Extreme Learning Machines

Zhu Qinyu

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University

in fulfillment of the requirement for the degree of

Doctor of Pilosophy

2006

Acknowledgments

I would like to express my gratitude to my supervisor Dr. Huang Guangbin. He is kind and knowledgeable. I was deeply impressed by his enthusiasm and perseverance on research. His stimulating suggestions and encouragement were tremendously helpful to my Ph.D 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, Associate Professor Dr. Ponnuthurai Nagaratnam Suganthan and Mr. Qin Kai for the useful discussions and comments on my research work.

Contents

Summary

In some practical applications of neural networks, fast response to external events within an extremely short time is expected. The extensively used gradient-based learning algorithms can give satisfactory response once they have been properly trained. However, they cannot satisfy the fast learning needs in many applications where frequent retraining is needed, especially for large scale applications and/or when higher learning accuracy and generalization performance are required.

This thesis first proposes a novel learning algorithm called extreme learning machine (ELM) for **s**ingle-hidden **l**ayer **f**eedforward neural **n**etworks (SLFNs). Unlike the iterative learning strategy used in conventional learning algorithms, it randomly generates the hidden nodes (by randomly assigning node parameters), and then analytically determines the output weights of the networks at once. Numerous experiments on different kinds of artificial and real-world benchmarking datasets/problems have been conducted to evaluate the performance of ELM. The comparative results shown in this thesis indicate that the proposed ELM can achieve better generalization performance at a learning speed hundreds or thousands of times faster than conventional learning algorithms. In addition, since the ELM does not have the issues like

learning rate, stopping criterion and local minima which are commonly faced by most tuning-based algorithms, it is easy to use even for ordinary users.

The feedforward neural networks with threshold activation functions attract a lot of interest mainly due to their easy implementation in digital hardware. However, the gradient-based learning algorithms cannot be used to train the threshold networks directly as the threshold activation functions are non-differentiable. These algorithms normally use analog functions to approximate the threshold functions, while the proposed ELM can directly train the threshold networks which makes the online training in chips possible. In this thesis, the learning capability and performance of the ELM on threshold networks are thoroughly investigated and tested.

Furthermore, the ELM is extended to train two-hidden layer feedforward neural networks by using a constructive method which splits the training dataset into groups. Each group is learned by a SLFN using the ELM and all the SLFNs share the same first hidden layer. The learning time is extremely short especially for the applications with large training datasets.

Although the main objective of this thesis is to propose fast learning algorithms, this thesis also investigates the variants of ELM which can achieve compact network architectures or better generalization performance. Two variants of the ELM are proposed. For the testing time concerned applications such as embedded systems, a hybrid learning algorithm of ELM and

differential evolution (DE) called evolutionary extreme learning machine (E-ELM) is proposed to achieve more compact networks. For stability and accuracy concerned applications, an algorithm to construct bagging ensemble of ELMs called bagging extreme learning machines (B-ELM) is presented.

List of Figures

List of Tables

Chapter 1

Introduction

This chapter gives an overview of the research project and the organization of the thesis. It includes:

- The Motivation section gives an overview of conventional learning algorithms for feedforward neural networks. It also explains the circumstance under which this research project was proposed.
- The Objectives section elaborates the goals that we plan to reach.
- The Major Contribution of the Thesis section states the main contribution of this research project.
- The Organization of the Thesis section briefly summarizes the content of each chapter and the organization of the chapters.

1.1 Motivation

The widespread popularity of feedforward neural networks in many fields is mainly due to their ability:

1) to approximate complex unknown nonlinear mappings directly from the input training samples [1, 2, 3, 4, 5, 6];

2) to form disjoint decision regions with arbitrary shapes and to determine unknown classes [7].

Whether neural networks can have fast learning capability is still a challenging and yet open question. A neural network system is called a fast learning system if it can complete a learning procedure with good generalization performance for a new application within expected fast response time defined by external requirements. This expected response time could be any reasonable time which an application can endure. It could be microseconds, milliseconds or even seconds. Fast learning capability of neural networks is highly expected whenever a new application is faced, where a new knowledge map has to be built.

Gradient-descent learning methodologies such as back-propagation (BP) learning algorithm [8] and its variants have been widely used in the training of neural networks. In the gradient-descent learning algorithms, the parameters (weights and biases) are adjusted along the steepest descent directions

(negative of the gradient) of the cost functions iteratively. In order to achieve a faster convergence, some algorithms perform the search along conjugate or second-order directions, while others [9, 10] using Extended Karlman Filter (EKF) [11] to speed up the training (reduce the number of learning iterations). Although these algorithms are faster than the gradient-descent methods, the parameters of the network are still updated iteratively. Thus, the learning is inevitably time consuming, especially for those applications with large amount of observations and/or high accuracy expectation. It is not surprising that this kind of learning algorithms may spend hours or days on training and sometimes faces local minima issues resulting in unsuccessful learning. In conclusion, conventional gradient-based learning algorithms are not applicable for many time-critical applications.

Besides learning algorithm for multilayer feedforward neural networks, there is a surge of interest recently in kernel learning methods such as support vector machines (SVM) [12] using quadratic programming methods. SVM have been empirically shown to give good generalization performance on many problems. However, it is slow in both training and prediction phase and complex to implement [13]. A lot of variants of SVM have been proposed to reduce the computational complexity and thus reduce the training time. For example, least square support vector machines (LS-SVM) [14] uses least square method to simplify procedure of choosing support vectors and sequential minimal optimization (SMO) [13] uses working set selection method to

reduce the computational complexity.

For classification problems, there are some solutions using non-adaptive classifiers such as Bayesian classifier and k-nearest neighbour (k-NN) classifier. These classifiers do not even require learning procedures. However, since all the computational cost is postponed from learning to prediction, they can be slow in prediction especially on the problems with large training datasets.

Therefore, alternative algorithms having fast learning and prediction speed is still highly demanded by time-critical applications and fast changing environments such as fault analysis and prediction, tracking and navigation, meteorology, geography information procession, expert systems, intelligent networking, new virus detection and disease diagnosis, etc. One of interesting approaches is using least square (LS) methods to train the neural networks. For example, Ferrari and Stengel [15] and Huang and Babri's [16] proposed LS learning algorithms for single-hidden-layer feedforward neural networks (SLFNs). The LS learning method is the main methodology of this thesis.

1.2 Objectives

This research aims to investigate fast learning algorithms and to propose novel fast learning algorithms for feedforward networks which are suitable for

time-critical applications. The proposed algorithms should have the following characteristics:

- Given a training dataset, they are capable to learn much faster than conversional gradient-based learning algorithms. Their training time can be just one tenth or even one hundredth of that needed by the fastest gradient-based learning algorithms such as Levenberg-Marquardt algorithm [17].
- They also have reasonably fast response during the prediction/testing phase which means they cannot postpone the computational cost from the training to the prediction.
- They achieve good or comparable generalization performance which means they cannot sacrifice accuracy for time.
- They have low computational and storage cost shall be in training and prediction such that the algorithms can be implemented not only on ordinary PCs but also on embedded systems.
- They should easy to use so that they can achieve good performance without much tuning of the parameters.

1.3 Major Contribution of the Thesis

We have made the following five major contributions:

1. A novel learning algorithm for **s**ingle-hidden **l**ayer **f**eedforward neural **n**etworks (SLFNs) called extreme learning machine (ELM) was proposed. Unlike conventional tuning-based learning algorithms which update all the parameters (weights and biases) in both layers iteratively, the ELM randomly selects the values for input weights and biases (linking the input layer to the hidden layer) and analytically determines the output weights (linking the hidden layer to the output layer) by using of the Moore-Penrose (MP) generalized inverse. Thus, it can achieve good generalization performance in very short learning time. Numerous benchmarking problems of different kinds were tested to compare the performance of ELM with the conventional gradient-based learning algorithms. The results concluded from thousands of experiments verify that the ELM can learn much faster than these algorithms and achieve better generalization performance as well. In some cases, the learning speed of the ELM can be thousands of times faster than the gradient-based learning algorithms. In addition, unlike the conventional gradient-based learning method, the ELM does not have the issues like learning rate, stopping criteria and local minima which are commonly faced by gradient-based learning algorithms. Although the generalization performance of the ELM is affected by the size of the network namely the number of hidden neurons, this thesis shows that the affection is marginal once the number of the hidden neurons is larger than a proper value and the generalization performance of the

ELM is stable on a wide range of network sizes.

2. Training of neural networks with threshold activation functions is always an interesting research area as they are often easier to implement in digital hardware and their learning capabilities are well understood. However, the conventional gradient-based learning algorithms cannot be directly applied to train the threshold neural networks as the threshold functions are nondifferentiable. These algorithms try to use analog functions to approximate the threshold function so they are only suitable for offline training (not in hardware), and the parameters obtained after the completion of the training are then transferred to the threshold networks in chips. We successfully applied the ELM to train threshold neural networks and its generalization performance was investigated through many real-world benchmarking problems. The comparative results show that generalization performance of the ELM is better than the other learning algorithms for threshold networks with much faster learning speed. In fact, the ELM is one of the most successful learning algorithms in decades for threshold networks and it is even suitable for on-line (hardware) training.

3. Huang [18] proposed a constructive method for two-hidden layer feedforward neural networks (TLFNs) which can learn training observations with arbitrarily small error. By using this constructive method, the ELM is capable of training TLFNs and the new algorithm is called

modular implementation of extreme learning machine (MI-ELM). In this algorithm, the training samples are split into several groups and every group is learned by a SLFN trained by ELM. All the SLFNs share a same hidden layer and the outputs of the SLFNs are selected by a group of neural quantizers. Similar to the ELM, the MI-ELM analytically determines the network parameters at once. In the MI-ELM, the learning cost is further reduced and its learning speed can be extremely fast. Theocratical proofs were given to estimate the network parameters. Many real-world benchmarking cases including both regression and classification problems were used to evaluate the performance of the proposed algorithm.

4. Evolutionary/genetic algorithms are known for their capabilities of searching for global optima. To provide an alternative to the applications which concern prediction time and network size more than learning time, a hybrid of ELM and a modified form of differential evolution (DE), which is one of the most effective evolutionary algorithms, has been proposed. This new hybrid algorithm is called evolutionary extreme learning machine (E-ELM). In the E-ELM, the input weights and biases are optimized by the DE instead of randomly selected and fixed so that less hidden neurons are required and more compact networks are obtained. It was shown in many experiments that E-ELM outperforms the other learning algorithms including the gradient-based

learning algorithms and evolutionary learning algorithms in terms of generalization performance and prediction time. The E-ELM may be useful to those prediction time concerned applications such as embedded systems.

5. Neural network (NN) ensembles constructed by bagging method are known for their higher accuracy and stability over individual neural network. However, the very long training time becomes the bottleneck in their applications when the neural networks of the ensemble are trained by the conventional gradient-based learning algorithms. A new algorithm called bagging extreme learning machines (B-ELM) which constructs bagging ensemble by using the ELM was proposed. The proposed ensemble can achieve not only higher accuracy but also shorter training time than the conventional NN ensembles trained by gradient-based learning algorithms. This algorithm is suitable for the applications which have demand of high accuracy.

1.4 Organization of the Thesis

This thesis is made up of two parts. Part I is concerned with fast learning algorithms; Part II is concerned with two variants of the ELM which can achieve compact network size or better generalization performance.

Chapter 2 introduces the background of our research and reviews the

typical learning algorithms for multilayer feedforward neural networks and radial basis function neural networks. Some other types of learning algorithms are also introduced.

Part I (Chapter 3 to 5) presents the fruits of our research work on novel fast learning algorithms for neural networks.

Chapter 3 proposes a novel learning algorithm, namely the extreme learning machine (ELM), for generic SLFNs which can achieve satisfactory generalization performance with very short learning time.

In Chapter 4, the ELM is applied to train the SLFNs with threshold activation functions and its performance is investigated.

Chapter 5 proposes a new learning algorithm, namely the modular implementation of extreme learning machine (MI-ELM), for TLFNs based on the constructive method proposed by Huang [18] and the ELM. A proper estimation of the learning parameters and the corresponding mathematical proofs are presented. Without loss in accuracy, the learning speed of MI-ELM is extremely fast.

Part II (Chapter 6 and 7) presents another two algorithms namely the evolutionary extreme learning machine (E-ELM) and bagging extreme learning machines (B-ELM) which can achieve compact network size or better generalization performance.

Chapter 6 proposes a hybrid learning algorithm, namely the evolutionary extreme learning machine (E-ELM), for SLFNs based on ELM which uses differential evolution (DE) algorithm to optimize the input weights so that better generalization performance and more compact networks can be achieved.

Chapter 7 proposes an algorithm to construct fast neural network ensemble with extreme learning machines (B-ELM) which outperforms conventional neural network ensembles in terms of generalization performance and learning speed.

Chapter 8 summarizes the thesis and give some suggestions on future research work.

Chapter 2

Literature Review

Many gradient-based learning algorithms were proposed in last two decades, and they have been successfully applied in many neural networks applications. However, the gradient-based algorithms cannot be used to train the networks with threshold/digital nodes directly as the threshold function are non-differentiable. Furthermore, since the training process of the gradient-based methods are time-consuming, they cannot satisfy the requirement of time-critical applications. To solve these problems, some analytical learning methods were proposed and they are attracting more and more research interests.

In this chapter, we first review the gradient-descent learning algorithms in Section 2.1. Then some advanced gradient-based learning methods are briefly discussed in Section 2.2. Existing learning algorithms for threshold

networks are introduced in Section 2.3. Finally, analytical learning algorithms are briefly discussed in Section 2.4.

2.1 Gradient-Descent Learning Algorithms

Gradient-descent learning algorithms are widely used to train the feedforward neural networks. Out of these algorithms, back-propagation (BP) is one of the most successful learning algorithms. A lot of variants of BP have been proposed to speed up convergence.

2.1.1 Back-Propagation

Back-propagation algorithm was initially proposed by Werbos [19], but it had been ignored for a decade. In 1980s, it was rediscovered independently by Rumelhart, et al [8] and Parker [20]. In the back-propagation algorithm, input vectors and the corresponding target vectors are used to train a network to approximate the target function iteratively. SLFNs with biases, a sigmoid layer, and a linear output layer are capable of approximating any function with a finite number of discontinuities [1, 2, 21]. Hence, for the sake of simplicity, the default feedforward network used in this chapter is SLFN unless otherwise stated.

Standard back-propagation adapts the weights along the negative of the

gradient of the cost function. The term backpropagation refers to that the weights are updated backwards from the output layer to the input layer. The back-propagation learning algorithm adjusts the weights as in Figure

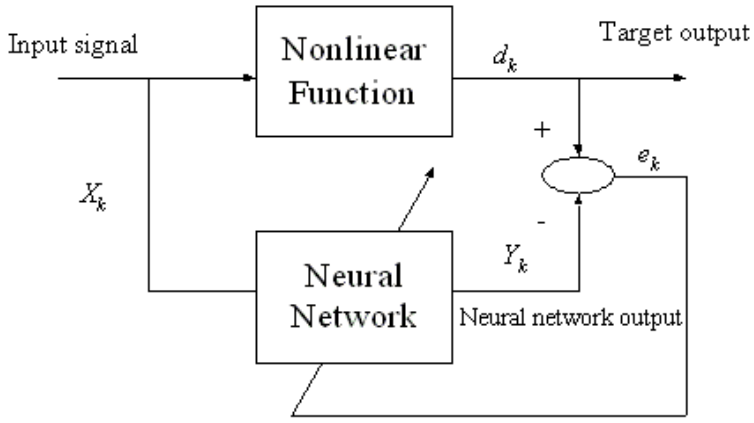


Figure 2.1: Learning of back-propagation

2.1. For each input, compute the instantaneous summed squared error (cost function)

$$E[e_k^2] = \sum_{n=1}^{p} (d_k^n - Y_k^n)^2 \tag{2.1}$$

Then we update the weights as

$$\begin{aligned} \mathbf{W}(k+1) &= \mathbf{W}(k) + \eta \Delta \mathbf{W}(k) \\ &= \mathbf{W}(k) - \eta \nabla \mathbf{E}(k) \end{aligned} \tag{2.2}$$

where the parameter η indicates learning rate, $\mathbf{W}(k)$ is the current weight vector, and $\nabla \mathbf{E}(k)$ denotes the current gradient of the cost function.

Properly trained back-propagation networks tend to give reasonable response when presented with inputs that they have never seen. Unfortunately, it has to test the whole training set through the network for calculating the

cost function E. This can slow down training for bigger training sets. Therefore, normally the update is based just on the gradient for the actual training patterns.

To provide faster convergence, Hagan, et al [22] and Pearlmutter [23] discussed a steepest descent method with momentum. Momentum allows a network to respond not only to the local gradient, but also to recent trends in the error surface. Acting like a low-pass filter, momentum allows the network to ignore small features in the error surface. Without momentum a network may get stuck in shallow local minima. With momentum a network can slide through such local minima.

Momentum can be added to back-propagation learning by making weight changes equal to the sum of a fraction of the last weight change and the new change suggested by the back-propagation rule (cf. equation (2.3)). The magnitude of the effect that the last weight change is allowed to have is mediated by a momentum constant α, which can be any number between 0 and 1. When the momentum constant is 0, a weight change is based solely on the gradient.

$$\Delta \mathbf{W}(k) = -\eta \nabla \mathbf{E}(k) + \alpha \Delta \mathbf{W}(k-1) \tag{2.3}$$

The momentum can speed up training in very flat regions of the error surface and suppresses weight oscillation in step valleys or ravines. A good choice of η and α is essential for training success and speed. Adjusting these parameters

by trial and error can be very difficult and might take a very long time for most complicated tasks.

2.1.2 Adaption of Learning Rate

The determination of the values for learning rate is critical to the training with back-propagation algorithms. A small learning rate may result in slow learning speed and make the networks easy to get stuck in local minima, while a large learning rate may cause the networks not to converge. One way to optimize the BP algorithm and speed up its convergence is to find proper values for the learning rate automatically. The following methods have been proposed to adjust the learning rate. Some of them also adjust the momentum parameter. The first five methods are global adaption methods which determine a learning rate for all the weights and biases, while the rest are local adaption methods where the learning rate for each different weight or bias is adapted separately at every learning epoch. Obviously, the local adaption methods are more effective in convergence but more time-consuming in adaption.

A. Fixed calculating of the learning rate

Eaton and Olivier [24] suggested a calculation of the learning rate for back-propagation using batched updates. This calculation is based on the as-

sumption that similar training patterns result in similar gradients. So it is desirable to reduce the learning rate if there are many similar training patterns. Therefore, the training set must be divided into m subsets of similar patterns. Let $N_1, N_2, \ldots, N_m$ be the sizes of these subsets. Learning rate and momentum can now be set in the following manner:

$$\eta = \frac{1.5}{\sqrt{N_1^2 + N_2^2 + \cdots + N_m^2}} \tag{2.4}$$

$$\alpha = 0.9 \tag{2.5}$$

It is the simplest the method to estimate the learning rate η and momentum α, however the performance is not ideal for most application [25].

B. Decreasing learning rate

Darken and John [26] proposed a method which decreases the learning rate during the training. This so called "Search-Then-Converge" strategy is suggested for back-propagation using updates for every training pattern. Starting with a big learning rate $\eta(0)$, the value is decreased during the training later on (cf. equation (2.6)).

$$\eta(k) = \frac{\eta(0)}{1 + \frac{k}{\gamma}}, \tag{2.6}$$

where k denotes the learning iteration and γ is a constant parameter used to adjust this learning rate schedule with respect to the total training period. After the first γ learning steps the learning rate is halved by this update rule.

Big learning rates are useful in the early training phase but result in oscillation later on. By using a decreasing learning rate during the training the advantages of big values (fast learning in the early learning phase) and small values (good asymptotic behavior) can be combined by a proper value for γ. However, a good γ is usually determined by trial and error, just like standard back-propagation learning algorithm.

C. Learning rate adaptation for each training pattern

Schmidhuber [27] proposed a method which updates the weights for every training pattern. It calculates a new learning rate for every update and doesn't use any momentum (cf. equation (2.7)). Therefore, the tangent in the error surface of the actual training pattern at the current position is used. The new values for every connection are found by calculating the point of intersection with the zero plane. For practical reasons, it is necessary to define an upper limit for a single learning step. There also may be some error surfaces which never reach the zero plane. For those surfaces, a small constant value E_{offset} is subtracted to make sure that zero points exist.

$$\eta(k) = \min\{\frac{E_p - E_{offset}}{\|\nabla \mathbf{E}_p(k)\|}, \eta_{max}\} \tag{2.7}$$

Schmidhuber [27] emphasized that his algorithm is able to escape from local minima. Nevertheless, this may result in very big updates, which might corrupt the whole network in one learning step. It is doubtful if this is desirable especially for networks which already classify most of the training

patterns correctly. This strategy also can't handle very big training sets where some wrong classified patterns are likely to exist [25].

D. Evolutionarily adapted learning rate

Salomon [28] proposed an algorithm called evolutionary adapted learning rate which uses a simple evolution strategy to adjust the learning rate. Starting with some η, the next update is done by using an increased and a decreased learning rate. The one which results in better performance is used as a starting point for the next update. The learning procedure of each iteration is described as follows:

1. Create two equal networks and an initial learning rate.

2. Adjust the weights of both networks as

$$\Delta \mathbf{W}(k) = -\eta(k) \frac{\nabla \mathbf{E}(k)}{\|\nabla \mathbf{E}(k)\|}. \tag{2.8}$$

3. Discard the networks and restart with the former network and the initial learning rate, if both total errors have been increased (Back-tracking). In the case of decreasing total errors use the network with a smaller total error with learning rates $\eta(k)\beta$ and $\eta(k)\frac{1}{\beta}$ to the next step.

This simple strategy can handle almost any initial learning rate and greatly improves the performance of back-propagation using batched updates. How-

ever, it doubles the calculation time.

E. Angle driven learning rate adaptation

Chan and Fallside [29] proposed an algorithm called angle driven learning rate adaption which adapts learning rate and momentum during the training. In this algorithm, the angle $\theta(k)$ between $\nabla E(k)$ and $\Delta \mathbf{W}(k-1)$ is calculated (cf. equation (2.10)). The adaptation tries to adjust this angel to 90^o. As long as the angle is less than 90^o, the learning rate is increased otherwise it is decreased. Every iteration of this learning algorithm is stated as follows:

1. Calculate
$$\cos\theta(k) = \frac{-\nabla \mathbf{E}(k) \cdot \Delta \mathbf{W}(k-1)}{\|\nabla \mathbf{E}(k)\| \|\Delta \mathbf{W}(k-1)\|}. \tag{2.9}$$

2. Adapt the learning rate
$$\eta(k) = \eta(k-1)(1 + 0.5\cos\theta(k)). \tag{2.10}$$

3. Adapt the momentum
$$\alpha(k) = \alpha(0)\frac{\|\nabla \mathbf{E}(k)\|}{\|\Delta \mathbf{W}(k-1)\|}. \tag{2.11}$$

4. Adjust the weights
$$\Delta \mathbf{W}(k) = \eta(k)(\nabla \mathbf{E}(k) + \alpha(k)\Delta \mathbf{W}(k-1)). \tag{2.12}$$

Unfortunately, the learning rate is often adapted too rapidly which results in very big learning rates [30].

F. Learning rate adaption by sign changes

Silva and Almeida [31] proposed an algorithm which uses separate learning rates of each weight η_{ij} for each connection. The adaptation of these learning rates is done by observing the signs of the gradients in the last two learning epoches. As long as no change in sign is detected the corresponding learning rate is increased. If the sign changes, the learning rate is decreased. Every learning iteration of this algorithm is described as follows:

1. Choose small initial value for each $\eta(0)_i$

2. Adapt the learning rate as

$$\eta_i(k) = \begin{cases} \eta_i(k-1) * u, & \text{if } \frac{\partial E}{\partial w_i}(k) * \frac{\partial E}{\partial w_i}(k-1) \geq 0 \\ \eta_i(k-1) * d, & \text{otherwise.} \end{cases} \tag{2.13}$$

3. Update the connections as

$$\Delta w_i(k) = -\eta_i(k)(\frac{\partial E}{\partial w_i} + \alpha * \Delta w_i(k-1)). \tag{2.14}$$

where $i = 1, \cdots, n$, and n is the number of weights. According to Silva and Almeida [31], the parameters u and d can be easily chosen as long as $u \approx d$ holds. It also uses a backtracking strategy which restarts an update step if the total error increases. For this restart, all learning rates are halved.

Tollenaere [32] proposed another algorithm called SuperSAB which is quite similar to Silva and Almeida's approach [31]. It has modified the update

rule (Equation (2.13)) as

$$\Delta w_i(k) = \begin{cases} \eta_i(k) * \frac{\partial E}{\partial w_i} + \alpha * \Delta w_i(k-1), & \text{if } \frac{\partial E}{\partial w_{ij}}(k) * \frac{\partial E}{\partial w_i}(k-1) \geq 0 \\ \Delta w_i(k-1), & \text{otherwise.} \end{cases} \tag{2.15}$$

Instead of using global backtracking only local backtracking is used. In this algorithm, the weight remains unchanged if the signs of the last two gradient are the same.

G. Delta-bar-delta technique

Jacobs [33] also proposed a local learning rate adaptation algorithm. In contrary to the former approaches, his delta-bar-delta algorithm controls the learning rates by observing the sign changes of an exponential averaged gradient. He increases the learning rates by adding a constant value instead of using multiplying factor. Every learning iteration of this algorithm is described as follows:

1. Choose some small initial value for every $\eta_i(0)$

2. Adapt the learning rates as:

$$\eta_i(k) = \begin{cases} \eta_i(k-1) * u, & \text{if } \frac{\partial E}{\partial w_i}(k) * \overline{\delta}_i(k-1) > 0 \\ \eta_i(k-1) + d, & \text{if } \frac{\partial E}{\partial w_i}(k) * \overline{\delta}_i(k-1) < 0 \\ \eta_i(k-1), & \text{otherwise,} \end{cases} \tag{2.16}$$

where the exponential averaged gradient is denoted by $\overline{\delta}_{ij}(k)$:

$$\overline{\delta}_i(k) = (1-\phi) * \frac{\partial E}{\partial w_i}(k) + \phi * \overline{\delta}_i(k-1) \tag{2.17}$$

3. Update the connections as

$$\Delta w_i(k) = -\eta_i(k) * \frac{\partial E}{\partial w_i} \tag{2.18}$$

When ϕ is set to 0, the algorithm is quite similar to Silva and Almeida's approach [31]. However, it is quite difficult to find proper values for all these parameters, particularly for u. Small values may result in slow adaptations while big ones endanger the learning process.

Resilient propagation

Riedmiller and Braun [34, 35] proposed an algorithm called **r**esilient **prop**agation (RPROP) uses an adaptive version of the "Manhattan-Learning" rule [36]. In contrast to all other described algorithms, the "Manhattan-Learning" rule uses a fixed update step size not influenced by the magnitude of the gradient. Only the sign of the derivative is used to find the proper update direction:

$$\Delta_i w_i(k) = \begin{cases} -\Delta_0, & \text{if } \frac{\partial E}{\partial w_i}(k) > 0 \\ +\Delta_0, & \text{if } \frac{\partial E}{\partial w_i}(k) < 0 \\ 0, & \text{otherwise,} \end{cases} \tag{2.19}$$

where Δ_0 denotes the update-value which is a problem-dependent constant. RPROP improves "Manhattan-Learning" rule by using independent update

step sizes Δ_i for every connection:

$$\Delta_i(k) = \begin{cases} \Delta_i(k-1) * \alpha^+, & \text{if } \frac{\partial E}{\partial w_i}(k) * \frac{\partial E}{\partial w_i}(k-1) > 0 \\ \Delta_i(k-1) * \alpha^-, & \text{if } \frac{\partial E}{\partial w_i}(k) * \frac{\partial E}{\partial w_i}(k-1) < 0 \\ \Delta_i(k-1), & \text{otherwise.} \end{cases} \tag{2.20}$$

where α^+ and α^- are constants and $0 < \alpha^- < 1 < \alpha^+$. The step sizes are bound by upper and lower limits in order to avoid oscillation and arithmetic underflow of floating point values. RPROP's learning speed is quite encouraging. Its performance is almost independent from the initial setting of the update step size. Networks with very good performance have been trained.

Fahlman [37] enhanced RPROP by modifying the derivative $\frac{\partial E}{\partial w_i}$. The enhanced learning algorithm is named "quickprop". However, some [38, 39] argue that the performance of the quickprop learning algorithm is not as good as expected. Vrahatis et al [39] proposed an improved quickprop learning algorithm which has a faster convergence than standard quickprop.

2.1.3 Stopping Criteria

Besides the local minima and learning rate issues, another problem in training with the BP algorithm is stopping criteria. It is a common sense that the training should be stopped at the right time. However, it is hard to determine when and how the training should be stopped.

A. Stopping Based on Learning Epochs

Limiting the number of learning epochs/iterations is the most straightforward way to stop the training. It simply stops the training once the predefined maximum number of training epochs is reached. However it does not depend on any information or feedback from the networks. Using this stopping criterion, one will never know the status of the training. Thus, the only merit of this method we can tell is simplicity.

B. Stopping Based on Training Error

A common way to stop training is based on the training mean-squared-error (MSE). When using this method, the training goes on until the goal of training MSE is achieved. However, one can never foretell before training what is the acceptable level of training for a specific problem. If the MSE goal is set too small, either it cannot be achieved (the training may never end) or the network is overtrained. Overtraining means a network is excessively trained such that it responds poorly to the unknown data though its training error is very small. On the contrary, if the MSE goal is set too high, this may lead the network to be insufficiently trained. This method is even difficult to implement for to classification problems as the MSE is an indirect measure in classification.

An alternative to this method is to keep the network trained until the

decrease of the training MSE from current learning epoch to the last is below a given threshold. The idea is to stop the training when it is about to converge. Unfortunately, this method may stop the training while it is stuck in a local minima instead of converging to the global minimum.

C. Stopping Based on Generalization Performance

It is well known that the training MSE is not a direct indicator of generalization performance. Even the training MSE is very small, the network may still be overtrained so that it does not perform well with the data not belonging to the training dataset.

One popular solution to the overtraining issue is cross-validation [40]. The idea is to stop the training when the network is about to achieve the best generalization performance. In the cross-validation, a validation dataset is needed which is normally obtained practically by splitting the available training dataset. The MSE on the validation dataset is used as an indicator of generalization performance as the observations in the validation dataset normally do not appear in the training dataset. The training is stopped once the validation MSE starts to increase.

The effect of cross-validation largely depends on the selection of validation dataset. Only when the validation data is able to well represent distribution and characteristics of the testing data, can a good generalization

performance be guaranteed. Many researcher have addressed this issue of selecting the proper validation dataset. Another problem with this method is that cross-validation shrinks the size of the training dataset. It makes the situation even worse for some applications with few training data. Obviously, no learning algorithm can perform well without sufficient training observations.

2.2 Advanced Gradient-Based Methods

Gradient-descent methods are also called first-order searching methods. These methods adjust the weights along the steepest descent (negative of gradient) direction of the cost function. However, the steepest descent of the cost function does not necessarily result in the fastest convergence. The essence of neural networks training is minimizing the cost function. As numeric optimization is a vast and mature field (cf. [41, 42, 43]), only some typical advanced gradient-based searching methods with faster convergence than gradient-descent methods are introduced in this section.

2.2.1 Conjugate Gradient Algorithms

In order to produce generally faster convergence than steepest descent directions, conjugate gradient algorithms were proposed. There are several

conjugated gradient learning algorithms available in literature. In this section, four different variations of conjugate gradient algorithms are briefly reviewed. In the above mentioned gradient-descent learning algorithms, the length of the weight update (step size) along the steepest gradient direction is decided by the learning rate which has various adaption methods as discussed. In the conjugate gradient algorithms, a line search is made along the conjugate gradient direction to determine the step size in order to minimize the cost function.

Fundamentals

Most conjugate learning algorithms [22, 44, 45] start with the gradient-descent direction:

$$\mathbf{S}(0) = -\nabla \mathbf{E}(k) \tag{2.21}$$

The weights are updated at each iteration as

$$\mathbf{W}(k+1) = \mathbf{W}(k) + \eta(k)\mathbf{S}(k) \tag{2.22}$$

where $\eta(k)$ is obtained by a line search routine along the current search direction, and the searching direction $\mathbf{S}(k)$ is determined to be conjugate to the previous direction:

$$\mathbf{S}(k) = -\nabla \mathbf{E}(k) + \alpha(k)\mathbf{S}(k-1) \tag{2.23}$$

There are three well-known ways to determine the $\alpha(k)$:

- Fletcher-Reeves [22, 44] method:

$$\alpha(k) = \frac{\nabla \mathbf{E}^T(k)\nabla \mathbf{E}(k)}{\nabla \mathbf{E}^T(k-1)\nabla \mathbf{E}(k-1)} \tag{2.24}$$

- Polak-Ribiere [22, 44] method:

$$\alpha(k) = \frac{[\nabla \mathbf{E}(k) - \nabla \mathbf{E}(k-1)]^T \nabla \mathbf{E}(k)}{\nabla \mathbf{E}^T(k-1)\nabla \mathbf{E}(k-1)} \tag{2.25}$$

- Hestenes-Steifel [45] method:

$$\alpha(k) = \frac{[\nabla \mathbf{E}(k) - \nabla \mathbf{E}(k-1)]^T \nabla \mathbf{E}(k)}{\nabla \mathbf{E}^T(k-1)\mathbf{S}(k-1)} \tag{2.26}$$

Restart Rules

In quadratic performance surfaces, the minimum will be found in n iterations, where n is the number of network parameters (biases and weights). However, the convergence cannot be guaranteed if the performance surface is nonquadratic. Fortunately, this problem can be overcome by resetting the searching direction to the negative of the gradient by every n iterations. Other restarting rules have also been proposed to improve the training efficiency. Powell-Beale [46] rule is one of the most effective ways. It is an improved version of Beale's [47] method.

In the Powell-Beale [46] rule, the searching direction will start if there is very little orthogonality left between the current gradient and the previous gradient, which is tested with the following inequality:

$$\nabla \mathbf{E}^T(k-1)\nabla \mathbf{E}(k) \geqslant 0.2\|\nabla \mathbf{E}(k)\|^2 \tag{2.27}$$

If inequality (2.27) is satisfied, the searching direction is reset to the negative of the gradient.

Line Search Methods

As mentioned above, a line search is performed at each iteration of learning. Some typical line search methods are briefed as follows:

1. *Golden Section Search* [22]: It is a linear search that does not require the calculation of the slope. Firstly, an interval in which the minimum of the cost function occurs is allocated. This is accomplished by evaluating the cost function at a sequence of points, starting at a distance of delta and doubling in distance each step, along the search direction. When the cost function increases between two successive iterations, a minimum has been bracketed. The next step is to reduce the size of the interval containing the minimum. Two new points are located within the initial interval. The values of the cost function at these two points determine a section of the interval that can be discarded, and a new interior point is placed within the new interval. This procedure is continued until the interval of uncertainty is reduced to a width of tolerance.

2. *Brent's Search* [48]: It is a linear search, which is a hybrid combination of the golden section search and a quadratic interpolation. Brent's

search begins with the same interval of uncertainty which is used in the golden section search, but some additional points are computed. A quadratic function is then fitted to these points and the minimum of the quadratic function is computed. If this minimum is within the appropriate interval of uncertainty, it is used in the next stage of the search and a new quadratic approximation is performed. Otherwise, a step of the golden section search is performed.

3. *Hybrid Bisection-Cubic Search* [49]: It is a hybrid combination of bisection and cubic interpolation. In the bisection algorithm, one point is located in the interval of uncertainty and the cost function and its derivative are computed. Based on this information, half of the interval of uncertainty is discarded. In the hybrid algorithm, a cubic interpolation of the function is obtained by using the value of the cost function and its derivative at the two end points. If the minimum of the cubic interpolation falls within the known interval of uncertainty, then it is used to reduce the interval of uncertainty. Otherwise, a step of the bisection algorithm is used.

4. *Charalambous' Search* [50]: It uses a cubic interpolation, together with a type of sectioning. It does require the computation of the derivatives (back-propagation) in addition to the computation of cost function, but it overcomes this limitation by locating the minimum with fewer steps.

5. *Backtracking Search* [51]: It begins with a step multiplier of 1 and then backtracks until an acceptable reduction in the performance is obtained. On the first step it uses the value of performance at the current point and at a step multiplier of 1. Also it uses the value of the derivative of performance at the current point, to obtain a quadratic approximation to the cost function along the search direction. The minimum of the quadratic approximation becomes a tentative optimum point (under certain conditions) and the performance at this point is tested. If the performance is not sufficiently reduced, a cubic interpolation is obtained and the minimum of the cubic interpolation becomes the new tentative optimum point. This process is continued until a sufficient reduction in the performance is obtained. It is known to be especially suitable for quasi-Newton learning algorithm which is discussed in the next subsection.

Scaled Conjugate Gradient Algorithm

As introduced above, the line search methods are computationally intensive, there is one algorithm called scaled conjugate gradient learning algorithm which does not require the line search. At each iteration, the $\eta(k)$ in equation (2.27) is determined as

$$\eta(k) = \frac{\nabla \mathbf{E}^T(k)\mathbf{S}(k)}{\mathbf{S}^T(k)\mathbf{H}(k)\mathbf{S}(k) + \lambda\|\mathbf{S}(k)\|^2} \tag{2.28}$$

where $\mathbf{H}(k)$ is the Hessian matrix (second derivatives) of the cost function index at the current values of the weights and biases and λ is a parameter decided by the user. One may think that this algorithm is computationally expensive as the Hessian matrix is involved. However, there are some fast methods to estimate the product of a vector by the Hessian matrix. One example could be the perturbation method [52, 53]:

$$\mathbf{S}^T\mathbf{H} = \frac{\nabla\mathbf{E}(\mathbf{W} + \epsilon\mathbf{S}) - \nabla\mathbf{E}(\mathbf{W})}{\epsilon} + O(\epsilon) \tag{2.29}$$

The parameter λ is normally determined by trial and error. The basic strategy is that if the error begins to increase, the λ should also be increased until the cost function decreases.

2.2.2 Quasi-Newton Algorithms

Newton's method is an alternative to the conjugate gradient methods for fast optimization. The basic step of Newton's method is

$$\mathbf{W}(k+1) = \mathbf{W}(k) - \mathbf{H}(k)\nabla\mathbf{E}(k) \tag{2.30}$$

It is well known that the Hessian matrix $\mathbf{H}(k)$ needs the calculation of the second derivatives of the cost function which is computational expensive. Fortunately, there are various methods to approximate the Hessian which are less computationally expensive.

A. Pseudo-Newton Algorithm

Pseudo-Newton algorithm [54] is the simplest quasi-Newton method. It just ignores the cross terms in the Hessian matrix and use only the diagonal terms. In the pseudo-Newton algorithm, the weights are updated as

$$\Delta w_i(k) = \frac{-\frac{\partial E(k)}{\partial w_i}}{|\frac{\partial^2 E(k)}{\partial w_i^2}| + \beta} \tag{2.31}$$

where β is small constant to avoid the problem of negative curvature and a zero dominator. This is a crude approximation of the Hessian matrix.

B. Levenberg-Marquardt Algorithm

Suppose the cost function is a sum of squared error:

$$E = \sum_{i=1}^{N} e_i^2 = \mathbf{e}^T \mathbf{e} \tag{2.32}$$

where N is the number of samples in the training set, e_i is the error of i-th training sample, and $\mathbf{e} = [e_1, e_2, \cdots, e_N]$ is the vector of errors. The gradient of the cost function can be calculated by

$$\nabla \mathbf{E} = 2\mathbf{J}\mathbf{e} \tag{2.33}$$

where $\mathbf{J}$ is the Jacobian matrix given by

$$\mathbf{J} = \begin{bmatrix} \frac{\partial e_1}{\partial w_i} & \cdots & \frac{\partial e_1}{\partial w_n} \\ \cdots & \cdots & \cdots \\ \frac{\partial e_N}{\partial w_i} & \cdots & \frac{\partial e_N}{\partial w_n} \end{bmatrix} \tag{2.34}$$

where n is number of weights. Levenberg-Marquardt (LM) algorithm [17] approximates Hessian matrix by

$$\mathbf{H} \approx 2\mathbf{J}^T\mathbf{J} \tag{2.35}$$

Thus, the Newton update of the weights can be written as

$$\mathbf{W}(k+1) = \mathbf{W}(k) - [\mathbf{J}^T(k)\mathbf{J}(k) + \mu\mathbf{I}]^{-1}\mathbf{J}(k)\mathbf{e}(k) \tag{2.36}$$

When the parameter μ is zero, this is just Newton's method, using the approximate Hessian matrix. When μ is large, this becomes gradient descent with a small step size. Newton's method is faster and more accurate near an error minimum, so the aim is to shift towards Newton's method as quickly as possible. Thus, μ is decreased after each successful step (reduction in cost function) and is increased only when a tentative step would increase the cost function. In this way, the cost function will always be reduced at each iteration of the algorithm.

LM is known as one of the fastest gradient-based learning algorithms [55]. Besides LM, there are other quasi-Newton algorithms such as Davidson-Fletcher-Powell (DFP) [41] algorithm and Broyden-Fletcher-Goldfarb-Shanno (BFGS) [56] algorithm which use similar approaches to approximate Hessian matrix.

2.2.3 Learning Method Using Extended Kalman Filter

Some algorithms use the Extended Kalman Filter (EKF) to update the weighs of the network. The basic idea of EKF based learning algorithm is to find the $\mathbf{W}$ by using the following recursion [57]:

$$\mathbf{W}_j = \mathbf{W}_{j-1} + \mathbf{K}_{j-1}(\mathbf{d} - \mathbf{Y}_{j-1}) \tag{2.37}$$

where $\mathbf{K}_j$ is the Kalman gain matrix at step j, $\mathbf{Y}$ is the actual output of the network, and $\mathbf{d}$ is the desired target. EKF has been shown to require less learning iterations than gradient-descent learning algorithms. However, the gradients still need to be computed in order to obtain the Kalman gain matrix $\mathbf{K}_j$ and the iterative adapting learning methodology is used just like the other gradient-based learning algorithms.

2.3 Learning Algorithms for Threshold Neural Networks

Although the feedforward neural networks with common analog activation functions such as sigmoid and sine have proved to have great computational capabilities in various areas, the networks with threshold or hard-limiting

activation functions

$$g(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases} \tag{2.38}$$

is still desirable mainly for following reasons:

1) The threshold units are easy for digital hardware implementation[58, 59], and many digital hardware implementations of threshold networks have been proposed [60, 61].

2) The relationship between the size of the networks with threshold units and the complexity of the training are better understood [62, 63]. For example, Baum and Haussier [63] suggested the theorical lower and upper bounds on the training dataset size vs. net size needed for a given problem with boolean output such that valid generalization performance can be expected.

However, the most extensively used gradient-based learning algorithms cannot be used to train the neural networks with threshold units directly as the threshold functions are non-differentiable. Hence, in literature, a lot of research efforts have been made to modify the gradient-based learning methods in order to apply them to threshold networks.

2.3.1 Toms Algorithm

Toms[64] proposed a gradient descent learning algorithm for networks with hybrid activations which linear combines the sigmoid and hard-limiting function as equation (2.39).

$$f(x) = bS(x) + (1 - b)\theta(x) \tag{2.39}$$

where the $S(x)$ is sigmoid function and $\theta(x)$ is threshold function. The training is initiated with $b = 1$. During the training, b is gradually decreased to 0. Thus, the activation functions of hidden neurons gradually transform from purely analog (sigmoid) units to purely binary (threshold) ones. However, there is no theoretical justification that such transformation of activation function during training does not affect network performance.

2.3.2 Corwin-Logar-Oldham Algorithm

Similarly, Corwin et al[59] proposed another iterative method for training multi-layer networks with threshold functions. The sigmoid function (cf. equation (2.40)) is used in the training.

$$f(x) = \frac{1}{1 + e^{-\lambda x}} \tag{2.40}$$

If the error is small, the gain parameter λ can be gradually increased as shown in Figure 2.2 during the training until the slope of the sigmoid is sufficiently large to allow a transfer of weights to a threshold function networks

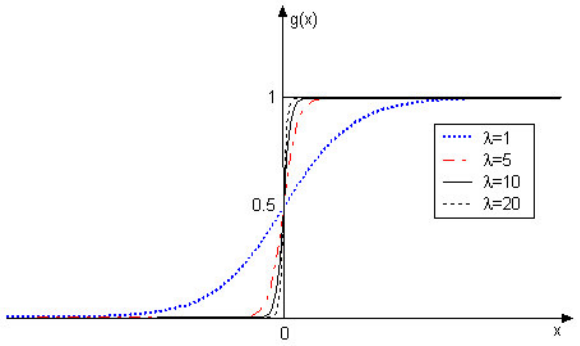


Figure 2.2: Threshold unit can only be approximated by sigmoid unit with sufficiently large gain parameter λ.

with the same architecture. In [59], it was proved that such transfer will not cause a big change in the network error. However, in many cases, the error may not be small enough to allow the λ to be increased. Thus, this algorithm cannot be applied in these applications. Similar to the Toms algorithm, this algorithm is only suitable for off-line (nonhardware implementation) training as the activation functions are fixed as threshold functions only after the training.

2.3.3 Bartlett-Downs Algorithm

Instead of calculating the derivative of the hidden units, Bartlett and Downs [58] proposed an algorithm which applies the gradient on the probability of the hidden unit's output. This algorithm is based on the assumption that weights are random variable with smooth distribution functions so that the probability of a threshold unit taking one of its two possible values is a

continuously differentiable function. The learning process of this algorithm is like BP's except that instead of the weights the distribution functions of the weights are adjusted instead of the weights. Bartlett and Downs [58] recommend using this algorithm for offline training only as the learning procedure is complicated and unsuitable for hardware implementation.

2.4 Analytical Learning Methods

Although the gradient-based learning algorithms have been extensively used, their learning efficiency is unsatisfactory especially when applied to time-critical applications. In addition, issues like choosing the proper stopping criterion and avoiding the local minima are also the obstacles in their applications. Some analytical learning methods were proposed as alternatives to the gradient-based learning algorithms. Unlike the gradient-based algorithms which adjust the weights iteratively based on the evaluation of the cost function/error, the analytical methods determine the weights directly from the relations between the training inputs and targets. The algebraic methods are normally used in the analytical algorithms. Two analytical learning methods are introduced in this section.

2.4.1 Methods for Single-Hidden Layer Feedforward Networks (SLFNs)

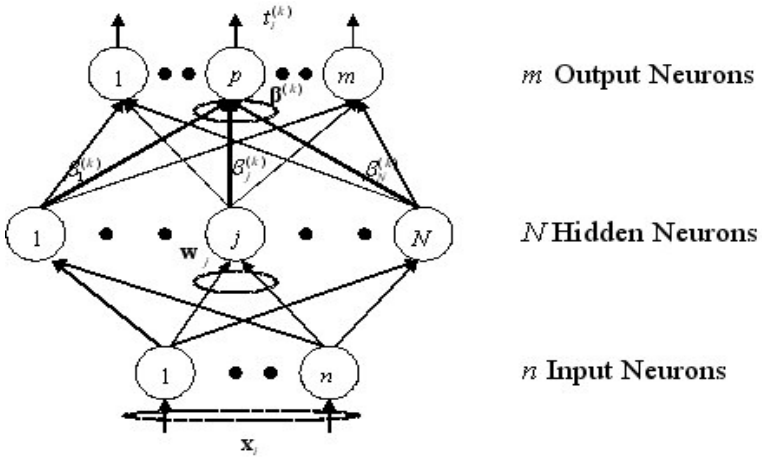


Figure 2.3: Model of SLFN

Figure 2.3 shows a single-hidden layer feedforward neural network.

Huang and Babri [16] proposed and rigorously proved an algorithm in which SLFNs with any bounded nonlinear activation function having a limit at one infinity (e.g. sigmoid, ramp, signum,etc.) and with at most N hidden neurons can learn N distinct observations $(\mathbf{x_i}, \mathbf{t_i})$ with zero error. Huang [18] further investigated this algorithm with sigmoid activation function. In this algorithm, firstly, the hidden layer output matrix $\mathbf{H}$ is calculated:

$$\mathbf{H} = [h_{ij}], \tag{2.41}$$

where

$$h_{ij} = g(\mathbf{w_j} \cdot \mathbf{x_i} + b_j), i, j = 1, \ldots, N.. \tag{2.42}$$

where $\mathbf{w}_j$ denotes the input weights linking the input layer to the j-th hidden

neuron and b_j denotes the bias of j-th hidden neuron (cf. Figure 2.3), and the input weights and hidden bias can be randomly chosen. The weight matrix $\beta = [\beta^{(1)}, \ldots, \beta^{(m)}]$, where $\beta^{(k)}$ denotes the weight connecting the hidden layer to the k-th neuron of the output layer. $\beta^{(k)}$ can be calculated as

$$\beta^{(k)} = \mathbf{H}^{-1} \begin{bmatrix} t_1^{(k)} \\ \vdots \\ t_N^{(k)} \end{bmatrix}, \tag{2.43}$$

where $t_i^{(k)}$ is the k-th element of the desired output vector t_i.

Just by using a simple matrix calculation and randomly choosing the hidden neurons, the single-hidden layer feedforward neural network with N hidden neuron is able to learn N distinct $(\mathbf{x_i}, \mathbf{t_i})$ with **zero** error. Obviously, the learning time is significantly decreased. According to the approximation theory proposed by Barron [65], the bound of approximation (generalization) error achieved by Huang and Babri's [16] method which adjusts the linear parameters of SLFNs only can be as low as $1/N^{2/n}$ on regression problems.

Ferrari and Stengel [15] also showed that SLFNs with N sigmoid hidden neurons and with randomly generated input weights and appropriately tuned hidden biases can exactly learn N distinct observations. This algorithm looks similar to the above mentioned Huang and Babri's [16] method, however they are different in following aspects:

1. Ferrari and Stengel's [15] algorithm randomly generates the input weights

but the determines the hidden biases according to the training observations and input weights, while both are randomly generated in Huang and Babri's [16] algorithm.

2. Ferrari and Stengel's [15] algorithm are only applicable to sigmoid hidden neurons, while Huang and Babri's [16] algorithm are suitable for any bounded nonlinear activation function.

3. Huang and Babri's [16] algorithm has been rigorously proven, while Ferrari and Stengel's [15] algorithm cannot be proved as the determination of the hidden biases depends on training observations and input weights.

2.4.2 Brief of Huang's Constructive Method for Two-Hidden Layer Feedforward Networks (TLFNs)

Huang [18] proposed a novel constructive method that a **t**wo-hidden **l**ayer **f**eedforward neural **n**etwork (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 remarkably reduces the space complexity of a network compared with the methods introduced in Section 2.4.1 especially for large scale problems and makes it feasible to implement large scale systems in an ordinary computer. In this method, the observations are divided into several groups, and each group is learned by a *common standard* SLFN. Furthermore, this

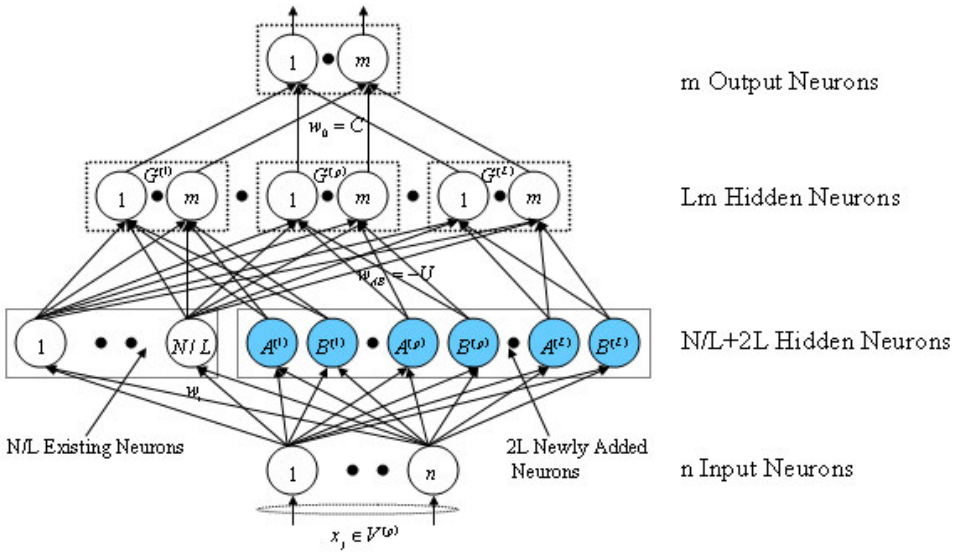


Figure 2.4: Huang's network for multi-output case

trained common standard SLFN 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 arbitrarily high accuracy (small error). Huang's network[1] (cf. Figure 2.4) 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

[1]For the sake of convenience, Huang's constructive network model [18] is abbreviated as 'Huang's network' in the context of this thesis.

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 conservative 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.

2.5 Other Types of Learning Methods

2.5.1 Radial Basis Function Networks

Radial basis function(RBF) neural networks are quite similar to feedforward neural network apparently. However, they are using distance of the sample from center of each neuron as the input. And the activation functions are radial basis function such as Gaussian function instead of sigmoidal function. With Gaussian activation function, the output of each hidden neuron is

$$g(x) = \exp(-\frac{\|\mathbf{x} - \mathbf{c}\|^2}{\gamma^2}) \tag{2.44}$$

where $\mathbf{c}$ is the center of the hidden neuron.

The common training method for RBF is also gradient descent which is similar to BP algorithm. The centers $\mathbf{c}$ and γ are updated iteratively to approximate the training observations.

Unlike global functions such as sigmoid function, radial basis function is suitable to implement sequential learning algorithms. Some algorithms [66][67][68] for RBF neural networks can selectively increase the hidden neurons during the training.

2.5.2 Support Vector Machine

Initially, support vector machines (SVM) [69] was proposed to solve binary classification or pattern recognition problems. The objective of SVM is to find a decision rule with acceptable generalization performance based on the given training dataset. The term support vector refers to a small subset of training data which is selected to estimate the unknown function. In order to solve nonlinear problems, the input space is mapped to a higher dimensional feature space by a nonlinear mapping. In some papers [70][71], when SVM approaches are used to solve regression problems, they are also called support vector regression (SVR).

A. Standard Support Vector Machines

Given a training set $\aleph = \{(\mathbf{x}_i, t_i) | \mathbf{x}_i \in R^n, t_i \in \mathbf{R}, i = 1, \cdots, N\}$. The output of SVM [72][71] is

$$f(x) = \sum_{i=1}^{N} (\alpha - \alpha')t_i K(\mathbf{x}, \mathbf{x}_i) + b, \tag{2.45}$$

where Lagrange multipliers α and α' correspond to the training sample $(\mathbf{x}_i, t_i)$. Training a SVM is equivalent to solving the quadratic programming problem:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i - \alpha_i')(\alpha_j - \alpha_j')K(\mathbf{x}, \mathbf{x}_i) \\ & +\epsilon\sum_{i=1}^{N}(\alpha_i + \alpha_i') - \sum_{i=1}^{N} t_i(\alpha_i - \alpha_i') \\ \text{subject to} & \sum_{i=1}^{N}(\alpha_i - \alpha_i') = 0 \\ & 0 \leqslant \alpha_i, \alpha_i' \leqslant C, i = 1, \cdots, N \end{array} \tag{2.46}$$

where C and ϵ are called cost and tolerance respectively. Both of them should be selected by users manually. In most approaches of SVM, the selection of kernel function $K(\mathbf{x}, \mathbf{x}_i)$ is dominant to the final performance. The commonly used kernel functions are linear, polynomial, radial basis function (RBF) and sigmoid. Without the loss of generality, we set the kernel function as RBF

$$K(\mathbf{x}, \mathbf{x}_i) = \exp(-\gamma \|\mathbf{x} - \mathbf{x}_i\|^2)$$

where γ is also a complexity parameter which needs to be decided by users.

The computational cost of solving the quadratic programming problem is high. Some algorithms such as sequential minimal optimization (SMO)

[13] uses working set selection method to reduce the computational cost of solving the quadratic programming problem.

Least Square Support Vector Machines

Least square support vector machines (LS-SVM) [13] is proposed as an alternative to the standard SVM. Given a training set $\aleph = \{(\mathbf{x}_i, t_i)|\mathbf{x}_i \in R^n, t_i \in \mathbf{R}, i = 1, \cdots, N\}$. The output of LS-SVM is

$$f(x) = sign(\mathbf{w}^T \varphi(\mathbf{x}) + b)$$

where $\varphi(\mathbf{x})$ is a mapping function that maps the sample from input space to feature space.

The LS-SVM classifier is obtained by finding the solution to the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & J_P(\mathbf{w}, b) = \tfrac{1}{2}\mathbf{w}^T\mathbf{w} + \tfrac{C}{2}\textstyle\sum_{i=1}^{N} e_i^2 \\ \text{subject to} \quad & e_i = t_i - [\mathbf{w}^T\varphi(x_i) + b], i = 1, \cdots, N \end{aligned} \tag{2.47}$$

The Lagrangian function for equation (2.47) is

$$L(\mathbf{w}, b; \alpha) = J_P(\mathbf{w}, b) - \sum_{i=1}^{N} \alpha_i(t_i - [\mathbf{w}^T\varphi(\mathbf{x}_i) + b] - e_i) \tag{2.48}$$

where α is Lagrangian multiplier.

LS-SVM tries to solve the following linear Karush-kuhn-Tucker (KKT) system instead of the quadratic programming problem:

$$\begin{bmatrix} \Omega + \mathbf{I}/C & 1_v \\ 1_v^T & 0 \end{bmatrix} \begin{bmatrix} \alpha \\ b \end{bmatrix} = \begin{bmatrix} y \\ 0 \end{bmatrix} \tag{2.49}$$

where $\mathbf{y} = [y_1, \cdots, y_N]^T$, $1_v = [1, \cdots, 1]^T$, $\alpha = [\alpha_1, \cdots, \alpha_N]^T$, $\Omega_{ij} = \varphi(\mathbf{x}_i)^T\varphi(\mathbf{x}_j) = K(\mathbf{x}_i, \mathbf{x}_j)$, $K()$ is a kernel function such as RBF. LS-SVM can be also implemented for online learning [73].

2.5.3 Non-Adaptive Classifiers

Unlike adaptive learning algorithms which adjust the learning parameters according to the training data and then apply the trained model to the unseen data in the prediction phase, non-Adaptive classifiers classify the unseen data directly by its similarity to the training data. There are two typical non-adaptive classifiers: one is k-nearest neighbour (k-NN) classifier and the other is Bayesian classifier.

A. k-Nearest Neighbour Classifier

k-th Nearest Neighbour (k-NN) classifier was first introduced by Fix and Hodges [74]. Given an input $\mathbf{x}$, k-NN first computes its distances to all the inputs $\mathbf{x}_i$ in the training dataset:

$$d_i = \sqrt{\sum_{i=1}^{n}(\mathbf{x} - \mathbf{x}_i)^2} \tag{2.50}$$

k-nearest neighbour refers to the k points in the training data having the closest distances to the given input $\mathbf{x}$. The class label that is most common in the k nearest neighbour are assigned by the classifier.

B. Bayesian Classifier

Bayesian classifier [75] uses Bayesian rule to determine the class that a given input should belong to and it is based on the assumption that all the attributes of the samples are independent. Given an instance vector $\mathbf{x} = [x_1, x_2, \cdots, x_n]$, the classifier computes

$$p(C_k, \mathbf{x}) = p(C_k) \prod_{i=1}^{n} p(x_i|C_k) \tag{2.51}$$

Then according Bayesian rule, classifier assign the instance to the class that can maximize equation (2.51).

There is one common problem faced by non-adaptive classifiers, i.e. their prediction can be very slow as they postpone all the computational cost to the prediction phase. For Bayesian classifier, there is another problem that its assumption on the independence of all the attributes is rare in real applications.

2.6 Summary

Various gradient-based learning algorithms have been reviewed in this chapter. Even though some advanced searching methods such as Levenberg-Marquardt learning algorithm converge faster than the standard BP algorithm, they are still slow in learning because of the iterative updating strategy and the intensive computation involved. Furthermore, all the gradient-

based learning algorithms are prone to stuck in local minima and proper stopping criterion needs to be selected in order to avoid insufficient or over training. As discussed in Section 2.3, the gradient-based learning algorithms cannot be applied in the training of threshold neural networks directly as the threshold functions are non-differentiable. In conclusion, the conventional gradient-based learning algorithms are unsuitable for applications needing fast learning capabilities or threshold activation functions.

Kernel methods such as SVM and its variants have been shown to give good generalization performance in recent research. However, they are complex in both training and prediction and difficult to implement. Non-adaptive classifiers waive the burden of training, but they suffer slow prediction time.

In contrast, analytical learning algorithms can determine the parameters of the networks at once so that the learning speed can be much faster. More activation functions can be used as analytical methods need not compute the gradients during the training process. Hence, analytical algorithm is the main approach of our research although not much work has been done in this area so far. Furthermore, the analytical learning algorithms are easy to use as users need not worry about selecting proper stopping criterion or being trapped in the local minima.

Part I

Fast Learning Algorithms

Chapter 3

Extreme Learning Machine (ELM) for Generic SLFNs

As we discussed in the chapter 2, the conventional gradient-based learning algorithms are usually far slower than required. It is not surprising to see that it may take several hours, several days, and even more time to train neural networks by using gradient-based learning methods. Besides learning algorithm for multilayer feedforward neural networks, there is a surge of interest recently in radial basis function (RBF) networks and kernel learning methods such as support vector machines (SVM). Although they have been shown to have good generalization performance on many problems, their learning procedures are complex and time-consuming as well. Hence, faster learning algorithms are needed for neural networks applications demanding

very short training time.

3.1 Introduction

From a mathematical point of view, research on the approximation capabilities of feedforward neural networks has focused on two aspects: universal approximation on compact input sets and approximation in a finite set of training samples. Many researchers have explored the universal approximation capabilities of standard feedforward neural networks. Hornik[76] proved that if the activation function is continuous, bounded and nonconstant, then continuous mappings can be approximated in measure by neural networks over compact input sets. Leshno[77] improved the results of Hornik[76] and proved that feedforward networks with a nonpolynomial activation function can approximate (in measure) continuous functions. In real applications, the neural networks are trained in finite training set. For function approximation in a finite training set, Huang and Babri [16] shows that a **s**ingle-hidden **l**ayer **f**eedforward neural **n**etwork (SLFN) with at most N hidden nodes and with almost any nonlinear activation function 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 works as well as in almost all practical learning algorithms of feedforward neural networks. According to the

approximation theory proposed by Barron [65], if only the linear parameters (the weights connecting the hidden layer to the output layer) of SLFN are adapted, the approximation error can be as low as $1/\tilde{N}^{2/n}$ on regression problems, where $\tilde{N}$ denotes the number of hidden neurons and n denotes the dimension of the input. In Huang and Babri's [16] method, since the $\tilde{N} = N$, the generalization error it can achieve can be as low as $1/N^{2/n}$.

Traditionally, the parameters of the feedforward networks need to be tuned and thus there exists the dependency between different layers of parameters (weights and biases). For many years gradient descent-based methods have been popular methods used in various learning algorithms of feedforward neural networks. However, as discussed in Chapter 2, they have some common issues which make them unsuitable for some applications.

It has been shown [18, 78] that SLFNs (with N hidden nodes) with randomly chosen input weights and hidden layer biases (and such hidden 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 need to adjust the input weights and first hidden layer biases in applications. In fact, some simulation results on artificial and real large applications in our work[79] have shown that this method not only makes learning extremely fast but also produces good generalization performance.

In this chapter, we first rigorously prove that the input weights and hidden layer biases of SLFNs can be randomly assigned if the activation functions in the hidden layer are infinitely differentiable. After the input weights and the hidden layer biases are chosen randomly, SLFNs are 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, we propose a simple learning algorithm for SLFNs called **e**xtreme **l**earning **m**achine (**ELM**) whose learning speed can be thousands of times faster than traditional feedforward network learning algorithms like back-propagation algorithm while obtaining better generalization performance. Different from traditional learning algorithms the proposed learning algorithm not only tends to reach the smallest training error but also the smallest norm of weights. Bartlett's theory on the generalization performance of feedforward neural networks[80] states for feedforward neural networks reaching smaller training error, the smaller the norm of weights is, the better generalization performance the networks tend to have. Therefore, the proposed learning algorithm tends to have good generalization performance for feedforward neural networks.

The new proposed learning algorithm has very small training error, obtains a small weight norm, has good generalization performance, and runs extremely fast. In order to differentiate it from other popular SLFN learning

algorithms, we call the algorithm the extreme learning machine.

3.2 Introduction of Moore-Penrose Generalized Inverse

In this section, the Moore-Penrose generalized inverse is introduced. We also consider in this section the minimum norm least-squares solution of a general linear system $\mathbf{Ax} = \mathbf{y}$ in Euclidean space, where $\mathbf{A} \in \mathbf{R}^{m\times n}$ and $\mathbf{y} \in \mathbf{R}^m$. As shown in [18, 78], the SLFNs are actually a linear system if the input weights and the hidden layer biases can be chosen randomly.

3.2.1 Moore-Penrose Generalized Inverse

The resolution of a general linear system $\mathbf{Ax} = \mathbf{y}$, where $\mathbf{A}$ may be singular and may even not be square, can be made very simple by the use of the Moore-Penrose generalized inverse[81].

Definition 3.2.1. [81, 82] *A matrix* $\mathbf{G}$ *of order* $n \times m$ *is the Moore-Penrose generalized inverse of matrix* $\mathbf{A}$ *of order* $m \times n$*, if*

$$\mathbf{AGA} = \mathbf{A}, \mathbf{GAG} = \mathbf{G}, (\mathbf{AG})^T = \mathbf{AG}, (\mathbf{GA})^T = \mathbf{GA} \tag{3.1}$$

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

3.2.2 Minimum Norm Least-Squares Solution of General Linear System

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

$$\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{y}\| = \min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{y}\| \tag{3.2}$$

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

Definition 3.2.2. *$\mathbf{x}_0 \in \mathbf{R}^n$ is said to be a minimum norm least-squares solution of a linear system $\mathbf{Ax} = \mathbf{y}$ if for any $\mathbf{y} \in \mathbf{R}^m$*

$$\|\mathbf{x}_0\| \leqslant \|\mathbf{x}\|, \forall \mathbf{x} \in \{\mathbf{x} : \|\mathbf{Ax} - \mathbf{y}\| \leqslant \|\mathbf{Az} - \mathbf{y}\|, \forall \mathbf{z} \in \mathbf{R}^n\} \tag{3.3}$$

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

Theorem 3.2.1. {p. 147 of [81], p. 51 of [82]} *Let there exist a matrix $\mathbf{G}$ such that $\mathbf{Gy}$ is a minimum norm least-squares solution of a linear system $\mathbf{Ax} = \mathbf{y}$. Then it is necessary and sufficient that $\mathbf{G} = \mathbf{A}^\dagger$, the Moore-Penrose generalized inverse of matrix $\mathbf{A}$.*

Remarks: From Theorem 3.2.1, we have the following properties that are key to our proposed ELM learning algorithm:

1. The special solution $\mathbf{x}_0 = \mathbf{A}^\dagger\mathbf{y}$ is one of the least-squares solutions of

a general linear system $\mathbf{Ax} = \mathbf{y}$:

$$\|\mathbf{Ax}_0 - \mathbf{y}\| = \|\mathbf{AA}^\dagger\mathbf{y} - \mathbf{y}\| = \min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{y}\| \tag{3.4}$$

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

$$\|\mathbf{x}_0\| = \|\mathbf{A}^\dagger\mathbf{y}\| \leqslant \|\mathbf{x}\|, \forall \mathbf{x} \in \{\mathbf{x} : \|\mathbf{Ax} - \mathbf{y}\| \leqslant \|\mathbf{Az} - \mathbf{y}\|, \forall \mathbf{z} \in \mathbf{R}^n\} \tag{3.5}$$

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

3.3 Single Hidden Layer Feedforward Networks (SLFNs) with Random Hidden Nodes

For N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$, where $\mathbf{x}_i = [x_{i1}, x_{i2}, \cdots, x_{in}]^T \in \mathbf{R}^n$ and $\mathbf{t}_i = [t_{i1}, t_{i2}, \cdots, t_{im}]^T \in \mathbf{R}^m$, standard SLFNs with $\tilde{N}$ hidden nodes and activation function $g(x)$ are mathematically modeled as

$$\sum_{i=1}^{\tilde{N}} \beta_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) = \mathbf{o}_j, \quad j = 1, \cdots, N, \tag{3.6}$$

where $\mathbf{w}_i = [w_{i1}, w_{i2}, \cdots, w_{in}]^T$ is the weight vector connecting the ith hidden node and the input nodes, $\beta_i = [\beta_{i1}, \beta_{i2}, \cdots, \beta_{im}]^T$ is the weight vector connecting the ith hidden node and the output nodes, and b_i is the threshold

of the ith hidden node. $\mathbf{w}_i \cdot \mathbf{x}_j$ denotes the inner product of $\mathbf{w}_i$ and $\mathbf{x}_j$. The output nodes are chosen linear in this algorithm.

That standard SLFNs with $\tilde{N}$ hidden nodes with activation function $g(x)$ can approximate these N samples with zero error means that $\sum_{j=1}^{\tilde{N}} \|\mathbf{o}_j - \mathbf{t}_j\| = 0$, i.e., there exist β_i, $\mathbf{w}_i$ and b_i such that

$$\sum_{i=1}^{\tilde{N}} \beta_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) = \mathbf{t}_j, \quad j = 1, \cdots, N. \tag{3.7}$$

The above N equations can be written compactly as:

$$\mathbf{H}\beta = \mathbf{T} \tag{3.8}$$

where,

$$\begin{aligned} &\mathbf{H}(\mathbf{w}_1, \cdots, \mathbf{w}_{\tilde{N}}, b_1, \cdots, b_{\tilde{N}}, \mathbf{x}_1, \cdots, \mathbf{x}_N) \\ &= \begin{bmatrix} g(\mathbf{w}_1 \cdot \mathbf{x}_1 + b_1) & \cdots & g(\mathbf{w}_{\tilde{N}} \cdot \mathbf{x}_1 + b_{\tilde{N}}) \\ \vdots & \cdots & \vdots \\ g(\mathbf{w}_1 \cdot \mathbf{x}_N + b_1) & \cdots & g(\mathbf{w}_{\tilde{N}} \cdot \mathbf{x}_N + b_{\tilde{N}}) \end{bmatrix}_{N \times \tilde{N}} \end{aligned} \tag{3.9}$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_{\tilde{N}}{}^T \end{bmatrix}_{\tilde{N} \times m} \quad \text{and} \quad \mathbf{T} = \begin{bmatrix} \mathbf{t}_1^T \\ \vdots \\ \mathbf{t}_N^T \end{bmatrix}_{N \times m} \tag{3.10}$$

As discussed in Huang et al [16, 18], $\mathbf{H}$ is called the hidden layer output matrix of the neural network; the ith column of $\mathbf{H}$ is the ith hidden node output with respect to inputs $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N$.

If the activation function g is infinitely differentiable we can prove that the required number of hidden nodes $\tilde{N} \leqslant N$. Strictly speaking, we have

Theorem 3.3.1. *Given a standard SLFN with N hidden nodes and activation function $g : R \to R$ which is infinitely differentiable in any interval, for N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$, where $\mathbf{x}_i \in \mathbf{R}^n$ and $\mathbf{t}_i \in \mathbf{R}^m$, for any $\mathbf{w}_i$ and b_i chosen from any intervals of $\mathbf{R}^n$ and $\mathbf{R}$, respectively, according to any continuous probability distribution* [1]*, then with probability one, the hidden layer output matrix $\mathbf{H}$ of the SLFN is invertible and $\|\mathbf{H}\beta - \mathbf{T}\| = 0$.*

Proof. Let us consider a vector $\mathbf{c}(b_i) = [g(\mathbf{w}_i \cdot \mathbf{x}_1 + b_i), \cdots, g(\mathbf{w}_i \cdot \mathbf{x}_N + b_i)]^T$, the ith column of $\mathbf{H}$, in Euclidean space $\mathbf{R}^N$, where $b_i \in (a, b)$ and (a, b) is any interval of $\mathbf{R}$.

Following the same proof method of Tamura and Tateishi[78](p.252) and our previous work([18],Theorem 2.1), it can be easily proved by contradiction that vector $\mathbf{c}$ does not belong to any subspace whose dimension is less than N.

Since $\mathbf{w}_i$ are randomly generated based on a continuous probability distribution, we can assume that $\mathbf{w}_i \cdot \mathbf{x}_k \neq \mathbf{w}_i \cdot \mathbf{x}_{k'}$ for all $k \neq k'$. Let us suppose that $\mathbf{c}$ belongs to a subspace of dimension $N - 1$. Then there exists a vector α which is orthogonal to this subspace:

$$(\alpha, \mathbf{c}(b_i) - \mathbf{c}(a)) = \alpha_1 \cdot g(b_i + d_1) + \alpha_2 \cdot g(b_i + d_2) + \cdots + \alpha_N \cdot g(b_i + d_N) - z = 0 \tag{3.11}$$

[1] A continuous probability distribution is a smooth density curve that models the distribution of a continuous random variable.

where $d_k = \mathbf{w}_i \cdot \mathbf{x}_k$, $k = 1, \cdots, N$ and $z = \alpha \cdot \mathbf{c}(a)$, $\forall b_i \in (a, b)$. Assume $\alpha_N \neq 0$, equation (3.11) can be further written as

$$g(b_i + d_N) = -\sum_{p=1}^{N-1} \gamma_p g(b_i + d_p) + z/\alpha_N \tag{3.12}$$

where $\gamma_p = \frac{\alpha_p}{\alpha_N}$, $p = 1, \cdots, N-1$. Since $g(x)$ is infinitely differentiable in any interval, we have

$$g^{(l)}(b_i + d_N) = -\sum_{p=1}^{N-1} \gamma_p g^{(l)}(b_i + d_p), \quad l = 1, 2, \cdots, N, N+1, \cdots \tag{3.13}$$

where $g^{(l)}$ is the l-th derivative of function g of b_i. However, there are only $N-1$ free coefficients: $\gamma_1, \cdots, \gamma_{N-1}$ for the derived more than $N-1$ linear equations, this is contradictory. Thus, vector $\mathbf{c}$ does not belong to any subspace whose dimension is less than N.

Hence, from any interval (a, b) it is possible to randomly choose N bias values $b_1, \cdots, b_N$ for the N hidden nodes such that the corresponding vectors $\mathbf{c}(b_1), \mathbf{c}(b_2), \cdots, \mathbf{c}(b_N)$ span $\mathbf{R}^N$. This means that for any weight vectors $\mathbf{w}_i$ and bias values b_i chosen from any intervals of $\mathbf{R}^n$ and $\mathbf{R}$, respectively, according to any continuous probability distribution, then with probability one, the column vectors of $\mathbf{H}$ can be made full-rank. □

Such activation functions include the sigmoidal functions as well as the radial basis, sine, cosine, exponential, and many other non-regular functions as shown in Huang and Babri[16].

Remark: Given any small positive value $\epsilon > 0$ and activation function g :

$R \to R$ which is infinitely differentiable in any interval, there exists $\tilde{N} \leqslant N$ such that for N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$, where $\mathbf{x}_i \in \mathbf{R}^n$ and $\mathbf{t}_i \in \mathbf{R}^m$, for any $\mathbf{w}_i$ and b_i chosen from any intervals of $\mathbf{R}^n$ and $\mathbf{R}$, respectively, according to any continuous probability distribution, then with probability one, $\left\|\mathbf{H}_{N\times\tilde{N}}\beta_{\tilde{N}\times m} - \mathbf{T}_{N\times m}\right\| < \epsilon$.

The validity of the remark above is obvious, otherwise, one could simply choose $\tilde{N} = N$ which makes $\left\|\mathbf{H}_{N\times\tilde{N}}\beta_{\tilde{N}\times m} - \mathbf{T}_{N\times m}\right\| < \epsilon$ according to Theorem 3.3.1.

Theorem 3.3.2. *Given a standard SLFN with $\tilde{N}$ hidden nodes with sigmoid activation function, for N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$ and $N > \tilde{N}$ and any $\mathbf{w}_i$ and b_i chosen from any intervals of $\mathbf{R}^n$ and $\mathbf{R}$, respectively, then*

$$\left\|\mathbf{H}_{N\times\tilde{N}}\beta_{\tilde{N}\times m} - \mathbf{T}_{N\times m}\right\|_2 \leqslant \frac{\sigma_1(\mathbf{H})\|T\|_2}{\sigma_{\tilde{N}}(\mathbf{H})} \tag{3.14}$$

where $\sigma_i(\mathbf{H})$ denotes the i-th largest sigular value of $\mathbf{H}$ and weights $\beta = \mathbf{H}^\dagger\mathbf{T}$.

Proof.

$$\|\mathbf{H}\beta - \mathbf{T}\|_2 = \|(\mathbf{H}\mathbf{H}^\dagger - \mathbf{I})\mathbf{T}\|_2$$

The activation function is sigmoid which means all the elements in $\mathbf{H}$ is positive, thus

$$\|(\mathbf{H}\mathbf{H}^\dagger - \mathbf{I})\mathbf{T}\|_2 \leqslant \|\mathbf{H}\mathbf{H}^\dagger\mathbf{T}\|_2 \leqslant \|\mathbf{H}\|_2\|\mathbf{H}^\dagger\|_2\|\mathbf{T}\|_2$$

Golub and Van Loan [83] (Theorem 5.7.1) proved that matrix $\mathbf{A}$ with $rank(\mathbf{A}) = m$, $\|\mathbf{A}\|_2\|\mathbf{A}^\dagger\|_2 = \sigma_1(\mathbf{A})/\sigma_m(\mathbf{A})$. From Theorem 3.3.1 we know all the

columns of $\mathbf{H}$ are not linear correlated. For $N > \tilde{N}$, $rank(\mathbf{H}) = \tilde{N}$ and

$$\|\mathbf{H}\|_2\|\mathbf{H}^\dagger\|_2 = \sigma_1(\mathbf{H})/\sigma_{\tilde{N}}(\mathbf{H})$$

therefore

$$\|\mathbf{H}\|_2\|\mathbf{H}^\dagger\|_2\|\mathbf{T}\|_2 = \frac{\sigma_1(\mathbf{H})\|T\|_2}{\sigma_{\tilde{N}}(\mathbf{H})}$$

Thus

$$\|\mathbf{H}\beta - \mathbf{T}\|_2 \leqslant \frac{\sigma_1(\mathbf{H})\|T\|_2}{\sigma_{\tilde{N}}(\mathbf{H})}$$

□

Remark: Theorems 3.3.1 and its remark assures the learning convergence of the network, while Theorem 3.3.2 indicates the upper error bound of the network with given number of hidden nodes. Theorem 3.3.2 also suggests that many factors are affecting the error bound such as the complexity of the datasets.

3.4 Proposed Extreme Learning Machine

Based on Theorems 3.3.1 and its remark we can propose in this section an efficient method to train SLFNs.

3.4.1 Learning Model of Conventional Gradient-Based Solution of SLFNs

Traditionally, in order to train an SLFN, one may wish to find specific $\hat{\mathbf{w}}_i, \hat{b}_i, \hat{\beta}$ $(i = 1, \cdots, \tilde{N})$ such that

$$\|\mathbf{H}(\hat{\mathbf{w}}_1, \cdots, \hat{\mathbf{w}}_{\tilde{N}}, \hat{b}_1, \cdots, \hat{b}_{\tilde{N}})\hat{\beta} - \mathbf{T}\| = \min_{\mathbf{w}_i, b_i, \beta} \|\mathbf{H}(\mathbf{w}_1, \cdots, \mathbf{w}_{\tilde{N}}, b_1, \cdots, b_{\tilde{N}})\beta - \mathbf{T}\| \tag{3.15}$$

which is equivalent to minimizing the cost function

$$E = \sum_{j=1}^{N} \left(\sum_{i=1}^{\tilde{N}} \beta_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) - \mathbf{t}_j \right)^2 \tag{3.16}$$

When $\mathbf{H}$ is unknown, gradient-based learning algorithms are generally used to search the minimum of $\|\mathbf{H}\beta - \mathbf{T}\|$. In the minimization procedure by using gradient-based algorithms, vector $\mathbf{W}$ which is the set of weights ($\mathbf{w}_i$,β_i) and biases (b_i) parameters $\mathbf{W}$ is iteratively adjusted in **b**ack**p**ropagation (BP) as follows:

$$\mathbf{W}_k = \mathbf{W}_{k-1} - \eta \frac{\partial E(\mathbf{W})}{\partial \mathbf{W}} \tag{3.17}$$

where η is a learning rate. In the BP learning algorithm gradients can be computed efficiently by propagating from the output to the input. However, there are several issues on BP learning algorithms:

1. When the learning rate η is too small, the learning algorithm converges very slowly. However, when η is too large, the algorithm becomes unstable and diverges.

2. Another peculiarity of the error surface that impacts the performance of the back-propagation learning algorithm is the presence of local minima. It is undesirable that the learning algorithm stops at a local minima if it is located far above a global minima.

3. Neural networks may be over-trained by using back-propagation algorithms and obtain worse generalization performance. Thus, validation and suitable stopping methods are required in the cost function minimization procedure.

4. Gradient-based learning is very time-consuming in most applications.

The aim of this chapter is to solve the above issues related with gradient-based algorithms and propose an efficient learning algorithm for feedforward neural networks.

3.4.2 Proposed Minimum Norm Least-Squares (LS) Solution of SLFNs

As rigorously proved in Theorems 3.3.1, unlike the traditional function approximation theories which require to adjust input weights and hidden layer biases, input weights and hidden layer biases can be randomly assigned if only the activation function is infinitely differentiable. It is very interesting and surprising that unlike the most common understanding that all the pa-

rameters of SLFNs need to be adjusted, the input weights $\mathbf{w}_i$ and the hidden layer biases b_i are in fact not necessarily tuned and the hidden layer output matrix $\mathbf{H}$ can actually remain unchanged once arbitrary values have been assigned to these parameters in the beginning of learning. For fixed input weights $\mathbf{w}_i$ and the hidden layer biases b_i, seen from equation (3.15), to train an SLFN is simply equivalent to finding a least-squares solution $\hat{\beta}$ of the linear system $\mathbf{H}\beta = \mathbf{T}$:

$$\|\mathbf{H}(\mathbf{w}_1, \cdots, \mathbf{w}_{\tilde{N}}, b_1, \cdots, b_{\tilde{N}})\hat{\beta} - \mathbf{T}\| = \min_{\beta} \|\mathbf{H}(\mathbf{w}_1, \cdots, \mathbf{w}_{\tilde{N}}, b_1, \cdots, b_{\tilde{N}})\beta - \mathbf{T}\| \tag{3.18}$$

If the number $\tilde{N}$ of hidden nodes is equal to the number N of distinct training samples, $\tilde{N} = N$, matrix $\mathbf{H}$ is square and invertible when the input weight vectors $\mathbf{w}_i$ and the hidden biases b_i are randomly chosen, and SLFNs can approximate these training samples with zero error.

However, in most cases the number of hidden nodes is much less than the number of distinct training samples, $\tilde{N} \ll N$, $\mathbf{H}$ is a nonsquare matrix and there may not exist $\mathbf{w}_i, b_i, \beta_i$ $(i = 1, \cdots, \tilde{N})$ such that $\mathbf{H}\beta = \mathbf{T}$. According to Theorem 3.2.1, the smallest norm least-squares solution of the above linear system is:

$$\hat{\beta} = \mathbf{H}^{\dagger}\mathbf{T} \tag{3.19}$$

where $\mathbf{H}^{\dagger}$ is the *Moore-Penrose generalized inverse* of matrix $\mathbf{H}$ [81, 82].

Remarks 1: As discussed in Appendix, we have the following important

properties:

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

$$\|\mathbf{H}\hat{\beta} - \mathbf{T}\| = \|\mathbf{H}\mathbf{H}^{\dagger}\mathbf{T} - \mathbf{T}\| = \min_{\beta} \|\mathbf{H}\beta - \mathbf{T}\| \tag{3.20}$$

   Although almost all learning algorithms attempt to reach the minimum training error, however, this is not easily achievable because of local minimum or very long training time.

2. **Smallest normed weights have best generalization performance**. Furthermore, the special solution $\hat{\beta} = \mathbf{H}^{\dagger}\mathbf{T}$ has the smallest norm among all the least-squares solutions of $\mathbf{H}\beta = \mathbf{T}$:

$$\|\hat{\beta}\| = \|\mathbf{H}^{\dagger}\mathbf{T}\| \leqslant \|\beta\|, \forall \beta \in \{\beta : \|\mathbf{H}\beta - \mathbf{T}\| \leqslant \|\mathbf{H}\mathbf{z} - \mathbf{T}\|, \forall \mathbf{z} \in \mathbf{R}^{\tilde{N} \times N}\} \tag{3.21}$$

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

3.4.3 Proposed Learning Algorithm for Generic SLFNs

Thus, a simple learning method for SLFNs called extreme learning machine (ELM) can be summarized as follows:

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

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

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

step 3 Calculate the output weight β

$$\beta = \mathbf{H}^{\dagger}\mathbf{T} \tag{3.22}$$

where $\mathbf{T} = [\mathbf{t}_1 \cdots \mathbf{t}_N]^T$

Remark 1: As shown in Theorem 3.3.1, this algorithm works for any infinitely differential activation function $g(x)$. Such activation functions include the sigmoidal functions as well as the radial basis, sine, cosine, exponential, and many non-regular functions as shown in Huang and Babri[16]. According to remark of Theorem 3.3.1, the upper bound of the required number of hidden nodes is the number of distinct training samples, that is $\tilde{N} \leqslant N$.

Remark 2: Several works [16, 18, 78, 84, 15] have shown that SLFNs with N hidden nodes can exactly learn N distinct observations. Tamura and Tateishi [78] and Huang [18, 84] rigorously prove that SLFNs (with N hidden nodes) with *randomly chosen sigmoidal hidden nodes* (with both input weights and hidden biases randomly generated) can exactly learn N distinct observations. Huang, et al[16, 84] also rigorously proves that if input weights and hidden

biases are allowed to be tuned (as done in most traditional implementations) SLFNs with at most N hidden nodes and with almost any nonlinear activation function can exactly learn N distinct observations and these activation functions include differentiable and nondifferentiable functions, continuous and noncontinuous functions, etc.

This chapter rigorously proves that for any infinitely differentiable activation function SLFNs with N hidden nodes can learn N distinct samples exactly and SLFNs may require less than N hidden nodes if learning error is allowed (the empirical analysis is provided in Section 3.5.2). Different from previous works [16, 18, 78, 84, 15] and the ELM algorithm introduced in this chapter, Ferrari and Stengel [15] shows that SLFNs with N *sigmoidal* hidden nodes and *with input weights randomly generated but hidden biases appropriately tuned* can exactly learn N distinct observations. Hidden nodes are not randomly generated in the work done by Ferrari and Stengel [15], although the input weights are randomly generated, the hidden biases need to be determined based on the input weights and input training data (cf. equation (18) of [15]). Because the hidden biases depends on the input weights and input data, as stated by Ferrari and Stengel [15], "the rigorous proof (of their algorithm) cannot be provided". However, since both input weights and hidden biases are randomly generated in ELM, the validity of ELM has been proved as in Theorem 3.3.1.

Remark 3: Modular networks have also been suggested in several works [78,

18, 84, 15], which partition the training samples into L subsets each learned by an SLFN separately. Suppose that the number of hidden nodes in i-th SLFN is s_i. For the methods proposed by Tamura and Tateishi [78] and Huang [18, 84], random hidden nodes (with randomly generated input weights and biases) are used in each SLFN and these SLFNs can actually share common hidden nodes. That means, the i-th hidden node of the first SLFN can also work as the i-th hidden node of the rest SLFNs and the total number of hidden nodes required in these L SLFNs is still $\max_i(s_i)$. Although Ferrari and Stengel [15] also randomly generates input weights for these sub-SLFNs, but the hidden biases of these sub-SLFNs need to tuned based on the input weights and input training data. Thus, the hidden biases of these SLFNs are different, which means these SLFNs cannot share the common hidden nodes and the total number of hidden nodes required in modular network implementation of Ferrari and Stengel [15] is $\sum_{i=1}^{L} s_i \gg \max_i(s_i)$. One can refer to Tamura and Tateishi [78] and Huang [18, 84] for details of modular network implementation.

Remark 4: Several methods can be used to calculate the Moore-Penrose generalized inverse of $\mathbf{H}$. These methods may include but are not limited to orthogonal projection, orthogonalization method, iterative method, and Singular Value Decomposition (SVD) [85, 82]. The orthogonalization method and iterative method have their limitations since searching and iteration are used which we wish to avoid in ELM. The orthogonal project method can

be used when $\mathbf{H}^T\mathbf{H}$ is nonsingular and $\mathbf{H}^\dagger = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T$ which is also used in Ferrari and Stengel [15]. However, $\mathbf{H}^T\mathbf{H}$ may not always be nonsingular or may tend to be singular in some applications and thus orthogonal projection method may not perform well in all applications. The Singular Value Decomposition (SVD) can be generally used to calculate the Moore-Penrose generalized inverse of $\mathbf{H}$ in all cases.

Remark 5: If SVD is used, the Moore-Penrose generalized inverse of matrix $\mathbf{H} \in \mathbf{R}^{N\times\tilde{N}}(N \geqslant \tilde{N})$ can be calculated as:

$$\mathbf{H}^\dagger = \mathbf{V}^T\mathbf{W}^{-1}\mathbf{U} \tag{3.23}$$

where $\mathbf{V}$, $\mathbf{W}$ and $\mathbf{U}$ is obtained from SVD of $\mathbf{H}$ such that

$$\mathbf{H} = \mathbf{U}\mathbf{W}\mathbf{V}^T \tag{3.24}$$

The matrix operations involved in the calculation of Moore-Penrose generalized inverse include SVD, transpose, multiplication and inverse of diagonal matrix $\mathbf{W}$. Computational complexity of the multiplication between the matrices is $O(\tilde{N}^3 + \tilde{N}^2N)$ without using any optimal algorithms, and the complexity to transpose matrix $\mathbf{V}$ is $O(\tilde{N}^2)$. As matrix $\mathbf{W}$ is diagonal, the complexity to invert it is only $O(\tilde{N})$. By using some optimal SVD algorithms [86, 87], the computational complexity of SVD can be as low as $O(\tilde{N}^3)$. Thus, the total computational complexity of the Moore-Penrose generalized inverse is $\mathbf{H}^\dagger$ is $O(\tilde{N}^3 + \tilde{N}^2N) + O(\tilde{N}^2) + O(\tilde{N}) + O(\tilde{N}^3)$ equivalent to $O(2\tilde{N}^3 + \tilde{N}^2N + \tilde{N}^2 + \tilde{N})$ i.e. $O(\tilde{N}^2N)$.

If H is a square matrix with a rank of N, the computational complexity to calculate its inverse is $O(N^3)$ if Gaussian Elimination is used. As in most time, $\tilde{N}$ is much smaller than N, the total computational complexity of the Moore-Penrose generalized inverse is lower than that of normal inverse. In batch learning applications where the number of training samples N is fixed, the learning computational complexity of the ELM increase quadratically with the number of hidden neurons $\tilde{N}$.

3.5 Performance Evaluation

In this section, the performance of the proposed ELM learning algorithm[2] is compared with the popular algorithms of feedforward neural networks like the conventional back-propagation (BP) algorithm and support vector machines (SVMs), least square support vector machine (LS-SVM) and radial basis function (RBF) networks on quite a few benchmark real problems in the function approximation and classification areas. All the simulations for the BP, RBF and ELM algorithms are carried out in MATLAB 6.5 environment running in a Pentium 4, 3 GHZ CPU. Although there are many variants of BP algorithm, the Levenberg-Marquardt learning algorithm is used in our simulations. As mentioned in Chapter 2 and tested on many benchmarking applications among all traditional BP learning algorithms,

[2]ELM Source Codes: http://www.ntu.edu.sg/home/egbhuang/

the Levenberg-Marquardt algorithm appears to be the fastest method for training moderate-sized feedforward neural networks (up to several hundred weights). There is a very efficient implementation of Levenberg-Marquardt algorithm provided by MATLAB package, which has been used in our simulations for BP. The RBF algorithm used is also provided in the MATLAB Neural Networks Toolbox [88]. This algorithm can gradually and selectively add hidden neurons and determine their centers to meet the given goal of error. As the RBF is used for classification problem only, the goal of error is set to 0.1 in all the experiments. The simulations for SVM are carried out using compiled C-coded SVM packages: LIBSVM[3] running in the same PC. This C-coded SVM packages is an implementation based three key works of fast SVMs: original SMO by Platt[89], SMO's modification by Keerthi et al[90], and SVM^{Light} by Joachims[91]. Thus it is one of the fastest SVM libraries. The kernel function used in SVM is radial basis function whereas the activation function used in our proposed algorithms is a simple sigmoidal function $g(x) = 1/(1+\exp(-x))$. In our experiments, all the inputs (attributes) have been normalized into the range $[0, 1]$ while the outputs (targets) have been normalized into [-1, 1]. As seen from ELM algorithm, the learning time of ELM is mainly spent on calculating the Moore-Penrose generalized inverse $\mathbf{H}^{\dagger}$ of the hidden layer output matrix $\mathbf{H}$. For BP and ELM, the number of hidden nodes are gradually increased by an interval of 5 and the nearly optimal number of nodes for BP and ELM are then selected based on cross-

[3]SVM Source Codes: http://www.csie.ntu.edu.tw/∼cjlin/libsvm/

validation method. The experimental settings throughout the thesis are the same as above unless otherwise stated.

3.5.1 Benchmarking with Regression Problems

Artificial Case: Approximation of 'SinC' Function with Noise

In this example, all the three algorithms (ELM, BP and SVR) are used to approximate the 'SinC' function, a popular choice to illustrate SVR (support vector machine for regression) in the literature:

$$y(x) = \begin{cases} \sin(x)/x, & x \neq 0 \\ 1, & x = 0 \end{cases} \tag{3.25}$$

A training set (x_i, y_i) and testing set (x_i, y_i) with 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.

There are 20 hidden nodes assigned for our ELM algorithm and BP algorithm. 50 trials have been conducted for all the algorithms and the average results and standard deviations (Dev) are shown in Table 3.1. It can be seen from Table 3.1 that ELM learning algorithm spent 0.125 seconds CPU time obtaining the testing root mean square error (RMSE) 0.0097, however, it

Algorithms	Training Time (seconds)	Training		Testing		# SVs/ nodes
		RMS	Dev	RMS	Dev	
ELM	0.125	0.1148	0.0037	0.0097	0.0028	20
BP	21.26	0.1196	0.0042	0.0159	0.0041	20
SVR	1273.4	0.1149	0.0007	0.0130	0.0012	2499.9

Table 3.1: Performance comparison for learning noise free function: SinC.

takes 21.26 seconds CPU time for BP algorithm to reach a much higher testing error 0.0159. The new ELM runs 170 times faster than the conventional BP algorithms. We also compare the performance of SVR (support vector machine for regression) and our ELM algorithm. The parameter C is tuned and set as $C = 100$ in SVR algorithm. Compared with SVR, the reduction for our ELM algorithm in CPU time is also above 10,000 times, even though the fact that C executable may be faster than MATLAB environment has not be taken into account.

Figure 3.1 shows the true and the approximated function of the ELM learning algorithm. Figure 3.2 shows the true and the approximated function of the BP and SVM learning algorithms.

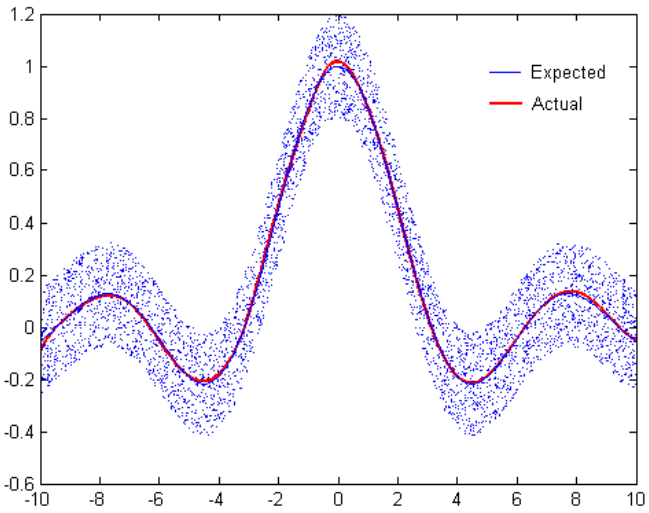
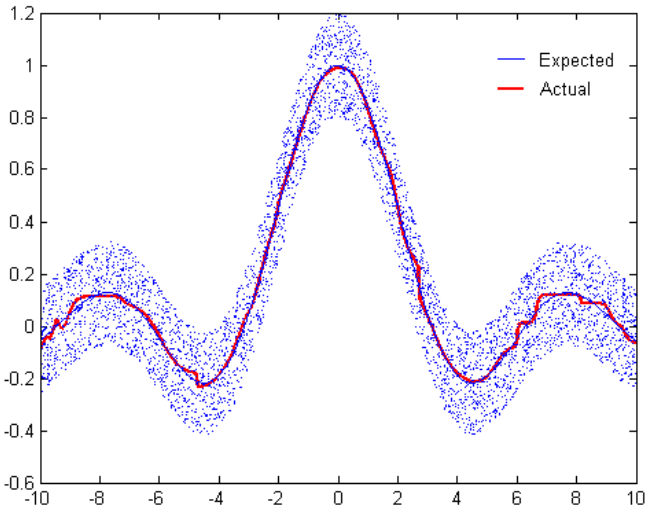
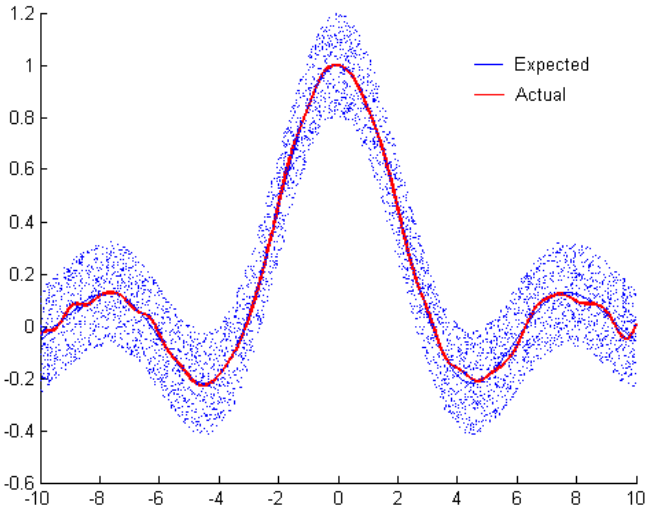


Figure 3.1: Outputs of the ELM learning algorithm.



(a) BP



(b) SVR

Figure 3.2: Outputs of the BP and SVR learning algorithms.

Real-World Regression Problems

Name	# Observations		# Attributes
	Training	Testing	
Abalone	2000	2177	8
Delta Ailerons	3000	4129	6
Delta Elevators	4000	5517	6
Pole Telecom	5000	10000	48
Computer Activity	4000	4192	8
Census	10000	12784	16
Auto Price	80	79	15
Boston Housing	250	256	13
Pyrimidines	40	34	27
Triazines	100	86	60
Machine CPU	100	109	6
Servo	80	87	4
Breast Cancer	100	94	32
Stocks Domain	450	500	10

Table 3.2: Specification of benchmark real world regression datasets

The performance of ELM, BP and support vector regression (SVR) are compared on numerous real-world benchmarking regression datasets[4] cover-

[4]http://www.niaad.liacc.up.pt/~ltorgo/Regression/ds_menu.html

Datasets	BP		ELM	
	Training	Testing	Training	Testing
Abalone	0.0785	0.0874	0.0803	0.0824
Delta Ailerons	0.0409	0.0481	0.0423	0.0431
Delta Elevators	0.0544	0.0592	0.0550	0.0568
Pole Telecom	0.0738	0.1192	0.0717	0.0917
Computer Activity	0.0273	0.0409	0.0316	0.0382
Census	0.0596	0.0685	0.0624	0.0660
Auto Price	0.0443	0.1157	0.0754	0.0994
Boston Housing	0.1049	0.1173	0.0927	0.1071
Pyrimidines	0.1339	0.1523	0.1243	0.1474
Triazines	0.1438	0.2197	0.1897	0.2002
Machine CPU	0.0352	0.0826	0.0332	0.0539
Servo	0.0794	0.1276	0.0707	0.1196
Breast Cancer	0.2788	0.3155	0.2470	0.2679
Stocks domain	0.0179	0.0358	0.0251	0.0348

Table 3.3: Comparison of training and testing RMSE of BP and ELM.

ing various fields. The specifications of the datasets are listed in Table 3.2. As in the real applications, the distributions of these dataset are unknown and most of them are not noisy-free. For each case, the training dataset and testing dataset are randomly generated from its whole dataset before each

trial of simulation.

Datasets	SVR		ELM	
	Training	Testing	Training	Testing
Abalone	0.0759	0.0784	0.0803	0.0824
Delta Ailerons	0.0418	0.0429	0.04230	0.0431
Delta Elevators	0.0534	0.0540	0.0545	0.0568
Pole Telecom	0.0627	0.896	0.0717	0.0917
Computer Activity	0.0464	0.0470	0.0316	0.0382
Census	0.0718	0.0746	0.0624	0.0660
Auto Price	0.0652	0.0937	0.0754	0.0994
Boston Housing	0.0936	0.1092	0.0927	0.1071
Pyrimidines	0.1285	0.1498	0.1243	0.1474
Triazines	0.1432	0.1829	0.1897	0.2002
Machine CPU	0.0574	0.0811	0.0332	0.0539
Servo	0.0840	0.1177	0.0707	0.1196
Breast Cancer	0.2278	0.2643	0.2470	0.2679
Stocks domain	0.0503	0.0518	0.0251	0.0348

Table 3.4: Comparison of training and testing RMSE of SVR and ELM.

In addition, early-stopping criteria is used for BP as well. Average results of 50 trials of simulations for each fixed size of SLFN are obtained and then finally the best performance obtained by BP and ELM are reported here.

Datasets	BP	SVR		ELM
	# nodes	(C, γ)	# SVs	# nodes
Abalone	10	$(2^4, 2^{-6})$	309.84	25
Delta Ailerons	10	$(2^3, 2^{-3})$	82.44	45
Delta Elevators	5	$(2^0, 2^{-2})$	260.38	125
Pole Telecom	5	$(2^0, 2^{-1})$	57.28	110
Computer Activity	45	$(2^5, 2^{-5})$	64.2	125
Census	10	$(2^1, 2^{-1})$	810.24	160
Auto Price	5	$(2^8, 2^{-5})$	21.25	15
Boston Housing	10	$(2^8, 2^{-5})$	36.04	20
Pyrimidines	5	$(2^2, 2^{-3})$	10.28	5
Triazines	5	$(2^{-1}, 2^{-9})$	48.42	10
Machine CPU	10	$(2^6, 2^{-4})$	7.8	10
Servo	10	$(2^2, 2^{-2})$	22.375	30
Breast Cancer	5	$(2^{-1}, 2^{-4})$	74.3	10
Stocks domain	20	$(2^3, 2^{-9})$	19.94	110

Table 3.5: Comparison of network complexity of BP, SVR and ELM.

As proposed by Hsu and Lin[92], for each problem, we estimate the generalized accuracy using different combination of cost parameters C and kernel parameters γ: $C = [2^{12}, 2^{11}, \cdots, 2^{-1}, 2^{-2}]$ and $\gamma = [2^4, 2^3, \cdots, 2^{-9}, 2^{-10}]$. Therefore, for each problem we try $15 \times 15 = 225$ combinations of para-

Datasets	BP (seconds)	SVR (seconds)	ELM (seconds)
	Training	Training	Training
Abalone	1.7562	1.6123	0.0125
Delta Ailerons	2.7525	0.6726	0.0591
Delta Elevators	1.1938	1.121	0.2812
Pole Telecom	19.921	10.738	0.6284
Computer Activity	67.44	1.0149	0.2951
Census	8.0647	11.251	1.0795
Auto Price	0.2456	0.0042	0.0016
Boston Housing	0.5281	0.3218	0.0072
Pyrimidines	0.1937	0.0171	$< 10^{-4}$
Triazines	0.5484	0.0086	$< 10^{-4}$
Machine CPU	0.2354	0.0018	0.0015
Servo	0.2447	0.0045	$< 10^{-4}$
Breast Cancer	0.3856	0.0064	$< 10^{-4}$
Stocks domain	1.0487	0.0690	0.0172

Table 3.6: Comparison of training time of BP, SVR and ELM.

meters (C, γ) for SVR. Average results of 50 trials of simulations with each combination of (C, γ) are obtained and the best performance obtained by SVR are shown in this section as well.

As observed from Tables 3.3 and 3.4, general speaking, ELM and SVR

Datasets	BP		SVR		ELM	
	Training	Testing	Training	Testing	Training	Testing
Abalone	0.0011	0.0034	0.0015	0.0013	0.0049	0.0058
Delta Ailerons	0.0015	0.0015	0.0012	0.0010	0.0030	0.0035
Delta Elevators	0.0007	0.0003	0.0006	0.0005	0.0028	0.0029
Pole Telecom	0.0008	0.0015	0.0009	0.0012	0.0010	0.0011
Computer Activity	0.0007	0.0007	0.0015	0.0016	0.0005	0.0033
Census (House8L)	0.0011	0.0050	0.0013	0.0013	0.001	0.0017
Auto Price	0.0405	0.0231	0.0025	0.0040	0.0119	0.0119
Boston Housing	0.0006	0.0004	0.0005	0.0008	0.0006	0.0009
Pyrimidines	0.0045	0.0026	0.0012	0.0011	0.0021	0.0033
Triazines	0.0656	0.0531	0.0152	0.0163	0.0212	0.0209
Machine CPU	0.0192	0.0715	0.0083	0.0180	0.0060	0.0156
Servo	0.0313	0.0475	0.0406	0.0185	0.0121	0.0113
Breast Cancer	0.1529	0.0962	0.0115	0.0151	0.0121	0.0167
Stocks domain	0.0012	0.0022	0.0016	0.0022	0.0011	0.0016

Table 3.7: Comparison of the standard deviation of training and testing RMSE of BP, SVR and ELM.

obtain similar generalization performance, which is slightly higher than BP's in many cases. If the difference of the two testing RMSE obtained by two algorithms is larger than 0.005 for a case, the winner's testing RMSE will be

shown in boldface in Tables 3.3 and 3.4. As observed from Table 3.5, ELM needs more hidden nodes than BP but it is more compact than SVR in most cases. The advantage of the ELM on training time is quite obvious. Table 3.7 shows the average standard deviations of training and testing RMSE of the BP, SVR and ELM.

Figure 3.3 shows the relationship between the generalization performance of ELM and its network size. As observed from Figure 3.3, the generalization performance of ELM is very stable on a wide range of number of hidden nodes although the generalization performance tends to become worse when too few or too many nodes are randomly generated. This result suggests that unlike other algorithms such as SVM it is not difficult to set a proper value to the parameter $\tilde{N}$ for the ELM. It is possible for the ELM to achieve a decent performance with a few trials of different network sizes.

3.5.2 Benchmarking with Small and Medium Real Classification Applications

Medical Diagnosis Application: Diabetes

The performance comparison of the new proposed ELM algorithm and many other popular algorithms has been conducted for a real medical diagnosis problem: Diabetes[5], using the "Pima Indians Diabetes Database" produced

[5]ftp://ftp.ira.uka.de/pub/node/proben1.tar.gz

in the Applied Physics Laboratory, Johns Hopkins University, 1988. The diagnostic, binary-valued variable investigated is whether the patient shows signs of diabetes according to World Health Organization criteria (i.e., if the 2 hour post-load plasma glucose was at least 200 mg/dl at any survey examination or if found during routine medical care). The database consists of 768 women over the age of 21 resident in Phoenix, Arizona. All examples belong to either positive or negative class. All the input values are within $[0, 1]$. For this problem, as usually done in literature[93, 94, 95, 96] 75% and 25% samples are randomly chosen for training and testing at each trial, respectively. The parameter C of SVM algorithm is tuned and set as: $C = 10$ and the rest parameters are set as default. Besides the ELM, BP and SVM, the performance of a SVM variant namely the least square support vector machine (LS-SVM) is also shown and compared. The LS-SVMlab[6] is used to conduct the experiments of LS-SVM.

50 trials have been conducted for all the algorithms except 10 trials for RBF for its long training time and the average results are shown in Table 3.8. Seen from Table 3.8, in our simulations SVM can reach the testing rate 77.31% with 317.16 support vectors at average. R*ä*tsch et al[93] obtained a testing rate 76.50% for SVM which is slight lower than the SVM result we obtained. The testing accuracy achieved by LS-SVM is lower than SVM and needs more support vectors which are as many as the training samples. The

[6]The library is available at http://www.esat.kuleuven.ac.be/sista/lssvmlab/

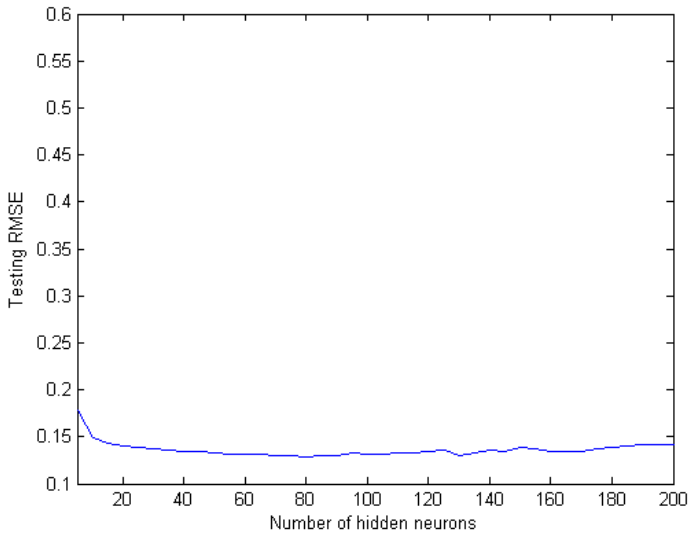


Figure 3.3: The generalization performance of ELM is stable on a wide range of number of hidden nodes.

Algorithms	Training Time (seconds)	Success Rate (%)				# SVs/ nodes
		Training		Testing		
		Rate	Dev	Rate	Dev	
ELM	0.0118	78.68	1.18	77.57	2.85	20
BP	3.0116	86.63	1.7	74.73	3.2	20
SVM	0.1860	78.76	0.91	77.31	2.35	317.16
RBF	123.23	77.48	0.84	76.54	1.03	75.2
LS-SVM	1.8590	86.81	0.74	72.75	1.29	576

Table 3.8: Performance comparison in real medical diagnosis Application: Diabetes.

Algorithms	Testing Rate (%)
ELM	77.57
SVM[93]	76.50
SAOCIF[94]	77.32
Cascade-Correlation[94]	76.58
AdaBoost[95]	75.60
C4.5[95]	71.60
RBF[96]	76.30
Heterogeneous RBF[96]	76.30

Table 3.9: Performance comparison in real medical diagnosis Application: Diabetes.

RBF achieves 76.54% in testing. Because it increases the hidden neurons iteratively, its training time is much longer than the other learning algorithms. In addition, it requires more hidden neurons than the multilayer feedforward neural networks (ELM and BP). In our experiments, the new ELM learning algorithm can achieve the average testing rate 77.57% with 20 nodes, which is obviously higher than all the results so far reported in literature using various popular algorithms such as SVM[93], SAOCIF[94], Cascade-Correlation algorithm[94], bagging and boosting methods[95], C4.5[95], and RBF[96] (cf. Table 3.9). BP algorithm performs very poor in our simulations for this case. It can also be seen that the ELM learning algorithm run around 300 times faster than BP, and 15 times faster than SVM for this small problem with-

out considering that C executable environment may run much faster than MATLAB environment.

Medium Size Classification Applications

The ELM performance has also been tested on the Banana database[7] and some other multi-class databases from the Statlog collection[97]: Landsat Satellite Image (SatImage), Image Segmentation (Segment) and Shuttle Landing Control Database. The information of the number of data, attributes and classes of these applications is listed in Table 3.10.

Problems	# Training Samples	# Testing Samples	# Attributes	# Classes
Satellite Image	4400	2000	36	7
Image Segmentation	1500	810	18	7
Shuttle	43500	14500	9	7
Banana	5200	490000	2	2

Table 3.10: Information of the benchmark problems: Landsat Satellite Image, Image Segmentation, Shuttle Landing Control Database, and Banana.

The setting of the experiments is the same as the experiments on Diabetes dataset. For each trial of simulation of SatImage and Segment, the training data set and testing data set are randomly generated from their

[7]http://www.first.gmd.de/~raetsch/data

Problems	Algorithms	Training Time (seconds)	Success Rate (%)				# nodes
			Training		Testing		
			Rate	Dev	Rate	Dev	
Satellite Image	ELM	14.92	93.52	1.46	89.04	1.50	500
	BP	12561	95.26	0.97	82.34	1.25	100
	RBF	5374.1	90.54	0.61	87.93	1.03	439.2
	SVM	2010.4	91.61	0.58	88.47	1.26	1028.5
	LS-SVM	680.84	85.34	0.53	85.88	0.98	4400
Image Segment	ELM	1.40	97.35	0.32	95.01	0.78	200
	BP	4745.7	96.92	0.45	86.27	1.80	100
	RBF	1093.3	96.29	0.25	93.16	0.63	229.6
	SVM	753.9	90.12	0.32	87.51	0.84	394.2
	LS-SVM	34.74	90.66	0.23	90.18	0.86	1500
Shuttle	ELM	5.740	99.65	0.12	99.40	0.12	50
	BP	6132.2	99.77	0.10	99.27	0.13	50
	RBF	8019.3	99.64	0.07	98.93	0.18	3982.8
	SVM	1849.2	99.77	0.03	99.51	0.13	8310.2
	LS-SVM	793	99.31	0.08	99.33	0.12	43500
Banana	ELM	2.19	92.36	0.17	91.57	0.25	100
	BP	6132.2	90.26	0.27	88.09	0.70	100
	RBF	1083.7	89.32	0.34	87.93	0.83	592.4
	SVM	938.3	92.47	0.29	91.02	0.67	895.6
	LS-SVM	380.5	90.98	0.25	89.12	0.36	5200

Table 3.11: Performance comparison in more benchmarking applications: Satimage, Segment, Shuttle, and Banana.

overall database. The training and testing data sets are fixed for Shuttle and Banana applications, as usually done in literature. Although there are 40,000 training data in the original Banana database, during our simulations it is found that there are only 5,200 distinct training samples and the redundancy of the training data has been removed. 50 trials have been conducted for the ELM and LS-SVM algorithms, and 5 trials for BP, SVM and RBF since it obviously takes very long time to train SLFNs using BP for these several cases.

As shown in Table 3.11, unsurprisingly ELM has the shortest training time among the all five learning algorithms and LS-SVM is the second faster learning algorithms. Both ELM and LS-SVM are using least square methods which effectively shorten the learning time. BP is the slowest on all the four problems. ELM wins on three datasets i.e. Satellite Image, Image Segment and Banana in terms of generalization performance, while SVM achieves the best generalization result on Shuttle problem which is slightly better than ELM. BP requires less storage (neurons) than the other algorithms, while kernel methods namely SVM and LS-SVM requires more storage (support vectors) than neural network learning methods. LS-SVM is the worst in terms of storage as it treats all the training samples as support vectors. Although RBF selectively choose hidden neurons, it still generally needs more hidden neurons than multilayer feedforward networks with sigmoid hidden node.

3.5.3 Benchmarking with Real-World Very Large Complex Applications

We have also tested the performance of our ELM algorithm for very large complex applications such as the Forest Cover Type Prediction.

Forest Cover Type Prediction is an extremely large complex classification problem with seven classes. The forest cover type[97] for 30 x 30 meter cells was obtained from US Forest Service (USFS) Region 2 Resource Information System (RIS) data. There are 581,012 instances (observations) and 54 attributes per observation. In order to compare with the previous work[98], it was modified as a binary classification problem where the goal was to separate class 2 from the other six classes. There are 100,000 training data and 481,012 testing data. The parameters for the SVM are $C = 10$ and $\gamma = 2$.

50 trials have been conducted for the ELM algorithm, and 1 trial for SVM since it takes very long time to train SVM for this large complex case[8]. Seen from Table 3.12, the proposed ELM learning algorithm obtains better generalization performance than SVM learning algorithm. However, the proposed ELM learning algorithm only spent 1.5 minutes on learning while SVM need to spend nearly 12 hours on training. The learning speed has

[8]Actually we have tested SVM for this case many times and always obtained similar results as presented here.

Algorithms	Time (minutes)		Success Rate (%)		# SVs/ nodes
	Training	Testing	Training	Testing	
ELM	1.6148	0.7195	92.35	90.21	200
ELM	3.1391	0.9326	93.59	92.55	400
SLFN[98]	12	N/A	82.44	81.85	100
SVM	693.6000	347.7833	91.70	89.90	31,806

Table 3.12: Performance comparison of the ELM, BP and SVM learning algorithms in Forest Type Prediction application.

dramatically been increased 430 times. On the other hand, since the support vectors obtained by SVM is much larger than the required hidden nodes in ELM, the testing time spent SVMs for this large testing data set is more than 480 times than the ELM. It takes more than 5.5 hours for the SVM to react to the 481,012 testing samples. However, it takes only less than 1 minute for the obtained ELM to react to the testing samples. That means, after training and deploying the ELM may react to new observations much faster than SVMs in such a real application. It should be noted that in order to obtain as good performance as possible for SVM, long time effort has been made to run the parameters for SVM. In fact the generalization performance of SVM we obtained in our simulation for this case is much higher than the one reported in literature[98].

We did try to run the efficient optimal BP package provided by MATLAB

for this application, however, it always ran out of memory and also showed that there are too many variables (parameters) required in the simulations, meaning that the application is too large to run in our ordinary PC. On the other hand, it has been reported [98] that SLFNs using gradient-based learning algorithm could only reach a much lower testing accuracy (81.85% only) than our ELM (90.21%).

3.6 Summary

This chapter proposed a simple and efficient learning algorithm for SLFNs called **e**xtreme **l**earning **m**achine (ELM), which has also been rigorously proved. The proposed ELM has several interesting and significant features different from traditional popular gradient-based learning algorithms for feed-forward neural networks:

1. The learning speed of ELM is very fast. In our simulations, the learning phase of ELM can be completed in seconds or fractions of seconds for many applications. Previously, it seems that there exists a virtual speed barrier which most (if not all) classical learning algorithms cannot break through and it is not unusual to take very long time to train a feedforward network using classical learning algorithms even for simple applications.

2. The proposed ELM has better generalization performance than the gradient-based learning such as back-propagation in most cases.

3. The traditional gradient-based learning algorithms may face several issues like local minima, improper learning rate and overfitting, etc. In order to avoid these issues, some methods such as weight decay and early stopping methods may need to be used often in these classical learning algorithms. The ELM avoids these problems. The ELM learning algorithm is much simpler to implement than most learning algorithms for feedforward neural networks.

4. Unlike the traditional gradient-based learning algorithms which only work for differentiable activation functions, as easily observed the ELM learning algorithm could be used to train SLFNs with many non-differentiable activation functions. The performance comparison of SLFNs with different activation functions is beyond the scope of this thesis and shall be investigated in the future work. It may also be interesting to further compare with the ELM for SLFNs and ELM for RBF networks[99, 100] in the future work.

It is worth pointing out that gradient-based learning algorithms like back-propagation can be used for feedforward neural networks which have more than one hidden layers while the proposed ELM algorithm at its present form is still only valid for SLFNs. Fortunately, it has been found in theory

that SLFNs can approximate any continuous function and implement any classification application[7]. The proposed ELM algorithm can be efficiently used in many applications.

According to Barron [65], the lower bound of the error that can be achieved is $1/\tilde{N}^{2/n}$, where n is the dimension of the input. Thus, given a ϵ, to make the error of the ELM smaller than ϵ, it needs at least $\epsilon^{-\frac{n}{2}}$ hidden neurons. Additionally, Theorem 3.3.2 gives an estimation of the upper bound of the error which shows the error is not only determined by the hidden nodes but also the inherent complexity of the datasets. The same problem are faced by most of the existing learning algorithms for multilayer feedforward neural networks such as BP and its variants which have been around for decades, consequently it is very hard to give an universal rule to estimate the number of hidden nodes for a specific learning algorithm in order to meet a given error criteria. This problem becomes more complex with regard to classification problems as the MSE/RMSE is not a very meaningful indication of networks' performance. Hence, normally users determine the number of hidden neurons for a particular problem through their experience or by trial and error. Fortunately, ELM has been empirically shown to have stable performance over a wide range of number of hidden neurons, and we have indicated that the cost of computing the Moore-Penrose generalized inverse is $O(\tilde{N}^2 N)$ which increases quadratically with the number of hidden neurons. A feasible guideline to determine the number of hidden neurons is to have a few

trials. User can first use the closest integer to the $\epsilon^{-\frac{n}{2}}$ as the initial number of hidden neurons, then increase the number by 5, 10 or 20 gradually until the network meet the desired error criteria or the generalization/validation performance starts to decrease. Normally, the number of hidden neuron should be much smaller than the number of training samples to avoid over-fitting. For classification problems, user also can use the error/success rate as the indicator of the performance. Since the learning time of each trial used by the ELM is much shorter than the gradient-based learning algorithm, it is much faster to find a suitable number of hidden neurons for the network resulting in good generalization performance.

Although it is not the objective of the thesis to compare the feedforward neural networks and another popular learning technique - support vector machines (SVM), a simple comparison between the ELM and SVM has also been conducted in our simulations, showing that the ELM may learn faster than SVM by a factor up to thousands. As shown in our simulations especially for the Forest Type Prediction application, the response speed of trained SVM to external new unknown observations is much slower than feedforward neural networks since SVM algorithms normally generate much large number of support vectors (computation units) while feedforward neural networks require very few hidden nodes (computation units) for this application. It is not easy for SVMs to make fast predication in this application since several hours may be spent for such prediction (testing) set, while the ELM appears

to be suitable in applications which request fast prediction and response capability.

Just like the other learning algorithms in the world, ELM is not perfect. In summary, despite of all the advantages that we have mentioned, it has the following limitations:

1. Currently, ELM cannot be used to train feedforward neural networks with more than one hidden layer.

2. Like the other learning algorithms for multilayer feedforward neural networks, it is hard to estimate the proper number of hidden neurons for ELM before training.

3. As shown in the experimental results, ELM needs a little more hidden neurons than the gradient-based learning algorithms on some problems.

In general, the output function of a generic SLFN with $\tilde{N}$ hidden nodes can be represented by

$$f_{\tilde{N}}(\mathbf{X}) = \sum_{i=1}^{\tilde{N}} \beta_i G(\mathbf{a}_i, b_i, \mathbf{x}), \mathbf{x} \in \mathbf{R}^n, \mathbf{a}_i \in \mathbf{R}^n \tag{3.26}$$

where $\mathbf{a}_i$ and b_i are the learning parameters of the hidden nodes and β_i is the weight connecting the i-th hidden node to the output node. $G(\mathbf{a}_i, b_i, \mathbf{x})$ is the output of the i-th hidden node with respect to the input $\mathbf{x}$. When the hidden node is additive with activation function $g(x) : R \to R$,

$$G(\mathbf{a}_i, b_i, \mathbf{x}) = g(\mathbf{a}_i \cdot \mathbf{x} + b_i), b_i \in R \tag{3.27}$$

For RBF hidden node with activation function $g(x) : R \to R$,

$$G(\mathbf{a}_i, b_i, \mathbf{x}) = g(b_i\|\mathbf{x} - \mathbf{a}_i\|), b_i \in R \tag{3.28}$$

The chapter has already shown that the ELM can perform well on SLFNs with additive hidden nodes such as sigmoid units. In fact, Huang, et al [99, 100] showed that the ELM are applicable to SLFNs with RBF hidden nodes as well. Furthermore, the universal approximation capability of the ELM with both additive and RBF hidden nodes has also been thoroughly investigated by Huang, et al [101].

Chapter 4

Training Threshold Networks with ELM

Chapter 3 proposed an algorithm called **e**xtreme **l**earning **m**achine (ELM) for generic SLFNs. In this chapter, we further investigate the performance of the ELM on threshold networks.

4.1 Introduction

Although the neural networks with analog activation functions such as sigmoid and sine in hidden layers have great computational capabilities, the networks with the threshold or hard-limiting activation function in hidden

layers:

$$h(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x < 0 \end{cases} \tag{4.1}$$

are still desirable due to 1)the threshold units are easy for digital hardware implementation[58, 59]; 2)the relationship between the size of the networks using threshold units and the complexity of the training are better understood[62, 63, 80].

However, the widely used gradient-based learning algorithms cannot be used to train the threshold neural networks directly as the threshold functions are non-differentiable. Hence, as introduced in Section 2.3, many research efforts[59, 64, 102] have been made to modify the gradient-based learning methods to be applicable indirectly for networks with threshold units.

Gradient-based learning algorithms are usually slow in learning and may face local minima problem. The validation process (selection of control parameters such as the learning rate, number of hidden neurons and stopping criteria) is complicated and challenging to the users, especially to those having little domain knowledge in the neural networks area. The large computational cost involved in the learning process makes implementing an online learning system on chips rather difficult. Thus, these algorithms are normally trained offline first and then the parameters (weights and biases) of neural networks are transferred to the threshold networks in hardware implementation.

We introduced a novel learning algorithm namely ELM for generic SLFNs in Chapter 3. In this algorithm, the input weights and the biases are randomly generated based on continuous probability distribution (the uniform probability distribution is used in our simulations) and the output weights are then analytically determined. ELM learns much faster than the traditional gradient-based learning algorithms without loss of generalization performance. As indicated by Huang, et al [103], the ELM algorithm will work for the neural networks with threshold units as well. However, a detailed performance study of the ELM for threshold neural networks has not been carried out so far and this chapter attempts to fill this gap. The aim of this chapter is two-fold: 1) to prove in theory that similar to the ELM for sigmoid networks in theory the input weights and biases of the threshold networks can also be assigned randomly based on continuous probability distribution (such as uniform probability distribution used in our simulations) and thus the ELM can be used to train such networks easily without any modifications; 2) to provide a detailed performance evaluation of the ELM for threshold units based on a number of real-world benchmark regression problems. Simulations results show that the ELM for threshold networks achieves better generalization performance than those trained by other gradient-based methods.

4.2 Extreme Learning Machine for Threshold Networks

In the Chapter 3, we have stated that ELM works for SLFNs with different activation functions including threshold functions. However, this has not been shown theoretically. In this section, we prove that in theory ELM can be used to directly train single hidden layer threshold networks.

4.2.1 Approximation Problem of SLFNs

For N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$, where the input $\mathbf{x}_i = [x_{i1}, x_{i2}, \cdots, x_{in}]^T \in \mathbf{R}^n$ and the target $\mathbf{t}_i = [t_{i1}, t_{i2}, \cdots, t_{im}]^T \in \mathbf{R}^m$, standard SLFNs with $\tilde{N}$ hidden neurons and activation function $g(x)$ are mathematically modeled as

$$\sum_{i=1}^{\tilde{N}} \beta_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) = \mathbf{o}_j, \quad j = 1, \cdots, N, \tag{4.2}$$

where $\mathbf{w}_i = [w_{i1}, w_{i2}, \cdots, w_{in}]^T$ is the weight vector connecting the ith hidden neuron and the input neurons, $\beta_i = [\beta_{i1}, \beta_{i2}, \cdots, \beta_{im}]^T$ is the weight vector connecting the ith hidden neuron and the output neurons, and b_i is the bias of the ith hidden neuron. $\mathbf{w}_i \cdot \mathbf{x}_j$ denotes the inner product of $\mathbf{w}_i$ and $\mathbf{x}_j$.

The fact that standard SLFNs with $\tilde{N}$ hidden neurons can approximate these N samples with zero error means that $\sum_{j=1}^{N} \|\mathbf{o}_j - \mathbf{t}_j\| = 0$, i.e., there

exist β_i, $\mathbf{w}_i$ and b_i such that

$$\sum_{i=1}^{\tilde{N}} \beta_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) = \mathbf{t}_j, \quad j = 1, \cdots, N. \tag{4.3}$$

The above N equations can be written compactly as:

$$\mathbf{H}\beta = \mathbf{T} \tag{4.4}$$

where $\mathbf{H}$ is called the hidden layer output matrix of the SLFN[16, 18]; the ith column of $\mathbf{H}$, $[g(\mathbf{w}_i \cdot \mathbf{x}_1 + b_i), \cdots, g(\mathbf{w}_i \cdot \mathbf{x}_N + b_i)]^T$, is the ith hidden neuron output with respect to inputs $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N$. $\beta = [\beta_1, \cdots, \beta_L]^T$ and $\mathbf{T} = [\mathbf{t}_1, \cdots, \mathbf{t}_N]^T$.

4.2.2 ELM Learning Algorithm for Threshold Networks

From Theorem 3.3.1 (can also refer to [18, 78]), we have

Theorem 4.2.1. *For a SLFN with the sigmoid activation function* $g(x) = \frac{1}{1+e^{-\lambda x}}$ *in the hidden layer, given any constant* $\epsilon > 0$*, there always exists an integer* $L \leqslant N$ *such that a SLFN with* L *hidden neurons and with randomly chosen input weights and hidden biases can learn* N *distinct observations with a training error less than* ϵ*.*

Proof. Since Theorem 3.3.1 holds for all the all the infinitely differentiable activation functions including the sigmoid function, the theorem holds. □

We now extend this theorem to threshold networks as given below:

Theorem 4.2.2. *Suppose that threshold activation function* $h(x) = 1_{x \geq 0} + 0_{x<0}$ *is used in the hidden layer. Given any non-zero constant* $\epsilon > 0$ *there always exists an integer* $\tilde{N} \leqslant N$ *such that a SLFN with* $\tilde{N}$ *such hidden neurons and with randomly chosen input weights and hidden biases can learn* N *distinct observations with its training error less than* ϵ.

Proof. Since Theorem 4.2.1 holds for all positive gain parameter λ and $\lim_{\lambda \to +\infty} g(x) = h(x)$, the theorem holds. □

Thus, very interestingly, similar to sigmoid SLFNs [18, 104] the input weights and hidden biases of threshold SLFNs are in fact not necessarily tuned and the hidden layer output matrix $\mathbf{H}$ can actually remain unchanged once random values have been assigned to input parameters and hidden neuron biases in the beginning of learning. According to Theorem 4.2.2, there exists $\tilde{N}$ and randomly generated input weights $\mathbf{w}_i$ and hidden neuron biases b_i $(i = 1, \cdots, \tilde{N})$ such that

$$E = \sum_{j=1}^{N} \left(\sum_{i=1}^{\tilde{N}} \beta_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) - \mathbf{t}_j \right)^2 < \epsilon \tag{4.5}$$

In most cases the number of required hidden neurons is much less than the number of distinct training samples, $\tilde{N} \ll N$, making $\mathbf{H}$ a nonsquare matrix. For fixed input weights $\mathbf{w}_i$ and the hidden layer biases b_i, from equation (4.5), *to train an SLFN* is simply equivalent to finding a least-squares solution $\hat{\beta}$ of the linear system $\mathbf{H}\beta = \mathbf{T}$. The unique smallest norm least-squares solution

of the above linear system is:

$$\hat{\beta} = \mathbf{H}^{\dagger}\mathbf{T} \tag{4.6}$$

where $\mathbf{H}^{\dagger}$ is the Moore-Penrose generalized inverse of matrix $\mathbf{H}$. The learning procedure of ELM algorithm for threshold neural networks can therefore be described as follows:

ELM Algorithm (threshold): Given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i)|\mathbf{x}_i \in \mathbf{R}^n, \mathbf{t}_i \in \mathbf{R}^m\}$, threshold activation function $h(x) = 1_{x\geq 0} + 0_{x<0}$ and number of hidden neurons $\tilde{N}$,

*s*tep 1: Assign random input weight $\mathbf{w}_i$ and bias of hidden neurons b_i, $i = 1, \cdots, L$.

*s*tep 2: Calculate the hidden layer output matrix $\mathbf{H}$.

*s*tep 3: Calculate the output weight β:

$$\beta = \mathbf{H}^{\dagger}\mathbf{T} \tag{4.7}$$

Above algorithm is a specific case of the generic ELM algorithm shown in Chapter 3 (p. 61).

4.2.3 Analysis of Generalization Performance

Unlike the BP learning algorithms, the ELM obtains both the smallest output weights and the smallest training error. As analyzed by Huang, et al

[103], from the viewpoint of *Vapnik-Chervonenkis* (VC) dimension (and hence number of parameters)[80], this method tends to achieve good generalization performance. Gradient-based learning algorithms like back-propagation have the difficulty of reaching the good generalization performance since they only try to obtain the small training errors without considering the magnitude of the weights. (For detailed discussions, refer to [80, 103, 18])

4.3 Performance Evaluation

In many applications threshold networks have been approximated by sigmoid SLFNs with different gain parameters λ and then trained by BP and its variants. In this section, the performance comparison will be conducted between such BP based threshold network training algorithms and ELM algorithm for SLFNs.

4.3.1 Benchmarking with Regression Problems

14 benchmark real life regression problems have been tested. The specifications of the datasets are listed in Table 3.2. 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]. Both the input weights and hidden biases are randomly generated from [-1,1]. The average performance is

obtained based on 50 trials of simulations done for each learning algorithm for each problem, and all datasets are randomly reshuffled into training set and testing set at each trial of simulations.

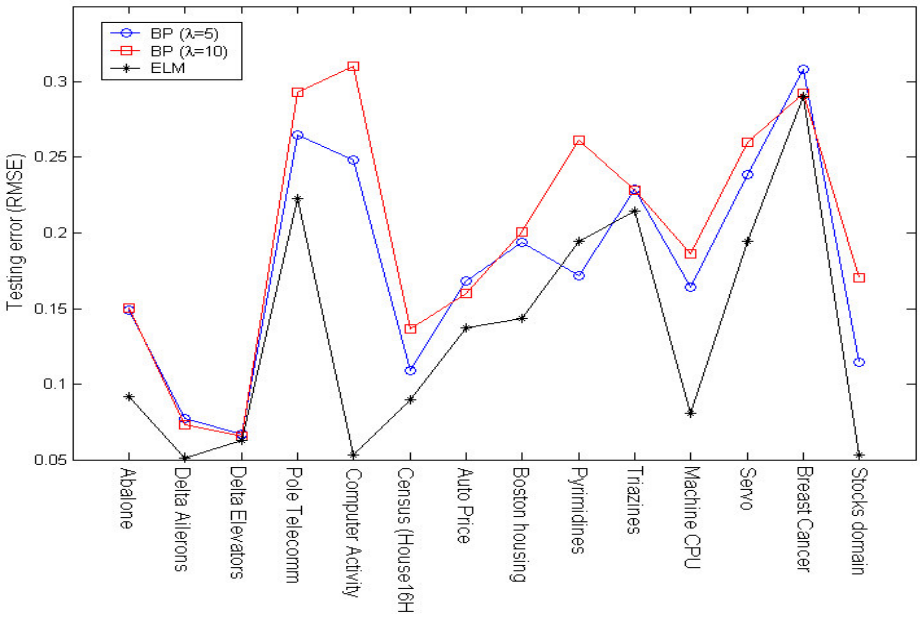


Figure 4.1: Average generalization performance comparison of threshold networks directly trained by ELM algorithm and threshold networks approximated by analog networks trained by BP algorithm.

Since BP algorithm cannot be applied to the threshold activation function directly, similar to Corwin, et al [59] sigmoid SLFNs with different gain parameters λ are used to approximate threshold networks in our experiments. BP and ELM are running in the MATLAB 6.5 (Windows version) environments. The experimental settings are the same as Section 3.5.1.

As shown in Figure 2.2, sigmoid units can approximate threshold units only when the gain parameters λ are set large enough; otherwise one actu-

Dataset	BP ($\lambda = 5$)	BP ($\lambda = 10$)	ELM (threshold)
Abalone	0.14919	0.15017	0.091948
Delta Ailerons	0.077026	0.073125	0.050801
Delta Elevators	0.06706	0.065261	0.0630
Pole Telecom	0.26473	0.29321	0.2225
Computer Activity	0.24818	0.31035	0.0530
Census (House16H)	0.109	0.13631	0.089744
Auto Price	0.16813	0.15976	0.13709
Boston housing	0.1939	0.20077	0.14325
Pyrimidines	0.17165	0.26147	0.19416
Triazines	0.22907	0.22836	0.21426
Machine CPU	0.16445	0.18598	0.08066
Servo	0.23886	0.26012	0.19474
Breast Cancer	0.3085	0.29246	0.29022
Stocks Domain	0.11432	0.17013	0.052997

Table 4.1: Comparison of average generalization performance (average root mean square error (RMSE) of testing) of different learning algorithms.

ally deals with a sigmoid network instead of threshold network. As shown in Figure 4.1 and Table 4.1, ELM for threshold networks usually obtains much better generalization performance than BP learning algorithms. Here the

threshold networks are approximated by sigmoid networks with gain parameters $\lambda = 5$ and $\lambda = 10$ as set in Corwin, et al [59]. We have tried to gradually increase the gain parameter λ and check the relationship between the generalization performance and λ. As observed from Figure 4.2, the generalization performance tends to decrease when the λ is increased[1] Thus, the gradient descent method which gradually transforms the activation functions from sigmoid to threshold may not provide satisfactory performance when threshold units are approximated by sigmoid functions with higher gain parameters λ.

As shown in Table 4.2 the ELM algorithm is very stable and able to obtain much smaller standard deviation of testing errors. As shown in Table 4.3, the ELM algorithm runs much faster than BP. Since the input weights and hidden biases of ELM are randomly generated instead of being fine-tuned, ELM usually needs more neurons than other learning algorithms. For example, if the target function is a sigmoid function: $f(x) = \frac{1}{1+e^{-x}}$, BP with sigmoid activation function $g(x) = \frac{1}{1+e^{-x}}$ and one hidden neuron can approximate this target function after fine-tuning the input weights and hidden bias. Obviously, in general, ELM with one hidden neuron (with random input weights and hidden bias) cannot approximate this target function and ELM may need more neurons (around 10 in this case) so that the combination

[1]For the sake of readability, only a partial of all simulation results are shown in Figure 4.2.

Dataset	BP ($\lambda = 5$)	BP ($\lambda = 10$)	ELM (threshold)
Abalone	0.078597	0.093745	0.007065
Delta Ailerons	0.056256	0.033776	0.0055414
Delta Elevators	0.026206	0.019181	0.005523
Pole Telecom	0.092299	0.11306	0.009235
Computer Activity	0.31722	0.32758	0.006495
Census (House16H)	0.034903	0.082774	0.0016687
Auto Price	0.078444	0.070159	0.024718
Boston Housing	0.1018	0.11585	0.021672
Pyrimidines	0.046733	0.16867	0.061273
Triazines	0.071486	0.052997	0.027469
Machine CPU	0.13201	0.12717	0.026763
Servo	0.11372	0.11216	0.021299
Breast Cancer	0.074661	0.035625	0.019929
Stocks Domain	0.080048	0.15098	0.0038313

Table 4.2: Comparison of standard deviations of testing RMSE of different learning algorithms.

of all these neurons can approximate it.

Dataset	BP ($\lambda = 5$)		BP ($\lambda = 10$)		ELM-threshold	
	time (s)	neurons	time (s)	neurons	time (s)	neurons
Abalone	6.6219	5	7.3251	5	0.24004	80
Delta Ailerons	0.87926	5	0.86224	5	0.35364	80
Delta Elevators	1.2919	5	1.3016	5	0.7771	100
Pole Telecom	8.0454	5	18.242	10	3.7634	300
Computer Activity	4.2516	10	1.8045	5	1.5155	180
Census (House16H)	17.472	10	16.301	10	9.1687	340
Auto Price	0.31904	5	0.3484	5	0.00188	20
Boston Housing	0.55224	10	0.57064	10	0.01726	70
Pyrimidines	0.3456	5	0.43542	10	0.00062	5
Triazines	0.59346	5	0.53716	5	0.00092	20
Machine CPU	0.2872	5	0.2154	5	0.01068	80
Servo	0.28686	5	0.36566	40	0.00156	20
Breast Cancer	0.46524	5	0.4335	5	0.00062	5
Stocks Domain	0.37216	5	0.55336	10	0.3582	200

Table 4.3: A comparison of training time and number of hidden neurons of different learning algorithms

4.3.2 Benchmarking with Classification Problems

Besides the regression problems, three classification problems from the statlog datasets are also tested. The specification of these problems are listed in

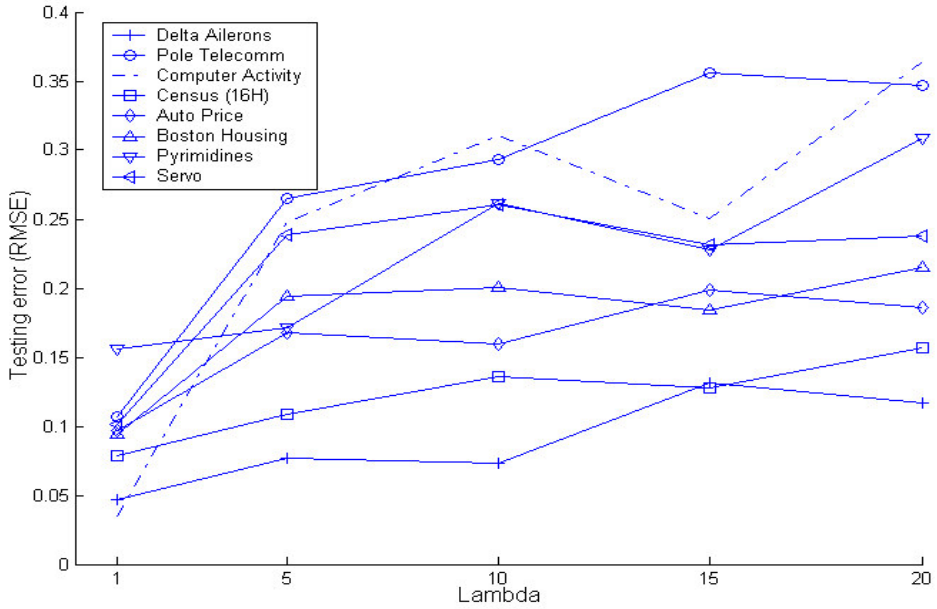


Figure 4.2: The generalization performance of analog approximated threshold networks depends closely on the gain parameter λ.

Table 3.10. The results are listed in Table 4.4. Besides the BP and ELM, the results of two crisp-logic decision tree algorithms C4.5 and CART reported in Michie et al [105] are compared. The C4.5 and CART algorithms achieve better testing accuracy than the other two neural networks algorithms on the Image Segment dataset, while the ELM wins on the rest two datasets in terms of testing accuracy. The ELM shows its fast learning capability again in these classification problems, its training time is shorter than the other three algorithms on all the three datasets and the BP is still the slowest algorithm.

Problems	Algorithms	Training Time (seconds)	Success Rate (%)				# nodes
			Training		Testing		
			Rate	Dev	Rate	Dev	
Satellite Image	ELM	11.64	92.66	1.12	88.45	1.61	500
	BP ($\lambda = 10$)	11697	91.98	1.39	87.06	1.57	100
	C4.5[105]	434.0	96	N/A	85	N/A	
	CART[105]	329.9	92.1	N/A	86.2	N/A	
Image Segment	ELM	1.65	96.09	0.46	94.57	0.85	200
	BP ($\lambda = 10$)	4598.9	80.47	0.37	76.2	0.78	100
	C4.5[105]	142.0	98.7	N/A	96	N/A	
	CART[105]	79.0	99.5	N/A	96	N/A	
Shuttle	ELM	5.485	97.27	0.37	97.3	0.59	50
	BP ($\lambda = 10$)	5823.6	96.2	0.13	95.91	0.19	50
	C4.5[105]	13742.4	96	N/A	90	N/A	
	CART[105]	79.0	96	N/A	92	N/A	

Table 4.4: Performance comparison in more benchmarking applications: Satimage, Segment and Shuttle.

4.4 Summary

It is well-known that compared with analog neural networks threshold networks can reduce the complexity of neural network implementation in hardware. The traditional back-propagation algorithm cannot be applied to such

threshold networks since the required derivatives are not available. Many learning algorithms do not deal with threshold networks directly and instead they use some analog networks to approximate them such that gradient-descent method can finally be used. Similar to conventional BP, these methods may face local minima and over fitting issues. To our knowledge, all these learning algorithms [58, 106, 64, 59, 102] may face difficulty for online and/or hardware implementation due to:

1. besides network size many control variables (manually tuned parameters) are required to be manually selected in advance;

2. learning parameters (connection weights and hidden neuron biases) can be transformed to threshold networks in hardware only after training is completed;

3. all of these algorithms face difficulty when the learning tasks change over time, which does often happen in real applications.

In this chapter, based on the results obtained for sigmoid networks[18, 78], we have shown in Theorem 4.2.2 that the input weights and hidden neurons biases for threshold networks can be randomly assigned instead of greedy tuning. Once the input weights and hidden neurons biases are randomly assigned, unlike the conventional gradient-descent based methods which often get stuck in local minima, ELM tends to reach global minimum directly by using Moore-Penrose generalized inverse method. We further showed that

the ELM learning algorithm can directly be used to train threshold networks with good generalization performance. This has been verified by the simulation results based on a number of real-world benchmark applications. The results also show that ELM for threshold networks takes lesser training time and obtains stable performance. Compared to other learning algorithms, ELM with a given network architecture requires no parameters to be manually tuned and hence is easy to use, especially for ordinary users who do not have domain knowledge in neural networks.

Chapter 5

ELM for Two-Hidden Layer Feedforward Neural Networks

The ELM algorithm introduced in the previous chapters can be used to train SLFNs only. Although many applications can be implemented by both SLFNs and **t**wo-hidden **l**ayer **f**eedforward neural **n**etworks (TLFNs) [7], TLFNs may be more powerful than SLFNs in many cases[18]. In this chapter, the ELM is further extended to train TLFNs.

5.1 Introduction

As discussed in Chapter 2, Huang[18] proved that a novel constructive method that a 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 remarkably reduces the space complexity of a network especially for large scale problems and makes it feasible to implement large scale systems in an ordinary computer. In this method, the observations are divided into several groups, and each group is learned by a *common standard* TLFN. Furthermore, this trained common standard TLFN and some additional neuron quantizers are combined to form a special *non-conventional* TLFN which can finally represent all observations with arbitrarily high accuracy (small error).

During our latest study, the results of some experiments show that the generalization performance of the original constructive method may not be very stable. And the appropriate way to adjust the quantizer factors T and U in Huang's network are not provided yet. In this chapter, an appropriate method to estimate the suitable value for the quantizer factors T and U are proposed. Furthermore, an improved learning algorithm for Huang's network is suggested, which can finish the training procedure with good generalization performance for many applications with low time complexity. Thus, it may be able to provide an alternative approach for some practical applications requiring fast learning capabilities. Extensive experiments on different types of real benchmark problems have been done to systematically evaluate the performance (learning and prediction speeds and generalization performance) of the proposed algorithm. The experimental results show that our proposed algorithm can not only produce good generalization performance but also

have fast learning and prediction capability.

5.2 Review of Huang's Constructive Network Model

Huang's constructive method [18] is briefed in Section 2.4.2. In this section, this constructive method is reviewed in detail.

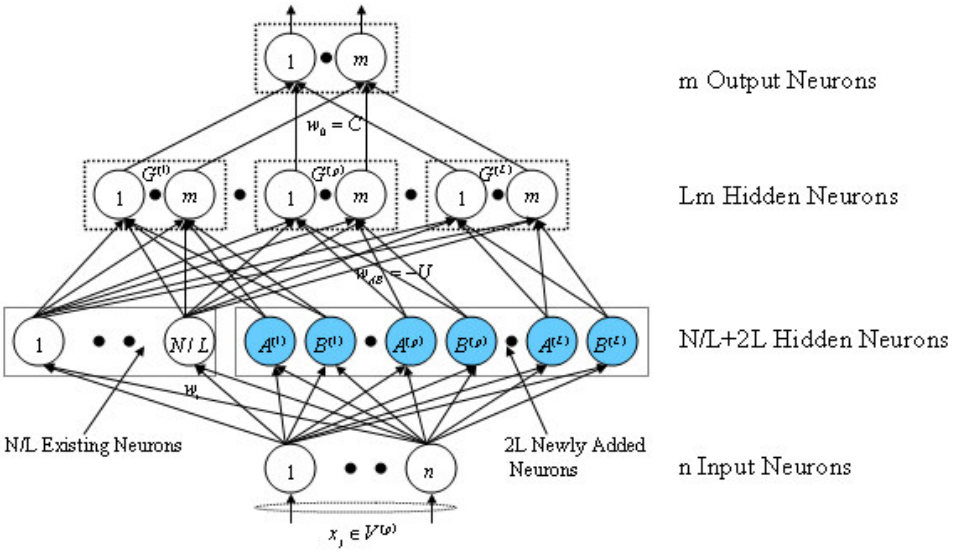


Figure 5.1: Huang's network for multi-output case

A. Basic structure of Huang's network:

It has been shown [18] that Huang's network with sigmoid activation function $g(x) = \frac{1}{1+e^{-x}}$ can approximate N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$, where $\mathbf{x}_i = [x_{i1}, x_{i2}, \cdots, x_{in}]^T \in \mathbf{R}^n$ and $\mathbf{t}_i = [t_{i1}, t_{i2}, \cdots, t_{im}]^T \in \mathbf{R}^m$, with arbitrarily small error. As shown in Figure 5.1, Huang's network has a larger

first hidden layer and narrower second hidden layer. It is a standard **t**wo-hidden **l**ayer **f**eedforward sub-**n**etwork (TLFN) if the L neuron quantizers namely neurons $A^{(p)}$ and $B^{(p)}$ $(p = 1, \cdots, L)$ are removed, where L is a user defined constant indicating how many neuron quantizers are used in the network. There are $\tilde{N}$ $(\tilde{N} = N/L)$ neurons in the first hidden layer and mL neurons in the second hidden layer in the standard sub-TLFN. Each pair of neurons $A^{(p)}$ and $B^{(p)}$ compose the p-th neuron quantizer. The p-th neuron quantizer only links to the p-th neuron group $G^{(p)}$ in the second hidden layer, $p = 1, \cdots, L$, each group $G^{(p)}$ consisting of m neurons. The training of Huang's network mainly consists of two phases: 1) determination of weights and bias of standard sub-TLFN and 2) determination of weights and biases of neural quantizers.

B. Determination of weights and bias of standard sub-TLFN:

One of the major features of Huang's network is that the values for weight $\mathbf{w}_i$ and bias b_i can be randomly choosen, where $\mathbf{w}_i$ denotes the weight linking the i-th neuron in the first hidden layer of the sub-TLFN to all the input neurons and b_i denotes the bias of the i-th neuron in the first hidden layer, $(i = 1, \ldots \tilde{N}, \tilde{N} = N/L)$. Since all the given $\mathbf{x}_i$ are distinctive, a vector $\mathbf{w}$ can be randomly chosen such that $\mathbf{w}\cdot\mathbf{x}_1, \ldots, \mathbf{w}\cdot\mathbf{x}_N$ are all different[18, 16]. Without loss of generality, assume that the index i is reindexed such that

$$\mathbf{w} \cdot \mathbf{x}_1 < \mathbf{w} \cdot \mathbf{x}_2 < \cdots < \mathbf{w} \cdot \mathbf{x}_N. \tag{5.1}$$

For simplicity, consider N to be an integral multiple of user defined positive integer L. The N inputs $\mathbf{x}_i$ are separated into L groups consisting of N/L input vectors each. These L input vector groups are denoted as $V^{(1)}, \ldots, V^{(L)}$,

$$V^{(p)} = \{\mathbf{x}_i | \mathbf{w} \cdot \mathbf{x}_{(p-1)N/L+1} \leqslant \mathbf{w} \cdot \mathbf{x}_i \leqslant \mathbf{w} \cdot \mathbf{x}_{pN/L}\}, \tag{5.2}$$

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

Randomly choose the weight $\mathbf{w}_i$ of the connection linking the input to the i-th neuron of the first hidden layer of the sub-TLFN and the bias b_i of the i-th neuron of first hidden layer of the sub-TLFN, $i = 1, \cdots, \tilde{N}, \tilde{N} = N/L$.

Define matrix $\beta^{(p)}$ as

$$\beta^{(p)} = [\beta_1^{(p)}, \ldots, \beta_{\tilde{N}}^{(p)}]^T, \tag{5.3}$$

where $\beta_i^{(p)}$ denotes the weight vector connecting i-th neuron in the first hidden-layer of the sub-TLFN to the p-th neuron group $G^{(p)}$ (m neurons) in the second hidden layer, $i = 1, \ldots, \tilde{N}, \tilde{N} = N/L$. $\beta^{(p)}$ can be determined as

$$\beta^{(p)} = (\mathbf{H}^{(p)})^{-1}\mathbf{T}^{(p)} \tag{5.4}$$

Matrix $\mathbf{T}^{(p)}$, $p = 1, \ldots, L$, is defined as

$$\begin{aligned} \mathbf{T}^{(p)} &= \mathbf{T}(\mathbf{t}_{(p-1)N/L+1}, \cdots, \mathbf{t}_{pN/L}) \\ &= \begin{bmatrix} \ln\left(\frac{0.5+t_{(p-1)N/L+1,1}/C}{0.5-t_{(p-1)N/L+1,1}/C}\right) & \cdots & \ln\left(\frac{0.5+t_{(p-1)N/L+1,m}/C}{0.5-t_{(p-1)N/L+1,m}/C}\right) \\ \cdots & \cdots & \cdots \\ \ln\left(\frac{0.5+t_{pN/L,1}/C}{0.5-t_{pN/L,1}/C}\right) & \cdots & \ln\left(\frac{0.5+t_{pN/L,m}/C}{0.5-t_{pN/L,m}/C}\right) \end{bmatrix} \end{aligned} \tag{5.5}$$

and matrix $\mathbf{H}^{(p)}$, $p = 1, \ldots, L$, is defined as

$$\begin{aligned}\mathbf{H}^{(p)} &= \mathbf{H}(\mathbf{x}_{(p-1)N/L+1}, \cdots, \mathbf{x}_{pN/L}, b_1, \cdots, b_{\tilde{N}}) \\ &= \begin{bmatrix} g\big(\mathbf{w}_1 \cdot \mathbf{x}_{(p-1)N/L+1} + b_1\big) & \cdots & g\big(\mathbf{w}_{\tilde{N}} \cdot \mathbf{x}_{(p-1)N/L+1} + b_{\tilde{N}}\big) \\ \cdots & \cdots & \cdots \\ g\big(\mathbf{w}_1 \cdot \mathbf{x}_{pN/L} + b_1\big) & \cdots & g\big(\mathbf{w}_{\tilde{N}} \cdot \mathbf{x}_{pN/L} + b_{\tilde{N}}\big) \end{bmatrix}\end{aligned} \tag{5.6}$$

where $\tilde{N} = N/L$ and weight size factor C can be chosen any value to make

$$-0.5 < t_{ij}/C < 0.5. \tag{5.7}$$

All the weights of connections linking to the output layer are set to C, and bias of the output neuron is set as $-0.5C$.

C. Determination of weights and biases of neural quantizers: As shown in Figure 2.4, the weights of connections linking the inputs to neuron $A^{(p)}$ and neuron $B^{(p)}$ can be chosen as $\overline{\mathbf{w}}_{A^{(p)}} = T \cdot \mathbf{w}$ and $\overline{\mathbf{w}}_{B^{(p)}} = -T \cdot \mathbf{w}$, respectively, where parameter T is called a quantizer factor by Huang [18] and should be large enough. That is, all the weights of connections linking the inputs to all A-type neurons can be chosen as $T \cdot \mathbf{w}$ and all the weights of connections linking the inputs to all B-type neurons chosen as $-T \cdot \mathbf{w}$. The biases $\overline{b}_{A^{(p)}}$ and $\overline{b}_{B^{(p)}}$ of neuron $A^{(p)}$ and neuron $B^{(p)}$ are simply analytically calculated as

$$\overline{b}_{A^{(p)}} = -T\Big(\mathbf{w} \cdot \mathbf{x}_{pN/L} + \frac{1}{2} \min_{i,j} |\mathbf{w} \cdot \mathbf{x}_i - \mathbf{w} \cdot \mathbf{x}_j|\Big) \tag{5.8}$$

and

$$\overline{b}_{B^{(p)}} = T\Big(\mathbf{w} \cdot \mathbf{x}_{(p-1)N/L+1} - \frac{1}{2} \min_{i,j} |\mathbf{w} \cdot \mathbf{x}_i - \mathbf{w} \cdot \mathbf{x}_j|\Big). \tag{5.9}$$

Quantizer factor T can be chosen large enough such that for any input $\mathbf{x}_i$ within input vector group $V^{(p)}, p = 1, \ldots, L$, only the outputs of both neurons $A^{(p)}$ and $B^{(p)}$ are almost zero while one of the outputs of neurons $A^{(l)}$and $B^{(l)}$ of each neural quantizer is almost one, where $l \neq p$. In other words, T can be chosen large enough such 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.

All the weights w_{AB} of the connections linking these $2L$ neurons $A^{(p)}$ and $B^{(p)}$ to their corresponding second hidden-layer neurons are chosen to the same value $w_{AB} = -U$, where parameter U is also called a quantizer factor by Huang [18] and can be set large enough such that the input to the p-th neuron group $G^{(p)}$ in the second hidden layer from 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 $V^{(l)}$, $p = 1, \cdots, L$ and $l \neq p$. E_i and ϵ_i can be made arbitrarily large and small, respectively, by setting quantizer factors T and U sufficiently large. E_i goes to negative infinity and ϵ_i goes to zero (cf. Figure 5.2).

Huang's constructive method can be summarized as follows:

step 1: **Sorting and grouping inputs**.

a) Random choose vector $\mathbf{w} \in \mathbf{R}^n$ and reindex i of inputs such that $\mathbf{w} \cdot \mathbf{x}_1 < \mathbf{w} \cdot \mathbf{x}_2 < \cdots < \mathbf{w} \cdot \mathbf{x}_N$.

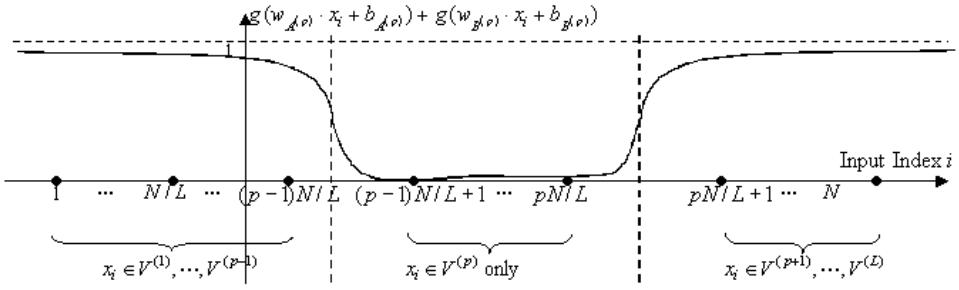


Figure 5.2: Original output of p-th neural quantizer of Huang's network.

b) Group sorted inputs into L groups $V^{(p)}$, $p = 1, \cdots, L$

$$V^{(p)} = \{\mathbf{x}_i | \mathbf{w} \cdot \mathbf{x}_{(p-1)N/L+1} \leqslant \mathbf{w} \cdot \mathbf{x}_i \leqslant \mathbf{w} \cdot \mathbf{x}_{pN/L}\}. \tag{5.10}$$

step 2: **Determination of weights and bias of standard sub-TLFN**.

a) Randomly choose the weights $\mathbf{w}_i$ and biases b_i, $i = 1, \cdots, N/L$.

b) Choose $C = a \max_{i=1,\cdots,N; j=1,\cdots,m} |t_{ij}|$, a can be any positive value larger than 2.

c) Calculate matrix $\beta^{(p)} = [\beta_1^{(p)}, \ldots, \beta_{N/L}^{(p)}]^T$:

$$\beta^{(p)} = (\mathbf{H}^{(p)})^{-1}\mathbf{T}^{(p)} \tag{5.11}$$

where $\beta_i^{(p)}$ denotes the weight vector connecting i-th neuron in the first hidden-layer of the sub-TLFN to the p-th neuron group $G^{(p)}$ (m neurons) in the second hidden layer, $i = 1, \ldots, N/L$.

step 3: **Determination of weights and bias of neural quantizers**.

a) Set the biases of neurons $A^{(p)}$ and $B^{(p)}$, $p = 1, \cdots, L$.

b) Set the weights $\mathbf{w}_{A^{(p)}}$ and $\mathbf{w}_{B^{(p)}}$ of the connections linking the input layer to neurons $A^{(p)}$ and $B^{(p)}$, $p = 1, \cdots, L$, as

$$\begin{cases} \mathbf{w}_{A^{(p)}} = T \cdot \mathbf{w}, \\ \mathbf{w}_{B^{(p)}} = -T \cdot \mathbf{w}. \end{cases} \tag{5.12}$$

c) Set the weights w_{AB} of the connections linking neurons $A^{(p)}$ and $B^{(p)}$ to the second hidden layer as

$$w_{AB} = -U \tag{5.13}$$

Huang's network is a partially connected two-hidden-layer neural network. If the neural quantizers namely the neurons $A^{(p)}$ and $B^{(p)}$ are removed, the rest of the network is called sub-TLFN. The sub-TLFN has n input neurons and m output neurons, where n and m are the dimensions of the input and output vectors respectively. The sub-TLFN has N neurons in the first hidden layer and Lm in the second hidden layer. The neurons in the second hidden layer can be divided into L groups $G^{(1)}, \cdots, G^{(L)}$. The sub-TLFN can be consider as a combination of L TLFNs which share the same first hidden layer. Each TLFN corresponds to one group of neurons in the second hidden layer $G^{(p)}$. The neural quantizers divides the input space $\mathbf{R}$ into L sub-spaces $\mathbf{R}^{(p)}$ each learned by one TLFN. Given an input $\mathbf{x} \in \mathbf{R}^{(p)}$, only the p-th TLFN gives the a more correct answer. The neural quantizers block the output of all the other TLFNs so that only the p-th TLFN can output.

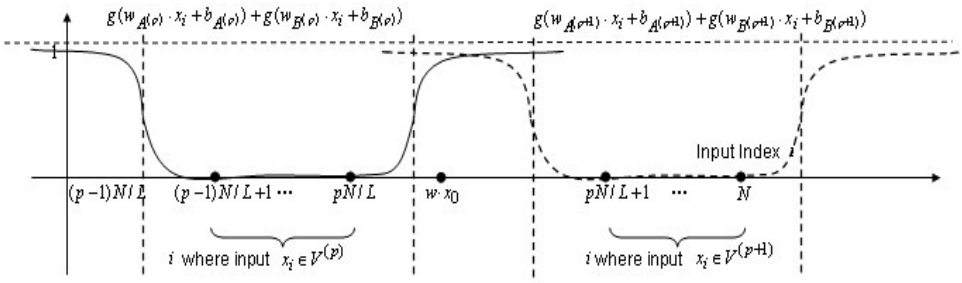
Seen from the above review, in Huang's network, the weights (matrix)

$\beta^{(p)}$ of the connections linking the first hidden layer of the sub-TLFN to the second hidden layer can be analytically calculated, while all the rest parameters (weights and biases) of Huang's network can be randomly pre-assigned (e.g., $\mathbf{w}_i$ and b_i) or pre-fixed (e.g., C). One may only adjust the quantizer factors T and U which may affect the weights of the neurons of each quantizer. In many applications *Huang's network may be fast enough to learn new stimuli from and response to external events within expected short time.* It is capable of learning new knowledge in large scale time-critical applications, thus, Huang's constructive method and an appropriate estimation of quantizer factors can form a modular implementation of extreme learning machine.

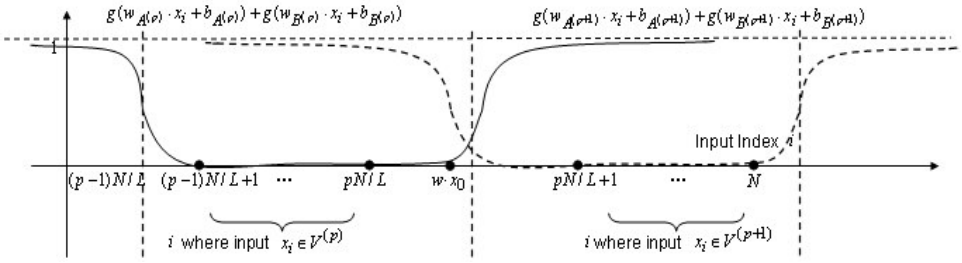
5.3 Proposed Modular Implementation of Extreme Learning Machine

5.3.1 Improved biases selection of neural quantizers

During the latest study, we found the originally proposed selection[18] of biases of neurons $A^{(p)}$ and $B^{(p)}$ (cf. equations (5.8) and (5.9)) can be further improved in order to make the generalization performance of Huang's network much more stable. It is guaranteed that by the originally proposed selection of biases of neurons $A^{(p)}$ and $B^{(p)}$ any training input vector $\mathbf{x}_i$ will



(a)



(b)

Figure 5.3: Outputs of two contiguous neural quantizers: (a) using previous selection of $\overline{b}_{A^{(p)}}$ and $\overline{b}_{B^{(p)}}$ there may exist a big gap not covered by both contiguous quantizers; (b) using improved selection criteria, in theory the size of the gap not covered by both contiguous quantizers becomes zero.

be always within only one of groups $V^{(p)}$, the output of one and only one neural quantizer is almost zero and the outputs of the rest neural quantizers are almost one. Thus, the training error can be made arbitrarily small using the originally proposed biases selection (refer to Section 5.2.C for detail discussions). However, when a testing input vector $\mathbf{x}_0$ happens to satisfy $\mathbf{w}\cdot\mathbf{x}_{pN/L}+\frac{1}{2}\min_{i,j}|\mathbf{w}\cdot\mathbf{x}_i-\mathbf{w}\cdot\mathbf{x}_j| < \mathbf{w}\cdot\mathbf{x}_0 < \mathbf{w}\cdot\mathbf{x}_{pN/L+1}-\frac{1}{2}\min_{i,j}|\mathbf{w}\cdot\mathbf{x}_i-\mathbf{w}\cdot\mathbf{x}_j|$, the outputs of two neural quantizers would be almost one, instead of one

neural quantizer only, which causes the testing error to be possibly larger and affects the generalization performance of the network. (cf. Figure 5.3(a))

To prevent this worse case happening and to achieve a better generalization performance, the selection of the biases $\overline{b}_{A(p)}$ and $\overline{b}_{B^{(p)}}$ of neural quantizers can be improved as:

$$\overline{b}_{A^{(p)}} = \begin{cases} -T\Big(\frac{1}{2}\mathbf{w}\cdot\mathbf{x}_{pN/L} + \frac{1}{2}\mathbf{w}\cdot\mathbf{x}_{pN/L+1}\Big), & \text{if } p \neq L \\ -T\Big(\mathbf{w}\cdot\mathbf{x}_N + \max_{j=1,\cdots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)\Big), & \text{if } p = L. \end{cases} \tag{5.14}$$

$$\overline{b}_{B^{(p)}} = \begin{cases} T\Big(\frac{1}{2}\mathbf{w}\cdot\mathbf{x}_{(p-1)N/L+1} + \frac{1}{2}\mathbf{w}\cdot\mathbf{x}_{(p-1)N/L}\Big), & \text{if } p \neq 1 \\ T\Big(\mathbf{w}\cdot\mathbf{x}_1 - \max_{j=1,\cdots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)\Big), & \text{if } p = 1. \end{cases} \tag{5.15}$$

By this bias selection of neural quantizers, the probability that a testing input vector $\mathbf{x}_0$ happens to make $\mathbf{w}\cdot\mathbf{x}_0 = \frac{1}{2}\mathbf{w}\cdot\mathbf{x}_{pN/L+1} + \frac{1}{2}\mathbf{w}\cdot\mathbf{x}_{pN/L}$ should be zero. Thus, for any testing input vector $\mathbf{x}_0$ the output of one and only one neural quantizer is almost zero and the outputs of the rest neural quantizers are almost one, which improves the generalization performance of the network. (cf. Figure 5.3(b))

5.3.2 Appropriate weights selection of neural quantizers

In order to determine the weights of neurons $A^{(p)}$ and $B^{(p)}$, one only needs to determine appropriate values for quantizer factors T and U. In fact, quantizer

factors T and U are the only parameters to be estimated for Huang's network. In this section, an appropriate method to estimate the suitable values of the quantizer factors T and U is proposed. These values can be decided based on Theorem 5.3.3. Two lemmas are needed to derive Theorem 5.3.3.

Lemma 5.3.1. *Given an arbitrarily small positive value $\eta < 0.5$, there exists a constant T_0^A*

$$T_0^A = \frac{2\ln(\frac{1-\eta}{\eta})}{\min_{j=1,\ldots N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)},$$

such that when $T \geqslant T_0^A$, $\forall \mathbf{x}_i \in V^{(k)}$, $k = 1, \ldots, L$, the corresponding output $O_{A^{(p)}}(\mathbf{x}_i)$ of neuron $A^{(p)}$ satisfies

$$O_{A^{(p)}}(\mathbf{x}_i) \begin{cases} \leqslant \eta, & \text{if } p \geqslant k \\ \geqslant 1-\eta, & \text{if } p < k \end{cases} \tag{5.16}$$

Proof. It can be proved in three cases.

Case 1: when $p \geqslant k$ and $p \neq L$.

According to selection (5.14), since

$$\begin{aligned} O_{A^{(p)}}(\mathbf{x}_i) &= \frac{1}{1+e^{-(T\mathbf{w}\cdot\mathbf{x}_i+\overline{b}_A^{(p)})}} \\ &= \frac{1}{1+e^{-T\left(\mathbf{w}\cdot\mathbf{x}_i-\frac{1}{2}(\mathbf{w}\cdot\mathbf{x}_{pN/L+1}+\mathbf{w}\cdot\mathbf{x}_{pN/L})\right)}}, \end{aligned} \tag{5.17}$$

to make $O_{A^{(p)}}(\mathbf{x}_i) \leqslant \eta$, T should satisfy

$$T\big(\mathbf{w}\cdot\mathbf{x}_i - \frac{1}{2}(\mathbf{w}\cdot\mathbf{x}_{pN/L+1} + \mathbf{w}\cdot\mathbf{x}_{pN/L})\big) \leqslant \ln\big(\frac{\eta}{1-\eta}\big).$$

$\forall \mathbf{x_i} \in V^{(k)}$, and $p \geqslant k$, $\mathbf{w} \cdot \mathbf{x}_i \leqslant \mathbf{w} \cdot \mathbf{x}_{pN/L}$, then T should satisfy

$$T \geqslant \frac{\ln(\frac{\eta}{1-\eta})}{\mathbf{w} \cdot \mathbf{x}_i - \frac{1}{2}(\mathbf{w} \cdot \mathbf{x}_{pN/L+1} + \mathbf{w} \cdot \mathbf{x}_{pN/L})} \tag{5.18}$$

Since, obviously we have

$$\frac{\ln(\frac{\eta}{1-\eta})}{\mathbf{w} \cdot \mathbf{x}_i - \frac{1}{2}(\mathbf{w} \cdot \mathbf{x}_{pN/L+1} + \mathbf{w} \cdot \mathbf{x}_{pN/L})} \leqslant \frac{2\ln(\frac{1-\eta}{\eta})}{\min_{j=1,\ldots,N-1}(\mathbf{w} \cdot \mathbf{x}_{j+1} - \mathbf{w} \cdot \mathbf{x}_j)}.$$

we can set $T_0^{A1} = \frac{2\ln(\frac{1-\eta}{\eta})}{\min_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)}$ such that when $T \geqslant T_0^{A1}$ Equation (5.18) holds.

Case 2: when $p \geqslant k$ and $p = L$.

According to selection (5.14).

$$O_{A^{(p)}}(\mathbf{x}_i) = \frac{1}{1 + e^{-\left(T\mathbf{w}\cdot\mathbf{x}_i - T(\mathbf{w}\cdot\mathbf{x}_N + \max_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j))\right)}}.$$

Similarly, to make $O_{A^{(p)}}(\mathbf{x}_i) \leqslant \eta$, T should satisfy

$$T \geqslant \frac{\ln(\frac{1-\eta}{\eta})}{-\mathbf{w} \cdot \mathbf{x}_i + \mathbf{w} \cdot \mathbf{x}_N + \max_{j=1,\ldots,N-1}(\mathbf{w} \cdot \mathbf{x}_{j+1} - \mathbf{w} \cdot \mathbf{x}_j)} \tag{5.19}$$

However, obviously we have

$$\begin{aligned} &\frac{\ln(\frac{1-\eta}{\eta})}{-\mathbf{w} \cdot \mathbf{x}_i + \mathbf{w} \cdot \mathbf{x}_N + \max_{j=1,\ldots,N-1}(\mathbf{w} \cdot \mathbf{x}_{j+1} - \mathbf{w} \cdot \mathbf{x}_j)} \\ &\qquad < \frac{\ln(\frac{1-\eta}{\eta})}{\max_{j=1,\ldots,N-1}(\mathbf{w} \cdot \mathbf{x}_{j+1} - \mathbf{w} \cdot \mathbf{x}_j)}. \end{aligned} \tag{5.20}$$

thus, we can set $T_0^{A2} = \frac{\ln(\frac{1-\eta}{\eta})}{\max_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)}$ such that when $T \geqslant T_0^{A2}$ Equation (5.19) holds.

Case 3: when $p < k$.

Similarly, according to selection (5.14), to make $O_{A^{(p)}}(\mathbf{x}_i) \geqslant 1 - \eta$,

T should satisfy

$$T \geqslant \frac{\ln(\frac{1-\eta}{\eta})}{\mathbf{w}\cdot\mathbf{x}_i - \frac{1}{2}(\mathbf{w}\cdot\mathbf{x}_{pN/L+1} + \mathbf{w}\cdot\mathbf{x}_{pN/L})} \tag{5.21}$$

However, obviously we have

$$\frac{\ln(\frac{1-\eta}{\eta})}{\mathbf{w}\cdot\mathbf{x}_i - \frac{1}{2}(\mathbf{w}\cdot\mathbf{x}_{pN/L+1} + \mathbf{w}\cdot\mathbf{x}_{pN/L})} \leqslant \frac{2\ln(\frac{1-\eta}{\eta})}{\min_{j=1,\ldots N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)}, \tag{5.22}$$

thus, we can set $T_0^{A3} = \frac{2\ln(\frac{1-\eta}{\eta})}{\min_{j=1,\ldots N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)}$ such that when $T \geqslant T_0^{A3}$ Equation (5.21) holds.

According to Cases 1-3, finally we can set

$$T_0^A = \max\{T_0^{A1}, T_0^{A2}, T_0^{A3}\} = \frac{2\ln(\frac{1-\eta}{\eta})}{\min_{j=1,\ldots N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)},$$

when $T \geqslant T_0^A$, $\forall \mathbf{x}_i \in V^{(k)}$, $k = 1, \ldots, L$, and any vector $\mathbf{w}$ satisfying $\mathbf{w}\cdot\mathbf{x}_i \neq \mathbf{w}\cdot\mathbf{x}_j$ $(i \neq j)$, the corresponding output $O_{A^{(p)}}(\mathbf{x}_i)$ of neuron $A^{(p)}$, $p = 1, \ldots, L$, satisfies inequality (5.16). This completes the proof. □

Similarly, we have

Lemma 5.3.2. *Given an arbitrarily small positive value $\eta < 0.5$, there exists a constant T_0^B*

$$T_0^B = \frac{2\ln(\frac{1-\eta}{\eta})}{\min_{j=1,\ldots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)},$$

such that when $T \geqslant T_0^B$, $\forall \mathbf{x}_i \in V^{(k)}$, $k = 1, \ldots, L$, the corresponding output $O_{B^{(p)}}(\mathbf{x}_i)$ of neuron $B^{(p)}$ satisfies

$$O_{B^{(p)}}(\mathbf{x}_i) \begin{cases} \leqslant \eta, & \text{if } p \leqslant k \\ \geqslant 1-\eta, & \text{if } p > k \end{cases} \tag{5.23}$$

Proof. The proof is similar to the proof of Lemma 5.3.1. □

From Lemma 5.3.1 and Lemma 5.3.2, we have

Theorem 5.3.1. *Given an arbitrarily small positive value* $\eta < 1$*, there exists a constant* T_0

$$T_0 = \frac{2\ln(\frac{2-\eta}{\eta})}{\min_{j=1,\ldots N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)}$$

such that when $T \geqslant T_0$*,* $\forall \mathbf{x}_i \in V^{(k)}$*,* $k = 1, \ldots, L$*, the output* $O_{A^{(p)}}(\mathbf{x}_i) + O_{B^{(p)}}(\mathbf{x}_i)$ *of p-th neural quantizer satisfies*

$$O_{A^{(p)}}(\mathbf{x}_i) + O_{B^{(p)}}(\mathbf{x}_i) \begin{cases} \leqslant \eta, & \text{if } p = k \\ \geqslant 1-\eta, & \text{if } p \neq k \end{cases} \tag{5.24}$$

Proof. Set T_0 as

$$T_0 = \frac{2\ln(\frac{2-\eta}{\eta})}{\min_{j=1,\ldots N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)}.$$

According to Lemma 5.3.1 when $T \geqslant T_0$, the output $O_{A^{(p)}}(\mathbf{x}_i)$ of neuron $A^{(p)}$ satisfies

$$O_{A^{(p)}}(\mathbf{x}_i) \begin{cases} \leqslant \frac{1}{2}\eta, & \text{if } p \geqslant k \\ \geqslant 1-\frac{1}{2}\eta, & \text{if } p < k \end{cases}. \tag{5.25}$$

and according to Lemma 5.3.2, when $T \geqslant T_0$, the output $O_{B^{(p)}}(\mathbf{x}_i)$ of neuron $B^{(p)}$ satisfies

$$O_{B^{(p)}}(\mathbf{x}_i) \begin{cases} \leqslant \frac{1}{2}\eta, & \text{if } p \leqslant k \\ \geqslant 1 - \frac{1}{2}\eta, & \text{if } p > k \end{cases} \tag{5.26}$$

Thus, when $T \geqslant T_0$, $\forall \mathbf{x}_i \in V^{(k)}$, $k = 1, \ldots, L$, the output $O_{A^{(p)}}(\mathbf{x}_i) + O_{B^{(p)}}(\mathbf{x}_i)$ of p-th neural quantizer satisfies inequality (5.24). □

Remark 1: Theorem 5.3.1 actually states, consistent with the design concept of Huang's network[18], that when the quantizer factor $T > T_0$, the output of one and only one neural quantizer is near to zero and the outputs of the rest are near to one. T_0 is the lower bound of the quantizer factor T. Now, we can further determine the appropriate value for quantizer factor U. For the sake of simplicity and convenience, we can consider single output case first.

Theorem 5.3.2. *Give arbitrarily small positive value $\epsilon < 1$, the quantizer factors T and U can be set as*

$$U = \ln\left(\frac{2CL}{\epsilon} - 1\right) + \max_{\substack{p=1,\ldots,L \\ q=1,\cdots,N/L \\ s=1,\cdots,L}} \{\mathbf{H}_q^{(p)} \cdot \beta^{(s)}\} + \min_{i=1,\cdots,N} \ln\left(\frac{(C+\epsilon-2t_i)(C+2t_i)}{(C-\epsilon+2t_i)(C-2t_i)}\right) \tag{5.27}$$

$$T = \frac{2\ln\left(2U \Big/ \min_{i=1,\cdots,N} \ln\left(\frac{(C+\epsilon-2t_i)(C+2t_i)}{(C-\epsilon+2t_i)(C-2t_i)}\right) - 1\right)}{\min_{j=1,\ldots N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)} \tag{5.28}$$

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

$$|O(\mathbf{x}_i) - t_i| \leqslant \epsilon, \tag{5.29}$$

where $\mathbf{H}_j^{(k)}$ *denotes the* j*-th row of the* $\mathbf{H}^{(k)}$ *matrix.*

Proof. According to Theorem 5.3.1, set T as

$$T = \frac{2\ln(\frac{2-\epsilon_0}{\epsilon_0})}{\min_{j=1,\ldots N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)} \tag{5.30}$$

the output $A^{(p)} + B^{(p)}$ of p-th neural quantizer satisfies

$$O_{A^{(p)}}(\mathbf{x}_i) + O_{B^{(p)}}(\mathbf{x}_i) \begin{cases} \leqslant \epsilon_0, & \text{if } p = k \\ \geqslant 1 - \epsilon_0, & \text{if } p \neq k \end{cases}$$

where ϵ_0 is a small positive value which will be decided later.

Let $o_j(\mathbf{x}_i)$ denote the actual output of j-th neuron in the second hidden-layer for the corresponding input $\mathbf{x}_i$, $i = 1, \ldots, N$ and $j = 1, \ldots, L$. Since $O(\mathbf{x}_i) = C(\sum_{j=1}^{L} o_j(\mathbf{x}_i) - 0.5)$, we have

$$|O(\mathbf{x}_i) - t_i| \leqslant |Co_k(\mathbf{x}_i) - 0.5C - t_i| + |C\sum_{j=1, j\neq k}^{L} o_j(\mathbf{x}_i)|.$$

Hence, if we can make

$$|Co_k(\mathbf{x}_i) - 0.5C - t_i| \leqslant \frac{\epsilon}{2} \tag{5.31}$$

and

$$\left|C\sum_{j=1, j\neq k}^{L} o_j(\mathbf{x}_i)\right| \leqslant \frac{\epsilon}{2}, \tag{5.32}$$

inequality (5.29) can be satisfied. Since $U(O_{A^{(k)}}(\mathbf{x}_i) + O_{B^{(k)}}(\mathbf{x}_i)) > 0$ we have,

$$Co_k(\mathbf{x}_i) - 0.5C - t_i = \frac{C}{1 + e^{-(t_i^{(k)} - U(O_{A^{(k)}}(\mathbf{x}_i) + O_{B^{(k)}}(\mathbf{x}_i)))}} - \frac{C}{1 + e^{-t_i^{(k)}}} < 0. \tag{5.33}$$

where $t_i^{(k)} = \ln\frac{0.5 + t_i/C}{0.5 - t_i/C}$. Then inequality (5.31) can be transformed into

$$Co_k(\mathbf{x}_i) \geqslant -0.5\epsilon + 0.5C + t_i$$

$$\frac{C-2t_i}{C+2t_i}e^{U(O_{A^{(k)}}(\mathbf{x}_i)+O_{B^{(k)}}(\mathbf{x}_i))} \leqslant \frac{C+\epsilon-2t_i}{C-\epsilon+2t_i}. \tag{5.34}$$

(In fact, equations (5.33) and (5.34) are valid as long as $\mathbf{H}^{(k)}\beta^{(k)}$ is close enough to $\mathbf{T}^{(k)}$ instead of $\mathbf{H}^{(k)}\beta^{(k)} = \mathbf{T}^{(k)}, k = 1, \cdots, L.$)

Since $\frac{C-2t_i}{C+2t_i} > 0$, we have

$$e^{U(O_{A^{(k)}}(\mathbf{x}_i)+O_{B^{(k)}}(\mathbf{x}_i))} \leqslant \frac{(C+\epsilon-2t_i)(C+2t_i)}{(C-\epsilon+2t_i)(C-2t_i)}.$$

then we have

$$U \leqslant \frac{\ln\left(\frac{(C+\epsilon-2t_i)(C+2t_i)}{(C-\epsilon+2t_i)(C-2t_i)}\right)}{O_{A^{(k)}}(\mathbf{x}_i)+O_{B^{(k)}}(\mathbf{x}_i)} \tag{5.35}$$

From Theorem 5.3.1, we know $0 < O_{A^{(k)}}(\mathbf{x}_i)+O_{B^{(k)}}(\mathbf{x}_i) \leqslant \epsilon_0$. Hence, if we can make

$$U \leqslant \frac{1}{\epsilon_0}\min_{i=1,\cdots,N}\ln\left(\frac{(C+\epsilon-2t_i)(C+2t_i)}{(C-\epsilon+2t_i)(C-2t_i)}\right), \tag{5.36}$$

inequality(5.31) can be satisfied.

To satisfy inequality(5.32), if for any j from 1 to L and $j \neq k$, there exists a U to make

$$|o_j(\mathbf{x}_i)| \leqslant \frac{\epsilon}{2CL}, \tag{5.37}$$

inequality (5.32) can be satisfied. Since $o_j(\mathbf{x}_i) > 0$, inequality (5.37) is simply equivalent to

$$o_j(\mathbf{x}_i) \leqslant \frac{\epsilon}{2CL}. \tag{5.38}$$

that is,

$$\frac{1}{1+e^{-\left(\mathbf{H}^{(k)}_{i-(k-1)L}\cdot\beta^{(j)}-U(O_{A^{(j)}}(\mathbf{x}_i)+O_{B^{(j)}}(\mathbf{x}_i))\right)}} \leqslant \frac{\epsilon}{2CL} \tag{5.39}$$

$$U \geqslant \frac{\ln\left(\frac{2CL}{\epsilon}-1\right)+\mathbf{H}^{(k)}_{i-(k-1)L}\cdot\beta^{(j)}}{O_{A^{(j)}}(\mathbf{x}_i)+O_{B^{(j)}}(\mathbf{x}_i))}. \tag{5.40}$$

(In fact, equations (5.39) to (5.40) are valid as long as $\mathbf{H}^{(k)}\beta^{(k)}$ is close enough to $\mathbf{T}^{(k)}$ instead of $\mathbf{H}^{(k)}\beta^{(k)} = \mathbf{T}^{(k)}, k = 1, \cdots, L$.)

Since

$$\begin{aligned} & \frac{\ln\left(\frac{2CL}{\epsilon} - 1\right) + \mathbf{H}^{(k)}_{i-(k-1)L} \cdot \beta^{(j)}}{O_{A^{(j)}}(\mathbf{x}_i) + O_{B^{(j)}}(\mathbf{x}_i))} \\ \leqslant & \frac{\ln\left(\frac{2CL}{\epsilon} - 1\right) + \max_{p=1,\ldots,L, q=1,\cdots,N/L, s=1,\cdots,L}\{\mathbf{H}^{(p)}_q \cdot \beta^{(s)}\}}{1-\epsilon_0}, \end{aligned}$$

if U can satisfy

$$U \geqslant \frac{\ln\left(\frac{2CL}{\epsilon} - 1\right) + \max_{p=1,\ldots,L, q=1,\cdots,N/L, s=1,\cdots,L}\{\mathbf{H}^{(p)}_q \cdot \beta^{(s)}\}}{1-\epsilon_0}, \tag{5.41}$$

inequality(5.32) can be satisfied. Considering inequalities (5.36) and (5.41), ϵ_0 should be chosen such that

$$\begin{aligned} & \frac{\ln\left(\frac{2CL}{\epsilon} - 1\right) + \max_{p=1,\ldots,L, q=1,\cdots,N/L, s=1,\cdots,L}\{\mathbf{H}^{(p)}_q \cdot \beta^{(s)}\}}{1-\epsilon_0} \\ & \leqslant \frac{1}{\epsilon_0} \min_{i=1,\cdots,N} \ln\left(\frac{(C+\epsilon-2t_i)(C+2t_i)}{(C-\epsilon+2t_i)(C-2t_i)}\right), \end{aligned} \tag{5.42}$$

Thus, ϵ_0 can be simply chosen as

$$\epsilon_0 = \frac{\min_{i=1,\cdots,N} \ln\left(\frac{(C+\epsilon-2t_i)(C+2t_i)}{(C-\epsilon+2t_i)(C-2t_i)}\right)}{\ln\left(\frac{2CL}{\epsilon} - 1\right) + \max\limits_{\substack{p=1,\ldots,L \\ q=1,\cdots,N/L \\ s=1,\cdots,L}} \{\mathbf{H}^{(p)}_q \cdot \beta^{(s)}\} + \min\limits_{i=1,\cdots,N} \ln\left(\frac{(C+\epsilon-2t_i)(C+2t_i)}{(C-\epsilon+2t_i)(C-2t_i)}\right)} \tag{5.43}$$

Furthermore, seen from inequality (5.41), U can be simply chosen as

$$U = \ln\left(\frac{2CL}{\epsilon} - 1\right) + \max\limits_{\substack{p=1,\ldots,L \\ q=1,\cdots,N/L \\ s=1,\cdots,L}} \{\mathbf{H}^{(p)}_q \cdot \beta^{(s)}\} + \min\limits_{i=1,\cdots,N} \ln\left(\frac{(C+\epsilon-2t_i)(C+2t_i)}{(C-\epsilon+2t_i)(C-2t_i)}\right) \tag{5.44}$$

and from inequality (5.30) we have

$$T = \frac{2\left(\ln\left(\frac{2CL}{\epsilon}-1\right)+\max\limits_{\substack{p=1,\ldots,L\\ q=1,\cdots,N/L\\ s=1,\cdots,L}}\{\mathbf{H}_q^{(p)}\cdot\beta^{(s)}\}\right)}{\min_{j=1,\ldots N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1}-\mathbf{w}\cdot\mathbf{x}_j)} \cdot \frac{2\ln\left(1+\frac{}{\min_{i=1,\cdots,N}\ln\left(\frac{(C+\epsilon-2t_i)(C+2t_i)}{(C-\epsilon+2t_i)(C-2t_i)}\right)}\right)}{} \tag{5.45}$$

This completes the proof. □

Remark 2: Observe from the proof of Theorem 5.3.2 that there is a tradeoff of quantizer factor U. As reviewed in Section 5.2.C, quantizer factor U cannot be set too small. In fact the lower bound of U has been given in equation (5.40)). Although Huang [18] requires U to be large enough, as shown in equation (5.36) there is also an upper bound on U, meaning U cannot be set too large either. If U is set too large the actual outputs of all the neurons in the second hidden-layer may be nearly zero and all the output may be inhibited. Thus, Theorem 5.3.2 gives an appropriate value for quantizer factor U.

Theorem 5.3.2 can be easily extended to the multi-output case:

Theorem 5.3.3. *Give arbitrarily small positive value $\epsilon < 1$, the quantizer factors T and U can be set as*

$$\begin{aligned} U &= \ln\left(\frac{2\sqrt{m}CL}{\epsilon}-1\right)+\max_{\substack{p=1,\ldots,L\\ q=1,\cdots,N/L\\ s=1,\cdots,L}}\|\mathbf{H}_q^{(p)}\cdot\beta^{(s)}\|_\infty \\ &+ \min_{i=1,\cdots,N}\ln\left(\frac{(C+\epsilon/\sqrt{m}-2t_{ij})(C+2t_{ij})}{(C-\epsilon/\sqrt{m}+2t_{ij})(C-2t_{ij})}\right) \end{aligned} \tag{5.46}$$

$$T = \frac{2\ln\left(2U/\min_{i=1,\cdots,N}\ln\left(\frac{(C+\epsilon/\sqrt{m}-2t_{ij})(C+2t_{ij})}{(C-\epsilon/\sqrt{m}+2t_{ij})(C-2t_{ij})}\right)-1\right)}{\min_{j=1,\ldots N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1}-\mathbf{w}\cdot\mathbf{x}_j)} \tag{5.47}$$

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

$$\|\mathbf{O}(\mathbf{x}_i) - \mathbf{t}_i\| \leqslant \epsilon. \tag{5.48}$$

Proof. According to Theorem 5.3.2, replacing ϵ with $\frac{\epsilon}{\sqrt{m}}$ we can set

$$\begin{aligned} U &= \ln\left(\tfrac{2\sqrt{m}CL}{\epsilon} - 1\right) + \max_{\substack{p=1,\ldots,L \\ q=1,\cdots,N/L \\ s=1,\cdots,L}} \|\mathbf{H}_q^{(p)} \cdot \beta^{(s)}\|_\infty \\ &+ \min_{i=1,\cdots,N} \ln\left(\tfrac{(C+\epsilon/\sqrt{m}-2t_{ij})(C+2t_{ij})}{(C-\epsilon/\sqrt{m}+2t_{ij})(C-2t_{ij})}\right) \end{aligned} \tag{5.49}$$

$$T = \frac{2\ln\left(2U \big/ \min_{i=1,\cdots,N} \ln\left(\tfrac{(C+\epsilon/\sqrt{m}-2t_{ij})(C+2t_{ij})}{(C-\epsilon/\sqrt{m}+2t_{ij})(C-2t_{ij})}\right) - 1\right)}{\min_{j=1,\ldots N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)} \tag{5.50}$$

$\forall \mathbf{x}_i$, we have,

$$|O_j(\mathbf{x}_i) - t_{ij}| \leqslant \frac{\epsilon}{\sqrt{m}},$$

where O_j denotes the output of the j-th output neuron of Huang's network. Then we have

$$\|\mathbf{O}(\mathbf{x}_i) - \mathbf{t}_i\| = \sqrt{\sum_{l=1}^{m} |O_j(\mathbf{x}_i) - t_{ij}|^2} \leqslant \epsilon$$

□

Remark 3: Theorem 5.3.3 gives a proper way to estimate the quantizer factor T and U based on the given goal of error ϵ.

5.3.3 Determination of Weight β

In Huang's constructive method, the weight β of sub-TLFN is determined as

$$\beta^{(p)} = (\mathbf{H}^{(p)})^{-1}\mathbf{T}^{(p)}, p = 1, \cdots, L \tag{5.51}$$

Huang [18] proved that the $\mathbf{H}^{(p)}$ is invertible when the number of neurons in the first hidden layer of sub-TLFN is N/L, which means that the total number of hidden neurons is $N/L + (m+2)L$. However, $N/L + (m+2)L$ hidden neurons may still be large for some problems, especially for those with large training set. In fact, zero training error is not required in real applications and may result in over-fitting. In fact, if we remove the output layer from the sub-TLFN, its just a ordinary SLFN as we discussed in Chapter 3. From Theorem 4.2.1, we know that *for a SLFN with the activation function* $g(x) = \frac{1}{1+e^{-x}}$ *in the hidden layer, given any constant* $\epsilon > 0$*, there always exists an integer* $\tilde{N} \leqslant N/L$ *such that a SLFN with* $\tilde{N}$ *hidden neurons and with randomly chosen input weights and hidden biases can learn* N/L *distinct observations with a training error less than* ϵ*.*

In this case, for SLFNs the weights linking the hidden layer and output layer can be calculated by the Moore-Penrose generalized inverse [103] as the same as the way that ELM determine the output weight. Hence, in the proposed new learning algorithm the weights β can be calculated as follows:

$$\beta^{(p)} = (\mathbf{H}^{(p)})^{\dagger}\mathbf{T}^{(p)}, \ p = 1, \cdots, L \tag{5.52}$$

where $(\mathbf{H}^{(p)})^{\dagger}$ is the Moore-Penrose generalized inverse of $\mathbf{H}^{(p)}$. Proofs of

Theorems (5.3.2) and (5.3.3) do not require $\mathbf{H}^{(p)}$ invertible. Seen from all the equations (5.33), (5.34), (5.39) and (5.40) which actually use $\mathbf{H}^{(p)}$, $\mathbf{H}^{(p)}\beta^{(p)}$ needs only to be close enough to $\mathbf{T}^{(p)}$ instead of the strict condition $\mathbf{H}^{(p)}\beta^{(p)} = \mathbf{T}^{(p)}, p = 1, \cdots, L$. According to Theorem 4.2.1, Theorems 5.3.2 and 5.3.3 are valid for some appropriate number $\tilde{N} \leqslant N/L$. The number of neurons in the first hidden layer of sub-TLFN no longer needs to be N/L but some appropriate number $\tilde{N}$ which can be set by users. Thus, it can help eliminate overfitting in noisy datasets.

5.3.4 Proposed Fast Learning Algorithm for TLFNs

Seen from Section 5.3.3, the new algorithm actually is an extension of the ELM for TLFNs. The constructive method (cf. Section 5.2) essentially is an approach of modular networks. It divides the training set into L groups, and each group is learned by a network module. Here network modules are SLFNs trained by the ELM algorithm. During the prediction, the neural quantizers activate the proper network module's output with respect to the input. Suppose that the learning complexity of a learning algorithm for the whole training set is $O(N^{\alpha})(\alpha > 1)$, where N is the number of training observations. Then, the learning complexity of each sub neural network is $O((N/L)^{\alpha}) = O(N^{\alpha})/L^{\alpha}$. Since the neural quantizers can be built in nearly zero time, the total learning time is almost equal to the total learning time of all sub neural network: $L \times O(N^{\alpha})/L^{\alpha} = O(N^{\alpha})/L^{(\alpha-1)}$. Obviously,

in principle, complexity factors α of all learning algorithms including the ELM are larger than one. Therefore, the new learning algorithm can be even faster than the ELM. Because the proposed learning algorithm can be extremely fast and can meet the fast learning needs of most applications, it is called modular implementation of extreme learning machine (MI-ELM). The learning algorithm can be described as follows:

The Proposed MI-ELM:

Input:

1. Given N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$, where $\mathbf{x}_i = [x_{i1}, x_{i2}, \cdots, x_{in}]^T \in \mathbf{R}^n$ and $\mathbf{t}_i = [t_{i1}, t_{i2}, \cdots, t_{im}]^T \in \mathbf{R}^m$.
2. Given the expected learning accuracy $\epsilon < 0$.
3. Given the number of groups L.
4. Given the number of neurons $\tilde{N}$ of the first hidden layer of the sub-TLFN.

Learning procedure of MI-ELM:

step 1: **Sorting and grouping inputs**.

a) Random choose vector $\mathbf{w} \in \mathbf{R}^n$ and reindex i of inputs such that $\mathbf{w} \cdot \mathbf{x}_1 < \mathbf{w} \cdot \mathbf{x}_2 < \cdots < \mathbf{w} \cdot \mathbf{x}_N$.

b) Group sorted inputs into L groups $V^{(p)}$, $p = 1, \cdots, L$

$$V^{(p)} = \{\mathbf{x}_i | \mathbf{w} \cdot \mathbf{x}_{(p-1)N/L+1} \leqslant \mathbf{w} \cdot \mathbf{x}_i \leqslant \mathbf{w} \cdot \mathbf{x}_{pN/L}\} \tag{5.53}$$

step 2: **Determination of weights and bias of standard sub-TLFN**.

a) Randomly choose the weights $\mathbf{w}_i$ and biases b_i, $i = 1, \cdots, \tilde{N}$, where $\tilde{N} \leqslant N/L$ is the number of neurons in the first hidden layer of sub-TLFN and it is set by user.

b) Choose $C = a \max_{i=1,\cdots,N;j=1,\cdots,m} |t_{ij}|$, a can be any positive value larger than 2.

c) Calculate matrix $\beta^{(p)} = [\beta_1^{(p)}, \ldots, \beta_{\tilde{N}}^{(p)}]^T$:

$$\beta^{(p)} = (\mathbf{H}^{(p)})^{\dagger} \mathbf{T}^{(p)} \tag{5.54}$$

step 3: **Determination of weights and bias of neural quantizers**.

a) Set the quantizer factor T and U as

$$T = \frac{2 \ln \left(2U / \min_{i=1,\cdots,N} \ln \left(\frac{(C+\epsilon/\sqrt{m}-2t_{ij})(C+2t_{ij})}{(C-\epsilon/\sqrt{m}+2t_{ij})(C-2t_{ij})} \right) - 1 \right)}{\min_{j=1,\ldots N-1} (\mathbf{w} \cdot \mathbf{x}_{j+1} - \mathbf{w} \cdot \mathbf{x}_j)} \tag{5.55}$$

$$\begin{aligned} U = & \ln \left(\frac{2\sqrt{m} C L}{\epsilon} - 1 \right) + \max_{\substack{p=1,\ldots,L \\ q=1,\cdots,N/L \\ s=1,\cdots,L}} \| \mathbf{H}_q^{(p)} \cdot \beta^{(s)} \|_{\infty} \\ & + \min_{i=1,\cdots,N} \ln \left(\frac{(C + \epsilon/\sqrt{m} - 2t_{ij})(C + 2t_{ij})}{(C - \epsilon/\sqrt{m} + 2t_{ij})(C - 2t_{ij})} \right) \end{aligned} \tag{5.56}$$

b) Set the weights $\mathbf{w}_{A^{(p)}}$ and $\mathbf{w}_{B^{(p)}}$ of the connections linking the input layer to neurons $A^{(p)}$ and $B^{(p)}$, $p = 1, \cdots, L$, as

$$\begin{cases} \mathbf{w}_{A^{(p)}} = T \cdot \mathbf{w}, \\ \mathbf{w}_{B^{(p)}} = -T \cdot \mathbf{w}. \end{cases} \tag{5.57}$$

c) Set the biases of neurons $A^{(p)}$ and $B^{(p)}$, $p = 1, \cdots, L$ as

$$\overline{b}_{A^{(p)}} = \begin{cases} -T\left(\frac{1}{2}\mathbf{w}\cdot\mathbf{x}_{pN/L} + \frac{1}{2}\mathbf{w}\cdot\mathbf{x}_{pN/L+1}\right), & \text{if } p \neq L \\ -T\left(\mathbf{w}\cdot\mathbf{x}_N + \max_{j=1,\cdots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)\right), & \text{if } p = L. \end{cases} \tag{5.58}$$

$$\overline{b}_{B^{(p)}} = \begin{cases} T\left(\frac{1}{2}\mathbf{w}\cdot\mathbf{x}_{(p-1)N/L+1} + \frac{1}{2}\mathbf{w}\cdot\mathbf{x}_{(p-1)N/L}\right), & \text{if } p \neq 1 \\ T\left(\mathbf{w}\cdot\mathbf{x}_1 - \max_{j=1,\cdots,N-1}(\mathbf{w}\cdot\mathbf{x}_{j+1} - \mathbf{w}\cdot\mathbf{x}_j)\right), & \text{if } p = 1. \end{cases} \tag{5.59}$$

d) Set the weights w_{AB} of the connections linking neurons $A^{(p)}$ and $B^{(p)}$ to the second hidden layer as

$$w_{AB} = -U \tag{5.60}$$

Remark 4: Observe that the basic algorithm of Huang's network [18] have the same form as the new proposed MI-ELM algorithm except that for the basic algorithm: 1) the number of neurons $\tilde{N}$ of the first hidden layer of the sub-TLFN was set N/L instead of some appropriate number less than N/L, and thus $\mathbf{H^{(p)}}$ of equation (5.54) was always a square matrix; 2) T and U are manually pre-defined by users instead of automatically computed as in (5.55) and (5.56); 3) the biases of neurons $A^{(p)}$ and $B^{(p)}$ were set according to equation (5.8) and (5.9) instead of (5.58) and (5.59). In MI-ELM, the limitation that the number of neurons $\tilde{N}$ of the first hidden layer of the sub-TLFN is equal to N/L (number of training data in each group $V^{(p)}$) has been removed. Setting $\tilde{N}$ to some appropriate number less than N/L can not only improve the generalization performance by preventing MI-ELM from

overfitting in noisy datasets but also further reduce the network complexity of Huang's network.

Remark 5: The proposed algorithm requires very small memory in computation: 1) Huang's network is not a fully-connected standard TLFN. It is a sparse network and it has small number of trainable parameters; 2) The L matrix functions $\mathbf{H}^{(p)}$ in equation (5.54) share and are computed from a single common $N/L \times \tilde{N}$ matrix $\mathbf{H}$, instead of L separated $N/L \times \tilde{N}$ matrices stored in the memory. The L matrix functions $\mathbf{T}^{(p)}$, $p = 1, \cdots, L$, share and are computed from a single common $N/L \times m$ matrix $\mathbf{T}$ as defined in equation (5.5) as well. The algorithm spends most of its learning time in the matrix computation of $\mathbf{H}$ and $\mathbf{T}$. However, in applications, these two matrices are normally not very large and the computation is very simple and extremely fast. Thus, in general, the algorithm can provide seconds very quick learning solution to many applications including large scale complex systems. The results reported in this chapter are averaged over 50 trials of simulations for each case. The training and testing sets are randomly regenerated before each trial of simulation. The results listed here are the ones with the best mean generalization performance. The same process takes place when using the MI-ELM. The parameter L of groups and number $\tilde{N}$ of hidden neurons in the sub-TLFN are also gradually increased and the network architecture of MI-ELM shown in different tables is therefore denoted by $(L, \tilde{N})$. We set $\epsilon = 10^{-6}$ for all cases. For detailed experiment settings, please refer to

Section 3.5

Remark 6: As discussed in Chapter 3, the computational complexity of Moore-Penrose generalized inverse in the ELM algorithm is $O(N\tilde{N}^2)$. Then we can get the total computational complexity of calculating the Moore-Penrose generalized inverse in the MI-ELM algorithm is $L \times O((N/L) \times \tilde{N}'^2)$ i.e. $O(N\tilde{N}'^2)$, where $\tilde{N}'$ is the number of neurons in the first hidden layer of sub-TLFN. It looks like the computational complexities of the both algorithm are about the same. However, one should be reminded that the number of hidden neurons needed by the MI-ELM $\tilde{N}'$ is much smaller than the number needed by the ELM $\tilde{N}$ as each sub training set only has N/L training samples, so the MI-ELM can be even faster than the ELM.

5.4 Benchmarking with Real-World Regression Problems

Numerous real-world regression datasets from UCI Machine Learning Repository [97] have been tested to verify the performance of the proposed algorithm.

Datasets	# Observations		# Attributes
	Training	Testing	
Auto Price	80	79	15
Boston Housing	250	256	13
Pyrimidines	40	34	27
Triazines	100	86	60
Machine CPU	100	109	6
Servo	80	87	4
Breast Cancer	100	94	32
Stocks	450	500	10

Table 5.1: Specifications of small real-world regression problems.

5.4.1 Small Real-World Regression Problems

Some small regression datasets (with less than 1000 total instances) are tested here. The specifications of these dataset are listed in Table 5.1. As observed from Table 5.2, MI-ELM wins over the BP on most of the datasets in terms of testing accuracy and obtains a much shorter training time on all the datasets. The proposed MI-ELM shows its fast learning and prediction capability on these datasets. In some datasets like Auto Price, Pyrimidines and etc., the training time of MI-ELM is less than 0.001 seconds.

Datasets	Algorithms	$(L, \tilde{N})$ or $\tilde{N}$	Testing		Training Time (s)
			RMSE	SD	
Auto Price	MI-ELM	(2, 10)	0.1059	0.0124	0.0002
	BP	5	0.1157	0.0245	0.2456
Boston Housing	MI-ELM	(2, 30)	0.0950	0.0078	0.0030
	BP	5	0.0976	0.0051	0.3755
Pyrimidines	MI-ELM	(2, 10)	0.1443	0.0408	0.0031
	BP	5	0.1391	0.0550	0.2675
Triazines	MI-ELM	(2, 10)	0.2107	0.0208	0.0015
	BP	5	0.2197	0.0564	0.5481
Machine CPU	MI-ELM	(2, 10)	0.0651	0.02560	0.0001
	BP	10	0.0826	0.0165	0.2354
Servo	MI-ELM	(2, 15)	0.1264	0.0279	0.0003
	BP	10	0.1276	0.0754	0.2447
Breast Cancer	MI-ELM	(2, 5)	0.2736	0.0135	0.0004
	BP	5	0.3155	0.0891	0.3856
Stocks Domain	MI-ELM	(2, 60)	0.0317	0.0023	0.0314
	BP	20	0.0358	0.0056	1.0487

Table 5.2: Performance comparison of MI-ELM and BP on small real-world regression problems: testing RMSE and its standard deviation (Dev), training time (seconds), and network architectures.

5.4.2 Medium to Large Real-World Regression Problems

Datasets	# Observations		# Attributes
	Training	Testing	
Abalone	2000	2177	8
Delta Ailerons	3000	4129	6
Delta Elevators	4000	5517	6
2D Planes	10000	30768	10
Kinematics of Robot Arm	4000	4192	8
Census (House8L)	10000	12784	8
Pumadyn	4500	3692	8
Bank	4500	3692	8
California Housing	8000	12460	8

Table 5.3: Specifications of medium to large real-world regression problems.

The performance of MI-ELM is also tested on some medium to large regression datasets. The specifications of these dataset are listed in Table 5.3.

As observed from Table 5.4, MI-ELM achieves the shorter training time and higher testing accuracy in most datasets as expected. MI-ELM still can complete the training procedure within 1 second for most of these medium to large regression problems.

5.5 Benchmarking with Real-World Classification Problems

5.5.1 Small to Medium Classification Problems

The datasets for classification are also selected from UCI Machine Learning Repository [97]. The experimental settings of classification problems are the same as regression problems. In addition, the performance of MI-ELM and k nearest neighbour (k-NN) algorithm are discussed in Subsection 5.5.5.

Datasets	Algorithms	$(L, \tilde{N})$ or $\tilde{N}$	Testing		Training Time (s)
			RMSE	Dev	
Abalone	MI-ELM	(5, 10)	0.0851	0.0089	0.0108
	BP	10	0.0874	0.0055	1.7562
Delta Ailerons	MI-ELM	(5, 20)	0.0467	0.0062	0.039
	BP	10	0.0481	0.0013	2.7525
Delta Elevators	MI-ELM	(5, 20)	0.0575	0.0055	0.0468
	BP	5	0.0592	0.0087	1.1939
2D Planes	MI-ELM	(5, 360)	0.0452	0.0010	16.513
	BP	5	0.0429	0.0036	21.67
Kinematics	MI-ELM	(5, 300)	0.0876	0.0061	6.0594
	BP	90	0.0741	0.0013	134.98
Census (house8L)	MI-ELM	(5, 40)	0.0677	0.0019	0.3281
	BP	10	0.0685	0.0038	8.0647
Pumadyn Domains	MI-ELM	(5, 140)	0.1325	0.0044	1.1766
	BP	5	0.1474	0.0052	0.2735
Bank Domains	MI-ELM	(5, 80)	0.0359	0.0012	0.3966
	BP	20	0.0379	0.0041	7.506
California Housing	MI-ELM	(5, 60)	0.1271	0.0027	0.5422
	BP	10	0.1285	0.0088	6.532

Table 5.4: Performance comparison of MI-ELM and BP on medium to large real-world regression problems: testing RMSE and its standard deviation (Dev), training time (seconds), and network architectures.

Datasets	# Attributes	# Classes	# Observations	
			Training	Testing
Glass	9	7	110	104
Ionosphere	33	2	175	176
Page Blocks	10	5	2,700	2,773
Pima	8	2	380	388
Vehicle	18	4	420	426
Heart	13	2	135	135
German	24	2	500	500
Image Segmentation	19	7	1,100	1,210
Satellite Image	36	6	4,400	2,035
Letter Recognition	16	26	10,000	10,000
Forest Cover Type	54	2	100,000	481,012
Spam Emails	57	2	3,000	1,601

Table 5.5: Specifications of real-world classification problems.

The performance of MI-ELM and BP are compared on ten small to medium classification datasets which have less than 10,000 instances. The specifications of these datasets are listed in Table 5.5 and the simulation results are shown in Table 5.6.

Datasets	Algorithms	$(L, \tilde{N})$ or $\tilde{N}$	Testing		Training Time (s)
			Rate	Dev	
Glass	MI-ELM	(2, 20)	65.442%	3.937%	0.0015
	BP	10	65.154%	4.728%	0.8938
Ionosphere	MI-ELM	(5, 15)	86.489%	3.066%	0.0077
	BP	5	85.727%	4.037%	0.7955
Page Blocks	MI-ELM	(5, 40)	95.914%	0.284%	0.0923
	BP	10	95.515%	0.665%	63.983
Pima	MI-ELM	(2, 20)	75.747%	1.779%	0.0045
	BP	10	75.216%	2.059%	0.6891
Vehicle	MI-ELM	(2, 60)	79.808%	2.292%	0.0311
	BP	25	79.502%	3.685%	66.089
Heart	MI-ELM	(5, 10)	78.741%	3.826%	0.0032
	BP	5	78.657%	5.847%	0.4062
Germen	MI-ELM	(2, 20)	72.660%	1.455%	0.0047
	BP	10	72.360%	2.812%	2.5219
Image Segment	MI-ELM	(2, 100)	94.388%	0.637%	0.1486
	BP	60	94.030%	0.886%	1030.7
Satellite Image	MI-ELM	(5, 140)	89.892%	0.526%	1.2157
	BP	90	89.440%	0.824%	228.48

Table 5.6: Performance comparison of MI-ELM and BP on small to medium real-world classification problems: successful classification rate on testing data and its standard deviation (Dev), training time (seconds), and network architectures.

In terms of testing accuracy, the performance of the both algorithms are very close in almost all the datasets though MI-ELM is slightly better than BP. Observed from Table 5.6, MI-ELM learns hundreds of times faster than BP in most cases. For example, MI-ELM learns 6700 times faster than BP in Image Segment case. Both algorithms are quite stable for the standard deviation of testing time in different trial are close and small. It is worth point out that BP generally results in more compact networks than ELM.

5.5.2 Large Classification Application: Letter Recognition

The dataset with 16 input attributes and 26 class labels was constructed by Slate [97]. The objective is to classify each of a large number of black and white rectangular pixel display as one of the 26 capital letters of the English alphabet. The character images produced were based on 20 different fonts and each letter within these fonts was randomly distorted to produce a file of 20,000 unique images. For each image, 16 numerical attributes were calculated using edge counts and measures of statistical moments which were scaled and discretized into a range of integer values from 1 to 15 (cf. [105]). Either training set or testing set has 10,000 instances.

As observed from Table 5.7, the successful classification rates on testing data obtained by both MI-ELM ($L = 5$) and ELM are more than 93%,

Algorithms	$(L, \tilde{N})$ or $\tilde{N}$	Testing		Training Time (s)
		Rate	Dev	
MI-ELM	(5, 740)	93.239%	0.559%	88.077
MI-ELM	(10, 420)	86.506%	0.750%	32.863
ELM	900	93.730%	0.398%	142.073
BP	100	85.390%	1.636%	3977.2

Table 5.7: Performance comparison between MI-ELM, ELM and BP on a large classification application - handwritten letter recognition: successful classification rate on testing data and its standard deviation (Dev), training time (seconds), and network architectures.

which is much higher than the testing accuracy obtained by BP. The best testing accuracy obtained by BP through cross-validation method is just above 85%. The best testing accuracy obtained by Frey and Slate [107] using other methods is only a little over 80% which is much lower than the testing accuracies obtained by the other three algorithms.

5.5.3 Very Large Classification Application: Forest Cover Type Prediction

This is an extremely large complex classification problem with seven classes. For dataset descriptions and experimental settings, please refer to Section

Algorithms	$(L, \tilde{N})$ or $\tilde{N}$	Testing		Training Time (s)
		Rate	Dev	
MI-ELM	(10, 350)	92.476%	1.493%	139.535
MI-ELM	(20, 200)	91.164%	1.924%	102.928
ELM	400	92.55%	0.9301%	188.346
BP	20	89.918%	0.9722%	1857.4
SVM [108]	31806 SVs	89.897%	-	28813.8
SVM Modular Network [108]	38023 SVs	89.74%	-	1542
Hard Probabilistic Mixture [109]	-	91.07%	-	17460

Table 5.8: Performance comparison between MI-ELM, ELM, BP and SVM on a very large classification application - forest cover type prediction: successful classification rate on testing data and its standard deviation (Dev), training time (seconds), and network architectures.

3.5.3. As observed from Table 5.8, the successful prediction rates obtained by MI-ELM and ELM are above 92%. Single SVM [108] and SVM Modular Network [108] spent 480.23 minutes and 25.7 minutes on training and obtained 89.897% and 89.74% prediction rates, respectively, which are similar to BP's. Hard probabilistic mixture of SVM with 20 SVM experts and a MLP gater with 150 hidden neurons spent 291 minutes on training and obtained 91.07% prediction rate for this case (cf. Table 5 of [109]). SVM [98, 108] and SVM Modular Network [108] need more than 30,000 support vectors (SVs),

which is much larger than the hidden neurons required by MI-ELM and BP. Thus, SVM needs much longer prediction testing time (> 3.5 hours) than BP and MI-ELM.

5.5.4 Application Requiring Fast Learning: Spam Emails Prediction

Algorithms	$(L, \tilde{N})$ or $\tilde{N}$	Testing		Training Time (s)
		Rate	Dev	
MI-ELM	(2, 140)	91.861%	0.440%	0.8358
MI-ELM	(5, 80)	90.657%	0.756%	0.2547
MI-ELM	(10, 70)	89.327%	0.578%	0.2468
ELM	200	92.083%	0.384%	1.4186
BP	50	91.836%	0.745%	4641.9

Table 5.9: Performance comparison between MI-ELM, ELM, BP and SVM on a classification application requiring fast learning - spam email prediction: successful classification rate on testing data and its standard deviation (Dev), training time (seconds), and network architectures.

This dataset consists of spam emails and non-spam emails. Classifiers are used to predict whether an incoming email is spam or not. Sometimes some new types of spam and on-spam emails may be indicated by users explictly, in this case, the classifiers should be able to online learn the new

spam and non-spam emails patterns as fast as possible such that it is able to recognize and ban the spam emails from the upcoming emails. There are 4601 instances and each instance has 57 attributes. In the simulations, 3000 randomly selected instances compose the training set and all the rest of the instances are used for testing. As shown in Table 5.9, the MI-ELM achieves good testing accuracy at very fast learning speed ($<$ 1 second), however, BP needs to spend 4641.9 seconds on learning which is not realistic in such as practical time-critical application. In fact, many new patterns of spam emails may come during such long period of time and BP classifiers may not be able to classify them as its learning may not be completed.

From the results of the previous three classification problems, the differences between the prediction accuracies achieved by the MI-ELM and ELM are marginal, but the training time of the MI-ELM is shorter than ELM. Hence, the MI-ELM is more suitable for applications concerning learning time more than accuracy.

5.5.5 Comparison between MI-ELM and k-NN Algorithm

MI-ELM is also compared with k-nearest neighbour approach (k-NN) on these real-world classification applications. Like the other two algorithms, the parameter k of k-NN is also gradually increased to search for the best general-

Datasets	MI-ELM			k-NN		
	$(L, \tilde{N})$	Rate	Time (s)	k	Rate	Time (s)
Glass	(2, 20)	65.442%	1.0e-5	1	65.308%	0.0016
Ionosphere	(5, 15)	86.489%	0.0032	2	86.466%	0.0281
Liver Disorders	(2, 10)	68.171%	1.0e-5	11	61.714%	0.0156
Page Blocks	(5, 40)	95.914%	0.0984	1	95.316%	3.1109
Pima	(2, 20)	75.747%	0.0047	15	73.995%	0.0594
Vehicle	(2, 60)	79.808%	0.0127	1	68.169%	0.0938
Heart	(5, 10)	78.741%	1.0e-5	15	77.556%	0.0109
Germen	(2, 20)	72.660%	0.0064	14	72.320%	0.1484
Image Segment	(2, 100)	94.388%	0.0468	1	93.694%	0.6781
Satellite Image	(5, 140)	89.892%	0.1421	3	88.747%	8.3781
Letter Recognition	(5, 740)	93.239%	4.3611	1	93.119%	87.783
Forest Type	(10, 350)	92.476%	65.648	1	92.081%	93893
Spam Prediction	(2, 140)	91.861%	0.0939	1	89.963%	8.4877

Table 5.10: Performance comparison of MI-ELM and k-NN on real-world classification problems.

ization performance. Table 5.10 shows the comparison between MI-ELM and k-NN on all these classification problems. Obviously, the testing accuracy achieved by k-NN is generally worse than MI-ELM. Its testing/prediction time is much longer than the other two algorithms especially on the datasets

having large numbers of testing data. For example of forest type prediction problem which has 100,000 training data and 481,012 testing data, the testing time of k-NN can be as long as 26 hours which is more than one day. Single SVM spent more than 3.5 hours on predicting such large number of unknown testing data [108].

5.6 Summary

Based on Huang's constructive method[18], the ELM learning algorithm were successfully extended to train TLFNs and training speed is further enhanced. Three improvements have been made in the proposed new algorithm: 1) New selection criteria for the biases of neural quantizer has been proposed to improve the generalization performance; 2) An automatic method to select the appropriate values of the quantizer factors T and U has been introduced. In fact the lower bound of T and lower and upper bounds of U have been derived and given in this chapter (see Remarks 1 and 2 for details); 3) Hidden output matrix $\mathbf{H}$ need not be invertible and the Moore-Penrose (MP) generalized inverse of $\mathbf{H}$ is introduced as the same as the ELM, so that the number of neurons in the first hidden layer of sub-TLFN $\tilde{N}$ can be set to some appropriate number smaller than N/L and thus the network architecture can be made smaller. The proposed MI-ELM algorithm has been successfully tested on many benchmarking datasets of different types. Huang's network is trained

in a simple efficient one-time way instead of gradient-based and/or iterative methods: 1) all the weights of the connections linking the input layer to the first hidden layer of the sub-TLFN are randomly generated; 2) all the biases of the first hidden neurons of the sub-TLFN are also randomly generated; 3) all the rest parameters (weights and biases) of Huang's network can be simply analytically computed based on a small size of first hidden output matrix of the sub-TLFN. Huang's network [18] is a sparse TLFN and needs small memory for computation. Thus, MI-ELM can implement many large scale systems in an ordinary PC.

Since the proposed MI-ELM algorithm is purely based on analytical method it outperforms the other conventional gradient-based and/or iterative approaches without requiring stopping criterion and avoiding hidden local minima issues. Local minima problem existing in conventional learning algorithms make networks almost incapable of learning in some applications.

The performance evaluation of the proposed MI-ELM has been systematically investigated in this section. Seen from the simulations results on many real-world regression and classification problems, the MI-ELM can not only reach good generalization performance but also provide very short learning and prediction time in many cases, which may be within most users' expected time in many applications. In other words, the proposed MI-ELM algorithm or Huang's network would provide fast alternative solutions to time-critical applications. Thus, the proposed MI-ELM algorithm may be

a good candidate to the applications: 1) which have large training samples and require high learning accuracy/generalization performance and/or high learning speed; 2) where the application patterns and environments change fast and long time learning is not allowed. Even for those applications where long training time is allowed, the proposed MI-ELM can be used and tried as the first choice since almost zero-time 'effort' needs to be made for this trial. If the performance is accepted tedious time-consuming trials of other algorithms can be prevented.

Part II

Variants of Extreme Learning Machine

Chapter 6

Evolutionary Extreme Learning Machine(E-ELM)

Although the main objective of this thesis is to propose fast learning algorithms for feedforward neural networks, we also investigate the algorithms to reduce the network size and improve the generalization performance. In this chapter, a new hybrid of **d**ifferential **e**volution (DE) algorithm and ELM which can achieve compact network architecture and better generalization performance is proposed.

6.1 Introduction

In the ELM introduced in Chapter 3, the input weights (linking the input layer to the hidden layer) and hidden biases are randomly chosen, and the output weights (linking the hidden layer to the output layer) are analytically determined by using Moore-Penrose (MP) generalized inverse. ELM not only learns much faster with higher generalization performance than the traditional gradient-based learning algorithms but also avoid many difficulties faced by gradient-based learning methods such as stopping criteria, learning rate, learning epochs, and local minima. However, it is also found that ELM tends to require more hidden neurons than conventional tuning based algorithms in many cases. In this chapter, we suppose that in the applications the requirements of the prediction time and storage complexity have higher priority than the learning time, a novel learning algorithm which can achieve more compact network architecture (less hidden neurons) than the ELM is proposed.

There are some existing algorithms which can intelligently choose hidden units or support vectors for RBF networks or support vector machines (SVM). Resource allocation network (RAN) [66] and minimal resource allocation network (MRAN) [67] are two typical sequential learning algorithms for RBF network. RAN was brought forward by Platt[66] first. The main idea of this algorithm is hidden neurons are added sequentially based on the

novelty of the new data. The network learns by allocating new units and adjusting the parameters of existing units. If the network performs poorly on presented pattern, then a new unit is allocated that corrects the response to the presented pattern. If the network performs well on presented pattern, then the network parameters are updated using standard Widrow-Hoff LMS (least mean square) gradient descent. Based on Platt's , Kadirkamanathan and Niranjan's [110] work, Yingwei, et al[67] found an algorithm known as MRAN (Minimum Resource Allocation Network) that has been used in a number of applications by introducing a pruning strategy based on the relative contribution of each hidden neuron to the overall network output. Similar to learning algorithms for RBF networks, most learning methods for SVM can intelligently and selectively choose support vectors from training set as well. However, all these methods cannot be easily applied to multilayer feedforward neural networks. Unlike the RBF hidden node or kernels which are only effective within certain ranges from the centers, the sigmoid and sine functions are global functions thus it is hard to add or remove a neuron without affecting the overall performance of the network. New methods need to be explored to optimize the hidden neurons of multilayer feedforward neural networks.

Since evolutionary algorithms (EAs) are widely used as a global searching method for optimization, the hybrids of EA and analytical methods should be promising for network training. In the method proposed by Ghosh

and Verma [111] namely GALS, the input weights are tuned by EA and the output weights are *iteratively* tuned/updated using the QR factorization method. Instead of iterative optimization of the output weights, our approach takes the advantages of both the ELM and the differential evolution (DE) [112] is proposed. DE is known for its ability and efficiency to locate global optimum over other EAs (cf. [112]). In the proposed algorithm, a modified DE is used to search for the optimal input weights and hidden biases, while the MP generalized inverse is used to analytically calculate the output weights. Hence, with the new algorithm, a more compact network architecture can be expected.

6.2 Differential Evolution

Differential evolution (DE) proposed by Storn and Price [112] is known as one of the most efficient evolutionary algorithms (EAs). The basic strategy of DE can be described as follows:

Given a set of parameter vectors $\{\theta_{i,G}|i = 1, 2, \ldots, NP\}$ as a population at each generation G, we do:

Mutation: For each target vector $\theta_{i,G+1}, i = 1, 2, \ldots, NP$, a mutant vector is generated according to

$$\nu_{i,G+1} = \theta_{r_1,G} + F \cdot (\theta_{r_2,G} - \theta_{r_3,G}) \tag{6.1}$$

with random and mutually different indices $r_1, r_2, r_3 \in \{1, 2, \ldots, NP\}$ and $F \in [0, 2]$. The constant factor F is used to control the amplification of the differential variation $(\theta_{r_2,G} - \theta_{r_3,G})$.

Crossover: In this step, the D-dimensional trial vector

$$\mu_{i,G+1} = (\mu_{1i,G+1}, \mu_{2i,G+1}, \ldots, \mu_{Di,G+1})$$

is formed so that

$$\mu_{ji,G+1} = \begin{cases} \nu_{ji,G+1} & \text{if } randb(j) \leqslant CR \text{ or } j = rnbr(i) \\ \theta_{ji,G} & \text{if } randb(j) > CR \text{ and } j \neq rnbr(i) \end{cases} \tag{6.2}$$

In equation (6.2), $randb(j)$ is the jth evaluation of a uniform random number generator with outcome in $[0, 1]$. CR is the crossover constant in $[0, 1]$ which is determined by user. $rnbr(i)$ is a random chosen integer index $\in [1, D]$ which ensures that $\mu_{i,G+1}$ gets at least one parameter from $\nu_{ji,G+1}$.

Selection: If vector $\mu_{i,G+1}$ is better than $\theta_{i,G}$, then $\theta_{i,G+1}$ is set to $\mu_{i,G+1}$. Otherwise, the old value $\theta_{i,G}$ is retained as $\theta_{i,G+1}$.

6.3 Proposed Evolutionary Extreme Learning Machine (E-ELM) Algorithm

Since the ELM just randomly chooses the input weights and hidden biases, a lot of learning time traditionally spent in tuning these parameters is saved.

However, as the output weights are computed based on the prefixed input weights and hidden biases, there may exist a set of nonoptimal or unnecessary input weights and hidden biases. ELM may require more hidden neurons than conventional tuning based learning algorithms in some applications, which may make ELM respond slowly to unknown testing data. Thus, one may expect more compact networks in the applications which requires faster response of the trained networks.

In this section, a hybrid approach named **e**volutionary **e**xtreme **l**earning **m**achine (E-ELM) using DE and MP generalized inverse is proposed. First, we randomly generate the population. Each individual in the population is composed of a set of input weights and hidden biases

$$\begin{aligned}\theta &= [w_{11}, w_{12}, \cdots, w_{1\tilde{N}}, w_{21}, w_{22}, \cdots, w_{2\tilde{N}}, \\ &\quad \cdots, w_{n1}, w_{n2}, \cdots w_{n\tilde{N}}, \cdots, b_1, b_2, \cdots, b_{\tilde{N}}]\end{aligned}$$

All w_{ij} and b_j are randomly initialized within the range of [-1,1].

Secondly, for each individual (a set of weights and biases), the corresponding output weights are analytically computed by using the MP generalized inverse as done in ELM instead of any iterative tuning [111].

Then the fitness of each individual is evaluated. Suppose that the cost function is root-mean-squared-error (RMSE)

$$E = \sqrt{\frac{\sum_{j=1}^{N} \| \sum_{i=1}^{\tilde{N}} \beta_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) - \mathbf{t}_j \|_2^2}{m \times N}} \tag{6.3}$$

In the normal case, the RMSE on the whole training dataset is used as the fitness. However, it may cause the networks to overfit. In BP, a validation dataset, which normally has no overlap with the training dataset, is used to avoid the overfitting. In addition, as the β is the minimum norm least square solution to the training dataset already, the training error with respect to different individual should be very similar. Therefore, in order to save time, we set the fitness to the RMSE on the validation set only instead of the whole training set as used in [111].

After the fitness of all individuals in the population is calculated, we apply the three steps of DE: mutation, crossover and selection. In DE, during selection, the mutated vectors are compared with the original ones, and the vectors with better fitness values are retained to the next generation. However, for neural networks training, using the fitness (validation RMSE) alone as the selection criteria is not appropriate. A small validation error does not necessarily lead to a small testing error, which largely depends on the distribution of the validation data. As analyzed by Bartlett [113], networks tend to have better generalization performance with smaller weights. Therefore, to further improve the generalization performance, we add one more criteria into the selection: the norm of output weights $\|\beta\|$. In our new selection strategy, when the difference of the fitness between different individuals is small, the one resulting in smaller $\|\beta\|$ is selected. The determination of new

population $\theta_{i,G+1}$ can be described as follows:

$$\theta_{i,G+1} = \begin{cases} \mu_{i,G} & \text{if } f(\theta_{i,G}) - f(\mu_{i,G}) > \epsilon f(\theta_{i,G}) \\ \mu_{i,G} & \text{if } |f(\theta_{i,G}) - f(\mu_{i,G})| < \epsilon f(\theta_{i,G}) \text{ and } \|\beta^{\mu_{i,G}}\| < \|\beta^{\theta_{i,G}}\| \\ \theta_{i,G} & \text{else} \end{cases} \tag{6.4}$$

where $f(\cdot)$ is the fitness function (validation error) and ϵ is a preset tolerance rate. Once the new population is generated, repeat the same DE process until the goal is met or a preset maximum learning epochs is completed.

6.4 Experimental Results

In this section, we compare the following algorithms: E-ELM, BP, ELM, GALS (cf. [111]), MRAN for RBF, SVM and another hybrid DE-BP. DE-BP is much alike to the proposed E-ELM except that the output weights are trained by BP learning algorithm instead of MP generalized inverse. All the smilations run on a same PC with Pentium 4 3GHz CPU and 1GB RAM. All the algorithms except SVM are implemented in MATLAB. Although there are many variants of BP algorithm, the Levenberg-Marquardt learning algorithm is used in our simulations. As mentioned in Chapter 2 and tested on many benchmarking applications among all traditional BP learning algorithms, the Levenberg-Marquardt algorithm appears to be the fastest method for training moderate-sized feedforward neural networks (up

to several hundred weights). There is a very efficient implementation of Levenberg-Marquardt algorithm provided by MATLAB package, which has been used in our simulations for BP and DE-BP. MRAN is one of the most effective algorithms which reduce the number of hidden nodes by intelligently choosing hidden nodes and there centers. The simulations for SVM are carried out using compiled C-coded SVM packages: LIBSVM[1] running in the same PC. This C-coded SVM packages is an implementation based three key works of fast SVMs: original SMO by Platt[89], SMO's modification by Keerthi et al[90], and SVM^{Light} by Joachims[91]. Thus it is one of the fastest SVM libraries. The kernel function used in SVM is radial basis function whereas the activation function used in our proposed algorithms is a simple sigmoidal function $g(x) = 1/(1 + \exp(-x))$.

The parameters of the DE used in the E-ELM and DE-BP are set as follows: Population size NP is set to twice the number of networks' parameters; F and CR are set to 1 and 0.8 respectively. In every simulation, we gradually increase the hidden neurons, and select the results with best generalization performance as the final one. All the results shown in this section are the mean values of 50 trails for ELM and GALS and 5 trials for the other algorithms as they are relatively slow. The inputs of all cases are normalized into $[-1, 1]$.

We first simply try to reconstruct the sigmoid function $f(x) = 1/(1+e^{-x})$

[1]SVM Source Codes: http://www.csie.ntu.edu.tw/~cjlin/libsvm/

Algorithm	Training Time(s)	RMSE		Hidden Neurons
		Training	Testing	
E-ELM	1.18	5.8193e-5±3.9384e-7	8.6764e-5±5.1838e-7	1
ELM	0.0016	6.8591e-5±3.7340e-7	7.0865e-5±3.9776e-6	10
DE-BP	290.313	2.2948e-4±1.9774e-5	4.6584e-4±2.9837e-5	1
BP	12.297	4.9961e-5±1.7083e-6	7.6952e-5±8.2876e-6	1
GALS	1.328	4.3848e-4±5.8382e-7	4.4913e-4±1.8371e-5	1

Table 6.1: Results of regression problem: to approximate a sigmoid function

using the proposed E-ELM. Theoretically, only one hidden neuron is needed to learn this function. 100 observations are randomly generated for training, testing and validation datasets, respectively. The results can been seen in Table 6.4. The E-ELM, BP, DE-BP and GALS are able to achieve a very small RMSE with single hidden neuron, whereas the learning time used by the E-ELM and GALS is much less than the BP and DE-BP. Although the ELM can achieve a similar accuracy at the fastest speed, it needs 10 neurons.

The performance of the E-ELM is also tested on four real benchmark classification problems. Although these datasets appeared in the previous chapters, we still show specification of these problems in Table 6.4 for the convenience of the readers. The training and testing are randomly regenerated at each trial of simulations according to Table 6.4 for all the algorithms.

Name	Attributes	Classes	Number of Observations	
			Training	Testing
Diabetes	17	2	202	516
Satellite Image	36	7	4400	2000
Image Segmentation	18	7	1500	820
Shuttle	10	7	43500	14500

Table 6.2: Specification of classification problems

From Table 6.4, we can see that the E-ELM outperforms the other three algorithms in all the simulations except Satellite Image in terms of testing accuracy. DE-BP and SVM achieve slightly higher testing accuracy than the E-ELM on the Satellite Image dataset. Obviously, DE helps to reduce the hidden neurons in both E-ELM and DE-BP as well as to improve the generalization performance. Both E-ELM and DE-BP achieve more compact network than the other algorithms, and even the DE-BP yields a higher testing accuracy than the pure LM. Both MRAN and SVM are using some optimization methods to intelligently choose the hidde nodes/suport vectors, they still require more storage (hidden nodes/support vectors) than the multilayer feedforward networks even without any optimization method (ELM and LM), let alone E-ELM and DE-BP. It is worth pointing out that SVM generally needs more storage (support vectors) than neural networks (hidden neurons) which resulting its longer prediction time. Although DE

prolongs the training process, the training time of the E-ELM is still reasonable which is still much less than the time needed by BP, GALS and MRAN (RBF) in all the simulations. One should keep it in mind that the Levenberg-Marquardt used here is one of the fastest gradient-based learning algorithms. Unfortunately, the training of GALS on Shuttle case could not be completed as it ran out of memory during the QR factorization step with 10 hidden neurons only, while all the other algorithms are successfully tested on this dataset with even higher number of hidden neurons.

6.5 Summary

In this chapter, we proposed a novel learning algorithm named evolutionary extreme learning machine (E-ELM) which makes use of the advantages of both ELM and DE. It uses the fast minimum norm least-square scheme to analytically determine the output weights instead of tuning, while a modified form of DE is used to optimize the input weights and hidden biases. Unlike the gradient-based methods, the proposed E-ELM does not require the activation functions to be differentiable, implying that E-ELM can be used to train SLFNs with many nonlinear hidden units such as threshold units which are claimed to be easier for hardware implementation. Experimental results show that E-ELM generally achieves higher generalization performance than other algorithms including BP, GALS and the original

ELM. The results also indicate that GALS is relatively slow and need more memory due to the large storage and computational complexity involved in computing the QR factorization. While doing simulations of GALS, the networks are easily over-trained due to GALS uses the training RMSE only as the fitness, whereas E-ELM considers both validation RMSE and norm of weights during searching to achieve a better generalization performance. Hence, GALS is sensitive to the population size and the number of learning epochs. Compared with conventional gradient-based BP algorithms and GALS, E-ELM has faster learning speed and higher testing accuracy, while compared with the ELM algorithm, E-ELM can obtain much more compact network architecture which would increase the response speed and be helpful in fast response (to unknown testing data) applications. Despite of the above advantages that DE brings, it also makes the training of E-ELM more time-consuming and thus E-ELM is not suitable for applications requiring quick learning. However, the applications needing fast prediction and compact networks will benifit by using the proposed E-ELM.

Problem	Algorithm	Training Time(s)	Accuracy(%)		Hidden Neurons
			Training	Testing	
Diabetes	E-ELM	1.2936	82.36±1.28	80.69±1.83	10
	ELM	0.0116	78.68±1.18	78.2±2.65	20
	DE-BP	864.31	79.47±1.51	78.33±2.09	10
	BP	3.0116	86.63±1.7	74.73±3.2	20
	GALS	86.76	78.45±1.59	75.36±2.61	10
	MRAN (RBF)	127.81	77.78±1.38	78.13± 1.93	94.2
	SVM	0.1860	78.76±0.91	77.31±2.35	317.16
Satellite Image	E-ELM	1569.27	92.39±0.97	88.46±1.36	90
	ELM	14.9164	93.52±1.46	89.04±1.5	500
	DE-BP	36924	93.58±1.26	85.31±2.14	80
	BP	12561	95.26±0.96	82.34±1.25	100
	GALS	29209.4	88.48±1.17	86.5±2.06	80
	MRAN (RBF)	24228	83.82±0.86	81.05±1.67	75.4
	SVM	2010.4	91.61±0.58	88.47±1.26	1028.5
Image Segmentation	E-ELM	154.1	96.37±0.49	95.27±1.56	70
	ELM	1.4015	97.35±0.32	95.01±0.78	200
	DE-BP	13196.6	97.02±0.86	88.65±2.05	80
	BP	4745.7	97.35±0.32	86.27±1.80	100
	GALS	3423.7	95.67±0.89	94.27±1.95	80
	MRAN (RBF)	1578.8	89.18±0.79	88.76±1.32	57.6
	SVM	753.9	90.12±0.32	87.51±0.84	394.2
Shuttle	E-ELM	1336.9	99.63±0.11	99.53±0.12	20
	ELM	5.740	99.65±0.12	99.40±0.12	100
	DE-BP	10892.3	99.72±0.1	99.33±0.09	20
	BP	6132.2	99.77±0.1	99.27±0.13	50
	GALS	out of memory			
	MRAN (RBF)	28373.4	98.92±0.09	98.79±0.11	493.4
	SVM	1849.2	99.77±0.03	99.51±0.13	8310.2

Table 6.3: Results of classification problems

Chapter 7

Bagging Extreme Learning Machine (B-ELM)

In this chapter, the ELM is used to construct neural network ensembles which can achieve better generalization performance.

7.1 Introduction

Many methods for constructing ensembles have been developed. *Bagging* [114] is one of the most straightforward methods to create accurate ensemble which trains each individual predictor in its ensemble on a random redistribution of the training set with replacement. Breiman [114] claimed that bagging is effective for unstable learning algorithms such as neural networks

and decision trees where small changes in the training set may result in large changes in prediction.

It was shown in Opitz and Maclin [115] that the neural network ensembles are superior to individual neural networks in terms of generalization accuracy. However, the extremely long training time becomes the bottleneck of its application when the conventional gradient-based learning algorithms are applied. As introduced in Chapter 3, the ELM not only learns much faster with higher generalization performance than the traditional gradient-based learning algorithms but also avoids many difficulties faced by gradient-based learning methods such as stopping criteria, learning rate and local minima.

In this chapter, a new algorithm to construct bagging ensembles with ELMs is proposed for the applications which needs higher accuracy and stability but is less strict on the computational and time cost.

7.2 Bagging Predictors

Bagging [114] is a method for generating multiple versions of a predictor and using these predictors to obtain an aggregated predictor. Suppose there is a learning set $\mathcal{L}$ consisting of training data $\{(y_i, \mathbf{x}_i), i = 1, \cdots, N\}$ where $\mathbf{x}_i$ denotes the input and y_i denotes corresponding target which is either a class label or a numerical value. $y = \varphi(\mathbf{x}, \mathcal{L})$ denotes the predictor trained on this

learning set $\mathcal{L}$.

Bagging takes repeated bootstrap samples $\{\mathcal{L}^{(B)}\}$ from $\mathcal{L}$, and form an ensemble of predictors $\{\varphi(\mathbf{x}, \mathcal{L}^{(B)})\}$. $\{\mathcal{L}^{(B)}\}$ form replicated datasets, each consisting of N cases, drawn at random, but with replacement, from $\mathcal{L}$. It means each $(y, \mathbf{x}_i)$ may appear repeatedly or not at all in any particular $\{\mathcal{L}^{(B)}\}$. If y is numerical, the output of the ensemble φ_B is $\varphi_B = \frac{\sum \varphi(\mathbf{x}, \mathcal{L}^{(B)})}{K}$, where K is the number of bootstraps. If y is a class label, let the $\varphi(\mathbf{x}, \mathcal{L}^{(B)})$ vote to form φ_B. This procedure is called "**b**ootstrap **agg**regat**ing**", and that is where the name "bagging" comes.

7.3 Constructing Bagging ELMs (B-ELM)

Many researchers [114][115][116] have shown that bagging is able to improve the accuracy of neural networks. However, the training time of bagging neural networks becomes the bottleneck in its application. When conventional gradient-based learning algorithms are used, the training process can be tremendously long as one ensemble usually consists of tens of neural networks. As we mentioned in the previous section, the newly proposed ELM is able to achieve satisfactory generalization accuracy within much shorter learning time than neural networks trained by gradient-based learning algorithms such as the back-propagation, it is very promising to construct the neural networks ensemble by using ELMs to significantly shorten the total

training time.

When constructing the bagging ELMs (B-ELM), three factors affecting the generalization performance should be taken into account. The first factor is the number of bootstraps or ELMs used to construct the ensemble. Choosing the number of bootstraps is a tradeoff between the time complexity and stability of the generalization accuracy. The more bootstraps the ensemble has, the more stable it predicts, but obviously more training and testing time is expected. Most researchers [115][116] suggest that around twenty bootstraps should be enough, which will also studied empirically in Section 7.4. The second issue to be considered is how many unique training samples should be contained in each bootstrap. Although every bootstrap has the same number of training samples as the original training set, some samples may appear in one bootstrap more than once. Obviously, the less unique training samples each bootstrap contains, the larger differences the bootstraps have. As Breiman [114] claimed, the diversity of the bootstraps really helps to improve the generalization accuracy of the bagging predictor. However, inadequate training samples in each bootstraps can result in low accuracy achieved by each neural network, and finally cause the bagging ensemble to have a poor generalization accuracy. Fortunately, this issue has already been thoroughly studied in literature [115][116] and these researchers suggest that bootstraps consisting of 2/3 samples from the original training set look better. The third issue is on the appropriate number of hidden neu-

rons in each SLFN. This is a common issue when using neural networks. One of the most popular methods is using cross-validation. There are two kinds of cross-validation strategies: 1) to cross-validate each SLFN, which means that different SLFNs in the ensemble may have different number of hidden neurons; or 2) to assign the same number of hidden neurons to all the SLFNs and then to use the final accuracy of the ensemble on the validation set as the selection criteria. The first strategy should be able to obtain a better architecture for every SLFN but more time-consuming than the latter. However, in our experience, performance of the ELM is quite stable once the number of hidden neurons reaches a certain level. Thus, we choose the latter strategy in order to save the validation time.

7.4 Experimental Results

In this section, we compare the bagging ELMs (B-ELM) with bagging BP (B-BP) and single ELM. Both B-ELM and B-BP have 25 bootstraps in each ensemble, and 2/3 samples in each bootstrap are unique. All the smilations run on a same PC with Pentium 4 3GHz CPU and 1GB RAM. All the algorithms are implemented in MATLAB. Although there are many variants of BP algorithm, the Levenberg-Marquardt learning algorithm is used in our simulations. As mentioned in Chapter 2 and tested on many benchmarking applications among all traditional BP learning algorithms, the Levenberg-

Name	Attributes	Classes	Number of Observations	
			Training	Testing
Glass	9	7	110	104
Letter Recognition	16	26	10000	10000
Ionosphere	33	2	175	176
Liver Disorders	6	2	170	175
Page Blocks	10	5	2700	2775
Pima	8	2	380	388
Heart	13	2	135	135
Vehicle	18	4	420	426
German	24	2	500	500
Image Segmentation	19	7	1100	1210
Satellite Image	36	6	4400	2035

Table 7.1: Specification of datasets

Marquardt algorithm appears to be the fastest method for training moderate-sized feedforward neural networks (up to several hundred weights). There is a very efficient implementation of Levenberg-Marquardt algorithm provided by MATLAB package, which has been used in our simulations for BP.

Various real-world benchmarking classification problems are tested. The specification of these datasets are listed in Table 7.3. Unsurprisingly, B-ELM outperforms single ELM on all the datasets as shown in Table 7.3 in terms of

Dataset	B-ELM (%)		B-BP (%)		ELM (%)	
	Accuracy	Dev	Accuracy	Dev	Accuracy	Dev
Glass	66.421	1.106	66.292	1.093	63.738	3.861
Letter Recognition	93.797	0.913	85.767	1.332	93.001	3.724
Ionosphere	89.375	1.166	89.258	1.355	85.852	4.102
Liver Disorders	71.029	1.162	70.797	1.297	67.71	3.482
Page Blocks	95.9	0.855	96.76	1.077	95.383	2.151
Pima	77.191	0.825	77.007	0.818	75.284	1.687
Heart	84.296	0.958	83.577	1.737	79.556	3.988
Vehicle	82.6767	1.13	82.255	1.266	78.732	2.881
German	76.58	1.177	76.533	0.92	73.62	2.195
Image Segmentation	96.463	0.375	97.339	0.347	94.57	0.841
Satellite Image	90.1	0.344	89.922	0.397	89.56	0.921

Table 7.2: Comparing testing accuracy and standard deviation (Dev) of B-ELM, B-BP and single ELM

prediction accuracy and its standard deviation (Dev), which indicates bagging method is really helpful to improve the generalization performance of the ELM. In addition, thanks to the inherent superiority of the ELM algorithm, B-ELM generally achieves better generalization performance (testing accuracy) than the B-BP. Seen from Table 7.4, we can find that bagging sacrifices time for stability and accuracy when we compare the bagging ELM with a

Dataset	B-ELM		B-BP		ELM	
	Neurons	Time (s)	Neurons	Time (s)	Neurons	Time (s)
Glass	25	0.078125	15	21.3367	25	0.003125
Letter Recognition	1000	2919.7	100	3977.2	1000	86.403
Ionosphere	25	0.084375	10	45.135	35	0.0063
Liver Disorders	25	0.78118	10	7.7343	25	0.0016
Page Blocks	80	2.6422	30	2435.2	120	0.23125
Pima	15	0.054688	10	11.969	20	0.0046
Heart	10	0.026562	5	86.773	10	0.0016
Vehicle	100	1.1812	80	3431.4	180	0.2031
German	50	0.4375	40	942.79	50	0.0235
Image Segmentation	150	7.5906	60	12156	160	0.31094
Satellite Image	500	174.7	50	1588.5	500	9.2718

Table 7.3: Comparing number of hidden neurons and training time of B-ELM, B-BP and single ELM

single ELM, nevertheless the training time of the B-ELM is still much shorter than that of the B-BP. One should consider the fact that the LM used here is one of the fastest gradient-based learning methods. Hence, the sacrifice is reasonable and worthwhile especially to those applications which need to strike a balance between time and accuracy. Furthermore, we study the dependence of testing accuracy and training time on the number of bootstraps.

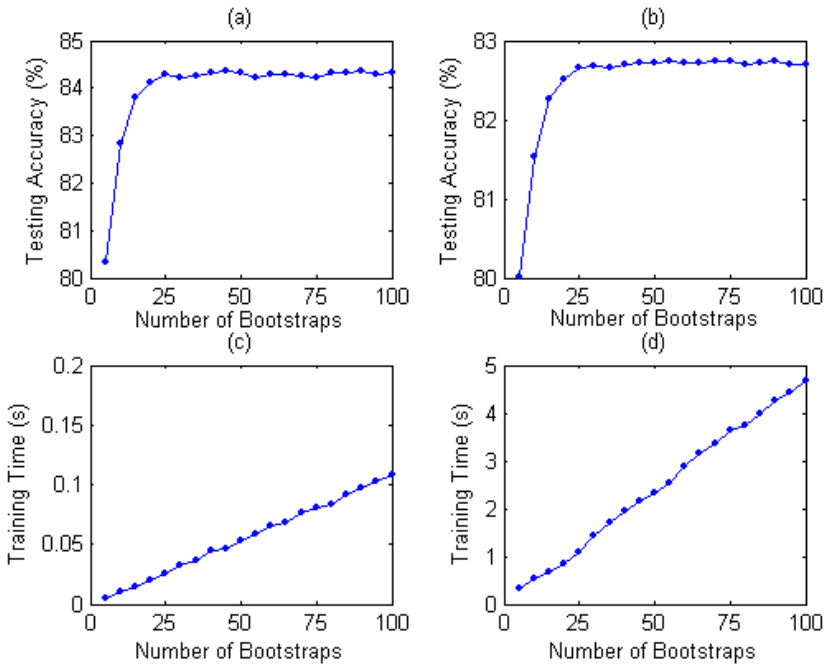


Figure 7.1: Dependence of testing accuracy and training time of B-ELM on the number of bootstraps: (a) Testing accuracy of Heart; (b) Testing accuracy of Vehicle; (c) Training time of Heart; (d) Training time of Vehicle.

For each dataset, we gradually increase the number of bootstraps from 5 to 100 in step of 5 and test the performance of the B-ELM. The results of problems Heart and Vehicle are shown here as examples.

As shown in Figures 7.1(a) and (b), when there are less than 25 bootstraps, the testing accuracy increase dramatically with the number of bootstraps. However, when bootstraps are more than 25, there is little improvement in the generalization performance. As for the training time, it increases linearly with the number of bootstraps as shown in Figures 7.1(c) and (d). Obviously 25 is the optimal number of bootstraps for both generalization

performance and training time, which is also compliant with the conclusions of other papers.

7.5 Summary

In this chapter, a new algorithm to construct bagging ensembles with extreme learning machine is proposed. The experimental results indicate that the proposed algorithm outperforms the conventional bagging neural networks ensembles trained by gradient based learning methods in terms of generalization performance and training time. The selection of the number of bootstraps and the bootstrap size has also been discussed. However, the B-ELM is not suitable for time-critical applications although its learning time is much shorter than other neural networks ensembles. In fact, the B-ELM is suitable for applications needing relatively high accuracy.

Chapter 8

Conclusions and Recommendations

8.1 Conclusions

Neural networks are extensively used in the areas of classification, pattern recognition and regression. The widely used conventional gradient-based learning algorithms for feedforward neural networks iteratively update the weights of the networks to minimize the cost function. The iterative updating strategy and the expensive computational cost involved during computing the gradient and performing the line search make the training process of these algorithms slow. However, nowadays more and more applications require the learning algorithms to have fast learning capabilities. To meet the needs

of these applications, this thesis has made some important contributions as presented in Chapter 3 to 7. The contributions of this thesis are summarized as follows:

- A novel and fast learning algorithm namely the extreme learning machine (ELM) for SLFN has been proposed in Chapter 3. The input weights and biases are randomly selected and the output weights are analytically determined by using of the MP generalized inverse. The learning capability of the ELM has been rigorously proved. In Chapter 3, the simulation results has shown that the proposed ELM can not only learn much faster than conventional learning algorithms but also achieve better generalization performance in most cases. The ELM is also easier to use. It does not have the issues like local minima and stopping criterion which are commonly faced by gradient-based learning algorithms. In addition, the gradient-based learning algorithms can only be used to train networks with differentiable activation functions, whereas the ELM are applicable for most activation functions.

- We have proved that the ELM can be also used to train neural networks with threshold units where the conventional gradient-based learning algorithms cannot be directly applied as the threshold activation functions are non-differentiable. The simulation results in Chapter 4 have shown that the ELM are able to achieve better generalization performance than the existing learning algorithms for threshold networks

with much faster learning speed.

- An extremely fast learning algorithm namely the modular implementation of extreme learning machine (MI-ELM) for TLFNs has been proposed in chapter 5. In essence, it is an extension of ELM for TLFNs based on the constructive method proposed in [18]. Three improvements have been made in the proposed new algorithm: 1) New selection criteria for the biases of neural quantizer has been proposed to improve the generalization performance; 2) An automatic method to select the appropriate values of the quantizer factors T and U has been introduced; 3) The MP generalized inverse has been used to compute the weights β instead of the normal inverse. Hence, the hidden neurons in the first hidden layer of the sub-TLFN can be any natural number $\tilde{N} \leqslant N/L$ which not only shrinks the network size but also improves the generalization performance. The simulation results in Chapter 5 have shown that the MI-ELM can not only reach good generalization performance but also provide very short learning and prediction time in many cases, which may be within most users' expected time in many applications. In other words, the proposed MI-ELM algorithm or Huang's network would provide fast alternative solutions to time-critical applications.

- In addition to increasing the fast learning capability of the feedforward neural networks, this thesis has also managed to shrink the network size

for those storage concerned or prediction time concerned applications such as embedded systems. A hybrid learning algorithm namely the evolutionary extreme learning machine (E-ELM) has been proposed in Chapter 6 which uses the DE algorithm to optimize the input weights and biases of SLFN and MP generalized inverse to determine the output weights. The network achieved by the E-ELM is more compact than those needed by the ELM although the learning time of the E-ELM is longer than ELM. The simulation results in Chapter 6 has also shown that the E-ELM tends to achieve better generalization performance than the ELM and gradient-based learning algorithms.

- The ELMs are used to construct neural network (NN) ensemble using the bagging method. Thanks to the inherent fast learning capability of the ELM, the learning of B-ELM is much faster than the traditional NN ensembles which make it more practical in real-world applications. The simulation results in Chapter 7 has shown that the B-ELM outperforms the single ELM in terms of the generalization performance and stability. Hence, B-ELM is suitable for the applications which has no restriction in time but requires high generalization accuracy.

8.2 Recommendations for Further Research

A very fast learning algorithm namely the ELM and its variant for various applications were proposed in this thesis. During our research, we found some issues are interesting and need further investigation. Hence, the following directions are recommended for future research:

- As discussed in Chapter 3, the computational complexity of ELM is $O(N\tilde{N}^2)$. If its computational complexity can be further reduced, this algorithm can be even faster. One way to achieve it is to investigate fast algorithms computing the Moore-Penrose generalized inverse.

- The ELM is proposed for the applications having time-critical learning requirement, while the E-ELM focuses on shrinking the network size and improving the generalization performance. Although E-ELM still learns faster than conventional gradient-based learning algorithms or evolutionary learning algorithms, its training time is prolonged compared with the ELM learning algorithm. Hence, it will be better if a more efficient global optimization method than DE can be found such that the new learning algorithm can achieve fast learning speed and compact network size at the same time.

- In general, the ELM is suitable for SLFNs with different kinds of activation functions. In this thesis, the performance of the ELM with sigmoid and threshold activation functions was thoroughly investigated.

Huang, et al [99, 100] also showed the performance of the ELM with RBF hidden neurons. However, there are still quite some other kinds of activation functions yet to be tested. In the future, other types of activation functions may be tested to broaden the application of the ELM.

- As the focus of this thesis is to propose fast learning algorithms for multi-layer feedforward neural networks, the learning algorithms reviewed and compared in this thesis are mainly gradient-based learning algorithms. We understand that many faster learning methods have been proposed such as fast RBF learning algorithms and kernel learning methods. Although some results of the SVM and RBF have been shown in this thesis, it will be interesting that more systematic comparisons with different kinds of learning machine can be made in the future.

- The algorithms proposed in this thesis are for the batch learning purposes only. We noted that there are increasing demands for on-line learning algorithms. It is a promising direction to propose a fast on-line learning algorithm based on the ELM algorithm.

- Overall, this thesis focuses on regression problems. The classification capability of the ELM still needs further investigation. In addition, more practical applications is desired to test the performance of the ELM in the future.

Author's Publications

1. Guang-Bin Huang, Qin-Yu Zhu, K. Z. Mao, Chee-Kheong 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.

2. Qin-Yu Zhu, A. K. Qin, P. N. Suganthan, and Guang-Bin Huang, "Evolutionary Extreme Learning Machine", *Pattern Recognition*, vol. 38, pp. 1759-1763, 2005.

3. Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew, "Real-Time Learning Capability of Neural Networks," *IEEE Transactions on Neural Networks* (in press), vol. 17, no. 4, 2006.

4. Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew "Extreme Learning Machines: Theory and Applications," *Neurocomputing*, vol. 70, pp. 489-501, 2006.

5. Qin-Yu Zhu, Guang-Bin Huang, and Chee-Kheong Siew, "A Fast Modular Implementation of Neural Networks," in *Proceedings of The Eighth International Conference on Control, Automation, Robotics and Vision (ICARCV 2004)*, Kunming, China, Dec 6-9, 2004.

6. Qin-Yu Zhu, Guang-Bin Huang, and Chee-Kheong Siew, "A Fast Constructive Learning Algorithm Single-Hidden Layer Neural Networks,"

in *Proceedings of The Eighth International Conference on Control, Automation, Robotics and Vision (ICARCV 2004)*, Kunming, China, Dec 6-9, 2004.

7. Guang-Bin Huang, Qin-Yu Zhu, and Chee-Kheong Siew, "Extreme Learning Machine: A New Learning Scheme of Feedforward Neural Networks," in *Proceedings of 2004 International Joint conference on Neural Networks (IJCNN'2004)*, Budapest, Hungary, July 25-29, 2004.

8. Qin-Yu Zhu, Guang-Bin Huang, and Chee-Kheong Siew, "Realtime Two-Hidden Layer Feedforward Neural Networks," in *Proceedings of The Fourth International Conference on Control and Automation (ICCA'03)*, Montreal, Canada, June 9-12, 2003.

Bibliography

[1] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, pp. 359–366, 1989.

[2] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989.

[3] K. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Networks*, vol. 2, pp. 183–192, 1989.

[4] 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.

[5] 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.

[6] 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.

[7] G.-B. Huang, Y.-Q. Chen, and H. A. Babri, "Classification ability of single hidden layer feedforward neural networks," *IEEE Transactions on Neural Networks*, vol. 11, no. 3, pp. 799–801, 2000.

[8] D. E. Rumellhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," *Parallel Distribution Processing: Explanations in the Microstructure of Cognition*, vol. 1, pp. 318–362, 1986.

[9] S. Singhal and L. Wu, "Training multilayer perceptrons with the extended kalman algorithm," *Advances in Neural Information Processing Systems*, vol. 1, pp. 133–140, 1989.

[10] D. J. Lary and H. Y. Mussa, "Using an extended kalman filter learning algorithm for feed-forward neural networks to describe tracer correlations," *Atmospheric Chemistry and Physics*, vol. 4, pp. 3653–3666, 2004.

[11] R. E. Kalman, "A new approach to linear filtering and prediction problems," *Transactions of the ASME–Journal of Basic Engineering*, vol. 82, no. Series D, pp. 35–45, 1960.

[12] V. Vapnic, *The Nature of Statistical Learning Theory.* Springer, New Yok, 1995.

[13] J. Platt, "Sequential minimal optimization: A fast algorithm for training support vector machines," in *Microsoft Research Technical Report MSR-TR-98-14*, 1999.

[14] J. Suykens, T. V. Gestel, J. D. Brabanter, B. D. Moor, and J. Vandewalle, *Least Squares Support Vector Machines.* World Scientific, Singapore, 2002.

[15] S. Ferrari and R. F. Stengel, "Smooth function approximation using neural networks," *IEEE Transactions on Neural Networks*, vol. 16, no. 1, pp. 24–38, 2005.

[16] 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.

[17] M. T. Hagan and M. B. Menhaj, "Training feedforward networks with the marquardt algorithm," *IEEE Transactions on Neural Networks*, vol. 5, no. 6, pp. 989–993, 1994.

[18] 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.

[19] P. J. Werbos, "Beyond regression: new tools for prediction and analysis in the behavioral science," in *Master's Thesis*, Harvard University, 1974.

[20] D. B. Parker, "Learning logic: casting the cortex of the human brain in silicon," in *MIT Technical Report TR-47*, 1985.

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

[22] M. T. Hagan, H. B. Demuth, and M. H. Beale, *Neural Network Design*. Boston, MA: PWS Publishing, 1996.

[23] 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.

[24] H. A. C. Eaton and T. L. Olivier, "Learning coefficient dependence on training set size," *Neural Networks*, vol. 5, no. 2, pp. 283–288, 1992.

[25] M. Moreira and E. Fiesler, "Neural networks with adaptive learning rate and momentum terms," in *Technical Report*, (Martigny, Switzerland), Institut Dalle Molle D'Intelligence Artificielle Perceptive, 1995.

[26] C. Darken and J. Moody, "Note on learning rate schedules for stochastic optimization," in *Neural Information Processing Systems* (R. P. Lippmann, J. E. Moody, and D. S. Touretzky, eds.), (San Mateo, CA), pp. 832–838, Morgan Kaufmann, 1991.

[27] 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.

[28] R. Salomon, "Adaptive learning rate for back-propagation," in *Technical Report*, University Berlin, 1989.

[29] L. W. Chan and F. Fallside, "An adaptive training algorithm for backpropagation networks," *Computer Speech and Language*, vol. 2, pp. 205–218, 1987.

[30] J. Schmidt, A. Siegel, and A. Srinivasan, "Chernoff-hoeffding bounds for applications with limited independence," *SIAM Journal on Discrete Mathematics*, vol. 8, no. 2, pp. 223–250, 1995.

[31] F. M. Silva and L. B. Almeida, "Speeding up backpropagation," in *Advanced Neural Computers* (R. Eckmiller, ed.), (Amsterdam, Holland), pp. 151–158, 1990.

[32] T. Tollenaere, "Supersab: Fast adaptive backpropagation with good scaling properties," *Neural Networks*, vol. 3, pp. 561–573, 1990.

[33] R. A. Jacobs, "Increased rates of convergence through learning rate adaption," *Neural Networks*, vol. 1, pp. 295–307, 1988.

[34] R. Martin and H. Braun, "A direct adaptive method for faster backpropagation learning: The RPROP algorithm," in *Proceedings of the*

IEEE International Conference on Neural Networks, pp. 586–591, San Francisco, CA, 1993.

[35] M. Riedmiller and H. Braun, "Rprop – description and implementation details," in *Technical Report*, Universitat Karlsruhe, 1994.

[36] C. Anderson, "Learning and problem solving with multilayer connectionist systems," in *Technical Report COINS TR 86-50*, (Amherst, MA), University of Massachusetts, 1986.

[37] S. E. Fahlman, "An empirical study of learning speed in back-propagation networks," in *Technical Report CMU-CS-88-162*, 1998.

[38] S. Waugh and A. Adams, "A practical comparison between quickprop and back-propagation," in *Proceedings of the Eighth Australian Conference on Neural Networks (ACNN'97)*, (Melbourne), 1997.

[39] M. Vrahatis, G. Magoulas, and V. Plagianakos, "Convergence analysis of the quickprop method," in *Proceedings of the International Joint Conference on Neural Networks (IJCNN'99)*, (Washington DC), 1999.

[40] S. Amari, N. Murata, K.-R. Müller, M. Finke, and H. Yang, "Statistical theory of overtraining — is cross-validation asymptotically effective?," in *Advances in Neural Information Processing Systems* (D. S. Touretzky, M. C. Mozer, and M. E. Hasselmo, eds.), vol. 8, pp. 176–182, The MIT Press, 1996.

[41] D. Luenberger, *Linear and Nonlinear Programming.* Massachusetts: Addison Wesley, Reading, 1984.

[42] R. Fletcher, *Practical Methods of Optimization.* New York: Wiley, 1987.

[43] B. Cornet, *Nonlinear Analysis and Optimization.* North-Holland, 1987.

[44] R. Fletcher and C. M. Reeves, "Function minimization by conjugate gradients," *Computer Journal*, vol. 7, pp. 149–154, 1964.

[45] G. H. Golub and C. F. VanLoan, *Matrix Computations.* Baltimore, Maryland: John Hapkins, University Press, 1996.

[46] M. J. D. Powell, "Restart procedures for the conjugate gradient method," *Mathematical Programming*, vol. 12, pp. 241–254, 1977.

[47] E. M. L. Beale, "A derivation of conjugate gradients," in *Numerical methods for nonlinear optimization* (F. A. Lootsma, ed.), London: Academic Press, 1972.

[48] R. P. Brent, *Algorithms for Minimization without Derivatives.* Englewood Cliffs, NJ: Prentice-Hall, 1973.

[49] L. E. Scales, *Introduction to Non-Linear Optimization.* New York: Springer-Verlag, 1985.

[50] C. Charalambous, "Conjugate gradient algorithm for efficient training of artificial neural networks," *IEEE Proceedings*, vol. 39, no. 3, pp. 301–310, 1992.

[51] J. E. Dennis and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Englewood Cliffs, NJ: Prentice-Hall, 1983.

[52] Y. LeCun, P. Simard, and B. Pearlmutter, "Automatic learning rate maximization by on-line estimation of the hession eigenvectors," in *Advances of Neural Information Processing Systems* (S. J. Hanson, J. D. Cowan, and C. L. Giles, eds.), vol. 5, pp. 156–163, San Mateo, CA: Morgan Kaufmann, 1993.

[53] B. Pearlmutter, "Fast exact multiplications by hessian," *Neural Computation*, vol. 6, no. 1, pp. 147–160, 1994.

[54] S. Becker and Y. LeCun, "Improving the convergence of back-propagation learning with second order methods," in *Proceedings of the 1988 Connectionists Models Summer School* (D. Touretzky, G. Hinton, and T. Sejnowski, eds.), (Carnegie-Mellon University), Morgan Kauffmann, 1989.

[55] U. Nowak and L. Weimann, "A family of newton codes for systems of highly nonlinear equations-algorithm," in *Technical Report COINS TR*

90-10, (Berlin), Konradzuse-Zentrum fur Informationstechnik Berlin, 1990.

[56] J. E. Dennis and R. B. Schnabel, *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Englewood Cliffs, NJ: Prentice-Hall, 1983.

[57] S. Haykin, *Kalman Filtering and Neural Networks*. Wiley-Interscience, 2001.

[58] P. L. Bartlett and T. Downs, "Using random weights to train multilayer networks of hard-limiting units," *IEEE Transactions on Neural Networks*, vol. 3, no. 2, pp. 202–210, 1992.

[59] E. M. Corwin, A. M. Logar, and W. J. B. Oldham, "An iterative method for training multilayer networks with threshold function," *IEEE Transactions on Neural Networks*, vol. 5, no. 3, pp. 507–508, 1994.

[60] A. P. Thakoor, A. Moopenn, J. Lambe, and S. K. Khanna, "Electronic hardware implementations of neural networks," *Applied Optics*, no. 23, pp. 5085–5092, 1987.

[61] P. Ienne, "Digital hardware architectures for neural networks," *SPEEDUP Journal*, vol. 9, no. 1, pp. 18–25, 1995.

[62] S. E. Hampson and D. J. Volper, "Representing and learning boolean functions of multivalued features," *IEEE Transactions on Systems, Man & Cybernetics*, vol. 20, pp. 67–80, 1990.

[63] E. B. Baum and D. Haussler, "What size net gives valid generalizations?," *Neural Computation*, vol. 1, pp. 151–160, 1989.

[64] D. J. Toms, "Training binary node feedforward neural networks by backpropagation of error," *Electronics Letters*, vol. 26, no. 21, pp. 1745–1746, 1990.

[65] 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.

[66] J. Platt, "A resource-allocation network for function interpolation," *Neural Computation*, vol. 3, pp. 213–225, 1991.

[67] L. Yingwei, N. Sundararajan, and P. Saratchandran, "A sequential learning scheme for function approximation using minimal radial basis function (rbf) neural networks," *Neural Computation*, vol. 9, pp. 461–478, 1997.

[68] G.-B. Huang, P. Saratchandran, and N. Sundararajan, "An efficient sequential learning algorithm for growing and pruning rbf (gap-rbf) networks," *IEEE Trans Syst Man Cybern B Cybern.*, vol. 34, no. 6, 2284-2292.

[69] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[70] 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, p. 155, The MIT Press, 1997.

[71] A. Smola and B. Schölkopf, "A tutorial on support vector regression," in *NeuroCOLT2 Technical Report NC2-TR-1998-030*, (Royal Holloway College, London, U.K.), 1998.

[72] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Englewood Cliffs, NJ: Prentice-Hall, 1999.

[73] Z. Hao, X. Y. Shu Yu, R. H. Feng Zhao, and Y. Liang, "Online ls-svm learning for classification problems based on incremental chunk," in *Proceedings of International Symposium on Neural Networks*, (Dalian China), pp. 558–564, 2004.

[74] E. Fix and J. Hodges, "Nonparametric discrimination: Consistency properties," in *Report 11*, USAF School of Aviation Medicine, Randolph Field, Texas, 1951.

[75] R. O. Duda and P. E. Hart, *Patten Classification and Scene Analysis*. Wiley, New York, 1973.

[76] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4, pp. 251–257, 1991.

[77] 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.

[78] S. Tamura and M. Tateishi, "Capabilities of a four-layered feedforward neural network: Four layers versus three," *IEEE Transactions on Neural Networks*, vol. 8, no. 2, pp. 251–255, 1997.

[79] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Real-time learning capability of neural networks," in *Technical Report ICIS/45/2003*, (School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore), Apr. 2003.

[80] P. L. Bartlett, "The sample complexity of pattern classification with neural networks: The size of the weights is more important than the size of the network," *IEEE Transactions on Information Theory*, vol. 44, no. 2, pp. 525–536, 1998.

[81] D. Serre, *Matrices: Theory and Applications*. Springer-Verlag New York, Inc, 2002.

[82] C. R. Rao and S. K. Mitra, *Generalized Inverse of Matrices and Its applications*. New York: John Wiley & Sons, Inc, 1971.

[83] G. H. Golub and C. F. V. Loan, *Matrix Computation*. Johns Hopkins University Press, Baltimore, Maryland, 1989.

[84] G.-B. Huang, *Learning Capability of Neural Networks*. Singapore: Ph.D. thesis, Nanyang Technological University, 1998.

[85] J. M. Ortega, *Matrix Theory*. Plenum Press, New York and London, 1987.

[86] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. D. Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen, *LAPACK User's Guide*. Third Edition, SIAM, Philadelphia, 1999.

[87] J. Demmel, "Accurate svds of structured matrices," in *Technical Report CS97 -375, LAPACK Working Note 130*, (Knoxville, TN, USA), Department of Computer Science, University of Tennessee, 1997.

[88] MATLAB, "Neural networks toolbox," in *http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/newrb.html/*, 2004.

[89] J. Platt, "Sequential Minimal Optimization: A fast algorithm for training support vector machines," *Microsoft Research Technical Report MSR-TR-98-14*, 1998.

[90] L. Yingwei, N. Sundararajan, and P. Saratchandran, "S. s. keerthi and s.k. shevade and c. bhattacharyya and k. r. k. murthy," *Neural Computation*, vol. 13, pp. 637–649, 2001.

[91] T. Joachims, "SVM^{light} – support vector machine," in *http://svmlight.joachims.org/*, (Department of Computer Science, Cornell), 2003.

[92] 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.

[93] G. Rätsch, T. Onoda, and K. R. Müller, "An improvement of AdaBoost to avoid overfitting," in *Proceedings of the 5th International Conference on Neural Information Processing (ICONIP'1998)*, 1998.

[94] 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.

[95] Y. Freund and R. E. Schapire, "Experiments with a new boosting algorithm," in *International Conference on Machine Learning*, pp. 148–156, 1996.

[96] D. R. Wilson and T. R. Martinez, "Heterogeneous radial basis function networks," in *Proceedings of the International Conference on Neural Networks (ICNN 96)*, pp. 1263–1267, June 1996.

[97] 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.

[98] R. Collobert, S. Bengio, and Y. Bengio, "A parallel mixtures of SVMs for very large scale problems," *Neural Computation*, vol. 14, pp. 1105–1114, 2002.

[99] 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), 6-9 Dec, 2004.

[100] 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.

[101] 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 (in press)*, vol. 17, no. 3, 2006.

[102] R. M. Goodman and Z. Zeng, "A learning algorithm for multi-layer perceptrons with hard-limiting threshold units," in *Proceedings of the 1994 IEEE Workshop of Neural Networks for Signal Processing*, pp. 219–228, 1994.

[103] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: A new learning scheme of feedforward neural networks," in *Proceedings of 2004 International Joint Conference on Neural Networks (IJCNN'2004)*, (Budapest, Hungary), July 2004.

[104] E. B. Baum, "On the capabilities of multilayer perceptrons," *Journal of complexity*, vol. 4, pp. 193–215, 1988.

[105] D. Michie, D. Spiegelhalter, and C. Taylor, eds., *Machine Learning, Neural and Statistical Classification*. Ellis Horwood, 1994.

[106] V. P. Plagianakos, G. D. Magoulas, N. K. Nousis, and M. N. Vrahatis, "Training multilayer networks with discrete activation functions," in *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN'2001)*, (Washington D.C., U.S.A.), 2001.

[107] P. W. Frey and D. J. Slate, "Letter recognition using holland-style adaptive classifiers," *Machine Learning*, vol. 6, no. 2, pp. 161–182, 1991.

[108] G.-B. Huang, K. Z. Mao, C.-K. Siew, and D.-S. Huang, "Fast modular network implementation for support vector machines," *Accepted by IEEE Transactions on Neural Networks*, 2005.

[109] R. Collobert, Y. Bengio, and S. Bengio, "Scaling large learning problems with hard parallel mixtures," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 17, no. 3, pp. 349–365, 2003.

[110] V. Kadirkamanathan and M. Niranjan, "A function estimation approach to sequential learning with neural networks," *Neural Computation*, vol. 5, pp. 954–975, 1993.

[111] R. Ghosh and B. Verma, "A hierarchical method for finding optimal architecture and weights using evolutionary least square based learning," *International Journal of Neural Systems*, vol. 12, no. 1, pp. 13–24, 2003.

[112] R. Storn and K. Price, "Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.

[113] P. L. Bartlett, "The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network," *IEEE Transactions on Information Theory*, vol. 44, no. 2, pp. 525–536, 1998.

[114] L. Breiman, "Bagging predictor," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[115] D. Optiz and R. Maclin, "Popular ensemble methods: An empirical study," *Journal of Artificial Intelligence Research*, vol. 11, pp. 169–198, 1999.

[116] T. G. Dietterich, "Machine-learning research: Four current directions," *The AI Magazine*, vol. 18, no. 4, pp. 97–136, 1998.