with randomly chosen input weights and hidden layer biases (and such hidden

⁵Any bounded nonlinear activation function which has a limit at one infinity.[70]

nodes can thus be called random hidden nodes) can exactly learn N distinct observations. Unlike the popular thinking and most practical implementations that all the parameters of the feedforward networks need to be tuned, one may not necessarily adjust input weights and first hidden layer biases in applications.

Huang[7] rigorously proved that the input weights and hidden layer biases or RBF centers and impact factors of SLFNs can be randomly assigned if the activation functions in the hidden layer are infinitely differentiable. After the hidden nodes are chosen randomly, SLFNs can be simply considered as a linear system and the output weights (linking the hidden layer to the output layer) of SLFNs can be *analytically determined* through simple generalized inverse operation of the hidden layer output matrices. Based on this concept, Huang[7] proposed a simple learning algorithm for SLFNs called **extreme learning machine (ELM)** where the learning speed of ELM could be faster than some of traditional feedforward network learning algorithms like back-propagation algorithm while maintaining the same or better generalization performance[1]-[7]. Next we will briefly introduce the theory of ELM.

The output function of a standard single-hidden layer network (SLFN) with L hidden neurons is

$$f_L(\mathbf{x}) = \sum_{i=1}^L \beta_i G(\mathbf{a}_i, b_i, \mathbf{x}), \mathbf{x} \in \mathbf{R}^d, \mathbf{a}_i \in \mathbf{R}^d, b_i \in \mathbf{R}, \mathbf{x} \in \mathbf{R}^m \quad (2.10)$$

where $G(\mathbf{a}_i, b_i, \mathbf{x})$ is the output of the i th hidden neuron corresponding to the input $\mathbf{x}$, and $\beta_i = [\beta_{i1}, \beta_{i2}, \dots, \beta_{im}]^T$ is the weight vector connecting the i th hidden node and the output neurons.

For additive hidden neurons with activation function $g(x) : R \rightarrow R$, the output of the i th hidden neuron is

$$G(\mathbf{a}_i, b_i, \mathbf{x}) = g(\mathbf{a}_i \cdot \mathbf{x} + b_i), b_i \in R \quad (2.11)$$

where $\mathbf{a}_i$ is the input weight vector connecting the input neurons and the i th hidden neuron, and b_i is the bias of the i th hidden neuron.

For RBF nodes with activation function $g(x) : R \rightarrow R$, the output of the i th hidden neuron is

$$G(\mathbf{a}_i, b_i, \mathbf{x}) = g(b_i \|\mathbf{x} - \mathbf{a}_i\|), b_i \in R^+ \quad (2.12)$$

where $\mathbf{a}_i$ and b_i are the center and impact factor of i th RBF hidden neuron. R^+ is the set of all positive reals.

Given N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$, where $\mathbf{x}_i \in \mathbf{R}^d$ are the inputs and $\mathbf{t}_i \in \mathbf{R}^m$ are the targeted outputs. That a standard SLFN with L hidden neurons can approximate these N samples with zero error implies that for some β_i , $\mathbf{a}_i$ and b_i

$$\sum_{i=1}^L \beta_i G(\mathbf{a}_i, b_i, \mathbf{x}_j) = \mathbf{t}_j, \quad j = 1, \dots, N. \quad (2.13)$$

The above N equations can be compactly written as:

$$\mathbf{H}\beta = \mathbf{T} \quad (2.14)$$

where,

$$\mathbf{H}(\mathbf{a}_1, \dots, \mathbf{a}_L, b_1, \dots, b_L, \mathbf{x}_1, \dots, \mathbf{x}_N) = \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_1) & \cdots & G(\mathbf{a}_L, b_L, \mathbf{x}_1) \\ \vdots & \cdots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_N) & \cdots & G(\mathbf{a}_L, b_L, \mathbf{x}_N) \end{bmatrix}_{N \times L} \quad (2.15)$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_N^T \end{bmatrix}_{L \times m} \quad \text{and} \quad \mathbf{T} = \begin{bmatrix} \mathbf{t}_1^T \\ \vdots \\ \mathbf{t}_N^T \end{bmatrix}_{N \times m} \quad (2.16)$$

$\mathbf{H}$ defines the hidden layer output matrix of the SLFN with the i th column of $\mathbf{H}$ being the i th hidden neuron output with respect to inputs $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$.

In general, the number of hidden nodes is much less than the number of distinct training samples, i.e., $L \ll N$. This means that $\mathbf{H}$ is a nonsquare matrix, and there may exist $\mathbf{a}_i, b_i, \beta_i$ ($i = 1, \dots, L$) that satisfy the equality $\mathbf{H}\beta = \mathbf{T}$. However, as discussed in [7], the parameters of hidden neurons need not be tuned and can be randomly generated permanently based on any continuous probability distribution. This special consideration on the network parameters translates the training of an SLFN to simply finding a least square solution for the network's output bias matrix $\hat{\beta}$ of the linear system $\mathbf{H}\beta = \mathbf{T}$. The unique smallest norm least squares solution of this linear system is:

$$\hat{\beta} = \mathbf{H}^\dagger \mathbf{T} \quad (2.17)$$

where $\mathbf{H}^\dagger$ is the *Moore-Penrose generalized inverse* of hidden layer output

matrix $\mathbf{H}$ [138]-[139]. Thus, given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbf{R}^d, \mathbf{t}_i \in \mathbf{R}^m, i = 1, \dots, N\}$, activation function $g(x)$, and hidden node number L , the ELM algorithm can be summarized in the following three steps:

step 1) Randomly assign input weight $\mathbf{w}_i$ and bias $b_i, i = 1, \dots, \tilde{N}$.

step 2) Calculate the hidden layer output matrix $\mathbf{H}$.

step 3) Calculate the output weight β

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

Remark: Recently, Lowe[140] also proposed a similar method, where neural networks with RBF hidden nodes randomly select RBF centers from training data and determine the RBF impact factor from training data distribution, and then hidden-to-output weights are calculated by using Moore-Penrose generalized inverse. Although his method and ELM both calculate hidden-to-output weights by using Moore-Penrose generalized inverse, their difference is that ELM randomly select parameters independently from training data.

Although we randomly choose hidden nodes and then analytically determine the output weights connecting with the hidden layer and the output layer, ELM provides good generalization performance at a fast learning speed based on some batch of benchmark real world regression and classification problems.

2.2.4 Ensemble of Extreme Learning Machines

In some practical applications of neural networks, fast response to external events within a short time are highly demanded and expected, especially for the applications with large scale input data or when higher generalization performance is required. In order to make neural networks achieve good generalization performance within an expected fast response time, Huang [137] proposed a new constructive method that a two-hidden layer feedforward neural network (TLFN) can learn any N distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$ with any arbitrarily small error with at most $2\sqrt{(m+2)N} (\ll N)$ hidden neurons, which can reduce the space complexity of a network, especially for large scale problems and makes it feasible to implement large scale systems in a short time. In this method, the observations are divided into several groups, and each group is a small SLFN actually. It has been known that SLFNs with too few hidden neurons may be incapable of learning whereas SLFNs with too many hidden neurons may lead to overfitting. Some appropriate number of hidden nodes is set by users, which is normally far smaller than the number of training data. The corresponding algorithm is referred to as ELM, which can help eliminate overfitting in noisy datasets. In the ensemble of ELM, those small standard SLFN trained by ELM and some additional neuron quantizers are combined and linked to the output layer to form a special non-conventional TLFN which can finally represent all observations with arbitrary accuracy. Figure 2.1 is a single-out Huang's TLFN.

Figure 2.1: Model of single-output TLFNs

We briefly introduce TLFNs mentioned in Huang's paper[137]. Huang divides N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$ into L groups. Standard sigmoid function $g(x) = \frac{1}{1+e^{-x}}$ is used in all hidden neurons of Huang's network[137]. As shown in Figure 2.1, weight and bias vectors $\mathbf{w}_i$ and $b_i, (i = 1, \dots, N/L)$ can be determined almost arbitrarily, where $\mathbf{w}_i$ is the weight of the connection linking the input neurons to i -th neuron in the first hidden layer and b_i is the bias of the i -th neuron.

Theorem 2.2.1. [70], [137] *For any N distinct vectors $\mathbf{x}_i \in \mathbf{R}^d, i = 1, \dots, N$, for any vector $\mathbf{w}$ chosen from $\mathbf{R}^d$ according to any continuous probability distribution, then with probability one, the N inner products $\mathbf{w} \cdot \mathbf{x}_1, \dots, \mathbf{w} \cdot \mathbf{x}_N$ are different from each other.*

2.2. Some Popular Learning Algorithms

29

According to Theorem 2.2.1, when all the given $\mathbf{x}_i$ are distinctive, a vector $\mathbf{w}$ can be almost randomly chosen such that $\mathbf{w} \cdot \mathbf{x}_1, \dots, \mathbf{w} \cdot \mathbf{x}_N$ are all different. Without loss of generality, assume that the index i is reindexed so that $\mathbf{w} \cdot \mathbf{x}_1 < \mathbf{w} \cdot \mathbf{x}_2 < \dots < \mathbf{w} \cdot \mathbf{x}_N$. For simplicity, consider N to be an integral multiple of one positive integer L . It is obvious that the finite N can be separated into L groups consisting of N/L input vector each. Let us denote these L input vector groups as $V^{(1)}, \dots, V^{(L)}$,

$$V^{(p)} = \{\mathbf{x}_i | \mathbf{w} \cdot \mathbf{x}_{(p-1)N/L+1} \leq \mathbf{w} \cdot \mathbf{x}_i \leq \mathbf{w} \cdot \mathbf{x}_{pN/L}\}, \quad (2.19)$$

where $p = 1, \dots, L$.

The weights of connections linking the inputs to A -type neuron and B -type neuron can be chosen as $\bar{\mathbf{w}}_{A^{(p)}} = T \cdot \mathbf{w}$ and $\bar{\mathbf{w}}_{B^{(p)}} = -T \cdot \mathbf{w}$, respectively, where T is a large value. The biases $\bar{b}_{A^{(p)}}$ and $\bar{b}_{B^{(p)}}$ of hidden neurons $A^{(p)}$ and $B^{(p)}$ are analytically calculated. The bias of hidden neuron $A^{(p)}$ is chosen as

$$\bar{b}_{A^{(p)}} = -T(\mathbf{w} \cdot \mathbf{x}_{pN/L} + \frac{1}{2} \min_{i,j} \|\mathbf{w} \cdot \mathbf{x}_i - \mathbf{w} \cdot \mathbf{x}_j\|) \quad (2.20)$$

and

$$\bar{b}_{B^{(p)}} = T(\mathbf{w} \cdot \mathbf{x}_{(p-1)N/L+1} - \frac{1}{2} \min_{i,j} \|\mathbf{w} \cdot \mathbf{x}_i - \mathbf{w} \cdot \mathbf{x}_j\|). \quad (2.21)$$

Parameter T can be chosen large enough so that for any input $\mathbf{x}_i$ within input vector group $V^{(p)}$, $p = 1, \dots, L$, only the outputs of both hidden neurons $A^{(p)}$ and $B^{(p)}$, which are linked to the p -th output neuron, are almost zero. However, one of the outputs of hidden neurons $A^{(l)}$ and $B^{(l)}$, which are linked

to the l -th output neuron, is almost one, where $l \neq p$. In other words, T can be chosen large enough so that for each input vector group $V^{(p)}$, only the output of the p -th neural quantizer consisting of neurons $A^{(p)}$ and $B^{(p)}$ is almost zero, and the outputs of the other $L - 1$ quantizers are almost one. (cf. Figure 2.2)

Figure 2.2: Output of p -th quantizer

All the weights $\mathbf{w}_{AB}$ of the connections linking these $2L$ hidden neurons $A^{(p)}$ and $B^{(p)}$ to their corresponding second hidden-layer neurons are chosen same value $\mathbf{w}_{AB} = -U$, where parameter U is a positive value and can be set large enough so that the input of p -th output neuron from hidden neurons $A^{(p)}$ and $B^{(p)}$ (p -th quantizer) has small values ϵ_i for any input $\mathbf{x}_i$ within input vector group $V^{(p)}$ and large negative values E_i for any input within other vector groups $V^{(l)}, l \neq p$. We can make E_i and ϵ_i arbitrarily large and small, respectively, by setting T and U sufficiently large. E_i goes to negative infinity and ϵ_i goes to zero.

The weight matrix $\beta = [\beta^{(1)}, \dots, \beta^{(L)}]$, where $\beta^{(p)} = [\beta_1^{(p)}, \dots, \beta_{N/L}^{(p)}]^T, \beta_i^{(p)}$

denotes the weight connecting i -th first hidden-layer neuron to the second hidden-layer neuron, $i = 1, \dots, N$. $\beta^{(p)}$ can be determined as

$$\beta^{(p)} = (\mathbf{H}^{(p)})^{-1} \begin{bmatrix} t'_{(p-1)N/L+1} \\ \vdots \\ t'_{pN/L} \end{bmatrix} \quad (2.22)$$

where

$$t'_i = \ln\left(\frac{0.5 + t_i/C}{0.5 - t_i/C}\right) \quad (2.23)$$

and matrix $\mathbf{H}^{(p)} = [h_{ij}^{(p)}]$, where $h_{ij}^{(p)} = g(\mathbf{w}_j \cdot \mathbf{x}_{(p-1)N+i} + b_j)$, $p = 1, \dots, L$.

Before analytically calculating the weights β , we should determine a positive parameter C , such that $-0.5 < t/C < 0.5$. All the weights of connections linking to the output-layer are set to C , and bias of the output neuron is $-0.5C$.

The constructive structure with multi-output networks(cf. Figure 2.3) can be easily derived from the single-output network.

Lemma 2.2.1. [137] *For any N arbitrary distinct input vectors $\{\mathbf{x}_i | \mathbf{x}_i \in \mathbf{R}^d, i = 1, \dots, N\}$, for any weights $\mathbf{w}_i$ and b_i chosen from any intervals of $\mathbf{R}^d$ and $\mathbf{R}$, respectively, according to any continuous probability distribution, then with probability one, the hidden layer output matrix $\mathbf{H}$ of the network as shown in Figure is invertible for all input vector groups $V^{(p)}$, $p = 1, \dots, L$.*

According to Lemma2.2.1, given N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$, where $\mathbf{x}_i = [x_{i1}, x_{i2}, \dots, x_{in}]^T \in \mathbf{R}^d$ and $\mathbf{t}_i = [t_{i1}, t_{i2}, \dots, t_{im}]^T \in \mathbf{R}^m$, we

Figure 2.3: Model of multi-output TLFN

show in following section that TLFNs with $2\sqrt{(m+1)N}$ hidden neurons can learn them with negligible error.

In order to more precisely estimate the suitable values of the quantizer factors T and U , Huang [6] quantitatively gave the corresponding values. The following theorem (for the multioutput case) is give:

Theorem 2.2.2. [6] *Give arbitrarily small positive value $\epsilon < 1$, the quantizer factors T and U can be set as*

$$U = \ln \left(\frac{2\sqrt{m}CL}{\epsilon} - 1 \right) + \max_{\substack{p=1,\dots,L \\ q=1,\dots,N/L \\ s=1,\dots,L}} \|\mathbf{H}_q^{(p)} \cdot \beta^{(s)}\|_{\infty} \\ + \min_{i=1,\dots,N} \ln \left(\frac{(C+\epsilon/\sqrt{m}-2t_{ij})(C+2t_{ij})}{(C-\epsilon/\sqrt{m}+2t_{ij})(C-2t_{ij})} \right) \quad (2.24)$$

$$T = \frac{2 \ln \left(2U / \min_{i=1,\dots,N} \ln \left(\frac{(C+\epsilon/\sqrt{m}-2t_{ij})(C+2t_{ij})}{(C-\epsilon/\sqrt{m}+2t_{ij})(C-2t_{ij})} \right) - 1 \right)}{\min_{j=1,\dots,N-1} (\mathbf{w} \cdot \mathbf{x}_{j+1} - \mathbf{w} \cdot \mathbf{x}_j)} \quad (2.25)$$

such that $\forall \mathbf{x}_i \in V^{(k)}$, $k = 1, \dots, L$, the output $\mathbf{O}(\mathbf{x}_i)$ of Huang's network satisfies

$$\|\mathbf{O}(\mathbf{x}_i) - \mathbf{t}_i\| \leq \epsilon. \quad (2.26)$$

Huang's TLFN has several features:

1. it has larger first hidden layer and narrower second hidden layer, which comprises a standard SLFN and several neuron quantizers, each linking to the input layer and one or several neurons in the second hidden layer only;
2. all the weights of the connections linking the input layer to the first hidden layer can be simply pre-fixed and most of them can be assigned *randomly*;
3. the weights of the connections linking the first hidden layer and second hidden layer can be determined *analytically* at one time, instead of conventional or iterative adjustment method popularly adopted in most neural network learning algorithms nowadays;
4. the weights of the connections linking the second hidden layer and output layer can be simply *analytically* determined and remain as a constant value C ;
5. it may be trained by only adjusting weight size factor λ or C and quantizer factors T and U so as to optimize generalization performance.

Huang [6], [137] also analyzed that a good generalization performance may be achieved by the network.

2.3 Summary

In this chapter, we briefly review the approximation capability issue of feedforward neural networks, and list various learning algorithms for feedforward neural networks. Most of the traditional algorithms use some optimization techniques to seek network parameters such that neural networks can achieve better generalization performance, which normally lead to extensive computation. However, in many practical applications, fast response to external signals is often demanded. A new learning algorithm, Extreme Learning Machine (ELM), was proposed by Huang[1]-[3],[7],[137],[141], which could randomly choose the hidden node parameters and analytically determine the output weights of networks. Due to the random characteristic, neural networks can avoid make networks being trapped in the local minima, and provide good generalization performance at a fast learning speed. Though several experimental results show that the parameters of networks need not be adjusted at all, whether neural networks with random parameters are still universal approximation is unknown for us. In later chapters, we will answer the question, and rigorously prove that neural networks with random nodes can have universal approximation capabilities.

Chapter 3

Proposed New Universal Approximation Theory On Neural Networks

As mentioned in Chapter 2, some of conventional learning algorithms are usually far slower than required. Though ELM can provide good generalization performance for finite training samples at a fast learning speed due to its random characteristics, whether neural networks with random hidden nodes (also referred as ELM) can work as universal approximators is still unknown. In this chapter, we will prove that SLFNs with random hidden nodes can work as universal approximators.

3.1 Introduction

The widespread popularity of neural networks in many fields is mainly due to their ability to approximate complex nonlinear mappings directly from the input samples. Neural networks can provide models for a large class of natural and artificial phenomena that are difficult to handle using classical parametric techniques. Out of many kinds of neural networks single-hidden-layer feedforward neural networks (SLFNs) have been investigated more thoroughly.

Seen from the viewpoint of network architectures, two main SLFN network architectures have been investigating: 1) the SLFNs with additive hidden nodes and 2) radial basis function (RBF) networks which apply RBF nodes in the hidden layer. The output function of an SLFN with n additive nodes can be represented by

$$f_n(\mathbf{x}) = \sum_{i=1}^n \beta_i g(\mathbf{a}_i \cdot \mathbf{x} + b_i), \mathbf{a}_i \in \mathbf{R}^d, b_i, \beta_i \in R \quad (3.1)$$

where $\mathbf{a}_i$ is the weight vector connecting the input layer to the i -th hidden node, β_i the weight connecting the i -th hidden node to the output node, and g the hidden node activation function. $\mathbf{a}_i \cdot \mathbf{x}$ denotes the inner product of vectors $\mathbf{a}_i$ and $\mathbf{x}$ in $\mathbf{R}^d$. The RBF network is considered a specific SLFN which apply RBF nodes in its hidden layer. Each RBF node has its own centroid and impact factor, and its output is some radially symmetric function of the distance between the input and the center. The output function of an SLFN

with n RBF nodes can be represented by

$$f_n(\mathbf{x}) = \sum_{i=1}^n \beta_i g\left(\frac{\|\mathbf{x} - \mathbf{a}_i\|}{b_i}\right), \mathbf{a}_i \in \mathbf{R}^d, b_i \in R^+, \beta_i \in R \quad (3.2)$$

where $\mathbf{a}_i$ and b_i are the center and impact factor of i -th RBF node, β_i the weight connecting the i -th RBF hidden node to the output node. R^+ indicates the set of all positive real value. From a mathematical point of view, research on the approximation capabilities of SLFNs has focused on two aspects: universal approximation on compact input sets and approximation in a set consisting of finite number of training samples.

Hornik[12] proved that if the activation function is continuous, bounded and nonconstant, then continuous mappings can be approximated by SLFNs with additive hidden nodes over compact input sets. Leshno[18] improved the results of Hornik[12] and proved that SLFNs with additive hidden nodes and with a nonpolynomial activation function can approximate any continuous target functions. Regarding the universal approximation capability of RBF network, Park and Sandberg [13] has proved that RBF network with the same impact factor for suitably chosen kernels can approximate any continuous target function. For function approximation in a set consisting of N training samples $\{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in X \subset \mathbf{R}^d, \mathbf{t}_i \in \mathbf{R}^m, i = 1, 2, \dots, N\}$, some researchers have proposed constructive methods for SLFNs. For instance, Huang and Babri[70] shows that a SLFN with at most N hidden nodes and with almost any nonlinear activation function can learn N distinct observations with zero error. It should be noted that the network parameters

(including input weights and biases of additive nodes or centers and impact factor of RBF nodes) need to be adjusted in these previous research works.

It has been shown in our recent work [137] that with the input weights chosen randomly both SLFNs (with N additive hidden nodes) and two-hidden layer feedforward neural networks (TLFNs) (with at most $O(\sqrt{N})$ additive hidden nodes) can learn N distinct observations with arbitrarily small error. One may only simply adjust the rest parameters except for the input weights and first hidden layer biases in applications [1]-[7]. Baum[142] has claimed that (seen from simulations for SLFN with additive hidden nodes) one may fix the weights of the connections on one level and simply adjust the connections on the other level and no gain is possible by using an algorithm able to adjust the weights on both levels simultaneously. Furthermore, Huang, et al [2], [141] show that the centers and impact factors of the RBF nodes of RBF networks can also be randomly chosen. In fact, some simulation results on artificial and real large scale complex cases [1]-[7], [141] have shown that this method (with fixed network architectures) may be dramatically efficient and have good generalization performance. Some researchers [19], [28], [38] have proved in theory that for SLFN with additive hidden nodes one may freeze input weights and biases of hidden nodes once they have been *tuned* and needn't further adjust them when new nodes are added.

These research results may imply that in the many applications of SLFNs input weights and biases of additive hidden nodes or centers and impact

factors of RBF hidden nodes needn't be adjusted at all. However, so far there lacks a strict theoretical justification for this viewpoint and it isn't clear what functions can be chosen as activation functions of hidden layers in such SLFNs with randomly generated nodes.

This chapter mainly proves in theory that given any *bounded nonconstant piecewise continuous* activation function $g : R \rightarrow R$ for additive nodes or any *integrable piecewise continuous* activation function $g : R \rightarrow R$ (and $\int_R g(x)dx \neq 0$) for RBF nodes, the network sequence $\{f_n\}$ with randomly generated hidden nodes can converge to any continuous target function f by only properly adjusting the output weights linking the hidden layer to the output nodes.

3.2 Necessary Lemmas

We first prove in theory that single-hidden layer feedforward networks (SLFNs) with randomly generated additive or RBF nodes and with a broad of activation functions can universally approximate any continuous target functions in any compact subset X of the Euclidean space $\mathbf{R}^d$.

Without loss of generality for universal approximation, assume that the network has only one linear output node. It can be easily seen that the extension of all the analysis conducted in this chapter to multi nonlinear

output nodes cases is straightforward. SLFN network functions with n hidden nodes can be represented by

$$f_n(\mathbf{x}) = \sum_{i=1}^n \beta_i g_i(\mathbf{x}), \mathbf{x} \in \mathbf{R}^d, \beta_i \in R \quad (3.3)$$

where g_i denotes the hidden node output function. For additive nodes with activation function g , g_i is defined as

$$g_i = g(\mathbf{a}_i \cdot \mathbf{x} + b_i), \mathbf{a}_i \in \mathbf{R}^d, b_i \in R \quad (3.4)$$

and for RBF nodes with activation function g , g_i is defined as

$$g_i = g\left(\frac{\|\mathbf{x} - \mathbf{a}_i\|}{b_i}\right), \mathbf{a}_i \in \mathbf{R}^d, b_i \in R^+ \quad (3.5)$$

It has been shown [8], [12]-[13], [18], [71]-[73] that given activation function $g(x)$ satisfying certain mild conditions there exists a sequence of network functions $\{f_n\}$ approximating to any given continuous target function f . It should be noted that all parameters in any f_n in the sequence are required freely adjustable in all these previous works. Some researchers [19], [28], [38] proposed incremental methods for SLFNs with additive nodes which can allow input weights and biases of new hidden nodes to be tuned when added and then fixed after tuned. Romero [39] also proposed an incremental algorithm for SLFNs with additive and RBF nodes and showed that the input weights and biases or centers and impact factors of hidden nodes are tuned when added and can then be kept fixed.

The major objective of this chapter is to show that given any *bounded nonconstant piecewise continuous* activation function $g : R \rightarrow R$ for additive

nodes or any *integrable piecewise continuous* activation function $g : R \rightarrow R$ (and $\int_R g(x)dx \neq 0$) for RBF nodes, the network sequence $\{f_n\}$ with randomly generated hidden nodes can converge to any continuous target function f by only properly adjusting the output weights. Different from other increment algorithms for SLFNs with additive nodes [19], [28], [38], [39], [41] and incremental RBF networks [39], [64]-[69] new nodes needn't be tuned at the time they are added. Thus, the efficiency of SLFNs could be considerably increased.

Let $L^2(X)$ be a space of functions f on a compact subset X in the d -dimensional Euclidean space $\mathbf{R}^d$ such that $|f|^2$ are integrable, that is, $\int_X |f(\mathbf{x})|^2 d\mathbf{x} < \infty$. Let $L^2(\mathbf{R}^d)$ denoted by L^2 . Similar to Kwok and Yeung[28], for $u, v \in L^2(X)$, the inner product $\langle u, v \rangle$ is defined by

$$\langle u, v \rangle = \int_X u(\mathbf{x})v(\mathbf{x})d\mathbf{x} \quad (3.6)$$

The norm in $L^2(X)$ space will be denoted as $\|\cdot\|$, and the closeness between the network function f_n and the target function f is measured by the $L^2(X)$ distance:

$$\|f_n - f\| = \left[\int_X |f_n(\mathbf{x}) - f(\mathbf{x})|^2 d\mathbf{x} \right]^{1/2} \quad (3.7)$$

Definition 3.2.1. {P.334 of [143]} *A function $g(x) : R \rightarrow R$ is said to be piecewise continuous if it has only a finite number of discontinuities in any interval, and its left and right limits are defined (not necessarily equal) at each discontinuity.*

Definition 3.2.2. *The function sequence $\{g_n = g(\mathbf{a}_n \cdot \mathbf{x} + b_n)\}$ or $\left\{g_n = g\left(\frac{\|\mathbf{x} - \mathbf{a}_n\|}{b_n}\right)\right\}$ is said randomly generated if the corresponding parameters $(\mathbf{a}_n, b_n)$ are randomly generated from $\mathbf{R}^d \times R$ or $\mathbf{R}^d \times R^+$ based on a continuous sampling distribution probability.*

Remark 1: As done in our simulations, one may randomly generate sequence $\{g_n\}$ based on a uniform sampling distribution probability.

Definition 3.2.3. *A node is called a random node if its parameters $(\mathbf{a}, b)$ are randomly generated based on a continuous sampling distribution probability.*

Some lemmas which are used to prove our main theorem 3.3.1 are provided in this section.

Lemma 3.2.1. {P.81 of [144]} *The space of L^2 is complete.*

Lemma 3.2.2. {P.80 of [144]} *The sum of two functions of L^2 is an element of L^2 .*

Lemma 3.2.3. {Proposition 1 of [18]} *Given $g : R \rightarrow R$, $\text{span}\{g(\mathbf{a} \cdot \mathbf{x} + b) : (\mathbf{a}, b) \in \mathbf{R}^d \times R\}$ is dense in L^p for every $p \in [1, \infty)$, if and only if g is not a polynomial (almost everywhere).*

Lemma 3.2.4. [13] *Let $k : \mathbf{R}^d \rightarrow R$ be an integrable bounded function such that k is continuous (almost everywhere) and $\int_{\mathbf{R}^d} k(\mathbf{x}) d\mathbf{x} \neq 0$. Then $\text{span}\{k(\frac{\mathbf{x} - \mathbf{a}}{b}) : (\mathbf{a}, b) \in \mathbf{R}^d \times R^+\}$ is dense in L^p for every $p \in [1, \infty)$.*

Remark 2: As mentioned by Park and Sandberg [13] (p. 252) that there is no requirement of radial symmetry of the kernel function k in Lemma 3.2.4.

The radial basis functions case considered in this chapter is a specific case of Lemma 3.2.4: $k\left(\frac{\mathbf{x}-\mathbf{a}}{b}\right) = g\left(\frac{\|\mathbf{x}-\mathbf{a}\|}{b}\right)$.

Lemma 3.2.5. *Given a bounded nonconstant piecewise continuous function $g : R \rightarrow R$, we have*

$$\lim_{(\mathbf{a},b) \rightarrow (\mathbf{a}_0,b_0)} \|g(\mathbf{a} \cdot \mathbf{x} + b) - g(\mathbf{a}_0 \cdot \mathbf{x} + b_0)\| = 0, \forall (\mathbf{a}_0, b_0) \in \mathbf{R}^d \times R \quad (3.8)$$

and

$$\lim_{(\mathbf{a},b) \rightarrow (\mathbf{a}_0,b_0)} \left\| g\left(\frac{\|\mathbf{x}-\mathbf{a}\|}{b}\right) - g\left(\frac{\|\mathbf{x}-\mathbf{a}_0\|}{b_0}\right) \right\| = 0, \quad (3.9)$$

$$\forall (\mathbf{a}_0, b_0) \in \mathbf{R}^d \times R^+$$

Proof. Let $\hat{g}_0$ and $\hat{g}$ denote $g(\mathbf{a}_0 \cdot \mathbf{x} + b_0)$ and $g(\mathbf{a} \cdot \mathbf{x} + b)$ (or $g\left(\frac{\|\mathbf{x}-\mathbf{a}_0\|}{b_0}\right)$ and $g\left(\frac{\|\mathbf{x}-\mathbf{a}\|}{b}\right)$), respectively. Since g is bounded, suppose that $|g(x)| < M$ for all x . Given any small positive value ϵ , set $\gamma = \frac{1}{2} \left(\frac{\epsilon}{8M^2}\right)^{1/d}$. Without loss of generality, suppose that $g(x)$ has only one discontinuity $\check{x} \in R$, and $\hat{g}_0$ and $\hat{g}$ have single discontinuity points $\hat{\mathbf{x}}_0 \in \mathbf{R}^d$ and $\hat{\mathbf{x}} \in \mathbf{R}^d$, respectively. Thus, for given $(\mathbf{a}_0, b_0)$ and $(\mathbf{a}, b)$, we have $\mathbf{a}_0 \cdot \hat{\mathbf{x}}_0 + b_0 = \mathbf{a} \cdot \hat{\mathbf{x}} + b = \check{x}$ (or $\frac{\|\hat{\mathbf{x}}_0 - \mathbf{a}_0\|}{b_0} = \frac{\|\hat{\mathbf{x}} - \mathbf{a}\|}{b} = \check{x}$). Since $\mathbf{a} \cdot \mathbf{x} + b$ (or $\frac{\|\mathbf{x}-\mathbf{a}\|}{b}$) is a continuous function of $\mathbf{a}$ and b , there exists $\gamma_1 > 0$ such that when $\|(\mathbf{a}, b) - (\mathbf{a}_0, b_0)\| \leq \gamma_1$ we have $\|\hat{\mathbf{x}} - \hat{\mathbf{x}}_0\| < \gamma$. Let Δ denote a neighborhood of point $\hat{\mathbf{x}}_0$ (as shown in Figure 3.1):

$$\Delta = \{\mathbf{x} : \|\mathbf{x} - \hat{\mathbf{x}}_0\| < \gamma\} \quad (3.10)$$

3.2. Necessary Lemmas

44

Obviously, the discontinuous points ($\hat{\mathbf{x}}_0$ and $\hat{\mathbf{x}}$) of $\hat{g}_0$ and $\hat{g}$ are then located

Figure 3.1: Bounded piecewise continuous functions $g(\mathbf{a} \cdot \mathbf{x} + b)$ and $g\left(\frac{\|\mathbf{x} - \mathbf{a}\|}{b}\right)$ can respectively be as close to $g(\mathbf{a}_0 \cdot \mathbf{x} + b_0)$ and $g\left(\frac{\|\mathbf{x} - \mathbf{a}_0\|}{b_0}\right)$ as possible by adjusting $(\mathbf{a}, b)$.

in the small subset Δ . Thus, $\hat{g} - \hat{g}_0$ shall be continuous in the subset $X - \Delta$.

And further there exists $\gamma_2 < \gamma_1$ when $\|(\mathbf{a}, b) - (\mathbf{a}_0, b_0)\| \leq \gamma_2$ we have

$(\hat{g} - \hat{g}_0)^2 \leq \frac{\epsilon}{2 \int_X dx}$ for all $\mathbf{x} \in X - \Delta$. Thus, when $\|(\mathbf{a}, b) - (\mathbf{a}_0, b_0)\| \leq \gamma_2$ we

have

$$\begin{aligned}
& \int_X (\hat{g} - \hat{g}_0)^2 d\mathbf{x} \\
& \leq \int_{X-\Delta} (\hat{g} - \hat{g}_0)^2 d\mathbf{x} + \int_{\Delta} (\hat{g} - \hat{g}_0)^2 d\mathbf{x} \\
& < \frac{\epsilon}{2 \int_X d\mathbf{x}} \cdot \int_X d\mathbf{x} + (2\gamma)^d \cdot 4M^2 = \epsilon
\end{aligned} \tag{3.11}$$

This completes the proof. $\square$

Lemma 3.2.6. *Given a bounded nonconstant piecewise continuous function $g : R \rightarrow R$ and any given positive value $\hat{\theta} < \pi/2$ and any given g_0 , for any randomly generated function sequence $\{g_n\}$ there exists a positive integer M such that for any continuous segment $G_{(n,M)} = \{g_n, g_{n+1}, \dots, g_{n+M-1}\}$ ($n = 1, 2, \dots$), with probability as close to one as desired there exists $g_i \in G_{(n,M)}$ satisfying*

$$\theta_{(g_i, g_0)} < \hat{\theta} \tag{3.12}$$

where $\theta_{(g_i, g_0)}$ denotes the angle formed by g_i and g_0 .

Proof. Given a positive value $\hat{\theta} < \pi/2$, according to Lemma 3.2.5 there exists $\delta > 0$ such that $\forall (\mathbf{a}, b) \in Q_0 = \{(\mathbf{a}, b) : \|(\mathbf{a}, b) - (\mathbf{a}_0, b_0)\| < \delta\}$, $\|g - g_0\| < \|g_0\| \sin(\hat{\theta})$. Suppose that the continuous sampling distribution probability in $\mathbf{R}^d \times R$ (for the additive node case or $\mathbf{R}^d \times R^+$ for the RBF node case) is $p(\mathbf{x})$ and $\int_{\mathbf{R}^d \times R} p(\mathbf{x}) d\mathbf{x} = 1$ (or $\int_{\mathbf{R}^d \times R^+} p(\mathbf{x}) d\mathbf{x} = 1$). Then for any continuous segment $G_{(n,M)} = \{g_n, g_{n+1}, \dots, g_{n+M-1}\}$, the probability that among all the M elements the parameters $(\mathbf{a}, b)$ of some elements are sampled from the area Q_0 is $\int_{Q_0} p(\mathbf{x}) d\mathbf{x}$. That is, from probability point

3.3. Universal Approximation of SLFNs with Random Hidden Nodes (ELM) 46

of view there probably have $\sum_{i=0}^{M-1} \left(1 - \int_{Q_0} p(\mathbf{x}) d\mathbf{x}\right)^i \cdot \int_{Q_0} p(\mathbf{x}) d\mathbf{x}$ elements $g_i \in G_{(n,M)}$ such that $\|g_i - g_0\| < \|g_0\| \sin(\hat{\theta})$. Thus, M can be selected to make $\sum_{i=0}^{M-1} \left(1 - \int_{Q_0} p(\mathbf{x}) d\mathbf{x}\right)^i \cdot \int_{Q_0} p(\mathbf{x}) d\mathbf{x} \approx 1$, then for any continuous segment $G_{(n,M)} = \{g_n, g_{n+1}, \dots, g_{n+M-1}\}$ at least there exists an element $g_i \in G_{n,M}$ and $(\mathbf{a}_i, b_i) \in Q_0$. That is, there exists M such that $\exists g_i \in G_{(n,M)}$ ($\forall n$) satisfying $\sin(\theta_{(g_i, g_0)}) \leq \frac{\|g_i - g_0\|}{\|g_0\|} < \sin(\hat{\theta})$ where $\theta_{(g_i, g_0)} < \pi/2$, that is, $\theta_{(g_i, g_0)} < \hat{\theta}$. □

3.3 Universal Approximation of SLFNs with Random Hidden Nodes (ELM)

Let $e_n \equiv f - f_n$ denote the residual error function for the current network f_n with n hidden nodes where $f \in L^2(X)$ is the target function. We can now prove that given any *bounded nonconstant piecewise continuous* activation function $g : R \rightarrow R$ for additive nodes or *integrable piecewise continuous* activation function $g : R \rightarrow R$ (and $\int_R g(x) dx \neq 0$) for RBF nodes, for any continuous target function f and any randomly generated sequence $\{g_n\}$, $\lim_{n \rightarrow \infty} \|e_n\| = 0$ holds if $\beta_n = \frac{\langle e_{n-1}, g_n \rangle}{\|g_n\|^2}$.

Theorem 3.3.1. *Given any bounded nonconstant piecewise continuous function $g : R \rightarrow R$ for additive nodes or any integrable piecewise continuous function $g : R \rightarrow R$ and $\int_R g(x) dx \neq 0$ for RBF nodes, for any continu-*

3.3. Universal Approximation of SLFNs with Random Hidden Nodes (ELM)

47

ous target function f and any randomly generated function sequence $\{g_n\}$,

$\lim_{n \rightarrow \infty} \|f - f_n\| = 0$ holds with probability one if

$$\beta_n = \frac{\langle e_{n-1}, g_n \rangle}{\|g_n\|^2}. \quad (3.13)$$

Proof. We first prove that the expression $\|e_n\| = \|f - (f_{n-1} + \beta g_n)\|$ achieves its minimum iff $\beta = \beta_n = \frac{\langle e_{n-1}, g_n \rangle}{\|g_n\|^2}$ and the sequence $\{\|e_n\|\}$ converges. Then we further prove $\lim_{n \rightarrow +\infty} e_n = 0$.

(a) Since g is bounded nonconstant piecewise continuous and f is continuous, we have $g_n, f \in L^2(X)$ and $\|g_n\| = \int_X g_n^2 dx \neq 0$. According to Lemma 3.2.2, $e_n \in L^2(X)$. Let $\Delta = \|e_{n-1}\|^2 - \|e_n\|^2$, then we have

$$\begin{aligned} \Delta &= \|e_{n-1}\|^2 - \|f - (f_{n-1} + \beta_n g_n)\|^2 \\ &= \|e_{n-1}\|^2 - \|e_{n-1} - \beta_n g_n\|^2 \\ &= 2\beta_n \langle e_{n-1}, g_n \rangle - \beta_n^2 \|g_n\|^2 \\ &= \|g_n\|^2 \left(\frac{\langle e_{n-1}, g_n \rangle^2}{\|g_n\|^4} - \left(\beta_n - \frac{\langle e_{n-1}, g_n \rangle}{\|g_n\|^2} \right)^2 \right) \end{aligned} \quad (3.14)$$

Δ is maximized iff $\beta_n = \frac{\langle e_{n-1}, g_n \rangle}{\|g_n\|^2}$, meaning that $\|e_n\| = \|f - (f_{n-1} + \beta_n g_n)\|$ achieves its minimum iff $\beta = \beta_n = \frac{\langle e_{n-1}, g_n \rangle}{\|g_n\|^2}$. From the function space point of view, $e_n \perp g_n$ when $\|e_n\| = \|f - (f_{n-1} + \beta_n g_n)\|$ achieves its

3.3. Universal Approximation of SLFNs with Random Hidden Nodes (ELM)

48

minimum (cf. Figure 3.2). In fact, when $\beta_n = \frac{\langle e_{n-1}, g_n \rangle}{\|g_n\|^2}$ we have

$$\begin{aligned}
 \langle e_n, g_n \rangle &= \langle e_{n-1} - \beta_n g_n, g_n \rangle \\
 &= \langle e_{n-1}, g_n \rangle - \beta_n \langle g_n, g_n \rangle \\
 &= 0
 \end{aligned} \tag{3.15}$$

When $\beta_n = \frac{\langle e_{n-1}, g_n \rangle}{\|g_n\|^2}$, $\Delta = \Delta_{\max} = \frac{\langle e_{n-1}, g_n \rangle^2}{\|g_n\|^2} \geq 0$. So the sequence $\{\|e_n\|\}$ are decreasing and bounded below by zero and the sequence $\{\|e_n\|\}$ converges.

Figure 3.2: Error residual function e_n is always orthogonal to g_n : $e_n \perp g_n$.

The sequence $\{\|e_n\|\}$ are decreasing and bounded below by zero.

(b) We can now prove $\lim_{n \rightarrow +\infty} \|e_n\| = 0$ by contradiction method.

(b.1) Since the sequence $\{\|e_n\|\}$ converges, there exists $r \geq 0$ such as

3.3. Universal Approximation of SLFNs with Random Hidden Nodes (ELM)

49

$\lim_{n \rightarrow +\infty} \|e_n\| = r$. Suppose $r > 0$, then the sequence $\{\|e_n\|\}$ are decreasing and bounded below by r , that is $\|e_n\| \geq r$ for all positive integer n . Thus, $\forall \delta > 0$, $\exists \tilde{N}_1 > 0$ such that when $n > \tilde{N}_1$, we have $r \leq \|e_n\| < r + \delta$, which implies infinite number of e_n ($\forall n > \tilde{N}_1$) covered by a compact set. Thus, there exists a subsequence $\{e_{n_k}\}$ which converges to a limit denoted by e^* and $\|e^*\| = \lim_{k \rightarrow +\infty} \|e_{n_k}\| = r$. Since $e_{n_k} \in L^2(X)$ and $L^2(X)$ is complete (cf. Lemma 3.2.1), we have $e^* \in L^2(X)$.

Furthermore, there should exist $g^* = g(\mathbf{a}^* \cdot \mathbf{x} + b^*)$ or $g^* = g(\frac{\|\mathbf{x} - \mathbf{a}^*\|}{b^*})$ such that g^* is not orthogonal to e^* . Otherwise, e^* is orthogonal to $\text{span}\{g(\mathbf{a} \cdot \mathbf{x} + b) : (\mathbf{a}, b) \in \mathbf{R}^d \times R\}$ or $\text{span}\{g(\frac{\|\mathbf{x} - \mathbf{a}\|}{b}) : (\mathbf{a}, b) \in \mathbf{R}^d \times R^+\}$, which is contradictory to the fact that $\text{span}\{g(\mathbf{a} \cdot \mathbf{x} + b) : (\mathbf{a}, b) \in \mathbf{R}^d \times R\}$ (cf. Lemma 3.2.3) or $\text{span}\{g(\frac{\|\mathbf{x} - \mathbf{a}\|}{b}) : (\mathbf{a}, b) \in \mathbf{R}^d \times R^+\}$ is dense in $L^2(X)$ (cf. Lemma 3.2.4).

Let $\theta_{(g^*, e^*)}$ denote the angle formed by g^* and e^* and we have $0 \leq \theta_{(g^*, e^*)} < \pi/2$. (cf. Figure 3.3)

(b.2) Let $\theta_{(e_n, e^*)}$ be the angle formed by e_n and e^* and $0 \leq \theta_{(e_n, e^*)} \leq \pi/2$.

Choose any small positive value ξ such that $\xi < 1 - \sin(\theta_{(g^*, e^*)})$.

Since $\lim_{k \rightarrow +\infty} \|e_{n_k}\| = \|e^*\| = r$, $\exists \tilde{N}_2$ when $k > \tilde{N}_2$,

$$\begin{aligned} \|e_{n_k}\| &< \frac{r}{1 - \xi} \quad \text{and} \\ \theta_{(e_{n_k}, e^*)} &< \arcsin \left(\frac{1 - \xi - \sin(\theta_{(g^*, e^*)})}{2} \right). \end{aligned} \tag{3.16}$$

Since for all positive integer n , $e_n \perp g_n$ and $e_n = e_{n-1} - \beta_n g_n$ (cf.

3.3. Universal Approximation of SLFNs with Random Hidden Nodes (ELM)

50

Figure 3.3: For the limit e^* of a subsequence of $\{e_n\}$, if $\|e^*\| = r \neq 0$ there exists a g^* which is not orthogonal to e^* . Then there exists e_{n_k+p+1} such that $\|e_{n_k+p+1}\| < \lim_{n \rightarrow +\infty} \|e_n\| = r$, which is contradictory to the fact that $\|e_{n_k+p+1}\| \geq r$. Thus r must be zero. For the sake of simplicity, $\theta_{(g^*, e^*)}$, $\theta_{(e_{n_k+p}, e^*)}$, $\theta_{(g_{n_k+p+1}, g^*)}$ and $\theta_{(e_{n_k+p}, g_{n_k+p+1})}$ are denoted by θ^* , $\theta^{(1)}$, $\theta^{(2)}$, and θ , respectively. $\theta \leq \theta^* + \theta^{(1)} + \theta^{(2)}$.

equation (3.15)), we have $e_n \perp (e_{n-1} - e_n)$. Furthermore, we have

$$\begin{aligned}
 \|e_{n-1} - e_n\|^2 &= \|e_{n-1}\|^2 + \|e_n\|^2 - 2\langle e_{n-1}, e_n \rangle \\
 &= \|e_{n-1}\|^2 + \|e_n\|^2 \\
 &\quad - 2(\langle e_{n-1}, e_n \rangle - \langle e_{n-1} - e_n, e_n \rangle) \\
 &= \|e_{n-1}\|^2 - \|e_n\|^2
 \end{aligned} \tag{3.17}$$

3.3. Universal Approximation of SLFNs with Random Hidden Nodes (ELM)

51

Thus, we have $\lim_{n \rightarrow +\infty} \|\beta_n g_n\|^2 = \lim_{n \rightarrow +\infty} (\|e_{n-1}\|^2 - \|e_n\|^2) =$

$r^2 - r^2 = 0$, then, for any given positive integer m we have

$$\lim_{n \rightarrow +\infty} \sum_{p=n+1}^{n+m} \|\beta_p g_p\| = 0. \quad (3.18)$$

Since $\|e_{n_k+m} - e_{n_k}\| = \|\sum_{p=n_k+1}^{n_k+m} \beta_p g_p\| \leq \sum_{p=n_k+1}^{n_k+m} \|\beta_p g_p\|$, for any

fixed positive integer m we have

$$\lim_{k \rightarrow +\infty} \|e_{n_k} - e_{n_k+m}\| = 0 \quad (3.19)$$

We then further have

$$\begin{aligned} & \lim_{k \rightarrow +\infty} \|e_{n_k+m} - e^*\| \\ &= \lim_{k \rightarrow +\infty} \|e_{n_k+m} - e_{n_k} + e_{n_k} - e^*\| \\ &\leq \lim_{k \rightarrow +\infty} \|e_{n_k+m} - e_{n_k}\| + \lim_{k \rightarrow +\infty} \|e_{n_k} - e^*\| = 0. \end{aligned} \quad (3.20)$$

where e_{n_k+m} need not be an element of the subsequence $\{e_{n_k}\}$.

According to equations (3.16) and (3.20) there exists $\tilde{N}_3 > \tilde{N}_2$,

$\forall k > \tilde{N}_3$ we have (cf. Figure 3.3)

$$\begin{aligned} \|e_{n_k+p}\| &< \frac{r}{1-\xi} \quad \text{and} \\ \theta_{(e_{n_k+p}, e^*)} &< \arcsin\left(\frac{1-\xi - \sin(\theta_{(g^*, e^*)})}{2}\right), \quad (3.21) \\ 0 &\leq p \leq L(n_k) \end{aligned}$$

where $L(n_k)$ is a function of k : $L(n_k) = \max_m \{m : \|e_{n_k+p}\| < \frac{r}{1-\xi}$

and $\theta_{(e_{n_k+p}, e^*)} < \arcsin\left(\frac{1-\xi - \sin(\theta_{(g^*, e^*)})}{2}\right), 0 \leq p \leq m\}$. Obviously

for any given positive integer M , $\exists k > \tilde{N}_3$ $L(n_k) > M$ since for any

given positive integer m we have $\lim_{k \rightarrow +\infty} \sum_{p=n_k+1}^{n_k+m} \|\beta_p g_p\| = 0$.

3.3. Universal Approximation of SLFNs with Random Hidden Nodes (ELM)

52

(b.3) Let $\theta_{(g_n, g^*)}$ denote the angle formed by g_n and g^* and $0 \leq \theta_{(g_n, g^*)} \leq \pi/2$. Since function sequences $\{g_n\}$ are randomly generated at a continuous sampling distribution probability, according to Lemma 3.2.6 there exists a positive integer M such that for any continuous segment $G_{(n, M)} = \{g_n, g_{n+1}, \dots, g_{n+M-1}\}$ ($n = 1, 2, \dots$) of function sequence $\{g_n\}$ $\exists g_{n+p} \in G_{(n, M)}$ satisfying $\theta_{(g_{n+p}, g^*)} < \arcsin\left(\frac{1-\xi-\sin(\theta_{(g^*, e^*)})}{2}\right)$. As $\exists k > \tilde{N}_3$ such that $L(n_k) + 1 > M$, according to Lemma 3.2.6 for continuous segment $G_{(n_k+1, L(n_k)+1)} = \{g_{n_k+1}, \dots, g_{n_k+L(n_k)+1}\}$ $\exists p: 0 \leq p \leq L(n_k)$ satisfying (cf. Figure 3.3)

$$\theta_{(g_{n_k+p+1}, g^*)} < \arcsin\left(\frac{1-\xi-\sin(\theta_{(g^*, e^*)})}{2}\right) \quad (3.22)$$

(b.4) Let $\theta_{(e_{n_k+p}, g_{n_k+p+1})}$ denote the angle formed by e_{n_k+p} and g_{n_k+p+1} .

As $\theta_{(e_{n_k+p}, g_{n_k+p+1})} \leq \theta_{(e_{n_k+p}, e^*)} + \theta_{(e^*, g_{n_k+p+1})}$ and $\theta_{(e^*, g_{n_k+p+1})} \leq \theta_{(g^*, e^*)} + \theta_{(g_{n_k+p+1}, g^*)}$, where $\theta_{(e^*, g_{n_k+p+1})}$ is the angle formed by e^* and

g_{n_k+p+1} , we have $\theta_{(e_{n_k+p}, g_{n_k+p+1})} \leq \theta_{(g^*, e^*)} + \theta_{(e_{n_k+p}, e^*)} + \theta_{(g_{n_k+p+1}, g^*)}$

(cf. Figure 3.3). Thus, according to equations (3.21) and (3.22)

we have

$$\sin(\theta_{(e_{n_k+p}, g_{n_k+p+1})}) \leq \sin(\theta_{(g^*, e^*)}) + \sin(\theta_{(e_{n_k+p}, e^*)}) + \sin(\theta_{(g_{n_k+p+1}, g^*)})$$

$< 1 - \xi$. Furthermore, as $\|e_{n_k+p}\| < \frac{r}{1-\xi}$ (cf equation (3.21)), we

have $\|e_{n_k+p+1}\| = \|e_{n_k+p}\| \sin(\theta_{(e_{n_k+p}, g_{n_k+p+1})}) < \frac{r}{1-\xi} * (1 - \xi) = r$

which is contradictory to the fact that all $\|e_n\| \geq r$. Thus $r = 0$,

that is $\lim_{n \rightarrow +\infty} \|e_n\| = r = 0$. This completes the proof.

3.3. Universal Approximation of SLFNs with Random Hidden Nodes (ELM)

53

□

Remark 3: Seen from Theorem 3.3.1, in order to let the incremental network f_n with random hidden nodes converge to any continuous target function f one may only need to choose a bounded nonconstant piecewise continuous activation function $g : R \rightarrow R$ for additive nodes (such as sigmoidal, threshold, Gaussian, sine, cosine functions) or an integrable piecewise continuous activation function $g : R \rightarrow R$ and $\int_R g(x)dx \neq 0$ for RBF nodes (such as Gaussian function). Such activation functions also include some non-regular functions as shown in Huang and Babri[70]. It should be noted that all the previous results [8], [11]-[14], [19], [28], [38], [64]-[81] are based on the conventional network model where the parameters of hidden nodes need to be tuned during training phase. Furthermore, these previous theoretical results may not be implementable either.

Remark 4: Barron [19] proposed an incremental algorithm taking the form $f_n = \alpha_n f_n + (1 - \alpha_n)g_n$ which tries to achieve the nearly minimal value for $\|\alpha_n f_n + (1 - \alpha_n)g_n - f\|$ by adjusting the input weights and the bias of the newly added node and then fixing them after tuning. Meir and Maiorov [38] also proposed an incremental algorithm for the Sobolev class of target functions $f \in W_p^r(K)$ instead of any continuous function f discussed in this chapter. The Sobolev class of functions $W_p^r(K)$ for any nonnegative integer r is defined as $W_p^r(K) = \left\{ f : \max_{0 \leq |\vec{k}| \leq r} \left\| \frac{\partial^{|\vec{k}|} f}{\partial x_1^{k_1} \dots \partial x_d^{k_d}} \right\|_{L^p(K)} < \infty \right\}$ where

3.3. Universal Approximation of SLFNs with Random Hidden Nodes (ELM)

54

$|\vec{k}| = k_1 + \dots + k_d$. The incremental algorithm proposed by Meir and Maierov [38] takes the form $f_n = \frac{n-1}{n}f_n + \frac{Q}{n}h_n$ or $f_n = \frac{n-1}{n}f_n + \frac{1}{n}g_n$ and minimizes the value of $\|f_n - f\|$ by optimizing the input weights and the bias of the newly added node and then fixing them after tuning. Similarly Romero [39, 41] proposed an incremental algorithm taking the form $f_n = \sum_{i=1}^n \beta_i^n g_i$. β_i^n ($i = 1, \dots, n$) and g_n are chosen to minimize $\|f - (f_{n-1} + \mu g)\| + \alpha_n$ ($\forall \mu$ and $\forall g$) where $\alpha_n \geq 0$. Its universal approximation capability was only proved under the hypothesis $\lim_{n \rightarrow +\infty} \alpha_n = 0$. It should be noted that all these previous incremental algorithms [19], [38]-[41] need to recalculate the output weights of all the existing nodes when a new node is added. Different from these methods, Kwok and Yeung [28] only adjusts the parameters (including input weights, bias, and output weights) of the newly added node and then fix all of them. All the parameters (including output weights) of the previous existing nodes will be no longer tuned when a new node is added. Our incremental method randomly generates input weights and biases for newly added nodes instead of tuning them, and the output weights are remained fixed once tuned. Unlike the theoretical works done by Barron [19] and Meir and Maierov [38] which don't give the explicit methods on how to tune all the parameters, our theory indeed gives an efficient learning implementation by obtaining the value for the only parameters (output weights) directly:

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

Remark 5: Different from other incremental learning theories [19], [28], [38]

which are proposed for SLFNs with additive nodes, our main theorem 3.3.1 is also valid for RBF networks. The algorithm proposed by Romero[39], [41] works for both additive nodes and RBF nodes, however, all the parameters need to be tuned in that algorithm.

Remark 6: The network with the fixed architecture and randomly assigned hidden nodes is called Extreme Learning Machine (ELM) [1]-[7], [141] where the output parameters are determined by the ordinary least square method. According to Theorem 3.3.1 $\lim_{n \rightarrow \infty} \|f - f_n\| = 0$ holds for ELM.

Remark 7: Igel'nik, et al[22] proposed a kind of random selection approach. But it should be noted that “random” mentioned in [22] is not totally random. In fact, in their theorem, neural networks only randomly choose input-to-hidden weights $\mathbf{a}_i$, but hidden biases b_i must dependent on these input-to-hidden weights. In our theory, hidden biases are totally independent of input-to-hidden weights, i.e., input-to-hidden weights and hidden biases both are randomly chosen.

3.4 Conclusions

In this chapter, we successfully prove in an incremental constructive method that the input weights and hidden layer biases of SLFNs with additive hidden nodes or centers and impact factors of RBF nodes of RBF networks need

not be tuned at all. In order to make the incremental network f_n with randomly generated hidden nodes converge to any continuous target function f in a compact subset of $\mathbf{R}^d$ one may only need to choose any bounded nonconstant piecewise continuous activation function $g : R \rightarrow R$ for additive nodes or any integrable piecewise continuous activation function $g : R \rightarrow R$ and $\int_R g(x)dx \neq 0$ for RBF nodes. It is also noted that the previous results [8], [11]-[14], [19], [28], [38], [64]-[81] are based on the conventional network model where additive or RBF hidden nodes need to be tuned during training. These previous proof methods may not be implementable either. For example, some previous constructive proof methods[19], [21] need to find enough number of appropriate interpolation or partition points for network construction purpose, however, in real applications only finite number of training samples are available and they may not be the required interpolation points for such universal approximation problems.

Chapter 4

Incremental Extreme Learning Machine (I-ELM)

In Chapter 3, we prove that single-hidden layer feedforward neural networks (SLFNs) with random input weights and hidden layer biases of additive hidden nodes or centers and impact factors of RBF nodes are universal approximation. The incremental constructive method provided in the proof of chapter 3 actually works and performs efficiently in real applications. In this chapter, we apply the incremental constructive method referred to as Incremental Extreme Learning Machine (I-ELM) to real applications. The experimental results on several benchmark real world regression problems will verify our conclusion. Further, an ensemble of incremental extreme learning machine (I-ELM) will be proposed. While retaining the important universal

approximation feature and the simplicity of I-ELM, the ensemble of I-ELM not only reduces the number of hidden nodes, but also achieves good generalization performance at a fast speed.

4.1 Introduction

In recent years, neural networks have attracted considerable attention in many fields mainly because of their capability to approximate complex nonlinear mappings directly from the input samples. Many kinds of multi-layer feedforward neural networks have been investigated thoroughly. From a mathematical point of view, research on the approximation capabilities of multi-layer feedforward neural networks has focused on universal approximation in $\mathbf{R}^n$ or one compact set of $\mathbf{R}^n$ [11]-[12], [18]-[19], [21], [29], [42], [70]. Many researches show that feedforward neural networks can approximate complex nonlinear mappings directly from several input variables to several output variables [8], [12]-[13], [18]-[19], [21], [29]. However, the complexity of neural networks and activation functions may bring computational burdens, especially for solving real applications with large scale data. Meantime, many practical applications illustrate that fast response to external inputs is often demanded. Therefore, whether neural networks can own real-time learning capability is always a challenging question in neural network research.

In Chapter 3, an incremental constructive method provided in the proof

of chapter 3 actually works as universal approximators. In this chapter, we apply the incremental constructive method referred to as Incremental Extreme Learning Machine (I-ELM) to real applications. The experimental results on some benchmark real world regression problems will verify our conclusion, i.e. I-ELM performs efficiently in real applications. As observed from the simulation results, I-ELM can actually achieve good performance. Compared to other popular algorithms such as support vector regression (SVR) [106], [145], stochastic gradient descent backpropagation (BP) [92], and incremental RBF networks (RAN [64], RANEKF [67], MRAN [65]-[66], GAP-RBF [68], GGAP-RBF [69]), the proposed incremental SLFN is fully automatic in the sense that except for target errors and the allowed maximum number of hidden nodes no control parameters need to be manually tuned by users.

Recently, from Huang's research[70], it is known that N arbitrary distinct samples can be learned precisely by standard SLFNs with N hidden neurons with almost any activation function found in applications. However, in most applications, input samples often need to be dealt with. If it was necessary to use N hidden neurons to precisely learn N distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$, where $\mathbf{x}_i \in \mathbf{R}^d$ and $\mathbf{t}_i \in \mathbf{R}^m$, the complexity (size) of such neural networks would become very large with increasing number of input samples that would increase the running time and reduce the computation speed, thereby leading to the dropping performance of whole systems.

In order to learn finite training samples, Huang[137] constructs a specific class of TLFNs which need much less additive nodes than SLFNs by letting separate learning machines (SLFNs locally trained on different data regions) share common additive nodes in the first hidden layer. In that implementation, the input weights and the biases of the first hidden layers are also randomly given. TLFNs proposed in Huang[137] can approximate those finite training samples with arbitrarily small errors, however, it is still an open issue whether it can universally approximate any continuous functions. The new theory of Chapter3 proves that SLFNs with random hidden nodes can work as function approximators. During this incremental implementation, the input weights and hidden layer biases need only to be randomly given. Based on the constructive TLFNs in Huang[137] and the introduced incremental implementation of SLFN in Chapter4, naturally we can construct very compact TLFNs similar to the one proposed by Huang[137] but with the important universal approximation feature. Some simulation results based on real-world benchmarking problems also show that the ensemble of I-ELM can achieve good generalization performance in a short time.

4.2. Proposed Incremental Extreme Learning Machine (I-ELM) 61

4.2 Proposed Incremental Extreme Learning Machine (I-ELM)

According to Theorem 3.3.1, when the n -th hidden node is added, the weight β_n linking the new node to the output node should be chosen as $\frac{\langle e_{n-1}, g_n \rangle}{\|g_n\|^2}$. In practice, it could not be calculated since the exact functional form of e_{n-1} is unavailable. Similar to Kwok and Yeung[28] (p. 1134), a consistent estimate of the weight β_n based on the training set is

$$\beta_n = \frac{E \cdot H^T}{H \cdot H^T} = \frac{\sum_{p=1}^N e(p)h(p)}{\sum_{p=1}^N h^2(p)} \quad (4.1)$$

where $h(p)$ is the activation of the new hidden node for the input of p -th training sample and $e(p)$ is the corresponding residual error before this new hidden node is added. $H = [h(1), \dots, h(N)]^T$ is the activation vector of the new node for all the N training samples and $E = [e(1), \dots, e(N)]^T$ is the residual vector before this new hidden node added. In real applications, one may not wish to get zero approximation error by adding infinite nodes to the network; a maximum number of hidden nodes are normally given. Thus, such an incremental constructive method for SLFNs can be summarized as follows:

I-ELM: Given a training set $\aleph = \{(\mathbf{x}_i, t_i) | \mathbf{x}_i \in \mathbf{R}^d, t_i \in R, i = 1, \dots, N\}$, activation function $g(x)$, maximum node number $\tilde{N}_{\max}$ and expected learning accuracy ϵ ,

4.2. Proposed Incremental Extreme Learning Machine (I-ELM) 62

step 1 Initialization: Let $\tilde{N} = 0$ and residual error $E = t$, where $t = [t_1, \dots, t_N]^T$.

step 2 Learning step: while $\tilde{N} < \tilde{N}_{\max}$ and $\|E\| > \epsilon$

- (a) Increase by 1 the number of hidden nodes $\tilde{N}$: $\tilde{N} = \tilde{N} + 1$.
- (b) Assign random input weight $\mathbf{a}_{\tilde{N}}$ and bias $b_{\tilde{N}}$ (or random center $\mathbf{a}_{\tilde{N}}$ and impact factor $b_{\tilde{N}}$) for new hidden node $\tilde{N}$.
- (c) Calculate the output weight $\beta_{\tilde{N}}$ for the new hidden node:

$$\beta_{\tilde{N}} = \frac{E \cdot H_{\tilde{N}}^T}{H_{\tilde{N}} \cdot H_{\tilde{N}}^T} \quad (4.2)$$

- (d) Calculate the residual error after adding the new hidden node $\tilde{N}$:

$$E = E - \beta_{\tilde{N}} \cdot H_{\tilde{N}} \quad (4.3)$$

endwhile

Before learning, there is no node in the network and the residual error is initially set as the expected target vector of the training data set as shown in *step 1*. Learning will stop when the number $\tilde{N}$ of hidden nodes has exceeded the predefined maximum number $\tilde{N}_{\max}$ or the residual error E becomes less than the expected one. The E in the right side of equation (4.3) represents the residual error vector before the new node added and the E in the left side represents the residual error vector after the new node added, which is consistent to $E_{\tilde{N}} = f - f_{\tilde{N}} = E_{\tilde{N}-1} - \beta_{\tilde{N}} g_{\tilde{N}}$.

Remark: Different from most popular learning algorithms which require the activation function of hidden nodes differentiable, the proposed incremental learning algorithm is not only suitable for differentiable activation function such as the sigmoidal, Gaussian, sine and cosine functions but also for non-differentiable functions such as threshold functions.

4.3 Performance Evaluation

In this section, the performance of the proposed incremental learning algorithms are compared with other popular learning algorithms on benchmark problems in the function approximation area: 1) approximation of a SinC function and 2) 8 real world function approximation problems from UCI database¹ [146].

The popular learning algorithms to be compared include support vector machine for regression (SVR) [106], [145], stochastic gradient descent back-propagation (BP) [92], and other popular incremental learning algorithms such as RAN[64] and MRAN[65]-[66]. The sigmoidal activation function with gain parameter λ : $g(x) = \frac{1}{1+e^{-\lambda x}}$ is used in the stochastic gradient BP algorithm. The kernel function used in SVM and our SLFN with RBF nodes are the radial basis function $\phi(\mathbf{x}) = \exp(-\gamma\|\mathbf{x} - \mu\|^2)$. The specification

¹Auto Price dataset, which was originally from [146], can be downloaded from <http://www.liacc.up.pt/~ltorgo/Regression/price.html>.

Table 4.1: Specification of benchmark datasets.

Problems	# Training Data	# Testing Data	# Attributes	
			Continuous	Normal
Abalone	2000	2177	7	1
Auto Price	80	79	14	1
Boston	250	256	13	0
California	8000	12640	8	0
Census	10000	12784	8	0
Delta Ailerons	3000	4129	5	0
Delta Elevators	4000	5517	6	0
Machine CPU	100	109	6	0

of these benchmark problems are shown in Table 4.1. In our experiments, all the inputs (attributes) have been normalized into the range $[-1, 1]$ while the outputs (targets) have been normalized into $[0, 1]$. For the case of SLFN with sigmoidal and Sin additive hidden nodes, the input weights $\mathbf{a}_i$ and hidden biases b_i are randomly chosen from the range $[-1, 1]$. For the case of SLFN with hardlimit additive hidden nodes ($g(x) = -1_{x \leq 0} + 1_{x > 0}$), the input weights $\mathbf{a}_i$ and hidden biases b_i are randomly chosen from the range $[-0.1, 0.1]$ and $[-1, 1]$, respectively. For SLFN with RBF activation function $\phi(\mathbf{x}) = \exp(-\gamma \|\mathbf{x} - \mu\|^2)$, the centers μ_i are randomly chosen from the range $[-1, 1]$ whereas the impact factor γ is chosen from the range $(0, 0.5)$. For SLFN, our target error is $\epsilon = 0.01$.

All the simulations for BP, RAN, MRAN and our new incremental learn-

ing algorithms are carried out in MATLAB 6.5 environment running in a Pentium 4 2.99 GHZ CPU. The simulations for SVM are carried out using compiled C-coded SVM packages: LIBSVM[127] running in the same PC. The basic algorithm of this C-coded SVM packages is a simplification of three works: original SMO by Platt[147], SMO's modification by Keerthi et al[116], and SVM^{Light} by Joachims[148]. It should be noted that a C implementation would be faster than MATLAB by up to 10-50 times for the same applications [149]. So a comparison with LIBSVM gives SVM an advantage. 20 trials have been conducted for each problem with training and testing datasets randomly generated for each trial. Simulation results including the average testing root mean square error (RMSE) and the corresponding standard deviation (Dev) are given in this section.

4.3.1 Comparison between I-ELM and other Popular Incremental Learning Algorithms

The performance of the proposed incremental learning algorithm is first compared with other popular incremental learning algorithms such as RAN and MRAN. Some parameters for BP, RAN and MRAN need to be manually tuned by trial-and-error. For both RAN and MRAN, $\eta_{\max} = 1.15$, $\eta_{\min} = 0.04$, $\epsilon = 0.0001$, and $\gamma = 0.999$. For MRAN, some additional parameters are set as: $Q = 0.0005$, and growing and pruning thresholds are set

4.3. Performance Evaluation

66

as 0.0001. In addition, for RAN and MRAN algorithms, distance parameter $\kappa = 0.3$ is set for Abalone, California, and Census and $\kappa = 0.8$ is set for other cases. The allowed maximum number of nodes of SLFNs is $\tilde{N}_{\max} = 300$.

Figure 4.1 shows the average testing RMSE decreasing trends and the spent training time of the SLFN with RBF nodes for some datasets. As observed from Figure 4.1(a) the testing RMSE are decreasing with the increase of hidden nodes. Seen from Figure 4.1(b), the spent learning time is linearly increased with the learning steps since the learning time spent for each step (a new node added) is almost fixed (three fixed-length vector multiplication operations only). For the sake of readability, only a partial of simulation results are shown in Figure 4.1. In fact, similar learning curves can also be plotted for the SLFN with different type of nodes trained on different datasets.

During our simulations it is found that without further increasing hidden nodes the SLFN with 200 hidden nodes can generally obtain good performance for all the tested cases which are comparable to the results obtained by RAN and MRAN. Table 4.2 and Table 4.3 show the average testing RMSE and the corresponding standard deviation (Dev) of the proposed SLFN with 200 hidden nodes, RAN and MRAN. As observed from Table 4.2 and Table 4.3, except California dataset the generalization performance of the proposed SLFN algorithm with different activation functions is generally better than or comparable to RAN and MRAN. The training time spent for each

4.3. Performance Evaluation

67

(a)

(b)

Figure 4.1: Performance of the proposed SLFN learning algorithm with RBF nodes on some benchmark problems: (a) average testing RMSE, (b) training time (seconds).

4.3. Performance Evaluation

68

Problems	SLFN			RAN	MRAN
	Sigmoid	RBF	Sin		
	Mean	Mean	Mean	Mean	Mean
Abalone	0.0920	0.0938	0.0886	0.1183	0.0906
Auto Price	0.0977	0.1261	0.1162	0.1418	0.1373
Boston	0.1167	0.1320	0.1404	0.1474	0.1321
California	0.1683	0.1731	0.1550	0.1506	0.1480
Census	0.0923	0.0922	0.0842	0.1061	0.0903
Delta Ailerons	0.0525	0.0632	0.0635	0.1018	0.0618
Delta Elevators	0.0740	0.0790	0.0739	0.1322	0.0807
Machine CPU	0.0504	0.0674	0.0665	0.1069	0.1068

Table 4.2: Average testing RMSE of different algorithms: SLFN (with 200 nodes) with different activation functions (Sigmoid, Sin and RBF), and two popular sequential learning algorithms: RAN and MRAN.

learning algorithm is shown in Table 4.4, it can be seen that the proposed SLFN algorithm runs much faster than other sequential learning algorithms. The network architectures of RAN and MRAN are automatically determined through their own internal growing and/or pruning mechanism, which is also shown in Table 4.4. Different from RAN and our SLFN algorithms, MRAN can prune insignificant nodes based on some criterion, thus, among all algorithms MRAN obtains most parsimonious network architectures. Among all the algorithms, RAN obtains the largest size of networks for many tested cases excepted for small datasets.

Problems	SLFN			RAN	MRAN
	Sigmoid	RBF	Sin		
	Dev	Dev	Dev	Dev	Dev
Abalone	0.0046	0.0053	0.0049	0.0076	0.0065
Auto Price	0.0069	0.0255	0.0179	0.0261	0.0381
Boston	0.0112	0.0126	0.0114	0.0177	0.0140
California	0.0049	0.0081	0.0052	0.0035	0.0030
Census	0.0023	0.0029	0.0015	0.0038	0.0042
Delta Ailerons	0.0078	0.0116	0.0090	0.0083	0.0050
Delta Elevators	0.0126	0.0123	0.0065	0.0130	0.0068
Machine CPU	0.0079	0.0177	0.0278	0.0246	0.0367

Table 4.3: The standard deviations (Dev) of average testing RMSE for different algorithms: SLFN (with 200 nodes) with different activation functions (Sigmoid, Sin and RBF), and two popular sequential learning algorithms: RAN and MRAN.

4.3.2 Comparison between I-ELM with SVR and Stochastic Gradient Descent BP

The proposed SLFN algorithm is also compared with the popular function regression algorithms such as SVR (Support Vector Machine for Regression) and Stochastic Gradient Descent BP. Compared with other methods, SVR can usually obtain higher generalization performance. Thus, the allowed maximum number of nodes for SLFN is increased to 500 in this comparison. The sigmoidal activation function is used in the SLFN. In order to

4.3. Performance Evaluation

70

Problems	Training time of SLFN			Training Time		# nodes	
	Sigmoid	RBF	Sin	RAN	MRAN	RAN	MRAN
Abalone	0.2214	0.5030	0.1778	39.928	255.84	186.3	67.7
Auto Price	0.0329	0.0468	0.0188	0.3565	2.5015	23.8	22.5
Boston	0.0515	0.0657	0.0470	2.0940	22.767	40.5	36.2
California	0.5448	1.3656	0.3872	3301.7	2701.1	4883.0	93.0
Census	0.8667	1.7928	0.5194	5399.0	3805.3	6393.2	77.3
Delta Ailerons	0.2620	0.4327	0.1715	237.96	175.07	1118.1	76.6
Delta Elevators	0.2708	0.6321	0.2261	661.78	331.75	2417.4	76.8
Machine CPU	0.0234	0.0447	0.0297	0.1735	0.2454	6.9	7.0

Table 4.4: Training time (seconds) and network complexity comparison of different algorithms: SLFN with different activation functions (Sigmoid, Sin and RBF), and two popular sequential learning algorithms: RAN and MRAN. In all these case, 200 nodes are incrementally added to SLFNs.

get the best generalization performance, the cost parameter C and kernel parameter γ of SVR need to be chosen appropriately. Similar to Hsu and Lin[121] we estimate the generalized accuracy using different combinations of cost and kernel parameters (C, γ) : $C \in \{2^{12}, 2^{11}, \dots, 2^{-1}, 2^{-2}\}$ and $\gamma \in \{2^4, 2^3, \dots, 2^{-9}, 2^{-10}\}$. Therefore, for each problem we try $15 \times 15 = 225$ combinations of parameters (C, γ) for SVM and the average performances over 20 trials are obtained for each combination of parameters (C, γ) .

The average performance (testing RMSE, the corresponding standard deviation (Dev) and the training time) of SLFN and SVR are shown in

4.3. Performance Evaluation

71

Problems	SLFN ^a		BP ^a ($\lambda = 1$)		SVR ^b			
	Testing RMSE		Testing RMSE		Testing RMSE		# SVs	(C, γ)
	Mean	Dev	Mean	Dev	Mean	Dev		
Abalone	<u>0.0878</u>	0.0032	0.1175	0.0095	<u>0.0846</u>	0.0013	309.84	$(2^4, 2^{-6})$
Auto Price	0.0883	0.0036	0.2383	0.0587	0.1052	0.0040	21.25	$(2^8, 2^{-5})$
Boston	0.1095	0.0090	0.1882	0.0243	0.1155	0.0079	46.44	$(2^4, 2^{-3})$
California	0.1555	0.0021	0.1579	0.0033	0.1311	0.0011	2189.2	$(2^3, 2^1)$
Census	0.0871	0.0021	0.0866	0.0025	0.0683	0.0013	810.24	$(2^1, 2^{-1})$
Delta Ailerons	<u>0.0472</u>	0.0049	<u>0.0459</u>	0.0033	<u>0.0467</u>	0.0010	82.44	$(2^3, 2^{-3})$
Delta Elevators	<u>0.0639</u>	0.0067	<u>0.0653</u>	0.0019	<u>0.0603</u>	1.1210	260.38	$(2^0, 2^{-2})$
Machine CPU	0.0491	0.0089	0.1988	0.0429	0.0620	0.0180	7.8	$(2^6, 2^{-4})$

^a run in MATLAB environment.^b run in C executable environment which is usually much faster than MATLAB.

Table 4.5: Performance comparison of SLFN with incrementally added 500 nodes, stochastic gradient descent BP, and SVR. The close generalization performance (testing RMSE) obtained by different algorithms are underlined and the best results among three algorithms are shown in boldface.

Table 4.5 and Table 4.6. Table 4.5 also shows the number of support vectors obtained by SVR and the optimal values of the cost and kernel parameters (C, γ) . The learning rate η and the momentum constant m of BP are set as $\eta = 0.06$ and $m = 0.8$. The number of nodes assigned to stochastic gradient descent BP algorithm is decided by trial-and-error in order to obtain as good generalization performance as possible. During our simulations, 15 nodes and 30 nodes are assigned to BP when tested on Auto Price and

4.3. Performance Evaluation

72

Problems	SLFN ^a	BP ^a ($\lambda = 1$)	SVR ^b
	Training Time (s)	Training Time (s)	Training Time (s)
Abalone	0.5560	0.4406	1.6123
Auto Price	0.0954	0.0154	0.0042
Boston	0.1419	0.0579	0.0494
California	1.3763	2.0307	74.184
Census	1.7295	2.7814	11.251
Delta Ailerons	0.7058	0.6610	0.6726
Delta Elevators	0.7296	0.8830	1.1210
Machine CPU	0.0765	0.0206	0.0018

^a run in MATLAB environment.

^b run in C executable environment which is usually much faster than MATLAB.

Table 4.6: Training time comparison of SLFN with incrementally added 500 nodes, stochastic gradient descent BP, and SVR.

Census datasets, respectively, and 10 nodes are assigned to BP tested on other datasets. It can be seen that the generalization performance of SLFN is close to that of SVR in most cases. Among 8 datasets, the proposed SLFN obtains much better generalization performance than BP in 4 datasets and almost same generalization performance in the other 4 datasets. Compared to the results of RAN and MRAN shown in Table 4.2, the proposed SLFN with 500 sigmoidal nodes produces better generalization performance than RAN and MRAN in all cases except California dataset. For medium or large size of applications, SVR spent much more time on training than SLFN.

4.3.3 Performance Evaluation of I-ELM for Threshold Networks

Problems	Testing RMSE		Dev of Testing RMSE		Training Time (s)	
	Threshold	BP($\lambda = 10$)	Threshold	BP($\lambda = 10$)	Threshold	BP($\lambda = 10$)
Abalone	0.0951	0.1332	0.0142	0.0102	0.2908	0.4313
Auto Price	0.1141	0.3209	0.0130	0.0665	0.0735	0.0172
Boston	0.1346	0.2196	0.0104	0.0279	0.0907	0.0548
California	<u>0.1828</u>	<u>0.1806</u>	0.0179	0.0226	0.8186	1.9548
Census	0.0941	0.1032	0.0062	0.0068	1.0117	2.7359
Delta Ailerons	0.0790	0.0400	0.0397	0.0055	0.3550	0.6375
Delta Elevators	0.0713	0.0895	0.0110	0.0090	0.5102	0.8970
Machine CPU	0.0739	0.2281	0.0140	0.0479	0.0658	0.0215

Table 4.7: Performance comparison between the approximated threshold network ($\lambda = 10$) trained by stochastic gradient descent BP and the true threshold networks trained by the proposed SLFN with 500 threshold nodes:

$$g(x) = -1_{x < 0} + 1_{x \geq 0}.$$

The widely BP learning algorithm and its variants cannot be used to train the threshold neural networks (SLFN with threshold hidden activation function) directly as the threshold functions are non-differentiable. One method is to approximate threshold activation function using sigmoid activation function with higher gain parameters λ . For example, a threshold network may be approximated by a sigmoid network with $\lambda = 10$. One may then train such an approximated threshold network using BP instead of training the threshold network directly[3]. The performance comparison be-

4.4. Proposed Ensemble of Incremental Extreme Learning Machines with Random Additive Hidden Nodes

74

tween our SLFN algorithm for threshold network and the stochastic gradient descent BP for sigmoid network with $\lambda = 10$ have been conducted. There are 500 threshold nodes assigned to the SLFN. The average performance of our SLFN and stochastic gradient descent BP for threshold networks is shown in Table 4.7. As observed from Table 4.7, overall speaking our proposed SLFN algorithm tends to have better generalization performance than the approximated threshold network trained by BP. It should be noted that the stochastic gradient descent BP just trains an approximated threshold network instead of a true threshold network whereas our SLFN algorithm provides a method to train the true threshold network directly. Figure 4.2 shows the testing RMSE decreasing trends for some datasets.

4.4 Proposed Ensemble of Incremental Extreme Learning Machines with Random Additive Hidden Nodes

In this part, an ensemble of I-ELM will be proposed. For simplicity, we first construct a subnetwork which can approximate the following target:

$$\bar{f} = f/C + 0.5 \quad (4.4)$$

which is a scaled function of the continuous target f and C is a scale factor to make $0 < \bar{f} < 1$.

4.4. Proposed Ensemble of Incremental Extreme Learning Machines with Random Additive Hidden Nodes

75

Figure 4.2: Performance of the SLFN learning algorithm with 500 hardlimit additive nodes on some benchmark problems: average testing RMSE.

Similar to Huang[137], the bounded sample input set X can be divided into L sub-regions $V^{(1)}, \dots, V^{(L)}$ by setting:

$$V^{(p)} = \begin{cases} \{\mathbf{x} | \mathbf{x} \in X \text{ and } b_{p-1} \leq \mathbf{a} \cdot \mathbf{x} < b_p\}, & 1 \leq p < L \\ \{\mathbf{x} | \mathbf{x} \in X \text{ and } b_{L-1} \leq \mathbf{a} \cdot \mathbf{x} \leq b_L\}, & p = L \end{cases} \quad (4.5)$$

where $\mathbf{a}$ is a random vector and $b_0 = \min_{\mathbf{x} \in X} \{\mathbf{a} \cdot \mathbf{x}\}$, $b_L = \max_{\mathbf{x} \in X} \{\mathbf{a} \cdot \mathbf{x}\}$, $b_p = b_0 + p \cdot \frac{b_L - b_0}{L}$ is the p -th equal partition point of the interval $[b_0, b_L]$. Given a bounded, nonconstant and piecewise continuous activation function $g(x)$ and any small positive value ϵ , for randomly chosen sequence $\{(\mathbf{a}_n, b_n)\}_{n=1}^{\infty}$, according to Theorem 3.3.1, we can construct an SLFN denoted by M_p such

4.4. Proposed Ensemble of Incremental Extreme Learning Machines with Random Additive Hidden Nodes

76

that for sub-region $V^{(p)}$, when $\beta_n^{(p)}$ between the hidden layer and the output node of M_p is selected as $\frac{\langle e_{n-1}, g_n \rangle}{\|g_n\|^2}$ and the number of additive hidden nodes $\tilde{N}_p$ of M_p is large enough we have $\|\bar{f}_{\tilde{N}_p} - \bar{f}\|_{\mathbf{x} \in V^{(p)}} < \epsilon$. Obviously, the sigmoidal activation function can be selected for the output layer of all these L SLFNs: $\{M_p\}_{p=1}^L$.

We can construct a subnetwork. Let the hidden nodes of the L separate SLFNs (learning machines) M_p link to the same input nodes and the output nodes of these L learning machines construct the output layer of the sub-network. The activation function used in the output layer is sigmoidal (i.e., $g_{\text{output}}(x) = \frac{1}{1+e^{-x}}$). Furthermore, since the input weights and hidden layer biases of all the L learning machines (SLFNs) M_p can be chosen randomly, these L learning machines can share hidden nodes. For example, without loss of generality, suppose $\tilde{N}_1 \leq \dots \leq \tilde{N}_L$, then we can provide $\tilde{N}_L$ nodes shared by the L learning machines where the first $\tilde{N}_p$ nodes are linked to the p -th node in the output layer. Thus, the p -th learning machine M_p (p -th sub-SLFN) comprises the input nodes, the first $\tilde{N}_p$ nodes in the first hidden layer and the p -th nodes in the second hidden layer. (cf. Figure 4.3)

Similar to Huang[137], now we can construct the expected two hidden layers feedforward network (TLFN) by introducing a quantizer neural module which consists of two sigmoid nodes, A and B. First, we can add $2L$ nodes in the hidden layer. These $2L$ nodes are labeled as $A^{(p)}$ and $B^{(p)}$, where $p = 1, \dots, L$. All the newly added L $A^{(p)}$ nodes and L $B^{(p)}$ nodes are fully

4.4. Proposed Ensemble of Incremental Extreme Learning Machines with Random Additive Hidden Nodes

77

Figure 4.3: A compact TLFN constructed by L quantizers and L local learning machines which share common hidden nodes.

connected to the input layer with weights $\mathbf{a}_{A^{(p)}}$ and $\mathbf{a}_{B^{(p)}}$, $p = 1, \dots, L$, respectively. However, for each output node there are only two newly added nodes linking to it, that is, only two nodes $A^{(p)}$ and $B^{(p)}$ of the $2L$ newly added nodes are linked to the p th output node, where $p = 1, \dots, L$.

The biases of nodes $A^{(p)}$ and $B^{(p)}$ are set as $\bar{b}_{A^{(p)}} = -b_p$ and $\bar{b}_{B^{(p)}} = b_{p-1}$, respectively. The weight vectors of the connections linking the input nodes to the newly added hidden nodes $A^{(p)}$ and $B^{(p)}$ are chosen as $\bar{\mathbf{a}}_{A^{(p)}} = T \cdot \mathbf{a}$ and $\bar{\mathbf{a}}_{B^{(p)}} = -T \cdot \mathbf{a}$, respectively, where neural quantizer factor T is a given positive value. All the weights w_{AB} of the connections linking these newly added $2L$ hidden nodes to their corresponding output nodes are chosen same value $w_{AB} = -U$. The neural quantizer modules can inhibit the outputs of those SLFNs and make them nearly zero by adjusting its quantizer factor

4.4. Proposed Ensemble of Incremental Extreme Learning Machines with Random Additive Hidden Nodes

78

U if the input does not come from their specialized sub-regions. Thus, the p -th node in the second layer of the TLFN (the output of the p -th SLFN) produces the target output if and only if the input comes from the p -th sub-region whereas the outputs of the other nodes are inhibited by neural quantizer modules and can be neglected.

We add an output layer with a linear node which links to all those L output nodes (now the second hidden layer nodes) of the above constructed subsetnetwork. The scale factor C is simply chosen as the weights between the second hidden layer and the output layer of the finally constructed TLFN, and the bias of the new output layer is $b_o = -0.5C$. (cf. Figure 4.3) The output of the final constructed TLFN can universally approximates the target function f .

Similar to I-ELM algorithm, the ensemble of I-ELM can be summarized as follows:

Ensemble of I-ELM: Given a training set $\aleph = \{(\mathbf{x}_i, t_i) | \mathbf{x}_i \in \mathbf{R}^d, t_i \in R, i = 1, \dots, N\}$, activation function $g(x)$, number of groups (sub-regions) L , maximum node number $\tilde{N}_{\max}$ and expected learning accuracy ϵ . Let quantizer factors T and U large enough.

step 1 **Dividing training sample space into sub-regions:**

- (a) Randomly generate a vector $\mathbf{a}$ and reindex the training samples

such that $\mathbf{a} \cdot \mathbf{x}_1 < \dots < \mathbf{a} \cdot \mathbf{x}_N$.

4.4. Proposed Ensemble of Incremental Extreme Learning Machines with Random Additive Hidden Nodes

79

- (b) Divide the sorted training samples into L groups $G^{(p)}$ each consisting of about N/L training samples.

step 2 Initialization: Set the parameters queue $W = \emptyset$, which stores the input weight vector and hidden bias sequence $\{(\mathbf{a}_1, b_1), (\mathbf{a}_2, b_2), \dots\}$.

Let $|W|$ denote the total number of elements in W .

step 3 Learning step:

for $p = 1$ to L

Let $\tilde{N}_p = 0$ and residual error E initialized with the target vector of the training samples of the p -th training group $G^{(p)}$.

while $\tilde{N}_p < \tilde{N}_{\max}$ and $\|E\| > \epsilon$

- (a) Increase by 1 the number of hidden nodes $\tilde{N}_p$: $\tilde{N}_p = \tilde{N}_p + 1$.
- (b) If $\tilde{N}_p \leq |W|$, use the $\tilde{N}_p$ -th parameters $(\mathbf{a}_{\tilde{N}_p}, b_{\tilde{N}_p})$ in W as the input weight vector and bias of the new hidden node. Otherwise, assign random input weight vector $\mathbf{a}_{\tilde{N}_p}$ and bias $b_{\tilde{N}_p}$ for new hidden node $\tilde{N}_p$, and add the parameter $(\mathbf{a}_{\tilde{N}_p}, b_{\tilde{N}_p})$ into W .
- (c) Calculate the output weight $\beta_{\tilde{N}_p}^{(p)}$ for the newly added hidden node of the p -th learning SLFN M_p :

$$\beta_{\tilde{N}_p}^{(p)} = \frac{E \cdot H_{\tilde{N}_p}^T}{H_{\tilde{N}_p} \cdot H_{\tilde{N}_p}^T} \quad (4.6)$$

- (d) Calculate the residual error after adding the new hidden node $\tilde{N}_p$:

$$E = E - \beta_{\tilde{N}_p}^{(p)} \cdot H_{\tilde{N}_p} \quad (4.7)$$

4.4. Proposed Ensemble of Incremental Extreme Learning Machines with Random Additive Hidden Nodes

80

endwhile

endfor

In real applications, quantizer factors T and U can be simply set as a very large value which can be considered infinity in the specified computing environments. For example, in many ordinary PC, T and U can be set as 10^{128} as done in our simulations.

Compared with Huang[137] where only infinite differential activation functions like sigmoidal function have been rigorously proved for such TLFNs, the activation functions which could be used in such TLFNs have been extended very widely in this chapter, include almost all popular activation functions which can be used in the applications of neural networks.

Next, the performance of the proposed SLFN and ensemble of I-ELM algorithms have been compared on medium or large size of problems such as Abalone, California, Census, Delta Ailerons and Delta Elevators from UCI database² [146]. The performance of these two algorithms are also evaluated in the approximation of the ‘SinC’ function:

$$y(x) = \begin{cases} \sin(x)/x, & x \neq 0 \\ 1, & x = 0 \end{cases} \quad (4.8)$$

²Auto Price dataset, which was originally from [146], can be downloaded from <http://www.liacc.up.pt/~ltorgo/Regression/price.html>.

4.4. Proposed Ensemble of Incremental Extreme Learning Machines with Random Additive Hidden Nodes

81

A training set (x_i, y_i) and testing set (x_i, y_i) with 10000 and 5000 data respectively are created where x_i 's are uniformly randomly distributed on the interval $(-10, 10)$. In order to make the regression problem 'real', large uniform noise distributed in $[-0.2, 0.2]$ has been added to all the training samples while testing data remain noise-free. Table 4.8 gives the specification of 6 real datasets used in such comparisons.

Table 4.8: Specification of benchmark datasets.

Problems	# Training Data	# Testing Data	# Attributes
SinC	10000	5000	1
Abalone	2000	2177	7
California	8000	12640	8
Census	10000	12784	8
Delta Ailerons	3000	4129	5
Delta Elevators	4000	5517	6

The sin activation function $g(x) = \sin(x)$ is used in all the simulations. Similar, in our experiments, all the inputs (attributes) have been normalized into the range $[-1, 1]$ while the outputs (targets) have been normalized into $[0, 1]$. For Sin additive hidden nodes, the input weights $\mathbf{a}_i$ and hidden biases b_i are randomly chosen from the range $[-1, 1]$. Our target error is $\epsilon = 0.01$. All the simulations for the proposed SLFN and TLFN algorithms are carried out in MATLAB 6.5 environment running in a Pentium 4 2.99 GHZ CPU. 20 trials have been conducted for each problem with training and test-

4.4. Proposed Ensemble of Incremental Extreme Learning Machines with Random Additive Hidden Nodes

82

ing datasets randomly generated for each trial. Simulation results including the average testing root mean square error (RMSE) and the corresponding standard deviation (Dev) are given in this section.

Problems	TLFN					SLFN		
	Testing RMSE		Spent Time (s)		L	Testing RMSE		Training Time (s)
	Mean	Dev	Training	Sorting		Mean	Dev	
SinC	0.0091	0.0010	0.5078	0.0078	20	0.0096	0.0049	0.6297
Abalone	0.0866	0.0040	0.3141	0.0016	20	0.0886	0.0049	0.1778
California	0.1527	0.0026	0.2453	0.0031	2	0.1550	0.0052	0.3872
Census	0.0813	0.0031	0.3937	0.0094	10	0.0842	0.0015	0.5194
Delta Ailerons	0.0528	0.0108	0.0891	0.0016	2	0.0635	0.0090	0.1715
Delta Elevators	0.0719	0.0039	0.1125	0.0016	2	0.0739	0.0065	0.2261

Table 4.9: Performance comparison of TLFN with 100 Sin nodes and SLFN with 200 Sin nodes. The training times of TLFNs are the total time spent on separate SLFNs trained on different sub-regions and do not include the time spent on sorting training data.

The maximum number of hidden nodes is set as 200 for the SLFN learning algorithm and the maximum number of hidden nodes is set as 100 for the TLFN learning algorithm. The target error is set 0.001 for SinC problem and 0.01 for the rest problems. Table 4.9 shows the average performance of these two algorithms. It can be observed that TLFN with 100 nodes can get better generalization performance than SLFN with 200 nodes. The TLFN algorithm runs faster than SLFN in these cases. The training time shown

in Table 4.9 is the total time spent for training all SLFNs in all sub-regions but does not include the time spent for sorting the input data into different L sub-regions. It can be seen that the sorting time spent in TLFN is only 0.5% to 2.5% of the training time, which could be ignored.

4.5 Discussion

I-ELM is similar to Kowk's work, where a new hidden neuron g_n is added by maximizing the equation $\frac{\langle e_{n-1}, g_n \rangle}{\|g_n\|^2}$. In fact, the maximized equation is calculated by gradient methods. Its mean square error is scaled by an integrated squared error of order $O(1/n)$. Based on the equation, we can estimate the number of hidden neurons to achieve the corresponding error target. Since the hidden neurons of I-ELM are randomly generated, the convergent rate of I-ELM should be less than the squared error of order $O(1/n)$. Though the convergent rate of I-ELM is unknown for us, we can describe the complexity of learning time precisely. For Kowk's algorithm, it normally uses gradient methods to calculate those optimal parameters of input-to-hidden neurons. Due to the uncertainty and complexity of activation functions, it leads to the difficulty to estimate their learning time. But since CI-ELM need not calculate the derivative of activation functions, we can describe its learning time precisely. The training time of I-ELM is linearly increased (c.f Figure 4.1 (b)). It means that the learning time spent for each new neuron is almost

fixed. So the computational complexity of training time is linear.

4.6 Conclusions

Chapter 3 proves in theory that the SLFNs with randomly generated hidden nodes are actually universal approximators. In fact, the proof itself is a practical incremental constructive method, which actually shows an efficient way to construct an incremental feedforward network. In this chapter, this simple incremental constructive method (I-ELM) with different activation functions has been compared with other popular learning algorithms on some benchmark artificial and real-world problems. The simulation results show that our proposed method can reach good generalization performance in a very efficient and simple way. Interestingly, theoretically the learning algorithms derived from our constructive method can be applied to a wide of activation functions no matter whether they are sigmoidal or nonsigmoidal, continuous or noncontinuous, and differentiable or non-differentiable. The traditional gradient descent based learning algorithms cannot be applied to networks with non-differential activation functions such as threshold networks since the required derivatives are not available. These methods may also face local minima issues. Many popular learning algorithms do not deal with threshold networks directly either and instead use some analog networks to approximate them such that gradient-descent method can finally be used. However,

the proposed learning algorithm can be used to train threshold networks directly. A thorough performance evaluation of the proposed algorithms using different activation functions for different applications should be further systematically conducted in future.

Some incremental algorithms proposed for SLFNs with additive hidden nodes in literature [19], [28], [38]-[39], [41] require one to find the best input weights and hidden node biases for newly added nodes based on complicated methods as well. Although many other incremental learning algorithms have been proposed in the literature for training RBF networks [64]-[69], unlike our incremental learning algorithms the universal approximation capability of these previous learning algorithms has not been proved yet. The universal approximation capability of these incremental RBF learning algorithms could be proved based on our proposed theory.

Further, a special ensemble of I-ELM has also been proposed in this chapter, which consists of many learning machines (SLFNs) with additive nodes each locally trained on one sub-region of the whole sample input space. Since all these local learning machines can use the same input weights and hidden layer biases, these local learning machines can actually share the same hidden nodes. Compared to Huang[137] which only shows that infinite differentiable sigmoidal functions can be used as activation function in such TLFNs, it has been extended in this chapter that a large broad of activation functions can be used if only they are bounded nonconstant piecewise continuous. It is

not difficult to see that for modular networks such as ensembling computing and mixture of local experts[152]-[153], one may actually be able to obtain compact networks by allowing those local learning machines to share common hidden nodes. Some researchers [19], [28], [38]-[41] proposed incremental constructive methods for SLFNs which will fix parameters of existing nodes after trained and will not be adjusted when new nodes are added. However, similar compact TLFNs cannot be constructed based on such incremental methods [19], [28], [38]. Based on these constructive methods [19], [28], [38]-[41] the input weights of each SLFN need to be trained at least one time and different SLFNs have different input weights and hidden biases, thus, different SLFNs cannot share common hidden nodes.

Chapter 5

Enhanced Incremental Extreme Learning Machine (EI-ELM)

In Chapter 4, I-ELM has been demonstrated as an effective real-time learning algorithm. Since nodes with I-ELM are randomly chosen, some of nodes may play very small contribution for decreasing the residual error. In this chapter, an enhanced method (EI-ELM) is proposed, which randomly generates hidden node parameters several times at each step when a new hidden node needs to be added, and then the hidden node leading to the smallest residual error is chosen. This method may achieve a more parsimonious network architecture than I-ELM.

5.1 Introduction

In the past decades single-hidden-layer feedforward neural networks (SLFNs) have been investigated extensively from both theoretical and application aspects. SLFN network functions with n hidden nodes can be represented by

$$f_n(\mathbf{x}) = \sum_{i=1}^n \beta_i g_i(\mathbf{x}), \mathbf{x} \in \mathbf{R}^d, \beta_i \in R \quad (5.1)$$

where g_i denotes the hidden node output function and β_i is the (output) weight of the connection between the i -th hidden node and the output node.

For additive nodes with activation function g , g_i is defined as

$$g_i(\mathbf{x}) = g(\mathbf{a}_i \cdot \mathbf{x} + b_i), \mathbf{a}_i \in \mathbf{R}^d, b_i \in R \quad (5.2)$$

where $\mathbf{a}_i$ is the (input) weight vector connecting the input layer to the i -th hidden node and b_i is the (hidden) bias of the i -th hidden node. $\mathbf{a}_i \cdot \mathbf{x}$ denotes the inner product of vectors $\mathbf{a}_i$ and $\mathbf{x}$ in $\mathbf{R}^d$. For RBF nodes with activation function g , g_i is defined as

$$g_i(\mathbf{x}) = g\left(\frac{\|\mathbf{x} - \mathbf{a}_i\|}{b_i}\right), \mathbf{a}_i \in \mathbf{R}^d, b_i \in R^+ \quad (5.3)$$

where $\mathbf{a}_i$ and b_i are the center and impact factor of i -th RBF hidden node. R^+ indicates the set of all positive real value.

Conventional neural network theories [13], [18] show that SLFNs with additive or RBF hidden nodes can work as universal approximators when all the parameters of the networks are allowed adjustable. However, tuning all parameters of the networks may cause learning complicated and inefficient

since tuning based learning may easily converge to local minima and/or may generally be very slow due to improper learning steps. Many iterative learning steps may be required by such learning algorithms in order to obtain better learning performance. It may be difficult to apply tuning methods in networks with non-differentiable activation functions such as threshold networks. Interestingly, different from the common understanding on the neural networks and its theories, Huang, et al [150] has recently rigorously proved that SLFNs with randomly assigned hidden parameters $(\mathbf{a}_i, b_i)$ can work as universal approximators by only calculating the output weights β_i linking the hidden layer to the output nodes. In such SLFNs implementations, the activation functions for additive nodes can be any bounded nonconstant piecewise continuous functions and the activation functions for RBF nodes can be any integrable piecewise continuous functions. The resulting (“*random search*”) method referred to as Incremental Extreme Learning Machines (I-ELM) [150] randomly adds nodes to the hidden layer one by one and freezes the output weights of the existing hidden nodes when a new hidden node is added. The original batching learning mode Extreme Learning Machines (ELM) [1]-[7] and their online sequential learning mode ELM (OS-ELM) [5] are developed for SLFNs with fixed network architectures. The activation functions used in ELM include differentiable [1]-[2], [4], [6]-[7] and nondifferentiable [3] functions, continuous [1]-[2], [4], [6]-[7] and noncontinuous [3] functions, etc.

5.2. Proposed Enhanced Incremental Extreme Learning Machine (EI-ELM) 90

Although Huang, et al [150] has demonstrated that I-ELM has better performance than other popular learning methods such as stochastic gradient descent backpropagation (BP) [92], and incremental RBF networks (RAN[64], MRAN[65]-[66], RANEKF[67], GAP-RBF[68], GGAP-RBF[69]), this chapter aims to provide a simple improved implementation of I-ELM in order to achieve a more compact network architecture for the same applications compared with the original I-ELM. Similar to the original I-ELM, the improved I-ELM is fully automatic in the sense that except for target errors and the allowed maximum number of hidden nodes no control parameters need to be manually given by users.

5.2 Proposed Enhanced Incremental Extreme Learning Machine (EI-ELM)

The I-ELM [150] adds random nodes to the hidden layer one by one and freezes the output weights of the existing hidden nodes after a new hidden node is added. Let $e_n \equiv f - f_n$ denote the residual error function for the current network f_n with n hidden nodes where $f \in L^2(X)$ is the target function. Huang, et al[150] has proved that:

Lemma 5.2.1. *[150] Given any bounded nonconstant piecewise continuous function $g : \mathbf{R} \rightarrow \mathbf{R}$ for additive nodes or any integrable piecewise continuous function $g : \mathbf{R} \rightarrow \mathbf{R}$ and $\int_{\mathbf{R}} g(x)dx \neq 0$ for RBF nodes, for any*

5.2. Proposed Enhanced Incremental Extreme Learning Machine (EI-ELM) 91

continuous target function f and any randomly generated function sequence

$\{g_n\}$, $\lim_{n \rightarrow \infty} \|f - f_n\| = 0$ holds with probability one if

$$\beta_n = \frac{\langle e_{n-1}, g_n \rangle}{\|g_n\|^2} \quad (5.4)$$

Lemma 5.2.1 shows that different from common understanding in neural networks from the functional approximation point of view the hidden nodes of SLFNs need not be tuned at all. In order to train SLFNs one only need to randomly assign the hidden nodes (“*random searching*”) and analytically the output weights only. No tuning is required in the training of SLFNs. Furthermore, the universal approximation capability of ELM for fixed network architectures [1]-[7] is the straightforward corollary of Lemma 5.2.1.

It should be noted that although hidden nodes can be added randomly some newly added hidden nodes may reduce less residual error and some newly added hidden nodes may make residual error reduce more. Based on this observation, in this chapter we propose an improved I-ELM algorithm. In this new method, at any step, among k trial of hidden nodes, the hidden nodes with greatest residual error reduction will be added where k is a fixed constant.

Theorem 5.2.1. *Given any bounded nonconstant piecewise continuous function $g : \mathbf{R} \rightarrow \mathbf{R}$ for additive nodes or any integrable piecewise continuous function $g : \mathbf{R} \rightarrow \mathbf{R}$ and $\int_{\mathbf{R}} g(x)dx \neq 0$ for RBF nodes, for any continuous*

5.2. Proposed Enhanced Incremental Extreme Learning Machine (EI-ELM) 92

target function f and any randomly generated function sequence $\{g_n\}$ and any positive integer k , $\lim_{n \rightarrow \infty} \|f - f_n^*\| = 0$ holds with probability one if

$$\beta_n = \frac{\langle e_{n-1}, g_n^* \rangle}{\|g_n^*\|^2} \quad (5.5)$$

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

Proof. Given the function sequence $\{g_n\}$ which is randomly generated based on any continuous sampling distribution probability, obviously, the function sequence $\{g_{k_n}\}_{n=1}^\infty$ will also be randomly generated based on some continuous sampling distribution probability where g_{k_n} is an element of the n -th segment $\{g_{(n-1)k+1}, \dots, g_{nk}\}$, $n = 1, 2, \dots$. Typically, we can choose g_{k_n} as g_n^* which is the element of the n -th segment $\{g_{(n-1)k+1}, \dots, g_{nk}\}$ to minimize $\|(f - f_{n-1}^*) - \beta_n g_i\|$, $i = (n-1)k+1, \dots, nk$. Thus, according to Theorem 3.3.1, for the function sequence $\{g_n^*\}$, $\lim_{n \rightarrow \infty} \|f - f_n^*\| = 0$ holds with probability one if

$$\beta_n = \frac{\langle e_{n-1}, g_n^* \rangle}{\|g_n^*\|^2} \quad (5.6)$$

□

According to Theorem 3.3.1, when the n -th hidden node is added, the weight β_n linking the new node to the output node should be chosen as $\frac{\langle e_{n-1}, g_n \rangle}{\|g_n\|^2}$. In practice, it could not be calculated since the exact functional form of e_{n-1} is unavailable. Similar to Kwok and Yeung[28] (p. 1134), a

5.2. Proposed Enhanced Incremental Extreme Learning Machine (EI-ELM) 93

consistent estimate of the weight β_n based on the training set is

$$\beta_n = \frac{E \cdot H^T}{H \cdot H^T} = \frac{\sum_{p=1}^N e(p)h(p)}{\sum_{p=1}^N h^2(p)} \quad (5.7)$$

where $h(p)$ is the activation of the new hidden node for the input of p -th training sample and $e(p)$ is the corresponding residual error before this new hidden neuron is added. $H = [h(1), \dots, h(N)]^T$ is the activation vector of the new node for all the N training samples and $E = [e(1), \dots, e(N)]^T$ is the residual vector before this new hidden node added. In real applications, one may not really wish to get zero approximation error by adding infinite neurons to the network, a maximum number of hidden neurons is normally given. Thus, such an incremental constructive method for SLFNs can be summarized as follows:

Algorithm: Given a training set $\aleph = \{(\mathbf{x}_i, t_i) | \mathbf{x}_i \in \mathbf{R}^d, t_i \in R, i = 1, \dots, N\}$, activation function $g(x)$, maximum number $\tilde{N}_{\max}$ of hidden nodes, maximum number k of trials of assigning random hidden nodes at each step, and expected learning accuracy ϵ ,

step 1 Initialization: Let $\tilde{N} = 0$ and residual error $E = t$, where $t =$

$$[t_1, \dots, t_N]^T.$$

step 2 Learning step:

while $\tilde{N} < \tilde{N}_{\max}$ and $\|E\| > \epsilon$

(a) Increase by 1 the number of hidden nodes $\tilde{N}$: $\tilde{N} = \tilde{N} + 1$.

5.2. Proposed Enhanced Incremental Extreme Learning Machine (EI-ELM) 94

(b) For $i = 1 : k$

- i. Assign random parameters $\mathbf{a}_{(i)}$ and $b_{(i)}$ for the new hidden node $\tilde{N}$.
- ii. Calculate the output weight $\beta_{(i)}$ for the new hidden node:

$$\beta_{(i)} = \frac{E \cdot H_{(i)}^T}{H_{(i)} \cdot H_{(i)}^T} \quad (5.8)$$

- iii. Calculate the residual error after adding the new hidden node $\tilde{N}$:

$$E_{(i)} = E - \beta_{(i)} \cdot H_{(i)} \quad (5.9)$$

(c) end For

- (d) Let $i^* = \{i | \min_{1 \leq i \leq k} \|E_{(i)}\|\}$. Set $E = E_{(i)}$, $\mathbf{a}_{\tilde{N}} = \mathbf{a}_{(i^*)}$, $b_{\tilde{N}} = b_{(i^*)}$, and $\beta_{\tilde{N}} = \beta_{(i^*)}$.

endwhile

Before learning, there is no node in the network and the residual error is initially set as the expected target vector of the training data set as shown in *step 1*. Learning will stop when the number $\tilde{N}$ of hidden neurons has exceeded the predefined maximum number $\tilde{N}_{\max}$ or the residual error E becomes less than the expected one. *step 2(b) - 2(d)* will get the most appropriate hidden node to be added after k trials of selecting hidden nodes at each step. $H_{(i)}$ of equation (5.8) is the activation vector of the new node for all the N training samples for the i -th trial of each step and $\beta_{(i)}$ is the corresponding output

weight for i -th trial of hidden node. $E_{(i)}$ of equation (5.9) is the residual error vector if that i -th trial of hidden node is added. The E in the right side of equation (5.9) represents the residual error vector before the new node added. *step 2(d)* choose the most appropriate hidden node (i^* -th trial at that step, $1 \leq i^* \leq k$).

5.3 Performance Evaluation

In this section, the performance of the proposed improved incremental learning algorithm is compared with the original I-ELM on benchmark regression problems from UCI database [146]. The specification of these benchmark problems are shown in Table 5.1. In our experiments, all the inputs (attributes) have been normalized into the range $[-1, 1]$ while the outputs (targets) have been normalized into $[0, 1]$. For the case of SLFN with sigmoidal activation function $g(x) = \frac{1}{1+e^{-x}}$ the input weights $\mathbf{a}_i$ and hidden biases b_i are randomly chosen from the range $[-1, 1]$. For SLFN with RBF activation function $\phi(\mathbf{x}) = \exp(-\gamma\|\mathbf{x} - \mu\|^2)$, the centers μ_i are randomly chosen from the range $[-1, 1]$ whereas the impact factor γ is chosen from the range $(0, 0.5)$. For SLFN, our target error is $\epsilon = 0.01$. All the simulations are running in MATLAB 6.5 environment and the same PC with Pentium 4 2.99 GHZ CPU and 1G RAM. For each problem, the average results over 20 trials are obtained for EI-ELM and I-ELM over 20-30 runs.

5.3. Performance Evaluation

96

Name	No. of Observations		Attributes
	Training data	Testing data	
Abalone	2000	2177	8
Ailerons	7154	6596	39
Airplane	450	500	9
Auto Price	80	79	15
Bank	4500	3692	8
Boston	250	256	13
California	8000	12640	8
Census (House8L)	10000	12784	8
ComputerActivity	4000	4192	12
Delta Ailerons	3000	4129	5
Delta Elevators	4000	5517	6
Kinematics	4000	4192	8
Machine CPU	100	109	6
Puma	4500	3692	8
Pyrim	40	34	26
Servo	80	87	4

Table 5.1: Specification of 16 Benchmarking Regression Datasets

5.3.1 Comparison between EI-ELM and I-ELM with the same number of hidden nodes

We first compare the generalization performance of EI-ELM and I-ELM with the same number of hidden nodes. Table 5.2 and Table 5.3 show the performance of EI-ELM and I-ELM with 200 hidden additive and RBF nodes

respectively. Seen from Table 5.2 and Table 5.3, the testing root mean square error (RMSE) of EI-ELM are generally smaller than that of the original I-ELM when both EI-ELM and I-ELM use the same number of hidden nodes. It shows that EI-ELM may achieve faster convergence rate than I-ELM under the same nodes. This can also be verified from Fig. 5.1 which shows testing error curves of EI-ELM and I-ELM for both Airplane and Census cases. Furthermore, as observed from Table 5.2 and Table 5.3, EI-ELM obtain much smaller standard deviation of testing RMSE than I-ELM. This means that the performance of EI-ELM is more stable than I-ELM.

Figure 5.1: The testing error updating curves of EI-ELM and I-ELM

The training time curves for several cases are shown in Fig 5.2 (for

5.3. Performance Evaluation

98

Name	EI-ELM (Sigmoid, $k = 10$)			I-ELM (Sigmoid, $k = 1$)		
	Mean	Dev	Time(s)	Mean	Dev	Time(s)
Abalone	0.0818	0.0020	2.5801	0.0920	0.0046	0.2214
Ailerons	0.0558	0.0024	9.5017	0.1023	0.0353	0.7547
Airplane	0.0804	0.0039	0.5669	0.1016	0.0093	0.0499
Auto Price	0.0896	0.0022	0.3141	0.0977	0.0069	0.0329
Bank	0.0631	0.0031	3.9838	0.1173	0.0068	0.3237
Boston	0.1055	0.0098	0.4332	0.1167	0.0112	0.0515
California	0.1494	0.0015	9.3336	0.1683	0.0049	0.5448
Census (8L)	0.0829	0.0012	11.476	0.0923	0.0023	0.8667
ComputerActivity	0.0941	0.0028	3.7511	0.1201	0.0125	0.2794
Delta Ailerons	0.0445	0.0062	2.7735	0.0525	0.0078	0.2620
Delta Elevators	0.0582	0.0032	3.7971	0.0740	0.0126	0.2708
Kinematics	<u>0.1393</u>	0.0028	3.4373	<u>0.1418</u>	0.0033	0.2810
Machine CPU	<u>0.0466</u>	0.0060	0.3112	<u>0.0504</u>	0.0079	0.0234
Puma	<u>0.1840</u>	0.0017	3.9531	<u>0.1861</u>	0.0041	0.3236
Pyrim	0.1414	0.0341	0.3062	0.1867	0.0628	0.0374
Servo	0.1518	0.0116	0.3235	0.1662	0.0124	0.0218

Table 5.2: Performance comparison between EI-ELM and I-ELM (both with 200 Sigmoid hidden nodes)

Sigmoid additive node case). Seen from Fig 5.2, the spent learning time is still linearly increasing with the learning steps, which is the same as I-ELM's conclusion. Since the training time of I-ELM linearly increases with the number of hidden nodes, in theory the training time of EI-ELM should be around k times that of I-ELM, where k is optimal trial times. In our

5.3. Performance Evaluation

99

Name	EI-ELM (RBF, $k = 10$)			I-ELM (RBF, $k = 1$)		
	Mean	Dev	Time(s)	Mean	Dev	Time(s)
Abalone	0.0829	0.0027	5.6006	0.0938	0.0053	0.5030
Ailerons	0.0774	0.0129	36.016	0.1430	0.0298	3.2769
Airplane	0.0633	0.0057	0.9578	0.0992	0.0166	0.0751
Auto Price	0.1139	0.0189	0.4031	0.1261	0.0255	0.0468
Bank	0.0730	0.0022	9.8079	0.1157	0.0097	0.7782
Boston	0.1077	0.0084	0.7972	0.1320	0.0126	0.0657
California	0.1503	0.0022	17.133	0.1731	0.0081	1.3656
Census (8L)	0.0810	0.0016	19.922	0.0922	0.0029	1.7928
ComputerActivity	0.1153	0.0021	10.092	0.1552	0.0282	0.8220
Delta Ailerons	0.0448	0.0065	4.6169	0.0632	0.0116	0.4327
Delta Elevators	0.0575	0.0047	7.3541	0.0790	0.0123	0.6321
Kinematics	0.1213	0.0017	8.3114	0.1555	0.0122	0.6953
Machine CPU	0.0554	0.0148	0.4114	0.0674	0.0177	0.0447
Puma	0.1752	0.0022	9.7983	0.1913	0.0180	0.7872
Pyrim	0.1209	0.0431	0.4423	0.2241	0.1752	0.0434
Servo	0.1379	0.0151	0.4031	0.1524	0.0200	0.0391

Table 5.3: Performance comparison between EI-ELM and I-ELM (both with 200 RBF hidden nodes)

simulations, we set $k = 10$. The training time of EI-ELM should be almost 10 times of I-ELM. However, EI-ELM needs to seek the best parameters from k trial times, which also takes computation time, furthermore, for all the applications, we need to store some temp variables, which normally use large memory. When memory reaches computer's choke-point, it will disturb

CPU's computation remarkably. Especially for applications with large scale data, the influence becomes more obvious. So the training time relationship between EI-ELM and I-ELM should be that the training time of EI-ELM should be $k + \eta$ times of I-ELM, where η is impact factor determined by seeking time, CPU and memory, etc. The training time results of Table 5.2 and Table 5.3 verify our conclusion.

Figure 5.2: Training time spent by EI-ELM is linearly increased with the number of hidden nodes to be added

5.3.2 Comparison between EI-ELM and I-ELM with the different number of hidden nodes

5.3. Performance Evaluation

101

Problems	EI-ELM (50 Sigmoid hidden nodes)						I-ELM (500 Sigmoid hidden nodes, $k = 1$)		
	$k = 10$			$k = 20$					
	Mean	Dev	Time(s)	Mean	Dev	Time(s)	Mean	Dev	Time(s)
Abalone	<u>0.0878</u>	0.0033	0.6506	<u>0.0876</u>	0.0015	1.5785	<u>0.0876</u>	0.0033	0.7695
Ailerons	0.0640	0.0066	2.3766	0.0571	0.0022	6.2519	0.0824	0.0232	1.8810
Airplane	0.0922	0.0061	0.1389	<u>0.0862</u>	0.0040	0.2921	<u>0.0898</u>	0.0067	0.1466
Auto Price	0.0924	0.0112	0.0814	0.0897	0.0104	0.1658	0.0948	0.0158	0.0561
Bank	0.1066	0.0058	0.9965	0.0896	0.0036	3.1058	0.0757	0.0032	0.7914
Boston	<u>0.1133</u>	0.0101	0.1065	<u>0.1102</u>	0.0061	0.2232	<u>0.1084</u>	0.0096	0.1033
California	<u>0.1591</u>	0.0034	2.2423	<u>0.1548</u>	0.0033	4.9486	<u>0.1543</u>	0.0019	1.5665
Census (8L)	<u>0.0899</u>	0.0017	2.8655	<u>0.0865</u>	0.0011	6.1100	<u>0.0871</u>	0.0018	2.1199
Computer	0.1075	0.0057	0.9342	0.0991	0.0036	2.3311	0.1057	0.0078	0.7185
Delta Ailerons	<u>0.0474</u>	0.0062	0.7006	<u>0.0467</u>	0.0042	1.4570	<u>0.0468</u>	0.0052	0.6340
Delta Elevators	0.0615	0.0049	0.9502	0.0586	0.0038	2.5385	0.0640	0.0055	0.6516
Kinematics	<u>0.1420</u>	0.0029	0.8655	<u>0.1416</u>	0.0019	2.9017	<u>0.1406</u>	0.0014	0.7117
Machine CPU	<u>0.0498</u>	0.0155	0.0750	<u>0.0467</u>	0.0148	0.1577	<u>0.0474</u>	0.0040	0.0645
Puma	<u>0.1846</u>	0.0018	0.9856	<u>0.1827</u>	0.0017	2.7264	<u>0.1856</u>	0.0039	0.7983
Pyrim	0.1514	0.0419	0.0782	0.1300	0.0405	0.1533	0.1712	0.0626	0.0810
Servo	0.1634	0.0129	0.0795	<u>0.1558</u>	0.0121	0.1611	<u>0.1589</u>	0.0124	0.0642

Table 5.4: Performance comparison between EI-ELM with 50 Sigmoid hidden nodes and I-ELM with 500 Sigmoid hidden nodes

During our simulations we also try the case where EI-ELM and I-ELM are given different number of hidden nodes. The comparison between EI-ELM with 50 hidden nodes and I-ELM with 500 hidden nodes are shown in Table 5.4 and Table 5.5. Here we set $k = 10$ and $k = 20$ for EI-ELM respectively. As observed from Table 5.4 and Table 5.5, the testing RMSE obtained by EI-ELM with 50 hidden nodes are usually slightly better than

5.3. Performance Evaluation

102

Problems	EI-ELM (50 RBF hidden nodes)						I-ELM (500 RBF		
	$k = 10$			$k = 20$			hidden nodes, $k = 1$)		
	Mean	Dev	Time(s)	Mean	Dev	Time(s)	Mean	Dev	Time(s)
Abalone	<u>0.0907</u>	0.0034	1.4036	<u>0.0871</u>	0.0023	3.0006	<u>0.0872</u>	0.0022	1.2121
Ailerons	0.0973	0.0229	9.0306	0.0775	0.0033	19.071	0.1129	0.0295	8.1818
Airplane	0.0943	0.0168	0.2347	<u>0.0813</u>	0.0102	0.5487	<u>0.0772</u>	0.0082	0.1940
Auto Price	0.1187	0.0159	0.0998	0.1104	0.0148	0.2110	0.1231	0.0133	0.1189
Bank	0.0989	0.0031	2.4460	<u>0.0888</u>	0.0023	5.4199	<u>0.0843</u>	0.0058	1.9382
Boston	0.1197	0.0107	0.1845	0.1171	0.0078	0.3621	0.1214	0.0103	0.1872
California	<u>0.1624</u>	0.0049	4.2339	<u>0.1579</u>	0.0027	8.8326	<u>0.1582</u>	0.0027	3.8482
Census (8L)	<u>0.0864</u>	0.0026	4.9858	<u>0.0846</u>	0.0020	11.796	<u>0.0860</u>	0.0018	4.8536
Computer	0.1295	0.0068	2.4905	0.1201	0.0024	5.5878	0.1358	0.0177	2.1267
Delta Ailerons	0.0469	0.0067	1.1800	0.0466	0.0039	2.4763	0.0544	0.0076	1.0361
Delta Elevators	0.0603	0.0049	1.8515	0.0602	0.0039	4.2506	0.0685	0.0099	1.5399
Kinematics	0.1346	0.0025	2.0913	0.1306	0.0019	4.6727	0.1425	0.0095	1.7042
Machine CPU	0.0622	0.0281	0.1067	0.0511	0.0114	0.2031	0.0614	0.0274	0.0875
Puma	0.1789	0.0020	2.4465	0.1770	0.0012	5.2821	0.1850	0.0119	1.9709
Pyrim	0.1214	0.0345	0.1016	0.0989	0.0286	0.2079	0.2179	0.1545	0.1071
Servo	0.1487	0.0133	0.0985	<u>0.1434</u>	0.0120	0.1958	<u>0.1410</u>	0.0151	0.0982

Table 5.5: Performance comparison between EI-ELM with 50 RBF hidden nodes and I-ELM with 500 RBF hidden nodes

those obtained by I-ELM with 500 hidden nodes. It further demonstrates that EI-ELM can converge faster than I-ELM. It can be also seen that the training time spent by I-ELM with 500 hidden nodes and EI-ELM with 50 hidden nodes are in the same order. However parsimonious network architecture implies faster prediction time. EI-ELM only has 50 hidden nodes while I-ELM has 500 hidden nodes EI-ELM has faster prediction time than

Figure 5.3: Testing RMSE performance comparison between EI-ELM and I-ELM (with Sigmoid hidden nodes) for Abalone case

I-ELM. Fig 5.3 shows the testing error curves of I-ELM and EI-ELM with sigmoid activation functions for the Abalone case. It can be seen that in order to obtain the testing RMSE 0.09, EI-ELM only needs 21 nodes and spends 0.2722 seconds on training while I-ELM needs 331 nodes and spends 0.5231 seconds on training.

5.3.3 Effects of number of selecting trials k

Effects of number of selecting trials k on the performance of EI-ELM have also been investigated in our study. Fig. 5.4 and Fig. 5.5 show that the generalization performance and stability of EI-ELM with 100 sigmoid hidden nodes will become better when number of selecting trials k increases for the Airplane case. Fig 5.6 shows the testing RMSE curves of EI-ELM with increasing number of hidden nodes and different number of selecting trials k . It can be seen that the effect of number of selecting trials k on the generalization performance and its stability tend to become stable after k increases up to certain number (20 in Airplane case). The above observations are true for the rest cases as well.

Figure 5.4: Effect of number of selecting trials k on the generalization performance of EI-ELM in Airplane case

Figure 5.5: Effect of number of selecting trials k on the stability of the generalization performance of EI-ELM in Airplane case

Figure 5.6: Testing RMSE updating progress with new hidden nodes added and different number of selecting trials k in Airplane case

5.4 Discussion

Based on the analysis of simulation results, EI-ELM can be regarded as k computation for a new neuron. Since the training time of I-ELM is linearly increased, thus EI-ELM is also linearly increased (c.f Figure 5.2). Its training time is k times than I-ELM.

As we know, the convergent rate analysis of neural network is always a critical issue, which directly leads to the complexity of neural network structure and computation. It is well known that a lower complexity implies a reduced generalization capability and a reduced accuracy. When convergent rate of neural networks can be calculated accurately, we can determine the number of hidden neurons to satisfy target error. In Kowk's paper[28], neural networks are also constructed by the similar iteration $f_n = f_{n-1} + \beta_n g_n$, where g_n is chosen to maximize $\frac{\langle e_{n-1}, g \rangle^2}{\|g\|^2}$. We should note that our difference is that for new added neurons, they are random in our theory. Although without the maximum condition, the approximation capability of neural networks is still correct. Next, Kowk, et al pointed out the convergent rate of the constructive algorithm in the follow lemma:

Lemma 5.4.1. {P.1135 of [28]} *If the target function is an exact summation of basis functions form Γ as in $f = \sum_{j=1}^m v_j g_j^*$ and the network is constructed as Kowk's form, then*

$$\|e_n\|^2 < \frac{b_f^2 \|e_0\|^2}{n \|e_0\|^2 + b_f^2} \quad (5.10)$$

where $b_f \equiv m \max_{j=1, \dots, m} |v_j| \|g_j^*\|$. In other words, $\|e_n\|^2 = O(\frac{1}{n})$

In order to distinguish our method from Kowk's method, for a new n -th hidden node, it is denoted as g_n in our method, while it is denoted as $\bar{g}_n$ in Kowk's method. As k increases, g_n definitely approximates to $\bar{g}_n$ from statistical viewpoint. Therefore, when trial times k is large enough, neural networks with EI-ELM have higher probability to achieve the convergent rate $O(\frac{1}{n})$. In Kowk's paper, in order to maximize $\frac{\langle e_{n-1}, g \rangle^2}{\|g\|^2}$, they normally used gradient method to calculate the corresponding optimal hidden nodes, however, sometimes for those non-differential functions, gradient method is not feasible. EI-ELM gives a replaceable method to realize the same target.

5.5 Conclusion

In this chapter, based on our theory proved in Chapter 3, we proposed an enhanced incremental algorithm EI-ELM for I-ELM. Similar to I-ELM, EI-ELM also randomly generates hidden nodes and then analytically determines the output weights. EI-ELM does not only work for differentiable activation functions but also for noncontinuous (such as threshold) activation functions¹. It should be noted that I-ELM is a specific case of EI-ELM when

¹Both I-ELM and EI-ELM provide direct learning methods for the threshold network which may not be trained directly before the work done by Huang, et al[3]. Due to the length of this letter, the performance analysis of EI-ELM with the threshold activation

$k = 1$. The difference between I-ELM and EI-ELM is that at each learning step among several randomly generated hidden nodes EI-ELM picks the best hidden node which leads to the smallest residual error. Compared with the original I-ELM, EI-ELM can achieve faster convergence rate and more compact network architectures, which have been further verified by the simulation results on many benchmark real-world regression problems.

Chapter 6

Convex Incremental Extreme Learning Machine (CI-ELM)

In Chapter 3, we prove that SLFNs with random nodes can work as universal approximators. The corresponding algorithm, I-ELM, can provide good generalization performance at a fast learning speed. In this chapter, an improved method for I-ELM (CI-ELM) is proposed, which consists of a convex structure. The more important point is that the new algorithm still ensures the approximation capability of neural networks. Since it adjusts the existing neurons, while retaining the same simplicity and efficiency, such a new network can achieve a faster convergence rate than I-ELM.

6.1 Introduction

Neural networks have been successfully applied in many applications. Out of many kinds of neural networks single-hidden-layer feedforward neural networks (SLFNs) have been investigated more thoroughly. SLFN network functions with n hidden nodes can be represented by

$$f_n(\mathbf{x}) = \sum_{i=1}^n \beta_i g_i(\mathbf{x}), \mathbf{x} \in \mathbf{R}^d, \beta_i \in R \quad (6.1)$$

where g_i denotes the hidden node output function and β_i is the weight of the connection between the i -th hidden node and the output node. For additive nodes with activation function g , g_i is defined as

$$g_i(\mathbf{x}) = g(\mathbf{a}_i \cdot \mathbf{x} + b_i), \mathbf{a}_i \in \mathbf{R}^d, b_i \in R \quad (6.2)$$

where $\mathbf{a}_i$ is the weight vector connecting the input layer to the i -th hidden node and b_i is the bias of the i -th hidden node. $\mathbf{a}_i \cdot \mathbf{x}$ denotes the inner product of vectors $\mathbf{a}_i$ and $\mathbf{x}$ in $\mathbf{R}^d$. For RBF nodes with activation function g , g_i is defined as

$$g_i(\mathbf{x}) = g\left(\frac{\|\mathbf{x} - \mathbf{a}_i\|}{b_i}\right), \mathbf{a}_i \in \mathbf{R}^d, b_i \in R^+ \quad (6.3)$$

where $\mathbf{a}_i$ and b_i are the center and impact factor of i -th RBF node, β_i the weight connecting the i -th RBF hidden node to the output node. R^+ indicates the set of all positive real value.

From a mathematical point of view, research on the approximation capabilities of SLFNs has focused on two aspects: universal approximation on

compact input sets and approximation in a set consisting of finite number of training samples. According to conventional neural network theories[8], [13], [18]-[19], SLFNs with additive or RBF hidden nodes are universal approximators when all the parameters of the networks are allowed adjustable. However, as observed in most neural network implementations tuning all the parameters of the networks may cause learning complicated and inefficient, and it may be difficult to directly train networks with non-differential activation functions such as threshold networks. Unlike conventional neural network theories, in Chapter 3, we rigorously proves in an incremental constructive method that in order to let SLFNs work as universal approximators one may simply randomly choose hidden nodes (with random values for both the hidden biases and input weights linking the input layer to the hidden layer) and then only need to analytically calculate (instead of tuning) the output weights linking the hidden layer and the output layer. In such SLFNs implementations, the activation functions for additive nodes can be any bounded nonconstant piecewise continuous functions $g : R \rightarrow R$ and the activation functions for RBF nodes can be any integrable piecewise continuous functions $g : R \rightarrow R$ and $\int_R g(x)dx \neq 0$. The resulting method referred to as Incremental Extreme Learning Machine (I-ELM) [150] randomly adds nodes to the hidden layer one by one and freezes the output weights of the existing hidden nodes when a new hidden node is added. I-ELM is not only efficient for SLFNs with continuous (including non-differentiable) activation functions but also for SLFNs with piecewise continuous (such as threshold)

6.2. Proposed Convex Incremental Extreme Learning Machine (CI-ELM) 112

activation functions. Extreme Learning Machine (ELM) with fixed network architectures has also been investigated in several earlier works [1]-[7].

In this chapter, Barron's convex optimization learning method [19] is incorporated into I-ELM and the improved method is referred to as convex I-ELM (CI-ELM). Different from I-ELM, based on a convex optimization method CI-ELM recalculates the output weights of the existing hidden nodes after a new hidden node is added. Different from Barron's work [19], [151], CI-ELM still adds randomly generated hidden nodes (randomly generated input weights and biases for the newly added hidden nodes). The more important point is that the new algorithm still ensures the approximation capability of neural networks. The proposed algorithm (CI-ELM) can achieve faster convergence rates and more compact network architectures than I-ELM while retaining the I-ELM's simplicity and efficiency.

6.2 Proposed Convex Incremental Extreme Learning Machine (CI-ELM)

Without loss of generality, we assume that the network has only one linear output node. All the analysis can be easily extended to multi nonlinear output nodes cases. Let $e_n \equiv f - f_n$ denote the residual error function for the current network f_n with n hidden nodes where $f \in L^2(X)$ is the target

6.2. Proposed Convex Incremental Extreme Learning Machine (CI-ELM) 113

function. The I-ELM algorithm [150] does not recalculate the output weights of the existing hidden nodes after a new hidden is added. In other words, once the output weights of hidden nodes are calculated they will remain frozen and will not be changed any more. That means, the output function f_n of the SLFNs of size n trained by the I-ELM is:

$$f_n(\mathbf{x}) = f_{n-1}(\mathbf{x}) + \beta_n g_n(\mathbf{x}) \quad (6.4)$$

In I-ELM, one randomly generates the newly added hidden node g_n and then analytically calculates the corresponding output weight $\beta_n = \frac{\langle e_{n-1}, g_n \rangle}{\|g_n\|^2}$ of the new hidden node regardless of those previous existing nodes. Although hidden node parameters are chosen randomly in I-ELM, as rigorously proved by Huang, et al[150] I-ELM can work as a universal approximator:

Theorem 6.2.1. [150] *Given any bounded nonconstant piecewise continuous function $g : \mathbf{R} \rightarrow \mathbf{R}$ for additive nodes or any integrable piecewise continuous function $g : \mathbf{R} \rightarrow \mathbf{R}$ and $\int_{\mathbf{R}} g(x)dx \neq 0$ for RBF nodes, for any continuous target function f and any randomly generated function sequence $\{g_n\}$, $\lim_{n \rightarrow \infty} \|f - f_n\| = 0$ holds with probability one if*

$$\beta_n = \frac{\langle e_{n-1}, g_n \rangle}{\|g_n\|^2} \quad (6.5)$$

Theorem 6.2.1 not only shows the universal approximation capability of I-ELM but also proposes a method on how to calculate the output weights

6.2. Proposed Convex Incremental Extreme Learning Machine (CI-ELM) 114

of the newly added hidden node. In fact, there may exist different ways to find the output weights of such SLFNs with different characters. Different from the network updating function (6.4) used in I-ELM, CI-ELM adopts the Barron's convex optimization concept [19] by remaining the main feature of I-ELM: hidden nodes are randomly generated:

$$f_n = \beta_n f_{n-1} + (1 - \beta_n) g_n \quad (6.6)$$

where $0 \leq \beta_n \leq 1$. In fact, we have:

Theorem 6.2.2. *Given any bounded nonconstant piecewise continuous function $g : \mathbf{R} \rightarrow \mathbf{R}$ for additive nodes or any integrable piecewise continuous function $g : \mathbf{R} \rightarrow \mathbf{R}$ and $\int_{\mathbf{R}} g(x)dx \neq 0$ for RBF nodes, for any continuous target function f and any randomly generated function sequence $\{g_n\}$, for the iteration expression $f_n = \beta_n f_{n-1} + (1 - \beta_n) g_n$, $\lim_{n \rightarrow \infty} \|f - f_n\| = 0$ holds with probability one if*

$$\beta_n = \frac{\langle e_{n-1}, g_n - f_{n-1} \rangle}{\|g_n - f_{n-1}\|^2}. \quad (6.7)$$

Proof. The proof method of I-ELM [150]{P.882} can be adopted to prove the validity of this theorem without any major modification. The proof consists of two steps: a) we first prove that the sequence $\{\|e_n\|\}$ converges; b) we then further prove that the sequence $\{\|e_n\|\}$ converges to zero.

6.2. Proposed Convex Incremental Extreme Learning Machine (CI-ELM)

115

(a) According to equation (6.6), we have

$$\begin{aligned}
 e_n &= f - f_n \\
 &= f - (\beta_n f_{n-1} + (1 - \beta_n)g_n) \\
 &= f - (1 - (1 - \beta_n))f_{n-1} - (1 - \beta_n)g_n \\
 &= f - f_{n-1} + (1 - \beta_n)f_{n-1} - (1 - \beta_n)g_n \\
 &= e_{n-1} - \beta_n(g_n - f_{n-1})
 \end{aligned} \tag{6.8}$$

Let $\Delta = \|e_{n-1}\|^2 - \|e_n\|^2$, then according to equation (6.8), we have

$$\begin{aligned}
 \Delta &= \|e_{n-1}\|^2 - \|e_{n-1} - \beta_n(g_n - f_{n-1})\|^2 \\
 &= 2\beta_n \langle e_{n-1}, g_n - f_{n-1} \rangle - \beta_n^2 \|g_n - f_{n-1}\|^2 \\
 &= \|g_n - f_{n-1}\|^2 \left(\frac{\langle e_{n-1}, g_n - f_{n-1} \rangle^2}{\|g_n - f_{n-1}\|^4} - \left(\beta_n - \frac{\langle e_{n-1}, g_n - f_{n-1} \rangle}{\|g_n - f_{n-1}\|^2} \right)^2 \right)
 \end{aligned}$$

Δ is maximized iff $\beta_n = \frac{\langle e_{n-1}, g_n - f_{n-1} \rangle}{\|g_n - f_{n-1}\|^2}$, meaning that $\|e_n\| = \|f - ((1 - \beta_n)f_{n-1} + \beta_n g_n)\|$ achieves its minimum iff

$$\beta_n = \frac{\langle e_{n-1}, g_n - f_{n-1} \rangle}{\|g_n - f_{n-1}\|^2} \tag{6.9}$$

Moreover, we have the following fact: when $\beta_n = \frac{\langle e_{n-1}, g_n - f_{n-1} \rangle}{\|g_n - f_{n-1}\|^2}$, $\Delta = \Delta_{\max} = \frac{\langle e_{n-1}, g_n - f_{n-1} \rangle^2}{\|g_n - f_{n-1}\|^2} \geq 0$. So the sequence $\{\|e_n\|\}$ are decreasing and bounded below by zero and the sequence $\{\|e_n\|\}$ converges.

(b) Seen from the proof of the original I-ELM [150], the sequence $\{\|e_n\|\}$

converges to zero as long as the three sufficient conditions are satisfied:

1) $\text{span}\{G(\mathbf{x}, \mathbf{a}, b) : (\mathbf{a}, b) \in \mathbf{R}^d \times R\}$ is dense in L^2 (to support b.1 of

6.2. Proposed Convex Incremental Extreme Learning Machine (CI-ELM)

116

the proof of I-ELM [150], p. 882); 2) $e_n \perp (e_{n-1} - e_n)$ (to support b.2 of the proof of I-ELM [150], p. 883); and 3) g is a bounded nonconstant piecewise continuous function (to support b.3 of the proof of I-ELM [150], p. 884). Since conditions 1) and 3) have been given as the preconditions of the theorem, in order to prove $\lim_{n \rightarrow +\infty} \|e_n\| = 0$ we only need to prove $e_n \perp (e_{n-1} - e_n)$.

From formula (6.8) we have

$$\begin{aligned}
 \langle e_n, g_n - f_{n-1} \rangle &= \langle e_{n-1} - \beta_n(g_n - f_{n-1}), g_n - f_{n-1} \rangle \\
 &= \langle e_{n-1}, g_n - f_{n-1} \rangle - \beta_n \langle g_n - f_{n-1}, g_n - f_{n-1} \rangle \\
 &= \langle e_{n-1}, g_n - f_{n-1} \rangle - \beta_n \|g_n - f_{n-1}\|^2 \quad (6.10)
 \end{aligned}$$

According to formula (6.9), we further have

$$\begin{aligned}
 \langle e_n, g_n - f_{n-1} \rangle &= \langle e_{n-1}, g_n - f_{n-1} \rangle - \frac{\langle e_{n-1}, g_n - f_{n-1} \rangle}{\|g_n - f_{n-1}\|^2} \cdot \langle g_n - f_{n-1}, g_n - f_{n-1} \rangle \\
 &= \langle e_{n-1}, g_n - f_{n-1} \rangle - \langle e_{n-1}, g_n - f_{n-1} \rangle = 0 \quad (6.11)
 \end{aligned}$$

According to formula (6.8), $e_n - e_{n-1} = -\beta_n(g_n - f_{n-1})$. Thus, from formula (6.11) we have

$$\langle e_n, e_n - e_{n-1} \rangle = \langle e_n, -\beta_n(g_n - f_{n-1}) \rangle = 0 \quad (6.12)$$

which means $e_n \perp (e_n - e_{n-1})$.

This completes the proof of this theorem.

□

6.2. Proposed Convex Incremental Extreme Learning Machine (CI-ELM)

117

Seen from formula (6.6) and Theorem 6.2.2, if a new hidden node is added the output weight of the newly added hidden node is set as: $\beta_n = \frac{\langle e_{n-1}, g_n - f_{n-1} \rangle}{\|g_n - f_{n-1}\|^2}$ and the output weights of the existing hidden nodes will be multiplied by $1 - \beta_n = 1 - \frac{\langle e_{n-1}, g_n - f_{n-1} \rangle}{\|g_n - f_{n-1}\|^2}$. Similar to Kwok and Yeung[28], a consistent estimate of β_n based on the training set is

$$\beta_n = \frac{E \cdot [E - (F - H)]^T}{[E - (F - H)] \cdot [E - (F - H)]^T} = \frac{\sum_{p=1}^N e(p) [e(p) - (f(p) - h(p))]}{\sum_{p=1}^N [e(p) - (f(p) - h(p))]^2} \quad (6.13)$$

where $h(p)$ is the activation of the new hidden node for the input of p -th training sample and $e(p)$ is the corresponding residual error before this new hidden node is added. $H = [h(1), \dots, h(N)]^T$ is the activation vector of the new node for all the N training samples and $E = [e(1), \dots, e(N)]^T$ is the residual vector before this new hidden node added. $F = [t_1, \dots, t_N]^T$ is the target vector of the target function, where t_p is the target output of the p -th training data $\mathbf{x}_p$.

In real applications, one may not really wish to get zero approximation error by adding infinite nodes to the network; a maximum number of hidden nodes is normally given. Thus, such an incremental constructive method for SLFNs can be summarized as follows:

Algorithm CI-ELM: Given a training set $\aleph = \{(\mathbf{x}_i, t_i) | \mathbf{x}_i \in \mathbf{R}^d, t_i \in R, i = 1, \dots, N\}$, activation function $g(x)$, maximum number of hidden nodes $\tilde{N}_{\max}$ and expected learning accuracy ϵ ,

6.2. Proposed Convex Incremental Extreme Learning Machine (CI-ELM)

118

step 1 Initialization: Let $\tilde{N} = 0$ and residual error $E = t$, where $t = [t_1, \dots, t_N]^T$.

step 2 Learning step:

while $\tilde{N} < \tilde{N}_{\max}$ and $\|E\| > \epsilon$

(a) Increase by 1 the number of hidden nodes $\tilde{N}$: $\tilde{N} = \tilde{N} + 1$.

(b) Randomly assign input weight $\mathbf{w}_{\tilde{N}}$ and bias $b_{\tilde{N}}$ for new hidden node $\tilde{N}$.

(c) Calculate the output weight $\beta_{\tilde{N}}$ for the newly added hidden node:

$$\beta_{\tilde{N}} = \frac{E \cdot [E - (F - H_{\tilde{N}})]^T}{[E - (F - H_{\tilde{N}})] \cdot [E - (F - H_{\tilde{N}})]^T} \quad (6.14)$$

(d) Recalculate the output weight vectors of all existing hidden nodes if $\tilde{N} > 1$:

$$\beta_i = (1 - \beta_{\tilde{N}})\beta_i, \quad i = 1, \dots, \tilde{N} - 1 \quad (6.15)$$

(e) Calculate the residual error after adding the new hidden node $\tilde{N}$:

$$E = (1 - \beta_{\tilde{N}})E + \beta_{\tilde{N}}(F - H_{\tilde{N}}) \quad (6.16)$$

endwhile

β_i in the right side of equation (6.15) represents the output weight of i -th hidden node before a new hidden is added and β_i in the left side represents the output weight of i -th hidden node after a new hidden added. The residual

6.2. Proposed Convex Incremental Extreme Learning Machine (CI-ELM)

119

error E in the right side of equation (6.16) represents the residual error vector before the new node added and the E in the left side represents the residual error vector after the new node added, which is consistent to $E_{\tilde{N}} = f - f_{\tilde{N}} = (1 - \beta_{\tilde{N}})E_{\tilde{N}-1} + \beta_{\tilde{N}}(f - g_{\tilde{N}})$.

Remark 1: For additive hidden nodes with activation function g and hidden node parameters $(\mathbf{a}, b)$, $G(\mathbf{x}, \mathbf{a}, b)$ is defined as $G(\mathbf{x}, \mathbf{a}, b) = g(\mathbf{a} \cdot \mathbf{x} + b)$. For RBF hidden nodes with activation function g and hidden node parameters $(\mathbf{a}, b)$, $G(\mathbf{x}, \mathbf{a}, b)$ is defined as $G(\mathbf{x}, \mathbf{a}, b) = g\left(\frac{\|\mathbf{x} - \mathbf{a}\|}{b}\right)$. According to [13, 18] both $\text{span}\{g(\frac{\mathbf{x} - \mathbf{a}}{b}) : (\mathbf{a}, b) \in \mathbf{R}^d \times \mathbf{R}^+\}$ and $\text{span}\{g(\mathbf{a} \cdot \mathbf{x} + b) : (\mathbf{a}, b) \in \mathbf{R}^d \times \mathbf{R}\}$ are dense in L^2 . Thus, Theorem 6.2.2 is valid for both bounded nonconstant piecewise continuous additive and RBF hidden nodes.

Remark 2: Seen from Theorem 6.2.2, when the network architecture is fixed (with fixed n) we have

Theorem 6.2.3. *Given any bounded nonconstant piecewise continuous function $g : \mathbf{R} \rightarrow \mathbf{R}$, if $\text{span}\{G(\mathbf{x}, \mathbf{a}, b) : (\mathbf{a}, b) \in \mathbf{R}^d \times \mathbf{R}\}$ is dense in L^2 , for any continuous target function f and any function sequence $\{g_n(\mathbf{x}) = G(\mathbf{x}, \mathbf{a}_n, b_n)\}$ randomly generated based on any continuous sampling distribution, $\lim_{n \rightarrow \infty} \|f - f_n\| = 0$ holds with probability one if the output parameters are determined by ordinary least square to minimize $\|f(\mathbf{x}) - \sum_{i=1}^n \beta_i g_i(\mathbf{x})\|$.*

Theorem 6.2.3 means that the Extreme Learning Machine (ELM) with fixed

6.2. Proposed Convex Incremental Extreme Learning Machine (CI-ELM) 120

network architectures [1]-[7] where the output parameters are determined by ordinary least square can work as universal approximators if the activation function g is bounded nonconstant piecewise and $\text{span}\{G(\mathbf{x}, \mathbf{a}, b) : (\mathbf{a}, b) \in \mathbf{R}^d \times R\}$ is dense in L^2 .

Remark 3: The conventional learning algorithms for SLFNs always need to adjust the hidden node parameters $(\mathbf{a}, b)$. Interestingly, Theorems 6.2.2 and 6.2.3 indicate that if SLFNs can work as universal approximators with adjustable hidden parameters $(\mathbf{a}, b)$ and one does not care much the network size, from function approximation point of view the hidden node parameters can actually be randomly generated and then fixed instead of being tuned.

Remark 4: It may be worth mentioning the differences between ELM and the earlier work done by Lowe [140]. Lowe [140] randomly selected RBF centers from the training data and fixed the impact factors of RBF nodes according to the spread of those RBF centers while ELM for RBF case randomly generates centers and impact factors respectively from $\mathbf{R}^d$ and R^+ according to any continuous sampling distribution. In ELM, the hidden node parameters are totally randomly generated and completely independent from the training data. If RBF centers and impact factors are selected based on the training data, it may give advantages to the training data and thus easily causes overfitting. ELM works not only for RBF network but also for SLFNs with other type of hidden nodes (i.e, additive nodes, etc). In addition, the universal approximation capability of ELM has been thoroughly proved.

6.3 Performance Evaluation

6.3.1 Benchmarking with Regression Problem

Problems	# Training data	# Testing data	Attributes
Abalone	2000	2177	8
Auto Price	80	79	15
Boston	250	256	13
California	8000	12640	8
Census	10000	12784	8
Delta Ailerons	3000	4129	5
Delta Elevators	4000	5517	6
Machine CPU	100	109	6

Table 6.1: Specification of Benchmarking Datasets

In Chapter 3, we have demonstrated that I-ELM can outperform many popular algorithms like the stochastic gradient descent BP [92], SVR [100] and other well-known incremental learning algorithms like RAN [64] and MRAN [65] in many cases, thus we may only need to compare CI-ELM with I-ELM in this letter. Table 6.1 gives the specification of 8 real datasets [146] used in such comparisons. All the inputs of these datasets are normalized into the range $[-1, 1]$ in our experiments. All the simulations are running in MATLAB 6.5 environment and the same PC with Pentium IV 3.0 GHZ CPU and 1G RAM.

Two popular activation functions: $g(x) = 1/(1 + \exp(-x))$ and $g(\mathbf{x}) = \exp(-b\|\mathbf{x} - \mu\|^2)$ are used for additive and RBF nodes, respectively. The input weights and hidden biases are randomly chosen from the range $[-1, 1]$ for additive nodes. The centers are randomly chosen from the range $[-1, 1]$ and the impact factor b is chosen from the range $[0, 0.5]$ for RBF nodes.

Figure 6.1: Performance comparison (testing RMSE) between CI-ELM and I-ELM for Abalone case.

CI-ELM and I-ELM with up to 1000 hidden nodes have been run for all the datasets. Figure 6.1 shows the performance comparison (testing RMSE) between CI-ELM and I-ELM for abalone dataset. In fact, the same general-

Problems	Sigmoid		RBF	
	CI-ELM	I-ELM	CI-ELM	I-ELM
Abalone	0.2531	0.2214	0.4409	0.5030
Auto Price	0.0300	0.0329	0.0358	0.0468
Boston	0.0311	0.0515	0.0576	0.0657
California	0.6902	0.5448	1.2851	1.3656
Census	0.9330	0.8667	1.7113	1.7928
Delta Ailerons	0.2078	0.2620	0.3739	0.4327
Delta Elevators	0.2939	0.2708	0.5539	0.6321
Machine CPU	0.0342	0.0234	0.0376	0.0447

Table 6.2: Training time (seconds) comparison between CI-ELM and I-ELM with 200 hidden nodes.

ization performance trends have been obtained for all the tested cases.

Tables 6.2 and 6.3 show the average training time and generalization performance obtained over 20 trials for each case. As observed from Tables 6.2 and 6.3 CI-ELM and I-ELM spend almost the same training time for the same network size but CI-ELM generally obtains better generalization performance. As observed from Figure 6.1, CI-ELM has much faster convergence rate than I-ELM. In order to reach the same generalization performance CI-ELM may need much fewer hidden nodes. Thus CI-ELM may need less training time in order to reach the same generalization performance.

In Table 6.4 and Table 6.5, we record the number of neurons under given expected learning accuracy for 10 trials. Seen from Table 6.4 and Table

6.3. Performance Evaluation

124

Problems	Sigmoid				RBF			
	CI-ELM		I-ELM		CI-ELM		I-ELM	
	Mean	Dev	Mean	Dev	Mean	Dev	Mean	Dev
Abalone	0.0828	0.0030	0.0920	0.0046	0.0858	0.0029	0.0938	0.0053
Auto Price	0.0927	0.0083	0.0977	0.0069	0.1196	0.0177	0.1261	0.0255
Boston	0.1106	0.0059	0.1167	0.0112	0.1455	0.0076	0.1320	0.0126
California	0.1547	0.0049	0.1683	0.0049	0.1660	0.0055	0.1731	0.0081
Census	0.0873	0.0018	0.0923	0.0023	0.0860	0.0023	0.0922	0.0029
Delta Ailerons	0.0480	0.0058	0.0525	0.0078	0.0494	0.0069	0.0632	0.0116
Delta Elevators	0.0604	0.0067	0.0740	0.0126	0.0622	0.0040	0.0790	0.0123
Machine CPU	0.0489	0.0084	0.0504	0.0079	0.0589	0.0116	0.0674	0.0177

Table 6.3: Performance comparison (testing RMSE) between CI-ELM and I-ELM with 200 hidden nodes.

6.5, CI-ELM take fewer neuron numbers than I-ELM to achieve expected accuracy in almost all the cases, except for Boston case with RBF nodes. We also draw training time curves of two cases with different nodes are shown in Fig 6.2. Seen from Fig 6.2, the spent learning time is still linearly increased with the learning steps, which is consistent with I-ELM's conclusion.

6.3. Performance Evaluation

125

Name	ErrorTarget	Number of Neurons		Training Time (s)	
		CI-ELM	I-ELM	CI-ELM	I-ELM
Abalone	0.09	60	398	0.0773	0.618
Auto Price	0.075	136	359	0.0229	0.0405
Boston	0.11	157	373	0.0265	0.0813
California	0.155	189	401	0.6529	1.232
Census	0.088	188	396	0.8747	1.6457
Delta Ailerons	0.048	85	342	0.0876	0.4342
Delta Elevators	0.065	32	410	0.0464	0.5233
Machine CPU	0.04	195	348	0.0327	0.0409

Table 6.4: Number of neurons comparison between CI-ELM and I-ELM with additive sigmoidal nodes

Name	ErrorTarget	Number of Neurons		Training Time (s)	
		CI-ELM	I-ELM	CI-ELM	I-ELM
Abalone	0.09	88	338	0.1911	0.825
Auto Price	0.098	204	380	0.0518	0.0702
Boston	0.15	225	71	0.0684	0.0327
California	0.16	401	457	2.5534	3.5214
Census	0.088	119	310	1.0169	3.0136
Delta Ailerons	0.058	29	290	0.0568	0.6044
Delta Elevators	0.07	46	398	0.1329	1.2417
Machine CPU	0.04	148	189	0.025	0.0283

Table 6.5: Number of neurons comparison between CI-ELM and I-ELM with Gaussian nodes

Figure 6.2: Performance of CI-ELM with Sigmoidal activation function for Census case and Gaussian activation function for Abalone case

6.3.2 Function Approximation SinC Case

The performance of these two algorithms are also evaluated in the approximation of the ‘SinC’ function:

$$y(x) = \begin{cases} \sin(x)/x, & x \neq 0 \\ 1, & x = 0 \end{cases} \quad (6.17)$$

A training set (x_i, y_i) and testing set (x_i, y_i) with 10000 and 5000 data respectively are created where x_i ’s are uniformly randomly distributed on

the interval $(-10, 10)$. In order to make the regression problem ‘real’, large uniform noise distributed in $[-0.2, 0.2]$ has been added to all the training samples while testing data remain noise-free. We choose additive neurons, and the activation functions used in the simulations is the sigmoidal function $g(x) = 1/(1 + \exp(-x))$ and the sin function $g(x) = \sin(x)$, where the input weights are randomly chosen from the range $[-1, 1]$ and hidden biases are randomly chosen from the range $[-10, 10]$. Figure 6.3 shows the SinC performance (testing RMSE) of sigmoidal and sin functions for CI-ELM algorithm.

6.4 Discussion

Convex I-ELM (CI-ELM) is similar to Barron’s work. Barron also uses the convex iteration to construct neural networks. When a new added neuron is optimized, he pointed out that neural networks constructed by the convex iteration can approximate smooth target functions within an integrated squared error of order $O(1/n)$. However, we should note that CI-ELM is different from Barron’s work, where all the parameters of neural networks must be optimize and not random. Similarly, due to unpredictable property of random characteristic, the convergent rate of CI-ELM is unknown for us, but we can describe the complexity of learning time precisely. For conventional algorithms, they normally use gradient methods to calculate those optimal parameters of input-to-hidden neurons. Due to the uncertainty of activation

(a)

(b)

Figure 6.3: Performance of the proposed SLFN learning algorithm with sig and sin nodes on the SinC problem: (a) Sigmoidal node, (b) Sin node.

functions, it leads to the difficulty to estimate their learning time. But since CI-ELM avoids the computation of derivative, we can describe its learning time accurately. The training time of CI-ELM is linearly increased (c.f Figure 6.2). It means that though we recalculate all the existing neurons, the learning time spent for each new neuron is still fixed. So the computational complexity of training time is linear.

6.5 Conclusion

Motivated by the new universal approximation theory on neural networks proposed in Chapter 3 and the convex optimization method for neural networks [19], [151], this chapter further proposes a convex incremental Extreme Learning Machine (CI-ELM). CI-ELM randomly generates and adds computational nodes to the hidden layer and only analytically calculates the output weights of the hidden nodes. Different from I-ELM, based on a convex optimization method CI-ELM recalculates the output weights of the existing hidden nodes after a new hidden node is added. CI-ELM can obtain a faster convergence rate and more compact network architecture while retaining the I-ELM's simplicity and efficiency. It seems that the proof of universal approximation capability of I-ELM could not be extended to this convex optimization method directly. Both the bound of convergence rate and the universal approximation capability of CI-ELM are worth investigating in future.

Chapter 7

Conclusions and Recommendations

7.1 Conclusions

Neural networks have been successfully applied in many applications, such as pattern recognition, pattern classification or function approximation, etc. So far, a variety of neural network architectures with different learning algorithms have been developed. However, the complexity of neural networks and the complication of activation functions may bring computational burdens. Especially for real applications with large scale data, they normally lead to extensive computation, which makes it impossible for neural networks to respond to external signals in time. Meantime, many practical applications

illustrate that fast response to external inputs is often demanded. Therefore, whether neural networks can own real-time learning capability is a challenging issue in neural network research. In this thesis, we focus on investigating fast learning algorithms for feedforward neural networks, and from mathematical viewpoint rigorously prove that neural networks with random hidden nodes can work as universal approximators. My thesis is summarized as follows:

- A new theory about the approximation capability of SLFNs with random hidden nodes is proved in Chapter 3. The corresponding incremental algorithm, I-ELM, has been proposed in Chapter 4. I-ELM randomly adds nodes 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 then the output weights are analytically determined. The simulation results have shown that the proposed I-ELM not only learns faster than some conventional learning algorithms but also achieves better generalization performance in most cases. In addition, I-ELM is suitable for most activation functions, which for additive nodes can be any bounded nonconstant piecewise continuous functions $g : R \rightarrow R$ and the activation functions for RBF nodes can be any integrable piecewise continuous functions $g : R \rightarrow R$ and $\int_R g(x)dx \neq 0$.
- An ensemble of I-ELM is also proposed to decrease the number of hidden nodes and computation. The new architecture allows a single

I-ELM to be trained on each region of a dataset. While retaining the universal approximation feature, it also reduces the number of hidden nodes.

- An enhanced learning algorithm (EI-ELM) for I-ELM is proposed in chapter 5, which randomly chooses random hidden node parameters several times, and then we choose the parameters leading to the smallest residual error. Therefore, it can remove these useless hidden nodes so as to maximum optimize network structure and make neural networks more effective, so it may achieve a more parsimonious network architecture than I-ELM.
- An improved learning algorithm (CI-ELM) for I-ELM is proposed in chapter 6, which consists of a convex iterative sequence. Unlike I-ELM, in order to achieve smaller residual error, CI-ELM recalculates the output weights of the existing nodes based on a convex optimization method when a new hidden node is randomly added. While retaining the same simplicity and efficiency, it can achieve faster convergence rate than I-ELM. Moreover, the approximation capability of CI-ELM is still ensured. Experimental results on several benchmark real world regression problems will verify our conclusion.

7.2 Recommendations for Further Research

A new learning theory and the several corresponding incremental learning algorithms have been proposed in this thesis. During our research, some more research topics are found interesting and they are worth investigating further:

- Due to the conflict between the boundedness and the differentiability of a nonlinear complex function in the complex domain, one of the challenges in designing a neural network on complex domain is to find a proper nonlinear complex activation function for specific applications[154]. Recently, Li, et al[4] has successfully applied Extreme Learning Machine (ELM) in some complex domain applications and many fully complex activation functions can be used in ELM[4]. Kim and Adali[154] proved the approximation capability of feedforward neural networks in the complex domain for the case where the parameters of hidden nodes need to be tuned during the training phase. However, it is still an open problem whether ELM and I-ELM with fully complex activation functions can work as universal approximators in complex domain. We believe that the proposed learning theory in this thesis could be extended to the complex domain which is worth investigating systematically.
- Fuzzy neural network is also a kind of important neural network model, which combines neural networks and fuzzy logic by adding fuzzy mem-

berships into neural networks. Due to the complexity of parameters and network structure, the training procedure of fuzzy neural networks is usually slower than required. In order to overcome this disadvantage, the fuzzy model is worth investigating, and I wish that I can find a breaking through in future.

- The main focus of this thesis is on function regressions. I-ELM and its variants for classification applications will be investigated in future.
- Convex I-ELM (CI-ELM) may be interesting to prove in theory whether convex I-ELM converges faster than original I-ELM in most applications. However, to our knowledge, it may not have a simply way to prove it. I wish that I can solve it in the future.

Author's Publications

1. Lei Chen, Guang-Bin Huang, and Chee-Kheong Siew, "Simplification of a Specific Two-Hidden-Layer Feedforward Networks", in ICICS-PCM 2003, Singapore.
2. Guang-Bin Huang, Lei Chen and Chee-Kheong Siew, "Universal Approximation Using Incremental Constructive Feedforward networks with Random Hidden Nodes", IEEE Transactions on Neural Networks, vol. 17, no.4, p. 879- 892, 2006.
3. Guang-Bin Huang and Lei Chen, "Convex Incremental Extreme Learning Machine", accepted by Neurocomputing, 2006
4. Lei Chen and Guang-Bin Huang, "Enhanced Incremental Extreme Learning Machine", submitted to IEEE Transactions on Neural Networks, 2006
5. Guang-Bin Huang, Ming-Bin Li and Lei Chen, "Complex Incremental Extreme Learning Machine", accepted by Neurocomputing, 2007

Bibliography

- [1] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, “Extreme learning machine: A new learning scheme of feedforward neural networks,” in *Proceedings of International Joint Conference on Neural Networks(IJCNN2004)*, (Budapest, Hungary), pp. 985–990, 25-29 July, 2004.
- [2] G.-B. Huang and C.-K. Siew, “Extreme learning machine: RBF network case,” in *Proceedings of the Eighth International Conference on Control, Automation, Robotics and Vision (ICARCV 2004)*, (Kunming, China), pp. 1029–1036, 6-9 Dec, 2004.
- [3] 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.
- [4] M.-B. Li, G.-B. Huang, P. Saratchandran, and N. Sundararajan, “Fully complex extreme learning machine,” *Neurocomputing*, vol. 68, pp. 306–314, 2005.

-
- [5] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate on-line sequential learning algorithm for feedforward networks," *accepted (with minor revision) by IEEE Transactions on Neural Networks*, 2006.
- [6] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Real-time learning capability of neural networks," *IEEE Transactions On Neural Networks*, vol. 17, no. 4, pp. 863–878, 2006.
- [7] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theorey and applications," *Neurocomputing*, vol. 70, pp. 489–501, 2006.
- [8] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.
- [9] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, pp. 303–314, 1989.
- [10] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1481–1497, 1990.
- [11] Y. Ito, "Approximation of functions on a compact set by finite sums of a sigmoid function without scaling," *Neural Networks*, vol. 4, pp. 817–826, 1991.

-
- [12] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, pp. 251–257, 1991.
- [13] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Computation*, vol. 3, pp. 246–257, 1991.
- [14] Y. Ito, "Approximation of continuous functions on $\mathbf{R}^d$ by linear combinations of shifted rotations of a sigmoid function with and without scaling," *Neural Networks*, vol. 5, pp. 105–115, 1992.
- [15] L.-X. Wang, "Fuzzy systems as universal approximators," in *IEEE International Conference on Fuzzy Systems*, pp. 1153–1162, 1992.
- [16] L. K. Jones, "A simple lemma on greedy approximation in hilbert space and convergence rates for projection pursuit regression and neural networks," *The Annals of Statistics*, vol. 20, no. 1, pp. 608–613, 1992.
- [17] K. Hornik, "Some new results on neural network approximation," *Neural Networks*, vol. 6, pp. 1069–1072, 1993.
- [18] 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.
- [19] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoid function," *IEEE Transactions on Information Theory*, vol. 39, no. 3, pp. 930–945, 1993.

-
- [20] P. Koiran, “Efficient learning of continuous neural networks,” in *Proceedings of the Seventh Annual ACM Conference on Computational Learning Theory*, (New Brunswick, New Jersey), pp. 348–355, 12-15 July 1994.
 - [21] T. Chen, H. Chen, and R.-W. Liu, “Approximation capability in $C(\overline{\mathbf{R}}^n)$ by multilayer feedforward networks and related problems,” *IEEE Transactions on Neural Networks*, vol. 6, no. 1, pp. 25–30, 1995.
 - [22] B. Igelnik and Y.-H. Pao, “Stochastic choice of basis functions in adaptive function approximation and the functional-link net,” *IEEE Transactions on Neural Networks*, vol. 6, no. 6, pp. 1320–1329, 1995.
 - [23] T. Chen and H. Chen, “Approximation capability to functions of several variables, nonlinear functionals, and operators by radial basis function neural networks,” *IEEE Transactions on Neural Networks*, vol. 6, no. 4, pp. 904–910, 1995.
 - [24] F. Girosi, M. Jones, and T. Poggio, “Regularization theory and neural networks architectures,” *Neural Computation*, vol. 7, pp. 219–269, 1995.
 - [25] S. Lawrence, A. C. Tsoi, and A. D. Back, “Function approximation with neural networks and local methods: Bias, variance and smoothness,” in *Australian Conference on Neural Networks (ACNN96)*

- (P. Bartlett, A. Burkitt, and R. Williamson, eds.), pp. 16–21, Australian National University, 1996.
- [26] W. S. Lee, P. L. Bartlett, and R. C. Williamson, “Efficient agnostic learning of neural networks with bounded fan-in,” *IEEE Transactions on Information Theory*, vol. 42, no. 6, pp. 2118–2132, 1996.
- [27] M. J. Donahue, L. Gurvits, C. Dalken, and E. Sontag, “Rates of convex approximation in non-hilbert spaces,” *Constructive Approximation*, vol. 13, pp. 187–220, 1997.
- [28] T.-Y. Kwok and D.-Y. Yeung, “Objective functions for training new hidden units in constructive neural networks,” *IEEE Transactions on Neural Networks*, vol. 8, no. 5, pp. 1131–1148, 1997.
- [29] G.-B. Huang and H. A. Babri, “General approximation theorem on feedforward networks,” in *Proceedings of the International Conference on Information, Communications and Signal Processing (ICICS’97)*, (Singapore), pp. 698–702, 9-12 September, 1997.
- [30] S. Tamura and M. Tateishi, “Capabilities of a four-layer feedforward neural networks: Four layers versus three,” *IEEE Transactions on Neural Networks*, vol. 8, no. 2, pp. 251–255, 1997.
- [31] A. T. Dingankar and I. W. Sandberg, “A note on error bounds for function approximation using nonlinear networks,” in *Proc. of 40th*

- Midwest Symposium on Circuits and Systems*, (Sacramento, CA), 3-6 August 1997.
- [32] G.-B. Huang and H. A. Babri, "Comments on 'approximation capability in $C(\overline{\mathbf{R}}^n)$ by multilayer feedforward networks and related problems'," *IEEE Transactions on Neural Networks*, vol. 9, no. 4, pp. 714–715, 1998.
- [33] H. N. Mhaskar, "Approximation of smooth functions by neural networks," in *Perspectives in Neural Computing*, vol. 1112, (Springer Verlag, London), pp. 189–204, 1998.
- [34] V. Maiorov and A. Pinkus, "Lower bounds for approximation by mlp neural networks," *Neurocomputing*, vol. 25, pp. 81–91, 1999.
- [35] P. Morgan, B. Curry, and M. Beynon, "Comparing neural network approximations for different functional forms," *Expert Systems*, vol. 16, no. 2, pp. 60–71, 1999.
- [36] P. Niyogi and F. Girosi, "Generalization bounds for function approximation from scattered noisy data," in *Advances in Computational Mathematics*, vol. 10, pp. 51–80, 1999.
- [37] A. Pinkus, "Approximation theory of the mlp model in neural networks," *Acta Numerica*, pp. 143–195, 1999.

-
- [38] R. Meir and V. E. Maiorov, “On the optimality of neural-network approximation using incremental algorithms,” *IEEE Transactions on Neural Networks*, vol. 11, no. 2, pp. 323–337, 2000.
- [39] E. Romero, “Function approximation with SAOCIF: A general sequential method and a particular algorithm with feed-forward neural networks,” in <http://www.lsi.upc.es/dept/techreps/html/R01-41.html>, Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, 2001.
- [40] M. Burger and A. Neubauer, “Error bounds for approximation with neural networks,” *Journal of Approximation Theory*, vol. 112, pp. 235–250, 2001.
- [41] E. Romero, “A new incremental method for function approximation using feed-forward neural networks,” in *Proc. INNS-IEEE International Joint Conference on Neural Networks (IJCNN’2002)*, pp. 1968–1973, 2002.
- [42] E. Lavretsky, “On the geometric convergence of neural approximations,” *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 274–282, 2002.
- [43] V. Kurkova and M. Sanguinetti, “Comparison of worst case errors in linear and neural network approximation,” *IEEE Transactions On Information Theory*, vol. 48, no. 1, pp. 264–275, 2002.

-
- [44] T. Kim and T. Adali, “Universal approximation of fully complex feed-forward neural networks,” in *International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, pp. I–973–I–976, 2002.
 - [45] S.-T. Li and S.-C. Chen, “Function approximation using robust wavelet neural networks,” in *14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI’02)*, 2002.
 - [46] Q. Tao, J. Wang, G. Wu, and J. Wang, “The theoretical analysis of kernel technique and its applications,” in *Proceedings of the 2002 International Joint Conference on Neural Networks (IJCNN’02)*, vol. 1, pp. 571–576, 2002.
 - [47] H. Hedenmalm and E. Saksman, “Carleson’s convergence theorem for dirichlet series,” *Pacific Journal of Mathematics*, vol. 208, no. 1, pp. 85–109, 2003.
 - [48] G. Gripenberg, “Approximation by neural networks with a bounded number of nodes at each level,” *Journal of Approximation Theory*, vol. 122, pp. 260–266, 2003.
 - [49] C. Xiang, Shenqiang, and T. H. Lee, “Geometrical interpretation and architecture selection of mlp,” *IEEE Transactions on Neural Networks*, vol. 16, no. 1, pp. 84–96, 2005.
 - [50] H. L. Royden, *Real Analysis (Third edition)*. Macmillan Publishing Company, New York, 1988.

-
- [51] B. Widrow and M. A. LEHR, “30 years of adaptive neural networks : perceptron, madaline, and backpropagation,” *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1415–1442, 1990.
- [52] L.-X. Wang and J. M. Mendal, “Fuzzy basis functions, universal approximation and orthogonal least-squares learning,” *IEEE Transactions on Neural Networks*, vol. 3, no. 5, pp. 807–814, 1992.
- [53] H. M. Kim and J. M. Mendel, “Fuzzy basis functions: Comparisons with other basis functions,” *IEEE Transactions on Neural Networks*, vol. 3, no. 2, pp. 158–168, 1995.
- [54] M. Bianchini, P. Frasconi, and M. Gori, “Learning without local minima in radial basis function networks,” *IEEE Transactions on Neural Networks*, vol. 6, no. 3, pp. 749–756, 1995.
- [55] K. J. Hunt, R. Haas, and R. Murray-Smith, “Extending the functional equivalent of radial basis function networks and fuzzy inference systems,” *IEEE Transactions on Neural Networks*, vol. 7, no. 3, pp. 776–781, 1996.
- [56] T. Blu and M. Unser, “Wavelets, fractals and radial basis functions,” *IEEE Transactions on Signal Processing*, vol. 50, no. 3, pp. 543–553, 2002.

-
- [57] J. S. R. Jang and C. T. Sun, "Functional equivalence between radial basis function networks and fuzzy inference systems," *IEEE Transactions on Neural Networks*, vol. 4, no. 1, pp. 156–159, 1993.
- [58] H. Peng, T. Ozaki, V. Haggan-Ozaki, and Y. Toyoda, "A parameter optimization method for radial basis function type models," *IEEE Transactions on Neural Networks*, vol. 14, no. 2, pp. 432–438, 2003.
- [59] 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.
- [60] W. Kaminski and P. Strumillo, "Kernel orthonormalization in radial basis function neural networks," *IEEE Transactions On Neural Networks*, vol. 8, no. 5, pp. 1177–1183, 1997.
- [61] J. B. Gomm and D. L. Yu, "Selecting radial basis function network centers with recursive orthogonal least squares training," *IEEE Transactions On Neural Networks*, vol. 11, no. 2, pp. 306–314, 2005.
- [62] C. Drioli and D. Rocchesso, "Orthogonal least squares algorithm for the approximation of a map and its derivatives with a rbf network," *Signal Processing*, vol. 83, pp. 283–296, 2003.
- [63] S. Chen, X. Hong, and C. J. Harris, "Orthogonal forward selection for constructing the radial basis function network with tunable nodes," in

- In Proceedings of IEEE International Conference on Intelligent Computing Part I*, (Hefei, China), pp. 777–786, 2005.
- [64] J. Platt, “A resource-allocating network for function interpolation,” *Neural Computation*, vol. 3, pp. 213–225, 1991.
- [65] L. Yingwei, P. Saratchandran, and N. Sundararajan, “A sequential learning scheme for function approximation using minimal radial basis function (rbf) neural networks,” *Neural Computation*, vol. 9, pp. 461–478, 1997.
- [66] L. Yingwei, P. Saratchandran, and N. Sundararajan, “Performance evaluation of a sequential minimal radial basis function (rbf) neural networks learning algorithm,” *IEEE Transactions On Neural Networks*, vol. 9, no. 2, pp. 308–318, 1998.
- [67] V. Kadiramanathan and M. Niranjana, “A function estimation approach to sequential learning with neural networks,” *Neural Computation*, vol. 5, pp. 954–975, 1993.
- [68] G.-B. Huang, P. Saratchandran, and N. Sundararajan, “An efficient sequential learning algorithm for growing and pruning rbf (GAP-RBF) networks,” *IEEE Transactions On Systems, Man, and Cybernetics-Part B: Cybernetics*, vol. 34, no. 6, pp. 2284–2291, 2004.
- [69] G.-B. Huang, P. Saratchandran, and N. Sundararajan, “An generalized growing and pruning rbf (GGAP-RBF) neural networks for function ap-

- proximation,” *IEEE Transactions On Neural Networks*, vol. 16, no. 1, pp. 57–67, 2005.
- [70] G.-B. Huang and H. A. Babri, “Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions,” *IEEE Transactions on Neural Networks*, vol. 9, no. 1, pp. 224–229, 1998.
- [71] G. Cybenko, “Approximation by superpositions of a sigmoidal function,” *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989.
- [72] K. Funahashi, “On the approximate realization of continuous mappings by neural networks,” *Neural Networks*, vol. 2, pp. 183–192, 1989.
- [73] M. Stinchcombe and H. White, “Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions,” in *Artificial Neural Networks: Approximation and Learning Theory* (H. White, ed.), pp. 29–40, Blackwell, 1992.
- [74] A. Gallant and H. White, “There exists a neural network that does not make avoidable mistakes,” in *Artificial Neural Networks: Approximation and Learning Theory* (H. White, ed.), pp. 5–11, Blackwell, 1992.
- [75] F. Girosi and T. Poggio, “Networks and the best approximation property,” in *A.I. Memo No. 1164, Artificial Intelligence Laboratory, Massachusetts Institute of Technology*, 1989.

-
- [76] T. Poggio and F. Girosi, “A theory of networks for approximation and learning,” in *A.I. Memo No. 1140, Artificial Intelligence Laboratory, Massachusetts Institute of Technology*, 1989.
 - [77] V. Kůrková, “Kolmogorov’s theorem and multilayer neural networks,” *Neural Networks*, vol. 5, pp. 501–506, 1992.
 - [78] V. Y. Kreinovich, “Arbitrary nonlinearity is sufficient to represent all functions by neural networks: A theorem,” *Neural Networks*, vol. 4, pp. 381–383, 1991.
 - [79] C.-H. Choi and J. Y. Choi, “Constructive neural networks with piecewise interpolation capabilities for function approximations,” *IEEE Transactions On Neural Networks*, vol. 5, no. 6, pp. 936–944, 1994.
 - [80] J. Wray and G. G. Green, “Neural networks, approximation theory and finite precision computation,” *Neural Networks*, vol. 8, no. 1, pp. 31–37, 1995.
 - [81] R. Hecht-Nielsen, “Kolmogorov’s mapping neural network existence theorem,” in *Proceedings of the International Conference on Neural Networks*, pp. 11–14, 1987.
 - [82] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning internal representations by error propagation,” in *Parallel Distributed Processing: Explorations in the Microstructures of Cognition, Vol: Founda-*

- tions* (D. E. Rumelhart and J. L. McClelland, eds.), pp. 318–362, MIT Press, Cambridge, MA, 1986.
- [83] L. W. Chan and F. Fallside, “An adaptive training algorithm for backpropagation networks,” *Computer Speech and Language*, vol. 2, pp. 205–218, 1987.
- [84] J. Schmidhuber, “Accelerated learning in back-propagation nets,” in *Connectionism in Perspective* (R. Pfeifer, Z. Schreter, and L. Steels, eds.), (Amsterdam, Holland), pp. 439–445, Elsevier Science Publishers B.V., 1989.
- [85] R. Salomon, “Adaptive learning rate for back-propagation,” in *Technical Report*, University Berlin, 1989.
- [86] F. M. Silva and L. B. Almeida, “Speeding up backpropagation,” in *Advanced Neural Computers* (R. Eckmiller, ed.), (Amsterdam, Holland), pp. 151–158, 1990.
- [87] T. Tollenaere, “Supersab: Fast adaptive backpropagation with good scaling properties,” *Neural Networks*, vol. 3, pp. 561–573, 1990.
- [88] B. Pearlmutter, “Gradient descent: Second order momentum and saturating error,” in *Advances in Neural Information Processing Systems*, vol. 4, pp. 887–894, Morgan Kaufmann Publishers, Inc., 1992.
- [89] R. Martin and H. Braun, “A direct adaptive method for faster back-propagation learning: The RPROP algorithm,” in *Proceedings of the*

- IEEE International Conference on Neural Networks*, pp. 586–591, San Francisco, CA, 1993.
- [90] M. T. Hagan, H. B. Demuth, and M. H. Beale, *Neural Network Design*. Boston, MA: PWS Publishing, 1996.
- [91] S.-H. Oh, “Improving the error backpropagation algorithm with a modified error function,” *IEEE Transactions on Neural Networks*, vol. 8, no. 3, pp. 799–803, 1997.
- [92] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, “Efficient Back-Prop,” *Lecture Notes in Computer Science*, vol. 1524, pp. 9–50, 1998.
- [93] C. Cortes and V. Vapnik, “Support-vector networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [94] B. Scholkopf, C. Burges, and V. Vapnik, “Extracting support data for a given task,” in *Proc. First Int’l Conf. on Knowledge Discovery and Data Mining (KDD’95)*, pp. 252–257, 1995.
- [95] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A training algorithm for optimal margin classifiers,” in *Computational Learning Theory*, pp. 144–152, 1992.
- [96] V. Vapnik, S. Golowich, and A. Smola, “Support vector method for function approximation, regression estimation, and signal processing,” in *Advances in Neural Information Processing Systems*, pp. 281–287, The MIT Press, 1996.

-
- [97] B. Scholkopf, A. J. Smola, and K.-R. Muller, "Nonlinear component analysis as a kernel eigenvalue problem," in *http://www.mpi-kueeb.mpg.ed/bu.html*, 1996.
- [98] C. J. C. Burges, "Simplified support vector decision rules," in *International Conference on Machine Learning*, pp. 71–77, 1996.
- [99] B. Scholkopf, C. Burges, and V. Vapnik, "Incorporating invariances in support vector learning machines," in *In Artificial Neural Networks (ICANN'96)*, vol. 1112, (Berlin), pp. 47–52, 1996.
- [100] H. Drucker, C. J. C. Burges, L. Kaufman, A. Smola, and V. Vapnik, "Support vector regression machines," in *Advances in Neural Information Processing Systems* (M. C. Mozer, M. I. Jordan, and T. Petsche, eds.), vol. 9, pp. 155–, The MIT Press, 1997.
- [101] K.-R. Muller, A. J. Smola, G. Ratsch, B. Scholkopf, J. Kohlmorgen, and V. Vapnik, "Predicting time series with support vector machines," in *ICANN*, pp. 999–1004, 1997.
- [102] B. Scholkopf, K.-K. Sung, C. J. C. Burges, F. Girosi, P. Niyogi, T. Poggio, and V. Vapnik, "Comparing support vector machines with gaussian kernel to radial basis function classifiers," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2758–2765, 1997.
- [103] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods: Support Vector*

- Machines* (B. Schölkopf, C. Burges, and A. Smola, eds.), MIT Press, Cambridge, MA, 1998.
- [104] A. Smola, B. Scholkopf, and K.-R. Muller, “Convex cost functions for support vector regression,” in *Proceedings of the 8th ICANN, Perspectives in Neural Computing*, (Berlin), pp. 99–104, 1998.
- [105] A. J. Smola, B. Scholkopf, and K.-R. Muller, “The connection between regularization operators and support vector kernels,” *Neural Networks*, vol. 11, pp. 637–649, 1998.
- [106] A. Smola and B. Schölkopf, “A tutorial on support vector regression,” *NeuroCOLT2 Technical Report NC2-TR-1998-030*, 1998.
- [107] A. J. Smola, N. Murata, B. Scholkopf, and K.-R. Muller, “Asymptotically optimal choice of ϵ -loss for support vector machines,” pp. 105–110, *Proceedings of the 8th International Conference on Artificial Neural Networks, Perspectives in Neural Computing*, 1998.
- [108] J. C. Platt, “Sequential minimal optimization for svm,” in <http://www.research.microsoft.com/jplatt/smo.html>.
- [109] B. Scholkopf, S. Mike, C. J. C. Burges, P. Knirsch, K.-R. Muller, G. Ratsch, and A. J. Smola, “Input space versus feature space in kernel-based methods,” *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 1000–1015, 1999.

-
- [110] J. A. K. Suykens and J. Vandewalle, “Least squares support vector machine classifiers,” *Neural Processing Letters*, vol. 9, pp. 293–300, 1999.
- [111] B. Scholkopf, A. J. Smola, R. C. Williamson, and P. L. Bartlett, “New support vector algorithm,” *Neural Computation*, vol. 12, pp. 1207–1245, 2000.
- [112] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy, “A fast iterative nearest point algorithm for support vector machine classifier design,” *IEEE Transactions on Neural Networks*, vol. 11, no. 1, pp. 124–136, 2000.
- [113] M. E. Tipping, “Sparse bayesian learning and the relevance vector machine,” *Journal of Machine Learning Research*, vol. 1, pp. 211–244, 2001.
- [114] K.-R. Muller, S. Mike, G. Ratsch, K. Tsuda, and B. Scholkopf, “An introduction to kernel-based learning algorithms,” *IEEE Transactions on Neural Networks*, vol. 12, no. 2, pp. 181–201, 2001.
- [115] B. Scholkopf, J. C. Platt, J. Schawe-Taylor, A. J. Smola, and R. C. Williamson, “Estimating the support of a high-dimension distribution,” *Neural Computation*, vol. 13, pp. 1443–1471, 2001.

-
- [116] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy, "Improvement to Platt's SMO algorithm for SVM classifier design," *Neural Computation*, vol. 13, pp. 637–649, 2001.
- [117] V. Tresp, "Scaling kernel-based systems to large data sets," *Data Mining and Knowledge Discovery*, vol. 5, no. 3, pp. 1–18, 2001.
- [118] I. Steinwart, "On the influence of the kernel on the consistency of support vector machines," *Journal of Machine Learning Research*, vol. 2, pp. 67–93, 2001.
- [119] C.-J. Lin, "On the convergence of the decomposition method for support vector machines," *IEEE Transactions on Neural Networks*, vol. 12, no. 6, pp. 1288–1298, 2001.
- [120] C.-C. Chang and C.-J. Lin, "Training v-support vector classifiers: Theory and algorithms," *Neural Computation*, vol. 13, pp. 2119–2147, 2001.
- [121] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.
- [122] C.-F. Lin and S.-D. Wang, "Fuzzy support vector machines," *IEEE Transactions on Neural Networks*, vol. 13, no. 2, pp. 464–471, 2002.
- [123] C.-J. Lin, "Asymptotic convergence of an smo algorithm without any assumptions," *IEEE Transactions on Neural Networks*, vol. 13, no. 1, pp. 248–250, 2002.

-
- [124] K.-M. Lin and C.-J. Lin, “A study on reduced support vector machines,” *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1449–1456, 2003.
- [125] G. Daqi and Y. Genxing, “Influences of variable scales and activation functions on the performances of multilayer feedforward neural networks,” *The Journal of Pattern Recognition Society*, vol. 36, pp. 869–878, 2003.
- [126] R. Collobert, Y. Bengio, and S. Bengio, “Scaling large learning problems with hard parallel mixtures,” *International Journal of Pattern Recognition*, vol. 17, no. 3, pp. 349–365, 2003.
- [127] C.-C. Chang and C.-J. Lin, “LIBSVM – a library for support vector machines,” in <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>, Department of Computer Science and Information Engineering, National Taiwan University, Taiwan, 2003.
- [128] F. Perez-Cruz, A. Navia-Vazquez, A. R. Figueiras-Vidal, and A. Artes-Rodriguez, “Empirical risk minimization for support vector classifiers,” *IEEE Transactions on Neural Networks*, vol. 14, no. 2, pp. 296–303, 2003.
- [129] J. T. Kowk and I. W. Tsang, “Linear dependency between ϵ and the input noise in ϵ -support vector regression,” *IEEE Transactions on Neural Networks*, vol. 14, no. 3, pp. 544–553, 2003.

-
- [130] L. Zhang and W. Zhou, and L. Jiao, “Wavelet support vector machine,” *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, vol. 34, no. 1, pp. 34–49, 2003.
- [131] Y. Engel, S. Mannor, and R. Meir, “The kernel recursive least-squares algorithm,” *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2275–2285, 2004.
- [132] J. Kivinen, A. J. Smola, and R. C. Williamson, “Online learning with kernels,” *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2165–2176, 2004.
- [133] M. Sanchez-Fernandez, M. de Prado-Cumplido, J. Arenas-Garcia, and F. Perez-Cruz, “Svm multiregression for nonlinear channel estimation in multiple-input multiple output systems,” *IEEE Transactions on Signal Processing*, vol. 52, no. 8, pp. 2298–2307, 2004.
- [134] L. Zhang, W. Zhou, and L. Jiao, “Hidden space support vector machines,” *IEEE Transactions on Neural Networks*, vol. 15, no. 6, pp. 1424–1434, 2004.
- [135] G.-B. Huang, K. Z. Mao, C.-K. Siew, and D.-S. Huang, “Fast modular network implementation for support vector machines,” *IEEE Transactions on Neural Networks*, vol. 16, no. 6, pp. 1651–1663, 2005.
- [136] S. Haykin, *Neural Networks: A Comprehensive Foundation, Second Edition*. Prentice Hall, 1999.

-
- [137] 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.
 - [138] D. Serre, *Matrices: Theory and Applications*. Springer-Verlag New York, Inc, 2002.
 - [139] C. R. Rao and S. K. Mitra, *Generalized Inverse of Matrices and Its applications*. New York: John Wiley & Sons, Inc, 1971.
 - [140] D. Lowe., "Adaptive radial basis function nonlinearities and the problem of generalisation," in *First IEE International Conference on Artificial Neural Networks*, (London), pp. 171–175, 1989.
 - [141] G.-B. Huang and C.-K. Siew, "Extreme learning machine with randomly assigned RBF kernels," *International Journal of Information Technology*, vol. 11, no. 1, 2005.
 - [142] E. Baum, "On the capabilities of multilayer perceptrons," *J. Complexity*, vol. 4, pp. 193–215, 1988.
 - [143] W. L. Voxman and J. Roy H. Goetschel, *Advanced Calculus: An Introduction to Modern Analysis*. Marcel Dekker, Inc., 1981.
 - [144] A. N. Kolmogorov and S. V. Fomin, *Elements of the Theory of Functions and Functional Analysis, Volume 2: Measure, The Lebesgue Integral, Hilbert Space*. Graylock Press, Baltimore, MD, 1961.

- [145] V. N. Vapnik, *Statistical learning theory*. New York: Wiley, 1998.
- [146] C. Blake and C. Merz, “UCI repository of machine learning databases,” in <http://www.ics.uci.edu/~mlearn/MLRepository.html>, Department of Information and Computer Sciences, University of California, Irvine, USA, 1998.
- [147] J. Platt, “Sequential Minimal Optimization: A fast algorithm for training support vector machines,” *Microsoft Research Technical Report MSR-TR-98-14*, 1998.
- [148] T. Joachims, “SVM^{light} – support vector machine,” in <http://svmlight.joachims.org/>, Department of Computer Science, Cornell University, 2003.
- [149] MathWorks, “Compiler execution speed comparison,” in <http://www.mathworks.com/products/compiler/examples/example2.shtml>, The MathWorks, Inc, 2003.
- [150] 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.
- [151] Y. Bengio, N. L. Roux, P. Vincent, O. Delalleau, and P. Marcotte, “Convex neural networks,” in *Advances in Neural Information Process-*

- ing Systems 18* (Y. Weiss, B. Schölkopf, and J. Platt, eds.), pp. 123–130, Cambridge, MA: MIT Press, 2006.
- [152] M. I. Jordan and R. A. Jacobs, “Hierarchical mixtures of experts and the EM algorithm,” *Neural Computation*, vol. 6, pp. 181–214, 1994.
- [153] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, “Adaptive mixtures of local experts,” *Neural Computation*, vol. 3, pp. 79–87, 1991.
- [154] T. Kim and T. Adali, “Approximation by fully complex multilayer perceptrons,” *Neural Computation*, vol. 15, pp. 1641–1666, 2003.