A Fast MLP-based Learning Method and its Application to Mine Countermeasure Missions

by

Hang Shao

A thesis submitted to Faculty of Graduate and Postgraduate Studies
in partial fulfillment of the requirements
for the degree of Masters of Applied Science in
Electrical and Computer Engineering

Ottawa-Carleton Institute for Electrical and Computer Engineering
School of Electrical Engineering and Computer Science
University of Ottawa

Abstract

In this research, a novel machine learning method is designed and applied to Mine Countermeasure Missions. Similarly to some kernel methods, the proposed approach seeks to compute a linear model from another higher dimensional feature space. However, no kernel is used and the feature mapping is explicit. Computation can be done directly in the accessible feature space. In the proposed approach, the feature projection is implemented by constructing a large hidden layer, which differs from traditional belief that Multi-Layer Perceptron is usually funnel-shaped and the hidden layer is used as feature extractor.

The proposed approach is a general method that can be applied to various problems. It is able to improve the performance of the neural network based methods and the learning speed of support vector machine. The classification speed of the proposed approach is also faster than that of kernel machines on the mine countermeasure mission task.

Acknowledgments

I would like to express my thanks to my supervisor, Prof. Nathalie Japkowicz, for her time, support and valuable advices. This work couldn't be done without her help. I also want to thank my committee members, Prof. Stan Matwin and Prof. B. John Oommen, for their suggestions and comments.

In addition, I would like to thank Mr. Xiaoguang Wang, Dr. Alex Bourque and Dr. Bao Nguyen, who work with me on the DRDC project, for their helpful comments.

Finally, I wish to thank my family and friends, especially my parents. They have always been supportive along the way.

Contents

Abstract I

Acknowledgments II

Contents III

List of Figures V

List of Tables VII

Chapter 1 Introduction 1

1.1 Overview 2

1.2 Plans 3

1.3 Contribution 4

1.4 Organization 5

Chapter 2 Related Work 6

2.1 The MLP and Error Backpropagation 6

2.1.1 The architecture of Multilayer Perceptrons 6

2.1.2 Error Backpropagation 9

2.1.3 Approximation ability of the MLP 11

2.2 Extreme Learning Machine 12

2.2.1 Extreme learning method 12

2.2.2 Approximation ability of ELM 14

2.2.3 Discussions and recent advance 15

2.3 Kernel methods 16

2.3.1 Support Vector Machines 17

2.4 Summary and Discussions 21

Chapter 3 Proposed Approach 23

3.1 The norm of SLFNs weights 25

3.2 The proposed approach 28
3.2.1 SLFN with Weight Decay 29
3.2.2 Input weights training 31
3.2.3 Output weights training 33
3.2.4 The learning steps of the Proposed Approach 34
3.2.5 Relation with other work 35
3.3 Experiments 36
3.3.1 Performance Measure 36
3.3.2 XOR problem 37
3.3.3 Side Scan Sonar Dataset 39
3.3.4 UCI Datasets 46
3.4 Conclusions 52
Chapter 4 Proposed Approach for Regression 53
4.1 Proposed Approach for Regression 54
4.1.1 Weight Decay in regression 55
4.1.2 SLFN weights training 57
4.1.3 The learning steps of the Proposed Regression Method 60
4.1.4 Relationship with SVR 61
4.2 Experiment 62
4.2.1 Synthetic datasets 62
4.2.2 Real life datasets 68
4.3 Conclusions 78
Chapter 5 Conclusion and Future Work 80
5.1 Summary 80
5.2 Conclusion 81
5.3 Future work 83
Bibliography 85

List of Figures

Figure 2.1 Example of the MLP 7
Figure 2.2 Sigmoid with different bias b 8
Figure 2.3 Separating margin 18
Figure 2.4 Soft margin 19
Figure 3.1 Sigmoid in linear working region 26
Figure 3.2 Sigmoid in nonlinear working region 27
Figure 3.3 Different decision boundaries 28
Figure 3.4 Huber-like function 30
Figure 3.5. Output weight calculation 33
Figure 3.6 Output of the proposed approach on the XOR problem 38
Figure 3.7 Decision boundary of the proposed approach on the XOR problem 38
Figure 3.8 Example of image processing results 40
Figure 3.9 Example of Grayscale Histogram 41
Figure 3.10 Contour plot of AUC on η-λ_1 plane 44
Figure 3.11 Contour plot of AUC on γ-C plane 44
Figure 3.12 ROC Curves 1 45
Figure 3.13 ROC curves 2 45
Figure 3.14 Convergence of the proposed approach 46
Figure 3.15 Convergence of the proposed approach on UCI datasets (WDBC) 51
Figure 4.1 Prediction curves produced by SLFN with different norm of weights 53
Figure 4.2 Comparison of square and Huber-like Error 57
Figure 4.3 Equivalent kernel with different norm of input weights 60
Figure 4.4 SinC function 63
Figure 4.5 Comparison of SinC learning results 64
Figure 4.6 Comparison of testing RMSE under different network size 65

Figure 4.7 Impact of network size on the performance of the proposed approach 68
Figure 4.8 Raw time series data 70
Figure 4.9 Pre-processed time series data 71
Figure 4.10 Plot of Testing RMSE of the proposed approach on η-λ_1 plane 73

List of Tables

Table 3.1 Confusion Matrix 36
Table 3.2 Training set of XOR problem 37
Table 3.3 Side Scan Sonar Dataset information..... 41
Table 3.4 Comparison of performance on Side Scan Sonar Data..... 43
Table 3.5 Model parameters and comparison of time on Side Scan Sonar Data 43
Table 3.6 UCI datasets information 47
Table 3.7 Comparison of Training Time(s) on UCI domains..... 48
Table 3.8 Comparison of model parameters on UCI domains..... 49
Table 3.9 Comparison of performance (AUC) on UCI domains 50
Table 4.1 Comparison of different models on SinC approximation problem..... 63
Table 4.2 Comparison of training RMSE and Std. on Multi-dimensional Synthetic Data ……. 66
Table 4.3 Comparison of training time and Std. on Multi-dimensional Synthetic Data……. 67
Table 4.4 Comparison of model parameters on Multi-dimensional Synthetic Data..... 67
Table 4.5 Comparison of testing RMSE and Std. on Multi-dimensional Synthetic Data……. 67
Table 4.6 Detail information of time series data..... 69
Table 4.7 Comparison of training RMSE on time series data..... 71
Table 4.8 Comparison of training time (s) on time series data 71
Table 4.9 Comparison of model parameters on time series data 71
Table 4.10 Comparison of testing RMSE on time series data 72
Table 4.11 Detail information of Real life Benchmarking Regression Datasets 73
Table 4.12 Comparison of training RMSE on Real life Benchmarking Regression Datasets…… 74

Table 4.13 Comparison of training time(s) on Real life Benchmarking Regression Datasets…… 75
Table 4.14 Comparison of model parameters on Real life Benchmarking Regression Datasets…… 76
Table 4.15 Comparison of validation RMSE on Real life Benchmarking Regression Datasets…… 76

Chapter 1

Introduction

The Mine Countermeasure Missions (MCMs) is usually completed by Autonomous Underwater Vehicles (AUVs) which are capable of performing underwater tasks without an operator [51]. In this way, the operators can be kept away from exposure to potential dangers. One of the purposes of MCMs is to detect and locate Mines or Mine-Like Objects (MLOs) in specified areas of interest in order to reduce the risk for potential traffic that will pass through the region. Subsequent mine countermeasure actions will be taken based on the mine detection and location results.

Usually AUVs are equipped with side-looking sonar systems such as Side Scan Sonar (SSS) or Synthetic Aperture Sonar (SAS) [13, 20]. In most cases, sonar images produced by SAS will have higher resolution and better quality than those produced by SSS. The underwater conditions can be studied by analyzing the images produced by the sonar systems. It is worth mentioning that as a military task, there will usually be a time limit for the MCMs. The AUVs should be able to support fast (or near real time) target detection.

The introduction of machine learning methods will increase the intelligence of the AUVs and free the operators from the tedious and time consuming decision making tasks [55]. Thus, it is of great practical meaning to design and apply proper machine learning algorithms to MCMs.

As mentioned before, the MCMs have a time limit. Considering the low computational capability of the CPUs equipped on the AUVs, the speed of the designed algorithm becomes more crucial. A new machine learning method will be proposed in this work to meet the fast detection requirement of the MLOs. However, the proposed approach is not limited to MCMs application. It is also a general learning method that can

be applied to solve a wide variety of problems.

1.1 Overview

The available data in MCMs is mainly the sonar images produced by the AUVs. Considering the type of data in MCMs, the Artificial Neural Network (ANN) and kernel machines are two families of methods that could be applied to the MLOs detection task.

ANN is a method inspired by real neural systems in the human brain [5]. Similarly to biological neural networks, ANN consists of artificial neurons. In some cases, the neurons are grouped into several layers that are interconnected. Such network is called the Multi-Layer Perceptron (MLP). The neurons inside the network will be fired if certain conditions are satisfied. Many real life problems have been well solved by this approach.

Traditionally, the MLP is iteratively trained according to gradient or gradient-related information and they usually take a long time to train [50]. Moreover, such methods only guarantee the convergence to local minimum. When training with improper learning parameters, the MLP is very likely to overfit the data. There is still much improvement that can be made to the traditional MLP regarding their generalization performance.

Compared with ANN that require the optimization of a large number of weights, kernel learning is a less parameterized method [54]. However, when the size of training set (the number of training instances) is large, the kernel training and testing can both be slow. If the size of the training data becomes very large, the traditional kernel learning may even become practically infeasible.

In kernel methods, the feature mapping is usually implicit and the higher dimensional feature space can not be seen. The computation of two points in the new feature space can only be made possible by evaluating each pair of data points in the original input space. That is one reason why kernel methods can take a long time to be trained on large datasets, or even worse, learning could be virtually impossible if the size of training set is too large. Similarly, the computation between testing points and model parameters needs to be realized through pair-wise evaluation of the kernel function. The prediction speed can also be slow when the size of the training set is large. Unfortunate many real life

applications have to deal with large dataset, such as the MLOs detection task considered in this study.

1.2 Plans

In order to avoid the above mentioned problems, a new learning method is proposed under the framework of ANN. In theory, a MLP with a single hidden layer can perform universal approximation, which means that they have the ability to approximate any continuous function within any given error. Therefore, any continuous classification boundary can also be modeled. This study will focus on how to properly build and solve this kind of ANN.

It is interesting that the feature mapping idea used in kernel machines can be borrowed to construct ANN. Unlike traditional ANN learning methods which spend much time tuning all the weights slowly and tediously, the proposed approach does not tediously tune all the hidden layer parameters. More importance will be placed on how to properly calculate the output layer parameters, which should be robust to the large noise in the MCMs data. A special loss will be adopted to enhance the robustness of the network.

Similarly to sparse kernel machines, in classification, the output decision boundary will be built on only a part of the training points that lie on or near the boundary. More importantly, the classification time will be under control which is independent of the size of training dataset.

As mentioned before, for the MCMs, considering the computational capability of onboard CPUs, the speed problem of traditional ANN and kernel methods will become more serious and prominent.

It is worth mentioning that previous work has developed some random feature based models [1, 49] that look similar to the proposed approach. However the proposed approach will adjust the hidden layer parameter after seeing the data. Therefore, the hidden layer feature mapping will not be random anymore. Moreover, the proposed method comes to a sparse and robust solution, neither of which is seen in the previous

work. More differences between the proposed method and previous work will be discussed in greater length later in Chapter 3.

1.3 Contribution

Compared to traditional MLP and kernel machines, this study proposes a method that has a faster learning speed. This is the result of explicit feature mapping and sparse representation. The solution of the model parameters is simpler and more straightforward, since the higher dimensional feature space is accessible and computation can be directly conducted in the accessible space. As a result, unlike traditional kernel machines, no dual formulation is required. More importantly, when training on a very large data set, most kernel methods could be practically infeasible (or they have to resort to certain kinds of approximation methods to make them feasible). No such problem exists in the proposed approach.

Furthermore, in the proposed approach, the prediction speed is also under control since prediction is also done in the same accessible feature space. The classification time is predetermined when the ANN is constructed in the first place.

Moreover, the gap between ANN and kernel methods is bridged at some point. Traditionally, ANN and the kernel methods are considered two separate families of learning methods. The proposed approach can be viewed as the midway between ANN and kernel methods. By borrowing the idea of feature mapping from kernel machines to construct ANN, they are connected. Under the framework of ANN, the proposed classification model will have a sparse solution which will build on only parts of the training points just like SVMs. Also in classification, the proposed approach is a maximum margin classifier.

On the application side, the intelligence of AUVs will be improved, leading to the increase of efficiency of the whole MCMs. Fast autonomous decision making based on current observations will be made possible.

1.4 Organization

The remainder of the thesis is organized as follows. Chapter 2 reviews some previous research related to this study. The widely used MLP and its most commonly used training method Error Backpropagation are introduced. Furthermore, this chapter will also highlight some recent advance in ANN, a recently proposed method called extreme learning machine will be introduced. The ideas and principles of kernel methods will also be introduced in detail. Chapter 3 compares the kernel methods and ANN, and a novel classification method is proposed. The proposed learning method is applied and tested on the MCMs problem as well as some other general classification problems. Chapter 4 goes beyond classification. In this chapter, the proposed learning method is extended to regression and the proposed method is verified on both synthetic datasets as well as several real world problems. Chapter 5 summarizes the thesis. Conclusions are made and some possible future work is discussed.

Chapter 2

Related Work

Artificial neural networks and kernel machines are two widely applied machine learning methods. The proposed approach in this study borrows ideas from both artificial neural networks and kernel machines. In the chapter, some previous work related to this research will be introduced and discussed.

This chapter will begin by first reviewing the MLP and its most widely used learning method, Error Backpropagation (BP), followed by the discussion of the approximation ability of the MLP. In the second section, a recently proposed MLP-based method called Extreme Learning Machine (ELM) will be introduced and some recent advance of ELM will also be covered. Finally, the idea of kernel methods will be introduced and compared with that of the extreme learning method.

2.1 The MLP and Error Backpropagation

The MLP is an important and widely applied machine learning model. From the 1980s, much effort has been made to improve the MLP, including its computational ability, learning methodology and network structure. The MLP has shown its learning ability by effectively solving a variety of real life problems [22, 31, 35].

2.1.1 The architecture of Multilayer Perceptron

The MLP is a kind of feedforward neural networks with multiple layers that can be divided into an input layer, one or several hidden layer(s) and an output layer. The input layer is composed of passive nodes that will be fed with the input data and the output layer will produce the learning results of the whole networks. Other layer(s) that can not

be seen from neither the input nor the output side are called the hidden layer(s).

Figure 2.1 shows an example of MLP with one hidden layer, which is also called a Single hidden Layer Feedforward Network (SLFN). In this example, the output layer is composed of only one node. Traditionally, each layer will be fully connected to the next layer, which means that each node in one layer is linked to each node in the next layer via a weight connection.

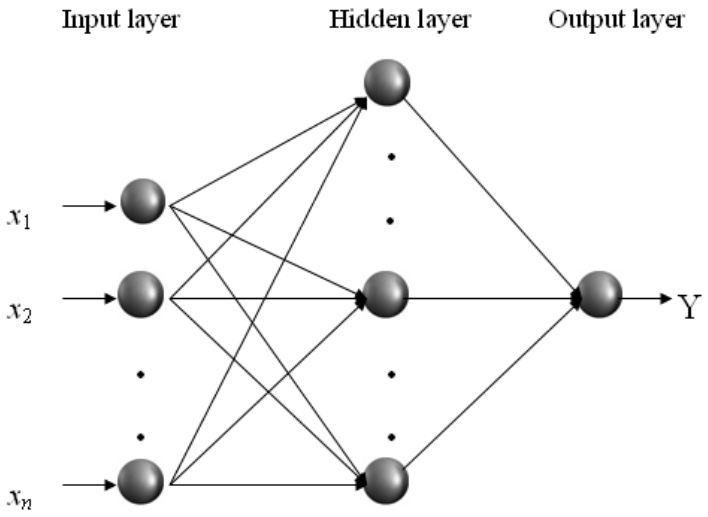


Figure 2.1 Example of the MLP

It can be found from the MLP architecture that the input layer will duplicate the data (x_1, x_2,…, x_n) and pass the information to the hidden layer(s) after actively multiplying the input weights. The outputs of the last hidden layer will be multiplied by the output weights and then be applied to the output layer. The information flow pattern in the MLP is unidirectional, so the MLP is a kind of feedforward neural networks.

The neurons in human brain either fire or do not fire. The two states can be represented by a step function, which is

$$g(x) = \begin{cases} 1, x > 0 \\ 0, otherwise \end{cases} \tag{2.1}$$

The node using the above equation as the activation function is known as the

McCulloch-Pitts neurons [44]. Unfortunately the step function is not smooth or differentiable at the point x=0, instead, the S shaped sigmoid is usually adopted as the activation function in most MLP. The definition of the sigmoid is

$$g(x) = \frac{1}{1+\exp(-x)} \tag{2.2}$$

It is noticed that $\lim_{x\to-\infty} f(x) = 0$ and $\lim_{x\to+\infty} f(x) = 1$. In many cases, we may not require the sigmoid function to pass through the origin, so a bias term, also known as the threshold, b can be added to (2.2), that is

$$g(x) = \frac{1}{1+\exp[-(x+b)]} \tag{2.3}$$

Figure 2.2 shows the sigmoid function with different bias term b. The red curve corresponds to a sigmoid with a positive b and the blue curve is a sigmoid with a negative b.

In the training process of a MLP, the bias term b can be treated as a special weight with a constant input value 1, so the bias term will have a similar learning rule as the regular weights.

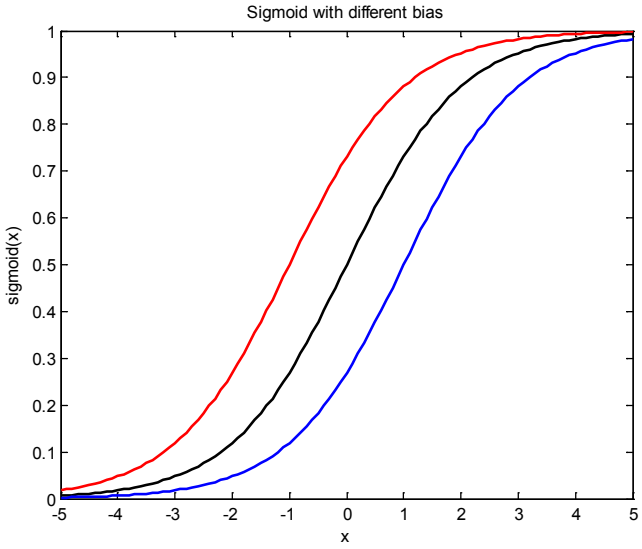


Figure 2.2 Sigmoid with different bias b

2.1.2 Error Backpropagation

Error Backpropagation is the most widely used MLP training method [11, 50]. In error Backpropagation, the weights are updated according to gradient descent, that is

$$w_{ji}^{t+1} = w_{ji}^{t} - \eta \frac{\partial E}{\partial w_{ji}} \tag{2.4}$$

where w_{ji} is the weight connecting the node j and its ith input, E is the error function which often takes a quadratic form,

$$E = \frac{1}{2}\sum_{i}(t_i - y_i)^2 \tag{2.5}$$

where t_i is the desired output and y_i is the actual network output.

Before digging further into the algorithm, let's first define some notations. For the jth node, let o_j be the output, net_j be the sum of input and $g(x)$ be the activation function, so we will have

$$net_j = \sum_{i} w_{ji} o_i, \; o_j = g(net_j) \tag{2.6}$$

Moreover, using the chain rule, the derivative of E with respect to w_{ji} can be written as

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial net_j} \times \frac{\partial net_j}{\partial w_{ji}} \tag{2.7}$$

Denoting $\delta_j = -\frac{\partial E}{\partial net_j}$ and considering (2.6), we have

$$\frac{\partial E}{\partial w_{ji}} = -\delta_j o_i \tag{2.8}$$

The second term o_i is easy to obtain by forward computation, so the key part of the Error Backpropagation is to calculate the delta term δ_i for each node. Using the chain rule again, we have

$$\begin{aligned} \delta_j &= -\frac{\partial E}{\partial net_j} = -\sum_{k} \frac{\partial E}{\partial net_k} \times \frac{\partial net_k}{\partial net_j} \\ &= -\sum_{k} \frac{\partial E}{\partial net_k} \times \frac{\partial net_k}{o_j} \times \frac{\partial o_j}{\partial net_j} \end{aligned} \tag{2.9}$$

where k denotes the nodes in the next layer. Taking into account the definition of δ and

(2.6), the above equation can be written as

$$\delta_j = \sum_k \delta_k w_{kj} g'(net_j) = g'(net_j) \sum_k \delta_k w_{kj} \tag{2.10}$$

It can be found from (2.10) that the delta term can be calculated based on the delta value in the next layer. Therefore, as long as the delta term for the output layer is available, we can backpropagate the delta value layer by layer to the whole network. Luckily, the delta term for the output layer can be easily obtained.

Considering (2.5), we have

$$\begin{aligned} \delta_j &= -\frac{\partial E}{\partial net_j} = -\frac{\partial E}{\partial y_j} \times \frac{\partial y_j}{\partial net_j} \\ &= (t_i - y_i) \times g'(net_j) \end{aligned} \tag{2.11}$$

If linear nodes are used in the output layer, the second term in the above equation will simply be 1. In this case, only the error term (t_i-y_i) will be backpropagated.

It worth pointing out that no delta term is needed for the input layer, as its only task is to copy the input data. Also, from (2.10) and (2.11), we can see that, to make backpropagation work, the activation function used in the MLP needs to be differentiable in any interval.

The learning steps of the Error Backpropagation algorithm can be summarized as follows,

Error Backpropagation Algorithm

Begin

While termination condition not satisfied

1. Randomly initialize all the network weights (bias included in the weights).
2. Given the training data $\boldsymbol{x}$, forward compute the input net_i and output o_i for each node using (2.6).
3. Calculate the delta term for all the output nodes according to (2.11)
4. Backpropagate the output layer delta term to the hidden layer(s).
5. Calculate the delta term for each hidden layer according to (2.10).
6. Update the weights according to (2.4)

end

end

There are many variations to the Error Backpropagation algorithm. A momentum term can be added to the updating formula (2.4). The momentum will take into account the direction of the previous update. With the momentum term, the algorithm is more likely to avoid local minima [46]. Moreover, it is worth pointing out that the essence of MLP training is a (nonlinear) optimization problem, so apart from gradient descent, many other standard nonlinear optimization techniques, such as conjugate gradient [36, 42], or particle swarm optimization [39], are also valid to train the MLP. We will not go into the details of these methods here.

2.1.3 Approximation ability of the MLP

From the universal approximation theorem, we know that the SLFN with one linear output layer is a universal approximator [23]. The universal approximation theorem says that given any continuous function $\varphi(\boldsymbol{x})$ on compact subset $\boldsymbol{I}^n$ of $\mathbf{R}^n$, for any $\varepsilon>0$, there exists a $f(\boldsymbol{x})$ defined as

$$f(\boldsymbol{x}) = \sum_{i=1}^{L} w_i g(\boldsymbol{a}_i^{\mathrm{T}} \boldsymbol{x} + b_i), \forall \boldsymbol{x} \in \boldsymbol{I}^n \tag{2.12}$$

such that

$$\left| f(\boldsymbol{x}) - \varphi(\boldsymbol{x}) \right| < \varepsilon \tag{2.13}$$

where $g(x)$ in (2.12) is a bounded, monotonously increasing continuous function.

It is obvious that the function defined in (2.12) can be implemented by a SLFN with one hidden layer of L nodes and one linear output node. It is noticed that the hidden activation function $g(x)$ does not need to be differentiable here, which differs from the requirement in Error Backpropagation.

2.2 Extreme Learning Machine

From the previous section, we can see that, in order to update the network weights, the Error Backpropagation method has to alternatively conduct forward computation and backward propagation of the delta term. Therefore, much time is consumed in the iterative computation and tedious weights tuning.

The Extreme Learning Machine (ELM) is a newly proposed learning algorithm based on SLFN [28]. Different from traditional neural network learning methods that learn the weights and bias through gradient or gradient-related information, ELM is a tuning-free method. Consequently, ELM enjoys a learning speed that is hundreds or even thousands of times faster than traditional Error Backpropagation. ELM has been successfully applied to many domains such as face recognition, biological system modeling and target recognition, etc. [3, 43, 64].

2.2.1 Extreme learning method

One of the most important features of ELM is its idea of randomness. In a basic ELM, hidden layer parameters, input weights and hidden biases, are randomly generated according to any continuous probability distribution, as long as the hidden layer activation function is infinitely differentiable in any interval. The hidden layer of a basic ELM is independent of the training data, and it is frozen after initialization. The output layer consists of linear nodes, and usually, no bias term is needed. The output part of the ELM is a linear model, and the output weights, the only trainable part of an ELM, is calculated analytically. Therefore, no tuning is required in the whole training process.

Given N training samples $(\boldsymbol{x}_i, t_i), \mathbf{x}_i = [x_{i1}, x_{i2}, ...x_{im}]^{\mathrm{T}} \in \mathbf{R}^m, t_i \in R$, where $\boldsymbol{x}_i$ are the attributes and t_i is the target output, similarly to (2.12), the output y of an ELM with L hidden nodes can be written as

$$y_j = \sum_{i=1}^{L} w_i g(\boldsymbol{a}_i, b_i, \boldsymbol{x}_j), \boldsymbol{a}_i \in \mathbf{R}^m, b_i \in R, j = 1, 2, ..., N \tag{2.14}$$

where $g(x)$ is the hidden layer activation function. $\boldsymbol{a}_i$ is the input weights and b_i is the

hidden bias, both randomly generated according to any continuous probability distribution, independent of the training data. The equation (2.14) can be compactly written as

$$\mathbf{H}\boldsymbol{w} = \mathbf{Y} \tag{2.15}$$

where

$$\mathbf{H}(\boldsymbol{a}_1,...,\boldsymbol{a}_L,b_1,...,b_L,\boldsymbol{x}_1,...,\boldsymbol{x}_N) = \begin{bmatrix} \phi(\boldsymbol{x}_1)^{\mathrm{T}} \\ \vdots \\ \phi(\boldsymbol{x}_N)^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} g(\boldsymbol{a}_1,b_1,\boldsymbol{x}_1)\cdots g(\boldsymbol{a}_L,b_L,\boldsymbol{x}_1) \\ \vdots \quad \cdots \quad \vdots \\ g(\boldsymbol{a}_1,b_1,\boldsymbol{x}_N)\cdots g(\boldsymbol{a}_L,b_L,\boldsymbol{x}_N) \end{bmatrix}_{N\times L} \tag{2.16}$$

$$\boldsymbol{w} = [w_1,...w_W]^{\mathrm{T}}, \mathbf{Y} = [y_1,...,y_N]^{\mathrm{T}} \tag{2.17}$$

$\phi(\bullet)$ in (2.16) is the hidden layer feature mapping. For each input sample $\boldsymbol{x}_i$, the corresponding ith row of $\mathbf{H}$ is the hidden layer feature mapping.

We assume that the true model is

$$\mathbf{T} = \mathbf{H}\boldsymbol{w} + \boldsymbol{\varepsilon} \tag{2.18}$$

where $\mathbf{T}=[t_1,t_2,\ldots,t_N]^{\mathrm{T}}$. $\boldsymbol{\varepsilon}=[\varepsilon_1, \varepsilon_2,\ldots, \varepsilon_{\mathrm{N}}]^{\mathrm{T}}$ is the residual which is usually assumed to be a zero-mean Gaussian. The problem can be solved by minimizing the quadratic loss

$$\boldsymbol{w} = \arg\min_{\boldsymbol{w}} \| \mathbf{H}\boldsymbol{w} - \mathbf{T} \|^2 \tag{2.19}$$

The only trainable part of ELM is the output weights, which can be calculated by

$$\boldsymbol{w} = \mathbf{H}^{\dagger}\mathbf{T} \tag{2.20}$$

$\mathbf{H}^{\dagger}$ is the Moore Penrose generalized inverse of matrix $\mathbf{H}$ .The solution takes the form of Least Square (LS) . If $\mathbf{H}^{\mathrm{T}}\mathbf{H}$ is non-singular, equation (2.20) is equivalent to

$$\boldsymbol{w}^{ls} = (\mathbf{H}^{\mathrm{T}}\mathbf{H})^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{T} \tag{2.21}$$

All in all, the learning steps of an extreme learning machine can be summarized as follows,

Extreme Learning Machine

Given N distinct training samples $(\boldsymbol{x}_i, t_i), \boldsymbol{x}_i \in \mathbf{R}^m, t_i \in R$, i=1,2,…,N, L hidden nodes with activation function $g(x)$,

Begin

1. Randomly assign the input weights $\boldsymbol{a}_i$ and hidden bias b_i, i=1,2,…,L.

2. Calculate the matrix **H** according to (2.16).

3. Calculate the output weights according to (2.20).

end

It is worth mentioning that only the multiple inputs/single output case of ELM is introduced here. Extreme learning methods can be easily extended to SLFN with multiple outputs. In addition, the hidden layer activation function $g(x)$ can also be a neural network, so extreme learning is not limited to MLP with only one hidden layer.

2.2.2 Approximation ability of ELM

For section 2.1.3, we know than SLFN with linear output nodes can work as a universal approximator. The ELM shares exactly the same architecture as SLFN, but with random hidden layer parameters, input weights and hidden bias. It is proven that ELM is also able to performance universal approximation.

According to the ELM learning theory [26], given any bounded non-constant piecewise continuous function g: $R \rightarrow R$, for any continuous target function f(x) and any randomly generated function sequence $\{g_L=\mathrm{g}(\boldsymbol{a}_L, b_L, \boldsymbol{x})$, the follow equation

$$\lim_{L\to+\infty} \| f(x) - f_L(x) \| = \lim_{L\to+\infty} \| f(x) - \sum_{i=1}^{L} w_i g(x) \| = 0 \tag{2.22}$$

will hold with probability one if the output weight sequence w_i is calculated via

$$w_L = \frac{< e_{L-1}, g_L >}{\| g_L \|^2} \tag{2.23}$$

where $e_{L\text{-}1}= f(x)\text{-}f_{L\text{-}1}(x)$, is the residual for the network with L-1 random nodes. Therefore, the ELM is able to perform universal approximation. In the above theorem, the randomly generated function sequence $\{g_i=\mathrm{g}(\boldsymbol{a}_i, b_i, \boldsymbol{x})\}$ means that the hidden parameters $(\boldsymbol{a}_i, b_i), \boldsymbol{a}_i \in \mathbf{R}^m, b_i \in R$, are randomly generated according to any continuous probability distribution.

However, in real life applications, we are seldom given an actual function to approximate. Usually, what we are given is a certain amount of sample data. In such cases, the number of hidden nodes will never grow to infinity as shown in (2.22).

It is proven that given N distinct samples $(\boldsymbol{x}_i, t_i), \mathbf{x}_i \in \mathbf{R}^m, t_i \in R$ and any ε>0, the ELM requires no more than N random hidden nodes (random $\boldsymbol{a}$ and b), to learn the training data within the given ε [28]. The input weights $\boldsymbol{a}$ and hidden bias b can be randomly assigned according to any continuous probability distribution as long as the hidden activation function $g(x)$ is infinitely differentiable in any interval.

It is worth pointing out that the requirement of hidden activation function is stronger than that of the previous case. Here $g(x)$ needs to be infinitely differentiable, while $g(x)$ used in (2.22) only has to be bounded non-constant piecewise continuous.

2.2.3 Discussions and recent advance

Most recent advance of extreme learning focused on how to properly calculate the output weights and optimize the number of hidden nodes [45, 63]. Regularization methods such as Lasso and Bayesian linear regression have been used to replace the least solution of the output weights [53, 56]. Optimizing the number of hidden nodes is, in essence, a selection for the hidden layer feature mapping $\phi(x)$ in (2.16). However, as a result of random hidden nodes, the mapping will always be random. The mapping will be nonlinear as long as a nonlinear g(x) is used.

It is worth mentioning that the random mapping idea is not a monopolization of extreme learning, there are other methods based on similar idea, such as random projection based feedforward neural network [61] and Echoes State Networks [32, 33].

It has been observed from experiments that, typically, ELM requires a larger hidden layer than networks train by Error Backpropagation. One important difference between ELM and traditional SFLN is the role of the hidden layer. In traditional SLFNs, the hidden layers are used to extract features, but for ELM, in virtue of the random hidden parameters, the hidden layer has no feature extraction ability. The only purposed of the ELM hidden layer is to randomly project the input data to another feature space, usually a

higher dimensional space, where ELM seeks to compute a linear model. This idea shares some similarity with another family of learning methods, which is called kernel machines.

2.3 Kernel methods

The kernel method [54] is a class of machine learning algorithm that has been successfully applied to solve a wide variety of problems. Kernel machines usually follow two steps. The first step will project the data from the input space to another feature space. The second step is to learn a model in the feature space using optimization techniques. In most cases, for simplicity, the model in the feature space will be linear.

Given two data points $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ and a fixed mapping function $\phi(x)$, the kernel function can be defined as

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \phi(\boldsymbol{x}_1)^{\mathrm{T}} \phi(\boldsymbol{x}_2) \tag{2.24}$$

It can be found that with a kernel function, the feature space computation can be realized directly with the original data vectors $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. With such a representation, the original mapping function $\phi(x)$ does not need to be known to the users.

There are also kernels that are defined based on the Euclidian distance between two vectors, such as radial basis functions [8],

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = k(\| \boldsymbol{x}_1 - \boldsymbol{x}_2 \|) \tag{2.25}$$

It is observed that all kernel functions are symmetric that is $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = k(\boldsymbol{x}_2, \boldsymbol{x}_1)$. This feature guarantees the semi-positive definite property of the Gram matrix which can be defined by $\mathbf{K}_{ij}=k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. This property is one necessary condition for the optimality of the solution of Support Vector Machines that will be introduced later.

The most commonly used kernels include the linear kernel, polynomial kernel, Gaussian kernel and sigmoid kernel.

2.3.1 Support Vector Machines

Support vector machines (SVMs) might be the most famous method among all kernel machines [12, 60]. Different from ANN where the solution is a set of weights and all the training data will be discarded during the prediction, SVMs will keep a part of the training data, known as the support vectors, and use some coefficients associated with these points to make predictions.

The decision function of SVMs is a simple linear model that can be expressed as

$$y(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}) + b \tag{2.26}$$

Instance $\boldsymbol{x}$ is predicted to the negative class if $y(\boldsymbol{x})<0$, otherwise, it will be predicted to be a member of the positive class. Now taking into account the class label $t \in \{-1,1\}$, we will have $y(\boldsymbol{x})>0$ if $t=1$ or $y(\boldsymbol{x})<0$ if $t=-1$, so

$$ty(x) = t[\boldsymbol{w}^{\mathrm{T}}\phi(\boldsymbol{x}) + b] = \left|y(\boldsymbol{x})\right| \tag{2.27}$$

Dividing the above equation by $||\boldsymbol{w}||$, we get

$$\delta = \frac{\left|y(\boldsymbol{x})\right|}{||\boldsymbol{w}||} \tag{2.28}$$

From analytic geometry, we know that δ is the Euclidian distance between the point $\boldsymbol{x}$ and the hyper-plane $y(\boldsymbol{x}) =0$.

Now denoting the positive examples with subscript + and the negative examples with subscript -, with proper scaling of $\boldsymbol{w}$ and b, we can set the positive points that have minimum distance to the decision hyper plane on $y(\boldsymbol{x}_+) =1$ and the negative points on $y(\boldsymbol{x}_-) =-1$. Therefore, there will be no other points between $\boldsymbol{x}_+$ and $\boldsymbol{x}_-$. The space between $\boldsymbol{x}_+$ and $\boldsymbol{x}_-$ is the known as the separating margin.

The width of the separating margin can be expressed as

$$\delta_+ + \delta_- = \frac{|\, y(\boldsymbol{x}_+)\,|}{||\boldsymbol{w}||} + \frac{|\, y(\boldsymbol{x}_-)\,|}{||\boldsymbol{w}||} = \frac{2}{||\boldsymbol{w}||} \tag{2.29}$$

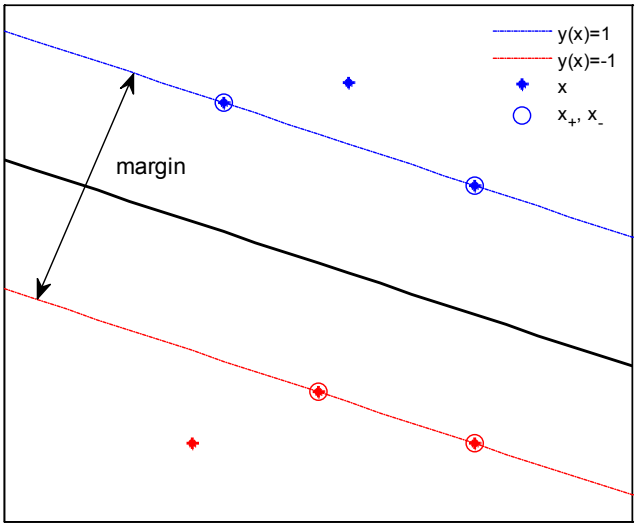


Figure 2.3 Separating margin

Figure 2.3 illustrates the separating margin with a 2-dimensional example. The two classes of data points are shown in red and blue. The red dot-dash line (hyper-plane in a higher dimensional space) is $y(\boldsymbol{x})$ =1 and the red dot-dash line on the other side is $y(\boldsymbol{x})$ =-1. The black line in the middle represents $y(\boldsymbol{x})$ =0. The data points $\boldsymbol{x}_+$ and $\boldsymbol{x}_-$ are shown in the circles.

SVM is a maximum margin classifier, which means that it will result in a decision boundary that maximizes the separating space, shown by (2.29), between the positive and negative examples. Therefore, in order to maximize the margin, we have to minimize $||\boldsymbol{w}||$, which is equivalent to minimizing $||\boldsymbol{w}||^2/2$.

In real applications, most datasets have a certain amount of noise, so sometimes we may allow some data points to fall into the margin or even on the wrong side of the margin. SVM is able to deal with such soft margin, but in such cases, an additional cost will be assigned for allowing such points. To deal with such points, a non-negative slackness variable can be introduced,

$$t_i y(x_i) \geq 1-\xi_i, \xi_i \geq 0, i=1,2,\ldots,N \tag{2.30}$$

The Primal objective function of SVMs can be written as

$$\min L_P(\boldsymbol{w},b,\xi) = \frac{1}{2}||\boldsymbol{w}||^2 + C\sum_{i=1}^{N}\xi_i, i=1,2,\ldots,N \tag{2.31}$$

where the second term is the penalty for the data points that lie on the wrong side or inside the margin, and C is a trade-off parameter adjusting the penalty.

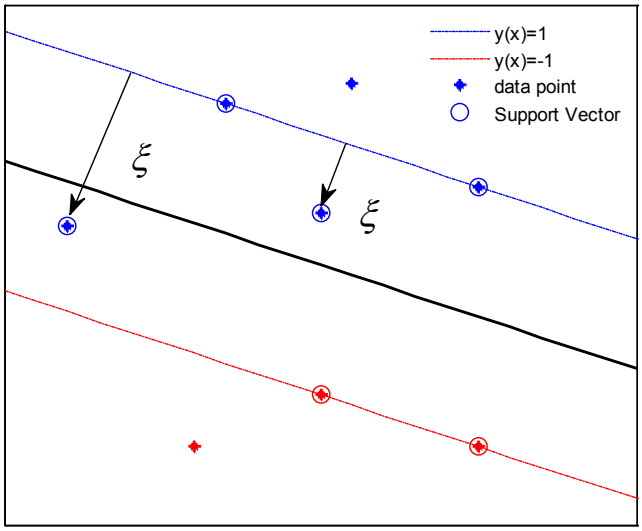


Figure 2.4 Soft margin

Figure 2.4 is an illustration of the soft margin. It will be later found that the final decision boundary of SVMs will only build on data points with a positive ξ and those lying exactly on the margin. Such points, shown by circles, are called support vectors.

With the Lagrangian representation, the constrained optimization (2.30) and (2.31) can be transformed into,

$$L(\boldsymbol{w},b,\xi,\alpha,\beta)=\frac{1}{2}\|\boldsymbol{w}\|^2+C\sum_{i=1}^{N}\xi_i-\sum_{i=1}^{N}\alpha_i[t_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i)+b)-1+\xi_i]-\sum_{i=1}^{N}\beta_i\xi_i, \quad \alpha_i\geq 0,\beta_i\geq 0,i=1,2,\ldots,N \tag{2.32}$$

where α_i, β_i are Lagrange multipliers. The Karush-Kuhn-Tucker (KKT) conditions of the above problems are [37, 41]

Primal feasibility,

$$t_i[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i)+b]-1+\xi_i\geq 0,i=1,2,\ldots,N \tag{2.33}$$

$$\xi_i\geq 0,i=1,2,\ldots,N \tag{2.34}$$

Dual feasibility,

$$\alpha_i\geq 0,\beta_i\geq 0,i=1,2,\ldots,N \tag{2.35}$$

Complementary slackness,

$$\alpha_i[t_i(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i)+b-1+\xi_i]=0, i=1,2,\ldots,N \tag{2.36}$$

$$\beta_i\xi_i=0, i=1,2,\ldots,N \tag{2.37}$$

Moreover, setting the derivative of L with respect to the primal variables $\boldsymbol{w}$, b and ξ to zero, we will get

$$\frac{\partial L}{\partial \boldsymbol{w}}=0\Rightarrow \boldsymbol{w}=\sum_{i=1}^{N}\alpha_i t_i\boldsymbol{\phi}(\boldsymbol{x}_i) \tag{2.38}$$

$$\frac{\partial L}{\partial b}=0\Rightarrow \sum_{i=1}^{N}\alpha_i t_i=0 \tag{2.39}$$

$$\frac{\partial L}{\partial \xi_i}=0\Rightarrow \alpha_i+\beta_i=C, i=1,2,\ldots,N \tag{2.40}$$

Considering (2.36), (2.37) and (2.39), and replacing $\boldsymbol{w}$ and C with (2.38) and (2.40), respectively, (2.32) can be turned into the Dual form that is

$$\begin{aligned}&\min L_D(\boldsymbol{\alpha})=\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j t_i t_j k(\boldsymbol{x}_i,\boldsymbol{x}_j)-\sum_{k=1}^{N}\alpha_k\\&s.t.\ 0\le\alpha_i\le C,\sum_{i=1}^{N}\alpha_i t_i=0.\end{aligned} \tag{2.41}$$

where $k(\boldsymbol{x}_i,\boldsymbol{x}_j)$ is the kernel function defined by (2.24). The above objective function can also be compactly written as

$$\begin{aligned}&L_D(\boldsymbol{\alpha})=\frac{1}{2}\boldsymbol{\alpha}^{\mathrm{T}}\Omega\boldsymbol{\alpha}-\boldsymbol{\alpha}^{\mathrm{T}}\vec{1}\\&s.t.\ 0\le\alpha_i\le C,\boldsymbol{\alpha}^{\mathrm{T}}\mathbf{T}=0, i=1,2,\ldots,N\end{aligned} \tag{2.42}$$

where $\boldsymbol{\alpha}=[\alpha_1,\ldots,\alpha_N]^{\mathrm{T}}$, $\mathbf{T}=[t_1,...,t_N]^{\mathrm{T}}$, $\vec{1}=[1,\ldots 1]^{\mathrm{T}}_{1\times N}$ and $\Omega_{ij}=t_i t_j k(\boldsymbol{x}_i,\boldsymbol{x}_j)$. The size of matrix Ω is N by N.

SVM is a sparse learning method. It is found from complementary slackness (2.36) that if the constraint (2.33) is not tight, the corresponding α_i will be forced to zero. Removing such data point will not affect the final solution of SVMs at all. Only support vectors, shown by circles in figure 2.4, will result in a non-zero α_i and appear in the decision function of SVMs during the prediction.

For the support vectors that lie exactly on the margin, the corresponding ξ_i=0,

and $0 \le \alpha_i < C$. For other support vectors, α_i will be set equal to C.

Given a new input $\boldsymbol{x}^*$ in prediction, the decision function of SVMs is

$$sgn(\sum_{SVs} \alpha_i k(\boldsymbol{x}_i, \boldsymbol{x}^*) + b) \tag{2.42}$$

where $sgn(x)$ is the signum function that is defined by

$$sgn(x) = \begin{cases} 1, & x > 0, \\ 0, & x = 0, \\ -1, & x < 0. \end{cases} \tag{2.43}$$

It is interesting that the function $sgn(x)$ will be 0 if x=0. From (2.42), SVMs seem unable to classify such instances. However in real applications, there is little chance that a data point will lie exactly on the decision (hyper) plane.

It is worth noting that only the classification model of SVMs is introduced here. SVMs can also be applied to solve regression problems. In such cases, the method is called SVR (Support Vector Regression).

Apart from SVMs, there are many other methods that are built on kernel learning, such as kernel PCA (Principle Components Analysis), kernel LR (Logistic Regression) and RBF networks, etc [38, 47, 52]. All these kernel methods will use kernel functions to implement the mapping $\phi(x)$ in (2.24). The details of such kernel methods will not be discussed here.

2.4 Summary and Discussions

In this chapter, the architecture of the MLP and the most widely used learning method, Error Backpropagation is introduced. Furthermore, the recently proposed extreme learning approach is presented and the universal approximation ability of both MLP and Extreme Learning Machine are stated and explained. In section 2.3, the idea of kernel learning and one of its most well known methods, Support Vector Machines is reviewed and discussed.

Extreme learning is not limited to traditional SLFN, it also works for generalized SLFN where the hidden layer is not required to be actually built. In such cases, kernel

learning can be incorporated into ELM to form the kernalized ELM [27]. Moreover, when certain conditions are satisfied, if the number of hidden nodes for SLFN and ELM increase to infinity, they will be equivalent to a kernel method using a special defined neural network kernel function [15, 48, 62].

In kernel methods, the inner product of higher dimensional space is converted to the pair-wise evaluation of the kernel function with the data points directly in the input space. However, as a trade-off, they always have to deal with the matrix Ω or Gram (kernel) matrix whose size is N square. When the size of the data becomes very large, the kernel methods may become practically infeasible. It is common that with parameter optimization process, SVMs can take more than hours to train on a dataset whose size is only in the thousands or tens of thousands. Many later works focus on how to solve this problem. The most common choice is low-rank approximation of the Gram matrix through certain decomposition or sampling methods [1, 14]. However, on very large datasets, the resulting rank could still be large. The result based on low-rank approximation may not be an optimal solution for the original problem, which goes against the original design of SVMs that an optimum is guaranteed. The details will not be discussed here.

In the proposed method that will be introduced in the next chapter, while a solution for very large dataset is sought, no kernel will be used and the problem just mentioned will be avoided. The proposed method will lead to a sparse classification model like SVMs, but it is should not be viewed as another speed-up version of SVMs, nor does it use any kind of approximation for the Gram matrix.

Chapter 3

Proposed Approach

It is known that a nonlinear decision boundary can be approximated by a linear hyper-plane in another space, usually with a higher dimensionality. Therefore, via a nonlinear feature mapping, the nonlinear boundary calculation can be transformed into a linear hyper-plane which is easier to calculate.

In kernel machines, the higher dimensional feature space does not need to be known to the users. Despite the high dimensional space, the curse of dimensionality in kernel machines can be avoided by transferring the computation in the high dimensional feature space to the originally input space. Thus, the dimensionality of the feature space, denoted by D, can be very high (or even infinity, for example where a Gaussian kernel is used) [57]. In kernel machines, the feature mapping is implicit. We don't, neither do we need to, have access to the higher dimensional feature space.

However, kernel methods are not the only way to perform the feature mapping. In some cases, the classification problem can be successfully solved in a feature space, D that is not too large. Directly solving the classifier in such a feature space becomes computationally feasible and, sometimes, even more efficient. In such cases, explicit feature mapping will be an alternative.

By doing the explicit feature projection, if D is not too large, the learning and classification could be speeded up. In kernel machines, the higher dimensional computation is realized by pair-wise evaluation of the kernel function on the N data points. Consequently, an $N \times N$ sized Gram matrix has to be built. It will become a problem when the size of the training set N is very large and for non-sparse kernel machines, N coefficients (the threshold is ignored here) have to be solved to represent the final decision boundary. If we alternatively perform explicit feature mapping, no such Gram matrix is necessary and any classification boundary in the space can be expressed

with D coefficients. Furthermore, the classification is directly done in the accessible feature space rather than as a comparison to some training points which could be very slow.

When doing explicit feature mapping, we have to limit the value of D, due to the curse of dimensionality, while in kernel methods, no such problem exists. As mentioned previously, kernel methods are able to work in a space where D can be very high or even infinity. By switching from kernel to explicit feature mapping, we could get some improvement in speed, but in return, the performance of the classifier may be compromised due to the limitation to D. However, in some cases, when the feature space created by explicit mapping is effective enough to solve the problem, no drop in performance will be experienced and doing explicit mapping will be a wise choice. Also, under certain circumstances, when the learning or classification speed is a major concern, like the fast MLOs detection considered in this research, the proposed approach will show its advantage.

The MLP can be viewed as a case of explicit feature mapping. Therefore, we can perform the mapping via building a MLP. The feature space will be defined by the parameters and activation function of the MLP hidden layer(s). As mentioned before, MLP with one hidden layer, also known as Single hidden Layer Feedforward Neural networks (SLFNs), is already a universal approximator, therefore, we will only focus on how to build such networks. However, the proposed approach is not limited to such networks. It can also be extended to the MLP with multiple hidden layers.

Training the SLFNs is in essence solving for a set of weights given the training data. Thus, different from traditional methods that focus only on the training error, the proposed approach will take into account the solution, the weights, itself in the SLFNs training process. The proposed method will result in a maximum margin classifier, which maximizes the separation space between the positive class and the negative class. Furthermore, a sparse and robust solution will be obtained. The decision boundary will only be built on a part of the training points, which will speed up the training process.

It was seen in the extreme learning approach, the hidden layer parameters can be independently and randomly assigned without any adjustment. However, we find that it is

unreasonable to assume independence between the input weights and the data. The hidden layer parameters used in the proposed approach will therefore be adjusted after seeing the data, so a relationship will be built between these parameters and the training data.

This chapter will be organized as follow. In the proposed approach, we will first discuss the relationship between the generalization ability of SLFNs and its weight norms and we will see that the norm should not be fixed to a random number as in the extreme learning approach. Furthermore, based on the discussion, we will propose a SVM-like model with explicit SLFNs feature mapping. The proposed approach will improve the prediction performance of traditional SLFNs. Also the resulting model will have a much faster speed, both in training and testing, than kernel methods. The experimental results will be shown in 3.3. Section 3.4 is the conclusion.

3.1 The norm of SLFNs weights

The generalization performance of SLFNs is closely related to the norm of the weights [4]. However, in the extreme learning approach, the input weights are random and independent of the data, and the norm of the input weights is ignored. As we know, the input data will always multiply the input weights before passing through the hidden layer(s), so it might be unreasonable to assume independence between the input weights and training data.

Consider the sigmoid neurons, the most widely used neuron in the MLP, for the same training data, if the norm of the input weights is small, the sigmoid will work in the nearly linear region, resulting in an output function, or decision boundary in classification, that is close to linear in the input space. On the contrary, if the norm of the input weights is large, the sigmoid node will be more non-linear. The output function will gain more complexity and non-linearity. If the weights keep growing, the sigmoid function will be saturated, and it will produce an output either too close to 0 or 1. In such a case, any variation or noise in the input data will be magnified by the weights and lead to a great variation in the network output. Such SLFNs are very likely to memorize and overfit,

since any kind of training error will be heavily magnified.

Similarly for the output layer, large output weights are also undesirable as the hidden layer will multiply the output weights to produce the final network outputs. In addition, when using a linear output layer, minimizing the norm of output weights is equivalent to maximizing the classification margin [25].

In many cases, when reaching a small training error, a SLFN with smaller weights is more likely to generalize well. That is one reason why initial weights of the MLP trained by BP are usually set to be small. In such cases, the BP algorithm is more likely to converge to a nearby local minimum where the norm of weights is small [2].

From the perspective of the Bias-Variance Tradeoff in statistical learning, the prediction error can be decomposed into two parts [19]

$$\begin{aligned} E\{(y_i - f_i)^2\} &= E\left\{(y_i - E\{f_i\})^2\right\} + E\{(E\{f_i\} - f_i)^2\} \\ &= Bias^2(f_i) + Var(f_i) \end{aligned} \tag{3.1}$$

where y_i is the true model and f_i is the predicted output. The Bias term in (3.1) measures the degree of approximation of the network to the target function, while the variance term measures the sensitivity of the model to the training samples. Usually a model with large weights will have a large variance but a small bias, as the red line in figure 3.3 shows. On the contrary, a model with a small variance tends to have a large bias. The best model should strike a balance between these two.

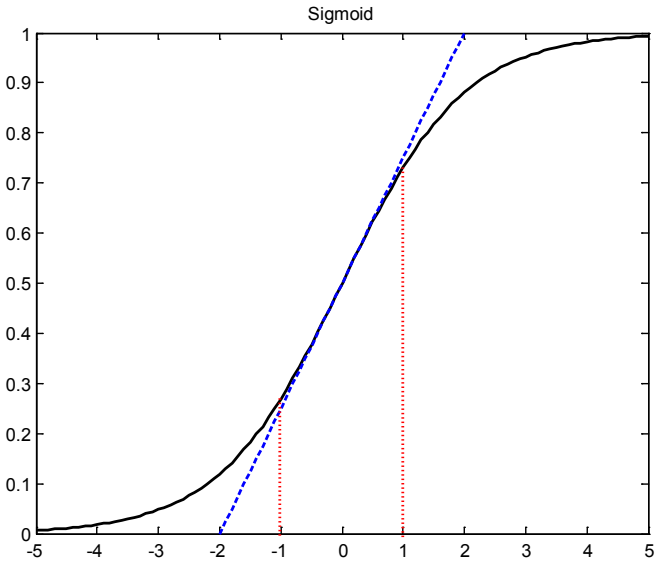


Figure 3.1 Sigmoid in linear working region

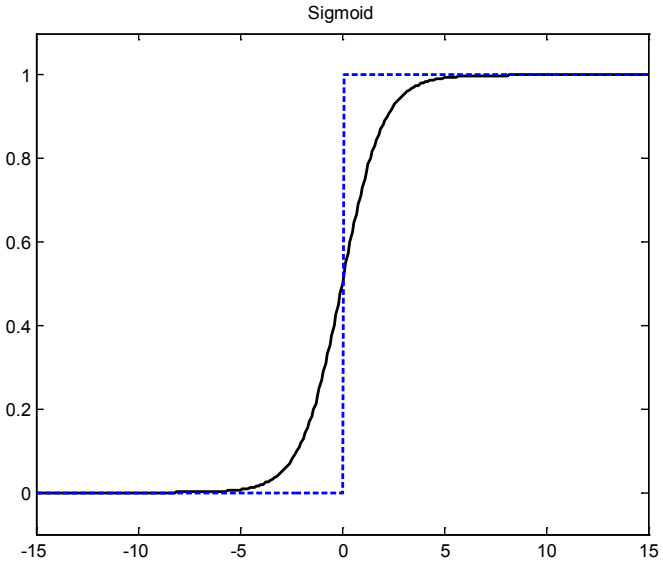


Figure 3.2 Sigmoid in nonlinear working region

Figure 3.1 and Figure 3.2 show the different working region of the sigmoid. Figure 3.3 shows a two dimensional example of a classification problem. The two classes are represented by the white circles and black squares. When the norm of the input weights is small, the sigmoid function can be approximated by the blue linear line in Figure 3.1. The decision boundary modeled by SLFN will be like the simple blue line in Figure 3.3. On the contrary, if the norm is large, the sigmoid will be more non-linear. Consequently, the SLFN will produce a waving decision boundary that is more non-linear (red line). In real applications, there will always be a certain amount of noise (or mislabel data points) in the dataset. Therefore, it is reasonable to allow for certain points lying on the wrong side. Strictly classifying every data right will encourage the network to memorize rather than to learn.

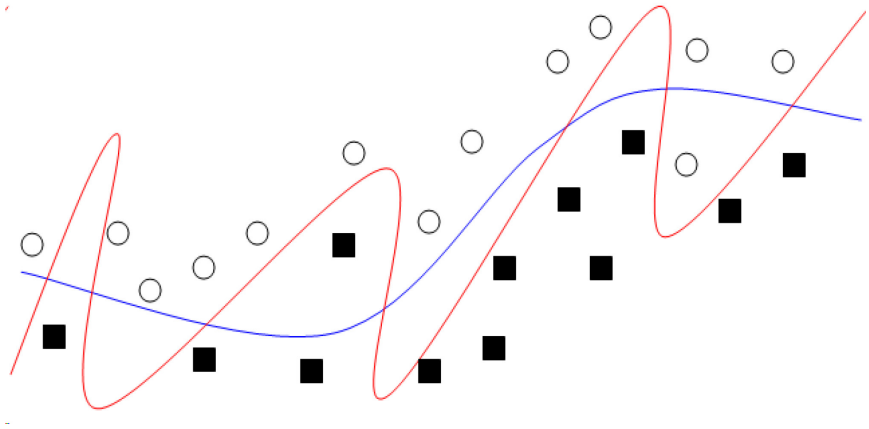


Figure 3.3 Different decision boundaries

Unfortunately in many traditional SLFN training methods, much effort is made on how to approximate the training points or how to reach the minimum of the error surface. The solution, the set of weights, is ignored.

3.2 The proposed approach

As discussed in the previous section, the norm of the weights is important for SLFNs. In order to avoid large weights, we include the weights in the objective function and decay the norm of the weights in training. Traditional SLFN includes almost every data points, either correctly or wrongly classified, in the error term. It seems unreasonable to penalize the data points that lie on the right side of the decision boundary. Therefore, similarly to SVMs, we only put penalties on the data points that lie inside the margin or on the wrong side of the decision boundary.

The training process of the proposed approach is mainly about the SLFN input weights which define the high dimensional feature space, and the output weights which define the final decision boundary. The training strategy for such parameters will be derived in the following subsections.

In this research, we will only focus on binary classification. In such a case, only one linear output node is needed in the SLFN output layer.

3.2.1 SLFN with Weight Decay

Given N training instances $(\boldsymbol{x}_i, t_i)$, $\boldsymbol{x}_i \in \mathbf{R}^m$, $t_i \in (-1, 1)$, $(i=1,2,\ldots,N)$, where $\boldsymbol{x}_i$ are the attributes and t_i is the class label, when using a linear output neuron, the output y of a SLFN with L hidden nodes can be written as

$$y_j = \sum_{i=1}^{L} w_i g(\boldsymbol{a}_i, \boldsymbol{x}_j),\ j = 1,2,...,N \tag{3.2}$$

where $g(x)$ is the hidden layer activation function, which is set to be sigmoid in this study. $\boldsymbol{a}_i$ is the input weights (the hidden bias can be included in the input weights).

In order to control the norm of the weights, a weight constraint can be applied to the SLFNs, so the objective function is to minimize the training error, while the weights are under constraints. The objective function can be written as

$$\begin{aligned} &\min L(\boldsymbol{w}, \boldsymbol{a}) = \frac{1}{2}\sum_{i=1}^{N} E(\xi_i) \\ &s.t. \frac{1}{2}\sum_{i=1}^{L} w_i^2 \le s_1, \frac{1}{2}\sum_{i=1}^{L}\sum_{j=1}^{M} a_{ij}^2 \le s_2, t_i y(\boldsymbol{x}_i) \ge 1 - \xi_i, i = 1,2,...,N \end{aligned} \tag{3.3}$$

where $\boldsymbol{w}=[w_1, w_2,\ldots,w_L]^{\mathrm{T}}$ is the output weight and a_{ij} is the input weight, M is the number of input neurons, s_1 and s_2 are two constants.

As mentioned before, similarly to SVMs, if $\boldsymbol{x}_i$ is picked to be a support vector in the final solution, the corresponding inequality constraint $\boldsymbol{t}_i y(\boldsymbol{x}_i) \ge 1-\xi_i$ will be tight and only support vectors that lie inside the margin ($0<\xi_i<1$) and those on the wrong side of the decision function ($\xi_i \ge 1$) will be penalized. Therefore we define an objective function that penalizes these points, equation (3.3) can be equivalently written as

$$\begin{aligned} &\min L(\boldsymbol{w}, \boldsymbol{a}) = \frac{1}{2}[\sum_{i=1}^{N} E(\xi_i) + \lambda_1 \sum_{i=1}^{L} w_i^2 + \lambda_2 \sum_{i=1}^{L}\sum_{j=1}^{M} a_{ij}^2] + const. \\ &subject\ to\ t_i y(\boldsymbol{x}_i) = 1 - \xi_i, i = 1,2,...,N \end{aligned} \tag{3.4}$$

where λ_1 and λ_2 are the tradeoff parameters. In our specific case, considering the poor quality of our side scan sonar images and the large noise, the error function is chosen to be a robust Huber-like function [29], which is defined as

$$E(\xi)=\begin{cases} u\xi-0.5u^2, & \xi>u \\ 0.5\xi^2, & 0\le\xi\le u \\ 0, & \xi<0 \end{cases} \tag{3.5}$$

Furthermore, we define the matrix **H**

$$\mathbf{H}(\boldsymbol{a}_1,...,\boldsymbol{a}_L,\boldsymbol{x}_1,...,\boldsymbol{x}_N)=\begin{bmatrix} \phi(\boldsymbol{a},\boldsymbol{x}_1)^{\mathrm{T}} \\ \vdots \\ \phi(\boldsymbol{a},\boldsymbol{x}_N)^{\mathrm{T}} \end{bmatrix}=\begin{bmatrix} g(\boldsymbol{a}_1,\boldsymbol{x}_1)\cdots g(\boldsymbol{a}_L,\boldsymbol{x}_1) \\ \vdots \quad \cdots \quad \vdots \\ g(\boldsymbol{a}_1,\boldsymbol{x}_N)\cdots g(\boldsymbol{a}_L,\boldsymbol{x}_N) \end{bmatrix}_{N\times L} \tag{3.6}$$

$\phi(\boldsymbol{a},\boldsymbol{x})$ in (3.6) is the hidden layer feature mapping. Let $\boldsymbol{y}=[y_1,\ldots,y_n]^{\mathrm{T}}$, therefore, equation (3.2) can be compactly written as

$$\boldsymbol{y}=\mathbf{H}\boldsymbol{w} \tag{3.7}$$

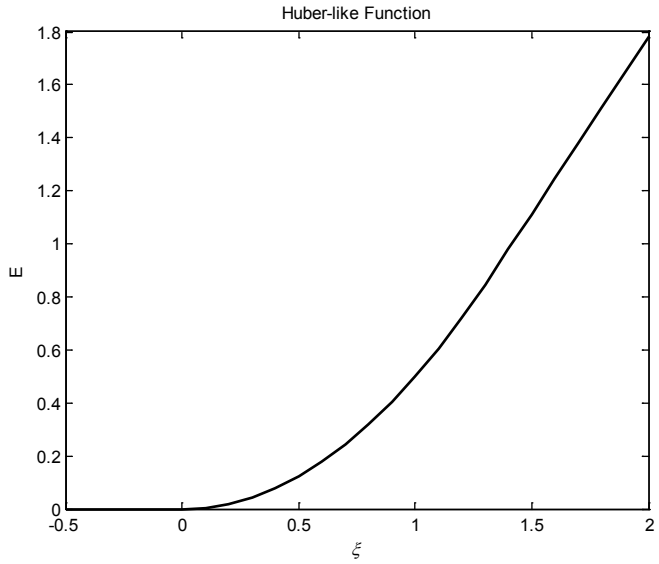


Figure 3.4 Huber-like function

Figure 3.4 shows the Huber-like function defined by (3.5). It is a function that is continuous and differentiable at any interval. Data points correctly classified and lying outside the margin (ξ_i<0) will not be penalized. When ξ_i is positive and small, the error function takes a quadratic form, and when ξ_i becomes large, the error function only increases linearly. Thus, compared with the quadratic loss, it is more robust. It also has the advantage of differentiability (at any interval) over the commonly used hinge loss.

3.2.2 Input weights training

Similarly to BP, we can use the chain rule to propagate $L(\boldsymbol{w},\boldsymbol{a})$, defined by (3.4), back to the input layer. Taking the derivative of L with respect to a_{ij}, we have

$$\frac{\partial L}{\partial a_{ij}} = \frac{\partial E}{\partial a_{ij}} + \lambda_1 \frac{\partial \| \boldsymbol{w} \|^2}{\partial a_{ij}} + \lambda_2 a_{ij} \tag{3.8}$$

For simplicity, we assume $\|\boldsymbol{w}\|^2$ to be a constant so that $\frac{\partial \| \boldsymbol{w} \|^2}{\partial a_{ij}} = 0$, Now in (3.8) the only unknown term is $\frac{\partial E}{\partial a_{ij}}$.Let net_i be the input of neuron i. Denoting the hidden node and output node by superscript h and o, we have,

$$net_i = \begin{cases} net_i^h = \sum_j a_{ij} x_{ij} = \boldsymbol{a}_i^{\mathrm{T}} \boldsymbol{x} \\ net_i^o = ty(x) = t \times \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}) \end{cases} \tag{3.9}$$

where t is the class label, and $y(x)$ is the network output. Using the chain rule, we have

$$\begin{aligned} \frac{\partial E}{\partial a_{ij}} &= \frac{\partial E}{\partial net_i^h} \times \frac{\partial net_i^h}{\partial a_{ij}} \\ &= \frac{\partial E}{\partial net_i^h} \times x_j \end{aligned} \tag{3.10}$$

where x_{ij} is the jth attribute value of the input data. Furthermore,

$$\frac{\partial E}{\partial net_i^h} = \sum_{j \in \text{output layer}} \frac{\partial E}{\partial net_j^o} \times \frac{\partial net_j^o}{\partial net_i^h} \tag{3.11}$$

Since only one output node will be used in binary classification, equation (3.11) can be written as

$$\frac{\partial E}{\partial net_i^h} = -\delta^o \times \frac{\partial net^o}{\partial net_i^h} \tag{3.12}$$

where the delta term for the output node is defined as $\delta_j^o = -\frac{\partial E}{\partial net_j}$.Denoting o_i the output of node i, equation (3.12) can be written as

$$\begin{aligned}\frac{\partial E}{\partial net_i^h} &= -\delta^o \times \frac{\partial net^o}{\partial net_j^h} \\ &= -\delta^o \times \frac{\partial net^o}{\partial o_i} \times \frac{\partial o_i}{\partial net_i^h} \\ &= -\delta^o \times w_i \times o_i(1-o_i)\end{aligned} \tag{3.13}$$

It is worth mentioning that for the first order derivative of sigmoid function, $\frac{\partial f(x)}{\partial x} = f(x)(1-f(x))$.This property is used in the above equation. Furthermore,

$$\delta_i^h = -\frac{\partial E}{\partial net_i^h} = \delta^o \times w_i \times o_i(1-o_i) \tag{3.14}$$

Since $E(\xi)$ is a Huber-like function, we have

$$\begin{aligned}\delta^o &= (1-t_i y(x_i)) \times \mathrm{W}_i \\ \mathrm{W}_i &= \begin{cases} u/\xi_i, \xi_i > u \\ 1 \quad , 0 \le \xi_i \le u \\ 0 \quad , \xi_i < 0 \end{cases}\end{aligned} \tag{3.15}$$

Considering (3.10), (3.14) and (3.15), equation (3.8) becomes

$$\begin{aligned}\frac{\partial L}{\partial a_{ij}} &= \frac{\partial E}{\partial a_{ij}} + \lambda_2 a_{ij} \\ &= -\delta_i^h x_j + \lambda_2 a_{ij}\end{aligned} \tag{3.16}$$

Therefore, the learning rule for the input weights is

$$a_{ij} = a_{ij} + \Delta a_{ij}, \Delta a_{ij} = -\eta \frac{\partial E}{\partial a_{ij}} = \eta(\delta_j^h x_i - \lambda_2 a_{ij}) \tag{3.17}$$

where η is a constant.

It can be seen from (3.17) that since we are penalizing large weights, a_{ij} will decay itself. Thus, the bias will be created this way.

The extreme learning approach claims that the input weights can be independent of the training data and the weights can be assigned before seeing the data. From our analysis, we find that it is unreasonable to assume such independence. In the proposed approach, the input weights are trained after seeing the data, so a relationship is built between the input weights and the data. Moreover, as the input weights are adjusted after initialization, the weights are not random anymore, neither is the hidden layer feature mapping. Based on the trained input weights, the output weights can be further

calculated.

3.2.3 Output weights training

Unlike the input weights that have to go through a sigmoid hidden layer, benefiting from the linear output layer, the output weights can be solved in closed form. With proper reformulation, equation (3.4) can be rewritten as

$$L(\boldsymbol{w},\boldsymbol{a}) = \frac{1}{2}[\| \mathbf{W}(\mathbf{H}\boldsymbol{w} - \mathbf{T}) \|^2 + \lambda_1 \| \boldsymbol{w} \|^2 + \lambda_2 \| \boldsymbol{a} \|^2] \tag{3.18}$$

where $\mathbf{W}$ is a diagonal matrix whose diagonal elements are W_i defined by (3.15).

For simplicity, consider $\| \boldsymbol{a} \|^2$ to be a constant and set the derivative of $L(\boldsymbol{w},\boldsymbol{a})$ with respect to $\boldsymbol{w}$, to zero,

$$\begin{aligned} \frac{\partial L}{\partial \boldsymbol{w}} &= -\mathbf{H}^{\mathrm{T}}\mathbf{W}(\mathbf{T} - \mathbf{H}\boldsymbol{w}) + \lambda_1 \boldsymbol{w} \\ &= \mathbf{H}^{\mathrm{T}}\mathbf{W}\mathbf{H}\boldsymbol{w} + \lambda_1 \boldsymbol{w} - \mathbf{H}^{\mathrm{T}}\mathbf{W}\mathbf{T} = 0 \end{aligned} \tag{3.19}$$

The output weights $\boldsymbol{w}$ can be solved by Iterative Reweighted Least Square (IRLS)

$$\boldsymbol{w}^{t+1} = (\mathbf{H}^{\mathrm{T}}\mathbf{W}^{t}\mathbf{H} + \lambda_1 \mathbf{I})^{-1}\mathbf{H}^{\mathrm{T}}\mathbf{W}^{t}\mathbf{T} \tag{3.20}$$

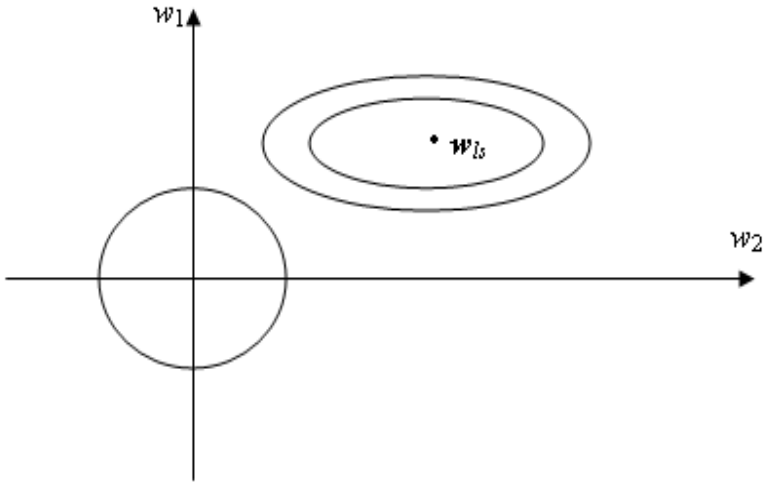


Figure 3.5. Output weight calculation

The principle of output weight decay can be illustrated by the following simple two-dimensional example. The constraint $\|\boldsymbol{w}\|^2 \leq s_1$ in (3.3) is represented by the circle. The final solution is the first point of the constraint circle to touch the contour of the ellipse cluster (sometimes it may not be a regular ellipse because of Huber) which stands for the

error function. The size of the circle and ellipse can be adjusted by the tradeoff parameter λ1 in (3.4). The point $\boldsymbol{w}_{ls}$ stands for the ordinary Least Square solution.

By further analyzing (3.18), after each IRLS step, $L(\boldsymbol{w},\boldsymbol{a})$ is found to be a convex function with respect to $\boldsymbol{w}$. Rewriting (3.18) to matrix form, we get

$$L(\boldsymbol{w},\boldsymbol{a}) = \frac{1}{2}[(\mathbf{H}\boldsymbol{w}-\mathbf{T})^{\mathrm{T}}\mathbf{W}(\mathbf{H}\boldsymbol{w}-\mathbf{T}) + \lambda_1 \boldsymbol{w}^{\mathrm{T}}\boldsymbol{w} + const. \tag{3.21}$$

The Hessian can be obtained by taking the second order derivative of (3.21) with respect to $\boldsymbol{w}$, that is

$$Hessian = \frac{\partial^2 L}{\partial \boldsymbol{w}^2} = \begin{bmatrix} \frac{\partial^2 L}{\partial w_1^2} & \frac{\partial^2 L}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 L}{\partial w_1 \partial w_L} \\ \frac{\partial^2 L}{\partial w_2 \partial w_1} & \frac{\partial^2 L}{\partial w_2^2} & \cdots & \frac{\partial^2 L}{\partial w_L \partial w_L} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 L}{\partial w_L \partial w_1} & \frac{\partial^2 L}{\partial w_L \partial w_2} & \cdots & \frac{\partial^2 L}{\partial w_L^2} \end{bmatrix}_{L\times L} + \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_1 \end{bmatrix}_{L\times L} = \mathbf{H}^{\mathrm{T}}\mathbf{W}\mathbf{H} + \lambda_1 \mathbf{I} \tag{3.22}$$

Noticing that $\lambda_1>0$ and W_i, defined by (3.15), is non-negative, for any $\boldsymbol{v} \in \boldsymbol{R}^L$, we have $\boldsymbol{v}^{\mathrm{T}}(\mathbf{H}^{\mathrm{T}}\mathbf{W}\mathbf{H} + \lambda_1\mathbf{I})\boldsymbol{v} = \left\|\mathbf{H}\times \mathrm{diag}(\sqrt{\mathrm{W}_i})\times \boldsymbol{v}\right\|^2 + \lambda_1 \left\|\boldsymbol{v}\right\|^2 > 0$, so the Hessian is positive definite. Therefore, we can say that L is a convex function with respect to $\boldsymbol{w}$.

3.2.4 The learning steps of the Proposed Approach

According to our experiments, tediously adjusting the input weights may not lead to better generalization ability. Simply adjusting the input weights once is enough to produce satisfying results. More adjustments constitute a waste of time. The learning steps of the proposed approach can be summarized as:

Learning steps of the proposed approach
Begin

1. Initialize all parameters, randomly generate inputs weights and bias, set **W**=diag(1,1,…1).
2. Pre-calculate the output weights according to (3.20).
3. Calculate the network's outputs according to (3.7).
4. Recalculate **W** according to (3.15)
5. Calculate the delta term for the hidden nodes and output nodes according to (3.14) and (3.15), respectively.
6. Update the input weights according to (3.17)

while convergence is not reached

7. Re-calculate the output weights according to (3.20)
8. Calculate the network's outputs according to (3.7).
9. Recalculate **W** according to (3.15)

end

end

3.2.5 Relation with other work

In the proposed approach, the feature mapping idea of the kernel learning is kept, but it is implemented in an untraditional way. As mentioned before, the proposed approach leads to a sparse classification model like SVMs, but it is not another speed-up version of SVMs. As mentioned in the previous chapter, many methods seek to speed up the kernel method by low-rank approximation of the Gram matrix via certain decomposition or sampling techniques. The proposed approach is not a kernel learning method, so no Gram matrix is necessary at all. Therefore, approximating the Gram matrix is irrelevant here.

In contrast with traditional MLP, the proposed approach does not extract features using the hidden layer. It will be seen in subsequent experiments that the hidden nodes can be set to a large number, up to hundreds, which is rare in traditional MLP. It gives in to extreme learning the tedious training of hidden parameter is unnecessary, but the hidden layer will not be random or independent of the input data. The proposed approach

can be viewed as an early stopping and regularization method [10], but it can lead to a robust and sparse learning model with a Huber-like loss.

3.3 Experiments

In this section, the proposed approach is tested and justified on several problems. First, the learning ability of the proposed approach is tested on the XOR problem. Then it will be evaluated on both real side scan sonar data and UCI datasets [16]. For comparison, the results of several other related machine learning algorithms are also given on these datasets.

3.3.1 Performance Measure

In this study, the AUC, the Area Under ROC Curve, is used to quantified the learning results [59]. The ROC (Receiver Operating Characteristic) curve is a two dimensional graph whose horizontal axis and vertical axis are the false positive rate (FPR) and the true positive rate (TPR), respectively. The definition of FPR and TPR are derived from the confusion matrix which is

Table 3.1 Confusion Matrix

actual / *Predicted*	*Positive*	*Negative*
Positive	True Positive (TP)	False Positive (FP)
Negative	False Negative (FN)	True Negative (TN)
Total	P	N

$$FPR = \frac{FP}{N} = \frac{FP}{FP + TN} \tag{3.23}$$

$$TPR = \frac{TP}{P} = \frac{TP}{TP + FN} \tag{3.24}$$

A point on the ROC curve corresponds to a FPR and TPR under a threshold. The ROC curve is a curve that connects the points (0, 0) and (1, 1) under various thresholds. The maximum AUC value is 1. Usually, any classifiers better than random guessing will

produce an AUC value larger than 0.5.

The FPR and TPR do not consider the class distribution of the positive and negative instances, so ROC and AUC are insensitive to class imbalances.

3.3.2 XOR problem

The XOR example is a well known problem to test the learning ability of Artificial Neural Networks. An experiment is conducted to solve the XOR problem using the proposed approach. The training set of the XOR problem only contains four instances listed in the following table.

Table 3.2 Training set of XOR problem

Instance No.	*Input* x_1	*Input* x_2	*Desired Output*
1	0	0	0
2	0	1	1
3	1	0	1
4	1	1	0

It is found from table 3.2 that the XOR example is a simple 2-dimensinal linearly inseparable problem，which means that no single decision line in the original 2-dimesnial space can perfectly separate the two classes. Therefore, the problem can only be solved with a (or several) non-linear classification boundary.

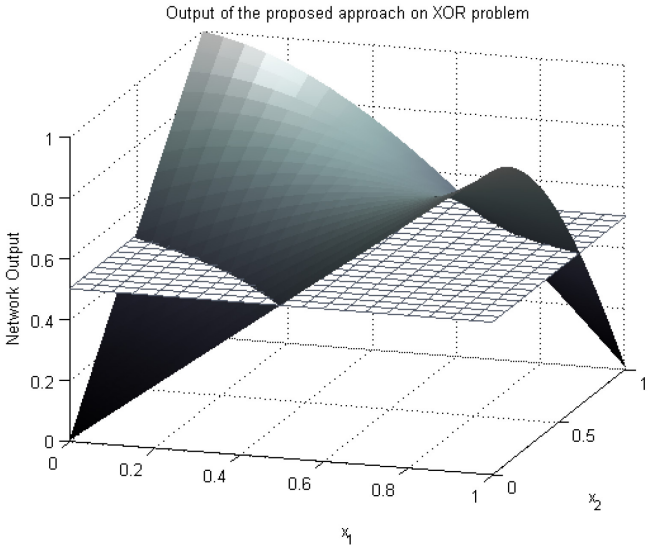


Figure 3.6 Output of the proposed approach on the XOR problem

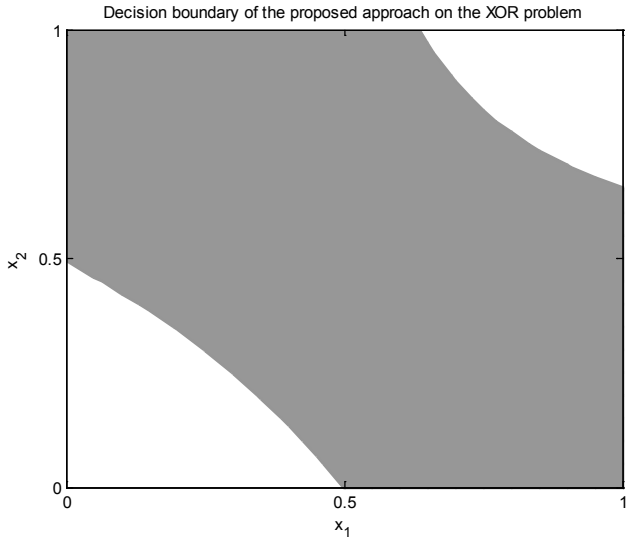


Figure 3.7 Decision boundary of the proposed approach on the XOR problem

Figure 3.6 shows the output of the proposed approach on the XOR problem. The plane in the figure shows $f(x_1, x_2) = 0.5$. Under the threshold of 0.5, the decision boundary produced by the proposed approach is shown in figure 3.7. It is found in figure 3.6 that the points (0, 0) and (1, 1) are under the plane $f(x_1, x_2) = 0.5$ and the points (0, 1) and (1, 0) are above that plane. For threshold 0.5, the two classes are perfectly separated. Furthermore, it is found from figure 3.6 that the output surface reaches its maximum at (0, 1) and (1, 0), and its minimum at (0, 0) and (1, 1). For other threshold in (0, 1), the two classes will also be separated. Therefore the XOR problem is successfully solved by the

proposed approach.

3.3.3 Side Scan Sonar Dataset

The sonar data used in this study is provided by the Ocean Systems Laboratory, Heriot-Watt University, Edinburgh, UK. The sonar images were colleted by an AUV fitted with a side scan sonar from a trail on Loch Earn (Scotland) on November 10th and November 11th, 2010. The sonar images gathered on November 10th are used as the training set and the data gathered on November 11th are used as the testing set. The size of each sonar image is 500×1024, so there are 512,000 pixel values in each raw image.

(1) Data pre-processing

Raw sonar images have to be properly pre-processed before machine learning algorithms are applied. We are only interested in the foreground objects. Therefore, the large amount of background (seabed) data has to be filtered out.

It is reasonable to assume that the foreground objects will have a more complex texture than the seabed. Thus, the foreground object areas are obtained by using local range and standard deviation filters [21].

The objective of the image processing procedures at this point is data reduction rather than MLOs detection. Thus, a relatively high false alarm rate is acceptable.

Figure 3.8 illustrates the extraction of foreground objects from the sonar image. Objects or areas that do not have a reasonable size will be ignored. It is worth mentioning that a side scan sonar is unable to produce images about the seabed right below the AUV, as no echoes are received there. As a result, there will be a large black area near the center line of the scan. Figure 3.8 only shows the left half of the original image.

For object detection tasks, an object should be detected through a single view, no matter where and how it lies on the seabed. Therefore, the feature used should be robust to the location and orientation of the object.

The grayscale histogram, a simple but informative statistical feature, is considered. In many image recognition systems, many complex features are used, but such features will inevitably increase the computational complexity, impeding the detection speed. The

histogram is easy to calculate and robust to rotation, moreover, the distribution of the grayscale value can be well described by this feature.

In our experiment, the grayscale value (0-255) is divided into 16 bins with width 16. The grayscale histogram is normalized to the frequency that a pixel value falls into each bin. Figure 3.9 shows an example of a grayscale histogram.

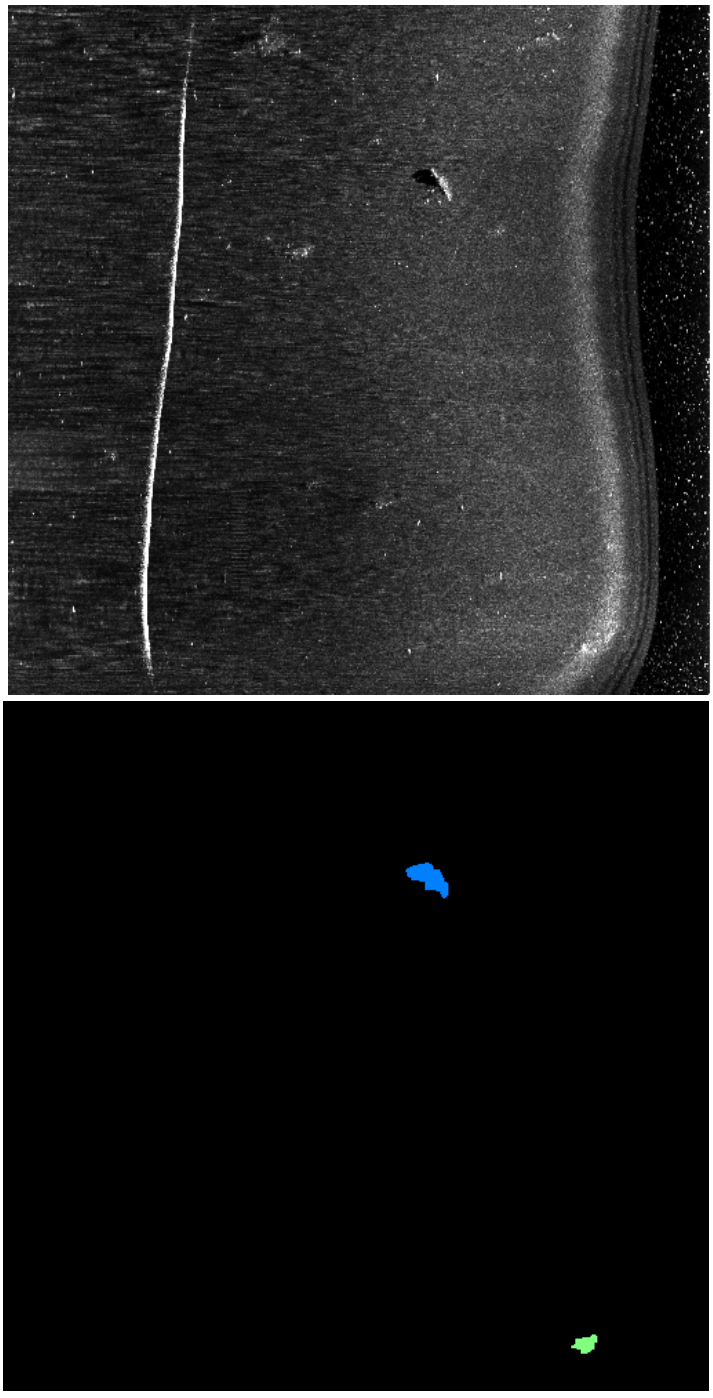


Figure 3.8 Example of image processing results

The dataset information is summarized in table 3.3. It can be found that the data is highly imbalanced. The MLOs are labeled as the positive examples. Since all the features are in the interval (0, 1), there is no need to normalize the data when applying learning algorithms.

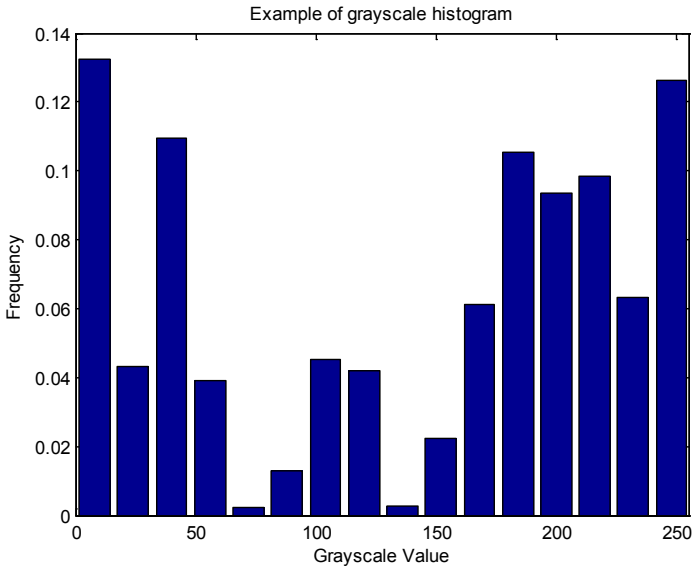


Figure 3.9 Example of Grayscale Histogram

Table 3.3 Side Scan Sonar Dataset information

	Training Set	*Testing Set*
# positive instances	18	17
# negative instances	2202	1130
# pos./ # neg.	0.0082	0.0150
Instances in total	2220	1147
# features	16	16

(2) Experimental Set Up

For the proposed method, the number of hidden nodes L is fixed at 200 and λ_2 is fixed at 0.5. The initial input weights are randomly generated from the input hyper-plane such that for each hidden node $i, \sum_{j=1}^{16} a_{ij} = 1$. For ELM, following its original implementation, the input weights are randomly generated in (-1, 1) according to the uniform distribution. For ELM and BP, the number of hidden nodes is selected to produce its best result and all other parameters in BP are default parameters. For the kernel methods, the Gaussian

kernel, one of the most widely used kernel functions, is adopted in our experiment. The definition of the Gaussian functions is

$$f(\boldsymbol{x}) = \exp(-\gamma \|\boldsymbol{x}\|^2) \tag{3.25}$$

Since the dataset is highly imbalanced, the positive and negative instances are weighted differently according to their ratio in the training set for all methods. For the kernel methods, the regularization parameter C and scaling parameter γ are both optimized by grid search from $\{2^{-10},2^{-9},\ldots,2^{10}\}$. For the proposed approach, the combination of parameters (λ_1, η) is optimized from $\{2^{-10},2^{-9},\ldots,2^{10}\}$ and $\{2^{-10},2^{-9},\ldots,2^{0}\}$. For a fair comparison, all models are implemented in Matlab, including the QP optimizer for SVMs [9]. The experiment is carried out on the same computer with a 2.00GHz CPU.

(3) Results

Table 3.4 shows the comparison of testing results (LR is short for Logistic Regression) [24]. The TP rate and FP rate are also given for reference. Table 3.5 shows the model parameters used as well as the training and classification time.

From the result tables we can see that the proposed approach is able to beat all neural network based models in prediction performance. The proposed approach can produce a prediction result close to or even better than the result obtained by kernel methods (proposed method beat both SVMs and kernel LR). In terms of training and classification time, the performance of the proposed method largely outperformed all kernel methods.

The SVM has to solve a quadratic programming problem and, therefore, it takes the longest time. Other kernel machines, LS-SVMs [58] and kernel-LR[38] solve linear equations, and the main computational load is to calculate the inverse of the Ω matrix whose size is the same as that of the Gram matrix. The complexity for calculating the inverse of the Ω matrix is $O(N^3)$. For the proposed approach, the feature space is known, and the decision boundary is directly conducted in the feature space. The computational complexity for (3.20) is $O(L^3+LN)$ [40]. In our experiment, L=200, N=2220. For SLFN, most parameters (weights) are in the hidden layer, unfortunately, BP consumes much time to tune these parameters, so it is much slower than the proposed approach. However, benefiting from the small network size, it is faster than the kernel machines.

Moreover, for kernel machines, the classification speed is directly related to the number of support vectors and the kernel function. When trained on large datasets, the classification speed of kernel machines will tend to be slow. In LS-SVMs and Kernel LR, every data point will be used to build the final decision boundary, so every data point will become a support vector. For SVMs, when using the same hyper parameter, in our case (C, γ), a large dataset is more likely to result in more support vectors.

However, similarly to other SLFN methods, the classification speed of the proposed approach is only related to the parameter L, independent of the size of the training set N. Therefore, the classification speed can be directly under control by properly setting the value of L when we first built the SLFN.

Table 3.4 Comparison of performance on Side Scan Sonar Data

Method	*Function*	*TP rate*	*FP rate*	*AUC*
SVMs	Gaussian	0.8824	0.0442	0.9865
LS-SVMs	Gaussian	0.9412	0.0540	0.9885
Kernel LR	Gaussian	0.9412	0.0416	0.9858
BP	Sigmoid	0.9353	0.3145	0.9231
ELM	Sigmoid	0.9059	0.0686	0.9747
LR	\	0.4117	0.1876	0.7989
Proposed Approach	Sigmoid	0.9177	0.0443	0.9871

Table 3.5 Model parameters and comparison of time on Side Scan Sonar Data

Method	*Pameters*	*SVs*	*Training time(s)*	*Classification time(s)*
SVMs	$C=2^{-5}$, $\gamma=2^{4}$	618	94.17	0.9063
LS-SVMs	$C=2^{2}$, $\gamma=2^{2}$	2220	13.07	2.1719
Kernel LR	$C=2^{-6}$, $\gamma=2^{0}$	2220	33.37	2.2031
BP	L=20	\	9.173	0.0313
ELM	L=40	\	0.090	<0.01
LR	$C=2^{-2}$	\	2.093	<0.01
Proposed Approach	$\lambda_1=2^{-8}$, $\eta=2^{-7}$	1078	1.123	0.0391

Figure 3.10 shows the performance of the proposed approach on the parameter space, η-λ_1 plane. The performance of SVM on its parameter space, the γ-C plane, is shown in Figure 3.11.

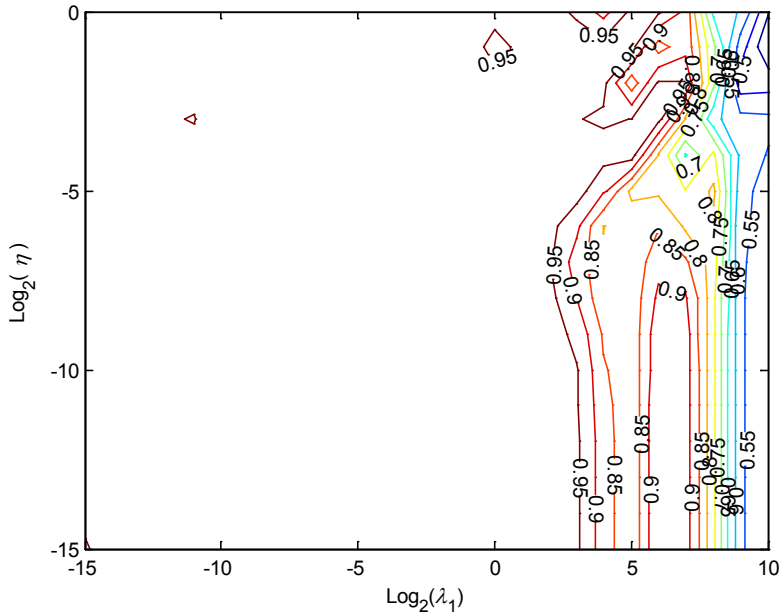


Figure 3.10 Contour plot of AUC on η-λ_1 plane

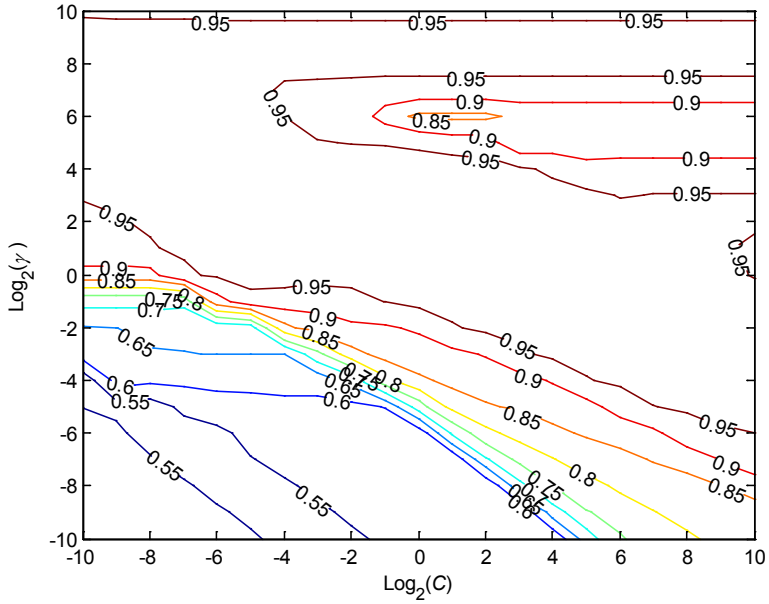


Figure 3.11 Contour plot of AUC on γ-C plane

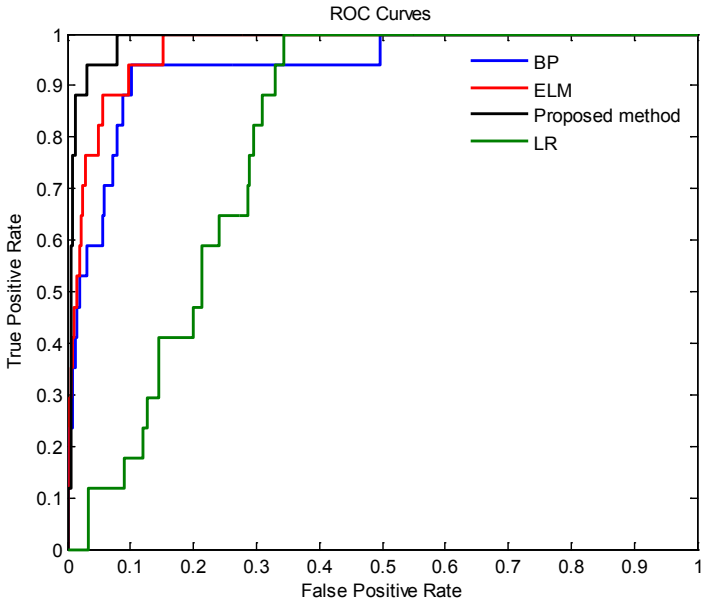


Figure 3.12 ROC Curves 1

Figure 3.12 and 3.13 show the comparison of ROC curves. It can be found in figure 3.13 that the ROC curve of the proposed method is very similar to ROC curves produced by kernel machines. Figure 3.14 shows the convergence of the proposed approach.

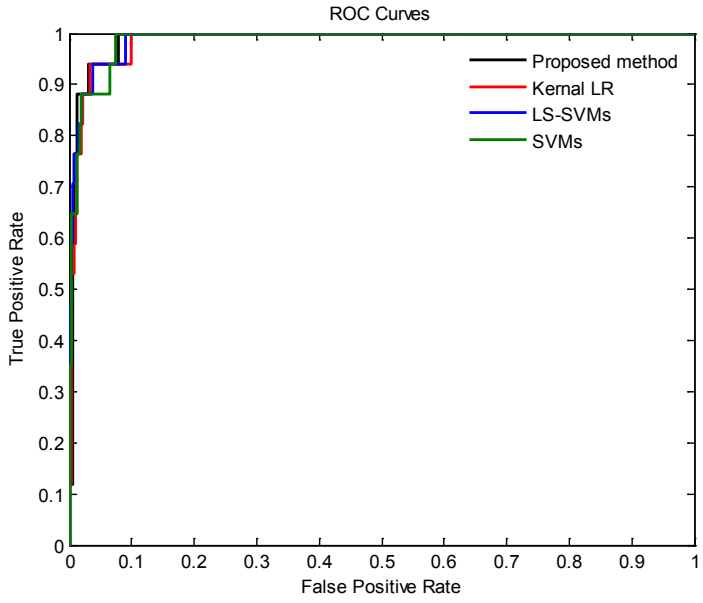


Figure 3.13 ROC curves 2

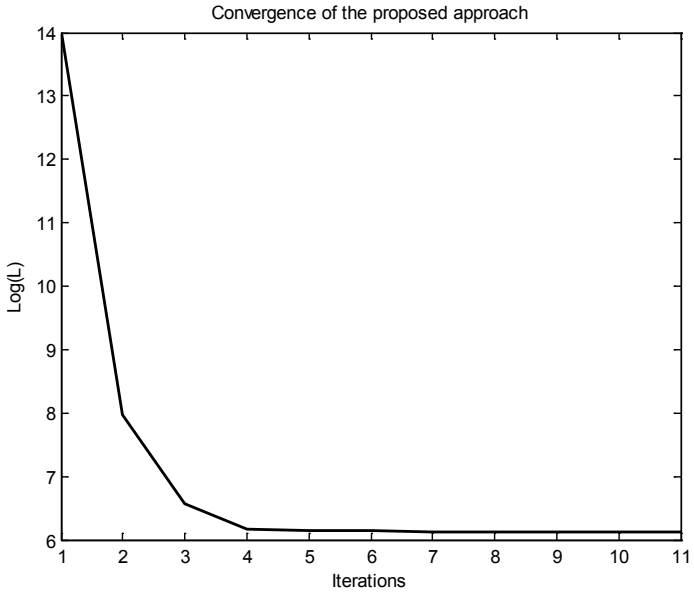


Figure 3.14 Convergence of the proposed approach

From the application side, as a military task, usually there is a time limit for the MCMs task. A large false alarm rate is unacceptable as much time could be consumed to examine and exclude the large number of false alarms. From the results, we can find that the proposed approach will produce about 50 false alarms, which means that about one quarter of all the alarms represent the true MLOs. From this perspective, the false alarm rate is acceptable in real MCMs.

3.3.4 UCI Datasets

The proposed approach is a general method. It can also be apply to other domains. In this section, the performance of the proposed approach is tested on UCI domains [16]. The details of the UCI datasets used in this study are listed in Table 3.6. As mentioned before, only binary classification is considered. Some of the datasets listed below were originally used for multiple classifications, so multiple classes are merged into two classes for such datasets. Before training, all attributes are normalized into the interval [-1, 1].

(1) Experimental set up

Similarly to section 3.3.3, the kernel for SVMs is the Gaussian kernel, which is a common choice, and the hidden activation function is chosen to be the logistic sigmoid for all the other models. The initial input weights for ELM and the proposed approach are

both generated according to uniform distribution in the interval (-1, 1).

A grid search is conducted to optimize the parameters of SVMs (C and γ) and the proposed approach (λ_1 and η). The number of hidden nodes L used for the proposed approach is set to a large number, without much careful optimization. Other parameters remain the same to those used in section 3.3.3 for the side scan sonar data. The details of the model parameters used in the experiment are listed in table 3.8.

In order to deal with the problem of class imbalance in training, for all models, the misclassification costs of positive and negative instances are weighted differently according to their ratio in the training set. The results shown below are from an experiment of 5×10-fold cross-validation. Both the average results and the standard deviation (shown in brackets below) are recorded. The definition of standard deviation is

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}, \text{ where } \mu = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{3.26}$$

Table 3.6 UCI datasets information

Dataset	*Attributes*		*# pos. Instance*	*# neg. Instance*	*# pos./ # neg.*
	Real	*Integer*			
WDBC	0	9	239	444	0.5283
CMC	0	9	629	844	0.7453
CTG	0	20	1655	471	3.514
diabetes	2	6	268	500	0.538
glass	9	0	68	146	0.4658
haberman	0	3	225	81	2.778
image segmentation	16	3	1320	990	1.333
ionosphere	32	2	225	126	1.786
irish	0	5	175	325	0.5385
liver	1	5	145	200	0.725
sat	0	36	918	1082	0.8484
spambase	55	2	1813	2788	0.6503
wine	11	2	107	71	1.507

(2) Results

Table 3.7 shows the comparison of training time. It can be seen that the proposed approach is much faster than BP and SVMs in training on all datasets.

As mentioned before, SVM has to solve a quadratic programming problem that deals

with an N square matrix, while in the proposed approach, the size of the Hessian matrix in (3.20) is L square. Therefore, on large datasets, such as the *spambase data*, the advantage of learning speed of the proposed approach is significant relative to SVMs. For BP, much training time is consumed to tune the hidden layer parameters, therefore, when the dataset (*spambase*) or network size (*sat*, L=35) becomes large, it will be very slow.

ELM uses a random hidden layer, so it has the fastest learning speed on most datasets. Even though ELM does not train the hidden layer, the proposed approach, which uses an even larger network, is still able to maintain a learning speed only slightly slower than ELM on some datasets (*image segmentation* and *sat*).

Table 3.8 shows the model parameters. It is found that, similarly to SVMs, the proposed approach will have a sparse solution. In classification, a testing point will be evaluated by the L hidden nodes in the proposed approach, while in SVMs, it will be evaluated with every support vector using the kernel function. In most of the cases, the value of L in the proposed approach is smaller that the number of support vectors in SVMs. Therefore the proposed approach will have a faster classification speed.

Figure 3.15 shows the convergence of the proposed approach on UCI dataset (WDBC). It is easy to see from the figure that solving equation (3.20) via IRLS converges within less than 10 iterations.

Table 3.9 is the comparison of performance. The mean AUC value and the standard deviation are recorded.

Table 3.7 Comparison of Training Time(s) on UCI domains

Dataset	*BP*	*ELM*	*SVMs*	*Proposed Approach*
WDBC	1.077	0.0031	3.895	0.2437
	(0.2027)	(0.0066)	(0.6330)	(0.0322)
CMC	1.411	0.1078	61.24	1.3843
	(0.3213)	(0.0172)	(2.361)	(0.0670)
CTG	23.01	0.7437	85.30	4.0125
	(6.127)	(0.0692)	(5.427)	(0.5266)
diabetes	0.9078	0.0047	8.0922	0.8409
	(0.1830)	(0.0075)	(1.3124)	(0.0656)
glass	0.7813	0.0019	0.9687	0.1562
	(0.0979)	(0.0060)	(0.1462)	(0.0165)
haberman	0.675	0.0016	2.300	0.0688

	(0.1296)	(0.0007)	(0.3075)	(0.0489)
image segmentation	14.948	0.900	32.20	1.9860
	(3.558)	(0.0391)	(3.096)	(0.2907)
ionosphere	2.255	0.0172	3.155	0.1640
	(1.190)	(0.0049)	(0.4969)	(0.0184)
irish	0.9989	0.0031	9.414	0.275
	(0.5665)	(0.0066)	(0.8928)	(0.0296)
liver	0.9934	0.0051	2.6406	0.0913
	(0.1409)	(0.0080)	(0.3212)	(0.0179)
sat	56.68	0.5549	32.91	0.8037
	(4.765)	(0.0482)	(3.247)	(0.0515)
spambase	111.5	1.973	461.2	11.05
	(48.82)	(0.0347)	(6.073)	(1.711)
wine	1.313	0.0034	0.8953	0.0336
	(0.5420)	(0.0065)	(0.1409)	(0.0186)

Table 3.8 Comparison of model parameters on UCI domains

Dataset	*BP*	*ELM*	*SVMs*	*Proposed Approach*
WDBC	L=15	L=15	C=2^{-2}, γ=2^{-8}, SVs=289.1	L=200, λ_1 =2^{4}, η=2^{-8},SVs=166.8
CMC	L=5	L=60	C=2^{6}, γ=2^{-4}, SVs=911.5	L=300, λ_1 =2^{0}, η=2^{-7},SVs=1273.6
CTG	L=30	L=150	C=2^{9}, γ=2^{-2}, SVs=625.0	L=400, λ_1 =2^{-8}, η=2^{-9},SVs=950.1
diabetes	L=5	L=15	C=2^{3}, γ=2^{-5}, SVs=414.5	L=300, λ_1 =2^{0}, η=2^{-4},SVs=608.1
glass	L=15	L=10	C=2^{-1}, γ=2^{4}, SVs=158.5	L=100, λ_1 =2^{4}, η=2^{-3},SVs=173.1
haberman	L=5	L=15	C=2^{6}, γ=2^{-1}, SVs=187.4	L=100, λ_1 =2^{-4}, η=2^{-6},SVs=272.1
image segmentation	L=20	L=160	C=2^{6}, γ=2^{0}, SVs=256.9	L=300, λ_1 =2^{-10}, η=2^{-10},SVs=263.4
ionosphere	L=10	L=60	C=2^{6}, γ=2^{1}, SVs=247.6	L=200, λ_1 =2^{1}, η=2^{-5},SVs=165.1
irish	L=10	L=15	C=2^{-1}, γ=2^{6}, SVs=411.6	L=200, λ_1 =2^{-7}, η=2^{-1},SVs=311.1
liver	L=30	L=20	C=2^{10}, γ=2^{-4}, SVs=204.1	L=100, λ_1 =2^{-6}, η=2^{-9},SVs=284.3
sat	L=35	L=120	C=2^{3}, γ=2^{-4}, SVs=306.8	L=200, λ_1 =2^{-2}, η=2^{-5},SVs=500.1
spambase	L=15	L=150	C=2^{6}, γ=2^{-1}, SVs=765.8	L=500, λ_1 =2^{-10}, η=2^{-10},SVs=1638.8
wine	L=30	L=20	C=2^{-10}, γ=2^{-4},	L=100, λ_1 =2^{-2},

	SVs=160.2	$\eta=2^{-10}$,SVs=30.8

Table 3.9 Comparison of performance (AUC) on UCI domains

Dataset	*BP*	*ELM*	*SVMs*	*Proposed Approach*
WDBC	0.9880	0.9937	0.9953	0.9956
	(0.0129)	(0.0049)	(0.0048)	(0.0047)
CMC	0.7331	0.7177	0.7474	0.7394
	(0.0379)	(0.0403)	(0.0365)	(0.0423)
CTG	0.9109	0.8898	0.9351	0.9295
	(0.0295)	(0.0241)	(0.0192)	(0.0185)
diabetes	0.8106	0.8308	0.8359	0.8367
	(0.0650)	(0.0427)	(0.0406)	(0.0496)
glass	0.8062	0.8321	0.8907	0.8426
	(0.1275)	(0.1097)	(0.0821)	(0.1098)
haberman	0.6352	0.6986	0.7020	0.7171
	(0.1223)	(0.1097)	(0.0984)	(0.1117)
image segmentation	0.9941	0.9903	0.9944	0.9955
	(0.0038)	(0.0070)	(0.0042)	(0.0046)
ionosphere	0.9307	0.9381	0.9799	0.9670
	(0.0514)	(0.0418)	(0.0221)	(0.0329)
irish	0.9256	0.9206	0.9620	0.9246
	(0.0401)	(0.0265)	(0.0171)	(0.0444)
liver	0.7316	0.7480	0.7667	0.7751
	(0.0926)	(0.0713)	(0.0899)	(0.0731)
sat	0.9838	0.9803	0.9864	0.9842
	(0.0122)	(0.0090)	(0.0069)	(0.0070)
spambase	0.9682	0.9566	0.9762	0.9702
	(0.0089)	(0.0093)	(0.0065)	(0.0086)
wine	0.9833	0.9954	0.9969	0.9977
	(0.0526)	(0.0089)	(0.0045)	(0.0036)

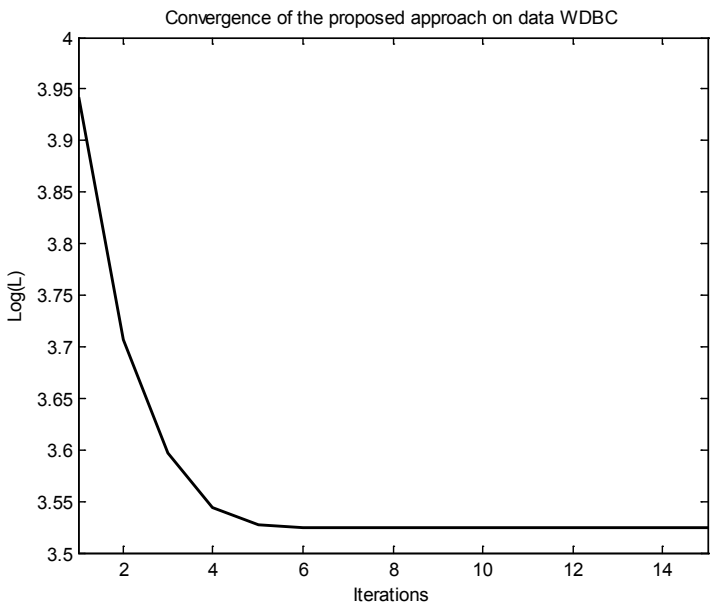


Figure 3.15 Convergence of the proposed approach on UCI dataset (WDBC)

(3) Statistical test

To evaluate the results, statistical tests are performed on Table 3.9. To compare multiple algorithms on multiple domains, we chose Friedman's test and post-hoc test [34]. First, Friedman's test is conducted. The test returns a χ_F^2 value of 28.94 and a *p*-value of $2.31\times10^{-6}<0.01$, so the H_0 hypothesis that all the classifiers have similar performance with each other on the test datasets is rejected.

Furthermore, the post-hoc test (Nemenyi's test) is conducted. The value of q_α is 2.72 for α=0.05 and *df*=36. We get the *q* statistics between the proposed approach and the two neural network methods, BP and ELM, which are both 3.65> q_α=2.72. Therefore, the null hypothesis, that the performance of the proposed approach is equivalent to those of BP and ELM, is rejected. The *q* statistic between the proposed approach and SVMs is 0.30< q_α=2.72 and the null hypothesis is accepted.

Therefore, from the results of Friedman's test and Nemenyi's test, we conclude that on the UCI datasets, the proposed approach outperformed BP and ELM. The difference between the performance of the proposed approach and SVMs is insignificant.

3.4 Conclusions

It can be concluded from the experimental results that the proposed approach will result in a sparse learning model. Due to the error function shown in figure 3.4, only a part of the training data points will be used to build the final decision boundary.

Furthermore, the proposed approach with explicit SLFN feature mapping is able to improve the prediction performance of neural network based methods. From our previous analysis, we found that it is more reasonable to look at the solution, the weights themselves, in the SLFN training process. Tedious training for the SLFN hidden layer parameters while ignoring the norm of the weights is inappropriate and unnecessary. Also the norm of the input weights, which affect the working region of sigmoid function, will affect the non-linearity of the decision boundary, so it can not be a random number independent of the training data.

The proposed approach is able to produce a satisfying generalization result as long as the norm of the both input weights and the output weights are properly considered in the training process. Benefiting from no tedious hidden layer training, the learning speed of the proposed approach is much faster than that of traditional BP.

Moreover, compared to kernel machines, the proposed approach will produce a generalization performance (AUC) close to that of kernel machines, with much improvement in the learning speed. On the Side Scan Sonar data, the classification speed will also be largely improved. The improvement of training and classification speed is the result of explicit feature mapping which enables the direct feature space computation. In return, the explicit feature space is limited due to the curse of dimensionality. However, in many cases, the limitation does not affect the generalization performance of the proposed approach.

Chapter 4

Proposed Approach for Regression

Similarly to other machine learning algorithm, the proposed approach is not limited to classification, it can be extended to solving regression problems. For SVMs and SLFNs, solving a classification problem is to approximate or learn a decision boundary defined by the positive and negative training points, which in essence is a function approximation problem.

Similarly to chapter 3, instead of using kernels, we will build a SLFN to implement explicit feature mapping. The resulting model will produce a hyper-plane in the feature space to approximate the non-linear surface in the input space.

According to our previous analysis, a SLFN with large weights will produce a more complex surface. When the sigmoid works in a region close to linear, the surface produced by the SLFN tends to be simple and nearly linear. Therefore, similarly to classification, for regression problems, we will also take into account the norm of the weights in the training process.

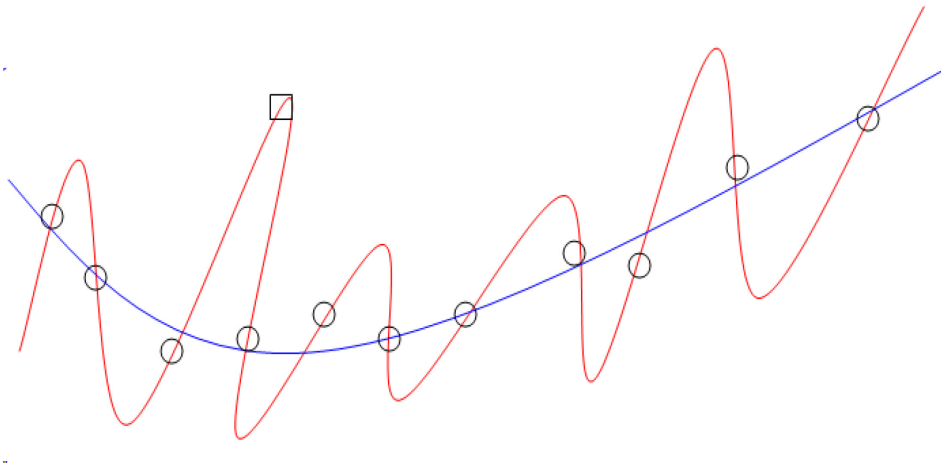


Figure 4.1 Prediction curves produced by SLFN with different norm of weights

The simple one dimensional regression example in figure 4.1 shows the impact of weights norm on the final output curve. The dots are the training points. The simple blue line shows the optimal fit. We can see that not all dots fall on the blue curve. This is because for most of the datasets, especially real life datasets, there will be a certain amount of noise.

However, if the training process is only focused on the training error and ignores the norm of the weights, then the resulting prediction curve, shown by the red line, is very likely to overfit the data. The resulting curve will exactly pass through every training point and becomes very complex. Due to the large weights, in prediction, the SLFN is likely to output a point that lies far away from the data distribution, as shown by the square in the above figure. Therefore, a network will large weights is unlikely to generalize well in regression problems.

For simplicity, only regression problem with one output, Single Input Single Output (SISO) or Multiple Input Single Output (MISO), will be considered in this chapter. Therefore, similarly to the classification model, only one linear output node will be used in the proposed approach. However, the proposed approach is not limited to such single output models and it can be easily extended to solving regression problems with multiple outputs.

In this chapter, we will discuss how to properly build and train a SLFN based on the classification model introduced in chapter 3. Firstly, the objective loss function for the regression model will be derived. Then we will focus on the proper way to solve for the weights. Furthermore, the relationship between the proposed model and Support Vector Regression (SVR) will also be discussed. Section 4.2 is the experimental part, and our conclusion will be presented in section 4.3.

4.1 Proposed Approach for Regression

As previously discussed, the norm of the weights is important to the generalization performance of the SLFN regression model. Similarly to the classification model, the

norms of both input and output weights will be addressed. The proposed regression model will be derived from a statistic view and it will be found that, while for classification, the output weights will be calculated analytically instead of by iterative computation.

4.1.1 Weight Decay in regression

We assume that we are approximating a model in the feature space defined by a SLFN,

$$\mathbf{T} = \mathbf{H}\boldsymbol{w} + \boldsymbol{\xi} \tag{4.1}$$

where $\mathbf{T}=[t_1, t_2, \ldots, t_N]^{\mathrm{T}}$, is the label and $\boldsymbol{\xi}=[\xi_1, \xi_2,\ldots, \xi_N]^{\mathrm{T}}$ is the noise or residual, which is usually assumed to be a zero-mean Gaussian. $\mathbf{H}$ is the hidden output matrix defined by (3.6) and N is the number of training instances.

We consider a zero-mean Gaussian prior with a precision parameter α_1 for the each of the output weight w_i, that is [6]

$$p(w_i \mid \alpha_1) = N(w_i \mid 0, \alpha_1^{-1}) = (\frac{\alpha_1}{2\pi})^{1/2} \exp(-\frac{\alpha_1 w_i^2}{2}),\ i = 1, 2, ..., L \tag{4.2}$$

where L is the number of hidden nodes. We assume independence for the prior of the weights, therefore we have

$$\begin{aligned} p(\boldsymbol{w} \mid \alpha_1) &= \prod_{i=1}^{L} p(w_i \mid \alpha_1) \\ &= N(\boldsymbol{w} \mid 0, \alpha_1^{-1}\mathbf{I}) = (\frac{\alpha_1}{2\pi})^{L/2} \exp(-\frac{\alpha_1 \boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}}{2}) \end{aligned} \tag{4.3}$$

Similarly, we assume a Gaussian prior for the input weights $\boldsymbol{a}$

$$p(\boldsymbol{a} \mid \alpha_2) = N(0, \alpha_2^{-1}\mathbf{I}) \tag{4.4}$$

Given the train data $\boldsymbol{x}$ and the precision β, the prior for $\mathbf{T}$ can be given by

$$p(\mathbf{T} \mid \beta, \boldsymbol{w}, \boldsymbol{a}) = N(\mathbf{T} \mid \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{a}, \boldsymbol{x}), \beta^{-1}\mathbf{I}) = (\frac{\beta}{2\pi})^{N/2} \exp(-\beta E_D) \tag{4.5}$$

where $\boldsymbol{\phi}(\boldsymbol{a}, \boldsymbol{x})$ is the hidden layer feature mapping and $E_D=\|\mathbf{H}\boldsymbol{w}-\mathbf{T}\|^2/2$.

The posterior for the weights $\boldsymbol{a}$ and $\boldsymbol{w}$ is

$$p(\boldsymbol{a},\boldsymbol{w}\,|\,\mathbf{T},\boldsymbol{\alpha},\beta)=\frac{p(\mathbf{T}\,|\,\boldsymbol{w},\beta)\,p(\boldsymbol{a},\boldsymbol{w}\,|\,\boldsymbol{\alpha})}{p(\mathbf{T}\,|\,\boldsymbol{\alpha},\beta)} \tag{4.6}$$

where

$$p(\boldsymbol{a},\boldsymbol{w}\,|\,\boldsymbol{\alpha})=p(\boldsymbol{w}\,|\,\alpha_1)\,p(\boldsymbol{a}\,|\,\alpha_2),\ \boldsymbol{\alpha}=[\alpha_1,\alpha_2]^{\mathrm{T}} \tag{4.7}$$

Considering (4.3) (4.4) and (4.5), the log likelihood of (4.6) becomes

$$\begin{aligned}\log p(\boldsymbol{a},\boldsymbol{w}\,|\,\mathbf{T},\boldsymbol{\alpha},\beta)&=-\frac{\beta}{2}\sum_{i=1}^{N}[\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x}_i)-t_i]^2-\frac{\alpha_1}{2}\sum_{i=1}^{L}w_i^2-\frac{\alpha_2}{2}\sum_{i=1}^{L}\sum_{j=1}^{M}a_{ij}^2+const.\\&=-\frac{\beta}{2}\left\|\mathbf{H}\boldsymbol{w}-\mathbf{T}\right\|^2-\frac{\alpha_1}{2}\left\|\boldsymbol{w}\right\|^2-\frac{\alpha_2}{2}\left\|\boldsymbol{a}\right\|^2+const.\end{aligned} \tag{4.8}$$

The constant term is from the denominator part in (4.6) which is independent of the weights $\boldsymbol{a}$ and $\boldsymbol{w}$. Therefore, maximizing the log likelihood of (4.8) is equivalent to minimizing the following loss function,

$$\begin{aligned}&L_0(\boldsymbol{a},\boldsymbol{w})=\frac{\beta}{2}\sum_{i=1}^{N}E(\xi_i)+\frac{\alpha_1}{2}\sum_{i=1}^{L}w_i^2+\frac{\alpha_2}{2}\sum_{i=1}^{L}\sum_{j=1}^{M}a_{ij}^2\\&s.t.\ \xi_i=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{a},\boldsymbol{x}_i)-t_i,\ E(\xi_i)=\xi_i^2\end{aligned} \tag{4.9}$$

where M is the number of input nodes. It is worth mentioning that the constant term in (4.8) is ignored here. Furthermore, dividing L_0 by β and letting $\lambda_1=\alpha_1/\beta$, $\lambda_2=\alpha_2/\beta$, the above loss can be rewritten as follows

$$\begin{aligned}&\min L(\boldsymbol{w},\boldsymbol{a})=\frac{1}{2}[\sum_{i=1}^{N}E(\xi_i)+\lambda_1\sum_{i=1}^{L}w_i^2+\lambda_2\sum_{i=1}^{L}\sum_{j=1}^{M}a_{ij}^2]\\&s.t.\ \xi_i=\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{a},\boldsymbol{x}_i)-t_i,\ E(\xi_i)=\xi_i^2\end{aligned} \tag{4.10}$$

It is noticed that the objective function (4.10) is very similar to the objective function (3.4) used in the classification model. The only difference is that the Huber-like error function is replaced by the two side square error.

Figure 4.2 shows the comparison between the square error function and the Huber-like error function used in the classification model. The red line is the Huber-like function and the black line is the square error.

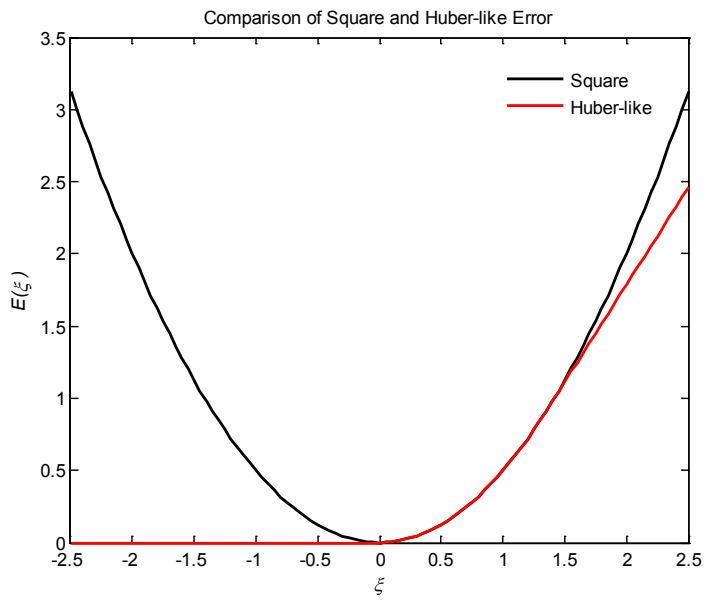


Figure 4.2 Comparison of square and Huber-like Error

In classification, no penalty is assigned to the points that lie on the right side of the classification boundary outside the margin (the space between the positive and negative class). Therefore, the error function used in the classification model is asymmetric and it will be zero if $\xi<0$. However, when it comes to regression, the data points rarely lie exactly on the plane (or hyper-plane) we are calculating, so in most regression cases, every data points will be penalized in the training process.

4.1.2 SLFN weights training

Similarly to the previous chapter, the learning rule for the input weights can be found by taking the derivative of (4.10) with respect to $\boldsymbol{a}_{ij}$,

$$\frac{\partial L}{\partial a_{ij}} = \frac{\partial E}{\partial a_{ij}} + \lambda_1 \frac{\partial \| \boldsymbol{w} \|^2}{\partial a_{ij}} + \lambda_2 a_{ij} \tag{4.11}$$

From (4.7), we can see that the input weights and output weights are assumed to be independent, so the second term $\lambda_1 \frac{\partial \| \boldsymbol{w} \|^2}{\partial a_{ij}} = 0$ .Recall from chapter 3 that the first term $\frac{\partial E}{\partial a_{ij}}$ is given by (3.11), that is

$$\begin{aligned}\frac{\partial E}{\partial a_{ij}} &= \frac{\partial E}{\partial net_i^h} \times \frac{\partial net_i^h}{\partial a_{ij}} \\ &= \frac{\partial E}{\partial net_i^h} \times x_j\end{aligned} \tag{4.12}$$

Since a different error function is used, when calculating the $\frac{\partial E}{\partial net_i^h}$, the definition of δ^o in (3.15) becomes

$$\delta^o = t_i - \boldsymbol{w}^{\mathrm{T}} \phi(x_i) \tag{4.13}$$

Following the same deduction as in chapter 3, we will come to the same learning rule (except for the different formula for δ^o) for the input weights,

$$a_{ij} = a_{ij} + \Delta a_{ij}, \Delta a_{ij} = -\eta \frac{\partial E}{\partial a_{ij}} = \eta(\delta_j^h x_i - \lambda_2 a_{ij}) \tag{4.14}$$

For the output weights $\boldsymbol{w}$, we take the derivative of L with respect to $\boldsymbol{w}$ and set it to 0. We have

$$\begin{aligned}\frac{\partial L}{\partial \boldsymbol{w}} &= -\mathbf{H}^{\mathrm{T}}(\mathbf{T} - \mathbf{H}\boldsymbol{w}) + \lambda_1 \boldsymbol{w} \\ &= \mathbf{H}^{\mathrm{T}}\mathbf{H}\boldsymbol{w} + \lambda_1 \boldsymbol{w} - \mathbf{H}^{\mathrm{T}}\mathbf{T} = 0\end{aligned} \tag{4.15}$$

Thus, the output weights can be analytically solved as

$$\boldsymbol{w} = (\mathbf{H}^{\mathrm{T}}\mathbf{H} + \lambda_1 \mathbf{I})^{-1} \mathbf{H}^{\mathrm{T}}\mathbf{T} \tag{4.16}$$

It is worth mentioning that by further analysis, we found that the Hessian $\mathbf{H}^{\mathrm{T}}\mathbf{H}+\lambda_1\mathbf{I}$ is positive definite, so we can conclude with probability equal to one that the output weights solution (4.16) is the global optimum.

With the help of (4.16), we can have more insight into the behaviors of the proposed regression model in prediction. In fact, the output of the SLFN can be viewed as a linear smoother in prediction [6].

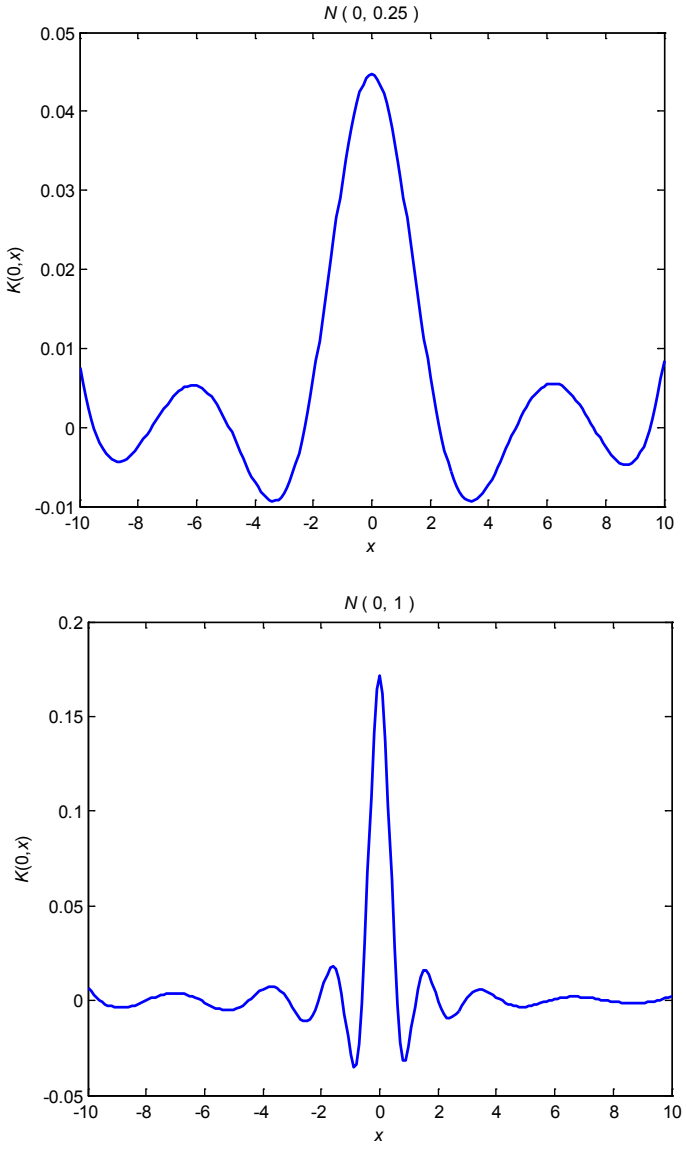
Given a testing sample ($\boldsymbol{x}^*$, t^*), the predictive output of the SLFN can be written as

$$y^* = \boldsymbol{w}^{\mathrm{T}} \phi(\boldsymbol{a}, \boldsymbol{x}^*) \tag{4.17}$$

Taking (4.16) into account, with proper reformulation, (4.17) can be rewritten as

$$y^* = \sum_{i=1}^{N} \phi(\boldsymbol{a}, \boldsymbol{x}^*)^{\mathrm{T}} (\mathbf{H}^{\mathrm{T}}\mathbf{H} + \lambda_1 \mathbf{I})^{-1} \phi(\boldsymbol{a}, \boldsymbol{x}_i) t_i = \sum_{i=1}^{N} k(\boldsymbol{x}^*, \boldsymbol{x}_i) t_i \tag{4.18}$$

where $k(\boldsymbol{x}^*, \boldsymbol{x}_i) = \phi(\boldsymbol{a}, \boldsymbol{x}^*)^{\mathrm{T}} (\mathbf{H}^{\mathrm{T}}\mathbf{H} + \lambda_1 \mathbf{I})^{-1} \phi(\boldsymbol{a}, \boldsymbol{x}_i)$ is known as the equivalent kernel. It can be seen from (4.18) that the predictive output of the SLFN is a linear combination of t_i, the targets of the N training data. The shape of the equivalent kernel $k(\boldsymbol{x}^*, \boldsymbol{x}_i)$ will be largely affected by the norm of the input weights $\boldsymbol{a}$, which will further cast light on why neural networks with large weights are more likely to overfit.



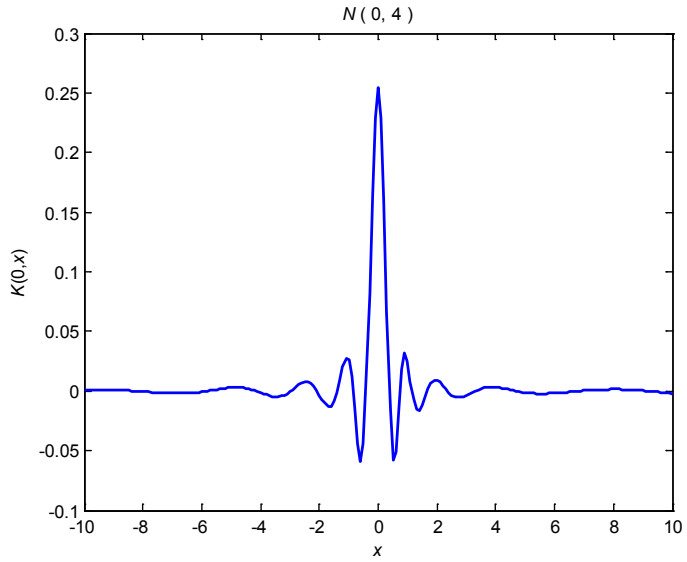


Figure 4.3 Equivalent kernel with different norm of input weights

Figure 4.3 above shows the shape of $k(0, \boldsymbol{x})$ under different norms of input weights. The curves from the top to the bottom correspond to a SLFN with different norms of input weights The first plot correspond to a SLFN with the largest input weights and the last plot correspond to a SLFN with the smallest input weights.

As shown in the first plot in figure 4.3, the smooth equivalent kernel will result in a flexible network that could take into account more distant data points. The network output is a weighted average of a lot of nearby training points. When the norm of the input weights increases, the equivalent kernel becomes more peaked and localized, as illustrated by the middle and bottom plot. In such cases, the SLFN will largely weigh only a few data points that are very close to the test sample $\boldsymbol{x}^*$, which might lead to overfitting. This figure partly explains why a SLFN with a smaller norm of weights is more likely to generalize well.

4.1.3 The learning steps of the Proposed Regression Method

Similarly to the classification model, the input weights will only be decayed once. It is worth mentioning that unlike in the classification model, the solution (4.16) for the output weights $\boldsymbol{w}$ is a one step calculation. No iteration is needed.

The learning steps of the proposed regression method can be summarized as follows

Learning steps of the proposed regression method

Begin

1. Initialize all parameters; randomly generate the inputs weights and bias.
2. Pre-calculate the output weights according to (4.16).
3. Calculate the network's outputs.
4. Calculate the delta term for the hidden nodes and output nodes according to (3.14) and (4.13), respectively.
6. Update the input weights according to (4.14)
7. Re-calculate the output weights according to (4.16)

End

4.1.4 Relationship with SVR

Following the same idea as in chapter 3, the proposed regression method uses explicit feature mapping instead of kernels. Similarly to the classification model, the output weights are directly calculated in the feature space and their dimensionality has to be limited due to the curse of dimensionality. However, in SVR, no such problem exists. In addition, the size of the Hessian in the proposed method, in most cases, will be much small than the Gram matrix in the kernel machines, so similarly to classification, the proposed regression method will gain much improvement in learning speed.

Moreover, in order to enforce sparsity, the ε-insensitive error, defined as follow, is used in SVR.

$$E(\xi)=\begin{cases}\xi-\varepsilon, & \xi>\varepsilon\\ -\xi-\varepsilon, & \xi<-\varepsilon\\ 0, & |\xi|\le\varepsilon\end{cases} \tag{4.19}$$

It can be learned from the ε-insensitive function that there will be no penalty for data points that $-\varepsilon<\xi_i<\varepsilon$. Deleting or moving such training non-support points within $(-\varepsilon,+\varepsilon)$ will not affect the final coefficients of SVR at all. Therefore, the parameter ε, together with the regularization parameter C and the inside kernel function, including the kernel

parameter(s), will select the SVR support vectors.

In the proposed regression method, as the result of the assumption made in (4.5), every training point will impact the final network output. Thus, from the perspective of SVR, every training data will become a support vector and the sparsity is lost in the final solution. However, we are also paid back by one benefit coming out of the proposed method: no parameter ε has to be estimated.

4.2 Experiment

In this section, the proposed regression model is tested and compared to several other machine learning algorithms on both synthetic and real world datasets. Similarly to the previous chapter, the kernel function for SVR is chosen to be Gaussian. For the SLFNs model, the hidden activation function is logistic sigmoid. The initial weights for ELM and the proposed approach are both randomly generated according to the uniform distribution in the interval (-1, 1).

The Root Mean Square Error (RMSE) is calculated to quantify the learning results. The definition of the RMSE is

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(t_i - y_i)^2}{N-1}} \tag{4.20}$$

where t_i is the desired output, y_i is the predicted result and N is the number of instances.

4.2.1 Synthetic datasets

The performance of the proposed approach is tested and evaluated on several synthetic benchmarking regression problems. The performances of some other related methods are also given for comparison purpose.

(1) SinC function approximation

The SinC function approximation is a famous one dimensional benchmarking regression problem, which is shown by figure 4.4. The definition of SinC is

$$f(x)=\begin{cases}\dfrac{\sin(x)}{x}, & x\neq 0\\ 1 \quad , & x=0\end{cases} \tag{4.21}$$

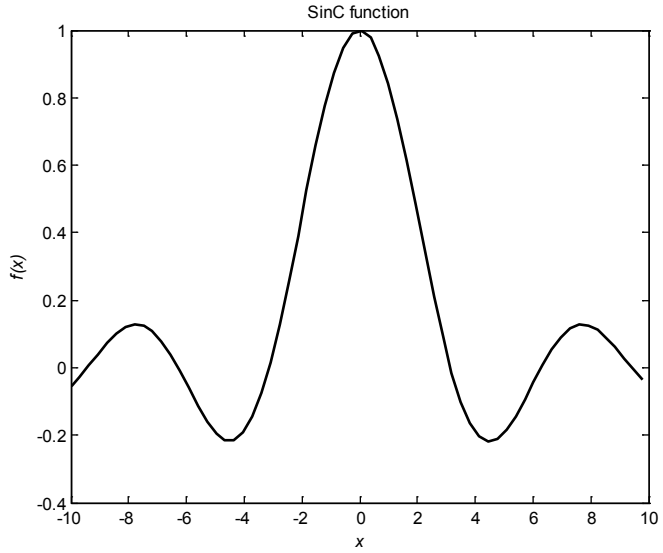


Figure 4.4 SinC function

A training set of 2000 instances (x_i, y_i) and a testing set of 1000 instances, are generated, where x_i are randomly drawn from the uniform distribution in the interval (-10, 10). To make the problem more challenging, zero-mean Gaussian noise with standard deviation 0.1 is added to the target of the training set. The testing set is noise free. The experiment is repeated 50 times. The average result and standard deviation are recorded.

Table 4.1 Comparison of different models on SinC approximation problem

Method	*Hidden nodes/SVs*	*Training RMSE*	*Std.*	*Testing RMSE*	*Std.*	*Training Time(s)*	*Std.*
BP	10	0.10007	0.00149	0.01528	0.00351	1.7803	0.8594
ELM	15	0.09968	0.00150	0.00934	0.00142	0.0168	0.0083
SVR	1066.54	0.09970	0.00182	0.00872	0.00164	194.48	23.50
Proposed Approach	20	0.09966	0.00161	**0.00761**	0.00147	0.0253	0.0104

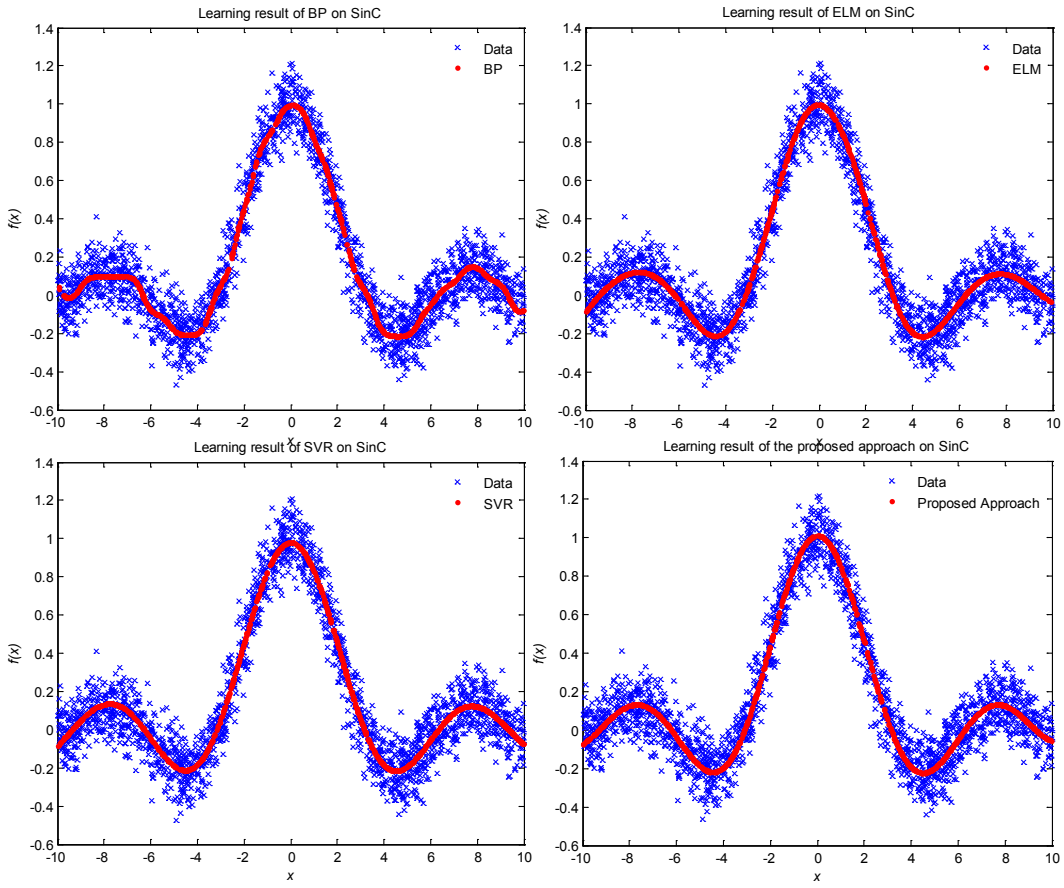


Figure 4.5 Comparison of SinC learning results

Table 4.1 shows the comparison of learning results. It is observed from the table that ELM, SVR and the proposed approach produce a similar training RMSE, around 0.097.

To evaluate the testing RMSE listed in table 4.1, paired *t*-tests are performed between the proposed approach and the other three methods. First we label the methods 1 to 4 in the order in which they appear in table 4.1. The result of the *t*-test returns three statistics, that are, t_{14}=14.2771, t_{24}=5.7255 and t_{34}=3.6360. By checking the *t*-test table, we find that at significance level α=0.05, the critical *t* value is 1.6776< t_{14}, t_{24}, t_{34}. Therefore, at the significance level α=0.05, the null hypothesis is rejected. The performance of the proposed approach, shown in bold, is better than that of the other three methods on the SinC data.

With respect to training time, the proposed method is much faster than BP and it is over one thousand times faster than SVR. The proposed method uses a larger network, so its learning speed is slightly slower than that of ELM.

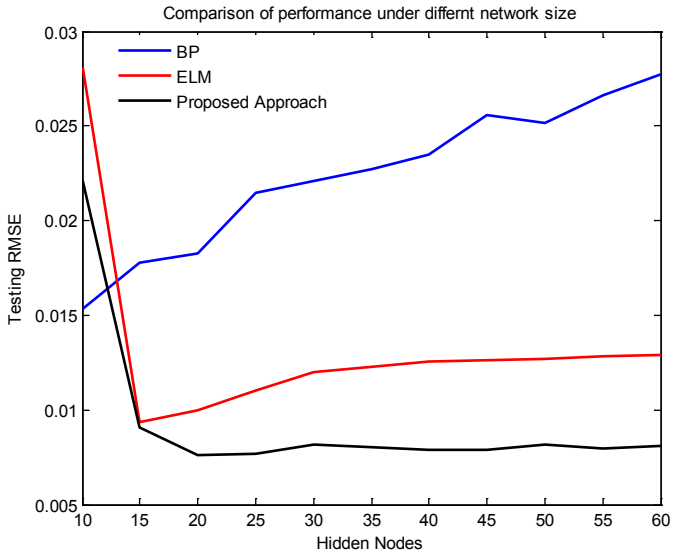


Figure 4.6 Comparison of testing RMSE under different network size

Figure 4.5 shows the learning results on different models. The upper two graphs display the prediction curves produced by BP and ELM. The lower two display prediction curves of SVR and the proposed approach. The blue dots are the training data points. The red dots are the predictive outputs. Similarly to figure 4.1, BP produced a waving and unstable output. By checking the final solution, we found that BP trapped the network in large weights and produces the worst prediction performance, which is in accord with the previous analysis.

The impact of network size on BP, ELM and the proposed approach is also studied on the SinC problem. The results are shown in figure 4.6. It is worth mentioning that when the network size is small (L=5), all three methods are unable to properly learn the SinC curve, so the comparison is conducted from L=10 to L=60. It is observed that the BP algorithm is the most sensitive to the network size as its performance degenerates very fast with the growth of network size. The learning result of ELM will also denigrate when the network grows, but the impact is much less profound than it is on BP. When the size of the network is large enough, L>15 in this case, the proposed approach will always produce the best result among the three methods and the performance is quite stable.

(2) Multi-dimensional Synthetic Data

Experiments were also conducted on four multi-dimensional Synthetic datasets taken

from [17]. The four cases are defined as

$$
\begin{aligned}
&dataset\ 1: y = 0.1\exp(4x_1) + \frac{4}{1+\exp[-20(x_2-50)]} + 3x_3 + 2x_4 + x_5 \\
&\qquad x_1, x_2, x_3, x_4 \in [-1,1]. \\
&dataset\ 2: y = 10\sin(\pi x_1 x_2) + 20(x_3 - 1/2)^2 + 10x_4 + 5x_5 \\
&\qquad x_1, x_2, x_3, x_4 \in [-1,1]. \\
&dataset\ 3: y = \sqrt{x_1^2 + (x_2 x_3 - \frac{1}{x_2 x_4})^2} \\
&\qquad x_1 \in [0,100], x_2 \in [40\pi, 560\pi], x_3 \in [0,1], x_4 \in [1,11]. \\
&dataset\ 4: y = \arctan\left(\frac{x_2 x_3 - (x_2 x_4)^{-1}}{x_1}\right) \\
&\qquad x_1 \in [0,100], x_2 \in [40\pi, 560\pi], x_3 \in [0,1], x_4 \in [1,11].
\end{aligned}
\tag{4.22}
$$

In each case, 300 training samples and 200 testing samples are generated. All the variables x_1 to x_4 are randomly generated according to uniform distribution. The target y is normalized to [0, 1]. Similarly to the previous SinC example, to make the problem more challenging, a zero-mean white Gaussian noise with standard deviation 0.1 is added to the training target y and the testing set remains noise free.

Before training, the inputs for dataset 3 and dataset 4 are normalized to [-1, 1]. No input normalization is needed for dataset 1 and dataset 2, as the input data are originally drawn from [-1, 1]. For SVR, the parameters C, γ, and ε are optimized from $\{2^{-5}, 2^{-4},\ldots,2^{5}\}$, $\{2^{-5}, 2^{-4},\ldots,2^{5}\}$ and $\{2^{-10}, 2^{-8},\ldots, 2^{0}\}$, respectively. The optimal parameters from the 726 combinations are reported in table 4.4. For the proposed approach, the combination of parameters (λ_1, η) are optimized from $\{2^{-15},2^{-9},\ldots,2^{5}\}$ and $\{2^{-10},2^{-9},\ldots,2^{0}\}$, respectively, so 231 different parameter combinations are tried. Similarly to Chapter 3, λ_2 is fixed at 0.5. In addition, the number of hidden nodes L is optimized for all SLFNs.

Table 4.2 shows the comparison for the training RMSE. Table 4.3 shows the training time and the testing RMSE is listed in table 4.5.

Table 4.2 Comparison of training RMSE and Std. on Multi-dimensional Synthetic Data

Dataset	*BP*		*ELM*		*SVR*		*Proposed approach*	
	RMSE	Std.	RMSE	Std.	RMSE	Std.	RMSE	Std.
1	0.10986	0.01134	0.10245	0.00457	0.10193	0.00443	0.10281	0.00430

2	0.10808	0.00809	0.09686	0.00426	0.09843	0.00408	0.09914	0.00355
3	0.10199	0.00616	0.09558	0.00324	0.09663	0.00424	0.09732	0.00402
4	0.11989	0.01105	0.10053	0.00511	0.14164	0.00641	0.10467	0.00512

Table 4.3 Comparison of training time and Std. on Multi-dimensional Synthetic Data

Dataset	*BP*		*ELM*		*SVR*		*Proposed approach*	
	Time(s)	Std.	Time(s)	Std.	Time(s)	Std.	Time(s)	Std.
1	0.7518	0.3356	0.0096	0.0099	1.1466	0.0932	0.0084	0.0078
2	0.6559	0.215	0.0150	0.0141	1.0866	0.0710	0.0112	0.007
3	0.6759	0.1389	0.0096	0.0129	1.0525	0.0765	0.0090	0.0114
4	0.6975	0.1748	0.0128	0.0087	0.9459	0.1031	0.0106	0.0080

Table 4.4 Comparison of model parameters on Multi-dimensional Synthetic Data

Dataset	*BP*	*ELM*	*SVR*	*Proposed approach*
1	L=5	L=40	$\gamma=2^{-5}$, $C=2^{5}$, $\varepsilon=2^{-4}$, SVs=175.14	L=40, $\eta=2^{0}$, $\lambda_1=2^{-10}$
2	L=6	L=50	$\gamma=2^{-4}$, $C=2^{4}$, $\varepsilon=2^{-4}$, SVs=171.28	L=50, $\eta=2^{0}$, $\lambda_1=2^{-10}$
3	L=5	L=30	$\gamma=2^{-5}$, $C=2^{5}$, $\varepsilon=2^{-4}$, SVs=162.74	L=30, $\eta=2^{0}$, $\lambda_1=2^{-10}$
4	L=7	L=50	$\gamma=2^{-3}$, $C=2^{3}$, $\varepsilon=2^{-4}$, SVs=189.74	L=50, $\eta=2^{0}$, $\lambda_1=2^{-15}$

Table 4.5 Comparison of testing RMSE and Std. on Multi-dimensional Synthetic Data

Dataset	*BP*		*ELM*		*SVR*		*Proposed approach*	
	RMSE	Std.	RMSE	Std.	RMSE	Std.	RMSE	Std.
1	0.06793	0.01652	0.06628	0.00614	<u>0.05319</u>	0.00452	<u>0.05352</u>	0.00541
2	0.06971	0.00975	0.06488	0.00785	0.05023	0.00562	**0.04753**	0.00415
3	0.04654	0.00931	0.03775	0.00471	<u>0.02694</u>	0.00532	<u>0.02608</u>	0.00416
4	0.08613	0.01513	0.07482	0.00975	0.10636	0.01078	**0.06765**	0.00983

Similarly to the previous section, paired t tests with significance level α=0.05 are conducted on table 4.5. The best result on each dataset that is statistically significant is shown in bold, and the insignificant results are underlined.

It is observed from table 4.5 that the proposed approach beats the other three methods on dataset 2 and dataset 4. It beats BP and ELM on dataset 1 and dataset 3, but tied with SVR.

In terms of learning speed, when using the same network size, the proposed approach produces a result that is similar to that of ELM. The proposed approach largely outperformed BP and SVR, even training a much larger network than BP. It is found from table 4.3 that, as a result of a small network, BP is able to learn faster than SVR.

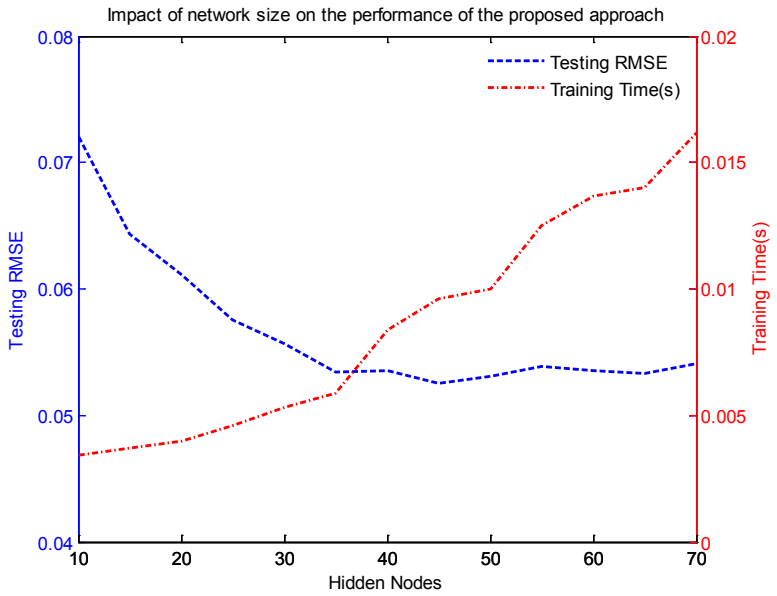


Figure 4.7 Impact of network size on the performance of the proposed approach

Figure 4.7 shows the performance of the proposed approach under various network sizes (L=10, 15,…, 70). The left Y-axis and the blue dash line illustrate the testing RMSE. The right Y-axis and the red dot dash line shows the training time. It is observed that similarly to the SinC approximation problem, when the number of hidden nodes is large enough, L>35 in this case, the learning result of the proposed approach will be quite stable. In another respect, the proposed approach will take more training time as the network size grows and the growth will be super-linear.

4.2.2 Real life datasets

In this section, the learning ability of the proposed approach will be tested on a number of real life regression problems.

(1) Time Series

Time series is a sequence of observation measured successively according to the time interval it is recorded. The task of time series prediction is to forecast the future values based on current and past observations. Therefore, time series prediction is a classic regression problem.

All time series datasets used in this experiment are from *Time Series Data Library*

[30], which is available online. The details of the time series datasets are listed in table 4.6

Table 4.6 Detail information of time series data

Dataset	*# observations*	*Granularity*	*Description*
1	468	Monthly from Jan. 1922 to Dec. 1960	Current river at Van Buren
2	365	Daily from 01 Jan. to 31 Dec. 1959	Total female births in California
3	192	Monthly from Jan. 1960 to Dec 1975	Gasoline demand in Ontario, Gallon millions
4	476	Monthly from Jan. 1956 to Aug. 1995	Production of blooms and slabs in Australia, Thousand Tonnes
5	528	Monthly from Jan. 1933 to Dec. 1976	Rain in Coppermine, mm
6	552	Monthly from Jan. 1915 to Dec. 1960	River flow in Mad river at Springfield, OH., cms
7	636	Monthly from Jan. 1924 to Dec. 1977	River flow in Pigeon river near middle fall ont., cms
8	100	Weekly	Demand for a plastic container (Montgomery & Johnson)
9	100	Weekly	Sales of a cutting tool (p.270, Montgomery: Fore. & T.S.)

Before applying learning algorithms, proper de-noising is necessary to pre-process the raw time series data. The moving average, a commonly used filter in time series analysis, is adopted to smooth the data [7]. The moving average can be expressed by

$$x(t) = \frac{x_0(t) + x_0(t-1) + \cdots + x_0(t-m+1)}{m} \tag{4.23}$$

where t is the time index, $x_0(t)$ is the original time series, $x(t)$ is the data after smoothing. Furthermore, after applying the moving average, the de-noised data is normalized into the interval [-1, 1].

Moreover, phase-space reconstruction has to be applied to turn the one dimensional data into a format that could be learned. The embedded data vector can be expressed as

$$\boldsymbol{d}(t) = [x(t), x(t-\tau), x(t-2\tau), \cdots, x(t-(m-1)\tau)]^{\mathrm{T}} \tag{4.24}$$

where τ is the time delay and m is the embedding dimension. In this experiment, the time

delay τ=1 and the embedding dimension m=6. The desired output is the next future value $y(t)=x(t+1)$, In this way, each value $y(t)$ is predicted given the past 6 observations contained in $\boldsymbol{d}(t)$.

For each dataset, the first 2/3 time series are used as training set and the later 1/3 data are used as testing set. Parameter optimization and other experimental settings remain the same as those used in the previous section (Multi-dimensional Synthetic Data).

Figure 4.8 is the line chart of raw time series data (on Dataset 1, Current river at Van Buren). Figure 4.9 is the line chart of the same dataset after pre-processing. It is found that compared to the raw time series data, the pre-processed data becomes much smoother and less noisy.

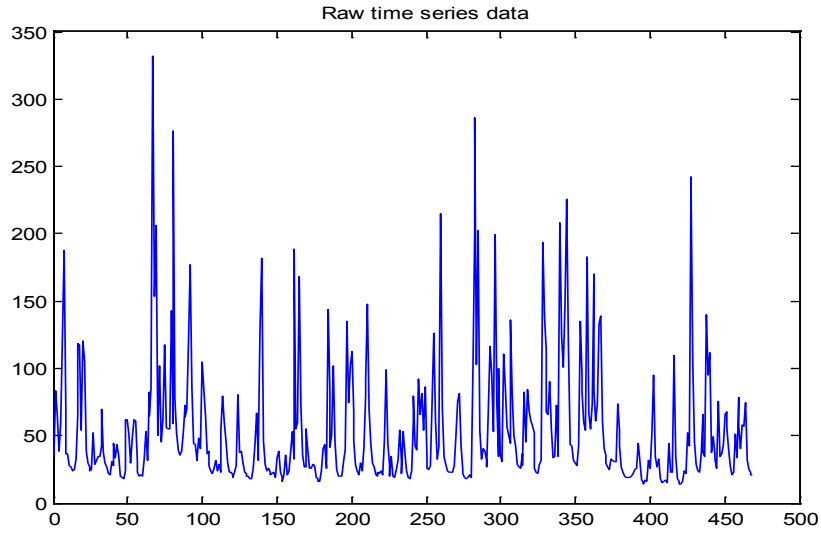


Figure 4.8 Raw time series data

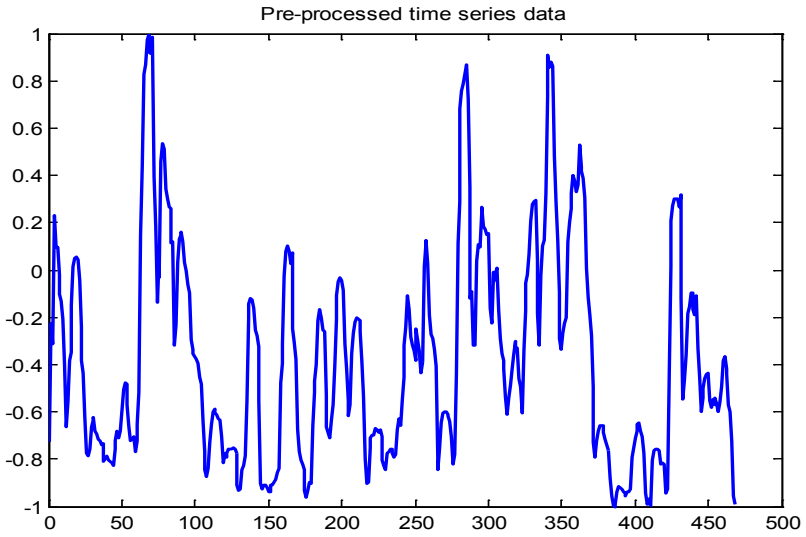


Figure 4.9 Pre-processed time series data

Table 4.7 Comparison of training RMSE on time series data

Dataset	*BP*	*ELM*	*SVR*	*Proposed Approach*
1	0.12155	0.11450	0.11428	0.11495
2	0.11119	0.10860	0.10901	0.10862
3	0.04180	0.04836	0.05071	0.04848
4	0.02654	0.02577	0.02581	0.02583
5	0.13471	0.12832	0.12417	0.12216
6	0.08992	0.08842	0.08628	0.08792
7	0.08088	0.07787	0.07847	0.07828
8	0.11235	0.10048	0.07810	0.08997
9	0.12065	0.10255	0.09665	0.11101

Table 4.8 Comparison of training time (s) on time series data

Dataset	*BP*	*ELM*	*SVR*	*Proposed Approach*
1	0.7534	0.0037	1.8594	0.0071
2	0.5493	0.0025	0.7812	0.0093
3	0.7203	0.0018	0.0937	0.0037
4	1.1547	0.0040	1.0156	0.0275
5	0.6090	0.0081	1.4219	0.0718
6	0.6093	0.0034	3.0469	0.0118
7	0.5887	0.0043	2.5469	0.0112
8	0.4921	0.0012	0.0625	0.0034
9	0.4678	0.0009	0.0625	0.0028

Table 4.9 Comparison of model parameters on time series data

Dataset	*BP*	*ELM*	*SVR*	*Proposed Approach*

1	L=2	L=15	$\gamma=2^{-5}$, $C=2^{5}$, $\varepsilon=2^{-8}$, SVs=308	L=30, $\eta=2^{0}$, $\lambda_1=2^{-7}$
2	L=3	L=10	$\gamma=2^{-5}$, $C=2^{4}$, $\varepsilon=2^{-10}$, SVs=239	L=50, $\eta=2^{0}$, $\lambda_1=2^{-8}$
3	L=3	L=10	$\gamma=2^{-5}$, $C=2^{4}$, $\varepsilon=2^{-4}$, SVs=33	L=30, $\eta=2^{0}$, $\lambda_1=2^{-8}$
4	L=2	L=20	$\gamma=2^{-2}$, $C=2^{3}$, $\varepsilon=2^{-6}$, SVs=146	L=100, $\eta=2^{-6}$, $\lambda_1=2^{-9}$
5	L=5	L=30	$\gamma=2^{-1}$, $C=2^{4}$, $\varepsilon=2^{-4}$, SVs=148	L=200, $\eta=2^{-1}$, $\lambda_1=2^{-14}$
6	L=10	L=15	$\gamma=2^{-3}$, $C=2^{5}$, $\varepsilon=2^{-8}$, SVs=343	L=50, $\eta=2^{-7}$, $\lambda_1=2^{-6}$
7	L=5	L=20	$\gamma=2^{-4}$, $C=2^{5}$, $\varepsilon=2^{-6}$, SVs=237	L=50, $\eta=2^{-9}$, $\lambda_1=2^{-8}$
8	L=7	L=10	$\gamma=2^{-4}$, $C=2^{5}$, $\varepsilon=2^{-4}$, SVs=43	L=50, $\eta=2^{0}$, $\lambda_1=2^{-9}$
9	L=3	L=10	$\gamma=2^{-2}$, $C=2^{2}$, $\varepsilon=2^{-6}$, SVs=56	L=50, $\eta=2^{0}$, $\lambda_1=2^{-3}$

Table 4.10 Comparison of testing RMSE on time series data

Dataset	*BP*	*ELM*	*SVR*	*Proposed Approach*
1	0.12064	0.11830	0.11093	0.11357
2	0.19983	0.14457	0.12203	0.11878
3	0.21768	0.19818	0.12729	0.09204
4	0.03062	0.03097	0.02933	0.02972
5	0.15834	0.15190	0.14685	0.14805
6	0.10636	0.09905	0.09595	0.09806
7	0.09658	0.09313	0.09047	0.09112
8	0.17200	0.12537	0.12346	0.11225
9	0.22517	0.14679	0.11022	0.11699

Table 4.7 compares the training RMSE produced by different models. The training times are listed in table 4.8. Table 4.9 shows the parameters used in each model. Table 4.10 is the comparison of testing RMSE.

Figure 4.10 shows the impact of parameters η and λ_1 on the performance of the proposed approach. This figure is the result obtained on dataset 1.

It is observed from table 4.10 that on the time series datasets, SVR produced the best prediction results, followed by the proposed approach. The proposed approach constantly beat the ELM and BP.

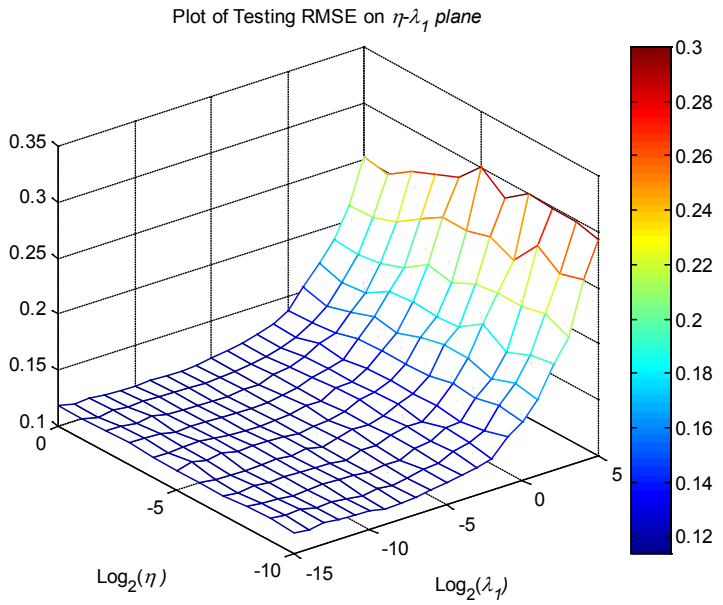


Figure 4.10 Plot of Testing RMSE of the proposed approach on η-λ_1 plane

In learning speed, the proposed approach is much faster than SVR and BP, and only slightly slower than ELM. From table 4.9, it is found that the proposed approach is training a network several times larger than ELM, which is the main reason why the proposed approach is slower. Similarly to the previous section, benefiting from a small network, BP takes less time to train than SVR on most of the datasets.

(2) Real life Benchmarking Regression Datasets

Experiments are conducted on several real life benchmarking regression datasets taken from the Belkent University Function Approximation Repository (some are originally from the UCI Machine Learning Repository) [18]. The detailed information of the datasets is listed in table 4.11. The number of both continuous and discrete attributes is listed and the number of total instances is also given.

It is worth mentioning that in the dataset Auto-Mpg, there were originally 398 instances, but there were 6 instances containing missing values which were removed in our experiment.

Table 4.11 Detail information of Real life Benchmarking Regression Datasets

Dataset	*Attributes*		*# instances*
	Continuous	Discrete	
Auto-Mpg	4	3	392

Baseball	2	14	337
Breast Cancer	32	0	194
Home Run Race	0	19	163
Housing	12	1	506
Machine CPU	6	0	209
Mortgage	15	0	1049
Northridge Earthquake	5	5	2929
Plastic	1	1	1650
Quake	2	1	2178
Stock	9	0	950
Weather Ankara	9	0	1609
Weather Izmir	9	0	1461

Before applying the learning algorithms, the attribute values are normalized to [-1, 1] and the target value is normalized to [0, 1]. Parameter optimization and other experimental settings remain the same as those used in the previous section. The average result and the standard deviation, shown in brackets, of 5×10-fold cross-validation are recorded.

Table 4.12 shows the comparison of training RMSE. The learning speed is listed in table 4.13. Information about the parameters used in the different learning algorithms is reported in table 4.14. Table 4.15 compares the validation RMSE obtained by different models.

Table 4.12 Comparison of training RMSE on Real life Benchmarking Regression Datasets

Dataset	*BP*	*ELM*	*SVR*	*Proposed Approach*
Auto-Mpg	0.06672	0.06109	0.06034	0.05599
	(0.00576)	(0.00181)	(0.00145)	(0.00156)
Baseball	0.10842	0.09931	0.10114	0.09836
	(0.01093)	(0.00344)	(0.00240)	(0.00214)
Breast Cancer	0.25624	0.23948	0.24676	0.24439
	(0.0222)	(0.00461)	(0.00436)	(0.00287)
Home Run Race	0.15235	0.12529	0.12571	0.12433
	(0.01857)	(0.0066)	(0.00455)	(0.00398)
Housing	0.08089	0.06062	0.04956	0.04133
	(0.01382)	(0.00264)	(0.00226)	(0.00144)
Machine CPU	0.05862	0.02882	0.03199	0.03042
	(0.02128)	(0.00216)	(0.00176)	(0.00141)

Mortgage	0.00475	0.00283	0.00239	0.00267
	(0.00007)	(0.00009)	(0.00005)	(0.00005)
Northridge	0.10250	0.10109	0.10191	0.10255
Earthquake	(0.00470)	(0.00339)	(0.00323)	(0.00398)
Plastic	0.14640	0.14610	0.14653	0.14617
	(0.00109)	(0.00075)	(0.00081)	(0.00085)
Quake	0.17126	0.17117	0.17502	0.17034
	(0.00122)	(0.00106)	(0.00137)	(0.00116)
Stock	0.02452	0.01661	0.01429	0.01659
	(0.00187)	(0.00040)	(0.00020)	(0.00030)
Weather Ankara	0.01612	0.01584	0.01465	0.01479
	(0.00083)	(0.00021)	(0.00013)	(0.00010)
Weather Izmir	0.01772	0.01745	0.01668	0.01516
	(0.00087)	(0.00023)	(0.00022)	(0.00016)

Table 4.13 Comparison of training time(s) on Real life Benchmarking Regression Datasets

Dataset	*BP*	*ELM*	*SVR*	*Proposed Approach*
Auto-Mpg	0.8075	0.0159	1.0341	0.0481
	(0.1416)	(0.0066)	(0.0406)	(0.0099)
Baseball	0.7937	0.0103	1.6138	0.0284
	(0.1678)	(0.0081)	(0.0935)	(0.0087)
Breast Cancer	0.7243	0.0025	0.3418	0.0184
	(0.1202)	(0.0057)	(0.0302)	(0.0072)
Home Run Race	0.6146	0.0062	0.2775	0.0141
	(0.1228)	(0.0077)	(0.0750)	(0.0071)
Housing	0.9906	0.0731	5.3703	0.5040
	(0.2842)	(0.0168)	(0.2050)	(0.0397)
Machine CPU	0.6325	0.0025	0.5303	0.0078
	(0.1662)	(0.0057)	(0.0418)	(0.0078)
Mortgage	2.5962	0.8803	48.0656	0.8943
	(1.0850)	(0.0636)	(2.7919)	(0.0508)
Northridge	0.6434	0.0025	0.3721	0.0090
Earthquake	(0.1134)	(0.0057)	(0.0351)	(0.0077)
Plastic	1.0962	0.0175	95.9813	0.8503
	(0.3690)	(0.0060)	(7.263)	(0.0386)
Quake	1.2881	0.0093	178.5219	0.3509
	(0.5152)	(0.0094)	(1.3805)	(0.0234)
Stock	3.0469	1.5159	50.0844	0.8262
	(1.0792)	(0.0719)	(1.7707)	(0.0539)
Weather Ankara	2.4756	0.1331	115.9538	0.7571
	(0.7929)	(0.0214)	(3.6668)	(0.0258)
Weather Izmir	1.8797	0.1212	94.0656	1.0303

	(0.7192)	(0.0190)	(1.5964)	(0.0401)

Table 4.14 Comparison of model parameters on Real life Benchmarking Regression Datasets

Dataset	*BP*	*ELM*	*SVR*	*Proposed Approach*
Auto-Mpg	L=10	L=50	$\gamma=2^{-1}$, $C=2^{-1}$, $\varepsilon=2^{-4}$, SVs=104.52	L=150, $\eta=2^{-8}$, $\lambda_1=2^{-9}$
Baseball	L=5	L=35	$\gamma=2^{-5}$, $C=2^{0}$, $\varepsilon=2^{-10}$, SVs=294.02	L=100, $\eta=2^{-6}$, $\lambda_1=2^{0}$
Breast Cancer	L=5	L=20	$\gamma=2^{-5}$, $C=2^{-3}$, $\varepsilon=2^{-4}$, SVs=150.58	L=100, $\eta=2^{-1}$, $\lambda_1=2^{4}$
Home Run Race	L=5	L=35	$\gamma=2^{-5}$, $C=2^{-1}$, $\varepsilon=2^{-6}$, SVs=133.82	L=100, $\eta=2^{0}$, $\lambda_1=2^{-1}$
Housing	L=5	L=85	$\gamma=2^{-2}$, $C=2^{1}$, $\varepsilon=2^{-6}$, SVs=328.96	L=500, $\eta=2^{-3}$, $\lambda_1=2^{-8}$
Machine CPU	L=10	L=20	$\gamma=2^{-4}$, $C=2^{2}$, $\varepsilon=2^{-10}$, SVs=177.52	L=50, $\eta=2^{-6}$, $\lambda_1=2^{-6}$
Mortgage	L=10	L=220	$\gamma=2^{-1}$, $C=2^{3}$, $\varepsilon=2^{-10}$, SVs=662.86	L=500, $\eta=2^{-10}$, $\lambda_1=2^{-15}$
Northridge Earthquake	L=5	L=15	$\gamma=2^{-5}$, $C=2^{-1}$, $\varepsilon=2^{-4}$, SVs=96.78	L=50, $\eta=2^{0}$, $\lambda_1=2^{0}$
Plastic	L=5	L=20	$\gamma=2^{0}$, $C=2^{5}$, $\varepsilon=2^{-4}$, SVs=966.18	L=400, $\eta=2^{-10}$, $\lambda_1=2^{-15}$
Quake	L=5	L=10	$\gamma=2^{-3}$, $C=2^{-3}$, $\varepsilon=2^{-4}$, SVs=1262.46	L=200, $\eta=2^{-4}$, $\lambda_1=2^{-6}$
Stock	L=25	L=300	$\gamma=2^{0}$, $C=2^{2}$, $\varepsilon=2^{-8}$, SVs=734.90	L=500, $\eta=2^{-10}$, $\lambda_1=2^{-15}$
Weather Ankara	L=5	L=65	$\gamma=2^{-3}$, $C=2^{3}$, $\varepsilon=2^{-8}$, SVs=1131.14	L=500, $\eta=2^{-2}$, $\lambda_1=2^{-8}$
Weather Izmir	L=10	L=65	$\gamma=2^{-3}$, $C=2^{3}$, $\varepsilon=2^{-8}$, SVs=1031.58	L=500, $\eta=2^{-2}$, $\lambda_1=2^{-7}$

Table 4.15 Comparison of validation RMSE on Real life Benchmarking Regression Datasets

Dataset	*BP*	*ELM*	*SVR*	*Proposed Approach*
Auto-Mpg	0.07794	0.07359	0.06702	0.06989
	(0.01486)	(0.01454)	(0.01299)	(0.01262)
Baseball	0.12969	0.11712	0.11246	0.11062
	(0.02639)	(0.02284)	(0.02264)	(0.01880)
Breast Cancer	0.29184	0.26398	0.25461	0.25362
	(0.04915)	(0.03826)	(0.03354)	(0.02567)
Home Run Race	0.17846	0.16057	0.14712	0.14221
	(0.03289)	(0.03463)	(0.03413)	(0.03136)
Housing	0.09558	0.08412	0.06712	0.06874

	(0.02368)	(0.01915)	(0.02138)	(0.01421)
Machine CPU	0.0790	0.05066	0.03703	0.03905
	(0.05626)	(0.03125)	(0.02296)	(0.01956)
Mortgage	0.00564	0.00509	0.00421	0.00423
	(0.00093)	(0.00122)	(0.00071)	(0.00084)
Northridge	0.11329	0.10438	0.10124	0.10102
Earthquake	(0.02614)	(0.02805)	(0.02785)	(0.03154)
Plastic	0.14691	0.14733	0.14735	0.14706
	(0.00896)	(0.00674)	(0.00760)	(0.00757)
Quake	0.17217	0.17176	0.17536	0.17154
	(0.01029)	(0.00958)	(0.01212)	(0.01048)
Stock	0.03158	0.02836	0.02345	0.02589
	(0.00367)	(0.00308)	(0.00226)	(0.00286)
Weather Ankara	0.01693	0.01772	0.01632	0.01644
	(0.00149)	(0.00275)	(0.00145)	(0.00101)
Weather Izmir	0.01914	0.01874	0.01784	0.01809
	(0.00281)	(0.0020)	(0.00209)	(0.00220)

It is observed that the learning speed of the proposed approach is constantly faster than that of BP and SVR, on all datasets, even though the proposed approach is training a network that is dozens, or in some cases, hundred of times (dataset Weather Ankara and Housing) larger than BP. Similarly to learning results obtained in previous experiments, the proposed approach is slower than ELM on most of the datasets, which is mainly because the network used in the proposed approach is larger than that of ELM. However, the proposed approach is still able to learn faster than ELM on one dataset (Stock). In the real life benchmarking regression datasets, when training the same network, the proposed approach does not lose to ELM in training speed, which agrees with the results observed in table 4.3.

It can be found from table 4.15 that in most of the cases, either the proposed approach or SVR produces the best results. To better evaluate the results in table 4.15, statistical tests are conducted. Similarly to chapter 3, Friedman's test is first carried out The Friedman's test returns a χ_F^2 value of 21.92 and a p-value of 6.76×10^{-5} <0.01, so we reject the null hypothesis that all methods perform equally on the benchmarking regression datasets.

Based on the result of Friedman's test, Nemenyi's test is conducted to further

analyze the result. Nemenyi's test will use the information on the rank of each method on the datasets. The mean ranks for the four methods are, BP=3.6154, ELM=3, SVR=1.7692 and the Proposed Approach=1.6153.

From Nemenyi's test, we obtain the q statistics between the proposed approach and the other three methods, which are 3.9497 (with BP), 2.7344 (with ELM) and 0.3038 (with SVR). The critical value of q_α is 2.72, for significant level α=0.05 and degree of freedom, df=36. Therefore, we conclude that the null hypotheses that the proposed approach is equivalent to BP and ELM on the benchmarking regression datasets are rejected, at significant level 0.05. The null hypothesis that the proposed approach tied with SVR is accepted.

Therefore, we can conclude that on the real life benchmarking regression datasets, in term of prediction performance, the proposed approach outperformed the two neural network based methods BP and ELM, and tied with SVR.

4.3 Conclusions

Similarly to the classification model introduced previously, the proposed regression model uses explicit feature mapping via building a SLFN. Due to the zero mean Gaussian noise assumption in (4.1), the output weights can be directly calculated in closed form, instead of through iterative computation. However, every data point will be included in the objective function and sparsity is lost.

Compared to neural network based methods, the proposed approach does not only focus on training error. The norm of the network weights is taken into account during the training process, so it is able to outperform both BP and ELM. In terms of learning speed, the proposed approach is always faster than BP. Even compared to ELM under that same network size, owing to the direct output layer calculation, the learning speed of the proposed approach is not inferior.

In addition, compared to SVR, the proposed approach is able to win on synthetic datasets and get a draw on benchmarking datasets. The inferior performance on the time series datasets is probably due to the limitation of explicit feature mapping previously

discussed in chapter 3. In terms of learning speed, the proposed approach constantly outperformed SVR, and was sometimes over 100 times faster.

Chapter 5

Conclusion and Future Work

In this research, a novel machine learning method based on Artificial Neural Networks is proposed. Based on the explicit feature mapping idea, both regression and classification models are built.

5.1 Summary

This work begins by first introducing the application of this study, Mine Countermeasure Missions as well as Autonomous Underwater Vehicles.

In the second chapter, some existing machine learning methods related to this research are reviewed. The MLP and the most commonly used training method, Error Backpropagation is introduced. Then, the extreme learning approach, a recently proposed learning algorithm based on SLFN is reviewed. The approximation ability of SLFN and extreme learning machine is also introduced. Next, the idea of kernel methods and one of its representatives, SVM, is covered and discussed.

In the third chapter, a novel learning algorithm based on SLFN is proposed. The importance of the norm of the weights is discussed and properly addressed in the proposed training process. The proposed approach leads to a sparse and robust classification model. In the fourth chapter, based on the same idea, the proposed approach is extended to regression. The proposed approach is tested and compared to several related machine learning algorithms on a wide variety of regression and classification problems, including synthetic and real life cases as well as the MCMs dataset.

5.2 Conclusion

In the proposed approach, the input space is mapped to another feature space via a hidden layer and the proposed method computes a linear model in that feature space. Different from traditional neural networks, the proposed approach does not extract features and, usually, the number of hidden nodes will be larger than that of traditional neural networks.

Traditional MLP treats classification problems as regression and even data points lying on the rights side of the decisional boundary may also be penalized. The proposed classification model is a maximum margin classifier. Only data points that lie on the wrong side of the decision boundary and inside the margin will be considered when building the separating (hyper) plane. Therefore, the resulting model will be a sparse learning model.

Training a MLP is in essence solving for a set of weights given the training data. Traditional MLP spend much effort trying to find the (local) minima, but the weight itself is ignored. The norm of the weights is important to the generalization ability of the MLP. The MLP with small weights is more likely to generalize well and the MLP with large weights is more likely to overfit. In the proposed approach, the norm of weights is properly addressed in the training process.

When the size of the training set is very large, traditional kernel methods could be practically infeasible. This is because feature mapping in kernel methods is implicit and the feature space computation can only be implemented via pair-wise evaluation of the kernel function. In the proposed approach, no kernel is used, so such problem is successfully avoided.

The results show that the proposed approach is able to produce a prediction performance close to that of the kernel methods, and better than that of other ANN based methods.

In terms of learning speed, the proposed approach will be much faster than traditional MLP and kernel methods. The reasons are two-fold. Firstly, benefiting from the explicit hidden layer feature mapping, the proposed approach is able to conduct the

computation directly in the feature space. Secondly, the proposed approach does not tediously tune the hidden layer parameter. In the MCMs, the application considered in this study, the proposed approach will also have a faster classification speed than traditional kernel methods.

All in all, the following conclusions can be drawn from this study.

- The norm of weights is important for the generalization performance of SLFN, and it should not be a random value that is independent of the data.

When training on the same dataset, the norm of input weights will decide the nonlinearity of the function the network is producing. The norm of the output weights is directly related to the width of the separating margin. Therefore, they should be taken into account in the network training process.

Moreover, the input weights will multiply the data value to produce the input for the hidden layer. The working region of the sigmoid function used in the hidden layer is decided by such input, so in the proposed approach, the input weights are not predetermined before seeing the data and they are not random numbers independent of the training data.

- The hidden layer of SLFN may not be used as feature extractors. Compared to kernels, using explicit hidden layer feature mapping can result in a faster learning method with little or no loss in prediction performance.

From common belief, the hidden layer should be used as feature extractors and the network is usually funnel shaped. However, from the proposed approach, we can see that, if properly trained, the size of the hidden layer can be much larger than that of the input layer.

The explicit feature projection enables a direct feature space computation. The learning process will be sped up and the classification speed will also be under control. Moreover, compared to kernel machines, when training on very large datasets, no large Gram matrix has to be constructed. The proposed method implements the feature mapping idea via constructing a SLFN, while in kernel machines the kernel feature mapping is implicitly realized by kernel functions. Therefore the kernel method and artificial neural network is bridged at this point.

- The Huber-like loss function will result in a spare classification model and speed up the training process.

Different from traditional MLP in which every training point is included in the error function, only parts of the training data, like the support vectors in SVMs, will be penalized by the Huber-like loss function proposed in this study. The final classification boundary is only built on such data points, and simply removing other points will not affect the decision boundary. The training can be sped up with the sparse representation.

5.3 Future work

On the machine learning side, there are several possible directions for further work on this study.

As a neural network method, the most appropriate number of hidden nodes for a given problem is always an open question. It has been previously mentioned that the proposed approach uses a larger hidden layer than traditional networks, but it is unnecessary if the size of the hidden layer is too large. A large hidden layer will compromise the speed of the proposed methods.

Furthermore, the hidden layer parameters, input weights and hidden bias, as well as the hidden layer activation function will affect the feature mapping. In this study, the input weights and hidden bias are randomly generated before seeing the training data. In Error Backpropagation, the initial condition of the weights will have some impact on the final solution, so it might be possible to generate the input weights according to certain statistical property of the training data (after seeing the data).

Moreover, only SLFN with a sigmoid hidden activation function is considered in this study, a hidden layer with other activation functions, such as linear, RBF or even a mixture of several functions could be further studied. Besides, the proposed approach is not limited to SLFN. It can be extended to multiple hidden layer cases.

In addition, the parameters of the proposed, such as λ_1 and η, are currently optimized through cross validation. It might be possible to include such parameters in the training process and conduct the optimization together with the network weights.

Finally on the application side, given the huge imbalance of the MCMs dataset, the idea of one class learning can be incorporated into the proposed approach in the future work. Currently, the AUVs are trained off-line before set out. Considering the fact that the sonar images in the MCMs are collected one by one (by AUVs), the proposed approach could be extended to an on-line version so that the intelligence of the AUVs could be improved after they are sent out.

Bibliography

1. Achlioptas, D., McSherry, F., Scholkopf, B.: Sampling Techniques for Kernel Methods. In: Advances in Neural Information Processing Systems. MIT Press, (2002)

2. Atiya, A., Chuanyi, J.: How initial conditions affect generalization performance in large networks. Neural Networks, IEEE Transactions on 8, 448-451 (1997)

3. Avci, E., Coteli, R.: A new automatic target recognition system based on wavelet extreme learning machine. Expert Systems with Applications 39, 12340-12348 (2012)

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

5. Bishop, C.M.: neural network for pattern recognition. Oxford University Press, Oxford, England (1996)

6. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer (2006)

7. Brockwell, P.J., Davis, R.A.: Introduction to Time Series and Forecasting. Springer (2002)

8. Buhmann, M.D.: Radial Basis Functions: Theory and Implementations. Cambridge University Press (2003)

9. Canu, S., Grandvalet, Y., Guigue, V., Rakotomamonjy, A.: SVM and Kernel Methods Matlab Toolbox Perception Systèmes et Information, INSA de Rouen, Rouen, France (2005)

10. Caruana, R., Lawrence, S., Giles, C.L.: Overfitting in Neural Nets: Backpropagation, Conjugate Gradient, and Early Stopping. In: Neural Information Processing Systems pp. 402-408. (2000)

11. Chauvin, Y., Rumelhart, D.: backpropagation theory architectures and applications. Hillsdale, NJ: Erlbaum (1995)

12. Cortes, C., Vapnik, V.: Support-vector networks. Machine learning 20, 273-297 (1995)

13. Fernandez, J.E., Christoff, J.T., Cook, D.A.: Synthetic aperture sonar on AUV. In: OCEANS 2003. Proceedings, pp. 1718-1722 Vol.1713. (2003)

14. Fine, S., Scheinberg, K.: Efficient SVM training using low-rank kernel representations. Journal of Machine Learning Research 2, 243-264 (2001)

15. Frénay, B., Verleysen, M.: Parameter-insensitive kernel in extreme learning for non-linear support vector regression. Neurocomputing 74, 2526-2531 (2011)

16. Frank, A., Asuncion, A.: UCI Machine Learning Repository. (2010) http://archive.ics.uci.edu/ml

17. Friedman, J.H.: Multivariate Adaptive Regression Splines. The annals of statistics 19, 1-67 (1991)

18. Güvenir, H.A., Uysal, I.: Bilkent University Function Approximation Repository. (2000) http://funapp.cs.bilkent.edu.tr

19. Geman, S., Bienenstock, E., Doursat, R.: Neural networks and the bias/variance dilemma. Neural computation 4, 1-58 (1992)

20. Glynn, J.M., Buffman, M.: A side scan sonar system for autonomous underwater vehicles. In: Autonomous Underwater Vehicle Technology, 1996. AUV '96., Proceedings of the 1996 Symposium on, pp. 154-159. (1996)

21. Gonzalez, R., Woods, R.: Digital Image Processing. Addison Wesley (1992)

22. Hong, Z., Jie, Z., Kai, W., Zhipeng, B., Aixie, L.: A GA-ANN model for air quality predicting. In: Computer Symposium (ICS), 2010 International, pp. 693-699. (2010)

23. Hornik, K., Stinchcombe, M., White, H.: Multilayer feedforward networks are universal approximators. Neural Networks 2, 359-366 (1989)

24. Hosmer, D.W.: Applied Logistic Regression Wiley (2000)

25. Huang, G.-B., Ding, X., Zhou, H.: Optimization method based extreme learning machine for classification. Neurocomputing 74, 155-163 (2010)

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

27. Huang, G.B., Zhou, H., Ding, X., Zhang, R.: Extreme learning machine for regression and multiclass classification. IEEE Transactions on Systems, Man, and Cybernetics—PART B: Cybernetics 42, 513-529 (2012)

28. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: theory and applications. Neurocomputing 70, 489-501 (2006)

29. Huber, P.J.: Robust Statistics. John Wiley & Sons, Inc., NJ (1981)

30. Hyndman, R.: Time Series Data Library. http://datamarket.com/data/list/?q=provider:tsdl

31. Imanieh, M.J.M., Malekjamshidi, Z.: Design, simulation and implementation of a Full Bridge Series-Parallel Resonant DC-DC converter using ANN controller. In: Control and Communications (SIBCON), 2011 International Siberian Conference on, pp. 97-103. (2011)

32. Jaeger, H.: The" echo state" approach to analysing and training recurrent neural networks-with an erratum note'. Technical Report GMD Report 148, German National Research Center for Information Technology (2001)

33. Jaeger, H.: Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. Science 304, 78-80 (2004)

34. Japkowicz, N., Shah, M.: Evaluating Learning Algorithms: A Classification Perspective. Cambridge University Press (2011)

35. Jayakumar, S., Kanna, R.: Inspection system for detecting defects in a transistor using Artificial neural network (ANN). In: Communication and Computational Intelligence (INCOCCI), 2010 International Conference on, pp. 76-81. (2010)

36. Johansson, E.M., Dowla, F.U., Goodman, D.M.: Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method. International Journal of Neural Systems 02, 291-301 (1991)

37. Karush, W.: Minima of Functions of Several Variables with Inequalities as Side

Constraints. Department of Mathematics, vol. M.Sc. University Chicago, Chicago (1939)

38. Keerthi, S., Duan, K., Shevade, S., Poo, A.: A Fast Dual Algorithm for Kernel Logistic Regression. Machine learning 61, 151-165 (2005)

39. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Neural Networks, 1995. Proceedings., IEEE International Conference on, pp. 1942-1948 vol.1944. (1995)

40. Komarek, P.: Logistic Regression for Data Mining and High-Dimensional Classification. Dept. of Math Sciences, Carnegie Mellon University, (2004)

41. Kuhn, H.W., Tucker, A.W.: Nonlinear Programming. In: Proceedings of 2nd Berkeley Symposium, pp. 481-492. University of California Press, (1951)

42. Lawrence, S., Giles, C.L.: Overfitting and neural networks: conjugate gradient and backpropagation. In: Neural Networks, 2000. IJCNN 2000, Proceedings of the IEEE-INNS-ENNS International Joint Conference on, pp. 114-119 vol.111. (2000)

43. Li, K., Deng, J., He, H., Du, D.: Compact Extreme Learning Machines for biological systems. International journal of computational biology and drug design 3, 112-132 (2010)

44. McCulloch, W., Pitts, W.: A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biology 5, 115-133 (1943)

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

46. Mitchell, T.M.: Machine learning. McGraw-Hill (1997)

47. Orr, M.: Introduction to Radial Basis Function Networks. Institute for Adaptive and Neural Computation, Edinburgh University (1996)

48. Parviainen, E., Riihim ki, J., Miche, Y., Lendasse, A.: Interpreting Extreme Learning Machine as an approximation to an infinite neural network. In: KDIR 2010: Proceedings of the International Conference on Knowledge Discovery and Information Retrieval (2010)

49. Rahimi, A., Recht, B.: Random Features for Large-Scale Kernel Machines. In: Neural Information Processing Systems (NIPS), pp. 1177-1184. (2007)

50. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Nature 323, 533-536 (1986)

51. Sariel, S., Balch, T., Erdogan, N.: Naval Mine Countermeasure Missions. Robotics & Automation Magazine, IEEE 15, 45-52 (2008)

52. Schölkopf, B., Smola, A., Müller, K.-R.: Nonlinear Component Analysis as a Kernel Eigenvalue Problem. Neural Computation 10, 1299-1319 (1998)

53. Shao, H., Japkowicz, N.: Applying Least Angle Regression to ELM. twenty fifth Canadian Conference on Artificial Intelligence, pp. 170-180. Springer, Toronto (2012)

54. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press (2004)

55. Siantidis, K., Hölscher-Höbing, U.: A System For Automatic Detection And Classification for A Mine Countermeasure AUV. 3rd Underwater Acoustics Measurements Conference, Nafplion, Greece (2009)

56. Soria-Olivas, E., Gomez-Sanchis, J., Martin, J.D., Vila-Frances, J., Martinez, M., Magdalena, J.R., Serrano, A.J.: BELM: Bayesian Extreme Learning Machine. Neural Networks, IEEE Transactions on 22, 505-509 (2011)

57. Steinwart, I., Hush, D., Scovel, C.: An Explicit Description of the Reproducing Kernel Hilbert Spaces of Gaussian RBF Kernels. Information Theory, IEEE Transactions on 52, 4635-4643 (2006)

58. Suykens, J.A.K., Gestel, T.V., Brabanter, J.D., Moor, B.D., Vandewalle, J.: Least Squares Support Vector Machines. World Scientific Pub. Co., Singapore (2002)

59. Swets, J.A.: Signal detection theory and ROC analysis in psychology and diagnostics : collected papers. Lawrence Erlbaum Associates, Mahwah, NJ (1996)

60. Vapnik, V.: The Nature of Statistical Learning Theory. Springer (1995)

61. Wang, L.P., Wan, C.R.: Comments on "The Extreme Learning Machine. Neural Networks, IEEE Transactions on 19, 1494-1495 (2008)

62. Williams, C.K.I.: Computation with Infinite Neural Networks Neural Computation 10, 1203-1216 (1998)

63. Yuan, Y., Wang, Y., Cao, F.: Optimization approximation solution for regression problem based on extreme learning machine. Neurocomputing 74, 2475-2482 (2011)

64. Zong, W., Zhou, H., Huang, G.-B., Lin, Z.: Face Recognition Based on Kernelized Extreme Learning Machine. Autonomous and Intelligent Systems 6752, 263-272 (2011)