Extreme Learning Machine for Classification and Regression

ZHOU HONGMING

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University

in fulfillment of the requirements for the degree of

Doctor of Philosophy

2013

Statement of Originality

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

. . . .

Date ZHOU HONGMING

Acknowledgments

I would like to express my sincerest gratitude to all those who helped me during the four years of my PhD study.

First of all, I would like to thank my supervisor, Assoc. Prof. Huang Guangbin, for his expert guidance and spiritual encouragement. He has guided me from the beginning in the research field towards now I gain enough expertise to complete my thesis. Without his consistent and illuminating instructions, I could not reach this stage to write the thesis. I am really grateful for all the efforts he has made to help me in the fours years of PhD study.

Secondly, I would like to express my heartful gratitude to my co-supervisor, Assoc. Prof. Lin Zhiping, for advice, discussion and helping me in reviewing and modifying my papers. He has being always prudent in every details. I really appreciate all his generous help.

Thirdly, I want thank my friends Dr. Lan Yuan, Dr. Zhang Rui, Dr. Ding Xiaojian and Dr. Zong Weiwei for their valuable technical discussions and support.

I would also like to thank my friends and lab-mates including Mr. Tan Peng Chye, the technical staff of Machine Learning Research Laboratory, for the fun and joy we have had during the past four years.

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

Table of Contents

Summary

The machine learning techniques have been extensively studied in the past few decades. One of the most common approaches in machine learning is the Artificial Neural Network (ANN), or commonly referred as Neural Network (NN) that is inspirited by how the human brain functions. Many learning algorithms and paradigms have been developed for Neural Network since 1940s. However, most of the traditional neural network learning algorithms suffer from problems like local minima, slow learning rate and trivial human intervene.

Extreme Learning Machine (ELM) proposed by Huang et al. in 2004 is an emergent technology that has great potential of overcoming the problems faced by traditional neural network learning algorithms. ELM is based on the structure of the "generalized" single hidden-layer feedforward neural networks, where the hidden node parameters are randomly generated. In the aspect of the standard optimization method, the ELM problem could be formulated as an optimization problem that is similar to the formulation of Support Vector Machine (SVM)'s optimization problem. However, SVM tends to obtain a solution that is sub-optimal to ELM's.

With the finding of the relationship between ELM and SVM, we could extend ELM to many of SVM's variants. In the work presented in chapter 3, the equality constrained approach from both Least Square SVM and Proximal SVM was adopted in the optimization method based ELM. By implementing the equality constraints in its optimization equations, ELM can provide a unified solution to different practical applications (e.g. regression, binary and multiclass classifications). ELM could also provide different solutions based on the application size thus to reduce the training complexity. The kernel

trick can also be used in ELM's solution. As supported in theory and by simulation results, ELM tends to have better generalization performance than SVM and its variants when the same kernel functions are used.

The equality constrained optimization method based ELM has shown promising results in the benchmark datasets. It is also important to test its performance in real-world applications. In chapter 4, the kernel based ELM was implemented in credit risk evaluation for two credit datasets. Simulation results showed that the kernel based ELM was more suitable for credit risk evaluation than the popular Support Vector Machines with consideration of overall, good and bad accuracy.

Compared with other machine learning techniques, ELM has greater potential of solving larger scale dataset problems due to its simple network structure. However when solving very large data problems, ELM requires a large number of hidden nodes to map the data to higher dimensional space where the data can be separated well. The large number of hidden nodes result in a large hidden layer matrix, which usually requires very large memory during computation. In chapter 5, a stacked ELMs (S-ELMs) method was proposed to solve the memory issue. Instead of using one single ELM with large hidden layer matrix, we broke it into multiple small ELMs and connected them serially. The stacked ELMs not only reduces the computational memory requirement, but also saves the training time. The generalization performance of S-ELMs can be further improved by implementing the unsupervised pretraining approach (usually an autoencoder) in each layer. The work presented in chapter 6 shows that when adding ELM based autoencoder to each layers of S-ELMs network, the testing accuracy could be significantly improved.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Research Background

Machine Learning, as a popular branch of computational intelligence, has been extensively studied by many researchers over the past few decades. It provides learning frameworks that can interpret the data without knowing the explicit relationship between the data instances. The two core tasks of many machine learning techniques are: 1) meaningful representation of data (feature extraction and selection); 2) strong generalization ability on unseen data (regression and classification). Machine learning techniques have been widely used in many real world applications and achieved great success. The current common challenges for most machine learning tasks are to constantly improve the learning ability on not well structured data and work efficiently on big data problems.

Artificial neural networks (ANNs), also named as "neural networks", are powerful mathematical models that are inspired by how the brain performs on a particular task or function of interest [1]. A neural network is usually defined as a massively parallel distributed processor that resembles the brain for analyzing and processing experiential knowledge. In a neural network, there are massive interconnection of simple computing

cells referred to as "neurons" that are used to carry out powerful computation through a process of learning. The input information is processed by the neurons and then all neurons are interconnected to build up a complex learning network that can learn and store the historical information. The interconnections between neurons could be very different thus resulting in many different architectures of neural networks such as: Single-hidden Layer Feedfarword Neural Networks (SLFNs), Multilayer Perceptron (MLP), Recurrent Neural Networks (RNNs) and etc.

Among the many different architectures of Neural Networks, SLFN has the simplest architecture yet be able to achieve fairly good generalization performance for most applications. The capabilities of SLFNs to approximate complicated nonlinear functions directly from the input data instances have been widely investigated [1, 2, 3, 4]. The SLFNs have been proven to be universal approximators under the condition that the hidden layer activation function is non-constant, bounded, piecewise continuous function [5]. Traditionally, a SLFN usually requires iterative learning to tune the hidden layer parameters, which is tedious and time consuming. However, such iterative learning process does not guarantee global optima and fast convergence rate.

Extreme Learning Machine (ELM), as an emergent technology was originally proposed by Huang et al. [6, 7, 8, 9, 10]. ELM is a special type of single-hidden layer feedforward networks (SLFNs), where the hidden nodes parameters are randomly generated, and the output weights are analytically determined using standard least square solutions. Essentially, ELM applies random computational nodes in the hidden layer without any prior knowledge of the training data instances. Different from traditional learning algorithms for a neural type of SLFNs, ELM tends to reach not only the smallest training error but also the smallest norm of output weights. According to Bartletts theory [11], for a feedforward neural network to have a smaller training error, the smaller the norm of weights is, the better generalization performance the networks tends to achieve. ELM has attracted more and more researchers' attentions. Many research work including the algorithm studies and real world applications have been developed [12, 13, 14, 15, 16, 17]. Due to the simple structure and fast learning speed, ELM

could have greater potential than other machine learning algorithms to solve larger scale problems.

Other than the neural networks discussed above, another well known popular machine learning technique is Support Vector Machine (SVM) [18]. In the past two decades, due to their surprising classification capabilities, SVM and its variants [19, 20, 21] have been widely used in solving many classification problems. In SVM, the training data are mapped into a higher dimensional space through some nonlinear mapping function. In the SVM mapping space, the standard optimization method is used to find a solution that can maximize the separation margin of the two different classes of data instances while minimizing the total training errors. Based on the original SVM, there are many SVM variants proposed to reduce the computational complexity, such as Least Square SVM (LS-SVM) [19] and Proximal SVM (PSVM) [20]. Both LS-SVM and PSVM employ equality constraints instead of inequality constraints in their optimization equations, which results in a direct least square solution that is much easier than solving the quadratic programming problem in the original SVM. With the introduction of the epsilon-insensitive loss function, the support vector machine has been extended to solve regression problems as well [22].

Deep Learning [23, 24, 25, 26, 27] is a new research area in Machine Learning that has achieved notable success in high-level data representation and abstraction. It uses multiple levels of non-linear operations instead of one shallow "fat" structure to solve the targeted machine learning problems. Before 2006, the training of deep supervised neural networks usually failed. The training and testing results tend to be worse in multiple hidden layers neural networks (deep structure) than single hidden layer neural networks (shallow structure). Since 2006, Hinton et al. [26, 27] and Bengio et al. [25] showed that the Restricted Boltzmann Machine (RBM) [28] and autoencoders [29] can be used as the unsupervised data pre-training step in training the multiple-layer neural networks, deep learning becomes popular thereafter.

1.2 Motivation and Objectives

Extreme learning machine (ELM) was proposed originally for the single-hidden layer feedforawd neural networks [6, 7, 8], then extended to the generalized SLFNs which may not be neuron alike [9, 10]. The main feature of ELM lies in that we generate random computational hidden nodes instead of iteratively tuning the hidden nodes parameters in the traditional way of training SLFNs. The output weight is analytically determined by solving a linear system. Hence ELM has extreme fast learning speed and is very convenient to be implemented in real applications. Compared with another popular used classification method: Support Vector Machine (SVM) [18], ELM can achieve similar classification performance with much faster training speed for most of the datasets. The parameter tuning process for ELM is much easier than SVM as well. However, due to the random feature mapping property of ELM that can cause high fluctuation on small training samples , ELM may not perform as well as SVM for small and sparse datasets. Thus it is interesting to find out the similarities and relationship between ELM and SVM, hence to improve ELM's classification ability.

As shown by Huang et al. [30], from standard optimization point of view, the minimization of output weights in ELM network is equivalent to the maximization of separation margin in SVMs. Based on Bartlett's theory [11], we can add the term that represents minimization of output weights to original ELM's error minimization equation. In this case, ELM and SVMs' optimization equation will become the same expect that ELM has fewer optimization constraints. According to the ELM theories [6, 7, 8, 9, 10] all the training data can be linearly separated by a hyperplane passing through the origin in the ELM feature space, hence the optimization method based ELM tends to achieve better generalization performance than SVM.

The computational complexity of training SVM algorithms is usually very intensive due to the need of solving the quadratic programming problem. Thus it is difficult to deal with big datasets using single traditional SVM. However, the two popular SVM variants: least square support vector machine (LS-SVM) [19] and proximal support vector ma-

chine (PSVM) [20], provide fast implementations of the traditional SVM. Both LS-SVM and PSVM use equality optimization constraints instead of inequality optimization constraints in the optimization equations, which result in much easier solutions by solving a set of linear equations other than the original quadratic programming problem.

Since LS-SVM and PSVM have been proven to achieve good classification performance, it is essential to examine whether the least square approach can be implemented in optimization method based ELM. The optimization method based ELM is proven to perform better than SVM in [30]. If least square approach (equality constrained) is used in conjunction with optimization method based ELM, a faster classifier with better generalization ability can be expected. SVM, LS-SVM, PSVM are originally proposed for binary classification problems. They require some additional coding schemes such as one-against-one (OAO) and one-against-all (OAA) methods in order to solving multi-class classification problems. ELM itself is an unified model that can solve regression, binary classification and multi-class classification problems directly. It is also interesting to see whether the equality constrained optimization method based ELM can provide unified solution to regression, binary classification and multi-class classification problems directly.

In [31], Huang et al. proposed the equality constrained optimization method based Extreme Learning Machine, and provided two solutions for different size of training data. The user can choose to use *training data size based* (complexity of computation is based on the size of training data) for the small to medium size training data, or *hidden nodes size based* (complexity of computation is based on the number of hidden nodes). The training data is usually mapped from original input space to a higher dimensional space, within which the tackled problem can be solved. The number of hidden nodes determines the dimensionality of the mapped space. In general, to solve a classification problem, higher number of training samples requires more hidden nodes so that they can be separated in the ELM mapping space. This brings problems to ELM when facing extremely large training datasets. The large number of required hidden nodes makes ELM network very large and the computational task becomes very costly. To solve

large data problems using ELM without incurring a memory problem, one should keep the network size small yet be able to achieve good generalization accuracy. From the literature [32, 24, 25, 23], it seems multiple-hidden-layer feedforward networks have more powerful learning capability than single-hidden-layer feedforward networks. Thus we can also work on multiple layers network structure to improve ELM's capability on solving very large data problems.

The Stacked ELMs (S-ELMs) is then proposed to break a large ELM network into multiple serially connected small ELMs. Compared with single large ELM network, the S-ELMs has low computational cost and low memory requirement. The idea is inspired by the deep architectures. e.g. stacked autoencoder networks [25], that each layer is serially connected and propagates the output to next layer. However, S-ELMs does not incorporate the key idea of unsupervised pre-training of data in deep learning, it is purely a solution that aims to help ELM solve large scale problems. To further improve the testing rate, especially for data without properly selected features, the autoencoder can be implemented in each layer of S-ELMs, and better classification rate can be expected.

1.3 Major Contributions

The major progress and contributions throughout the whole PhD research period are summarized as follows.

In the first work, associated with Dr. Ding Xiaojian, we investigated the similarities between ELM and SVMs and developed the standard optimization method based ELM for classification. Following this work, by studying the advantage of least square Support Vector Machine (LS-SVM) and Proximal Support Vector Machine (PSVM), we employed the equality constraints instead of the inequality constraints in the optimization equations of ELM. The developed equality constrained optimization method based ELM is a unified model that can solve regression, binary and multiclass classification problems directly. It provides different solutions based on the training data size, thus it is

more capable of solving larger scale problems with less computational cost. If the ELM feature mapping is not explicitly known, the kernel functions can also be used in the equality constrained optimization method based ELM. In theory, LS-SVM tends achieve suboptimal solutions compared with the equality constrained optimization method based ELM if the same kernels are used. As verified by the simulation results, the resultant ELM can perform better than LS-SVM with faster training speed.

In the second work, the equality constrained optimization method based ELM was implemented in a real-world application: credit risk evaluation. In this application, ELM (kernel based) was used as a classification tool to perform the credit risk evaluation on two credit risk evaluation datasets: the Australian credit and German credit datasets. The simulations were done on two credit risk evaluation datasets with three different kernel functions. Simulation results showed that, when the same kernel function was used, the kernel based ELM was more suitable for credit risk evaluation than the popular used Support Vector Machines (SVMs) with consideration of overall, good and bad accuracy.

In the third work, the concentration was put on solving large data problems using ELM. When facing super large datasets, in order to have a good generalization performance, ELM requires a large number of hidden nodes to map the data to high dimensional space. However, due to the physical memory limitation of most computing machines, the number of hidden nodes to be used is limited. Thus ELM usually cannot fully achieve its generalization capability. The Stacked ELMs learning network (S-ELMs) was proposed to divide a single large ELM network into small ELMs on many different layers. The eigenvalues of the output weight matrix of ELM on each layer were used to determine the most significant nodes. Those nodes are combined together with the random nodes in the next layer and function as the hidden nodes of the next layer ELM. With such multiple layers structure, ELM can potentially solve datasets of any size and fully achieve its generalization capability. The simulation results on MNIST dataset showed that the S-ELMs even with random hidden nodes could achieve similar testing accuracy to SVM with a faster training speed and low memory requirement.

In the fourth work, I further investigated S-ELMs and improved S-ELMs' testing rate by adding ELM based autoencoders in each layer of S-ELMs. The ELM based autoencoders help to capture useful data features that can be used for further ELM mapping. The ELM based autoencoders serve the purpose as unsupervised pretraining of data. Simulation results showed that when implementing ELM based autoencoders into each layer of S-ELMs, it improved the accuracy significantly and the accuracy was better than many state of the art techniques such as Support Vector Machines (SVMs) and Deep Belief Networks (DBNs). The main advantages of the proposed method are still its ability in solving large data problems with low memory requirements and very fast training speed.

1.4 Organization

The remaining part of the thesis is organized as follows:

Chapter 2 gives the literature reviews on current popular machine learning techniques including Extreme Learning Machines (ELMs), Support Vector Machine (SVM) and its variants including Least Square SVM, Proximal SVM and Support Vector Regression.

Chapter 3 introduces the equality constrained optimization method based ELM, discusses the advantage of the proposed algorithm over SVM and LS-SVM.

Chapter 4 presents a real-world application: credit risk evaluation based on kernelized ELM.

Chapter 5 proposes a Stacked ELMs that can solve very large data problems with low computational cost.

Chapter 6 implements the ELM based autoencoder into the Stacked ELMs for better classification accuracy on large data problems.

Chapter 7 concludes the thesis and addressed the future work.

Chapter 2

Literature Review

2.1 Extreme Learning Machines

Extreme Learning Machines (ELMs) were originally proposed by Huang et al. from 2004 onwards for training the Single-hidden Layer Feedforward Neural Networks (SLFNs) [6, 7, 8]. Different from the common understanding of training SLFNs, ELM employs random computational nodes in the hidden layer and computes its output weights analytically by solving a general linear system equation. Later, ELM theory was extended to the "generalized" SLFNs, where the hidden nodes need not be neuron alike [9, 10]. Two essential features of ELM are that: 1) the hidden layer parameters need not be tuned; 2) the hidden nodes are independent of training data. Different from traditional neural network learning algorithms, ELM tends to reach both smallest training error and smallest norm of output weights. Following the basic ELM, there are extensive studies on ELM by many researchers and many variants of ELM have been developed [30, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44]. In this section, I will mainly introduce the basic ELM and the optimization method based ELM [30] that is closely related to my research work.

2.1.1 Brief of basic Extreme Learning Machine

Given N arbitrary training examples $\{(\mathbf{x}_j, \mathbf{t}_j)\}_{j=1}^{N} \subset \mathbf{R}^d \times \mathbf{R}^m$, the output of the generalized SLFNs with L hidden nodes can be obtained as:

$$f(\mathbf{x}_j) = \sum_{i=1}^{L} \beta_i g_i(\mathbf{x}_j) = \sum_{i=1}^{L} \beta_i G(\mathbf{a}_i, b_i, \mathbf{x}_j) = \mathbf{o}_j, \ j = 1, ..., N \tag{2.1}$$

If the SLFNs can approximate all the N samples with zero error, that is $\sum_{j=1}^{L} \| \mathbf{o}_j - \mathbf{t}_j \| = 0$, there exist pairs of $(\mathbf{a}_i, b_i)$ and β_i such that:

$$\sum_{i=1}^{L} \beta_i G(\mathbf{a}_i, b_i, \mathbf{x}_j) = \mathbf{t}_j, \ j = 1, ..., N \tag{2.2}$$

The above N equations can also be equivalently expressed in the compact matrix form

$$\mathbf{H}\beta = \mathbf{T}, \tag{2.3}$$

where

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

$\mathbf{H}$ is called the hidden-layer output matrix of the SLFN, where the ith row represents the output vector of the hidden layer with respect to the ith input $\mathbf{x}_i$, and jth column denotes the jth hidden node's output vector with respect to the inputs $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N$. β represents the output weight matrix. $\mathbf{T}$ is the vector/matrix that consists of output labels for the N data samples.

Unlike other traditional training algorithms for SLFNs (such as Back Propagation algorithm [45]), ELM randomly generates the hidden node parameters $(\mathbf{a}_i, b_i)$. Thus the hidden layer matrix $\mathbf{H}$ is randomly generated.

To train a SLFN as mentioned in (2.1), it is equivalent to finding the least-sqaure solution $\hat{\beta}$ of the linear system (2.3), that is:

$$\|\mathbf{H}\hat{\beta} - \mathbf{T}\| = \min_{\beta} \|\mathbf{H}\beta - \mathbf{T}\| \tag{2.4}$$

In the case that the number of hidden nodes L is equal to the number of distinct training samples N, it is possible to find a $\hat{\beta}$ such that the training error reaches zero. The hidden layer output $\mathbf{H}$ is an invertable square matrix. Hence the solution of the linear system can be given as:

$$\hat{\beta} = \mathbf{H}^{-1}\mathbf{T} \tag{2.5}$$

Proposition 2.1 *[7] Given a SLFN with L hidden nodes and activation function $g(x)$ that is infinitely differentiable in any interval of R, for N arbitrary distinctinputs $\{(\mathbf{x}_i, \mathbf{t}_i) \in \mathbf{R}^d \times \mathbf{R}^m, i = 1, \ldots, N\}$ and $\{(\mathbf{a}_i, b_i)\}_{i=1}^{L}$ that is randomly generated from any intervals of $\mathbf{R}^d \times R$, according to any continuous probability distribution, if $L = N$, with probability one, $\|\mathbf{H}_{N\times L}\beta_{L\times m} - \mathbf{T}_{N\times m}\| = 0$.*

In the case that the number of hidden nodes L is less than the number of distinct training samples N, to achieve the smallest training error $\|\mathbf{H}\beta - \mathbf{T}\|$, the solution of the linear system (2.3) can be obtained as:

$$\hat{\beta} = \mathbf{H}^{\dagger}\mathbf{T} \tag{2.6}$$

where $\mathbf{H}^{\dagger}$ is called the Moore-Penrose generalized inverse [46, 47]. In theory, equation (2.6) is a unique solution that tends to achieve the smallest norm of $\hat{\beta}$ while keeping the training error minimal. According to Bartlett's theory [11], for FNNs achieving smallest training error, the smaller the norm of the weight is, the better generalization ability the networks tend to have. Therefore, ELM not only achieves smallest training error but also tends to have good generalization performance.

Proposition 2.2 *[7] Given any small possible value $\varepsilon > 0$ and the activation function $g(x)$ that is infinitely differentiable in any interval of R, for N arbitrary distinct inputs $\{(\mathbf{x}_i, \mathbf{t}_i) \in \mathbf{R}^d \times \mathbf{R}^m, i = 1, \ldots, N\}$ and $\{(\mathbf{a}_i, b_i)\}_{i=1}^{L}$ and $\{(\mathbf{a}_i, b_i)\}_{i=1}^{L}$ that is randomly generated from any intervals of $R^d \times R$, according to any continuous probability distribution,there exists $L \leq N$ such that $\|\mathbf{H}_{N\times L}\beta_{L\times m} - \mathbf{T}_{N\times m}\| < \varepsilon$ with probability one.*

The three-step learning algorithm for the basic ELM is summarized as follows:

Remark: According to ELM learning theory [6, 7, 8], a widespread type of infinitely differentiable activation actions $G(\mathbf{a}_i, b_i, \mathbf{x})$ can be used in ELM so that ELM can approximate any continuous target functions.

Algorithm 1 The three-step learning algorithm of basic ELM

Given the training data $\{(\mathbf{x}_i, \mathbf{t}_i) \in \mathbf{R}^d \times \mathbf{R}^m, i = 1, \ldots, N\}$, the activation function $G(\mathbf{a}_i, b_i, \mathbf{x})$ and the number of hidden nodes L, the training process of ELM is:

step 1) Randomly generate the hidden node parameters $(\mathbf{a}_i, b_i), i = 1, \cdots, L$;

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

step 3) Calculate the output weight $\beta = \mathbf{H}^{\dagger}\mathbf{T}$, where $\mathbf{H}^{\dagger}$ is the Moore-Penrose generalized inverse of hidden layer output matrix $\mathbf{H}$.

2.1.2 Optimization method based Extreme Learning Machine for classification

From the discussion of section 2.1.1, it can be seen that ELM not only tends to achieve the smallest training error $\|\mathbf{H}\beta - \mathbf{T}\|$, but also obtains the smallest norm of output weight β. That is :

$$\text{Minimize: } \|\mathbf{H}\beta - \mathbf{T}\|^2 \text{ and } \|\beta\| \tag{2.7}$$

Huang et al. [30] further studied ELM for classification from the standard optimization method point of view. In ELM, for the binary classification case, the equation for hyperplane that separates the ELM mapped feature space is : $\beta \cdot h(\mathbf{x}_i) = 0$, and the classifers for separable training sets are:

$$\begin{aligned} &\beta \cdot h(\mathbf{x}_i) \geq 1 - \xi_i \text{ if } t_i = 1 \\ &\beta \cdot h(\mathbf{x}_i) \leq -1 + \xi_i \text{ if } t_i = -1 \end{aligned} \tag{2.8}$$

That is:

$$t_i(\beta \cdot h(\mathbf{x}_i)) \geq 1 - \xi_i, \; i = 1, \cdots, N \tag{2.9}$$

where $h(\mathbf{x})$ is the ELM mapping for input $\mathbf{x}$ and ξ is the training error. The above equation (2.9) and (2.7) can be combined to formulate the following optimization problem for ELM:

$$\begin{aligned} &\text{Minimize: } L_P = \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N}\xi_i \\ &\text{Subject to: } t_i\beta \cdot h(\mathbf{x}_i) \geq 1-\xi_i, \quad i = 1,\cdots,N \\ &\qquad\qquad \xi_i \geq 0, \quad i = 1,\cdots,N, \end{aligned} \tag{2.10}$$

In equation 2.10, the decision variables are the output weights in ELM network β and the training error ξ. The optimization equation of ELM is quite similar to Support Vector Machine's (SVM) [18] optimization equation. However, there are two main differences: 1) different from the conventional SVM, the feature mapping $h(\mathbf{x})$ in ELM is completely random; 2) in the constraint functions, ELM doesnot have the bias term b compared with SVM. In ELM feature mapping space, the data points will be separated by a hyperplane that tends to pass through the origin.

Based on the Karush-Kuhn-Tucker theorem [48], the above optimization equation (2.10) for ELM can also be expressed in its dual optimization equation:

$$\begin{aligned} &\text{minimize: } L_D = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} t_i t_j K_{\text{ELM}}(\mathbf{x}_i,\mathbf{x}_j)\alpha_i\alpha_j - \sum_{i=1}^{N}\alpha_i \\ &\text{subject to: } 0 \leq \alpha_i \leq C, i = 1,\cdots,N. \end{aligned} \tag{2.11}$$

where $K_{\text{ELM}}(\mathbf{x}_i,\mathbf{x}_j)$ is called ELM kernel and can be defined as:

$$K_{\mathrm{ELM}}(\mathbf{x}_i, \mathbf{x}_j) = h(\mathbf{x}_i) \cdot h(\mathbf{x}_j)$$
$$= [G(\mathbf{a}_1, b_1, \mathbf{x}_i), \cdots, G(\mathbf{a}_L, b_L, \mathbf{x}_i)]^T \cdot [G(\mathbf{a}_1, b_1, \mathbf{x}_j), \cdots, G(\mathbf{a}_L, b_L, \mathbf{x}_j)]^T, \tag{2.12}$$

As discussed above, the aim of ELM is to minimize both the training error $\|\mathbf{H}\beta - \mathbf{T}\|$ and the norm of output weights $\|\beta\|$. However, in ELM feature mapping space, the distance between the two separation hyperplanes is $2/\|\beta\|$. Thus to minimize the norm of output weight β is the same as to maximize the separation margin between two hyperplanes, which is consistent with SVM's maximization of separation margin property. If we compare the optimization equations of ELM (2.11) and SVM (2.19), it is found that ELM has milder optimization constraints than SVM, thus SVM tends to obtain a solution that is sub-optimal to ELM's solution. As supported by the experimental results, ELM for classification is much simpler in tuning the user specified parameters and can achieve better generalization performance compared with conventional SVM and SVM with ELM kernel (the case that the existing kernels are replaced with ELM kernel in SVM).

2.2 Support Vector Machines

2.2.1 Brief of basic Support Vector Machine

Support Vector Machine (SVM) is a special type of feed-forward networks, pioneered by Cortes and Vapnik[18]. Like multilayer perceptrons and radial-basis function networks, support vector machine can be used for classification and nonlinear regression. The main idea of SVM is to construct a hyperplane as the decision surface in such a way that the margin of separation between positive and negative examples is maximized. The support vector machine achieves this desirable property by minimizing the cost function

under certain constraints and reduces the sum of training error.

Given a set of training data consisting of N data points $\{(\mathbf{x}_i, t_i)\}_{i=1}^{N}$,where $\mathbf{x}_i$ is the input pattern for the ith example and $t_i \varepsilon \{-1, +1\}$ is the corresponding desired response. The equation of the separation hyperplane is $\mathbf{w} \cdot \mathbf{x} + b = 0$. If the training data cannot be linearly separated by the hyperplane, we can map the training data to a higher dimensional feature space as: $\mathbf{x}_i \rightarrow \phi(\mathbf{x}_i)$. In this case, the classifiers for separable training sets are:

$$\mathbf{w} \cdot \phi(\mathbf{x}_i) + b \geq 1, \text{ if } t_i = +1 \tag{2.13}$$

$$\mathbf{w} \cdot \phi(\mathbf{x}_i) + b \leq -1, \text{ if } t_i = -1 \tag{2.14}$$

The above equations (1) and (2) can be combined as:

$$t_i(\mathbf{w} \cdot \phi(\mathbf{x}_i) + b) \geq 1, i = 1, 2, \cdots, N \tag{2.15}$$

In the case of nonseparable patterns, a new set of non-negative scalar variables $\{\xi_i\}_{i=1}^{N}$ are introduced into the definition of the separating hyperplane, hence equation (3) becomes:

$$\begin{aligned} &t_i(\mathbf{w} \cdot \phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, i = 1, \cdots, N \\ &\xi_i \geq 0, i = 1, 2, \cdots, N \end{aligned} \tag{2.16}$$

where ξ_i are called slack variables, and they measure the derivation of a data point from the ideal condition of pattern separability. To maximize the separation margin and minimize training error, we can formulate the optimization equation as:

$$\begin{aligned} &\text{Minimize: } L_p = \frac{1}{2}\mathbf{w} \cdot \mathbf{w} + C\sum_{i=1}^{N} \xi_i \\ &\text{subject to: } t_i(\mathbf{w} \cdot \phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, i = 1, \cdots, N \\ &\qquad\qquad \xi_i \geq 0, i = 1, 2, \cdots, N \end{aligned} \tag{2.17}$$

where C is the cost parameter. Therefore, we can construct the Lagrangian equation as:

$$\mathscr{L}_p = L_p - \sum_{i=1}^{N} \alpha_i (t_i(\mathbf{w} \cdot \phi(\mathbf{x}_i) + b) - 1 + \xi_i) - \sum_{i=1}^{n} \nu_i \xi_i \tag{2.18}$$

where $\alpha_i \geq 0$ and $\nu_i \geq 0$ are called Lagrange multipliers. Based on the Karush-Kuhn-Tucker(KKT) theorem [48], to solve the above Lagrangian equation (6) is equivalent to solving the dual optimization problem as:

$$\begin{aligned} &\text{minimize: } L_D = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} t_i t_j \alpha_i \alpha_j \phi(\mathbf{x}_i) \cdot \phi\left(\mathbf{x}_j\right)) - \sum_{i=1}^{N} \alpha_i \\ &\text{subject to: } \sum_{i=1}^{N} t_i \alpha_i = 0 \\ &\qquad\qquad 0 \leq \alpha_i \leq C, i = 1, \cdots, N \end{aligned} \tag{2.19}$$

From KKT optimality condition, we can also solve $\mathbf{w}$ to be:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i t_i \phi(\mathbf{x}_i) \tag{2.20}$$

In equation (7), we can write $\phi(\mathbf{x}_i) \cdot \phi\left(\mathbf{x}_j\right) = K\left(\mathbf{x}_i, \mathbf{x}_j\right)$, which is called a kernel function, motivated by Mercer's Theorem [18]. The desicion function of SVM with the kernel function then becomes:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{N} \alpha_i t_i K(\mathbf{x}, \mathbf{x}_i) + b\right) \tag{2.21}$$

2.2.2 Least Square Support Vector Machine

Suykens and Vandewalle [19] proposes a least square approach to solve SVM classification problem. They replace the inequality constraints to equality constraints for the optimization equation. Hence by solving a set of linear equations instead of quadratic programming, the least square approach is formulated. Least square SVM is proven to have excellent generalization performance and low computational cost.

The optimization equation is formulated as:

$$\begin{aligned} &\text{Minimize: } \mathrm{L}_{LS-SVM} = \frac{1}{2}\mathbf{w}\cdot\mathbf{w} + C\frac{1}{2}\sum_{i=1}^{N}\xi_i^2 \\ &\text{Subject to: } t_i(\mathbf{w}\cdot\phi(\mathbf{x}_i)+b)) = 1-\xi_i,\ i\ = 1,\cdots,N \end{aligned} \tag{2.22}$$

where ξ_i is the slack variable. The constraints of equation (18) are equality constraints derived from the inequality constraints in equation(4), and thus one can define the lagrangian equation with only one lagrange multiplier as:

$$\mathscr{L}_{LS-SVM} = L_{LS-SVM} - \sum_{i=1}^{N}\alpha_i(t_i(\mathbf{w}\cdot\phi(\mathbf{x}_i)+b) - 1 + \xi_i) \tag{2.23}$$

where α_i are the lagrange multiples, in this case, α_i can be either positive or negative due to the equality constraints used. From the Karush-Kuhn-Tucker(KKT) theorem, the conditions for optimality are given by:

$$\frac{\partial\mathscr{L}_{LS-SVM}}{\partial\mathbf{w}} = 0 \rightarrow \mathbf{w} = \sum_{i=1}^{N}\alpha_i t_i\phi(\mathbf{x}_i) \tag{2.24a}$$

$$\frac{\partial\mathscr{L}_{LS-SVM}}{\partial b} = 0 \rightarrow \sum_{i=1}^{N}\alpha_i t_i = 0 \tag{2.24b}$$

$$\frac{\partial\mathscr{L}_{LS-SVM}}{\partial\xi_i} = 0 \rightarrow \alpha_i = C\xi_i,\ i = 1,\ldots,N \tag{2.24c}$$

$$\frac{\partial\mathscr{L}_{LS-SVM}}{\partial\alpha_i} = 0 \rightarrow t_i(\mathbf{w}\cdot\phi(\mathbf{x}_i)+b) - 1 + \xi_i = 0,\ i = 1,\ldots,N \tag{2.24d}$$

Equivalently, by substituting equations (12a), (12b) and (12c) into equation in (12d), the above equations can be written as:

$$\begin{bmatrix} \mathbf{0} & \mathbf{T}^T \\ \mathbf{T} & \frac{\mathbf{I}}{C} + \Omega_{LS-SVM} \end{bmatrix}\begin{bmatrix} b \\ \alpha \end{bmatrix} = \begin{bmatrix} \mathbf{0} & \mathbf{T}^T \\ \mathbf{T} & \frac{\mathbf{I}}{C} + \mathbf{Z}\mathbf{Z}^T \end{bmatrix}\begin{bmatrix} b \\ \alpha \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \vec{\mathbf{1}} \end{bmatrix} \tag{2.25}$$

where

$$\mathbf{Z} = \begin{bmatrix} t_1\phi(\mathbf{x}_1) \\ \vdots \\ t_N\phi(\mathbf{x}_N) \end{bmatrix}$$
$$\Omega_{LS-SVM} = \mathbf{Z}\mathbf{Z}^T \tag{2.26}$$

The feature mapping $\phi(\mathbf{x})$ is a row vector, $\mathbf{T} = [t_1, t_2, \cdots, t_N]^T$, $\alpha = [\alpha_1, \alpha_2, \cdots, \alpha_N]^T$ and $\vec{\mathbf{1}} = [1, 1, \cdots, 1]^T$. In LS-SVM, as $\phi(\mathbf{x})$ is usually unknown, Mercer's condition [1] can be applied to matrix Ω_{LS-SVM}:

$$\Omega_{LS-SVM\,i,j} = t_i t_j \phi(\mathbf{x}_i) \cdot \phi(\mathbf{x}_j) = t_i t_j K(\mathbf{x}_i, \mathbf{x}_j) \tag{2.27}$$

The decision function of LS-SVM classifier is: $f(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{N} \alpha_i t_i K(\mathbf{x}, \mathbf{x}_i) + b\right)$.

2.2.3 Proximal Support Vector Machine

Fung and Mangasarian [20] proposes the proximal support vector machine classifier, which classifies data points depending on proximity to either one of the two separation planes that are aimed to be pushed away as far apart as possible. The key idea is that the separation hyperplanes now are not bounded planes anymore but "proximal" planes, and such effect is reflected in mathematical expressions that the inequality constraints are changed to equality constraints. In the objective equation of linear Proximal SVM, $(\mathbf{w} \cdot \mathbf{w} + b^2)$ is used instead of $\mathbf{w} \cdot \mathbf{w}$, making the optimization problem strongly convex, and has little or no effect on the original optimization problem.

The mathematical model built for linear PSVM is:

$$\begin{aligned} &\text{Minimize: } L_{LPSVM} = \frac{1}{2}(\mathbf{w} \cdot \mathbf{w} + b^2) + C\frac{1}{2}\sum_{i=1}^{N} {\xi_i}^2 \\ &\text{Subject to: } t_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1 - \xi_i,\ i = 1, \cdots, N \end{aligned} \tag{2.28}$$

For least square SVM classifier, the calculation is in the $\mathbf{w}-$space of R^d, where d is the dimensionality of input space, while the linear Proximal SVM is in the $(\mathbf{w},b)-$space of R^{d+1}. The bias b can be viewed as an additional dimension of $\mathbf{w}$, so the Lagrangian function is constructed as:

$$\mathscr{L}_{LPSVM} = \frac{1}{2}\left(\begin{bmatrix}\mathbf{w}\\ b\end{bmatrix}\cdot\begin{bmatrix}\mathbf{w}\\ b\end{bmatrix}\right) + C\frac{1}{2}\sum_{i=1}^{N}{\xi_i}^2 - \sum_{i=1}^{N}\alpha_i(t_i(\mathbf{w}\cdot\mathbf{x}_i+b)-1+\xi_i) \quad (2.29)$$

The KKT optimality condition of linear PSVM could be obtained as:

$$\frac{\partial\mathscr{L}_{LPSVM}}{\partial\mathbf{w}} = 0 \rightarrow \mathbf{w} = \sum_{i=1}^{N}\alpha_i t_i \mathbf{x}_i \quad (2.30a)$$

$$\frac{\partial\mathscr{L}_{LPSVM}}{\partial b} = 0 \rightarrow b = \sum_{i=1}^{N}\alpha_i t_i \quad (2.30b)$$

$$\frac{\partial\mathscr{L}_{LPSVM}}{\partial\xi_i} = 0 \rightarrow \alpha_i = C\xi_i,\ i=1,\ldots,N \quad (2.30c)$$

$$\frac{\partial\mathscr{L}_{LPSVM}}{\partial\alpha_i} = 0 \rightarrow t_i[\mathbf{w}\cdot\mathbf{x}_i+b]-1+\xi_i = 0,\ i=1,\ldots,N \quad (2.30d)$$

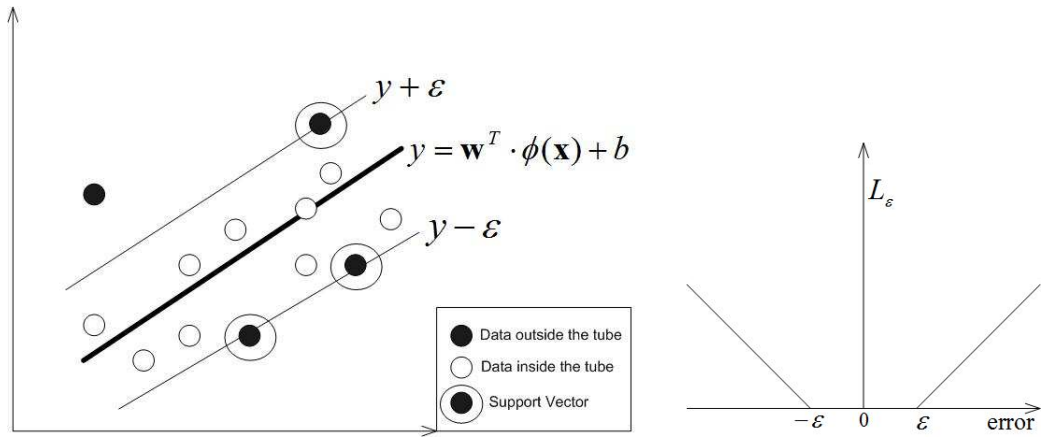
Equivalently,

$$\left(\frac{\mathbf{I}}{C}+\Omega_{PSVM}+\mathbf{T}\mathbf{T}^T\right)\alpha = \left(\frac{\mathbf{I}}{C}+\mathbf{Z}\mathbf{Z}^T+\mathbf{T}\mathbf{T}^T\right)\alpha = \vec{\mathbf{1}} \quad (2.31)$$

where $\mathbf{Z} = [t_1\mathbf{x}_1,\cdots,t_N\mathbf{x}_N]^T$ and $\Omega_{PSVM} = \mathbf{Z}\mathbf{Z}^T$.

Conventionally, we can extend the linear input space $\mathbf{x}$ into its SVM feature space $\phi(\mathbf{x})$: $\mathbf{x}\rightarrow\phi(\mathbf{x})$, and obtain the nonlinear version of proximal SVM. In the case of such nonlinear PSVM, Ω_{NLPSVM} is the same as Ω_{LSSVM} in least square SVM, that is:

$$\Omega_{NLPSVM\,i,j} = t_i t_j \phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j) = t_i t_j K(\mathbf{x}_i,\mathbf{x}_j) \quad (2.32)$$



(a) The linear ε-SVR (b) ε-insensitive loss function

Figure 2.1: An example of linear ε-support vector regression and its ε-insensitive loss function

2.2.4 Support Vector Regression

The Support Vector Machine could also be used for function approximation when combining with the ε-insensitive loss function, which is called ε-support vector regression (ε-SVR) [49]. The ε-insensitive loss function can be defined as follows:

$$L_\varepsilon(t,y)=\begin{cases}|t-y|-\varepsilon, & \text{for } |t-y|\geq\varepsilon\\ 0 & \text{otherwise}\end{cases} \tag{2.33}$$

where t is the target output and y is the calculated output. ε is a user specified parameter that creates an ε-tube. This loss function indicates the penalization for errors larger than ε and neglects any absolute error less than or equal to ε.

Given a set of training data $(\mathbf{x}_i,t_i)_{i=1}^N, \mathbf{x}\in R^d, t\in R$, the output function of a SVR can be obtained as:

$$y = \sum_{i=1}^{m} w_i \phi_i(\mathbf{x}) + b = \mathbf{w} \cdot \phi(\mathbf{x}) + b \tag{2.34}$$

where $\phi(\mathbf{x}) = [\phi_1(\mathbf{x}), \cdots, \phi_m(\mathbf{x})]^T$ is the SVR feature mapping that maps the training data from original input space to the SVR feature space, $\mathbf{w} = [w_1, \cdots, w_m]^T$ and m is the dimensionality of SVR feature mapping space. According to the definition of loss function in equation (2.33), data points inside the ε-tube will not have any impact on the solution of ε-SVR's optimization problem.

To minimize the training error and obtain the smallest norm $\|\mathbf{w}\|$ that can minimize the model complexity thus to achieve good generalization performance [50], the optimization problem can be formulated as the following convex optimization problem:

$$\begin{aligned} &\text{Minimize: } \frac{1}{2}\|\mathbf{w}\|^2 \\ &\text{Subject to: } t_i - \mathbf{w} \cdot \phi(\mathbf{x}_i) - b \leq \varepsilon, \\ &\qquad\qquad \mathbf{w} \cdot \phi(\mathbf{x}_i) + b - t_i \leq \varepsilon, \quad i = 1, \cdots, N, \end{aligned} \tag{2.35}$$

The slack variables ξ_i, ξ_i^* can be added to equation (2.35), thus the optimization problem can be obtained as:

$$\begin{aligned} &\text{Minimize: } \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}(\xi_i + \xi_i^*) \\ &\text{Subject to: } t_i - \mathbf{w} \cdot \phi(\mathbf{x}_i) - b \leq \varepsilon + \xi_i, \\ &\qquad\qquad \mathbf{w} \cdot \phi(\mathbf{x}_i) + b - t_i \leq \varepsilon + \xi_i^*, \\ &\qquad\qquad \xi_i, \xi_i^* \geq 0 \quad i = 1, \cdots, N. \end{aligned} \tag{2.36}$$

The primal Lagrange function for this optimization problem is:

$$L_p(\mathbf{w},\xi,\xi^*,\alpha,\alpha^*,\gamma,\gamma^*) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{N}(\xi_i+\xi_i^*) - \sum_{i=1}^{N}\alpha_i[\mathbf{w}\cdot\phi(\mathbf{x}_i)+b-t_i+\varepsilon+\xi_i] - \sum_{i=1}^{N}\alpha_i^*[t_i-\mathbf{w}\cdot\phi(\mathbf{x}_i)-b-\varepsilon+\xi_i^*] - \sum_{i=1}^{N}(\gamma_i\xi_i+\gamma_i^*\xi_i^*), \tag{2.37}$$

where $\alpha_i, \alpha_i^*, \gamma_i, \gamma_i^* \geq 0$ are the Lagrange multipliers. At the saddle points where the partial derivatives of L_p with respect to $(\mathbf{w}, b, \xi_i, \xi_i^*)$ are equal to zero, we can obtain the following sets of equations:

$$\begin{aligned} &\sum_{i=1}^{N}(\alpha_i-\alpha_i^*) = 0 \\ &\mathbf{w} = \sum_{i=1}^{N}(\alpha_i-\alpha_i^*)\phi(\mathbf{x}_i) \\ &\gamma_i = C-\alpha_i \\ &\gamma_i^* = C-\alpha_i^* \end{aligned} \tag{2.38}$$

By substituting the sub-equations in equation (2.38) into equation (2.37), we can get the dual optimization problem for ε-SVR:

$$\begin{aligned}
\text{Maximize: } L_d(\alpha_i, \alpha_i^*) &= \sum_{i=1}^{N} d_i(\alpha_i - \alpha_i^*) - \varepsilon \sum_{i=1}^{N}(\alpha_i + \alpha_i^*) \\
&\quad - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)K_{\text{SVR}}(\mathbf{x}_i, \mathbf{x}_j) \\
\text{Subject to: } &\sum_{i=1}^{N}(\alpha_i - \alpha_i^*) = 0 \\
&0 \le \alpha_i \le C, \quad i = 1, \cdots, N \\
&0 \le \alpha_i^* \le C, \quad i = 1, \cdots, N
\end{aligned} \tag{2.39}$$

where $K_{\text{SVR}}(\mathbf{x}_i, \mathbf{x}_j) = \phi^T(\mathbf{x}_i)\phi(\mathbf{x}_j)$ is the kernel function.

Based on equation (2.34) and equation (2.38), the output of ε-SVR can be calculated as:

$$f(x) = \mathbf{w} \cdot \phi(\mathbf{x}) = \sum_{i=1}^{N_s}(\alpha_i - \alpha_i^*)K_{\text{SVR}}(\mathbf{x}, \mathbf{x}_i) + b, \tag{2.40}$$

where N_s is the number of support vectors which reveals the complexity of the network.

2.3 Summary

In the first section of this chapter, both ELM and optimization methods based ELM were introduced. ELM is a specific type of generalized SLFNs, where the hidden layer parameters need not be tuned. It provides simple solution for training the SLFNs without iterative tuning. According to ELM theorem, ELM is an universal approximator that can approximate any continues target functions. From the optimization point of view, ELM can also be considered as a standard optimization problem. ELM aims to minimize both the training error and output weight norm to achieve good generalization

performance. Compared to SVM, ELM's optimization equation has milder constraints, thus ELM tends to obtain the same or better classification rate than SVM if the same kernel functions are adopted. There are many other developmemts of ELM, however in this thesis I only focus the original ELM and optimization method based ELM that is closely related to my research work.

In the second section of this chapter, SVM and its variants including LS-SVM, PSVM and ε-SVR were reviewed. SVM was originally proposed to solve the binary classification problem. It tries to minimize the training error and maximize the separation margin of the two hyperplanes. Kernel functions are widely used in SVM so that the SVM feature mapping does not need to be explicitly defined. The LS-SVM and PSVM change the inequality constraints of SVM's optimization problem into equality constraints. Instead of solving the complicated quadratic programming problem, LS-SVM and PSVM need only solve a set of linear equations, thus they can provide faster solutions than the orginal SVM. SVM can also be used for function approximation when combining with the ε-insensitive loss function.

Chapter 3

Equality Constrained Optimization Method based Extreme Learning Machine

In this chapter, the equality constrained optimization method based Extreme Learning Machine is proposed. This chapter also analyses and discusses the similarity and difference between the proposed equality constrained optimization based ELM and least square SVM and proximal SVM, including their feature mappings, optimization constraints, computational complexity and scalability.

3.1 Equality Constrained Optimization Method based Extreme Learning Machine

According to Huang, et al [8, 9, 10], there are widespread type of feature mapping functions $\mathbf{h}(\mathbf{x})$ that can be used in ELM so that ELM can approximate any continuous target functions. That is, for any continuous target function $f(\mathbf{x})$ there exist a series of

β_i in ELM such that

$$\lim_{L\to+\infty} \|f_L(\mathbf{x}) - f(\mathbf{x})\| = \lim_{L\to+\infty} \left\| \sum_{i=1}^{L} \beta_i h_i(\mathbf{x}) - f(\mathbf{x}) \right\| = 0 \tag{3.1}$$

With this universal approximation capability, there would be no bias term b in the optimization constraints of ELM's optimization problem. Compared with the optimization problem of SVM, LS-SVM, and PSVM, the resultant ELM learning algorithm has milder optimization constraints. Thus, better generalization performance can be expected. The proposed equality constrained optimization based Extreme Learning Machine can be directly used in solving multiclass classification problems. There are two formulations of ELM's equality constrained optimization problems for the multiclass classification case.

Multi-class classifier with single output

Since ELM has the universal approximation capability that can approximate any continuous target functions, the output of ELM $\mathbf{h}(\mathbf{x})\beta$ can be obtained as close to the class labels in the corresponding regions as possible for any classification problems. The optimization problem for the proposed equality constrained optimization method based ELM with single output node can be formulated as:

$$\begin{aligned} &\text{Minimize: } L_{P_{ELM}} = \frac{1}{2}\|\beta\|^2 + C\frac{1}{2}\sum_{i=1}^{N} \xi_i^2 \\ &\text{Subject to: } \mathbf{h}(\mathbf{x}_i)\beta = t_i - \xi_i,\ i = 1,\cdots,N \end{aligned} \tag{3.2}$$

Based on the KKT theorem, the above ELM primal optimization problem can be converted to the following dual optimization problem:

$$L_{D_{ELM}} = \frac{1}{2}\|\beta\|^2 + C\frac{1}{2}\sum_{i=1}^{N} \xi_i^2 - \sum_{i=1}^{N} \alpha_i(\mathbf{h}(\mathbf{x}_i)\beta - t_i + \xi_i) \tag{3.3}$$

where each Lagrange multiplier α_i corresponds to the ith training sample. The KKT optimality conditions of (3.3) can be obtained as follows:

$$\frac{\partial L_{D_{ELM}}}{\partial \beta} = 0 \rightarrow \beta = \sum_{i=1}^{N} \alpha_i \mathbf{h}(\mathbf{x}_i)^T = \mathbf{H}^T \alpha \tag{3.4a}$$

$$\frac{\partial L_{D_{ELM}}}{\partial \xi_i} = 0 \rightarrow \alpha_i = C\xi_i,\ i = 1, \ldots, N \tag{3.4b}$$

$$\frac{\partial L_{D_{ELM}}}{\partial \alpha_i} = 0 \rightarrow \mathbf{h}(\mathbf{x}_i)\beta - t_i + \xi_i = 0,\ i = 1, \ldots, N \tag{3.4c}$$

where $\alpha = [\alpha_1, \cdots, \alpha_N]^T$.

Multi-class classifier with multiple outputs

An alternative approach for solving multi-class problems in ELM is to have multiple output nodes instead of a single output node. In this case, m-class classification problems will require m output nodes. The original class label p can be represented in the output vector of the m output nodes as $\mathbf{t}_i = [0, ..., 0, \overset{p}{1}, 0, \cdots, 0]^T$. In this case, only the pth element of $\mathbf{t}_i = [t_{i,1}, \cdots, t_{i,m}]^T$ is one while the rest elements are set zero. In the actual calculation, the position of largest value in the output vector is recorded as the class label. The classification problem for ELM with multiple output nodes is formulated as follows:

$$\begin{aligned} &\text{Minimize: } L_{P_{ELM}} = \frac{1}{2}\|\beta\|^2 + C\frac{1}{2}\sum_{i=1}^{N} \|\xi_i\|^2 \\ &\text{Subject to: } \mathbf{h}(\mathbf{x}_i)\beta = \mathbf{t}_i^T - \xi_i^T,\ i = 1, \cdots, N \end{aligned} \tag{3.5}$$

where $\xi_i = [\xi_{i,1}, \cdots, \xi_{i,m}]^T$ is the training error vector of the m output nodes with respect to the training sample $\mathbf{x}_i$. Based on the KKT theorem, the above ELM primal

optimization problem is equivalent to the following dual optimization problem:

$$\begin{aligned} L_{D_{ELM}} =& \frac{1}{2}\|\beta\|^2 + C\frac{1}{2}\sum_{i=1}^{N}\|\xi_i\|^2 \\ & - \sum_{i=1}^{N}\sum_{j=1}^{m}\alpha_{i,j}\left(\mathbf{h}(\mathbf{x}_i)\beta_j - t_{i,j} + \xi_{i,j}\right) \end{aligned} \tag{3.6}$$

where β_j is the output weight vector that links hidden layer to the jth output node, and $\beta = [\beta_1, \cdots, \beta_m]$. The KKT optimality conditions can then be obtained as follows:

$$\frac{\partial L_{D_{ELM}}}{\partial \beta_j} = 0 \rightarrow \beta_j = \sum_{i=1}^{N}\alpha_{i,j}\mathbf{h}(\mathbf{x}_i)^T \rightarrow \beta = \mathbf{H}^T\alpha \tag{3.7a}$$

$$\frac{\partial L_{D_{ELM}}}{\partial \xi_i} = 0 \rightarrow \alpha_i = C\xi_i,\ i = 1, \ldots, N \tag{3.7b}$$

$$\frac{\partial L_{D_{ELM}}}{\partial \alpha_i} = 0 \rightarrow \mathbf{h}(\mathbf{x}_i)\beta - \mathbf{t}_i^T + \xi_i^T = 0,\ i = 1, \ldots, N \tag{3.7c}$$

where $\alpha_i = [\alpha_{i,1}, \cdots, \alpha_{i,m}]^T$ and $\alpha = [\alpha_1, \cdots, \alpha_N]^T$.

From the above dual optimization problem and KKT optimality conditions of single output node (3.3), (3.4a)-(3.4c), and multiple output nodes (3.6), (3.7a)-(3.7c), it can be seen that single output node case can be considered as a special case of multiple output nodes when the number of output nodes is set as one: $m = 1$. Thus, when dealing with the multi-class classification problems, we may only need to consider the multiple output nodes case. However for both cases, the hidden layer matrix $\mathbf{H}$ remains the same and the size of $\mathbf{H}$ will not be affected by the number of output nodes, it is only determined by the number of training samples N and the number of hidden nodes L.

Different solutions to above KKT optimality conditions can be obtained according to the size of training datasets, thus to minimize the computational complexity.

3.1.1 Case 1: Training dataset is relatively small

In this case, by substituting (3.7a) and (3.7b) into equation (3.7c), we can obtain a simpler equation as:

$$\left(\frac{\mathbf{I}}{C}+\mathbf{H}\mathbf{H}^T\right)\alpha=\mathbf{T} \tag{3.8}$$

where $T=[t_1,\cdots,t_N]^T$.

From equations (3.7a) and (3.8), we can obtain the equation for output weight β

$$\beta=\mathbf{H}^T\left(\frac{\mathbf{I}}{C}+\mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{T} \tag{3.9}$$

The output function of ELM classifier is:

$$\mathbf{f}(\mathbf{x})=\mathbf{h}(\mathbf{x})\beta=\mathbf{h}(\mathbf{x})\mathbf{H}^T\left(\frac{\mathbf{I}}{C}+\mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{T} \tag{3.10}$$

If a feature mapping $\mathbf{h}(\mathbf{x})$ is not explicitly defined, one may apply Mercer's conditions on ELM to implement the kernel concept. The kernel matrix for ELM can be defined as follows:

$$\Omega_{ELM}=\mathbf{H}\mathbf{H}^T:\Omega_{ELMi,j}=h(\mathbf{x}_i)\cdot h(\mathbf{x}_j)=K(\mathbf{x}_i,\mathbf{x}_j) \tag{3.11}$$

Given the kernel matrix, the output function (3.10) of ELM can then be written com-

pactly as follows:

$$\begin{aligned}\mathbf{f}(\mathbf{x}) &= \mathbf{h}(\mathbf{x})\mathbf{H}^T\left(\frac{\mathbf{I}}{C}+\mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{T} \\ &= \begin{bmatrix} K(\mathbf{x},\mathbf{x}_1) \\ \vdots \\ K(\mathbf{x},\mathbf{x}_N)\end{bmatrix}^T \left(\frac{\mathbf{I}}{C}+\Omega_{ELM}\right)^{-1}\mathbf{T}\end{aligned} \tag{3.12}$$

Similar to the kernel approach used in SVM, LS-SVM and PSVM, the feature mapping $\mathbf{h}(\mathbf{x})$ in ELM need not be known to users, instead its corresponding kernel $K(\mathbf{u},\mathbf{v})$ (e.g., $K(\mathbf{u},\mathbf{v}) = \exp(-\gamma\|\mathbf{u}-\mathbf{v}\|^2)$) is given to users. In this case, the dimensionality L (number of hidden nodes)of the feature space need not be defined either.

3.1.2 Case 2: Training dataset is very large

When the size of the training dataset is much larger than the dimensionality of the feature space, $N \gg L$, it is very computational costly if we employ the solution given in case 1, and in most cases, the normal computer may not handle such computation, thus we come out an alternative solution. From equations (3.4a) and (3.4b) we have

$$\beta = C\mathbf{H}^T\xi \tag{3.13}$$

$$\xi = \frac{1}{C}\left(\mathbf{H}^T\right)^{\dagger}\beta \tag{3.14}$$

From equation (3.4c), we have

$$\begin{aligned}&\mathbf{H}\beta - \mathbf{T} + \frac{1}{C}\left(\mathbf{H}^T\right)^{\dagger}\beta = 0 \\ &\mathbf{H}^T\left(\mathbf{H}+\frac{1}{C}\left(\mathbf{H}^T\right)^{\dagger}\right)\beta = \mathbf{H}^T\mathbf{T} \\ &\beta = \left(\frac{\mathbf{I}}{C}+\mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T}\end{aligned} \tag{3.15}$$

In this case, the output function of ELM classifier is:

$$\mathbf{f}(\mathbf{x}) = \mathbf{h}(\mathbf{x})\beta = \mathbf{h}(\mathbf{x})\left(\frac{\mathbf{I}}{C} + \mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T} \tag{3.16}$$

In theory, the above two solutions can be used for any size of applications if not considering the computational costs. However, the difference in computational cost can be huge in different applications especially when the datasets are large. During the simulation of ELM, it is found that the generalization performance of ELM is not very sensitive to the dimensionality of the feature space L and good performance can be obtained as long as L is chosen large enough for a specific application. In our simulations, for the moderate to large datasets, L is set as 1000. Thus, for the large training datasets ($N \gg L$), one may prefer to use solution (3.16) in order to reduce computational costs. However, if a feature mapping $\mathbf{h}(\mathbf{x})$ is unknown to users, one may prefer to apply solution (3.12), where the kernel concept is introduced.

3.1.3 Difference from other regularized ELMs

There are also efforts done on adding regularization terms into ELM's solutions to prevent possible singularity problems during computation. Toh [17] and Deng, et al [16] proposed two different types of weighted regularized ELMs.

The total error rate ELM (TER-ELM) by Toh et al. [17] uses m output nodes for m-class classification problems. In TER-ELM, the counting cost function adopts a quadratic approximation approach. However, TER-ELM is not used directly to solve multi-class classification problems, instead the one-against-all (OAA) approach is implemented. Similar to other OAA approach of solving multi-class classificaiton problems, TER-ELM consists of m binary TER-ELM where jth TER-ELM is trained with all the samples in the jth class with positive labels, and all the other examples from the remaining $m-1$ classes with negative labels. Suppose that there are m_j^+ number of positive category patterns and m_j^- number of negative category patterns in the jth binary TER-

ELM, we have a positive class output $y_j^+ = (\tau+\eta)\mathbf{1}_j^+$ for the jth class of samples, and a negative class output $y_j^- = (\tau-\eta)\mathbf{1}_j^-$ for all the non-jth class of samples, where $\mathbf{1}_j^+ = [1,\cdots,1]^T \in \mathbf{R}^{m_j^+}$ and $\mathbf{1}_j^- = [1,\cdots,1]^T \in \mathbf{R}^{m_j^-}$. The threshold ($\tau$) and bias ($\eta$) can be set the same for all the m outputs. The output weight vector β_j in the jth binary TER-ELM can be calculated as:

$$\begin{aligned}\beta_j = &\left(\frac{1}{m_j^-}\mathbf{H}_j^{-T}\mathbf{H}_j^- + \frac{1}{m_j^+}\mathbf{H}_j^{+T}\mathbf{H}_j + \right)^{-1} \\ &\cdot\left(\frac{1}{m_j^-}\mathbf{H}_j^{-T}y_j^- + \frac{1}{m_j^+}\mathbf{H}_j^{+T}y_j^+\right)\end{aligned} \tag{3.17}$$

where $\mathbf{H}_j^+$ denotes the hidden layer matrices of the positive samples in jth binary TER-ELM and $\mathbf{H}_j^-$ denotes the hidden layer matrices of the negative samples.

By defining two class-specific diagonal weighting matrices $\mathbf{W}_j^+ = \text{diag}(0,\cdots,0,1/m_j^+,\cdots,m_j^+)$ and $\mathbf{W}_j^- = \text{diag}(1/m_j^-,\cdots,m_j^-,0,\cdots,0)$, the solution of TER-ELM (3.17) can be written as

$$\beta_j = \left(\frac{\mathbf{I}}{C} + \mathbf{H}_j^T\mathbf{W}_j\mathbf{H}_j\right)^{-1}\mathbf{H}_j^T\mathbf{W}_j\mathbf{y}_j \tag{3.18}$$

where $\mathbf{W}_j = \mathbf{W}_j^+ + \mathbf{W}_j^-$ and the elements of $\mathbf{H}_j$ and $\mathbf{y}_j$ are ordered according to the positive and negative samples of the two classes (jth class samples and all the non-jth class samples). In order to improve the stability of the learning and prevent any possible singularity problems, $\frac{\mathbf{I}}{C}$ is introduced in the above solution. If the dimensionality of the hidden layer L is much larger than the number of the training data N, an alternative solution can be given as:

$$\beta_j = \mathbf{H}_j^T\left(\frac{\mathbf{I}}{C} + \mathbf{W}_j\mathbf{H}_j\mathbf{H}_j^T\right)^{-1}\mathbf{W}_j\mathbf{y}_j \tag{3.19}$$

However in TER-ELM, kernels and generalized feature mappings are not considered.

Deng, et al [16] mainly focuses on the case where $L \ll N$, and the solution of ELM

(3.15) and the solution of Deng, et al [16] look similar to each other. However, different from ELM solutions provided in this chapter, Deng, et al does not consider kernels and generalized feature mappings in their weighted regularized ELM. In the proposed solutions of ELM, the L hidden nodes may have different type of hidden node output function $h_i(\mathbf{x})$: $\mathbf{h}(\mathbf{x}) = [h_1(\mathbf{x}), \cdots, h_L(\mathbf{x})]$, while in [16] all the hidden nodes use the sigmoid type of activation functions. Deng, et al does not handle the alternative solution (3.9).

Seen from that (3.15), the multivariate polynomial model by Toh et al. [51] can be considered as a special case of ELM.

The original solution of ELM, TER-ELM and the weighted regularized ELM are not able to apply kernels in their implementations. While with the new approach in (3.12), kernels can be used in ELM.

3.2 Comparison between ELM and LS-SVM/PSVM

In this section, the differences between the equality constrained optimization method based ELM and LS-SVM/PSVM are analyzed. Due to different feature mappings, ELM tends to have less optimization constraints in the optimization equation. The simpler optimization solution of ELM results in a better scability for ELM than least sqaure SVM and proximal SVM.

3.2.1 Feature mappings and kernel matrices

Different from SVM, LS-SVM and PSVM, the feature mapping $\mathbf{h}(\mathbf{x}) = [h_1(\mathbf{x}), \cdots, h_L(\mathbf{x})]$ in ELM is usually explicitly defined and known to users. According to [9, 10], there exist many nonlinear piecewise continuous functions that can be used as the hidden node activation functions, and thus the ELM feature mappings can be very diversified.

The ELM feature mapping $\mathbf{h}(\mathbf{x})$ is usually defined as:

$$\mathbf{h}(\mathbf{x}) = [G(\mathbf{a}_1, b_1, \mathbf{x}), \cdots, G(\mathbf{a}_L, b_L, \mathbf{x})] \tag{3.20}$$

where $G(\mathbf{a}, b, \mathbf{x})$ is a nonlinear piecewise continuous function satisfying ELM universal approximation capability theorems [8, 9, 10] and $\{(\mathbf{a}_i, b_i)\}_{i=1}^{L}$ are randomly generated according to any continuous probability distribution. For example, such nonlinear piecewise continuous functions can be

i) Sigmoid function:

$$G(\mathbf{a}, b, \mathbf{x}) = \frac{1}{1 + \exp(-(\mathbf{a} \cdot \mathbf{x} + b))} \tag{3.21}$$

ii) Hardlimit function:

$$G(\mathbf{a}, b, \mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{a} \cdot \mathbf{x} - b \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{3.22}$$

iii) Gaussian function:

$$G(\mathbf{a}, b, \mathbf{x}) = \exp(-b\|\mathbf{x} - \mathbf{a}\|^2) \tag{3.23}$$

iv) Multiquadrics function:

$$G(\mathbf{a}, b, \mathbf{x}) = (\|\mathbf{x} - \mathbf{a}\|^2 + b^2)^{1/2} \tag{3.24}$$

Sigmoid function and Gaussian function are two commonly used hidden layer output functions in the feedforward neural networks and RBF networks respectively. According to our observation, ELM with hardlimit [52] and multiquadrics activation functions can also obtain good generalization performance.

Suykens et al. [53] also proposed a training method for SLFNs that uses SVM's feature

mapping as the hidden layer feature mapping. In their implementation of SVM's feature mapping, the hidden layer parameters need to be tuned iteratively through solving an optimization problem. Their proposed learning algorithm can be described as follows:

$$\text{minimize: } \|r\mathbf{w}\|^2 \tag{3.25}$$

subject to:

C1: QP subproblem:

$$\begin{aligned} &\mathbf{w} = \sum_{i=1}^{N} \alpha_i^* t_i \tanh(\mathbf{V}\mathbf{x}_i + \mathbf{B}) \\ &\alpha_i^* = \arg\max_{\alpha_i} \mathscr{Q}(\alpha_i; \tanh(\mathbf{V}\mathbf{x}_i + \mathbf{B})) \\ &0 \le \alpha_i^* \le c \end{aligned} \tag{3.26}$$

C2: $\|\mathbf{V}(:);\mathbf{B}\|_2 \le \gamma$

C3: r is radius of smallest ball containing$\{\tanh(\mathbf{V}\mathbf{x}_i) + \mathbf{B}\}_{i=1}^{N}$

where $\mathbf{V}$ denotes the interconnection matrix for the hidden layer, $\mathbf{B}$ is the bias vector, $(:)$ represents a column-wise scan of $\mathbf{V}$, and γ is a positive constant. And $\mathscr{Q}$ is the cost function of the corresponding SVM dual problem as seen below:

$$\max_{\alpha_i} \mathscr{Q}(\alpha_i; K(\mathbf{x}_i, \mathbf{x}_j)) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} t_i t_j \alpha_i \alpha_j K(\mathbf{x}_i, \mathbf{x}_j) + \sum_{i=1}^{N} \alpha_i \tag{3.27}$$

In Suykens's approach [53], QP subproblems need to be solved for hidden node parameters $\mathbf{V}$ and $\mathbf{B}$ while in ELM, the hidden node parameters are randomly generated and the hidden layer feature mapping is known to users.

We can formulate the feature mapping matrix for ELM as $\mathbf{H} = [\mathbf{h}(\mathbf{x}_1)^T, \cdots, \mathbf{h}(\mathbf{x}_N)^T]^T$. During the feature mapping of ELM, $\mathbf{h}(\mathbf{x}_i)$ maps the data $\mathbf{x}_i$ from the original input space to hidden layer feature space and it does not require any information from target t_i, hence

the feature mapping matrix for ELM also does not require any prior knowledge about the target $\mathbf{T}$. However, for least square SVM and proximal SVM, the feature mapping matrix $\mathbf{Z} = [t_1\phi(\mathbf{x}_1)^T, \cdots, t_N\phi(\mathbf{x}_N)^T]^T$ (2.26) requires the information of target labels t_i. It is reasonable to have a feature mapping that is independent from the target value thus it can be more easily implemented and computationally flexible.

In ELM, the kernel matrix can be explicitly defined as in (3.11) or implicitly defined such as Gaussian kernel matrix. However, in LS-SVM and PSVM, the kernel matrix shown in (2.27) and (2.32) are usually implicitly defined and also depend on the target labels.

3.2.2 Optimization constraints

In least square SVM and proximal SVM, the data samples are usually mapped to a higher dimensional space $\phi(\mathbf{x})$, then in the mapping space, they are separated by a hyperplane: $\phi(\mathbf{x}_i) + b = 0$. Here the bias term b is required as the separation hyperplane may not necessarily pass through the origin in the mapped space. Poggio, et al [54] proves in theory that the term bias b is not required in positive definite kernel and it is not incorrect to have the term bias b in SVM model. However, in ELM mapping space, the separation hyperplane for ELM trends to pass through the origin, and thus no bias required. This difference is also apparent from their target approximation equations: $\phi(\mathbf{x}_i) + b = t_i$ for LS-SVM/PSVM and $\mathbf{h}(\mathbf{x}_i)\beta = t_i$ for ELM. Thus when applied to the optimization equation of each method, the corresponding KKT condition for LS-SVM is $\sum_{i=1}^{N} \alpha_i t_i = 0$ while ELM does not have to be bounded with this constraint. Since ELM and LS-SVM/PSVM have the same primal optimization equation, and ELM has milder optimization constraints, ELM trends to achieve a better generalization performance than LS-SVM and PSVM.

3.2.3 Computational complexity and scalability

For LS-SVM and PSVM, the major computational cost comes from the calculation of the Lagrange multipliers α based on equation (2.25) and equation (2.31). From equation (3.8), it is obvious that ELM computes α based on a simpler method. when solving large scale applications, ELM's solution (3.15) uses $\mathbf{H}^T\mathbf{H}$ (size: $L \times L$) instead of $\mathbf{H}\mathbf{H}^T$ (size: $N \times N$). For large scale applications, usually the number of hidden nodes L is much smaller than the number of training samples: $L \ll N$, hence the computation cost cab be reduced dramatically. For the case $L \ll N$, compared with LS-SVM and PSVM which must use $\mathbf{H}\mathbf{H}^T$ (size: $N \times N$), ELM has much better computational scalability when the number of the training samples N increases as shown in Figure 3.1.

There are also efforts done on reducing the computational cost of LS-SVM for large scale problems. Suykens et al. [55, 56, 57, 58, 59] proposed the fixed-size LS-SVM that uses a M-sample subset of the original training dataset ($M \ll N$) to compute a finite dimensional approximation of the original feature map $\phi(\mathbf{x})$. However, different from fixed-size LS-SVM, in the case $L \ll N$, ELM's solution (3.15) still uses the entire N training samples. In any case, the feature map $\mathbf{h}(\mathbf{x})$ of ELM is based on the entire training samples not approximated samples. The kernel matrix of fixed-size LS-SVM is built based on approximation with subset of size $M \ll N$ while the kernel matrix of ELM is built with the entire dataset.

3.2.4 Unified learning model for all applications

The solutions given in (3.10) and (3.16) can be applied to solve regression, binary and multi-class classification directly. The kernel matrix $\Omega_{ELM} = \mathbf{H}\mathbf{H}^T$ is only related to the input data $\mathbf{x}_i$ and independent of the number of output nodes m or the training target values t_i. However, in multi-class LS-SVM besides the input data $\mathbf{x}_i$, the kernel matrix Ω_M (3.30) is also related to the number of output nodes m and the training target values t_i.

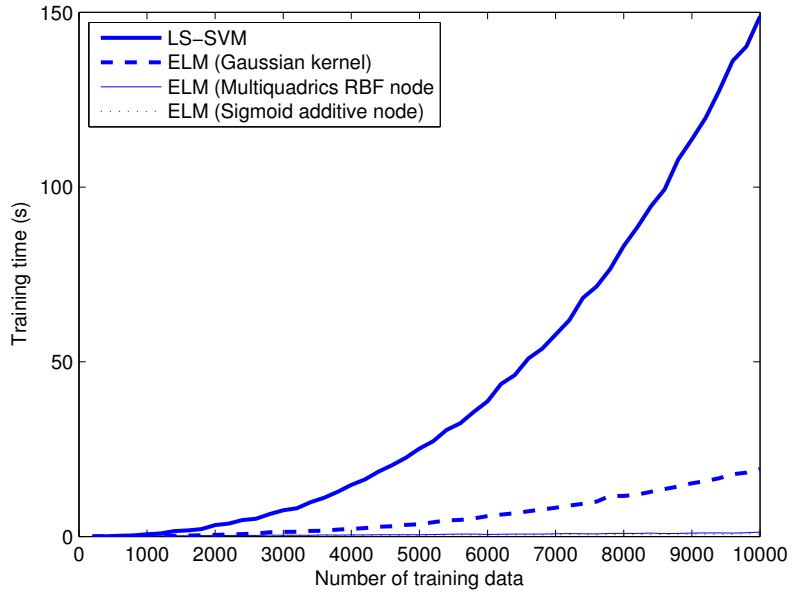


Figure 3.1: Scalability analysis on Letter dataset: when the number of training data increases, the training time spent by LS-SVM and ELM (Gaussian kernel) increases sharply, while the training time spent by ELM with sigmoid additive node and multiquadrics function node increases very slowly.

For the multi-class case with m labels, LS-SVM uses m output nodes in order to encode multiple classes where $t_{i,j}$ denotes the value of the jth output node for the training data $\mathbf{x}_i$ [60]. The m outputs can be used to encode a maximum of 2^m different classes. For multi-class case, the primal optimization problem of LS-SVM is given as [60]:

$$\text{Minimize: } L^{(m)}_{P_{LS-SVM}} = \frac{1}{2}\sum_{j=1}^{m}\mathbf{w}_j\cdot\mathbf{w}_j + C\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{m}\xi^2_{i,j}$$

$$\text{Subject to: } \begin{cases} t_{i,1}(\mathbf{w}_1\cdot\phi_1(\mathbf{x}_i)+b_1) = 1-\xi_{i,1} \\ t_{i,2}(\mathbf{w}_2\cdot\phi_2(\mathbf{x}_i)+b_2) = 1-\xi_{i,2} \\ \cdots \\ t_{i,m}(\mathbf{w}_m\cdot\phi_m(\mathbf{x}_i)+b_m) = 1-\xi_{i,m} \end{cases} \tag{3.28}$$

$$i = 1,\ldots,N$$

Similar to the LS-SVM solution (2.25) for the binary classification case, with KKT optimality conditions the corresponding LS-SVM solution for multi-class cases can be obtained as follows:

$$\begin{bmatrix} \mathbf{0} & \mathbf{T}^T \\ \mathbf{T} & \Omega_M \end{bmatrix} \begin{bmatrix} \mathbf{b}_M \\ \alpha_M \end{bmatrix} = \begin{bmatrix} \mathbf{0} \\ \vec{\mathbf{1}} \end{bmatrix} \tag{3.29}$$

$$\begin{aligned} &\Omega_M = \text{blockdiag}\left[\Omega^{(1)} + \frac{\mathbf{I}}{C}, \ldots, \Omega^{(m)} + \frac{\mathbf{I}}{C}\right] \\ &\Omega_{kl}^{(j)} = t_{k,j} t_{l,j} K^{(j)}(\mathbf{x}_k, \mathbf{x}_l) \\ &\mathbf{b}_M = [b_1, \ldots, b_m] \\ &\alpha_M = [\alpha_{1,1}, \ldots, \alpha_{N,1}, \ldots, \alpha_{1,m}, \ldots, \alpha_{N,m}] \end{aligned} \tag{3.30}$$

$$\begin{aligned} K^{(j)}(\mathbf{x}_k, \mathbf{x}_l) =& \phi_j(\mathbf{x}_k) \cdot \phi_j(\mathbf{x}_l) = \exp\left(-\frac{\|\mathbf{x}_k - \mathbf{x}_l\|^2}{\sigma_j^2}\right), \\ & j = 1, \ldots, N \end{aligned} \tag{3.31}$$

Seen from (3.30) multi-class LS-SVM needs to use m binary-class LS-SVM concurrently for m class labels of classifications, each of the m binary-class LS-SVMs may have different kernel matrix $\Omega^{(j)}$, $j = 1, \cdots, m$. However, ELM has only one hidden layer that is linked to all the m output nodes. In multi-class LS-SVM, different kernels may be used in different binary LS-SVMs, and the jth LS-SVM uses kernel $K^{(j)}(\mathbf{u}, \mathbf{v})$. Take gaussian kernel as an example, $K^{(j)}(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{\|\mathbf{x}_k - \mathbf{x}_l\|^2}{\sigma_j^2}\right)$, during the actual implementation, it would be very time-consuming and tedious for users to choose different kernel parameters σ_i, and thus one may want to set a common value $\sigma_i = \sigma$ for all the kernels. In multi-class LS-SVM, the size of Ω_M is $N \times Nm$, which is related to the

Table 3.1: Datasets for binary classification problems

Datasets	# train	# test	# features	Random Perm
Diabetes	512	256	8	Yes
Australian Credit	460	230	6	Yes
Liver	230	115	6	Yes
Banana	400	4900	2	No
Colon	30	32	2000	No
Colon (Gene Sel)	30	32	60	No
Leukemia	38	34	7129	No
Leukemia (Gene Sel)	38	34	60	No
Brightdata	1000	1462	14	Yes
Dimdata	1000	3192	14	Yes
Mushroom	1500	6624	22	Yes
Adult	4781	27780	123	No

number of output nodes m. However in ELM, the size of kernel matrix $\Omega_{ELM} = \mathbf{H}\mathbf{H}^T$ is $N \times N$, which is independent of output nodes m. The kernel matrix size is fixed for all the regression, binary and multi-class classification problems.

3.3 Performance Verification

This section compares the performance of the proposed ELM algorithm with SVM and LS-SVM on real-world benchmark datasets including regression, binary and multi-class classification problems. In order to show the performance of the proposed ELM with various feature mappings on super small datasets, we also tested ELM on the XOR problem.

Table 3.2: Datasets for multi-class classification problems

Datasets	# train	# test	# features	# classes	Random Perm
Iris	100	50	4	3	Yes
Glass	142	72	9	6	Yes
Wine	118	60	13	3	Yes
Ecoli	224	112	7	8	Yes
Vowel	528	462	10	11	No
Vehicle	564	282	18	4	Yes
Segment	1540	770	19	7	Yes
Satimage	4435	2000	36	6	No
DNA	2000	1186	180	3	No
Letter	13333	6667	16	26	Yes
Shuttle	43500	14500	9	7	No
USPS	7291	2007	256	10	No

Table 3.3: Datasets for regression problems

Datasets	# train	# test	# features	Random Perm
Baskball	64	32	4	Yes
Cloud	72	36	9	Yes
Autoprice	106	53	9	Yes
Strike	416	209	6	Yes
Pyrim	49	25	27	Yes
Bodyfat	168	84	14	Yes
Cleveland	202	101	13	Yes
Housing	337	169	13	Yes
Balloon	1334	667	2	Yes
Quake	1452	726	3	Yes
Space-ga	2071	1036	6	Yes
Abalone	2784	1393	8	Yes

3.3.1 Benchmark datasets

In order to extensively verify the performance of different algorithms, we tested on a wide type of datasets during simulations. The datasets consist of *small sizes*, *low dimensions*, *large sizes* and/or *high dimensions*. There are totally 36 datasets including 12 binary classification datasets, 12 multi-classification datasets and 12 regression datasets. Most of the datasets are taken from UCI Machine Learning Repository [62] and Statlib [61].

Binary class datasets

The 12 binary classification datasets shown in Table 3.1 can be classified into four groups:

i) Datasets with relatively small number of samples and low dimensions, e.g. Pima Indians *diabetes*, Statlog *Australian credit*, Bupa *Liver* disorders [62] and *Banana* [63];

ii) Datasets with relatively small number of samples and high dimensions, e.g. *leukemia* dataset [64] and *colon* microarray dataset [65];

iii) Datasets with relatively large number of samples and low dimensions, e.g. Star/Galaxy-*Bright* dataset [66], Galaxy *Dim* dataset [66] and *mushroom* dataset [62];

iv) Datasets with large number of samples and high dimensions, e.g. *adult* dataset [62].

The leukemia dataset is built from a collection of leukemia patient samples [67]. The dataset consists of 72 samples: 25 samples of AML, and 47 samples of ALL. Each sample of the leukemia dataset is measured over 7,129 genes (number of features for each sample). The colon microarray dataset consists of 22 normal and 40 tumor tissue samples, and each sample of colon microarray dataset contains 2,000 genes (number of features for each sample). Performance of different algorithms were also tested

on both leukemia dataset and colon microarray dataset after the minimum-redundancy-maximum-relevance (MRMR) feature selection method [68] being taken on them.

Multi-class datasets

The 12 multi-class datasets shown in Table 3.2 can be classified into four groups:

i) Datasets with relatively small number of samples and low dimensions, e.g. *Iris*, *Glass* Identification and *Wine* [62];

ii) Datasets with relatively medium number of samples and medium dimensions, e.g. *Vowel* Recognition, Statlog *Vehicle* Silhouettes and Statlog Image *Segmentation* [62];

iii) Datasets with relatively large number of samples and medium dimensions, e.g. *letter* and *shuttle* [62];

iv) Datasets with large number of samples and/or large dimensions, e.g. *DNA*, *Satimage* [62] and USPS [64].

Regression datasets

The 12 regression datasets shown in Table 3.3 can be classified into three groups:

i) Datasets with relatively small number of samples and low dimensions, e.g. *Basketball*, *Strike* [61], *Cloud*, and *Autoprice* [62];

ii) Datasets with relatively small number of samples and medium dimensions, e.g. *Pyrim*, *Housing* [62], *Bodyfat*, and *Cleveland* [61];

iii) Datasets with relatively large number of samples and low dimensions, e.g. *Balloon*, *Quake*, *Space-ga* [61] and *Abalone* [62].

The "random perm" shown in Tables 3.1, 3.2 and 3.3 indicates whether the training and testing data samples are reshuffled and the training and testing data files are re-constructed at each trial of simulation. If the training and testing data of the datasets remain fixed for all trials of simulations, it is marked as "No", otherwise, it is marked as "Yes".

3.3.2 Simulation environment settings

The simulations of different algorithms on most of the datasets (except for *Adult*, *Letter*, *Shuttle* and *USPS* datasets) are carried out in MATLAB 7.0.1 environment running in Core 2 Quad, 2.66GHZ CPU with 2GB RAM. The codes used for SVM and LS-SVM are downloaded from [69] and [70], respectively. Simulations on large datasets (*Adult*, *Letter*, *Shuttle* and *USPS*) are carried out in a high performance computer with 2.52GHz CPU and 48GB RAM. In Tables 3.5 - 3.6, the symbol "*" indicates that the corresponding datasets are tested in such a high performance computer.

3.3.3 User specified parameters

Gaussian kernel function $K(\mathbf{u},\mathbf{v}) = \exp(-\gamma\|\mathbf{u}-\mathbf{v}\|^2)$ is used in SVM, LS-SVM and Kernel based ELM. The performances of ELM with Sigmoid type of additive hidden nodes and Multiquadrics RBF hidden nodes are also tested .

For SVM, LS-SVM and Kernel based ELM, the cost parameter C and kernel parameter γ need to be chosen appropriately in order to achieve good generalization performance. A wide range of C and γ are tested in our simulation. For each dataset we did a grid search of 50 different values of C and 50 different values of γ resulting in a total of 2500 pairs of (C,γ). The 50 different values of C and γ are in the range $\{2^{-24},2^{-23},\cdots,2^{24},2^{25}\}$.

Table 3.4: Parameters of the conventional SVM, LS-SVM and ELM.

Datasets	SVM (Gaussian Kernel)		LSSVM (Gaussian Kernel)		Extreme Learning Machine					
					Gaussian Kernel		Sigmoid Additive Node		Multiquadrics RBF Node	
	C	γ	C	γ	C	γ	C	L	C	L
Binary class datasets										
Diabetes	2^{10}	2^{4}	2^{10}	2^{10}	2^{10}	2^{5}	2^{-2}	1000	2^{-2}	1000
Australian Credit	2^{14}	2^{4}	2^{7}	2^{10}	2^{10}	2^{9}	2^{-1}	1000	2^{-1}	1000
Liver	2^{18}	2^{4}	2^{5}	2^{7}	2^{8}	2^{5}	2^{1}	1000	2^{2}	1000
Banana	2^{25}	2^{2}	2^{14}	2^{2}	2^{0}	2^{4}	2^{22}	1000	2^{2}	1000
Colon	2^{12}	2^{6}	2^{4}	2^{16}	2^{7}	2^{20}	2^{1}	1000	2^{-7}	1000
Colon (Gene Sel)	2^{2}	2^{4}	2^{0}	2^{12}	2^{7}	2^{20}	2^{-14}	1000	2^{-13}	1000
Leukemia	2^{12}	2^{10}	2^{10}	2^{6}	2^{15}	2^{20}	2^{17}	1000	2^{13}	1000
Leukemia (Gene Sel)	2^{8}	2^{8}	2^{10}	2^{10}	2^{15}	2^{0}	2^{6}	1000	2^{-7}	1000
Brightdata	2^{12}	2^{4}	2^{2}	2^{3}	2^{4}	2^{-4}	2^{5}	1000	2^{5}	1000
Dimdata	2^{14}	2^{4}	2^{3}	2^{3}	2^{5}	2^{8}	2^{1}	1000	2^{1}	1000
Mushroom	2^{20}	2^{0}	2^{3}	2^{3}	2^{13}	2^{4}	2^{19}	1000	2^{19}	1000
Adult	2^{2}	2^{2}	2^{0}	2^{7}	2^{25}	2^{18}	2^{-6}	1000	2^{-6}	1000
Multi-class datasets										
Iris	1	2^{-2}	2^{14}	2^{6}	2^{0}	2^{0}	2^{5}	1000	2^{-3}	1000
Glass	1	2^{-2}	2^{2}	2^{2}	2^{5}	2^{0}	2^{3}	1000	2^{2}	1000
Wine	1	1	2^{16}	2^{14}	2^{5}	2^{3}	2^{0}	1000	2^{-1}	1000
Ecoli	2^{16}	2^{5}	2^{5}	2^{5}	2^{22}	2^{12}	2^{2}	1000	2^{0}	1000
Vowel	1	1	2^{8}	2^{2}	2^{5}	2^{-1}	2^{1}	1000	2^{0}	1000
Vehicle	2^{14}	2^{2}	2^{33}	2^{20}	2^{6}	2^{3}	2^{7}	1000	2^{10}	1000
Segment	2^{25}	2^{5}	2^{6}	2^{2}	2^{13}	2^{-5}	2^{10}	1000	2^{17}	1000
Satimage	1	1	2^{8}	2^{2}	2^{4}	2^{-2}	2^{7}	1000	2^{12}	1000
DNA	2^{18}	2^{12}	2^{12}	2^{8}	2^{6}	2^{6}	2^{-7}	1000	2^{1}	1000
Letter	2^{10}	2^{-4}	2^{8}	2^{0}	2^{3}	2^{-2}	2^{10}	1000	2^{24}	1000
shuttle	2^{10}	2^{-2}	2^{18}	2^{-4}	2^{20}	2^{-10}	2^{25}	1000	2^{25}	1000
USPS	2^{8}	2^{0}	2^{5}	2^{8}	2^{4}	2^{8}	2^{10}	1000	2^{20}	1000
Regression datasets										
Baskball	2^{0}	2^{0}	2^{0}	2^{3}	2^{0}	2^{0}	2^{0}	1000	2^{-4}	1000
Cloud	2^{2}	2^{0}	2^{-18}	2^{-17}	2^{15}	2^{9}	2^{-5}	1000	2^{-14}	1000
Autoprice	2^{13}	2^{5}	2^{5}	2^{2}	2^{6}	2^{4}	2^{-1}	1000	2^{6}	1000
Strike	2^{0}	2^{-4}	2^{0}	2^{-2}	2^{-1}	2^{5}	2^{-5}	1000	2^{8}	1000
Pyrim	2^{10}	2^{8}	2^{5}	2^{7}	2^{2}	2^{6}	2^{-3}	1000	2^{3}	1000
Bodyfat	2^{25}	2^{7}	2^{25}	2^{20}	2^{12}	2^{16}	2^{0}	1000	2^{6}	1000
Cleveland	2^{2}	2^{2}	2^{8}	2^{14}	2^{13}	2^{15}	2^{-3}	1000	2^{1}	1000
Housing	2^{4}	2^{2}	2^{4}	2^{4}	2^{2}	2^{8}	2^{5}	1000	2^{7}	1000
Balloon	2^{4}	2^{-2}	2^{10}	2^{0}	2^{-6}	2^{10}	2^{20}	1000	2^{15}	1000
Quake	2^{5}	2^{5}	2^{5}	2^{14}	2^{5}	2^{14}	2^{0}	1000	2^{10}	1000
Space-ga	2^{8}	2^{-1}	2^{8}	2^{2}	2^{2}	2^{20}	2^{4}	1000	2^{13}	1000
Abalone	2^{1}	2^{-1}	2^{4}	2^{4}	2^{0}	2^{0}	2^{0}	1000	2^{0}	1000

It is commonly known that the performance of SVM is sensitive to the parameters (C, γ). Similar to SVM, the generalization performance of LS-SVM and Kernel based ELM also depend closely on the combination of (C, γ). As shown in Figure 3.2, the performance of LS-SVM and ELM with Gaussian kernel are very sensitive on the user specified parameters (C, γ). The parameters that result in best generalization performance of SVM, LS-SVM, and ELM are usually in a very narrow range. Thus, the best combination of (C, γ) of SVM, LS-SVM, and ELM with Gaussian kernel need to be chosen properly and differently for each dataset.

For ELM with Sigmoid additive hidden node and Multiquadrics RBF hidden node (e.g. $\mathbf{h}(\mathbf{x}) = [G(\mathbf{a}_1, b_1, \mathbf{x}), \cdots, G(\mathbf{a}_L, b_L, \mathbf{x})]$ where $G(\mathbf{a}, b, \mathbf{x}) = 1/(1 + \exp(-(\mathbf{a} \cdot \mathbf{x} + b)))$ for Sigmoid additive hidden node or $G(\mathbf{a}, b, \mathbf{x}) = (\|\mathbf{x} - \mathbf{a}\|^2 + b^2)^{1/2}$ for Multiquadrics RBF hidden node), all the hidden node parameters $(\mathbf{a}_i, b_i)_{i=1}^{L}$ are randomly generated based on uniform distribution. The parameters that need to be tuned are (C, L). C is the regularization parameter and chosen from the range $\{2^{-24}, 2^{-23}, \cdots, 2^{24}, 2^{25}\}$. As shown in Figure 3.3, ELM can achieve good generalization performance as long as the number of hidden node L is set large enough. In all the simulations on ELM with Sigmoid additive hidden node and Multiquadrics RBF hidden node, we set $L = 1000$. In other words, the performance ELM with Sigmoid additive hidden node and Multiquadrics RBF hidden node is not very sensitive to the number of hidden nodes L so that L need not be specified by users, instead users only need to specify one parameter: C. For each dataset, to show a stable convincing result, fifty trials were conducted for ELM with Sigmoid additive hidden node and Multiquadrics RBF hidden node. Simulation results including the average testing accuracy, standard deviation of the testing accuracy, and the training time are given in this section.

The user specified parameters chosen during our simulations are given in Table 3.4.

3.3.4 Performance comparison on XOR problem

The performance of SVM, LS-SVM and ELM were also tested on the XOR problem which only has two training samples in each class. The aim of this simulation is to verify whether ELM can handle some rare cases well such as the cases with extremely few training samples. Figure 3.4 shows the classification boundaries of different classifiers for XOR problem. It can be seen from the separation boundaries that similar to SVM and LS-SVM, ELM is able to solve the XOR problem well. User specified parameters used in this XOR problem are as follows: (C, γ) for SVM is $(2^{10}, 2^0)$, (C, γ) for LS-SVM is $(2^4, 2^{14})$, (C, γ) for ELM with Gaussian kernel is $(2^5, 2^{15})$, and (C, L) for ELM with Sigmoid additive hidden node is $(2^0, 3000)$.

3.3.5 Performance comparison on real-world benchmark datasets

The performance of SVM, LS-SVM, ELM with Gaussian kernel, random Sigmoid hidden nodes and Multiquadrics RBF nodes are shown in tables 3.5 - 3.7. It is observed that ELM with random Sigmoid hidden nodes and Multiquadrics RBF nodes can always achieve comparable performance as SVM and LS-SVM but with much faster learning speed. Seen from Tables 3.5 - 3.7, different solutions of ELM can be used for different size of datasets in order to reduce the computational cost.

The training speed for ELM is generally much faster than SVM and LS-SVM. Take Shuttle (*large number of training samples*) and USPS (*medium number of training samples with high input dimensions*) datasets of Table 3.6 as examples:

i) For Shuttle datasets, ELM with Gaussian kernel and random Multiquadrics RBF nodes runs 6 and 4466 times faster than LS-SVM, respectively;

ii) For USPS datasets, ELM with Gaussian kernel and random Multiquadrics RBF nodes runs 6 and 65 times faster than LS-SVM, respectively and runs 11 and 146 times faster than SVM, respectively.

On the other hand, different from LS-SVM which is sensitive to the pair of parameters (C, γ), ELM with random Multiquadrics RBF nodes does not show sensitive differences on different the user specified parameter C (cf Figure 3.3(b)) and thus can be easily implemented in many applications.

The performance comparison between LS-SVM and ELM with Gaussian kernel were especially highlighted in Tables 3.5 - 3.7. The apparently better testing results were highlighted in boldface between the two algorithms. It can be seen that ELM with Gaussian kernel achieves the same generalization performance in almost all the binary classification and regression cases as LS-SVM at much faster learning speeds. However, as shown in Table 3.6, ELM usually achieves much better generalization performance in multi-class classification cases than LS-SVM.

Figure 3.5 shows the boundaries of different classifiers in Banana case. It can be seen that ELM can classify different classes well.

3.4 Summary

This chapter shows that under the ELM framework, both LS-SVM and PSVM can be further simplified by removing the bias term b and the resultant learning algorithms are unified with ELM. The proposed equality constrained optimization method based ELM can be directly applied as an unified learning algorithm for regression, binary and multi-class classification applications.

In ELM, the parameter tuning process is much easier than SVM and LS-SVM/PSVM. If the feature mappings $\mathbf{h}(\mathbf{x})$ are explicitly defined and known to users, only one parameter C need to be specified in ELM . The generalization performance of ELM is not very sensitive to the dimensionality of the feature space (the number of hidden nodes L) as long as L is set large enough. Different from SVM, LS-SVM and PSVM which usually request two parameters (C, γ) to be tuned by users, single parameter setting makes ELM

Table 3.5: Performance comparison of SVM, LS-SVM, and ELM on binary class datasets.

Datasets	SVM			LSSVM			Extreme Learning Machine								
							Gaussian Kernel			Sigmoid Additive Node			Multiquadrics RBF Node		
	Testing		Training	Testing		Training	Testing		Training	Testing		Training	Testing		Training
	Rate (%)	Dev (%)	Time (s)	Rate (%)	Dev (%)	Time (s)	Rate (%)	Dev (%)	Time (s)	Rate (%)	Dev (%)	Time (s)	Rate (%)	Dev (%)	Time (s)
							$f(\mathbf{x}) = \text{sign}\left(\mathbf{h}(\mathbf{x})\mathbf{H}^T\left(\frac{\mathbf{I}}{C}+\mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{T}\right)$								
Diabete	76.97	2.70	0.6759	77.18	2.07	0.1406	**77.52**	2.46	0.0528	77.95	2.18	0.2075	78.09	2.17	0.2306
Australian	85.79	2.03	0.7042	85.91	1.85	0.1250	**86.29**	1.43	0.0403	86.18	1.80	0.1709	86.70	1.90	0.1691
Credit															
Liver	72.65	3.61	0.5616	71.98	3.37	0.0625	72.14	3.74	0.0066	73.01	3.78	0.0528	71.57	0.04	0.0531
Banana	89.84	0	0.8120	89.63	0	0.0620	89.83	0	0.0469	89.61	0.05	0.1350	89.30	0.01	0.1416
Colon	84.38	0	0.1617	81.25	0	0.4531	**84.38**	0	0.0031	81.63	3.32	0.1103	82.13	1.55	0.1472
Colon	84.38	0	0.0462	87.50	0	0.0469	**90.63**	0	0.0010	89.62	2.85	0.0075	87.50	0	0.0072
(Gene Sel)															
Leukemia	82.34	0	1.007	**85.29**	0	1.703	82.35	0	0.0309	80.08	3.85	0.4288	83.47	3.41	0.5550
Leukemia	100	0	0.0494	100	0	0.0625	100	0	0.0003	100	0	0.0075	100	0	0.0106
(Gene Sel)															
							$f(\mathbf{x}) = \text{sign}\left(\mathbf{h}(\mathbf{x})\left(\frac{\mathbf{I}}{C}+\mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T}\right)$								
Brightdata	99.46	0.23	1.289	**99.24**	0.17	0.5413	98.91	0.25	0.2984	99.31	0.2	0.7573	99.31	0.17	0.7401
Dimdata	95.85	0.27	0.8908	95.19	0.35	0.5781	**95.89**	0.34	0.2734	95.75	0.29	0.749	95.67	0.26	0.75
Mushroom	89.88	0.43	46.56	88.87	0.41	1.531	88.84	0.36	0.8133	88.91	0.36	1.038	88.88	0.33	1.047
* Adult	84.51	0	29.382	84.79	0	5.5703	84.58	0	3.3116	84.59	0.05	0.4246	84.51	0.02	0.5682

Table 3.6: Performance comparison of SVM, LS-SVM and ELM on multi-class datasets.

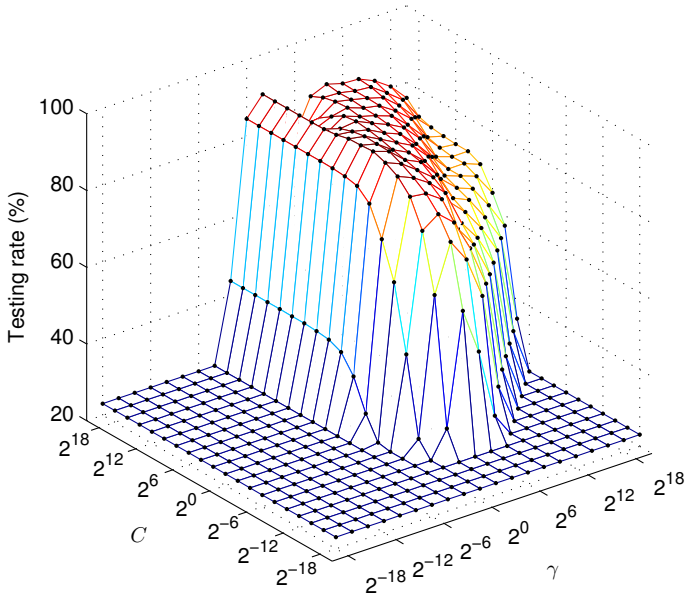
Datasets	SVM			LSSVM			Extreme Learning Machine								
							Gaussian Kernel			Sigmoid Additive Node			Multiquadrics RBF Node		
	Testing		Training	Testing		Training	Testing		Training	Testing		Training	Testing		Training
	Rate (%)	Dev (%)	Time (s)	Rate (%)	Dev (%)	Time (s)	Rate (%)	Dev (%)	Time (s)	Rate (%)	Dev (%)	Time (s)	Rate (%)	Dev (%)	Time (s)
										$\mathbf{f}(\mathbf{x}) = \mathbf{h}(\mathbf{x})\mathbf{H}^T\left(\frac{\mathbf{I}}{C} + \mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{T}$					
Iris	95.12	2.45	0.075	96.28	2.36	0.0021	96.04	2.37	0.0022	97.6	2.29	0.0156	97.33	2.12	0.0161
Glass	67.83	4.67	0.2871	67.22	5.04	0.0097	**68.41**	4.81	0.0026	67.12	4.99	0.0262	66.89	4.97	0.0264
Wine	98.37	1.41	0.075	97.63	1.82	0.0043	**98.48**	1.7	0.0019	98.47	1.81	0.0222	98.57	1.26	0.0206
Ecoli	86.56	3.65	0.2469	85.93	2.82	0.0244	**87.48**	2.8	0.008	87.23	2.88	0.053	87.79	2.74	0.054
Vowel	56.28	0	2.172	52.81	0	0.3290	**58.66**	0	0.0688	53.73	0.91	0.2187	54.75	0.89	0.2231
Vehicle	84.37	1.71	1.5144	83.19	1.93	0.2029	83.16	1.89	0.0831	83.48	1.78	0.2557	83.95	1.85	0.2547
										$\mathbf{f}(\mathbf{x}) = \mathbf{h}(\mathbf{x})\left(\frac{\mathbf{I}}{C} + \mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T}$					
Segment	96.53	0.64	14.30	96.12	0.75	4.302	**96.53**	0.54	0.9272	96.07	0.69	1.889	95.54	0.41	1.070
Satimage	89.75	0	698.4	90.05	0	82.67	**92.35**	0	15.70	89.8	0.32	2.808	89.06	0.38	2.803
DNA	92.86	0	7732	93.68	0	6.359	**96.29**	0	2.156	93.81	0.24	1.586	94.81	0.32	1.597
* Letter	96.72	0.23	163.75	97.28	0.22	308.9	**97.41**	0.13	41.89	93.51	0.15	0.7881	93.96	0.15	1.4339
*** shuttle**	*99.74*	*0*	*2864.0*	*99.82*	*0*	**24767.0**	*99.91*	*0*	*4029.0*	*99.64*	*0.01*	**3.3379**	*99.65*	*0.02*	**5.5455**
*** USPS**	*96.51*	*0*	**100.67**	*96.76*	*0*	*59.14*	**98.9**	*0*	*9.28*	*96.28*	*0.28*	**0.6877**	*97.25*	*0.24*	**0.9008**

Table 3.7: Performance comparison of SVM, LS-SVM and ELM on regression datasets.

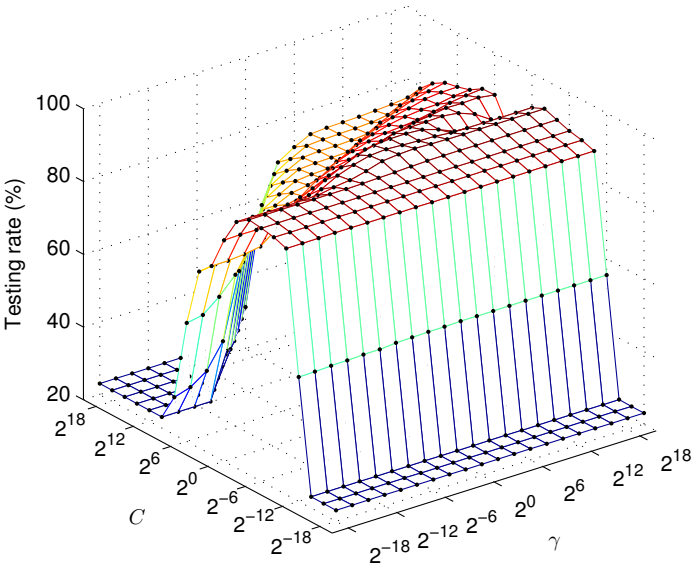
Datasets	SVR			LSSVR			Extreme Learning Machine								
							Gaussian Kernel			Sigmoid Additive Node			Multiquadrics RBF Node		
	Testing		Training	Testing		Training	Testing		Training	Testing		Training	Testing		Training
	RMSE	Dev	Time(s)	RMSE	Dev	Time(s)	RMSE	Dev	Time(s)	RMSE	Dev	Time(s)	RMSE	Dev	Time(s)
										$f(\mathbf{x}) = \mathbf{h}(\mathbf{x})\mathbf{H}^T\left(\frac{\mathbf{I}}{C} + \mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{T}$					
Baskball	0.162	0.0138	0.0160	**0.1564**	0.0165	0.0010	0.1617	0.0175	0.0005	0.1629	0.0169	0.0066	0.1630	0.0155	0.0094
Cloud	0.3262	0.0668	0.0151	0.3049	0.0203	0.0008	0.3061	0.0239	0.0013	0.3165	0.0178	0.0063	0.2969	0.0243	0.0130
Autoprice	0.1776	0.0179	0.0620	**0.1601**	0.0217	0.0031	0.1697	0.015	0.0020	0.1710	0.0165	0.0188	0.1678	0.0189	0.0247
Strike	0.2282	0.0078	1.578	0.2479	0.0106	0.0531	**0.2322**	0.0128	0.0358	0.2985	0.0053	0.0791	0.2646	0.0186	0.1651
Pyrim	0.128	0.0315	0.0160	0.1272	0.0388	0.0006	**0.0921**	0.0167	0.0003	0.1194	0.0311	0.0059	0.1486	0.0369	0.0083
Bodyfat	0.0279	0.0081	0.0470	0.0280	0.0129	0.0059	0.0242	0.0131	0.0028	0.0286	0.0083	0.0231	0.0262	0.0113	0.0354
Cleveland	0.1646	0.0067	0.0930	0.1605	0.0071	0.0075	0.1618	0.0088	0.0053	0.1603	0.0070	0.0284	0.1609	0.0093	0.0474
Housing	0.0976	0.0085	0.2040	0.0704	0.0068	0.0343	0.0744	0.0106	0.0197	0.0805	0.0077	0.0616	0.0779	0.0084	0.1005
										$f(\mathbf{x}) = \mathbf{h}(\mathbf{x})\left(\frac{\mathbf{I}}{C} + \mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T}$					
Balloon	0.059	0.0034	11.30	0.0536	0.0044	1.072	0.0516	0.0043	0.6084	0.0553	0.0013	0.7688	0.0588	0.0121	1.344
Quake	0.1797	0.0068	19.59	**0.1446**	0.0079	1.387	0.1712	0.0099	0.6972	0.1649	0.0061	0.7625	0.1696	0.0098	1.607
Space-ga	0.0648	0.0016	52.75	0.0330	0.0008	3.510	0.0338	0.0018	1.770	0.0624	0.0021	1.220	0.0335	0.0013	1.369
Abalone	0.0764	0.0015	113.1	0.0746	0.0021	7.674	0.0768	0.0021	4.112	0.0761	0.0019	1.324	0.0761	0.0023	1.793

be used easily and efficiently.

If feature mappings are unknown to users, kernels can be used in ELM's solution. However, since LS-SVM and ELM have the same optimization objective functions and ELM has less constraints on the bias term b, LS-SVM tends to obtain a suboptimal solution to ELM's. As verified by the simulation results, ELM obtains similar or better generalization performance for regression and binary class classification cases than SVM and LS-SVM, and much better generalization performance for multi-class classification cases. ELM also shows better scalability and runs at much faster learning speed than traditional SVM and LS-SVM.

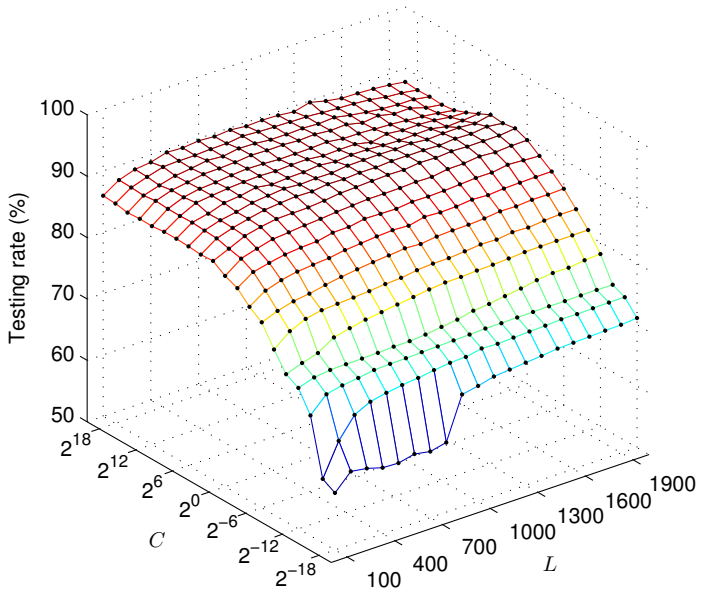


(a) LS-SVM with Gaussian Kernel

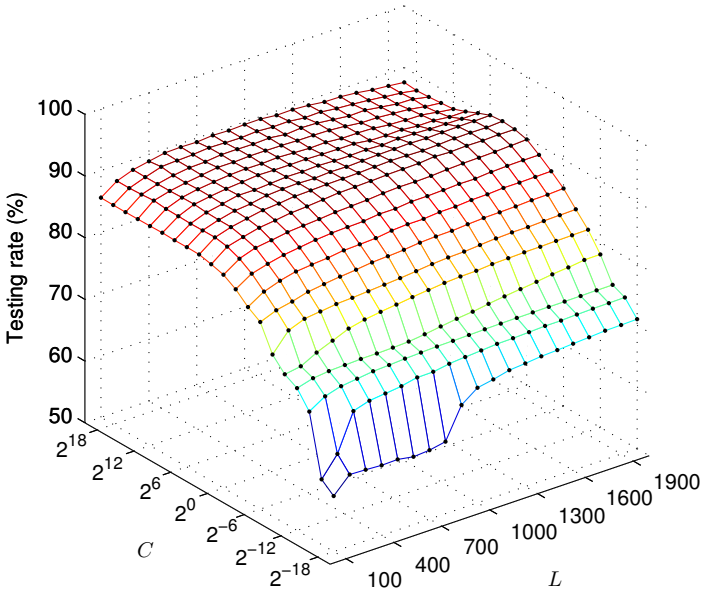


(b) ELM with Gaussian Kernel

Figure 3.2: Parameter analysis on Satimage dataset: the testing rate of LS-SVM and ELM with Gaussian Kernel are sensitive to the user specified parameters (C, γ).

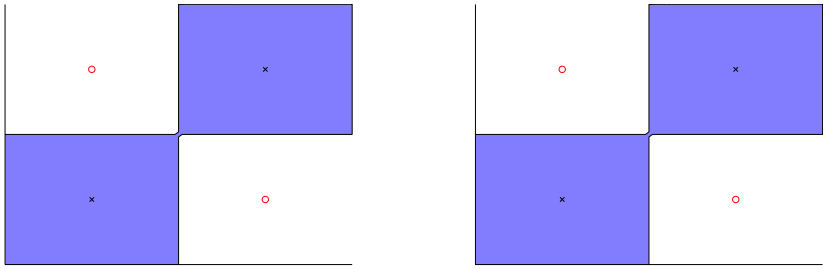


(a) ELM with Sigmoid additive node

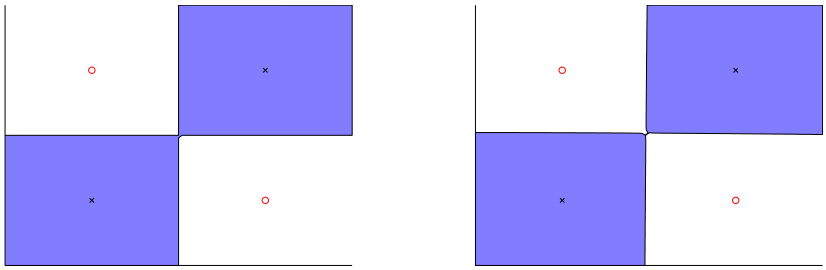


(b) ELM with Multiquadrics RBF node

Figure 3.3: Parameter analysis on Satimage dataset: the testing rate of ELM (with Sigmoid additive node and Multiquadrics RBF node) are not very sensitive to the user specified parameters (C, L) and good accuracies can be achieved as long as L is large enough.

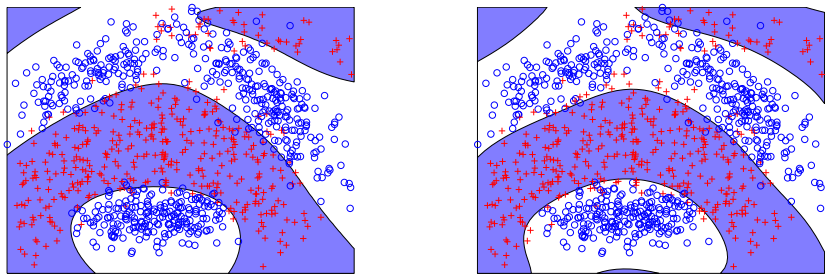


(a) SVM (b) LS-SVM

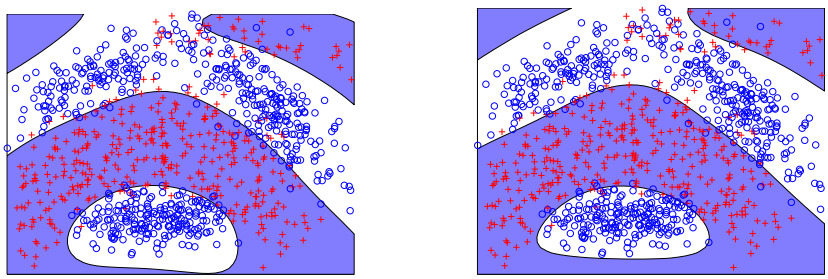


(c) ELM (Gaussian Kernel) (d) ELM (Sigmoid additive node)

Figure 3.4: The separating boundaries of different classifiers in XOR problem.



(a) SVM (b) LS-SVM



(c) ELM (Gaussian Kernel) (d) ELM (Sigmoid additive node)

Figure 3.5: The separating boundaries of different classifiers in Banana case.

Chapter 4

Credit Risk Evaluation with Kernelized Extreme Learning Machine

Since 2008 the financial crisis, credit risk evaluation has become an increasingly important field in financial risk management for financial institutions, especially for banks and credit card companies. It is much more desirable for a reliable evaluation model. Many data mining and statistical methods have been applied to this field. Extreme learning machine (ELM) classifier as a type of generalized single hidden layer feed-forward networks has been used in many applications and achieve good classification accuracy. Thus in this chapter, we use ELM (kernel based) as a classification tool to perform the credit risk evaluation. The simulations are done on two credit risk evaluation datasets with three different kernel functions. Simulation results show that the kernel based ELM is more suitable for credit risk evaluation than the popular used Support Vector Machines (SVMs) with consideration of overall, good and bad accuracy.

4.1 Background

Credit risk evaluation is an important field in financial risk management. It helps many credit-granting institution (i.e. commercial banks and certain retailers) to discriminate good customers from bad ones, which is crucial for them to develop lending strategies in order to optimize profit. In 1930s, the numerical score cards were first introduced by mail-order companies. After then, it is very common for financial institutions to build up credit risk evaluation models using data mining and statistical methods.

Many classification techniques have been applied to develop accurate and reliable credit evaluation models. There are many statistical methods that have been developed including discriminant analysis [71], logistic regression [72], and probit regression [73], and etc. In addition, there are also artificial intelligence (AI) techniques implemented including neural networks [74, 75], genetic algorithm (GA) [76, 77], case-based reasoning [78], support vector machines (SVM) [79, 80, 81], and etc. Among all the credit evaluation methods, SVM can achieve much better performance compared to other classification methods [82, 83].

Recently, Huang et al proposed the kernel based ELM, which is derived when solving ELM's optimization problem as an equality constrained optimization problem [84]. In kernel based ELM, the hidden layer feature mapping is unknown and it is not necessary to determine the network structure. It has been shown that ELM and its variants could provide good generalization performance for many regression and classification applications and they may run much faster than SVM classifiers [7]. Hence, we would like to apply kernel based ELM on credit risk evaluation and compare the results of kernel based ELM with SVM classifiers.

4.2 Datasets and Evaluation Methods

Credit risk evaluation can be seen as a two-class classification problem that is used to distinguish the "bad" creditors from the "good" creditors. In this work, we apply kernel based ELM and SVM for credit risk evaluation with different kernel functions.

4.2.1 Datasets

Two real world credit datasets are used in the evaluation: Australian credit dataset and German credit dataset, which are collected from UCI Machine Learning Repository [62]. The specification of the datasets is given in Table 4.1.

Table 4.1: Specification of Benchmark Datasets

Datasets	# Attributes	# Classes	# Cases	# Positive cases	# Negative cases
Australian credit	14	2	690	307	383
German credit	24	2	1000	700	300

The Australian credit dataset contains 690 instances of MasterCards applications. Among all the applications, 307 instances are classified as positive (i.e, "Good" applications), and 383 instances are classified as negative (i.e, "Bad" applications). Each instance has 14 attributes for Australian credit case.

The German credit dataset include 1000 instances, 700 instances of which are classified as credit-worthy cases and 300 instances are the cases that the credit should not be extended. Each instance has 24 attributes for German credit case.

4.2.2 Evaluation

In our experiments, p samples (p=40, 60, 80, 100, 120, 140, 160, 180) are randomly selected from both positive class and negative class to train kernel based ELM classifier and SVM classifier. In other words, the number of samples in the training dataset is $2p$ (i.e. $2p$=80, 120, 160, 200, 240, 280, 320, 360). For each case, the remaining samples are used to validate the performance of the classifiers.

Since kernel method is implemented in both kernel based ELM and SVM, we are curious about how different kernel functions will affect the performance of the classifiers. In this work, we have conducted the comparison between kernel based ELM and SVM with Gaussian kernel, polynomial kernel and hyperbolic tangent kernel. The mathematical expressions of the kernel functions are shown below:

Gaussian kernel

$$K(\mathbf{u},\mathbf{v}) = exp(-\gamma\|\mathbf{u}-\mathbf{v}\|^2) \tag{4.1}$$

Polynomial kernel

$$K(\mathbf{u},\mathbf{v}) = (\mathbf{u}\cdot\mathbf{v}+1)^{power} \tag{4.2}$$

Hyperbolic tangent kernel

$$K(\mathbf{u},\mathbf{v}) = \tanh(hyper_K\mathbf{u}\cdot\mathbf{v}-2) \tag{4.3}$$

For more precise evaluation, we evaluated the kernel based ELM and SVM classifiers based on the confusion matrix presented in Table 4.2. Three types of accuracies were listed to compare kernel based ELM and SVM classifiers. They are overall accuracy, good accuracy (i.e. or called recall) and bad accuracy (i.e. or called true negative rate). Overall accuracy is obtained by using the number of correct classification in test set over

Table 4.2: Confusion Matrix in Classification

Predicted Class	True Class	
	Good	Bad
Good	True positive(TP)	False positive(FP)
Bad	False negative(FN)	True negative (TN)
Overall Accuracy: Acc=(TP+TN)/(TP+FP+FN+TN)		
Good Accuracy: Good=TP/(TP+FN)		
Bad Accuracy: Bad=TN/(FP+TN)		

the number of samples in the test set. Good accuracy is calculated by using the number of correctly classified good samples in the test set over the number of good samples in the test set. Bad accuracy is computed by using the number of correctly classified bad samples in the test set over the number of bad samples in the test set. In addition to the overall accuracy, good and bad accuracies are significant criteria to examine the classifiers. In the real world, the capability of a credit risk model to distinguish the "bad" clients from the "good" clients is essential for the financial institutions. Hence, it is desired to improve the bad accuracy while maintaining the good accuracy at an acceptable standard.

4.3 Experiments and Results

The performance of kernel based ELM and SVM classifiers has been evaluated on two credit risk datasets: Australian credit dataset and German credit dataset. For SVM, the *SVM and Kernel Methods MATLAB Toolbox* [85] is used for the tasks. All the evaluations were carried out in the Matlab 7.4.0 environment running on a desktop with CPU 2.66GHz and 3GB RAM. For all the credit applications, the input data have been normalized into $[-1, 1]$. All the results presented are based on the average of 50 trials of random sampling for the training and testing sets.

4.3.1 Parameter selection

For both kernel based ELM and SVM classifiers, the generalization performance usually depends closed on the setting of the parameters. Especially for SVM classifiers, the combinations of cost parameter C and kernel parameter have to be chosen carefully to get the best results. In our simulations, for both of the credit risk cases, we have conducted parameter selection based on the overall testing accuracy.

Besides the parameter associated with the kernel function, there is one parameter for both of the kernel based ELM and SVM classifier, the cost parameter λ (i.e. for kernel based ELM) or C (i.e. for SVM). For each specific training dataset, we search the optimal cost parameter from 15 different values: λ or $C \in \{0.0001, 0.001, 0.01, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100, 1000, 10000\}$. The method of determining the parameters associated with the kernel functions is shown below.

Gaussian kernel

The parameter γ is selected from 15 different values: $\gamma \in \{0.0001, 0.001, 0.01, 0.1, 0.2, 0.4, 0.8, 1, 2, 5, 10, 20, 100, 1000, 10000\}$.

Polynomial kernel

The parameter *power* is selected from 15 different values: $power \in \{2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$.

Hyperbolic tangent kernel

The parameter $hyper_K$ is selected from 15 different values: $hyper_K \in \{0.0001, 0.001, 0.01, 0.1, 0.2, 0.4, 0.8, 1, 2, 5, 10, 20, 100, 1000, 10000\}$.

The optimal parameters selected for Australian credit case is presented in Table 4.3. While the optimal parameters selected for German credit case is shown in Table 4.4.

Table 4.3: Optimal Parameters of Australian Credit case

Training data	Gaussian				Polynomial				Hyperbolic			
	ELM		SVM		ELM		SVM		ELM		SVM	
	γ	λ	γ	C	$power$	λ	$power$	C	$hyper_K$	λ	$hyper_K$	C
80	100	10000	2	1000	2	0.5	3	0.05	0.1	10000	0.1	1000
120	100	10000	2	50	2	1	2	10	0.1	1000	0.1	1000
160	100	10000	2	10000	2	1	3	50	0.1	1000	0.1	100
200	100	10000	2	10000	2	1	3	10000	0.1	1000	0.1	50
240	20	50	2	20	2	1	2	0.2	0.1	1000	0.1	100
280	100	1000	2	50	2	1	2	0.2	0.1	1000	0.1	1000
320	20	100	2	20	2	2	2	0.2	0.1	1000	0.1	100
360	20	100	2	20	2	0.2	3	0.01	0.1	1000	0.1	100

Table 4.4: Optimal Parameters of German Credit case

Training data	Gaussian				Polynomial				Hyperbolic			
	ELM		SVM		ELM		SVM		ELM		SVM	
	γ	λ	γ	C	$power$	λ	$power$	C	$hyper_K$	λ	$hyper_K$	C
80	1000	0.1	10	10	2	0.0001	2	0.01	0.2	0.01	0.01	100
120	10000	0.0001	20	100	2	0.00001	2	0.01	0.2	0.01	0.01	0.001
160	10	0.00001	10	20	2	0.00001	2	0.01	0.2	0.01	0.01	0.001
200	1000	0.00001	10	100	2	0.00001	2	0.01	0.2	0.001	0.01	0.01
240	100	0.00001	20	1000	2	0.00001	2	0.01	0.1	0.001	0.01	0.001
280	10	0.00001	20	1000	2	0.00001	2	0.01	0.1	0.001	0.01	0.001
320	100	0.0001	10	20	2	0.00001	2	0.01	0.2	0.001	0.01	0.001
360	1000	0.00001	20	100	2	0.00001	2	0.01	0.2	0.001	0.01	0.001

4.3.2 Results on Australian credit case

The simulation results on Australian credit case are presented in Table 4.5 (Gaussian kernel), Table 4.6 (Polynomial kernel) and Table 4.7 (Hyperbolic tangent kernel), respectively.

We have highlighted the results when the performance of one classifier is much better than the other. For Australian credit dataset, from Table 4.5 to Table 4.7, we can observe that kernel based ELM outperforms SVM for the overall accuracy with all the three different kernels. The kernel based ELM can achieve better testing rate for both "good" and "bad" accuracy than SVM in majority cases as well. This shows that ELM can distinguish better the high credit rating individuals from low credit rating individuals. Simulation results also shows that the testing accuracy does not vary too much when

different kernel functions are used for Australian credit case. In general, SVM's result is either much worse than ELM's or similar to ELM's in a few cases. Thus, it is more reliable to apply the kernel based ELM for the Australian credit evaluation than SVM.

Table 4.5: Gaussian Kernel on Australian Credit case

Training data	Overall Accuracy		Good		Bad	
	ELM	SVM	ELM	SVM	ELM	SVM
80	**0.9253**	0.9068	**0.9466**	0.9045	0.9070	0.9082
120	**0.9505**	0.9283	**0.9528**	0.9300	**0.9454**	0.9279
160	**0.9566**	0.9378	**0.9667**	0.9359	**0.9508**	0.9442
200	**0.9594**	0.9442	**0.9657**	0.9411	**0.9516**	0.9496
240	**0.9612**	0.9542	**0.9628**	0.9543	0.9548	0.9530
280	**0.9641**	0.9572	**0.9725**	0.9509	0.9539	0.9547
320	**0.9651**	0.9585	**0.9660**	0.9524	0.9592	0.9623
360	0.9645	0.9653	0.9681	0.9669	0.9571	**0.9638**

Table 4.6: Polynomial Kernel on Australian Credit case

Training data	Overall Accuracy		Good		Bad	
	ELM	SVM	ELM	SVM	ELM	SVM
80	**0.9370**	0.9058	0.9279	0.9255	0.9249	0.8921
120	**0.9534**	0.9322	**0.9573**	0.9283	**0.9491**	0.9229
160	**0.9568**	0.9442	**0.9637**	0.9469	**0.9553**	0.9259
200	**0.9623**	0.9464	**0.9647**	0.9478	**0.9527**	0.9390
240	**0.9621**	0.9524	**0.9626**	0.9508	**0.9646**	0.9521
280	**0.9667**	0.9548	**0.9713**	0.9575	0.9566	**0.9611**
320	**0.9658**	0.9603	**0.9718**	0.9599	0.9601	0.9585
360	**0.9667**	0.9602	**0.9748**	0.9642	**0.9611**	0.9549

4.3.3 Results on German credit case

The simulation results on German credit case are presented in Table 4.8 (Gaussian kernel), Table 4.9 (Polynomial kernel) and Table 4.10 (Hyperbolic tangent kernel), respectively.

We have highlighted the values when the performance of one classifier is much better

Table 4.7: Hyperbolic Kernel on Australian Credit case

Training data	Overall Accuracy		Good		Bad	
	ELM	SVM	ELM	SVM	ELM	SVM
80	**0.9220**	0.8918	**0.9307**	0.9236	**0.8961**	0.8821
120	**0.9439**	0.9211	**0.9538**	0.9403	**0.9345**	0.9139
160	**0.9486**	0.9359	0.9570	0.9515	**0.9487**	0.9172
200	**0.9572**	0.9407	**0.9645**	0.9442	**0.9468**	0.9281
240	**0.9608**	0.9470	0.9583	0.9602	**0.9542**	0.9435
280	**0.9587**	0.9490	**0.9668**	0.9542	0.9494	0.9508
320	**0.9618**	0.9541	**0.9680**	0.9565	0.9576	0.9578
360	0.9633	0.9632	0.9650	0.9598	0.9601	0.9631

than the other. In German credit case, ELM with gaussian kernel function shows better overall and ”good” accuracy in most of the cases, while SVM with gaussian kernel function achieves better ”bad” accuracy. ELM with polynomial kernel shows much better overall and ”good” testing accuracy than SVM with polynomial kernel. Again, SVM with polynomial kernel has better ”bad” accuracy. Both ELM and SVM achieves their best overall accuracy when employing hyperbolic tengent kernel. SVM with hyperbolic tengent kernel shows better overall and ”good” accuracy. However, the ”bad” accuracy corresponded to such high overall accuracy is as low as 0. Though SVM with hyperbolic tangent kernel can achieve the best overall accuracy, the ”bad” accuracy is too low to be accepted. Hence, ELM classifier is more suitable and reliable to be used for credit risk evaluation in this case.

4.4 Conclusion

In this work, we compared the ELM classifier (kernel based) with SVM classifier using different kernel functions for credit risk evaluation on two credit datasets. The simulation results showed that the kernel based ELM has more accurate and reliable classification results on both of the credit risk applications. For Australian credit case, the performance of both classifiers did not vary too much when using different kernel

Table 4.8: Gaussian Kernel on German Credit case

Training data	Overall Accuracy		Good		Bad	
	ELM	SVM	ELM	SVM	ELM	SVM
80	0.6791	0.6835	0.6758	0.6796	0.6805	**0.6908**
120	0.6935	0.6965	**0.6861**	0.6701	0.6838	**0.7181**
160	**0.6989**	0.6933	0.6891	0.6872	0.6868	**0.6971**
200	0.7024	0.6985	**0.7048**	0.6841	0.6755	**0.7162**
240	**0.7071**	0.6986	**0.7012**	0.6944	0.6737	**0.7053**
280	0.7065	0.7047	0.7034	0.7022	0.7039	**0.7138**
320	**0.7137**	0.7043	**0.7063**	0.6866	0.6771	**0.7236**
360	**0.7188**	0.7125	**0.7165**	0.7033	0.6891	**0.7333**

Table 4.9: Polynomial Kernel on German Credit case

Training data	Overall Accuracy		Good		Bad	
	ELM	SVM	ELM	SVM	ELM	SVM
80	**0.6856**	0.6564	**0.7055**	0.6552	0.5987	**0.6645**
120	**0.6974**	0.6615	**0.7191**	0.6721	0.5833	**0.6833**
160	**0.7054**	0.6698	**0.7274**	0.6703	0.6101	**0.6992**
200	**0.7121**	0.6791	**0.7397**	0.6714	0.6218	**0.6962**
240	**0.7134**	0.6837	**0.7307**	0.6738	0.6771	**0.6976**
280	**0.7217**	0.6830	**0.7303**	0.6751	0.6461	**0.7121**
320	**0.7122**	0.6857	**0.7261**	0.6837	0.6716	**0.7143**
360	**0.7231**	0.6896	**0.7375**	0.6842	0.6560	**0.7172**

functions. For German credit case, though the SVM classifiers with hyperbolic tangent kernel showed best overall accuracy, it could not provide acceptable results on the ”bad” accuracy, hence compared to Kernel based ELM, SVM is not suitable to be used in this application.

Table 4.10: Hyperbolic Kernel on German Credit case

Training data	Overall Accuracy		Good		Bad	
	ELM	SVM	ELM	SVM	ELM	SVM
80	0.7006	**0.7174**	0.8027	**1**	**0.4803**	0
120	0.7188	**0.7273**	0.7398	**1**	**0.5473**	0
160	0.7196	**0.7381**	0.7827	**1**	**0.3957**	0
200	0.7302	**0.7501**	0.7628	**1**	**0.6418**	0
240	0.7332	**0.7632**	0.7509	**1**	**0.6108**	0
280	0.7354	**0.7778**	0.7458	**1**	**0.6138**	0
320	0.7402	**0.7941**	0.7651	**1**	**0.5954**	0
360	0.7532	**0.8125**	0.7809	**1**	**0.6175**	0

Chapter 5

Stacked Extreme Learning Machines

In the previous chapters, the equality constrained ELM was proposed as an unified learning platform for solving classification and regression problems. It provides different solutions suitable for different applications and works well in the majority of the cases. When facing super large datasets, in order to have a good generalization performance, ELM requires a large number of hidden nodes to map the data to high dimensional space. However, due to the physical memory limitation of most computing machines, the number of hidden nodes to be used is limited. Thus ELM usually cannot fully achieve its generalization capability. In this chapter, we proposed a Stacked ELMs learning network (S-ELMs) that can divide a single large ELM network into small ones on many different layers and use eigenvalues of the output weight matrix of ELM on each layer to determine the most significant nodes. Those nodes were combined together with the random nodes in the next layer and functioned as the hidden nodes of the next layer ELM. By creating such multiple layers structure, ELM could potentially solve datasets of any size and fully achieve its generalization capability. The simulation results on MNIST dataset showed that the S-ELMs even with random hidden nodes could achieve similar testing accuracy to SVM with a faster training speed while having low memory requirement.

5.1 Introduction

In Chapter 3, the equality constrained optimization method based Extreme Learning Machine was proposed. It provides two solutions for different size of training data. The user can choose to use *training data size based* (complexity of computation is based on the size of training data) for the small to medium size training data, or *hidden nodes size based* (complexity of computation is based on the number of hidden nodes). The training data is usually mapped from original input space to a higher dimensional space, within which the tackled problem can be solved. The number of hidden nodes determines the dimensionality of the mapped space. In general, to solve a classification problem, higher number of training samples requires more hidden nodes so that they can be separated in the ELM mapping space. This brings problems to ELM when facing extremely large training datasets. The large number of required hidden nodes makes ELM network very large and the computational task becomes very costly.

To tackle very large data problems, currently there are several approaches. One obvious solution is to use super computers, e.g. the Cray Titan super computer at Oak Ridge National Laboratory, with 560K cores, speed at 17.59 PFLOP/s and 710K GB memory [86]. Recently, with the introduction of NVidia CUDA parallel computing platform [87] in 2007, GPU computing [88] becomes popular to accelerate and solve larger scale problems. However, these solutions require extra hardware cost, hence may not be available to most researchers. There are also many existing approaches working on computational algorithms to reduce the computational complexity and memory requirement, e.g. approximation of training data samples to reduce the training data size such as Reduced Support Vector Machine [89], sequential learning algorithms such as Resource Allocation Network [90] and Online Sequential Extreme Learning Machine [91].

To solve large data problems using ELM without incurring a memory problem, one should keep the network size small yet be able to achieve good generalization accuracy. From the literature [32, 24, 25, 23], it seems multiple-hidden-layer feedforward networks have more powerful learning capability than single-hidden-layer feedforward

networks. Thus this work proposes a Stacked ELMs (S-ELMs) algorithm that tries to break one large ELM network into multiple sub-ELMs to complete the targeted machine learning task. The multiple sub-ELMs are stacked on multiple layers that are serially connected. From structural point of view, the proposed S-ELMs looks similar to multiple-hidden-layer feedforward networks. However it is still based on single-hidden-layer feedforward network. In multiple-hidden-layer feedforward networks, the multiple hidden layers are used to study higher feature representation of data. In our approach, the multiple layers of ELMs are used to function as different parts of hidden layer outputs of a large ELM and eventually simulate a large single-hidden-layer ELM.

As the hidden node parameters of ELM are randomly generated, the importance of different nodes that contribute to the target output may vary a lot. We can choose the most significant few percent of nodes (or modified nodes, such as a few nodes combined together as one node with some nonlinear relationship) to represent all the nodes for a single ELM in each layer. The lower layer can output such nodes to the ELM network in the upper layer. Those nodes are then combined with the new random nodes and function as the total hidden layer ouput of upper layer ELM. In the upper layer, same procedure as the previous layer can be taken to output the most significant nodes to the higher layer. In this manner, we can keep the whole network size fixed yet be able to learn the data in a higher dimensional ELM feature space. Suppose that the total number of hidden nodes in each layer is L, and L' nodes are retained for the next layer, and if S layers are used, the Stacked ELMs learning network can theoretically simulate nearly equivalent generalization capability to the single ELM network with $((S-1)\times(L-L')+L)$ number of hidden nodes. In this work, we choose the top few significant nodes using their output weights' eigenvalues, and reduce the nodes by multiplying the corresponding eigenvectors. This idea is also aligned with the concept of PCA dimension reduction [92]. We can treat the output weights as training data whose dimension needs to be reduced to a certain number, so the PCA dimension reduction method can be applied to the output weights. The principal components are now the output weights of the selected significant nodes which are computed from the basis vectors. The simulation result on MNIST

dataset shows that the Stacked ELMs can achieve very close testing accuracy to SVM's result even with the random hidden nodes. The process to find out the best possible accuracy is simpler than SVM. Most importantly, the S-ELMs is capable of solving very large data problems without incurring out of memory problem.

5.2 Related Work: PCA Dimension Reduction

Principal Component Analysis (PCA) [92] is a widely used statistical technique for unsupervised dimension reduction. It converts a set of possibly correlated variables into a set of linearly uncorrelated variables called principal components. The greatest variance by any projection of the data set comes to lie on the first axis (first principal component), and the second greatest variance on the second axis and so on. The number of chosen principal components may be much less than the number of original set of variables. PCA can be done by using eigenvalue decomposition of a data covariance matrix or singular value decomposition of a data matrix. For a given input matrix $\mathbf{A} \in R^{m\times n}$, the goal of PCA is to find a matrix $\mathbf{Y} \in R^{m\times n'}$, where $\mathbf{Y}$ is the Karhunen-Loève transform (KLT) of matrix $\mathbf{A}$: $\mathbf{Y} = \mathbb{KLT}\{\mathbf{A}\}$. Followings are the major steps taken when using covariance method [93]:

i) Normalize the matrix $\mathbf{A}$ along its columns by subtracting the mean of each column to get the mean-subtracted matrix $\mathbf{B}$, where each column of $\mathbf{B}$ satisfies:

$$\mathbf{B}_i = \mathbf{A}_i - \frac{1}{m}\sum_{j=1}^{m} \mathrm{A}_{i,j},\ i\ = 1,\cdots,n \tag{5.1}$$

ii) Find the covariance matrix: $\mathbf{C} = cov(\mathbf{B})$.

iii) Find the eigenvectors $\mathbf{V}$ and eigenvalues $\mathbf{D}$ of the covariance matrix $\mathbf{C}$ from:

$$\mathbf{V}^{-1}\mathbf{C}\mathbf{V} = \mathbf{D} \tag{5.2}$$

where **D** is the diagonal matrix of eigenvalues of **C**.

iv) Sort the eigenvalues **D** in descending order as well as matching the corresponding eigenvectors. The reordered eigenvectors are now $\tilde{\mathbf{V}}$.

v) Choose the top $n' \leq n$ significant principal components according to their eigenvalues. This can be done by multiplying the top n' rows of $\tilde{\mathbf{V}}$ to the normalized matrix of **A**. We then can obtain:

$$\mathbf{Y} = \mathbf{B}\tilde{\mathbf{V}}(1:n') = \mathbb{KLT}\{\mathbf{A}\} \tag{5.3}$$

5.3 Proposed Stacked Extreme Learning Machines

5.3.1 Problem with existing ELM solution for very large dataset

The equality constrained optimization method based ELM provides two solutions for datasets of different sizes. The solution for the very large dataset is given in equation (3.16). In [31], for a large dataset, the number of hidden nodes is chosen as 1000. The generalization performance is fairly good compared with SVM and Least Square SVM. However, this number may not be the optimal number for the best generalization performance of ELM. In theory, the larger the dataset is, the more nodes ELM requires to map the data to higher dimensional space for better generalization performance. When solving a very large dataset, e.g. the MNIST database of handwritten digits [95], we found that the testing accuracy increases when the number of hidden nodes increases beyond 1000. Due to the physical memory limitation, we can only test up to 5000 hidden nodes for the MNIST dataset. The following figure shows the relationship between testing accuracy and number of hidden nodes:

Figure 5.1 shows that the testing accuracy increases when increasing the number of

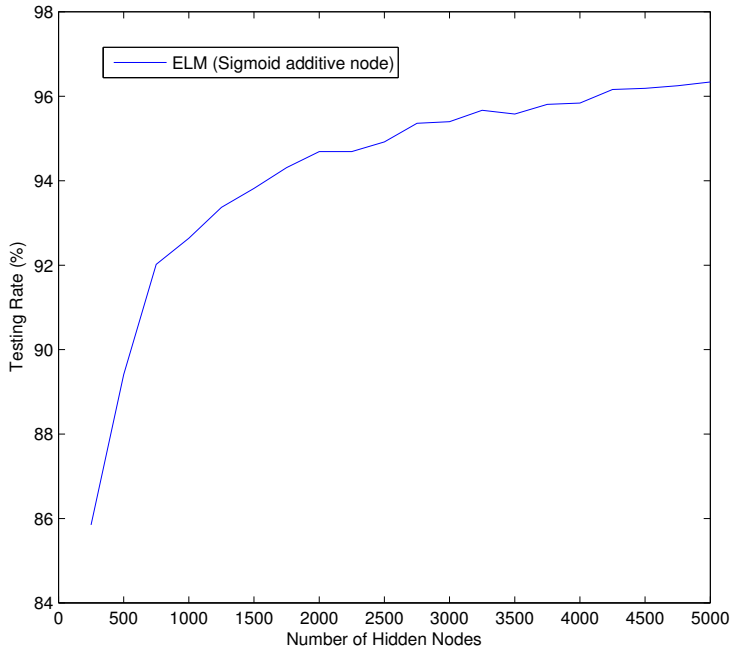


Figure 5.1: Relationship between the testing accuracy and the number of hidden nodes for MNIST dataset using the equality optimization method based ELM's solution for very large dataset

hidden nodes. It is reasonable to believe that ELM can achieve a better accuracy with more hidden nodes for this dataset. However, due to the physical memory limitation, it is very hard to test more nodes to find out the optimal number of hidden nodes for the best performance. In this work, we propose a Stacked ELMs that can help us solve this problem. Instead of having one huge ELM that requires many hidden nodes, we break it into multiple small ELMs, each stacking on top of another and forming a Stacked ELMs learning network.

5.3.2 Stacked ELMs

The Stacked ELMs learning network (S-ELMs) consists of multiple ELMs located in different layers. In each single layer of S-ELMs, the "fat" ELM is reduced to a "slim" ELM with a smaller number of hidden nodes as shown in figure 5.2. The lower layer

ELM (after being reduced to the "slim" ELM) will propagate its hidden nodes output to the upper layer ELM's hidden layer, and functions as a part of the hidden nodes in the upper layer ELM. After combining the hidden nodes output from the previous layer's ELM and the newly randomly generated hidden nodes, this layer's ELM will be reduced to a "slim" ELM and propagate the hidden nodes output to the upper layer. As shown in figure 5.3, this process will carry on until the last layer, where the hidden nodes output from the second last layer ELM contains almost all hidden nodes information from all the lower layers. In the last layer, the output of the newly generated hidden nodes will be combined together with output from the second last layer's hidden nodes, and form the ELM feature mapping matrix for calculating the final output function. The basic algorithmic structure of the proposed S-ELMs is shown in figure 5.4.

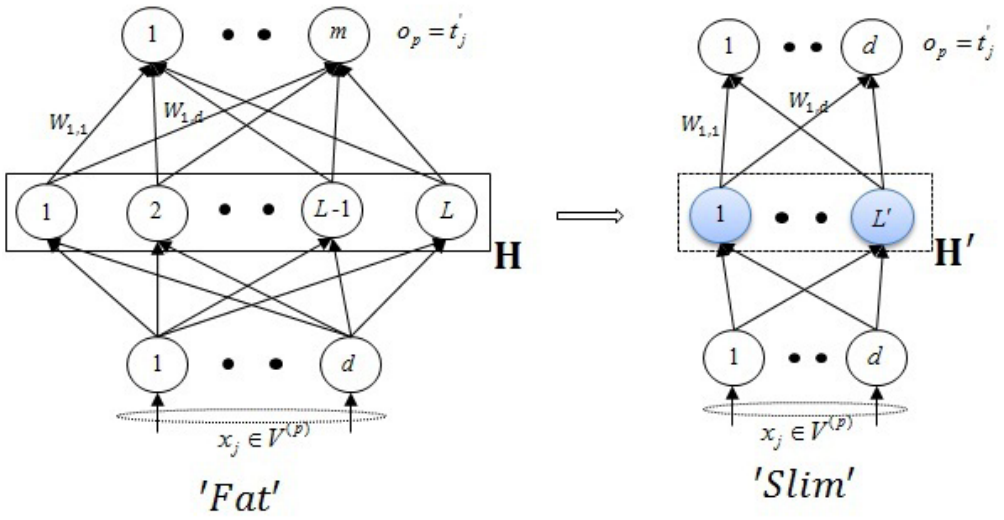


Figure 5.2: The "fat" ELM is reduced to a "slim" ELM using PCA dimension reduction method

In the bottom layer of S-ELMs, it is a simple ELM network with L hidden nodes. The hidden layer output can be recorded as $\mathbf{H}_1$, and the output weight of the bottom layer ELM β_1 can be obtained as:

$$\beta_1 = \left(\frac{\mathbf{I}}{C} + \mathbf{H}_1{}^T\mathbf{H}_1\right)^{-1} \mathbf{H}_1{}^T\mathbf{T} \tag{5.4}$$

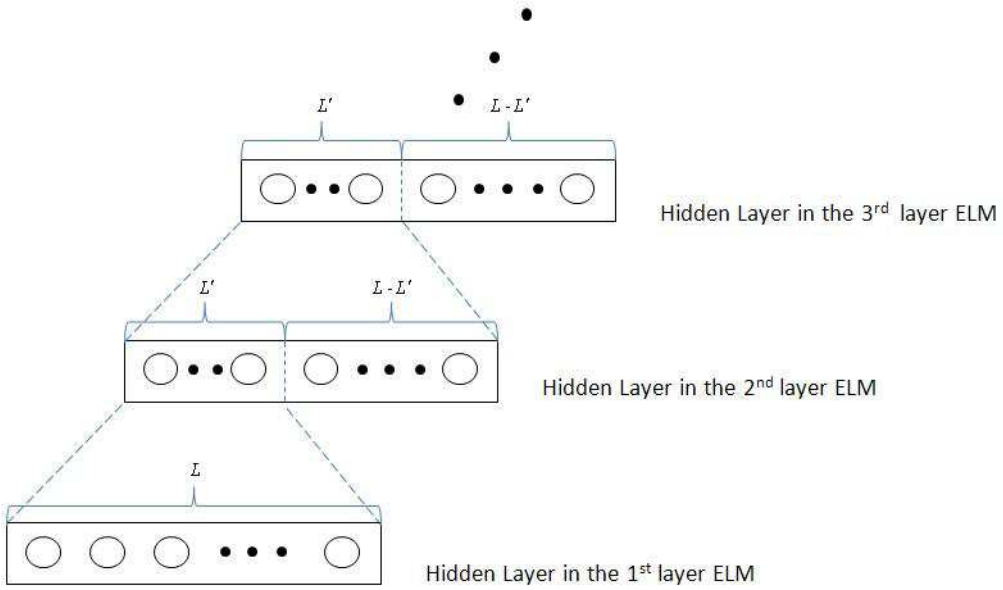


Figure 5.3: The lower layer ELM propagates its hidden nodes output to the upper layer ELM and functions as a part of hidden nodes in the upper layer ELM

Where β_1 represents the weights of all connections between the hidden nodes and the output nodes. Some of these connections may not be linearly independent, so we can combine them to reduce the number of the hidden nodes required. Also some of the hidden nodes are less important than other nodes in contributing to the output function, thus we can also remove these nodes to reduce the computational cost. This is the same task as reducing the dimensionality of β_1^T without losing much information. Here, PCA dimension reduction method is applied on β_1^T to reduce the dimension of β_1^T from L to L', where L' can be a user specified value, with $L' \leq L$. In the PCA dimension reduction process, there will be a set of eigenvectors generated based on the orders of their corresponding eigenvalues. The top L' number of eigenvectors are recorded as $\tilde{\mathbf{V}} \in \mathbf{R}^{L \times L'}$. The reduced output weight is then obtained as: ${\beta_1}' = {\beta_1}^T \tilde{\mathbf{V}}$. The original L random hidden nodes can now be replaced by L' combined significant nodes, and the reduced hidden layer output ${\mathbf{H}_1}'$ becomes:

$$ {\mathbf{H}_1}' = \mathbf{H}_1 \tilde{\mathbf{V}} \tag{5.5} $$

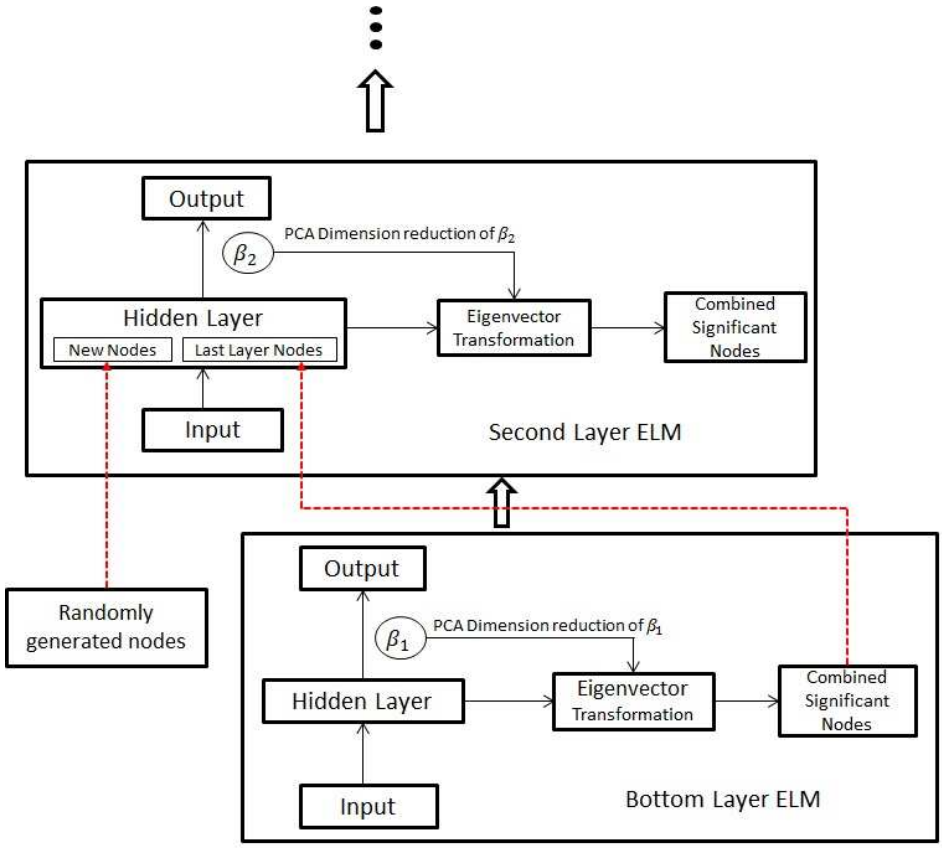


Figure 5.4: Block diagram showing the information flow of the first two layer of the Stacked ELMs learning network

The reduced hidden layer output $\mathbf{H}_1{}'$ is propagated to second layer to represent the total hidden nodes information of the first layer's ELM. In the second layer, there will be $(L-L')$ new random hidden nodes generated, and the output of these random hidden nodes $\mathbf{H}_{2new}$ is:

$$\mathbf{H}_{2new} = \begin{bmatrix} \mathbf{h}_{2new}(\mathbf{x}_1) \\ \vdots \\ \mathbf{h}_{2new}(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} h_1(\mathbf{x}_1) & \cdots & h_{L-L'}(\mathbf{x}_1) \\ \vdots & \vdots & \vdots \\ h_1(\mathbf{x}_N) & \vdots & h_{L-L'}(\mathbf{x}_N) \end{bmatrix} \tag{5.6}$$

The total hidden layer output in the second layer of S-ELMs is combined as:

$$\mathbf{H}_2 = [\mathbf{H}_1{}', \mathbf{H}_{2new}] \tag{5.7}$$

In the second layer, the same procedure as the previous layer can be taken. Firstly we calculate the output weight β_2 based on the $\mathbf{H}_2$ obtained in equation (5.4) and equation (5.7), then perform PCA dimension reduction to ${\beta_2}^T$ and use the eigenvectors obtained during the PCA dimension reduction process to calculate the reduced hidden layer output $\mathbf{H}_2{}'$ and output the corresponding significant nodes to upper layer to function as a part of the hidden nodes and so on until the last layer. In the last layer, there is no need of reducing the number of hidden nodes. Equation (3.16) is used to calculate the output of the whole network. The above PCA dimension reduction and ELM feature mapping combination process can be described as the following steps:

$$\begin{aligned}
\mathbf{H}_1 &\rightarrow \mathbf{H}_1{}' \\
[\mathbf{H}_1{}', \mathbf{H}_{2new}] &\rightarrow \mathbf{H}_2{}' \\
[\mathbf{H}_2{}', \mathbf{H}_{3new}] &\rightarrow \mathbf{H}_3{}' \\
&\vdots \\
[\mathbf{H}_{N-2}{}', \mathbf{H}_{N-1new}] &\rightarrow \mathbf{H}_{N-1}{}' \\
[\mathbf{H}_{N-1}{}', \mathbf{H}_{Nnew}]
\end{aligned}$$

The algorithm for Stacked ELMs learning network can be summarized in Algorithm 1.

5.3.3 Generalization capability

The proposed Stacked ELMs learning network breaks the large single hidden layer ELM network to a total number of U small ELM networks. From second layer onwords, there

Algorithm 2 Stacked ELMs Learning Network

Given a large training dataset $\Re = \{(\mathbf{x}_i, \mathbf{t}_i) | x_i \in \mathbf{R}^n, \mathbf{t}_i \in \mathbf{R}^m, i = 1, \cdots, N\}$, activation function $g(x)$, number of hidden nodes in each layer L, number of targeted combined nodes L', regularization coefficient C and number of total layers S:

step 1) **Apply Extreme Learning Mahcine algorithm for layer** 1**:**

a) Randomly generate the hidden layer parameters: input weight $\mathbf{w}_i$ and bias b_i, $i = 1, \cdots, L$.

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

c) Calculate the output weight β:

$$\beta = \left(\frac{\mathbf{I}}{C} + \mathbf{H}^T \mathbf{H} \right)^{-1} \mathbf{H}^T \mathbf{T} \tag{5.8}$$

step 2) **PCA dimension reduction of** β^T. Reduce the dimension of β^T from L to L'. The reordered eigenvectors produced in this step is recorded as $\tilde{\mathbf{V}} \in \mathbf{R}^{L,L'}$.

step 3) **Learning step for layer** $2 \rightarrow S$:

for $p = 1$ to $(S-1)$

a) Randomly generate $(L - L')$ hidden nodes and calculate corresponding $\mathbf{H}_{new}$.

b) Combined significant nodes $\mathbf{H}'$:

$$\mathbf{H}' = \mathbf{H}\tilde{\mathbf{V}} \tag{5.9}$$

c) Hidden layer output matrix: $\mathbf{H} = [\mathbf{H}', \mathbf{H}_{new}]$.

d) repeat step 1(c) and step 2.
Remark: step 2 is not needed in the last layer S.

endfor

step 4) **Output of the Stacked ELMs learning network in the final layer:**

$$\mathbf{F}_{out} = \mathbf{H}\beta \tag{5.10}$$

are $(L-L')$ new randomly generated nodes in each layer. Suppose during the PCA dimension reduction of output weight β and eigenvector transformation of hidden layer output $\mathbf{H}$ step, it retains 100% of the original information, the Stacked ELMs learning network can simulate a large single hidden layer ELM network with a total number with $\tilde{L}$ hidden nodes, where

$$\tilde{L} = L + (L-L') \times (S-1) \tag{5.11}$$

However, due to the information loss during the PCA dimension reduction step in every layer, the actual number of total hidden nodes that the Stacked ELMs learning network can simulate is smaller than $\tilde{L}$. Suppose after one PCA dimension reduction process, the remaining nodes contain a portion of η of original information, where η is a number between 0 and 1. Equation (5.11) becomes:

$$\tilde{L} = \eta^{S-1}L + \eta^{S-2}(L-L') + \cdots + (L-L') \tag{5.12}$$

Equation (5.12) indicates that the more number of layers, the more information will be lost for the lower layers. Figure 5.5 shows the relationship between the number of hidden nodes in each layer L', the total number of layer S and the total number of nodes $\tilde{L}$. We can observe that in order to obtain the same testing accuracy as a single ELM with 3000 hidden nodes, the S-ELMs requires different number of layers for different number of hidden nodes in each layer. When the number of hidden nodes L in each layer is 200, 500 and 1000, and the targeted combined nodes L' is chosen as $10\% \times L$, it requires about 27.46, 10.58 and 4.62 layers respectively from figure 5.5. However, from equation (5.11), we can calculate the ideal number of total layers as 16.56, 6.56 and 3.22, so the scale-up factor (measures the information lost, the higher the more information is lost) is: $27.46 \div 16.56 = 1.66$, $10.58 \div 6.56 = 1.61$ and $4.62 \div 3.22 = 1.43$. In other words, compared with the ideal number of layers, a smaller number of hidden nodes in each layer requires a larger number of layers to cover the information loss in each layer. Hence it is advisable to choose a not too small number of hidden nodes L for each layer,

yet L cannot be too large that is beyond the computing machine's memory capability.

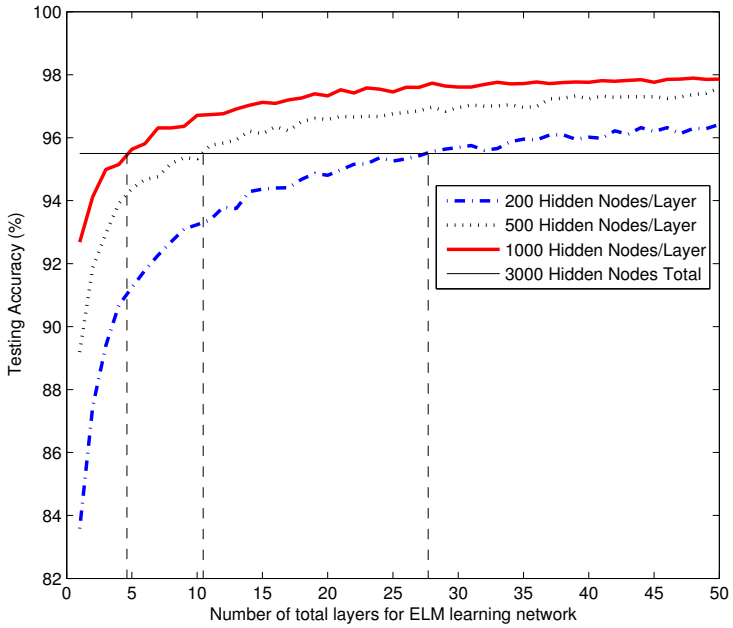


Figure 5.5: The testing accuracy of Stacked ELMs learning network when number of hidden nodes is chosen as 200, 500 and 1000 respectively. The horizontal line shows the testing accuracy of a single ELM with 3000 hidden nodes.

5.3.4 Computational complexity and memory requirement

In a single ELM network, when solving a large scale dataset application, the number of hidden nodes $\tilde{L}$ required is usually a large number so that ELM can map the data to high enough dimensional space and achieve a good generalization performance. In the solution given by equation (3.16), matrix calculation involves multiplication $\mathbf{H}^T\mathbf{H}$ of size $\tilde{L} \times \tilde{L}$ and inversion $(\frac{\mathbf{I}}{C} + \mathbf{H}^T\mathbf{H})^{-1}$ of size $\tilde{L} \times \tilde{L}$ as well. When $\tilde{L}$ becomes larger, the computational cost increases dramatically. In the Stacked ELMs learning network, matrix in each layer is of much smaller size $L \times L$. In the previous subsection, we have discussed the relationship between $\tilde{L}$ and L. In a rough estimation, $\tilde{L} = \sigma S \times L$, where σ is a number between 0 and 1, and S is the number of total layers. In the normal implementation of the S-ELMs, σ is usually close to 1. In this sense, the computational

complexity of single ELM network can be estimated as $\sigma^2 S^2$ while the Stacked ELMs learning network's computational complexity is only S, with $S \ll \sigma^2 S^2$. Noted that the PCA dimension reduction process will also consume some computational power, we also did some tests on how much extra computational time it will cost on the PCA dimension reduction step. It is observed that, the PCA dimension reduction step will typically cost about 20% computational time for each layer regardless of the size of the network on that layer.

The S-ELMs has the same advantage on memory requirement as well. For a single ELM, the memory required to store $\mathbf{H}^T\mathbf{H}$ is $\tilde{L} \times \tilde{L}$, and requires a lot more during the calculation. While for S-ELMs, the memory required to store $\mathbf{H}^T\mathbf{H}$ in each layer is $L \times L$ only, and this can be cleared when moving into the next layer. Thus the memory requirement in Stacked ELMs learning network is much less than a single ELM network.

5.4 Performance Verification

5.4.1 MNIST dataset and simulation environment description

The MNIST database of handwritten digits [94] is a popular database for many researchers to try machine learning techniques and pattern recognition methods. It consists of 60000 training samples, 10000 testing samples and 784 features for each sample. Many methods have been tested on this dataset including Support Vector Machines (SVMs), Neural Networks and many other classification algorithms [95]. In this work, we will only test on the original dataset without any data preprocessing techniques. The simulations are carried out in the Matlab 2011b environment runing on a Intel Xeon E7-8870 2.40 GHz 12G RAM computing node.

5.4.2 Effects of parameters on the generalization performance

There are four parameters the user can adjust to achieve the best testing accuracy: the number of hidden nodes in each layer L, the number of targeted significant nodes L', the total number of layer S and the regularization parameter C. As discussed in section 5.3.3, from equation (5.11) and (5.12), we can see that L, L' and S are correlated parameters. L' is the number of retained nodes in each layer, practically we can fix it as 10% of L. As shown in figure 5.5, different numbers of L would have effect on the total number of layers S required to simulate a certain size ELM network. In practice, the user can set fixed numbers for L and L', then tune S which nearly has a linear relationship with the total number of hidden nodes $\tilde{L}$ that the whole network can simulate. Figure 5.6 shows the relationship between S and the training and testing accuracy when $L = 1000$, $L' = 100$ and $C = 2^{(-6)}$. The figure is very similar to figure 5.1. In fact, they have the same mechanism: it shows the relationship between the number of hidden nodes of ELM network (or the total number of hidden nodes that the S-ELMs tries to simulate) and the testing accuracy. From figure 5.6 we can see that, the testing accuracy increases when the number of layers S increases. Of course, same as the single ELM, there will be an optimal number of S, beyond which over-fitting problem may occur.

The regularization factor C here in the Stacked ELMs learning network works similarly to that in the single ELM. When $C \to \infty$, equation (3.16) becomes:

$$\lim_{C\to\infty} \mathbf{h}(\mathbf{x})\left(\frac{\mathbf{I}}{C}+\mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T} = \mathbf{h}(\mathbf{x})\left(\mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T} \tag{5.13}$$

Equation (6.11) is the same as the *left pseudoinverse* solution to preliminary ELM [35]. Hence if we simply set C as a large enough number, the whole network can at least work as well as the original ELM with the same number of hidden nodes. Figure 5.7 shows the relationship between the testing accuracy and the regularization factor C. There is a small area of peak when C is around $2^{(-6)}$ and the testing accuracy tends to be stable after the peak. Theoretically, C has little effect on the Stacked ELMs learning network

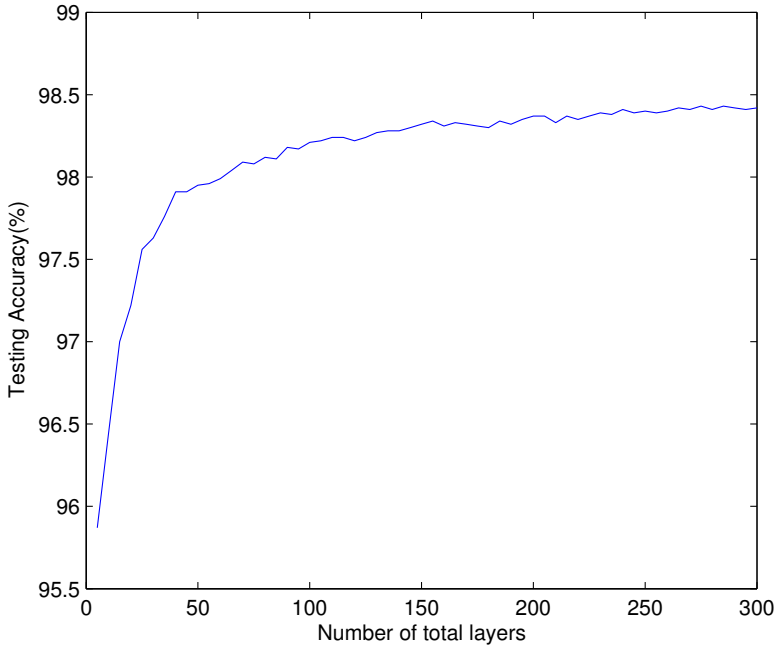


Figure 5.6: Relationship between the testing accuracy and total number of layers for the Stacked ELMs learning network on MNIST dataset

where in each layer for a single ELM, $N \gg L$, so that $\mathbf{H}^T\mathbf{H}$ is highly unlikely to be singular, then the regularization coefficient has little effect on the overall generalization performance.

5.4.3 Generalization performance

The performance of the Stacked ELMs learning network on the MNIST dataset is shown in table 5.1. We set $L = 1000$ and $L' = 100$ for each layer. The simulation started from 6000 to 60000 training samples, every time increasing 6000 training samples. The corresponding number of total layers S increased linearly as well. From figure 5.6, we can see that the testing accuracy begins to stabilize when the number of total layers S is close to 300. Hence we selected $S = 300$ as the total number of layers to train the whole MNIST dataset. We also tested the SVM's performance on those training datasets using *LibSVM*[97]. During the simulation of *LibSVM*, we searched the optimal cost parameter

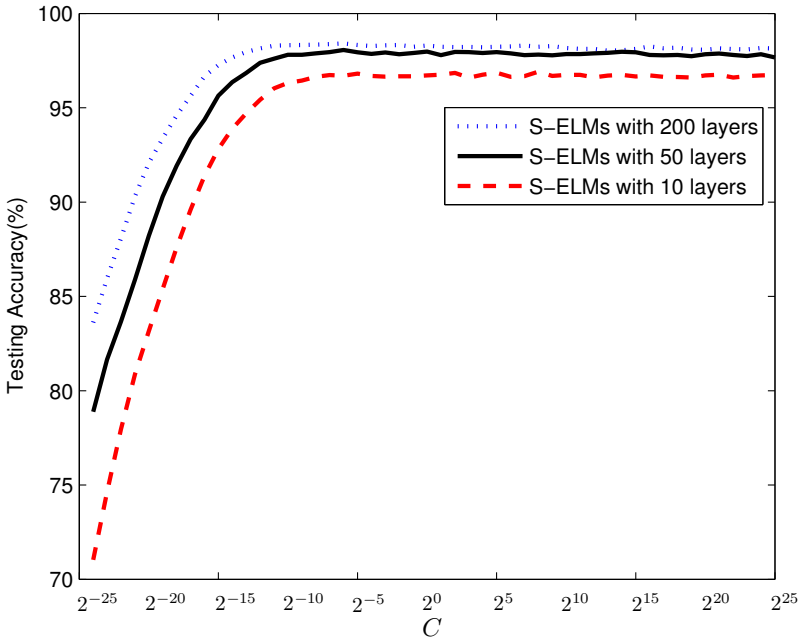


Figure 5.7: Effect of C on the testing accuracy when number of total layer U is chosen as 10, 50, 200

C and kernel parameter γ from 15 different values: $\{0.0001, 0.001, 0.01, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100, 1000, 10000\}$.

S-ELMs vs Single ELM

To check how well the S-ELMs can simulate a single large ELM, we tested the performance of a single ELM with up to 60000 hidden nodes and compared the performance with S-ELMs. The simulations of single ELM are carried out in a high-performance computer with 2.52-GHz CPU and 48-GB RAM. To obtain a fairly good testing rate, the optimal number of hidden nodes increases rapidly when increasing training samples. However, even with the help of a high-performance computer, we can only use maximum of 60000 hidden nodes when the training samples are more than 36000. Simulation results are shown in table 5.1. The accuracy of both single ELM and S-ELMs are taken as the average of 20 runs. As we can observe from table 5.1, for data size range from 6000 to 30000, S-ELMs obtains similar or slightly better overall testing rate

than single ELM while requiring less training time. For data size range from 36000 to 60000, S-ELMs tends to be more stable (smaller testing deviation) with a better testing rate. The large number of hidden nodes used in the single ELM will cost heavily during computation. Hence it requires more computational time and space. The computation of single ELM with a small number of hidden nodes used in each layer of S-ELMs is extremely fast. Therefore though S-ELMs usually requires a very deep number of layers, the training time and computational space are less than single ELM if similar level of testing accuracies are pursued.

S-ELMs vs SVM

Due to the randomness property of ELM, S-ELMs usually obtains a range of testing accuracies for certain training and testing samples. Figure 5.8 shows that the testing accuracies of SVM with different training samples are all addressed within the testing accuracy range of S-ELMs. Simulation results in table 5.1 also shows that S-ELMs can achieve very similar generalization performance on both testing accuracy and training time to that of *LibSVM*. However, since *LibSVM*'s main code runs under the C/C++ environment, and our proposed S-ELMs is purely matlab based, the actual time *LibSVM* required could be a lot more than S-ELMs if it ran in the same environment. We also tried using another SVM package: SVM and Kernel Methods Matlab Toolbox [69] which is fully Matlab based. However, we always encounter problems during optimization process, especially when training samples increase to a large number. Thus, in this work we only compare the performance of S-ELMs with *LibSVM*. To find the best testing accuracy of SVM, one usually will do a grid search for the best pair of parameters, which will consume a large amount of time. The training time of SVM also depends on the parameters chosen, some non-optimal parameters can cost a long time for the optimization process during training (eg. when $C = 10, \gamma = 0.001$, training will take 822 seconds while when $C = 10, \gamma = 0.1$, training will take 8205 seconds). Unlike SVM, the number of total layers S usually has one directional trend with the testing accuracy as shown in figure 5.6. Hence the parameter tuning process for S-ELMs could be quite

less time consuming than SVM. As discussed in the previous section, from figure 5.7 we know the value of C will not affect the final testing accuracy too much as long as it is beyond certain threshold value. If the application does not require a high precision of accuracy, one can simply set C to be a large number. This could make the Stacked ELMs learning network extremely flexible to be used in many industrial large data applications.

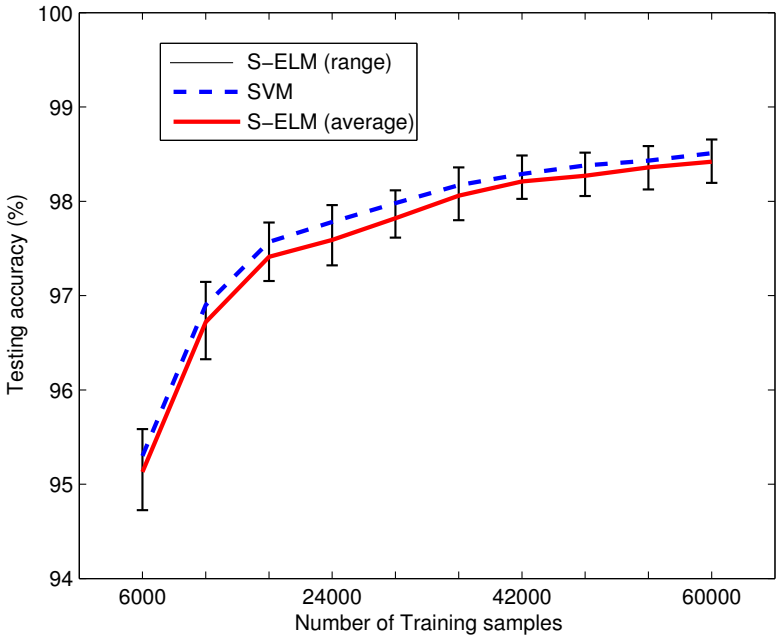


Figure 5.8: Performance comparison of Stacked ELMs and SVM on MNIST Dataset with different number of training samples

5.5 Summary

In this work, we proposed a Stacked ELMs learning network (S-ELMs). The network consists of multiple ELMs with a small number of hidden nodes in each layer to simulate a single ELM with large number of hidden nodes. It is especially useful to solve very large dataset problems because single ELM usually incurs difficulty in computation when its number of hidden nodes is very large. The proposed Stacked ELMs learning network employs the PCA dimension reduction method to reduce the number of hidden

Table 5.1: Performance comparison of Single ELM, Stacked ELMs, and SVM on MNIST Dataset.

# training data	Single ELM					S-ELMs(1000 nodes per layer)					LIBSVM				
	C	L	Training Time(s)	Testing Rate(%)	Testing Dev(%)	C	S	Training Time(s)	Testing Rate(%)	Testing Dev(%)	C	γ	# SVs	Training Time(s)	Testing Rate(%)
6000	2^{-6}	15000	50.09	94.66	0.27	2^{-6}	30	39.14	95.13 (94.94 $\sim$ 95.37)	0.26	1	0.01	3895	58.14	95.3
12000	2^{-6}	25000	221.5	96.60	0.17	2^{-6}	60	126.3	96.72 (96.53 $\sim$ 96.94)	0.16	10	0.01	6479	188.7	96.9
18000	2^{-6}	35000	988.4	97.10	0.26	2^{-6}	90	258.2	97.41 (97.31 $\sim$ 97.62)	0.13	5	0.01	8608	300.9	97.57
24000	2^{-6}	45000	1263	97.22	0.15	2^{-6}	120	439.2	97.59 (97.48 $\sim$ 97.80)	0.11	10	0.01	10435	543.1	97.78
30000	2^{-6}	50000	1836	97.46	0.25	2^{-6}	150	677.8	97.82 (97.74 $\sim$ 97.99)	0.09	10	0.01	12126	663.4	97.98
36000	2^{-6}	60000	3362	97.70	0.09	2^{-6}	180	958.6	98.06 (97.94 $\sim$ 98.22)	0.08	10	0.01	13736	984.2	98.17
42000	2^{-6}	60000	3461	97.79	0.17	2^{-6}	210	1255	98.21 (98.14 $\sim$ 98.37)	0.09	10	0.01	15208	1218	98.29
48000	2^{-6}	60000	3752	97.84	0.21	2^{-6}	240	1632	98.27 (98.17 $\sim$ 98.40)	0.08	10	0.01	16500	1458	98.38
54000	2^{-6}	60000	4064	98.01	0.20	2^{-6}	270	2061	98.36 (98.24 $\sim$ 98.47)	0.06	10	0.01	17830	1954	98.43
60000	2^{-6}	60000	4392	98.04	0.22	2^{-6}	300	2457	98.42 (98.31 $\sim$ 98.54)	0.08	10	0.01	19046	2578	98.51

nodes in each layer. The retained nodes (combined significant nodes) from previous layer are carried forward to the next layer to merge with new random nodes in that layer and so on. The users can determine the number of layers of the S-ELMs to make it deep enough for the given machine learning tasks.

Simulation result on MNIST dataset shows that: 1), comparing with original ELM, S-ELMs requires less training time and memory when similar accuracy is pursued; 2), S-ELMs has similar generalization performance comparing with the popular used *LibSVM* and the simulation time of S-ELMs is similar to *LibSVM* as well. Considering that the main code of *LibSVM* runs under C/C++ environment, S-ELMs could be expected faster than *LibSVM* if running in the same environment. However, the greatest feature of S-ELMs is still its low requirement on memory when solving very large data problems. The user can increase the total number of layers to simulate a large enough ELM network for larger size applications. Hence S-ELMs can serve as a basic framework for solving large data problems. Based on this framework, we plan to propose alternative ELM variants in each layer to improve the performance in the future.

Chapter 6

Extreme Learning Machine Autoencoder Based Stacked Networks

In Chapter 5, we introduced the Stacked ELMs (S-ELMs) that is designed to solve very large data problems without incurring memory problems and can achieve good testing accuracy. In this work, we further investigated S-ELMs and tried improve S-ELMs' testing accuracy by adding ELM based autoencoders in each layer of S-ELMs. The ELM based autoencoders help to capture useful data features that can be used for further ELM mapping. The ELM based autoencoders serve the purpose as unsupervised pretraining of data. Simulation results showed that when implementing ELM based autoencoders into each layer of S-ELMs, it could improve the accuracy significantly and the accuracy was better than several state of the art techniques such as Support Vector Machines (SVMs) and Deep Belief Networks (DBN). The main advantages of our proposed method are its ability in solving large data problems with low memory requirements and its very fast training speed.

6.1 Introduction

In the current fast growing world, information is collected and analyzed from every corner of the universe. The size and dimensionality of data become larger and larger. It is a challenging problem for many researchers to interpret such large datasets and perform meaningful analysis. In Chapter 5, we proposed a Stacked ELMs (S-ELMs) that breaks a large ELM network into multiple serially connected small ELMs. Compared with single large ELM network, the S-ELMs has low computational cost and low memory requirement. The idea is inspired by the deep architectures. e.g. stacked autoencoder networks [25], where each layer is serially connected and propagates the output to the next layer. However, S-ELMs does not incorporate the key idea of unsupervised pretraining of data in deep learning, it is purely a solution that aims to help ELM to solve large scale problems.

In this work, to further improve the testing accuracy, especially for data without properly selected features, we implemented ELM autoencoder to each layer of S-ELMs. ELM autoencoder is an autoencoder that uses ELM algorithm as the training algorithm to reconstruct the input data. The ELM autoencoder can serve as unsupervised pretraining of data and help to capture interesting input data structure that may be useful for improving model generalization ability. Thus, ELM autoencoder based S-ELMs may achieve better generalization performance. Similar to S-ELMs, ELM autoencoder based S-ELMs is capable of solving large dataset problems with low computational cost. Simulation results show that ELM autoencoder based S-ELMs can achieve similar or better testing accuracy compared with other state of the art results.

6.2 Related Work: Autoencoder Networks

Autoencoder/Auto-associative neural networks are neural networks that encode the input $\mathbf{X} \in \mathbb{R}^d$ in certain representation $\mathbf{H} \in \mathbb{R}^L$ to reconstruct the original input $\mathbf{X}$. That is,

the desired output of an autoencoder $\mathbf{Y} \in \mathbb{R}^d$ is set to be the same as the input: $\mathbf{Y} = \mathbf{X}$. The autoencoder networks usually apply back-propagation algorithm for training. A simple one hidden layer autoencoder network diagram is presented in figure 6.1. The number of hidden units L can be smaller or larger than the data dimension d, both can help us to discover interesting structure about the data (compressed or sparse representation of the input data). If the hidden layer is linear and mean square error criterion is used for training, the Autoencoder functions similarly to PCA: the hidden units learn to represent the principle components of the data [29]. If the hidden layer is non-linear, the autoencoder is capable of capture multi-model aspects of input data distribution [98].

Recently autoencoders have been working as a key stage in many deep architecture approaches [99, 26, 27]. In those deep learning algorithms, autoencoders perform unsupervised pretraining of data, which leads to state of the art performance on many challenging classification and regression problems.

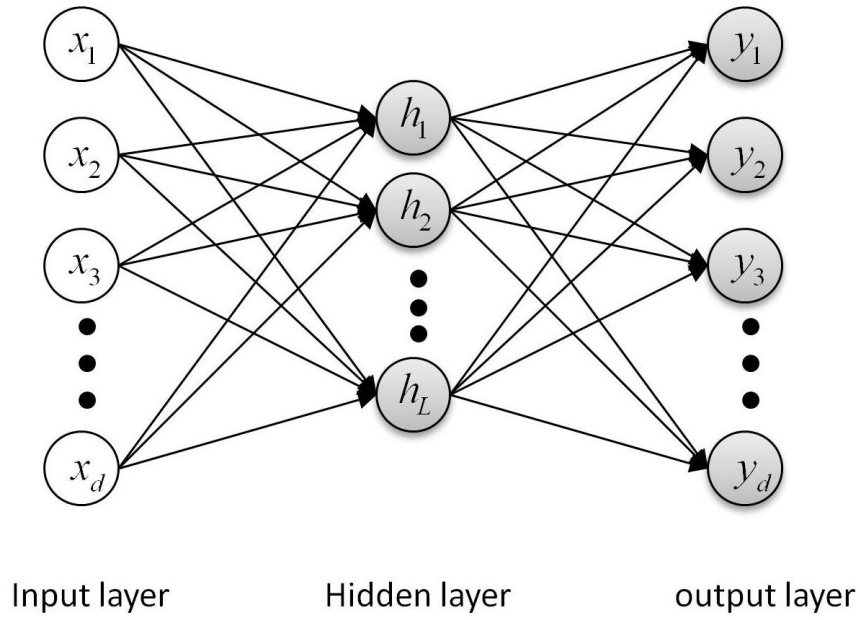


Figure 6.1: Network structure of one hidden layer autoencoder

6.3 ELM Auto-encoder based Stacked ELMs

6.3.1 ELM auto-encoder

Unlike conventional autoencoders that usually apply back-propagation algorithm for training to obtain the identity function, instead, we use Extreme Learning Machine (ELM) as the training algorithm for an autoencoder. Since the network structure of ELM and one hidden layer autoencoder are exactly the same, ELM algorithm could be immediately implemented to train the autoencoder by setting the desired output equal to its input. We name the ELM algorithm based autoencoder as ELM autoencoder. In [100], Kasun et al. also proposed an ELM based autoencoder. In their approach, the hidden layer parameters are not pure randomly generated but orthogonal randomly generated. In our approach of ELM autoencoder, the data $\mathbf{X} \in \mathbb{R}^d$ is firstly mapped to ELM feature space $\mathbf{H} \in \mathbb{R}^l$ (ELM space representation of the data), then the ELM space representation $\mathbf{H}$ is reconstructed to the original input $\mathbf{X}$ through the reconstruction matrix $\beta \in \mathbb{R}^{l,d}$, that is: $\mathbf{X} = \mathbf{H}\beta$. The reconstruction matrix β that may contain interesting input data distributions is then retained for the usage of data pretraining in a fresh ELM, where instead of randomly generating input weights, β^T is used as the input weights. Figure 6.2 shows such process. The reconstruction matrix of ELM autoencoder used in a fresh ELM can function as an unsupervised pretraining step of the data, and will likely result in better generalization performance.

6.3.2 Implementation of ELM autoencoder in Stacked ELMs

In the Stacked ELMs, from layers $2 \rightarrow S$, the hidden layer output of each ELM consists of output from the previous layer and the new random nodes. Here we can replace the random nodes to some "pretrained" sigmoid nodes. The "pretrained" sigmoid nodes are obtained from ELM autoencoder. In an ELM autoencoder with l number of hidden units, the input data with N arbitrary samples $(\mathbf{x}_i, \mathbf{t}_i)$, where $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{id}]^T \in \mathbb{R}^d$

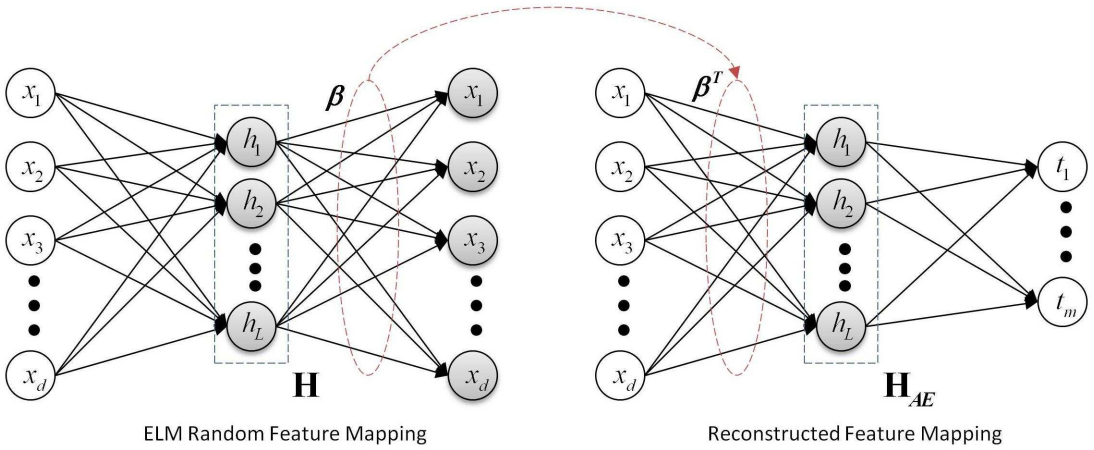


Figure 6.2: In ELM autoencoder, ELM is used as the training algorithm for the network, the transpose of output weight β (reconstruction matrix) is used as the input weight of a normal ELM

and $\mathbf{t}_i = [t_{i1}, t_{i2}, \ldots, t_{im}]^T \in \mathbb{R}^m$, is reconstructed at the output layer through the function:

$$\sum_{i=1}^{l} \beta_i g(\mathbf{a}_i \cdot \mathbf{x}_j + b_i) = \mathbf{x}_j, \quad j = 1, \ldots, N \tag{6.1}$$

where $\mathbf{a}_i = [a_{i1}, a_{i2}, \ldots, a_{id}]^T$ is the randomly generated input weight, $\beta_i = [\beta_{i1}, \beta_{i2}, \ldots, \beta_{id}]^T$ is the reconstruction matrix, and $g(\cdot)$ is the activation function of the hidden units. The above equation (6.1) can be written in the matrix format:

$$\mathbf{H}\beta = \mathbf{X} \tag{6.2}$$

The reconstruction matrix β can be simply obtained as: $\beta = \mathbf{H}^{\dagger}\mathbf{X}$. Or similar to the equality constrained optimization method based ELM [31], we can add a regularization term in the solution, which makes the solution unlikely to be singular. Then the reconstruction matrix β becomes: $\beta = \left(\frac{\mathbf{I}}{C} + \mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{X}$. When we use the inverse of reconstruction matrix of ELM encoder as the input weight of another ELM, the hidden

layer output $\mathbf{H}_{AE}$ can be obtained as:

$$\mathbf{H}_{AE} = \begin{bmatrix} g(\beta_1^T \cdot \mathbf{x}_1 + b_1) & \cdots & g(\beta_l^T \cdot \mathbf{x}_1 + b_l) \\ \vdots & \vdots & \vdots \\ g(\beta_1^T \cdot \mathbf{x}_N + b_1) & \vdots & g(\beta_l^T \cdot \mathbf{x}_N + b_l) \end{bmatrix}_{N \times l} \tag{6.3}$$

where $\beta = \left(\frac{\mathbf{I}}{C} + \mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{X}$.

Remark: If we set total number of hidden nodes in one layer ELM as L and number of targeted combined nodes as L', for the first layer of ELM encoder based S-ELMs, we choose $l = L$; for layers $2 \rightarrow S$, we choose $l = L - L'$.

As shown in figure 6.3 of the ELM autoencoder based S-ELMs network structure, ELM autoencoder is used in every layer, and help to generate the reconstruction matrix β. The inverse of β is used as the input weight to generate $\mathbf{H}_{AE}$. In the first layer, $\mathbf{H}_{AE} \in \mathbb{R}^{N,L}$ is reduced to $\mathbf{H}' \in \mathbb{R}^{N,L'}$ using PCA dimension reduction method. $\mathbf{H}'$ is propagated to second layer and functions as part of the hidden layer matrix. In the second layer, ELM autoencoder with $L - L'$ hidden nodes is created, thus to generate the hidden layer output in second layer $\mathbf{H}_{AE} \in \mathbb{R}^{N,L-L'}$. $\mathbf{H}_{AE}$ is then combined with the PCA reduced hidden layer output $\mathbf{H}'$ from previous layer to form the whole hidden layer matrix. $\mathbf{H}_{AE}$ plays the same role as $\mathbf{H}_{new}$ in equation (5.7). Similarly to the Stacked ELMs, this hidden output propagation process will repeat until the last layer. The detailed algorithm of ELM autoencoder based Stacked ELMs is shown in Algotithm 1.

6.3.3 Computational complexity analysis

Similarly to the Stacked ELMs, the ELM autoencoder based S-ELMs has an advantage on computational complexity compared with single large ELM network that has the same generalization ability. If ELM autoencoder is presented as an unsupervised data

Algorithm 3 Stacked ELMs Learning Network

Given a large training dataset $\Re = \{(\mathbf{x}_i, \mathbf{t}_i) | x_i \in \mathbb{R}^d, \mathbf{t}_i \in \mathbb{R}^m, i = 1, \cdots, N\}$, activation function $g(\cdot)$, number of hidden nodes in each layer L, number of targeted combined nodes L', regularization coefficient C and number of total layers S:

step 1) **ELM autoencoder on layer** 1**:**

a) Randomly generate the hidden layer parameters: input weight $\mathbf{a}_i$ and bias b_i, $i = 1, \cdots, L$.

b) Calculate hidden layer output matrix $\mathbf{H}$.

c) Calculate reconstruction matrix β_{rc}:

$$\beta_{rc} = \left(\frac{\mathbf{I}}{C} + \mathbf{H}^T\mathbf{H}\right)^{-1} \mathbf{H}^T\mathbf{X} \tag{6.4}$$

d) Use the reconstruction matrix as input weight of another ELM to generate hidden layer output matrix $\mathbf{H}$:

$$\mathbf{H} = \begin{bmatrix} g(\beta_{rc1} \cdot \mathbf{x}_1 + b_1) & \cdots & g(\beta_{rcL} \cdot \mathbf{x}_1 + b_L) \\ \vdots & \vdots & \vdots \\ g(\beta_{rc1} \cdot \mathbf{x}_N + b_1) & \vdots & g(\beta_{rcL} \cdot \mathbf{x}_N + b_L) \end{bmatrix}_{N \times L} \tag{6.5}$$

e) Calculate output weight of ELM β:

$$\beta = \left(\frac{\mathbf{I}}{C} + \mathbf{H}^T\mathbf{H}\right)^{-1} \mathbf{H}^T\mathbf{T} \tag{6.6}$$

step 2) **PCA dimension reduction of** β^T. Reduce the dimension of β^T from L to L'. The reordered eigenvectors produced in this step is recorded as $\tilde{\mathbf{V}} \in \mathbf{R}^{L,L'}$.

step 3) **Learning step for layer** $2 \rightarrow S$:

for $p = 1$ to $(S-1)$

a) Randomly generate $(L - L')$ hidden nodes and calculate corresponding $\mathbf{H}_{new}$.

b) Calculate reconstruction matrix β_{rc}:

$$\beta_{rc} = \left(\frac{\mathbf{I}}{C} + \mathbf{H}_{new}^T\mathbf{H}_{new}\right)^{-1} \mathbf{H}_{new}^T\mathbf{X} \tag{6.7}$$

step 3) c) Calculate the additional hidden layer output $\mathbf{H}_{AE}$ using the reconstruction matrix:

$$\mathbf{H}_{AE} = \begin{bmatrix} g(\beta_{rc1} \cdot \mathbf{x}_1 + b_1) & \cdots & g(\beta_{rcl} \cdot \mathbf{x}_1 + b_l) \\ \vdots & \vdots & \vdots \\ g(\beta_{rc1} \cdot \mathbf{x}_N + b_1) & \vdots & g(\beta_{rcl} \cdot \mathbf{x}_N + b_l) \end{bmatrix}_{N \times l} \tag{6.8}$$

where $l = L - L'$

d) Calculate the combined significant nodes $\mathbf{H}'$:

$$\mathbf{H}' = \mathbf{H}\tilde{\mathbf{V}} \tag{6.9}$$

e) Hidden layer output matrix: $\mathbf{H} = [\mathbf{H}', \mathbf{H}_{AE}]$.

f) repeat step 1(e) and step 2.

endfor

step 4) **Output of the Stacked ELMs learning network in the final layer:**

$$\mathbf{F}_{out} = \mathbf{H}\beta \tag{6.10}$$

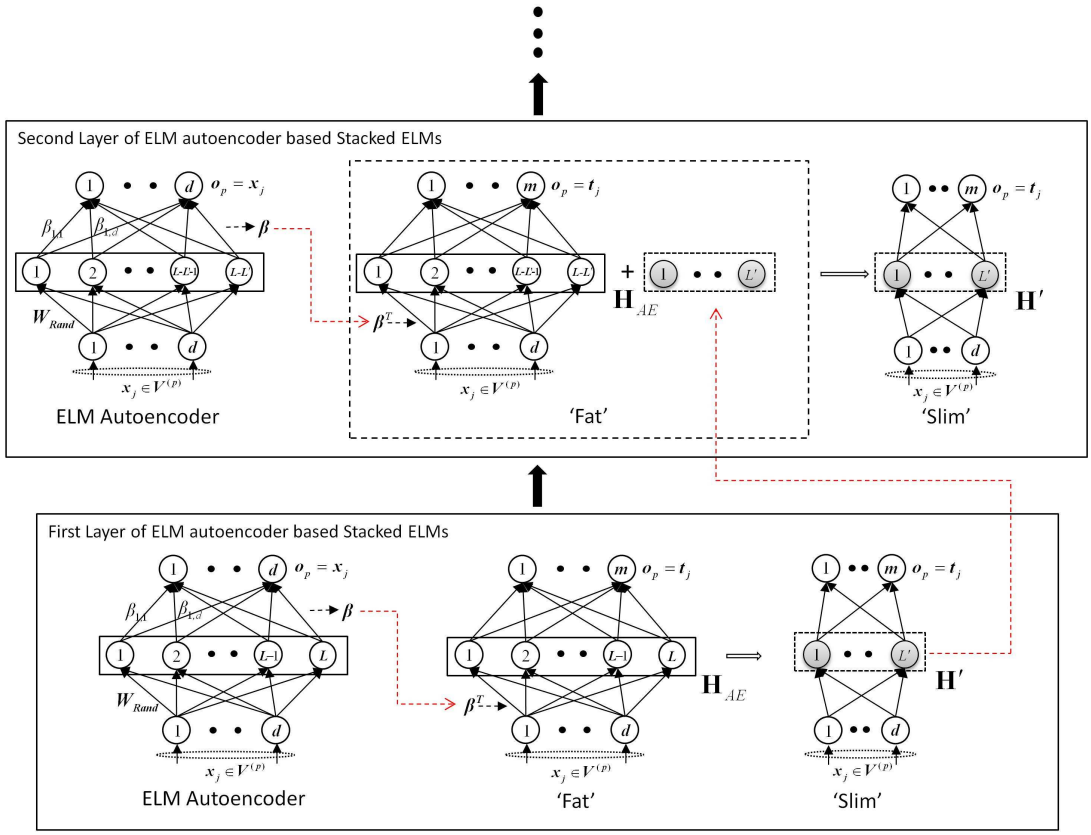


Figure 6.3: Network diagram of ELM autoencoder based S-ELMs

pretraining step for a single large ELM nework with $\tilde{L}$ hidden nodes, the majority of computational cost in equations (6.4) and (6.6) comes from the calculation of $\mathbf{H}^T\mathbf{H}$ and its inversion $(\frac{\mathbf{I}}{C}+\mathbf{H}^T\mathbf{H})^{-1}$. The computational cost can be estimated as $2\times(\tilde{L}\times\tilde{L})$ for single large ELM network with ELM autoencoder. The same assumption of the relationship between $\tilde{L}$ and L holds as the S-ELMs, that is: $\tilde{L}$ can be estimated as $\tilde{L}=\alpha S\times L$, where α is a number between $0\sim 1$, and close to 1. The computational cost of ELM autoencoder based S-ELMs can be estimated as $S\times(L\times L+(L-L')\times(L-L'))$. In our simulation, L' is chosen as 10% of L, hence the computational cost of ELM autoencoder based S-ELMs is $1.81\times S\times L^2$ while the computational cost of single ELM autoencoder based ELM network is $2\times\alpha^2S^2\times L^2$. Same conclusion can be made here, the ELM autoencoder based S-ELMs is much less computational complex than ELM autoencoder based single ELM.

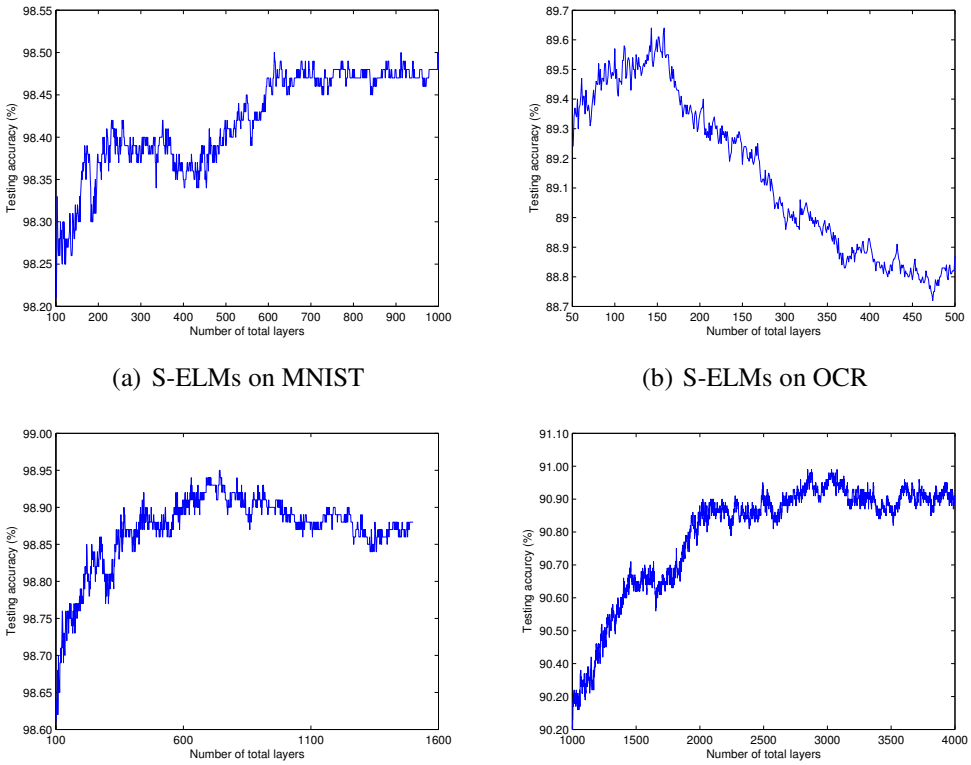
6.4 Performance Verification

6.4.1 Datasets and simulation environment description

In this section, we tested the performance of proposed Stacked ELMs (S-ELMs) and ELM autoencoder based S-ELMs (AE-S-ELMs) on two large datasets: MNIST and OCR Letters. The MNIST database of handwritten digits [94] is a popular database for many researchers to try different machine learning techniques. It consists of 60000 training samples, 10000 testing samples and 784 features for each sample. Many methods have been tested on this dataset including Support Vector Machines (SVMs), Neural Networks and many other classification algorithms [95]. OCR Letters dataset [96] is a public available dataset that corresponds to the handwritten words recognition problem. It contains 42152 training samples and 10000 testing samples with 128 features. The simulations are carried out in the Matlab 2013a environment runing on a Intel Xeon E5-2650 2.00 GHz 256G RAM computer.

6.4.2 Effects of parameters on the generalization performance

There are four parameters the user can adjust to achieve the best testing accuracy of the Stacked ELMs and ELM autoencoder based Stacked ELMs: the number of hidden nodes in each layer L, the number of combined nodes L', the total number of layer S and the regularization parameter C. As discussed in section 5.3.3, from equation (5.11) and (5.12), we can see that L, L' and S are correlated parameters for Stacked ELMs. L' is the number of retained nodes in each layer, practically set as 10% of L. As shown in Figure 5.5, different numbers of L would have effects on the total number of layers S required to simulate a certain size ELM network. In practice, the user can set fixed numbers for L and L', then tune S which nearly has a linear relationship with the total number of hidden nodes $\tilde{L}$ that the whole network can simulate. However for ELM autoencoder based S-ELMs, L and L' need to be properly chosen, and $(L-L')$



(a) S-ELMs on MNIST

(b) S-ELMs on OCR

(c) ELM-AE based S-ELMs on MNIST

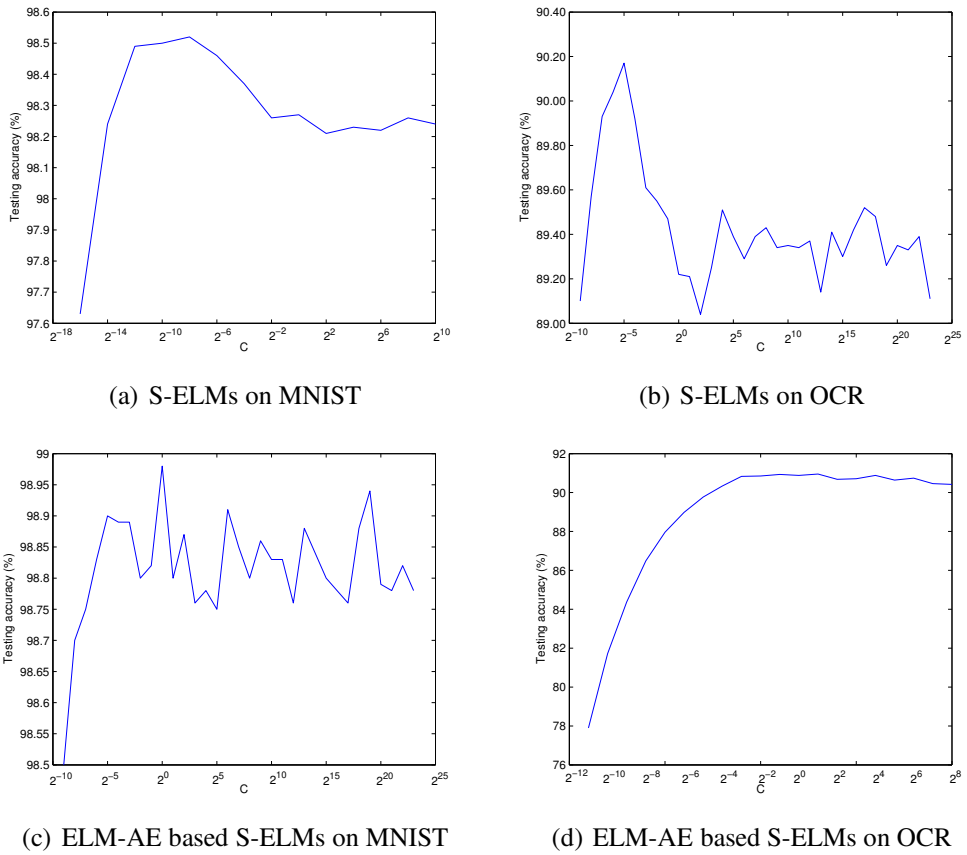
(d) ELM-AE based S-ELMs on OCR

Figure 6.4: The relationship between testing accuracy and number of total layers (iterations)

determines the number of hidden nodes in ELM autoencoder. From our observations, the equal dimension representation or around equal dimension representation usually gives better generalization performance. Hence L and L' need to be chosen such that $(L-L')$ is around the number of features of the training data. The impact of number of total layers and the regularization parameter C are shown in Figure 6.4 and Figure 6.5. From Figure 6.4, it is observed that the testing accuracy will go up when number of total layers increases until some saturate point, it will stay the same or start to drop.

The regularization factor C here in the Stacked ELMs learning network works similarly to that in the single ELM. When $C \rightarrow \infty$, equation (3.16) becomes:

$$\lim_{C\rightarrow\infty} \mathbf{h}(\mathbf{x})\left(\frac{\mathbf{I}}{C}+\mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T} = \mathbf{h}(\mathbf{x})\left(\mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T} \tag{6.11}$$



(a) S-ELMs on MNIST

(b) S-ELMs on OCR

(c) ELM-AE based S-ELMs on MNIST

(d) ELM-AE based S-ELMs on OCR

Figure 6.5: The relationship between testing accuracy and the regularization parameter C

Equation (6.11) is the same as the *left pseudoinverse* solution of the original ELM [91]. Hence if we simply set C as a large enough number, the whole network can at least work as well as the original ELM with the same number of hidden nodes. Figure 6.5 shows the relationship between the testing accuracy and the regularization factor C. One will be able to find the best C by conducting a search from $\{2^{-20}, 2^{-19}, \cdots, 2^{19}, 2^{20}\}$..

6.4.3 Generalization performance

The performance of both Stacked ELMs and ELM autoencoder based Stacked ELMs on the MNIST and OCR letters dataset are shown in table 6.1. We also did simulations on Single ELM, Support Vector Machine (SVM) and Deep Belief Network (DBN) to

compare the generalization capability and effectiveness of different algorithms.

S-ELMs vs Single ELM vs AE-S-ELMs

To check how well the Stacked ELMs can simulate a single large ELM, we tested the performance of a single ELM with 60000 hidden nodes for MNIST and 15000 for OCR letters and compared the performance with Stacked ELMs. The large number of hidden nodes used in the single ELM will cost heavily during computation. Hence it requires a large memory space for computation. Simulation results are shown in table 6.1. The maximum number of hidden nodes per layer shown in table 6.1 indicates the minimum working memory space required to cope with such network. The accuracy of both single ELM and Stacked ELMs are the average of 20 runs. As we can observe from table 6.1, Stacked ELMs with 1000 hidden nodes per layer can obtain much better testing accuracy than a single ELM with 60000 hidden nodes for MNIST and 15000 hidden nodes for OCR letters. Though the maximum number of hidden nodes per layer for Stacked ELMs is much smaller than single ELM, the training time taken for Stacked ELMs are similar or higher than single ELM. This is because Stack ELMs is an iterative learning algorithm, the accuracy increases when number of total layer increase until it reaches a saturate point. As shown in table 6.2, the Stacked ELMs requires 650 and 160 iterations of computation for MNIST and ORC data, respectively. The ELM autoencoder based Stacked ELMs requires even smaller network size compared with Stacked ELMs, and the testing accuracy is higher than Stacked ELMs with similar or more training time.

S-ELMs/AE-S-ELMs vs SVM

We tested SVM's performance on the MNIST and ORC letters datasets using *LibSVM*[97]. During the simulation of *LibSVM*, we searched the optimal cost parameter C and kernel parameter γ from 15 different values: $\{0.0001, 0.001, 0.01, 0.1, 0.2, 0.5, 1, 2, 5, 10,$

Table 6.1: Performance comparison of Single ELM, Stacked ELMs, ELM autoencoder based Stacked ELMs, SVM and DBN on MNIST and OCR letters dataset.

Training Methods	MNIST				OCR Letters			
	Training Time(S)	Testing Rate(%)	Testing Dev(%)	Max # of hidden units per layer	Training Time(S)	Testing Rate(%)	Testing Dev(%)	Max # of hidden units per layer
ELM	4127	98.04	0.22	60000	**172.83**	87.36	0.18	15000
S-ELMs	4621	98.51	0.08	1000	835.62	89.83	0.08	1000
AE-S-ELMs	4347	**98.89**	0.06	**500**	4833	**90.96**	0.03	**400**
SVM	**2453**	98.51	0	19046	605.7	90.15	0	29146
DBN	10.68 h	98.87	-	2000	13.51 h	90.59	-	2000

Table 6.2: Parameters of Single ELM, Stacked ELMs, ELM autoencoder based Stacked ELMs, and SVM on MNIST and OCR letters dataset.

Datasets	ELM		S-ELMs				AE-S-ELMs				SVM	
	L	C	L	L'	C	S	L	L'	C	S	σ	C
MNIST	60000	$2^{(-6)}$	1000	100	$2^{(-8)}$	650	500	50	2^2	700	0.01	10
OCR Letters	15000	2^0	1000	100	$2^{(-5)}$	160	400	280	2^0	3000	0.01	2

20, 50, 100, 1000, 10000}, which resulted in a total of 225 pairs of parameters. To find the best testing accuracy of SVM, one usually will do a grid search for the best pair of parameters, which will consume a large amount of time. The training time of SVM also depends on the parameters chosen, some non-optimal parameters can cost a long time for the optimization process during training (eg. for MNIST dataset, when $C = 10, \gamma = 0.001$, training will take 822 seconds while when $C = 10, \gamma = 0.1$, training will take 8205 seconds). Both the training time and testing accuracy for Stacked ELMs and SVM are at the similar level. However, the maximum number of hidden nodes for SVM (number of support vectors) is much larger than Stacked ELMs. Hence SVM requires a larger memory working space for computation. The ELM autoencoder based Stacked ELMs can achieve much better testing accuracy than SVM and with much smaller network size as well. Both Stacked ELMs and ELM autoencoder based Stacked ELMs can well address the memory issue faced by SVM on large dataset problems.

AE-S-ELMs vs DBN

We also tested the performance of Deep Belief Network (DBN) on this two datasets. The DBN codes are taken from [102]. Though in [27], the reported testing accuracy for MNIST data using the $500-500-2000$ netwrok is 98.8%, we managed to obtain a better accuracy at 98.87% as shown in table 6.1. For the OCR letter data, the reported testing accuracy is 90.32 with the $2000-2000-2000$ network in [103], while we managed to obtain the testing accuracy at 90.59% with the $1000-1000-2000$ network. Similarly to DBN, the ELM autoencoder based Stacked ELMs employs the unsupervised

data pretraining step during each iteration of Stacked ELMs. The simulation results in table 6.1 show that AE-S-ELMs can achieve similar or better testing accuracy than DBN. The maximum number of hidden nodes per layer in AE-S-ELMs is smaller than DBN as well. The greater advantage of AE-S-ELMs compared with DBN is its training speed. Due to the long time of training required, there are no testing deviation results shown for DBN on MNIST and OCR letters. The training time for DBN on MNIST and OCR letters data are 10.68 hours and 13.51 hours, which is 8.84 and 10.06 times slower than AE-S-ELMs, respectively. Thus with better accuracy, smaller network size and faster training speed, AE-S-ELMs would have greater potential in solving many large scale unstructured raw data problems.

6.5 Summary

In this chapter, we proposed an ELM based autoencoder that uses ELM as the training algorithm to reconstruct the original input. The reconstruction matrix was then applied on each layer of Stacked ELMs (S-ELMs) to create new hidden layer nodes. Stacked ELMs (S-ELMs) was developed to tackle large data problems, it can fully stretch the generalization capability of ELM and achieve good testing accuracy on large data problems. With the help of ELM based autoencoder, the performance of Stacked ELMs can be further improved. Simulation results showed that the ELM autoencoder based S-ELMs improved the testing accuracy of S-ELMs significantly, and was also better than other state of the art techniques such as Support Vector Machines (SVMs) and Deep Belief Networks (DBN).

In this work, we only implemented one hidden layer ELM based autoencoder on each layer of S-ELMs, and it already produced very good testing results. In the future, we will work on the stacked autoencoders using ELM algorithm for training and combine with the Stacked ELMs to solve large data problem for better testing accuracy.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis extended the study of Extreme Learning Machine (ELM) to standard optimization problems and built an unified model for solving regression, binary and multiclass classification problems directly. There are three main major contributions as summarized below:

- **The equality optimization method based ELM: a unified learning framework**

  The objective of ELM is to minimize both the training error and output weight norm, hence the ELM problem could be solved in the aspect of standard optimization problems. In chapter 3 of this thesis, same as Least Square Support Vector Machine (LS-SVM) and Proximal Support Vector Machine (PSVM), the equality constrained approach was adopt in ELM's optimization problem. The equality constrained optimization method based ELM provides a unified learning platform with a widespread type of feature mappings and can be applied to solve regression and multiclass classification problems directly. From the optimization method point of view, ELM has milder optimization constraints compared to LS-

SVM and PSVM. Thus in theory, compared to ELM, LS-SVM and PSVM tend to obtain suboptimal solutions and usually require higher computational cost. As verified by the simulation results, ELM tends to have better scalability and achieve similar (for *regression* and *binary class* cases) or much better (for *multi-class* cases) generalization performance at much faster learning speed (up to *thousands* times) than traditional SVM and LS-SVM.

- **Credit Risk Evaluation with Kernelized ELM**

  The kernel based ELM was used as a classification tool to perform the credit risk evaluation. The simulations were done on two credit risk evaluation datasets: the Australian credit and German credit datastes. Three different kernel functions were used during simulation. Simulation results showed that, when using the same kernel function, the kernel based ELM was more suitable for credit risk evaluation than SVM with consideration of overall, good and bad accuracy.

- **ELM for solving big data problems** Due to it simple network structure, ELM is capable of solving larger scale dataset problems than many other machine learning techniques. However if the application size is too large, e.g. the MNIST dataset, ELM also encounters the memory issue. The Stacked ELMs (S-ELMs) method was proposed to solve this issue. The S-ELMs broke a large parallel ELM network into multiple serially connected small ELMs. The Stacked ELMs not only reduced the computational memory requirement, but also saved the training time. In chapter 6, the ELM based autoencoder was implememnted in S-ELMs to further improved the generalization performance of S-ELMs. It was shown that when adding ELM based autoencoder to each layers of S-ELMs network, the testing accuracy could be significantly improved.

7.2 Future Work

This thesis studies the equality constrained optimization method based ELM and implement ELM in two real applications, which shows very promising performance. New learning methods were proposed for ELM to solve large scale data problems as well. Based on the research work conducted in the thesis, there are still some topics that the author thinks can be further investigated, they are:

• The kernel based ELM has shown very good testing rate in many benchmark and real-world applications. However, when dealing with large data or sequentially arriving data, the kernel based ELM will encounter problems, typically the long training time and out of memory problem. Hence there is a need to study online kernel learning methods that can be implemented in kernel based ELM.

• In chapter 6, one hidden layer ELM based autoencoder was implemented on each layer of S-ELMs, and it produced very good testing results. In fact, the stacked autoencoders could have the potential to provide more promising unsupervised pretraiing ability. In the future, the stacked autoencoders using ELM algorithm for training can be studied and combined with the Stacked ELMs to solve large data problem for better testing accuracy.

• As presented throughout the thesis, ELM can achieve good generalization performance for regression, binary and multiclass classification problems. The data can be approximated or classified well in the ELM feature mapping space. Thus it is also interesting to investigate how well ELM can perform in solve clustering problems.

Publication list

Journal Papers:

1. **Hongming Zhou**, Guang-Bin Huang, Han Wang, Zhiping Lin, and Yeng Chai Soh "Stacked Extreme Learning Machines," (under review) *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 2013.

2. Chamara K Liyanaarachchi Lekamalage, **Hongming Zhou**, Guang-Bin Huang and Chi Man Vong, "Representational Learning With Extreme Learning Machine", *IEEE Intelligent Systems-Trends & Controversies*, vol. 28, no. 6, pp.31–24, 2013.

3. Guang-Bin Huang, **Hongming Zhou**, Xiaojian Ding, and Rui Zhang, "Extreme Learning Machine for Regression and Multiclass Classification," *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, vol. 42, no. 2, pp. 513–529, 2012.

4. Arindam Basu, Sun Shuo, **Hongming Zhou**, Meng Hiot Lim, and Guang-Bin Huang, "Silicon spiking neurons for hardware implementation of extreme learning machines," *Neurocomputing*, vol. 102, pp. 125-134, 2012.

5. Guang-Bin Huang, Xiaojian Ding, and **Hongming Zhou**, "Optimization Method Based Extreme Learning Machine for Classification," *Neurocomputing*, vol. 74, pp. 155-163, 2010.

Conference Papers:

1. **Hongming Zhou**, Yuan Lan, Yeng Chai Soh, Guang-Bin Huang and Rui Zhang " Credit risk evaluation with extreme learning machine," *IEEE International Conference on Systems, Man, and Cybernetics (SMC),* Seoul, Korea, 14–17 Oct. 2012.

2. Jiantao Xu, **Hongming Zhou** and Guang-Bin Huang, "Extreme Learning Machine Based Fast Object Recognition," *15th International Conference on Information Fusion (FUSION),* Singapore, 9–12 July 2012.

3. Weiwei Zong, **Hongming Zhou**, Guang-Bin Huang and Zhiping Lin, "Extreme Learning Machine with Adaptive Growth of Hidden Nodes and Incremental Updating of Output Weights," *International Conference on Autonomous and Intelligent Systems (AIS 2011),* Burnaby, BC, Canada, June 22–24, 2011.

Bibliography

[1] S. Haykin. Neural networks: a comprehensive foundation. NY: Macmillan College Publishing Company, 1994.

[2] C. M. Bishop. Neural networks for pattern recognition. NY: Oxford University Press, 1995.

[3] M. T. Hagan, H. B. Demuth, and M. H. Beale. Neural network design. Boston: PWS Publshing Company, 1996.

[4] C. G. Looney. Pattern recognition using neural networks: theory and algorithms for engineers and scientists. NY: Oxford University Press, 1997.

[5] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[6] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: A new learning scheme of feedforward neural networks. In *Proceedings of International Joint Conference on Neural Networks*, pages 985–990, 2004.

[7] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew. Extreme learning machine: theory and applications. *Neurocomputing*, 70:489–501, 2006.

[8] G.-B. Huang, L. Chen, and C.-K. Siew. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Transactions on Neural Networks*, 17(4):879–892, 2006.

[9] G.-B. Huang and L. Chen. Convex incremental extreme learning machine. *Neurocomputing*, 70:3056–3062, 2007.

[10] G.-B. Huang and L. Chen. Enhanced random search based incremental extreme learning machine. *Neurocomputing*, 71:3460–3468, 2008.

[11] P. L. Bartlett. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE Transactions on Information Theory*, 44(2):525–536, 1998.

[12] X. Tang and M. Han, Partial lanczos extreme learning machine for single-output regression problems, *Neurocomputing*, 72(13-15):3066–3076, 2009.

[13] Q. Liu, Q. He, and Z. Shi, Extreme support vector machine classifier, *Lecture Notes in Computer Science*, 5012:222–233, 2008.

[14] B. Frénay and M. Verleysen, Using SVMs with randomised feature spaces: an extreme learning approach, in *Proceedings of The 18th European Symposium on Artificial Neural Networks (ESANN)*, pages 315–320, 2010.

[15] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, and A. Lendasse, OP-ELM: Optimally pruned extreme learning machine, *IEEE Transactions on Neural Networks*, 21(1):158–162, 2010.

[16] W. Deng, Q. Zheng, and L. Chen, Regularized extreme learning machine, in *IEEE Symposium on omputational Intelligence and Data Mining (CIDM2009)*, pages 389–395, 2009.

[17] K.-A. Toh, Deterministic neural classification, *Neural Computation*, 20(6):1565–1595, 2008.

[18] C. Cortes and V. Vapnik, Support vector networks, *Machine Learning*, 20(3):273–297, 1995.

[19] J. A. K. Suykens and J. Vandewalle, Least squares support vector machine classifiers, *Neural Processing Letters*, 9(3):293–300, 1999.

[20] G. Fung and O. L. Mangasarian, Proximal support vector machine classifiers, in *International Conference on Knowledge Discovery and Data Mining*, pages 77–86, 2001.

[21] Y.-J. Lee and O. L. Mangasarian, RSVM: Reduced support vector machines, in *Proceedings of the SIAM International Conference on Data Mining*, pages 01–07, 2001.

[22] H. Drucker, C. J. Burges, L. Kaufman, A. Smola, and V. Vapnik, Support vector regression machines, in *Neural Information Processing Systems 9* (M. Mozer, J. Jordan, and T. Petscbe, eds.), pages 155–161, 1997.

[23] P. Vincent, H. Larochelle Y. Bengio and P. A. Manzagol, Extracting and Composing Robust Features with Denoising Autoencoders, in *Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML08)*, pages 1096 – 1103 2008.

[24] Y. Bengio, Learning deep architectures for AI, *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[25] Y. Bengio, P. Lamblin, D. Popovici and H. Larochelle, Greedy Layer-Wise Training of Deep Networks, in *Advances in Neural Information Processing Systems 19*, pages 153–160, MIT Press 2007.

[26] G. E. Hinton, S. Osindero, and Y. W. Teh. A fast learning algorithm for deep belief nets. *Neural Computation* , 18(7):1527C-1554, 2006.

[27] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786): 504-507, 2006.

[28] Y. Freund and D. Haussler. Unsupervised learning of distributions of binary vectors using 2-layer networks. in *Proceeding of 5th Annual Conference on Neural Information Processing Systems*, pages 912–919, 1991.

[29] H. Bourlard and Y. Kamp, Auto-association by multilayer perceptrons and singular value decopmposition, *Biological Cybernetics*, 59:291–294, 1988.

[30] G.-B. Huang, X. Ding, and H. Zhou, Optimization method based extreme learning machine for classification, *Neurocomputing*, 74:155–163, 2010.

[31] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, Extreme Learning Machine for Regression and Multiclass Classification, *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 42(2):513–529, 2012.

[32] G.-B. Huang, Q.-Y. Zhu and C.-K. Siew, Real-Time Learning Capability of Neural Networks, *IEEE Transactions on Neural Networks*, 17(4):863–878, 2006.

[33] H.-J. Rong, Y.-S. Ong, A.-H. Tan, and Z. Zhu. A fast pruned-extreme learning machine for classfication problem. *Neurocomputing*, 72:359–366, 2008.

[34] Y. Miche, A. Sorjamaa, and A. Lendasse. OP-ELM: theory, experiments and a toolbox. *Lecture Notes in Computer Science*, 5163:145–154, 2008.

[35] N.-Y. Liang, G.-B. Huang, S. P. Saratchandran, and N. Sundararajan. A fast and accurate online sequential learning algorithm for feedforward networks. *IEEE Transactions on Neural Networks*, 17(6):1411–1423, 2006.

[36] J. Cao, Z. Lin, and G.-B. Huang. Composite function wavelet neural networks with extreme learning machine. *Neurocomputing*, 73:1405–1416, 2010.

[37] J. Cao, Z. Lin, G.-B. Huang, and N. Liu. Voting based extreme learning machine. *Information Sciences*, 185:66–77, 2012.

[38] F. Han and D.-S. Huang. Improved extreme learning machine for function approximation by encoding a priori information. *Neurocomputing*, 69:2369–2373, 2006.

[39] J. M. Martinez-Martinez, P. Escandell-Montero, E. Soria-Olivas, J. D. Martin-Guerrero, R. Magdalena-Benedito, and J. Gomez-Sanchis. Regularized extreme

learning machine for regression problems. *Neurocomputing*, 74:3716–3721, 2011.

[40] R. Savitha, S. Suresh, and N. Sundararajan. Fast learning circular complex-valued extreme learning machine (CC-ELM) for real-valued classification problems. *Information Sciences*, 187:227–290, 2012.

[41] E. Soria-Olivas, J. Gomez-Sanchis, J. D. Martin, J. Vila-Frances, M. Martinez, J. R. Magdalena, and A. J. Serrano. BELM: bayesian extreme learning machine. *IEEE Transactions on Neural Networks*, 22(3):505–509, 2011.

[42] X. Tang and M. Han. Partial lanczos extreme learning machine for single-output regression problems. *Neurocomputing*, 72:3066–3076, 2009.

[43] J. Yin, F. Dong, and N. Wang. Modified gram-schmidt algorithm for extreme learning machine. In *Proceedings of International Symposium on Computational Intelligence and Design*, pages 517–520, 2009.

[44] Q.-Y. Zhu, A. K. Qin, P. N. Suganthan, and G.-B. Huang. Evolutionary extreme learning machine. *Pattern Recognition*, 38:1759–1763, 2005.

[45] D. C. Plaut, S. J. Nowlan, and G. E. Hinton. Experiments on learning by back propagation. Technical Report CMU-CS-86-126, Carnegie-Mellon University, 1986.

[46] D. Serre. Matrices: theory and applications. NY: Springer-Verlag, 2002.

[47] C. R. Rao and S. K. Mitra. Generalized inverse of matrices and its applications. NY: Wiley, 1971.

[48] R. Fletcher, *Practical Methods of Optimization: Volume 2 Constrained Optimization*. John Wiley & Sons, 1981.

[49] S. Alex and S. Bernhard. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004.

[50] V. N. Vapnik, *Statistical learning theory*. New York: Wiley, 1998.

[51] K.-A. Toh, Q.-L. Tran, and D. Srinivasan, Benchmarking a reduced multivariate polynomial pattern classifier, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(6):740–755, 2004.65–1595, 2008.

[52] G.-B. Huang, Q.-Y. Zhu, M.-K. Zhi, C.-K. Siew, P. Saratchandran, and N. Sundararajan. Can threshold networks be trained directly? *IEEE Transactions on Circuits and Systems II*, 53(3):187–191, 2006.

[53] J. A. K. Suykens and J. Vandewalle, Training multilayer perceptron classifier based on a modified support vector method, *IEEE Transactions on Neural Networks*, 10(4):907–911, 1999.

[54] T. Poggio, S. Mukherjee, R. Rifkin, A. Rakhlin, and A. Verri, *b*, (A.I. Memo No. 2001-011, CBCL Memo 198, Artificial Intelligence Laboratory, Massachusetts Institute of Technology), 2001.

[55] J. A. K. Suykens, T. V. Gestel, J. D. Brabanter, B. D. Moor, and J. Vandewalle, Least squares support vector machines, Singapore: World Scientific, 2002.

[56] M. Espinoza, J. A. K. Suykens, and B. D. Moor, Fixed-size least squares support vector machines: A large scale application in electrical load forecasting, *Computational Management Science, Special Issue on Support Vector Machines*, 3(2):113–129, 2006.

[57] M. Espinoza, J. A. K. Suykens, R. Belmans, and B. D. Moor, Electric load forecasting - using kernel based modeling for nonlinear system identification, *IEEE Control Systems Magazine, Special Issue on Applications of System Identification*, 27(5):43–57, 2007.

[58] K. D. Brabanter, J. D. Brabanter, J. A. K. Suykens, and B. D. Moor, Optimized fixed-size kernel models for large data sets, *Computational Statistics & Data Analysis*, 54(6):1484–1504, 2010.

[59] P. Karsmakers, K. Pelckmans, K. D. Brabanter, H. V. Hamme, and J. A. K. Suykens, Sparse conjugate directions pursuit with application to fixed-size kernel models, *(in press) Machine Learning*, 2011.

[60] T. V. Gestel, J. A. K. Suykens, G. L. and A. Lambrechts, B. D. Moor, and J. Vandewalle, Multiclass LS-SVMs: Moderated outputs and coding-decoding schemes, *Neural Processing Letters*, 15(1):48–58, 2002.

[61] M. Mike, Statistical datasets, in *http://lib.stat.cmu.edu/datasets/*, Department of Statistics, University of Carnegie Mellon, USA, 1989.

[62] 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.

[63] A. Bordes, S. Ertekin, J. Weston, and L. Bottou, Fast kernel classifiers with online and active learning, *Journal of Machine Learning Research*, 6:1579–1619, September 2005.

[64] J. J. Hull, A database for handwritten text recognition research, *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 16(5):550–554, 1994.

[65] J. Li and H. Liu, Kent ridge bio-medical data set repository, in *http://levis.tongji.edu.cn/gzli/data/mirror-kentridge.html*, School of Computer Engineering, Nanyang Technological University, Singapore, 2004.

[66] S. C. Odewahn, E. B. Stockwell, R. L. Pennington, R. M. Humphreys, and W. A. Zumach, Automated star/galaxy discrimination with neural networks, *The Astronomical Journal*, 103(1):318–331, 1992.

[67] T. R. Golub, D. K. Slonim, P. Tamayo, C. Huard, M. Gaasenbeek, J. P. Mesirov, H. Coller, M. L. Loh, J. R. Downing, M. A. Caligiuri, C. D. Bloomfield, and E. S. Lander, Molecular classification of cancer: Class discovery and class prediction by gene expression monitoring, *Science*, 286:531–537, 1999.

[68] H. Peng, F. Long, and C. Ding, Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1226–1238, 2005.

[69] S. Canu, Y. Grandvalet, V. Guigue, and A. Rakotomamonjy, SVM and kernel methods matlab toolbox, in *http://asi.insa-rouen.fr/enseignants/ arakotom/toolbox/index.html*, Perception Systmes et Information, INSA de Rouen, Rouen, France, 2005.

[70] K. Pelckmans, J. A. K. Suykens, T. V. Gestel, J. D. Brabanter, L. Lukas, B. Hamers, B. D. Moor, and J. Vandewalle, LS-SVMLab toolbox, in *http://www.esat.kuleuven.be/sista/lssvmlab/*, Department of Electrical Engineering, ESAT-SCD-SISTA, Belgium, 2002.

[71] E. I. Altman, Financial ratios, discriminant analysis and the prediction of corporate bankruptcy, *Journal of Finance*, 23:589–609, 1968.

[72] A. Steenackers and M. J. Goovaerts, A credit scoring model for personal loans, *Insurance: Mathematics and Economics*, 8:31–34, 1989.

[73] B. J. Grablowsky and W. K. Talley, Probit and discriminant functions for classifying credit applicants: a comparison, *Journal of Economics and Business*, 33:254–261, 1981.

[74] V. S. Desai, J. N. Crook, and J. Overstreet, A comparison of neural networks and linear scoring models in the credit union environment, *European Journal of Operational Research*, 95:24–37, 1996.

[75] R. Malhotra and D. K. Malhotra, Evaluating consumer loans using neural networks, *Omega*, 31:83–96, 2003.

[76] F. Varetto, Genetic algorithms applications in the analysis of insolvency risk, *Journal of Banking Finance*, 22:1421–1439, 1998.

[77] S. C. Ong, J. J. Huang, and G. H. Tzeng, Buiding credit scoring models using genetic programming, *Expert Systems with Applications*, 29:41–47, 2005.

[78] H. Li, J. Sun, and B. Sun, Financial distress prediction based on or-cbr in teh principle of *k*-nearest neighbors, *Expert Systems with Applications*, 36:643–659, 2009.

[79] T. van Gestel, B. Baesens, J. Garcia, and P. van Dijcke, A support vector machine approach to credit scoring, *Bank en Financiewezen*, 2:73–82, 2003.

[80] L. Yu, S. Y. Wang, and J. Cao, A modified least squares support vector machine classifier with application to credit risk analysis, *International Journal of Information Technology and Decision Making*, 8:697–710, 2009.

[81] L. G. Zhou, K. K. Lai, and L. Yu, Credit scoring using support vector machines with sirect search for parameters selection, *soft Computing*, 13:149–155, 2009.

[82] T. Bellotti and J. Crook, Support vector machines for credit scoring and discovery of significant features, *Expert systems with applications*, 36(2):3302–3308, 2009.

[83] H. Li and J. Sun, Predicting business failure using multiple case-based reasoning combined with support vector machine, *Expert systems with applications*, 36:10085–10096, 2009.

[84] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, Extreme learning machine for rgression and multi-class classification, *submitted to IEEE Transactions on Systems, Man, and Cybernetics*, 42(2)513–529, 2012.

[85] S. Canu, Y. Grandvalet, V. Guigue, and A. Rakotomamonjy, Svm and kernel methods matlab toolbox. Perception Systèmes et Information, INSA de Rouen, Rouen, France, 2005.

[86] Directory page for Top500 lists. Result for each list since June 1993. in *www.top500.org*, 2013.

[87] NVidia CUDA Zone. in *http://www.nvidia.com/object/cuda_home _new.html*, 2007.

[88] What is gpu computing. in *http://www.nvidia.com/object/what–is–gpu–computing.html*, 2000.

[89] Y.-J. Lee and O. L. Mangasarian, RSVM: reduced support vector machines, in *Proceeding of 1st SIAM International Conference on Data Mining*, pages 01–07, 2001.

[90] J. Platt, A resource-allocating network for function interpolation, *Neural Computation*, 3:213–225, 1991.

[91] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, A Fast and Accurate On-line Sequential Learning Algorithm for Feedforward Networks, *IEEE Transactions on Neural Networks*, 17(6):1411–1423, 2006.

[92] I. T. Jolliffe, *Principal Component Analysis*, Springer-Verlag, 1986.

[93] L. I. Smith. *A Tutorial on Principal Components Analysis*, Cornell University, USA, 2002.

[94] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[95] Y. LeCun, C. Cortes, THE MNIST DATABASE of handwritten digits, in *http://yann.lecun.com/exdb/mnist/*, The Courant Institute of Mathematical Sciences, New York University, USA, 1998.

[96] R. Kassel, "OCR dataset", in *http://ai.stanford.edu/ btaskar/ocr/*, MIT Spoken Language Systems Group, USA.

[97] C.-C. Chang and C.-J. Lin, LIBSVM: a library for support vector machines, *ACM Transactions on Intelligent Systems and Technology*, 2(27):1–27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[98] N. Japkowicz, S. J. Hanson and M. A. Bluck, Nonlinear autoassociation is not equivalent to PCA. *Neural Computation*, 12(3):531–545, 2000.

[99] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research* , 11:625–660, 2010.

[100] L. L. C. Kasun, H. Zhou, G.-B. Huang and C. M. Vong, Representational Learning With Extreme Learning Machine, *IEEE Intelligent Systems-Trends and Controversies*, 28(6):31–34, 2013.

[101] G. E. Hinton and R. R. Salakhutdinov. Training a deep autoencoder or a classifier on MNIST digits, in *www.cs.toronto.edu/∼hinton/MatlabForSciencePaper.html*, Retrieved on 2013-03-25.

[102] R. Salakhutdinov and G. Hinton, "Training a deep autoencoder or a classifier on MNIST digits", in *http://www.cs.toronto.edu/h̃inton/MatlabForSciencePaper.html*, Department of Computer Science, University of Toronto, Canada.

[103] R. Salakhutdinov and H. Larochelle, "Efficient Learning of Deep Boltzmann Machines", in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 693–700, 2010.