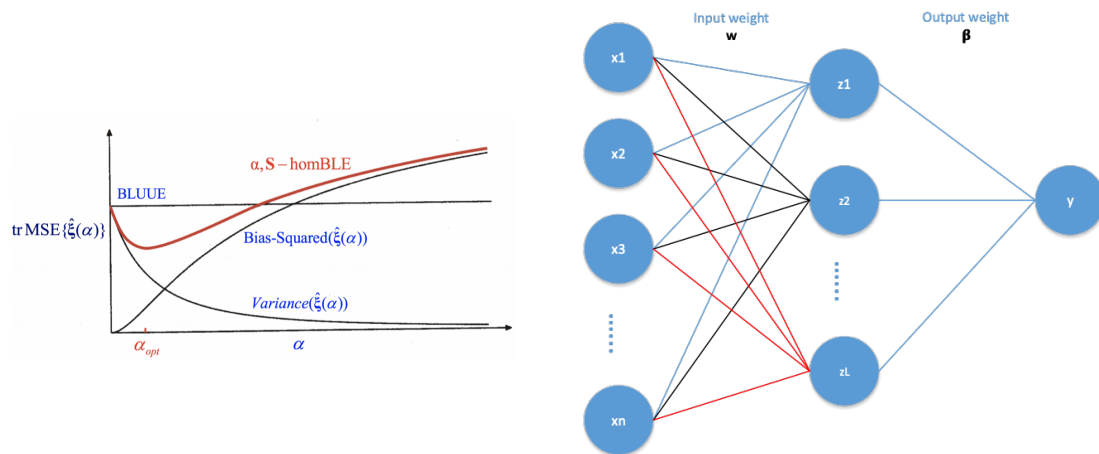


The Optimal Regularization and its Application in Extreme Learning Machine for Regression Analysis and Multi-class Classification



Master Thesis
Geodesy and Geoinformation
University of Stuttgart

Kun Qian

Stuttgart, October 2018

Supervisor: Dr. -Ing. Jianqing Cai
University of Stuttgart

Prof. Dr.-Ing. Nico Sneeuw
University of Stuttgart

Erklärung der Urheberschaft

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Arbeit ohne Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form in keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Ort, Datum

Unterschrift

Abstract

Extreme Learning Machine (ELM) proposed by Huang et al. (2006) is a newly developed single layer feed-forward neural network (SLFN). It is attractive for its high training efficiency and satisfactory performance, especially when dealing with a large amount of data, which are often in high-dimensional space. However, current ELM cannot solve the over-fitting problem among other several problems. While minimizing residuals of output errors for the training data, it tends to generate an over-fitting model, whose generalization ability is relatively weak. Even if the model fits the training data perfectly, it performs unsatisfactory for the testing data. In training process, we aim to minimize residuals of output errors of training data. It tends to generate an over-fitting model, which has poor generalization ability. The model maybe fit the training data perfectly, but performs bad in testing data. Furthermore, in order to improve accuracy, the traditional way is increasing the number of hidden-layer neurons, but excessive hidden-layer neurons result in an ill-posed normal matrix and a model which is over sensitive to the change of the training data. In such case, the performance of ELM is significantly affected by the outliers in the training data. In order to overcome these problems, we apply the regularization to the original ELM. In this study, the A-optimal design regularization is performed to improve the generalization ability and stability of ELM. The performance of ELM with the A-optimal design regularization will be evaluated through two main applications, respectively, regression analysis and satellite image multi-class classification.

Contents

List of Figures	XI
List of Tables	XIII
1 Introduction	1
1.1 Motivation	1
1.2 Outline	2
2 Artificial Neural Networks	3
2.1 Human Brain	3
2.2 Artificial Neurons	4
2.3 Activation Function	7
2.4 Neural Networks Architecture	9
2.5 Learning Process	12
2.6 The Back-Propagation Algorithm	12
3 Extreme Learning Machine and its Regularization	17
3.1 Single Hidden Layer Feed-forward Neural Networks (SLFNs) with Random Hidden Neurons	18
3.2 Learning with Extreme Learning Machine	19
3.3 Regularization on Extreme Learning Machine	23
4 A-optimal Design Regularization and its Application in ELM	27
4.1 λ -weighted Best Linear Estimation	27
4.2 A-optimal design Regularization	30
5 Case Studies: Application of A-optimal Design Regularization in Extreme Learning Machine	33
5.1 Approximation of Sine Function	33
5.2 Regression Analysis in Real-world Problems	36
5.3 Multi-class Classification in Satellite Images	39
5.4 Conclusions and Future Works	52
Bibliography	XV
A Proof of the equation (4.13), refers to Cai (2004), pages 65	XVII

List of Figures

2.1	Essential components of a neuron shown in stylized form; source: Zell et al. (1993)	4
2.2	Nonlinear model of a neuron; source: Haykin (2009)	5
2.3	Nonlinear model of neuron k after reformulation, ω_{k0} accounts for the bias b_k ; source: Haykin (2009)	6
2.4	Threshold Function	7
2.5	Linear activation function	8
2.6	Sigmoid function	9
2.7	Sigmoid function of different slope parameter	9
2.8	2-layer feed-forward neural networks; source: Haykin (2009)	10
2.9	Single hidden layer feed-forward network (SLFN); source: Haykin (2009)	11
2.10	An example for a weight in multi-layer neural networks: the weight from the 4 th neuron in the 2 nd hidden layer to the 2 nd neuron in the 3 rd hidden layer	13
3.1	Relationship between regularization λ and RMSE in validation set	26
4.1	Balance of the variance and the bias by the weighting factor λ ; source: Cai (2004)	29
5.1	Approximation of the sine function with both algorithms without outliers	35
5.2	Approximation of the sine function with both algorithms with outliers .	35
5.3	Position of the study area in Wuhan; source: Google Earth	40
5.4	Satellite image of the study area in Wuhan; source: Google Earth	41
5.5	Satellite image to be classified for study area in Wuhan	44
5.6	Classified study area in Wuhan by the original ELM	45
5.7	Classified study area in Wuhan by the ELM with A-optimal design regularization	46

5.8	Satellite image of study area in Karlsruhe from Google Earth	47
5.9	Satellite image to be classified for study area in Karlsruhe	50
5.10	Classified study area in karlsruhe by the original ELM	51
5.11	Classified study area in karlsruhe by the regularized ELM	52

List of Tables

5.1	Data Information for Approximation of the Sine Function	34
5.2	Comparison of RMSE of the original ELM and the ELM with A-optimal design regularization in both situations	34
5.3	Specification of the datasets used for regression analysis	36
5.4	Results for training data: RMSE and $\text{trace}(\text{MSE}(\hat{\mathbf{B}}))$	38
5.5	Results for testing data: RMSE	38
5.6	Specification of data information of the study area in Wuhan	42
5.7	Specification of labeled pixels of the study are in Wuhan	42
5.8	Confusion matrix of the classification in the study are in Wuhan by using the original ELM	42
5.9	Confusion matrix of the classification in the study area in Wuhan by using the regularized ELM	43
5.10	Accuracy and Cohens Kappa coefficient of the classification in the study area in Wuhan	44
5.11	Specification of data information of the study area in Karlstuhe	48
5.12	Specification of labeled pixels of the study area in Karlsruhe	48
5.13	Confusion matrix of the classification in the study are in Karlsruhe by using the original ELM	49
5.14	Confusion matrix of the classification in the study are in Karlsruhe by using the regularized ELM	49
5.15	Accuracy and Cohens Kappa coefficient of the classification in the study area in Karlsruhe	50

Chapter 1

Introduction

1.1 Motivation

Machine learning has been one of the rapidest developing field of computer science with far-reaching applications since the recent couple of decades. It becomes more and more popular for people to apply machine learning to solve real-life problems efficiently.

Artificial neural network (ANN) plays an important role in machine learning. An ANN is a model of computation, inspired by the structure of neural networks in brain. It provides a general, practical method for learning real-valued, discrete-valued and vector-valued functions from training data (Mitchell, 1997). Learning with ANNs was proposed in the mid-20th century. It formed an efficient learning algorithm and has achieved outstanding performance on many learning tasks (Hierons, 2015).

However, when we deal with a large amount of data in high-dimensional feature space, the training speed of the ANN are seriously affected, if we apply traditional feedforward propagation and back propagation to estimate weight matrices by the stochastic gradient descent. It often takes several or even more days to train the ANN. Besides, gradient descent learning algorithms can easily converge to local minimum, resulting in low accuracy of prediction.

Huang et al. (2006) have proposed extreme learning machine (ELM) to solve such problems mentioned above. It is a special single-hidden layer feedforward neural network (SLFN). Instead of estimating both input and output weight matrices with hundreds of learning iteration, we estimate only the output weight matrix or the output vector in ELM with a randomly initialized input weight matrix. The output weight matrix or the output vector can be estimated by Moore-Penrose generalized inverse with the least squares method.

Granted, ELM has successfully solved lots of real-life problems, which were difficulties for traditional ANNs, but its generalization ability remains unsolved problems, which are common in methods based on feedforward neural networks. In other words, how to acquire appropriate network architecture is the problem that we need to solve. On the one hand, we cannot model the data with sufficiently high accuracy through only a

few hidden-layer neurons. On the other hand, a neural network with excessive hidden-layer neurons tends to generate an over-fitting model.

The most common way to improve the generalization performance in machine learning is the regularization. By applying the regularization to ELM, it is necessary not only to minimize the sum of the output residuals of the training data, but to penalize the coefficients of the output matrix or the output vector as well. Our challenge is to find a precisely adaptive regularization parameter to balance one with the other.

In this study, we apply the A-optimal design regularization to ELM to determine an optimal regularization parameter, which can address the under-fitting/over-fitting trade-off.

1.2 Outline

The rest of this thesis is organized as follows. Firstly, some basic theories of feed-forward neural networks will be reviewed. For example, feedforward propagation, back-propagation and stochastic gradient descent. Here we will also talk about the problems of traditional training methods.

In addition, ELM proposed by Huang et al. (2006) will be introduced as a special schema of SLFN. The advantages of ELM over traditional SLFN and how to deal with problems with ELM will be discussed in detail. Moreover, we will emphasize the reason why regularization is necessary for ELM here.

In the next chapter, we will talk about how to determine an optimal regularization parameter for ELM by using A-optimal design regularization. The principle of A-optimal design regularization is to be interpreted. Besides, an existing regularization method used in ELM will be reviewed and some main weakness should be discussed.

Finally, we will inspect the performance of ELM with A-optimal design regularization on several case studies and demonstrate the comparison to original ELM.

Chapter 2

Artificial Neural Networks

Artificial neural networks are commonly referred to as "neural networks". Work on neural networks has been inspired right from its inception by the consciousness that human brain deal with problems in an entirely different way from digital computers. Human brain is a highly complex, nonlinear, and parallel information-processing system (Haykin, 2009). In human brain, neurons are known as structural constituents and they are organized to perform certain computation, for example, pattern recognition. And the computation speed of human brain is much faster than the fastest digital computer nowadays. To be specific, human brain can deal with complicated recognition tasks in only 100-200 ms, whereas tasks of much less complexity cost a great deal of time on a powerful computer.

A neural network is a machine system that is designed to imitate the way in which human brain performs a particular task or function of interest. The neural network consists of artificial neurons, which have similar function as information-processing units in human brain. In addition, the neural network is implemented in software on digital computers. In order to make neural networks performing useful computations, a massive interconnection of simple artificial neuron should be employed. With such neurons, neural network can acquire and store experimental knowledge, then make it available for handling similar problems. This process is so called learning. The procedure applied to carry out the learning process is named learning algorithm.

In this chapter, a basic type of neural networks, called single layer feed-forward neural networks (SLFN) and a traditional learning algorithm for SLFN will be reviewed.

2.1 Human Brain

At the beginning, we take a quick look at some basic neurobiology. A human brain consists of about 100 billion neurons. A highly stylized example of the neuron is shown in figure 2.1. Neurons communicate with each other via electrical signals which are transient impulses. Each neuron typically receives thousands of signals from other neurons. which eventually reach the cell body. Then, the signals are integrated in some

way to generate a voltage impulse for output. The generated impulse is transmitted to other neurons via a branching fibre known as axon.

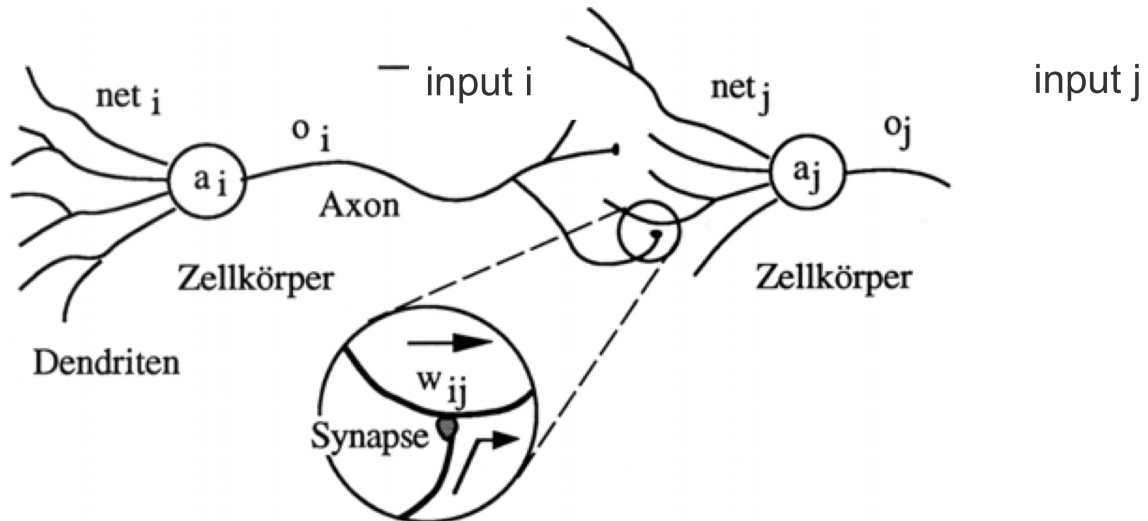


Figure 2.1: Essential components of a neuron shown in stylized form; source: Zell et al. (1993)

To perform a certain task, for example, pattern recognition, human brain can operate information in highly parallel processes on representation that are distributed over a large amount of neurons. This kind of highly parallel computation or operation based on distributed representation is an important motivation for ANN. With neurobiological analogy as the source of inspiration, and the wealth of theoretical and computational tools that we are developing, it is now possible to simplify networks in human brain as ANN to offer an alternative form of parallel computation that might be more appropriate for solving tasks in real world.

2.2 Artificial Neurons

An artificial neuron, called neuron in brief, in an ANN is an information-processing unit, which is fundamental to the operation of an ANN. Figure 2.2 shows the model of a neuron, which represents the basis for designing a general neural network.

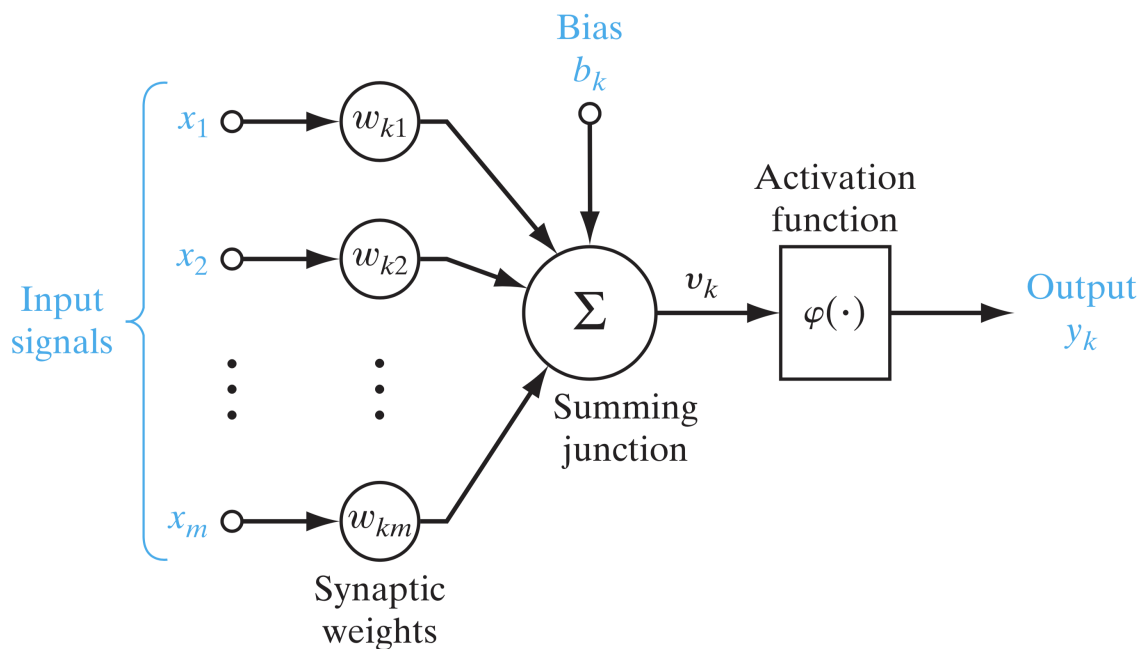


Figure 2.2: Nonlinear model of a neuron; source: Haykin (2009)

From figure 2.2, four basic elements of the model of neurons can be identified:

- A set of connecting links, which are characterized by a weight matrix.
- An adder for summing all input signals, the operation described here constitute a linear combiner.
- An activation function for limiting the amplitude of the output of an neuron. Typically, the amplitude range of the output after limitation is within the interval $[0,1]$ or alternatively $[-1,1]$, which are depended on the choice of activation function.
- Bias, which has the effect of increasing or lowering the net input of the activation function.

We can describe the neuron k depicted in figure 2.2 mathematically with a pair of equations:

$$u_k = \sum_{j=1}^m \omega_{kj} x_j \quad (2.1)$$

$$y_k = \varphi(u_k + b_k) \quad (2.2)$$

- $\mathbf{x} = (x_1, x_2, \dots, x_m)$ are the inputs
- ω_k is the respective input weight matrix of neuron k .
- u_k (not shown in figure 2.2) is the linear combiner output due to the inputs.
- φ is the activation function

- y_k is the output of the neuron k
- b_k is the bias, which can result in an affine transformation to u_k

If we assume the result of u_k after affine transformation as v_k :

$$v_k = u_k + b_k \quad (2.3)$$

We can rewrite the equation (2.3) in matrix form as follows.

$$\begin{aligned} v_k &= u_k + b_k \\ &= \sum_{j=1}^m \omega_{kj} x_j + b_k \\ &= \boldsymbol{\omega}_k \mathbf{x} + b_k \\ &= \begin{pmatrix} b_k & \boldsymbol{\omega}_k \end{pmatrix} \cdot \begin{pmatrix} 1 \\ \mathbf{x} \end{pmatrix} \end{aligned} \quad (2.4)$$

Therefore, it is possible to reformulate the model of neuron k as shown in figure 2.3

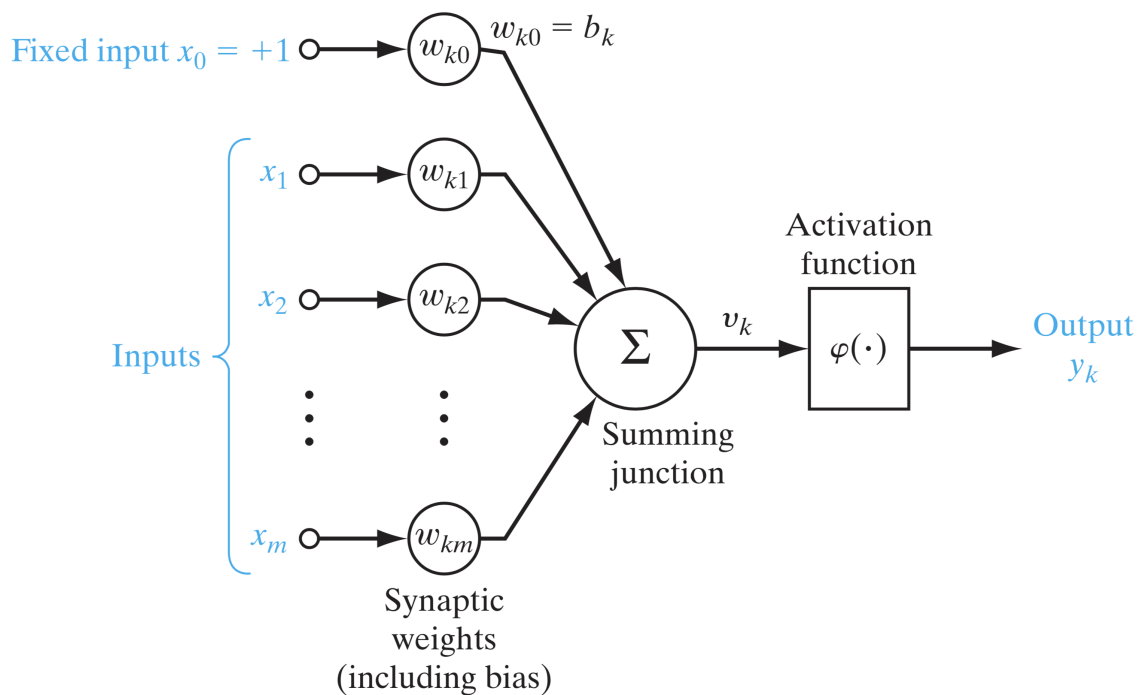


Figure 2.3: Nonlinear model of neuron k after reformulation, w_{k0} accounts for the bias b_k ; source: Haykin (2009)

In figure 2.3, the bias is accounted as an input unit, by these 2 operations:

- adding another input unit with constant value +1
- adding a new input weight equal to the bias b_k

For such model after reformulation, we can describe it in mathematical terms as equation

$$y_k = \varphi(\mathbf{w}_k \cdot \mathbf{X}) = \varphi(v_k) \quad (2.5)$$

- \mathbf{X} is the input adding a constant term with value +1
- \mathbf{w}_k is the new input weight including the bias b_k .
- φ is the activation function.

2.3 Activation Function

Activation functions are an extremely important part of the artificial neural networks. An activation function is also known as transfer function. It maps the resulting values of a neuron in between (0,1) or (-1,1) (depending upon the type of activation function). The activation functions take the decision whether a neuron should be activated or not, so in other words, the activation functions decide if the input information received should be passed or ignored. In addition, the activation functions help to transform the input information non-linearly and it is this non-linearity makes it possible to learn arbitrarily complicated transformation from the input to the output. Therefore, the neural networks are capable to learn and perform complex tasks.

In what follows, some types of activation functions are reviewed.

- **Threshold Function (Binary Step Function)**

For this type of activation function, described in figure 2.4, it is defined as

$$\varphi(v_k) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases} \quad (2.6)$$

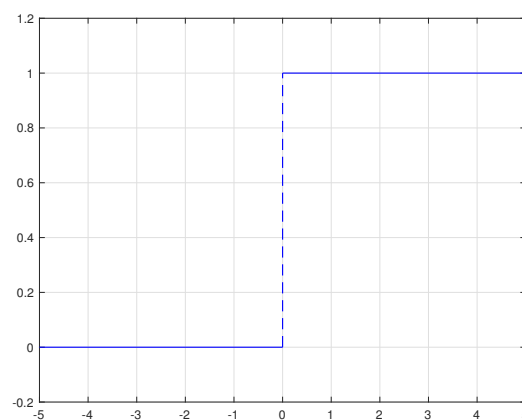


Figure 2.4: Threshold Function

If the value of v_k is above zero, then the neuron is activated. Otherwise, the neuron is ignored. This function is more theoretical than practical since in most cases the data should be classified into multiple classes rather than binary classified. Moreover, the gradient of the threshold function is equal to zero. This makes the threshold function not so meaningful because we need to use the gradient of the activation function to modify the weight matrices during back-propagation.

- **Linear Function**

A simplest example for the linear function is $y = x$, as shown in figure 2.5.

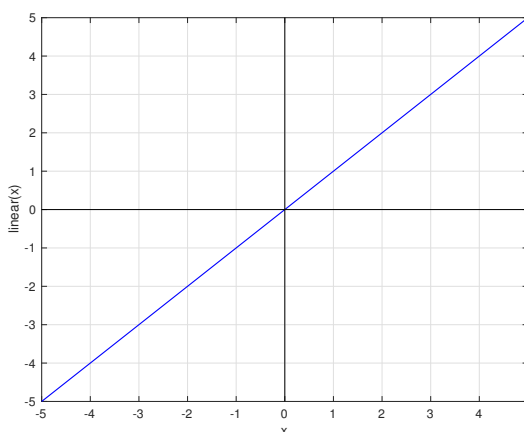


Figure 2.5: Linear activation function

When we use linear function as the activation function, the activation is proportional to the input information. The linear function gives a range of activation rather than binary activation.

There are two main problems for the linear function. In one hand, the gradient of the linear function is constant and not depending on the input information. Similar to the threshold function, it leads to problems in back-propagation. On the other hand, if we have many hidden layer, and each hidden layer is linear activated. Then, no matter how many layers we have, the output of the last layer is just a linear function of the input of the first layer. This means we just lost the ability of stacking layers if we use the linear function as the activation function.

- **Nonlinear Functions**

These functions are used to separate the data, which is not linearly separable. Nonlinear functions are the most used activation functions. A nonlinear activation function makes it easy for the model to generalize and adapt with variety of data. There are many different types of nonlinear activation function, for example, sigmoid function, *tanh* function, Rectified linear unit (Relu), etc. In this thesis, sigmoid function is used as the activation function, so we will discuss sigmoid function in details here.

The sigmoid function has the mathematical form $\varphi(v_k) = \frac{1}{1+e^{-a \cdot v_k}}$, where a is the slope parameter of the sigmoid function. The curve of the sigmoid function is "S-shaped" as shown in figure 2.6. By varying the parameter a , sigmoid functions of different slopes can be obtained, as illustrated in figure 2.7. The sigmoid function is the most common form of activation function used in the construction of neural networks.

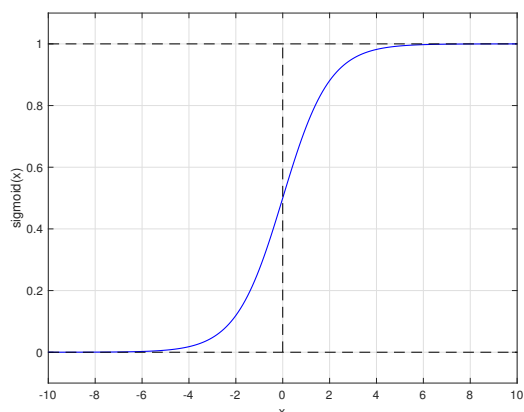


Figure 2.6: Sigmoid function

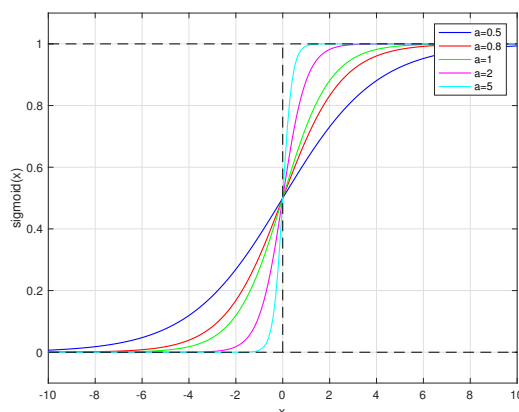


Figure 2.7: Sigmoid function of different slope parameter

As alluded above, the sigmoid function takes a real-valued number and "squashes" into range $(0, 1)$. Moreover, from the figure 2.6 and 2.7, we can see that large negative numbers become 0 and large positive numbers become 1.

2.4 Neural Networks Architecture

After the introduction of a single neuron in neural networks, we will talk about the architecture of an integrated neural network. In neural networks, we connect a plenty of neurons by hierarchical networks, with the outputs of some neurons being the inputs to others. These networks can be represented as connected layers of nodes. Each nodes in a layer means a neuron. In a fully fledged neural network, there are many such interconnection nodes. These nodes can come in a myriad of different forms. Here, we may introduce two fundamentally different classes of neural network architectures:

(i) 2-Layer Feedforward Neural Networks

2-layer feed-forward neural networks are the simplest form of layered neural networks, and it was first devised by Rosenblatt (1988). For a 2-layer neural network, we have an input layer of source nodes that projects directly onto an output layer of neurons (computation nodes). 2-layer neural network is strictly of a feed-forward type, as illustrated in figure 2.8

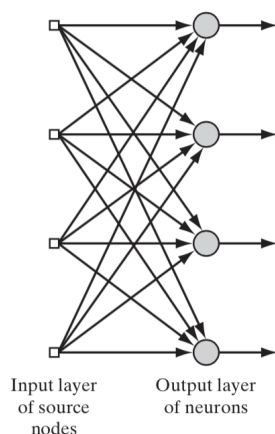


Figure 2.8: 2-layer feed-forward neural networks; source: Haykin (2009)

Such a neural network has only a single layer of computation neurons. For the input source nodes, there is no computation performed. With 2-layer neural networks, we can only solve linearly separable problems. For example, it is only capable of classifying patterns into binary class.

This limitation of the 2-layer neural networks leads to the introduction of hidden layers of neural networks, which are a significant feature of multi-layer feed-forward neural networks.

(ii) Multi-Layer Feed-forward Neural Networks

Multi-layer feed-forward neural networks are the second class of neural network architectures. They help to overcome many limitations of 2-layer neural networks and have proved useful in a wide variety of applications. Multi-layer feed-forward neural networks were generally not used before the mid-1980s due to a lack of available training algorithms. After back-propagation was proposed by Rumelhart et al. (1988), multi-layer feed-forward neural networks, sometimes called multi-layer perceptron (MLP) networks, have become a mainstay of neural networks research.

The multi-layer feed-forward neural network distinguishes itself by the presence of one or more hidden layers, whose computation nodes are correspondingly called hidden neurons. The term "hidden" refers to the fact that this part is not seen directly from either the input or output of neural networks. The function of hidden neurons is to intervene between the external input and the network output in some particularly useful manner.

Figure 2.9 illustrates a simple architecture of multi-layer feed-forward neural networks with a single hidden layer, which is also called a single hidden layer feed-forward network (SLFN).

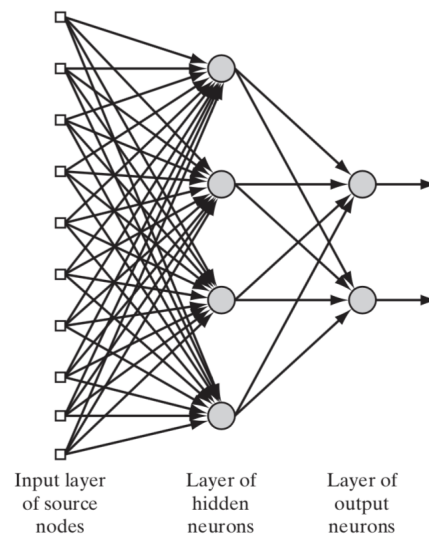


Figure 2.9: Single hidden layer feed-forward network (SLFN); source: Haykin (2009)

The source nodes in the input layer constitute the input information applied to the neurons in the hidden layer. Typically, the neurons in each hidden layer of multi-layer neural networks receive the outputs from the layer before as the input information and generate the output and transmit to the adjacent forward layer. The set of outputs of the neurons in the output layer of the multi-layer neural network constitute the overall response of the network to the input information supplied by the source nodes in the input layer.

The neural network in figure 2.9 is also said to be fully connected in the sense that every node in each layer of the network is connected to every other node in the adjacent forward layer. In general, three points highlight the basic features of multi-layer feed-forward neural networks as follows.

- For each neuron in multi-layer feed-forward neural networks, there is a non-linear and differentiable activation function.
- Each multi-layer feed-forward neural network contains one or more hidden layers.
- Each multi-layer feed-forward neural network exhibits a high degree of connectivity, the extent of which is determined by weight matrices of the networks.

These features, however, were also responsible for the deficiencies in applications of multi-layer feed-forward neural networks by lack of corresponding learning algorithm. On the one hand, the presence of a distributed form of nonlinearity and the high connectivity of the networks make the theoretical analysis difficult to undertake. On the other hand, the use of hidden neurons makes the learning process even more difficult to visualize. Briefly, in the learning process, it should

be decided which features of the input information should be represented by the hidden neurons. Therefore, the learning process is made harder since a choice must be made from a larger space of possible functions to represent the input information.

2.5 Learning Process

Before mid 1980s, there was not a generally good idea to solve such problems mentioned above and apply multi-layer neural networks to deal with complex tasks. In order to train multi-layer neural networks, Rosenblatt (1988) developed a learning algorithm called back-propagation, which is now the most popular and widely used learning algorithm. The training proceeds with back-propagation are divided into two phases:

- **Forward Phase**

In the forward phase, which is also called feed-forward propagation, all weight matrices in the network are fixed and the input information is propagated through the network, and outputs corresponding to the input information is generated in the end. Therefore, in forward phase, changes are only confined to the activation functions. Due to variant activation functions, we will obtain different outputs through the network.

- **Backward Phase**

The backward phase is the core of the back-propagation. In the backward phase, an output error is produced by comparing the outputs of the network with a supposed result. The output error is propagated through the network, again layer by layer, but in backward direction this time. In the backward phase, successive adjustments are made to the weight matrices of the network.

2.6 The Back-Propagation Algorithm

As it is already mentioned many times previously, the development of the back-propagation provides a computationally efficient method for training multi-layer neural networks. At the heart of back-propagation is an expression for the partial derivative $\frac{\partial C}{\partial \omega}$ of the cost function C with respect to an weight ω (or bias b) in the networks. This expression tells us how quickly the cost function C changes when the weights and biases are changed. The back-propagation algorithm gives us detailed insights into how changing the weights and biases changes the overall performance of the network.

Matrix-based Computation

The matrix-based approach to computing the outputs from a neural network is mentioned briefly in the previous section. Here, we will revisit the matrix-based computation in details. It provides a good way to understand the notation used in the back-propagation algorithm.

We will begin with a notation which refers to weight in the networks. w_{jk}^l is used to denote the weight for the connection from k^{th} neuron in the $(l-1)^{\text{th}}$ hidden layer to the j^{th} neuron in the l^{th} hidden layer. The diagram below shows us an example of the weight on a connection from the 4^{th} neuron in the 2^{nd} hidden layer to the 2^{nd} neuron in the 3^{rd} hidden layer of a network:

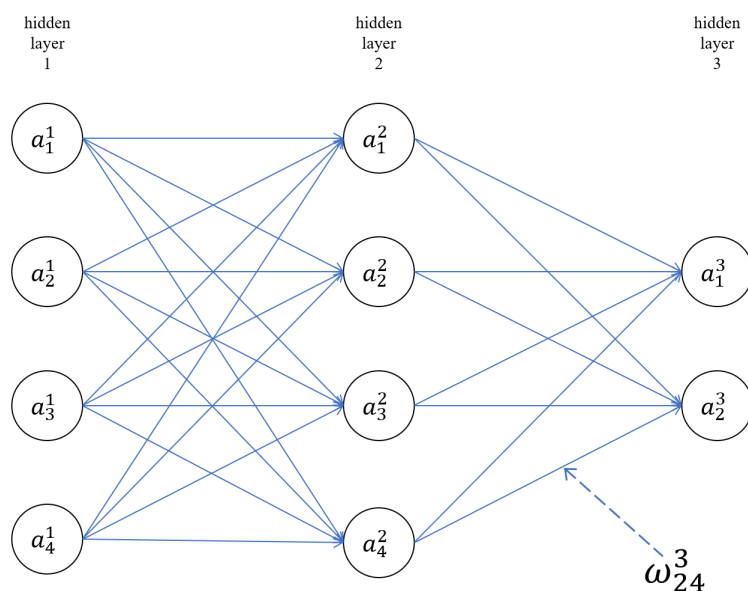


Figure 2.10: An example for a weight in multi-layer neural networks: the weight from the 4^{th} neuron in the 2^{nd} hidden layer to the 2^{nd} neuron in the 3^{rd} hidden layer

Similarly, we use b_j^l for the bias and a_j^l for the activated output of j^{th} neuron in the l^{th} hidden layer. With such notations, the relationship of the activation a_j^l to the activations in the $(l-1)^{\text{th}}$ layer can be presented by the equation (2.7)

$$a_j^l = \varphi\left(\sum_k w_{jk}^l a_k^{l-1} + b_j^l\right) \quad (2.7)$$

where the sum is over all neurons in the $(l-1)^{\text{th}}$ layer. To rewrite this expression in a matrix form, we define a weight matrix w^l and a bias vector b^l for each hidden layer l . It consists of all the weights connecting to the l^{th} layer of neurons. Specifically, the

entry in the j^{th} row and k^{th} column is w_{jk}^l . Afterwards, we can write the equation (2.7) in compact vectorized form

$$a^l = \varphi(w^l a^{l-1} + b^l) \quad (2.8)$$

Cost Function

The goal of back-propagation is to compute the partial derivatives $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$. The cost function is used to quantify how accurate the computed outputs of the neural networks are, comparing to the corresponding supposed outputs. It has the form

$$C(\mathbf{w}, \mathbf{b}) = \frac{1}{2n} \left\| \sum_{i=1}^n y_i - a_i^{\text{out}} \right\|^2 \quad (2.9)$$

where:

- n is the number of training samples
- \mathbf{y} is the vector of corresponding supposed outputs
- \mathbf{a}^{out} is the computed outputs

Procedure of Back-propagation

In order to compute the partial derivatives $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$, it is necessary to introduce an intermediate, δ_j^l , which refers to the error in the j^{th} neuron in the l^{th} layer.

$$\delta_j^l = \frac{\partial C}{\partial z_j^l} \quad (2.10)$$

where z_j^l is the unactivated output. Then, we use δ^l to denote the vector of error associated with layer l . The back-propagation algorithm gives us a way to compute δ^l for each layer, and afterwards, relate such errors to the quantities we really need, $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$.

Before introducing the detailed steps, it is impossible to understand some intermediates of back-propagation.

- **The error in the output layer δ^{out}**

$$\delta_j^{out} = \frac{\partial C}{\partial a_j^{out}} \varphi'(z_j^{out}) \quad (2.11)$$

The first term $\frac{\partial C}{\partial a_j^{out}}$ measures how fast the cost function C is changing as a function of a_j^{out} . The second term $\varphi'(z_j^{out})$ measures how fast the activation function φ is changing at z_j^{out} . The equation (2.11) is a component-wise expression for δ^{out} . We can rewrite it in a matrix-form, as

$$\delta^{out} = \nabla_{a^{out}} C \odot \varphi'(z^{out}) \quad (2.12)$$

Here, $\nabla_{a^{out}}$ is defined to be a vector whose components are the partial derivatives $\frac{\partial C}{\partial a_j^{out}}$, the symbol \odot means the element-wise product.

- **The error δ^l in terms of the error in the $(l+1)^{th}$ layer, δ^{l+1}**

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \varphi'(z^l) \quad (2.13)$$

where $(w^{l+1})^T$ is the transpose of the weight matrix (w^{l+1}) for the $(l+1)^{th}$ layer. Suppose we know the error δ^{l+1} at the $(l+1)^{th}$ layer, when we apply the transpose weight matrix $(w^{l+1})^T$, we can think intuitively of this as moving the error backward through the network, giving us some sort of measure of the error at the output of the l^{th} layer. By combining the equation (2.12) and (2.13), we can compute the error δ^l for any layer in the neural networks. We start by using the equation (2.12) to compute the error in the output layer δ^{out} , then apply the equation (2.13) to compute the error δ^l in the previous layers, all the way back through the network.

- **Changing rate of the cost function with respect to any bias in the network**

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \quad (2.14)$$

This equation tells us that the error δ_j^l is exactly equal to the changing rate $\frac{\partial C}{\partial b_j^l}$.

- **Changing rate of the cost function with respect to any weight in the network**

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l \quad (2.15)$$

In terms of the quantities δ^l and a^{l-1} , we can compute the partial derivatives $\frac{\partial C}{\partial w_{jk}^l}$

The equations (2.12 – 2.15) provide us with a way of computing the gradient of the cost function. The particular steps of the back-propagation algorithm is then presented in the following box.

- (1) Input the training data x in the network.
- (2) Initialize all weight matrices randomly but the elements in each weight matrix cannot be completely same.
- (3) Propagate the input x forward through the network. For each $l = 2, 3, \dots, out$, compute the unactivated output $z^l = w^l a^{l-1} + b^l$ and the activated output $a^l = \varphi(z^l)$
- (4) Compute the output error $\delta^{out} = \nabla_{a^{out}} C \odot \varphi'(z^{out})$
- (5) Propagate the output error backward through the network, for each layer except the output layer, compute $\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \varphi'(z^l)$
- (6) Use the method of gradient descent (Amari (1996)) to update each weight in the network.
- (7) Iterate the forward and backward computations, step (3)–(5), until some stopping criterion we choose is met.

As it is interpreted above, the key point of the back-propagation algorithm is computing the error vector δ^l backward from the output layer. The backward computation is a consequence of the fact that the cost function is a function of outputs from the network. To understand how the cost varies with earlier weights and biases, we need to repeatedly apply the chain rule, working backward through the network to obtain usable expressions.

Chapter 3

Extreme Learning Machine and its Regularization

Feed-forward neural networks have been extensively applied in many fields due to their strong abilities:

- Feed-forward neural networks can approximate complex nonlinear feature mappings directly from the training data.
- Feed-forward neural networks provide models for a large class of natural and artificial phenomena that are difficult to deal with using classical parametric methods.

However, in order to obtain better learning performance, many iterative learning steps based on the back-propagation algorithm and the method of gradient descent may be required. In real application, it may several hours, several days, and even more time to train the neural networks in this way, especially for cases when there is a large amount of training data with features in high-dimension space. Moreover, in the back-propagation algorithm, we implement the method of gradient descent to search through the space of all possible weights in the neural networks. Since the surface of cost function for multilayer feed-forward neural networks may contain many different local minimum, the learning results of weights by applying back-propagation algorithm are only guaranteed to converge toward some local minimum. The convergence toward local minimums can result in bad performance and low accuracy.

For the sake of overcoming these general problems of multi-layer feed-forward neural networks mentioned above, Huang et al. (2006) proposed a new learning schema called extreme learning machine (ELM). It is a special neural network with only one hidden layer. Precisely, ELM is a particular form of single hidden layer feed-forward neural network (SLFN).

Tamura and Tateishi (1997) proved that SLFNs with N hidden neurons, whose input weights and biases are randomly chosen, can exactly learn N distinct training samples. And such hidden neurons can thus be called random hidden neurons. Unlike the popular thinking and most practical implementation that all the parameters of the feed-forward networks need to be tuned, it is not always necessary to adjust the input

weights and the biases in the hidden layer. Some simulation results in the work of Huang and Siew (2006) have shown that this method not only takes much less time for learning but produce better generalization performance than normal neural networks as well.

After the input weights and the biases are fixed, SLFNs can be simply considered as a linear system and the output weights, which connect the hidden layer and the output layer of SLFNs, can be analytically determined through simple generalized inverse operation of feature mapping matrix (hidden layer matrix). SLFNs based on this concept are so called extreme learning machine, whose learning speed can be much faster than the back-propagation algorithm while obtaining better generalization performance.

In the next section, we will firstly review the mathematical model of SLFNs with random hidden neurons.

3.1 Single Hidden Layer Feed-forward Neural Networks (SLFNs) with Random Hidden Neurons

For N arbitrary distinct training samples $(\mathbf{x}_j, \mathbf{y}_j)$, where $\mathbf{x}_j = [x_{j1}, x_{j2}, \dots, x_{jn}]^T \in \mathbf{R}^n$ and $\mathbf{y}_j = [y_{j1}, y_{j2}, \dots, y_{jn}]^T \in \mathbf{R}^m$, normal SLFNs with L hidden neurons and activation function $\varphi(x)$ are mathematically modeled as

$$\sum_{i=1}^L \beta_i \varphi(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) = \mathbf{t}_j \quad (3.1)$$

$$j = 1, 2, \dots, N$$

where $\mathbf{w}_i = [w_{i1}, w_{i2}, \dots, w_{in}]$ is the weight vector connecting the i^{th} hidden neuron and the input layer, $\beta_i = [\beta_{i1}, \beta_{i2}, \dots, \beta_{im}]^T$ is the weight vector connecting the i^{th} hidden neuron and the output layer and b_i is the bias for the i^{th} hidden neuron, and \mathbf{t}_j is the computed output for the j^{th} training sample.

If a SLFN with L hidden neurons with the activation function $\varphi(x)$ can exactly approximate the N training samples with zero error, which means $\mathbf{t}_i = \mathbf{y}_i$, then there exist β_i, \mathbf{w}_i and b_i such that

$$\sum_{i=1}^L \beta_i \varphi(\mathbf{w}_i \cdot \mathbf{x}_j + b_j) = \mathbf{y}_j \quad (3.2)$$

$$j = 1, 2, \dots, N$$

The equation 3.2 can be written in matrix form as

$$\underset{N \cdot L}{\mathbf{H}} \cdot \underset{L \cdot m}{\mathbf{B}} = \underset{N \cdot m}{\mathbf{Y}} \quad (3.3)$$

where

$$H(\mathbf{w}, \mathbf{x}, \mathbf{b}) = \begin{bmatrix} \varphi(\mathbf{w}_1 \cdot \mathbf{x}_1 + b_1) & \varphi(\mathbf{w}_2 \cdot \mathbf{x}_1 + b_2) & \dots & \varphi(\mathbf{w}_L \cdot \mathbf{x}_1 + b_L) \\ \varphi(\mathbf{w}_1 \cdot \mathbf{x}_2 + b_1) & \varphi(\mathbf{w}_2 \cdot \mathbf{x}_2 + b_2) & \dots & \varphi(\mathbf{w}_L \cdot \mathbf{x}_2 + b_L) \\ \vdots & \ddots & & \vdots \\ \varphi(\mathbf{w}_1 \cdot \mathbf{x}_N + b_1) & \varphi(\mathbf{w}_2 \cdot \mathbf{x}_N + b_2) & \dots & \varphi(\mathbf{w}_L \cdot \mathbf{x}_N + b_L) \end{bmatrix} \quad (3.4)$$

$$\mathbf{B} = \begin{bmatrix} \beta_1^T \\ \beta_2^T \\ \vdots \\ \beta_L^T \end{bmatrix}_{L \times m} \quad \text{and} \quad \mathbf{Y} = \begin{bmatrix} \mathbf{y}_1^T \\ \mathbf{y}_2^T \\ \vdots \\ \mathbf{y}_N^T \end{bmatrix}_{N \times m} \quad (3.5)$$

H is called feature mapping matrix, or hidden layer matrix of the neural networks; the i^{th} column of H is the i^{th} hidden neuron output with respect to the input training samples x_1, x_2, \dots, x_N .

3.2 Learning with Extreme Learning Machine

Before introducing the learning algorithm of ELM, we need to review two important theorems proved by Huang et al. (2006).

Theorem 1. *Given a standard SLFN with N hidden neurons and activation function $\varphi: \mathbf{R} \rightarrow \mathbf{R}$, which is infinitely differentiable in any interval, for N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{y}_i)$, where $\mathbf{x}_i \in \mathbf{R}^n$ and $\mathbf{y}_i \in \mathbf{R}^m$, for any \mathbf{w}_i and b_i randomly chosen from any intervals of \mathbf{R}^n and \mathbf{R} , respectively, according to any continuous probability distribution, then with probability one, the feature mapping matrix \mathbf{H} of the SLFN is invertible and $\|\mathbf{H}\mathbf{B} - \mathbf{Y}\| = 0$*

Proof. Let us consider a vector $\mathbf{c}(b_i) = [\varphi(\mathbf{w}_i \cdot \mathbf{x}_1 + b_i), \varphi(\mathbf{w}_i \cdot \mathbf{x}_2 + b_i), \dots, \varphi(\mathbf{w}_i \cdot \mathbf{x}_N + b_i)]^T$, the i^{th} column of feature mapping matrix \mathbf{H} , in Euclidean space \mathbf{R}^N , where $b_i \in (p, q)$ and (p, q) is any interval of \mathbf{R} .

On the basis of the proof method proposed by Tamura and Tateishi (1997), it can be easily proved by contradiction that vector \mathbf{c} does not belong to any subspace whose dimension is less than N .

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

$$\langle \boldsymbol{\alpha}, \mathbf{c}(b_i) - \mathbf{c}(a) \rangle = \alpha_1 \cdot \varphi(b_i + d_1) + \alpha_2 \cdot \varphi(b_i + d_2) + \dots + \alpha_N \cdot \varphi(b_i + d_N) - \tau = 0 \quad (3.6)$$

where $d_k = \mathbf{w}_i \cdot \mathbf{x}_k$, for $k = 1, 2, \dots, N$ and $z = \boldsymbol{\alpha} \cdot \mathbf{c}(\boldsymbol{\alpha})$, $\forall b_i \in (p, q)$. Assuming that $\alpha_N \neq 0$, then equation (3.6) can be further transform into form

$$\varphi(b_i + d_N) = \tau - \sum_{j=1}^{N-1} \gamma_j \varphi(b_i + d_j) + z/\alpha_N \quad (3.7)$$

where $\gamma_j = \frac{\alpha_j}{\alpha_N}$, for $j = 1, 2, \dots, N-1$. Since $\varphi(x)$ is infinitively differentiable in any interval, we have

$$\begin{aligned} \varphi^{(l)}(b_i + d_N) &= - \sum_{j=1}^{N-1} \gamma_j \varphi^{(l)}(b_i + d_j) \\ l &= 1, 2, \dots, N, N+1, \dots \end{aligned} \quad (3.8)$$

where $\varphi^{(l)}$ is the l^{th} derivative of activation function φ . However, there are only $N-1$ free coefficients: $\gamma_1, \gamma_2, \dots, \gamma_{N-1}$ for the derived more than $N-1$ linear equations, this is contradictory. Thus, vector \mathbf{c} does not belong to any subspace whose dimension is less than N .

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

Theorem 2. *Given any small positive value $\epsilon > 0$ and activation function $\varphi : \mathbf{R} \rightarrow \mathbf{R}$, which is infinitively differentiable in any interval, there exists $L \leq N$ such that for N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{y}_i)$, where $\mathbf{x}_i \in \mathbf{R}^n$ and $\mathbf{y}_i \in \mathbf{R}^m$, for any \mathbf{w}_i and b_i randomly chosen from any intervals of \mathbf{R}^N and \mathbf{R} , respectively, according to any continuous probability distribution, then with probability one, $\|\mathbf{H}_{N \times L} \mathbf{B}_{L \times m} - \mathbf{Y}\| < \epsilon$.*

Proof. The validity of this theorem is obvious, otherwise, one could simply choose $L = N$, which makes $\|\mathbf{H}_{N \times L} \mathbf{B}_{L \times m} - \mathbf{Y}\| < \epsilon$ according to **Theorem 1**. \square

On the basis of **Theorem 1** and **Theorem 2**, the algorithm of ELM, which is an extremely simple and efficient method to train SLFNs, can be proposed. As it is mentioned before and rigorously proved in **Theorem 1** and **Theorem 2**, the input weights and the biased for the hidden neurons can be randomly assigned, if only the activation function φ is infinitively differentiable. It is very interesting and surprising that unlike the most common understanding that all the parameters of SLFNs need to be adjusted, the input weights \mathbf{w}_i and the biases b_i for hidden neurons are in fact not necessarily tuned and the feature mapping matrix \mathbf{H} can actually remain unchanged once random values have been assigned to these parameters in the beginning of learning. Then for

fixed feature mapping matrix, training a SLFN is simply equivalent to learning the output weights by finding a least squares solution $\hat{\mathbf{B}}$ of the linear system $\mathbf{HB} = \mathbf{Y}$.

$$\begin{aligned} \|\mathbf{H}(w_1, w_2, \dots, w_L, b_1, b_2, \dots, b_L) \cdot \hat{\mathbf{B}} - \mathbf{Y}\| = \\ \min_{\mathbf{B}} \|\mathbf{H}(w_1, w_2, \dots, w_L, b_1, b_2, \dots, b_L) \cdot \mathbf{B} - \mathbf{Y}\| \end{aligned} \quad (3.9)$$

If the number L of hidden neurons is equal to the number N of distinct training samples, $L = N$, then the feature mapping matrix \mathbf{H} is square and invertible, and SLFNs can approximate the training samples with zero error.

However, in most real applications, the number of hidden neurons L is much less than the number of distinct training samples, $L \ll N$, \mathbf{H} is a non-square matrix and there may not exist w, b, \mathbf{B} such that $\mathbf{HB} = \mathbf{Y}$. On the grounds of Moore-Penrose generalized inverse proposed by Rao and Mitra (1972), the least squares solution of \mathbf{B} with minimum norm is

$$\hat{\mathbf{B}} = \mathbf{H}^{\dagger} \mathbf{Y} \quad (3.10)$$

where \mathbf{H}^{\dagger} is the Moore-Penrose generalized inverse of the feature mapping matrix \mathbf{H} . Several important properties of such least squares solution are enumerated as follows:

- **Minimum Training Error**

The special solution $\hat{\mathbf{B}} = \mathbf{H}^{\dagger} \mathbf{y}$ is one of the least squares solutions of a general linear system $\mathbf{HB} = \mathbf{Y}$, meaning that the minimum training error can be reached by this special solution:

$$\begin{aligned} \|\mathbf{H}\hat{\mathbf{B}} - \mathbf{Y}\| &= \|\mathbf{H}\mathbf{H}^{\dagger} \mathbf{Y} - \mathbf{Y}\| \\ &= \min_{\mathbf{B}} \|\mathbf{HB} - \mathbf{Y}\| \end{aligned} \quad (3.11)$$

Although almost all learning algorithms wish to reach the minimum training error, however, most of them cannot reach it because global minimum cannot be reached in usual cases and infinite training iteration is not possible in real-life applications.

- **Minimum Norm of Output Weights**

This special solution $\hat{\mathbf{B}} = \mathbf{H}^{\dagger} \mathbf{y}$ has the minimum norm among all the least squares solutions of the linear system $\mathbf{HB} = \mathbf{y}$

$$\begin{aligned} \|\hat{\mathbf{B}}\| &= \|\mathbf{H}^{\dagger} \mathbf{y}\| \leq \|\mathbf{B}\| \\ \forall \mathbf{B} \in \left\{ \mathbf{B} : \|\mathbf{HB} - \mathbf{Y}\| \leq \|\mathbf{HB}' - \mathbf{Y}\|, \forall \mathbf{B}' \in \mathbf{R}^{L \times N} \right\} \end{aligned} \quad (3.12)$$

- This special least squares solution $\hat{\mathbf{B}} = \mathbf{H}^{\dagger} \mathbf{Y}$ with minimum norm is unique.

According to the basic theories interpreted above, the algorithm of ELM can be summarized as follows:

Algorithm ELM

Given a training set with N distinct samples,
 $\mathbf{x} = \{(x_i, \mathbf{y}_i) | x_i \in \mathbf{R}^n, \mathbf{y}_i \in \mathbf{R}^m, i = 1, 2, \dots, N\}$, activation function $\varphi(x)$, and the
 number of hidden neurons L

1. Randomly generate the input weight vector \mathbf{w}_i and the bias $b_i, i = 1, 2, \dots, L$.
2. Compute the feature mapping matrix H for given input weights and biases.

$$\begin{aligned} \mathbf{H}(\mathbf{w}, \mathbf{x}, \mathbf{b}) &= \begin{bmatrix} \varphi(\mathbf{w}_1 \cdot \mathbf{x}_1 + b_1) & \varphi(\mathbf{w}_2 \cdot \mathbf{x}_1 + b_2) & \dots & \varphi(\mathbf{w}_L \cdot \mathbf{x}_1 + b_L) \\ \varphi(\mathbf{w}_1 \cdot \mathbf{x}_2 + b_1) & \varphi(\mathbf{w}_2 \cdot \mathbf{x}_2 + b_2) & \dots & \varphi(\mathbf{w}_L \cdot \mathbf{x}_2 + b_L) \\ \vdots & \ddots & & \vdots \\ \varphi(\mathbf{w}_1 \cdot \mathbf{x}_N + b_1) & \varphi(\mathbf{w}_2 \cdot \mathbf{x}_N + b_2) & \dots & \varphi(\mathbf{w}_L \cdot \mathbf{x}_N + b_L) \end{bmatrix} \\ &= \begin{bmatrix} h(x_1) \\ h(x_2) \\ \vdots \\ h(x_N) \end{bmatrix} = h(\mathbf{x}) \end{aligned}$$

3. Compute the output weight matrix (vector) $\hat{\mathbf{B}}$

$$\hat{\mathbf{B}} = \mathbf{H}^\dagger \mathbf{Y}$$

where $\mathbf{Y} = [\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_N]^T$

To compute the Moore-Penrose generalized inverse of feature mapping matrix \mathbf{H} , several method can be applied, for example: orthogonal projection, orthogonalization method, iterative method and singular value decomposition (SVD). Because of the limitation of the orthogonalization method and iterative method that searching and iteration must be used, which we wish to avoid in ELM, the orthogonalization method and the iterative method are given up in computing this Moore-Penrose generalized inverse of the feature mapping matrix \mathbf{H} . The SVD can be generally applied in computing the Moore-Penrose generalized inverse in all cases. In ELM, the orthogonal projection method is preferred. We can use the orthogonal projection method in two cases:

- $\mathbf{H}^T \mathbf{H}$ is nonsingular

$$\mathbf{H}^\dagger = (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T \quad (3.13)$$

- $\mathbf{H} \mathbf{H}^T$ is nonsingular

$$\mathbf{H}^\dagger = \mathbf{H}^T (\mathbf{H} \mathbf{H}^T)^{-1} \quad (3.14)$$

Generally speaking, when we have more distinct training samples than hidden neurons, it is a over-determined case, $\mathbf{H}^T\mathbf{H}$ is then nonsingular, equation (3.13) is used to compute the Moore-Penrose generalized inverse of the feature mapping matrix \mathbf{H} . On the contrary, when there are more hidden neurons than training samples, it is a under-determined case, $\mathbf{H}\mathbf{H}^T$ is nonsingular, we have to use equation (3.14) to compute the Moore-Penrose generalized inverse.

In general application in real life, we have sufficient training samples so that there are more training samples than hidden neurons. Therefore, in usual cases, we applied equation (3.13) to compute the Moore-Penrose generalized inverse of the feature mapping matrix \mathbf{H} .

3.3 Regularization on Extreme Learning Machine

ELM has attracted many attentions for its extremely fast training speed. But similar to other learning algorithms based on SLFNs, there is an important unsolved problem in practical applications of ELM, which is how to obtain the most appropriate architecture of SLFNs. On the one hand, a SLFN with too few hidden neurons cannot model the data properly, which results in very inaccurate predictions. On the other hand, a SLFN with too many hidden neurons will lead to over-fitting, which means the trained model of the SLFN has really bad generalization ability, it can only approximate the training data in high accuracy.

The general method to improve the generalization ability and avoid over-fitting is regularization. The principle of the regularization for ELM is penalizing the coefficients of the output weight matrix or the output vector besides minimizing the output error. In other words, in the learning procedure, we need not only to minimize the term $\|\mathbf{H}\hat{\mathbf{B}} - \mathbf{Y}\|$, but to minimize the norm $\|\hat{\mathbf{B}}\|$ as well. Therefore, the cost function of ELM with regularization can be written in such form:

$$C_{RELM} = \frac{1}{2}\|\hat{\mathbf{B}}\|^2 + \frac{1}{\lambda} \cdot \frac{1}{2} \sum_{i=1}^N \|\xi_i\|^2 \quad (3.15)$$

with : $\xi_i^T = \mathbf{y}_i^T - \mathbf{h}(x_i)\hat{\mathbf{B}} \quad i = 1, 2, \dots, N$

The purpose of the training procedure can be formulated as:

$$\begin{aligned} \text{minimize : } C_{RELM} &= \frac{1}{2}\|\hat{\mathbf{B}}\|^2 + \frac{1}{\lambda} \cdot \frac{1}{2} \sum_{i=1}^N \|\xi_i\|^2 & (3.16) \\ \text{subject to : } \xi_i^T &= \mathbf{y}_i^T - \mathbf{h}(x_i)\hat{\mathbf{B}} \quad i = 1, 2, \dots, N \end{aligned}$$

Based on Karush-Kuhn-Tucker (KKT) theorem proposed by Fletcher (1980), to train a ELM with regularization is equivalent to solving the following dual optimization problem:

$$\mathcal{L}_{C_{RELM}} = \frac{1}{2} \|\hat{\mathbf{B}}\|^2 + \frac{1}{\lambda} \cdot \frac{1}{2} \sum_{i=1}^N \|\xi_i\|^2 - \sum_{i=1}^N \sum_{j=1}^m a_{ij} (\mathbf{h}(\mathbf{x}_i) \hat{\beta}_j - \mathbf{y}_{ij} + \xi_{ij}) \quad (3.17)$$

where each Lagrange multiplier vector α_i corresponds to the i^{th} training sample and $\alpha = [\alpha_1, \alpha_2, \dots, \alpha_N]$. $\hat{\beta}_j$ is the estimated output weight vector, which links the hidden layer to the j^{th} output neuron and the estimated output weight matrix is $\hat{\mathbf{B}} = [\hat{\beta}_1, \hat{\beta}_2, \dots, \hat{\beta}_N]$. Then we can have the KKT corresponding optimality conditions as follows:

$$\frac{\partial \mathcal{L}_{C_{ELM}}}{\partial \hat{\beta}_j} = 0 \rightarrow \hat{\beta}_j = \sum_{i=1}^N \alpha_{ij} \mathbf{h}^T(\mathbf{x}_i) \rightarrow \hat{\mathbf{B}} = \mathbf{H}^T \alpha \quad (3.18)$$

$$\frac{\partial \mathcal{L}_{C_{ELM}}}{\partial \xi_j} = 0 \rightarrow \alpha_i = \frac{1}{\lambda} \cdot \xi_i \quad (3.19)$$

$$\frac{\partial \mathcal{L}_{C_{ELM}}}{\partial \alpha_j} = 0 \rightarrow \mathbf{h}(\mathbf{x}_i) \hat{\mathbf{B}} - \mathbf{y}_i^T + \xi_i^T = \mathbf{0} \quad (3.20)$$

$$i = 1, 2, \dots, N \quad j = 1, 2, \dots, m$$

By substituting equation (3.19) into equation (3.18), the output weight matrix can be expressed as:

$$\hat{\mathbf{B}} = \frac{1}{\lambda} \mathbf{H}^T \xi \quad (3.21)$$

Then we can rewrite equation (3.21) by using Moore-Penrose generalized inverse as follows:

$$\xi = \lambda (\mathbf{H}^T)^\dagger \hat{\mathbf{B}} \quad (3.22)$$

From equation (3.20), we can obtain:

$$\mathbf{H} \hat{\mathbf{B}} - \mathbf{Y} + \xi = \mathbf{0} \quad (3.23)$$

By substituting equation (3.22) into equation (3.23), we have:

$$\begin{aligned} \mathbf{H} \hat{\mathbf{B}} - \mathbf{Y} + \lambda (\mathbf{H}^T)^\dagger \hat{\mathbf{B}} &= \mathbf{0} \implies \\ \mathbf{H} \hat{\mathbf{B}} + \lambda (\mathbf{H}^T)^\dagger \hat{\mathbf{B}} &= \mathbf{Y} \implies \\ \mathbf{H} \hat{\mathbf{B}} + \lambda (\mathbf{H}^\dagger)^T \hat{\mathbf{B}} &= \mathbf{Y} \implies \\ \mathbf{H} \hat{\mathbf{B}} + \lambda [(\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T]^T \hat{\mathbf{B}} &= \mathbf{Y} \implies \\ \mathbf{H} \hat{\mathbf{B}} + \lambda \mathbf{H} (\mathbf{H}^T \mathbf{H})^{-1} \hat{\mathbf{B}} &= \mathbf{Y} \implies \\ \mathbf{H}^T \mathbf{H} \hat{\mathbf{B}} + \lambda \mathbf{H}^T \mathbf{H} (\mathbf{H}^T \mathbf{H})^{-1} \hat{\mathbf{B}} &= \mathbf{H}^T \mathbf{Y} \implies \\ \mathbf{H}^T \mathbf{H} \hat{\mathbf{B}} + \lambda \hat{\mathbf{B}} &= \mathbf{H}^T \mathbf{Y} \\ \Downarrow \\ \hat{\mathbf{B}} &= (\mathbf{H}^T \mathbf{H} + \lambda \mathbf{I})^{-1} \mathbf{H}^T \mathbf{Y} \end{aligned} \quad (3.24)$$

According to the output weight matrix β derived by equation (3.24), we can compute the outputs of ELM by using the following equation:

$$\text{Output}(x) = h(x)\hat{B} = h(x)(H^T H + \lambda I)^{-1} H^T Y \quad (3.25)$$

With equation (3.25), it is now possible for us to deal with real-life problems in regression analysis and image multi-class classification. But in usual applications, people just choose an empirical value, such as 0.5 or 1, as the regularization parameter λ . It cannot be repudiated that using the empirical value as the regularization parameter can serve as a manner of improving generalization ability and avoiding over-fitting, but it cannot be guaranteed that choosing the empirical value as regularization parameter can always improve the performance and accuracy of ELMs. In some particular cases, the empirical value may lead to over-regularization, which will result in even worse performance and accuracy than normal ELMs.

So, it is important to select a proper regularization parameter for the ELM. However, there is no general method to choose a proper or an optimal regularization parameter so far. Deng et al. (2009) have proposed a numerical heuristic method based on cross-validation to determine the regularization parameter. The specific procedure can be described as follows:

Regularization method proposed by Deng et al. (2009)

Given a training set with N distinct samples,

$x = \{(x_i, y_i) | x_i \in \mathbf{R}^n, y_i \in \mathbf{R}^m, i = 1, 2, \dots, N\}$, activation function $\varphi(x)$, and the number of hidden neurons L

1. Randomize all the training samples and then divide them into two portion by a ratio of three to one. The portion with most training samples still serve as the training set (x_{train}, y_{train}) . The other portion does duty for a validation set (x_{val}, y_{val}) .
2. Choose all values in the interval $[2^{-50}, 2^{-49}, 2^{-48}, \dots, 2^{48}, 2^{49}, 2^{50}]$ as the regularization parameter, then train the regularized ELM with different regularization parameter and compute the corresponding output weight matrix $\hat{\beta}$ with the training set.
3. Compute the outputs of the validation set with each estimated output weight matrix:

$$\text{Output}(x_{val}) = h(x_{val})\hat{B} = h(x_{val})(H^T H + \lambda I)^{-1} H^T Y_{train}$$

4. Test the performance and accuracy of regularized ELM with different regularization parameter on the validation set. For each regularized ELM with different regularization parameter, compute the root-mean-square error (RMSE):

$$\text{RMSE} = \sqrt{\frac{\| \text{Output}(x_{val}) - Y_{val} \|^2}{n_{val}}} \quad (3.26)$$

5. Choose the value as the regularization parameter which makes the corresponding regularized ELM perform best and minimizes the RMSE in the validation set.

There are two vital drawbacks of this numerical heuristic method.

- It takes a lot of time to traverse all the candidate regularization parameter. For the interval $[2^{-50}, 2^{-49}, 2^{-48}, \dots, 2^{48}, 2^{49}, 2^{50}]$ mentioned above, we have to train 101 regularized ELMs. In real-life applications with a large amount of training data, which is in high-dimensional feature space, this method consumes such long time that it violates the purpose of ELM, which aim to shorten the training time significantly.
- In addition, it is possible that a proper regularization parameter cannot be found and determined with such method. The figure 3.1 shows an example, in which the regularization parameter is determined by this numerical heuristic method. However, in some particular case, the regularization may locate at the exact middle of the interval, for example, $[2^{30}, 2^{31}]$. It is not likely to find this regularization parameter, since we have not tried any other values in this interval besides 2^{30} and 2^{31} , so that we cannot obtain a similar curve as in 3.1 demonstrated to find the minimum of RMSE.

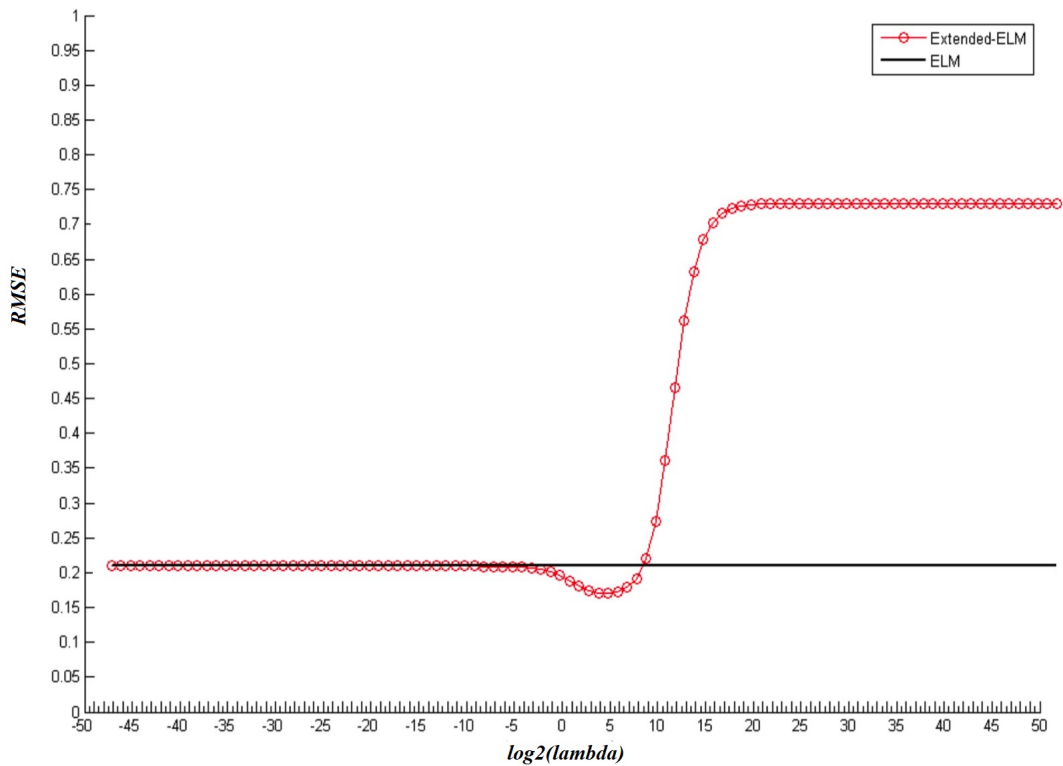


Figure 3.1: Relationship between regularization λ and RMSE in validation set

Chapter 4

A-optimal Design Regularization and its Application in ELM

As it is discussed in the previous chapter, a proper or even optimal regularization parameter is necessary for controlling the weight given to the term of quadratic norm of the coefficients of the output weight matrix, relative to the minimization of the term of the quadratic norm of the output errors. So far, there has been left the open problem to evaluate the regularization parameter for balancing the these two quadratic terms. On one hand, it is insufficient to overcome the over-fitting problems of normal ELMs when the value of the regularization parameter is too small. On the other hand, it will lead to under-fitting so that we cannot approximate the data in high accuracy, when the regularization parameter is of large value.

In order to acquire an optimal regularization parameter, we apply A-optimal design regularization, proposed by Cai (2004), to determinate the regularization parameter. Before introducing the A-optimal design regularization, it is necessary to review the theory of λ -weighted biased linear estimation.

4.1 λ -weighted Best Linear Estimation

For normal ELMs without any regularization, the least squares solution of the output weight matrix B according to Moore-Penrose generalized inverse is a type of best linear uniform unbiased estimate (BLUUE). But now, we aim to estimate the solution of the output weight matrix by giving up the unbiasedness and keeping the set-up of a linear estimate $\hat{B} = LY$ of homogeneous type, which is based on hybrid norm optimization of type:

- i minimum variance,
- ii minimum bias.

This type of linear estimate is name λ -weighted homogeneously Best Linear Estimate (λ -homBLE). This biased estimation is also called as Tikhonov-Phillips regularization

(Tikhonov (1963), Phillips (1962)). The regularization parameter determination is systematically developed by Cai et al. (2004) using A-optimal design regularization of type:

$$\begin{aligned} & \text{Minimize the trace of the } \lambda, S \text{ modified Mean Square Error matrix (MSE):} \\ & \text{tr}(MSE_{\lambda, S}(\hat{\mathbf{B}})) \text{ of the output weight matrix } \hat{\mathbf{B}}(\lambda - \text{homBLE}) \text{ to find} \\ & \lambda_{opt} = \text{argtr}(MSE_{\lambda, S}(\hat{\mathbf{B}})) = \min \end{aligned}$$

It is essential to understand the composition of the MSE matrix. Therefore, the bias vector and the bias matrix as well as of the MSE matrix with respect to the homogeneously linear estimate $\hat{\mathbf{B}} = \mathbf{L}\mathbf{y}$ of fixed effect \mathbf{B} , which represents the real value, in the following box.

dispersion matrix

$$D(\hat{\mathbf{B}}) = LD(\mathbf{Y})L^T \quad (4.1)$$

bias vector

$$\begin{aligned} \mathbf{b} &= E(\hat{\mathbf{B}} - \mathbf{B}) = E(\hat{\mathbf{B}}) - \mathbf{B} \\ &= LE(\mathbf{Y}) - \mathbf{B} = -(\mathbf{I} - \mathbf{LH})\mathbf{B} \end{aligned} \quad (4.2)$$

bias matrix

$$\mathbf{Bias} = \mathbf{I} - \mathbf{LH} \quad (4.3)$$

decomposition

$$\begin{aligned} \hat{\mathbf{B}} - \mathbf{B} &= (\hat{\mathbf{B}} - E(\hat{\mathbf{B}})) + (E(\hat{\mathbf{B}}) - \mathbf{B}) \\ &= L(\mathbf{Y} - E(\mathbf{Y})) - (\mathbf{I} - \mathbf{LH})\mathbf{B} \end{aligned} \quad (4.4)$$

MSE matrix

$$\begin{aligned} MSE(\hat{\mathbf{B}}) &= E((\hat{\mathbf{B}} - \mathbf{B})(\hat{\mathbf{B}} - \mathbf{B})^T) \\ &= LD(\mathbf{Y})L^T + (\mathbf{I} - \mathbf{LH})\mathbf{B}\mathbf{B}^T(\mathbf{I} - \mathbf{LH})^T \end{aligned} \quad (4.5)$$

The bias vector \mathbf{b} is conventionally defined by $E(\hat{\mathbf{B}} - \mathbf{B})$ subject to the homogeneously linear estimate $\mathbf{B} = \mathbf{L}\mathbf{Y}$. Since the fixed effects β is unknown in general situation, there has been made the proposal to instead use the matrix $\mathbf{I} - \mathbf{LH}$ as a matrix-valued measurement of the bias. The MSE matrix $MSE(\hat{\mathbf{B}})$ can be decomposed into two parts:

- the dispersion matrix $D(\hat{\mathbf{B}}) = LD(\mathbf{Y})L^T$

- the dynamic bias product bb^T

After introducing the MSE matrix, we can present λ -homBLE with hybrid minimum variance-minimum bias in the following definition

Definition(α -homBLE)

1. $\hat{\mathbf{B}}$ is a homogeneously linear form

$$\hat{\mathbf{B}} = \mathbf{L}\mathbf{Y}$$

2. in comparison to all other homogeneously linear estimate, $\hat{\mathbf{B}}$ has the minimum variance-minimum bias property in the sense of the λ -weighted hybrid norm

$$\begin{aligned} \|\mathbf{MSE}_{\lambda,S}(\hat{\mathbf{B}})\|^2 &= \text{trace}(\mathbf{L}\boldsymbol{\Sigma}_Y\mathbf{L}^T) + \frac{1}{\lambda}\text{trace}((\mathbf{I} - \mathbf{L}\mathbf{H})\mathbf{S}(\mathbf{I} - \mathbf{L}\mathbf{H})^T) \quad (4.6) \\ &= \|\mathbf{L}^T\|_{\boldsymbol{\Sigma}_Y}^2 + \frac{1}{\lambda}\|\mathbf{I} - \mathbf{L}\mathbf{H}\|_{\mathbf{S}}^2 = \min \end{aligned}$$

λ -homBLE is based upon the weighted sum of two norms:

- average variance $\|\mathbf{L}^T\|_{\boldsymbol{\Sigma}_Y}^2 = \text{trace}(\mathbf{L}\boldsymbol{\Sigma}_Y\mathbf{L}^T)$
- average bias $\frac{1}{\lambda}\|\mathbf{I} - \mathbf{L}\mathbf{H}\|_{\mathbf{S}}^2 = \frac{1}{\lambda}\text{trace}((\mathbf{I} - \mathbf{L}\mathbf{H})\mathbf{S}(\mathbf{I} - \mathbf{L}\mathbf{H})^T)$

λ -homBLE balances the variance and the bias by factor λ , which is illustrated by the following figure.

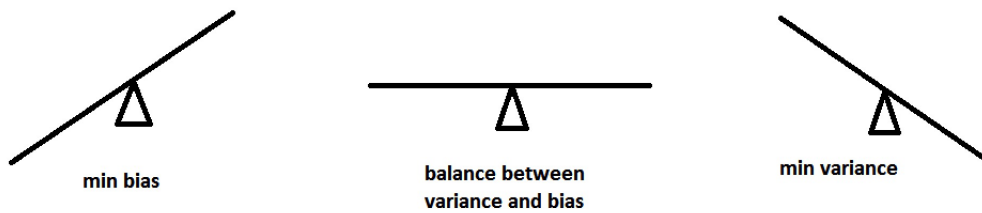


Figure 4.1: Balance of the variance and the bias by the weighting factor λ ; source: Cai (2004)

The hybrid norm $\|\mathbf{MSE}_{\lambda,S}(\hat{\mathbf{B}})\|^2$ established the Lagrangian for $\hat{\mathbf{B}}$ as λ -homBLE of x .

$$\mathcal{L}(\mathbf{L}) := \text{trace}(\mathbf{L}\boldsymbol{\Sigma}_Y\mathbf{L}^T) + \frac{1}{\lambda}\text{trace}((\mathbf{I} - \mathbf{L}\mathbf{A})\mathbf{S}(\mathbf{I} - \mathbf{L}\mathbf{A})^T) = \min \quad (4.7)$$

Then equivalent representation of the solution of normal equation is

$$\hat{\mathbf{B}} = (\mathbf{H}^T \boldsymbol{\Sigma}_Y \mathbf{H} + \lambda \mathbf{S}^{-1})^{-1} \mathbf{H}^T \boldsymbol{\Sigma}_Y^{-1} \mathbf{Y} \quad (4.8)$$

For ELMs, all training samples are equal weighted, so the weight matrix $\boldsymbol{\Sigma}_Y = \mathbf{I}$. In addition, the unit matrix \mathbf{I} is chosen as the regularization matrix \mathbf{S} . Thus, we can rewrite the equation (4.8) as

$$\hat{\mathbf{B}} = (\mathbf{H}^T \mathbf{H} + \lambda \mathbf{I}^{-1})^{-1} \mathbf{H}^T \mathbf{Y} \quad (4.9)$$

Complemented by the dispersion matrix

$$\mathbf{D}(\hat{\mathbf{B}}) = (\mathbf{H}^T \mathbf{H} + \lambda \mathbf{I})^{-1} \mathbf{H}^T \mathbf{H} (\mathbf{H}^T \mathbf{H} + \lambda \mathbf{I})^{-1} \quad (4.10)$$

by the bias vector

$$\begin{aligned} \mathbf{b} &= E(\hat{\mathbf{B}} - \mathbf{B}) \\ &= -(\mathbf{I} - (\mathbf{H}^T \mathbf{H} + \lambda \mathbf{I})^{-1} \mathbf{H}^T \mathbf{H}) \mathbf{B} \\ &= -\lambda (\mathbf{H}^T \mathbf{H} + \lambda \mathbf{I})^{-1} \mathbf{B} \end{aligned} \quad (4.11)$$

and by the MSE matrix

$$\begin{aligned} \mathbf{MSE}(\hat{\mathbf{B}}) &= E((\hat{\mathbf{B}} - \mathbf{B})(\hat{\mathbf{B}} - \mathbf{B})^T) = \mathbf{D}(\hat{\mathbf{B}}) + \mathbf{b}\mathbf{b}^T \\ &= (\mathbf{H}^T \mathbf{H} + \lambda \mathbf{I})^{-1} \mathbf{H}^T \mathbf{H} (\mathbf{H}^T \mathbf{H} + \lambda \mathbf{I})^{-1} \\ &\quad + [(\mathbf{H}^T \mathbf{H} + \lambda \mathbf{I})^{-1} \lambda \mathbf{I}] \mathbf{B} \mathbf{B}^T [\lambda \mathbf{I} (\mathbf{H}^T \mathbf{H} + \lambda \mathbf{I})^{-1}] \\ &= (\mathbf{H}^T \mathbf{H} + \lambda \mathbf{I})^{-1} [\mathbf{H}^T \mathbf{H} + (\lambda \mathbf{I}) \mathbf{B} \mathbf{B}^T (\lambda \mathbf{I})] (\mathbf{H}^T \mathbf{H} + \lambda \mathbf{I})^{-1} \end{aligned} \quad (4.12)$$

4.2 A-optimal design Regularization

In order to minimize the trace of MSE matrix, the A-optimal design regularization is applied to determine the optimal regularization parameter, which optimize the the trace of MSE matrix. The regularization parameter λ follows by A-optimal design regularization in the sense of minimizing the trace of MSE matrix ($\text{trace}(\mathbf{MSE}(\hat{\mathbf{B}})) = \min$), if and only if

$$\hat{\lambda} = \frac{\text{trace}(\mathbf{H}^T \mathbf{H} (\mathbf{H}^T \mathbf{H} + \hat{\lambda} \mathbf{I})^{-3})}{\mathbf{B}^T (\mathbf{H}^T \mathbf{H} + \hat{\lambda} \mathbf{I})^{-2} \mathbf{H}^T \mathbf{H} (\mathbf{H}^T \mathbf{H} + \hat{\lambda} \mathbf{I})^{-1} \mathbf{B}} \quad (4.13)$$

The proof of the formula of type 4.13 will be given in the appendix. Moreover, we will interpreted how the A-optimal design regularization is applied in improving the performance of ELMs. We will discuss in two different cases.

1), ELMs with Single Output

For the cases, where there is only a single output neuron in the output layer, the output of an ELM is a single numerical consequence, for example, regression problems and binary classifications. In such cases, the number of output neurons is $m = 1$, therefore, the output weight is a $L \times 1$ vector.

For ELMs with single output, A-optimal design regularization can be directly used to determine the regularization parameter.

2), ELMs with Multiple Outputs

When we use ELMs to classify images into m classes in multi-class classification, the ELMs will have m output neurons. The output of the ELM is a vector rather than a single number. The expected output vector of the m output neurons is $\mathbf{y}_i = [0, \dots, 0, \overset{p}{1}, 0, \dots, 0]$, if the original class label is p . Specifically, only the p^{th} element of $\mathbf{y}_i = [y_{i1}, y_{i2}, \dots, y_{im}]^T$ is one, while the rest of the elements are set to zero.

In such cases, the A-optimal design regularization cannot be applied to compute the regularization parameter for the ELMs directly, since the denominator in the right hand of equation (4.13) is a matrix, however, the numerator is rather a number.

In order to apply A-optimal design regularization in ELMs for multiple outputs, we have to go back to equation (3.24).

$$\hat{\mathbf{B}} = (\mathbf{H}^T \mathbf{H} + \lambda \mathbf{I})^{-1} \mathbf{H}^T \mathbf{Y}$$

In such equation demonstrated above, we have the output \mathbf{Y} as a $N \times m$ matrix, when there are N distinct training samples and m output neurons. As it is already discussed in chapter 3, when the number of the hidden neurons in ELMs counts L , the output weight is thus a $L \times m$ matrix. According to Cai (2004), the equation (3.24) can be decomposed into the following equations.

$$\begin{aligned} \hat{\boldsymbol{\beta}}_1 &= (\mathbf{H}^T \mathbf{H} + \lambda_1 \mathbf{I})^{-1} \mathbf{H}^T \mathbf{y}_1 \\ \hat{\boldsymbol{\beta}}_2 &= (\mathbf{H}^T \mathbf{H} + \lambda_2 \mathbf{I})^{-1} \mathbf{H}^T \mathbf{y}_2 \\ &\vdots \\ \hat{\boldsymbol{\beta}}_m &= (\mathbf{H}^T \mathbf{H} + \lambda_m \mathbf{I})^{-1} \mathbf{H}^T \mathbf{y}_m \end{aligned} \tag{4.14}$$

The i^{th} column of the output matrix and the output weight is corresponding $\mathbf{y}_i = [y_{1i}, y_{2i}, \dots, y_{Ni}]$ and $\hat{\boldsymbol{\beta}}_i = [\hat{\beta}_{1i}, \hat{\beta}_{2i}, \dots, \hat{\beta}_{Li}]$, for $i = 1, 2, \dots, m$. We can determine the regularization parameter λ_i and the corresponding column $\hat{\boldsymbol{\beta}}_i$ of the output weight matrix by the A-optimal design regularization, for $i = 1, 2, \dots, m$.

Chapter 5

Case Studies: Application of A-optimal Design Regularization in Extreme Learning Machine

In order to inspect whether A-optimal design regularization helps to improve the performance of ELMs or not, we implement three case studies to research ELMs with A-optimal design regularization and compare the performance of regularized ELMs with the original ones.

5.1 Approximation of Sine Function

The first case study is an artificial simulation about approximating sine function $y = f(x) = \sin(x)$ by both regularized and original ELMs. We will approximate the sine function in two different situations.

- In the first situation, there is no outliers in the training set. The training set is created by 5000 distinct samples of sine function, which are uniformly and randomly distributed on the interval $(-10, 10)$. In addition, random noise distributed on the interval $[-0.2, 0.2]$ will be added to all the training samples. Similarly, 5000 distinct samples are randomly chosen to generate the testing set, while all the testing samples still remain noise-free.
- In the second situation, 100 outliers randomly distributed in the range of $[-2, 2]$ will be added into the training set. Moreover, we choose other 4900 distinct samples with the same method as used in the first situation to constitute the training set. As same as the first situation, the testing set is composed of 5000 noise-free distinct samples of the sine function.

In the following table, the information of training and testing data in both situation is listed in detail.

Table 5.1: Data Information for Approximation of the Sine Function

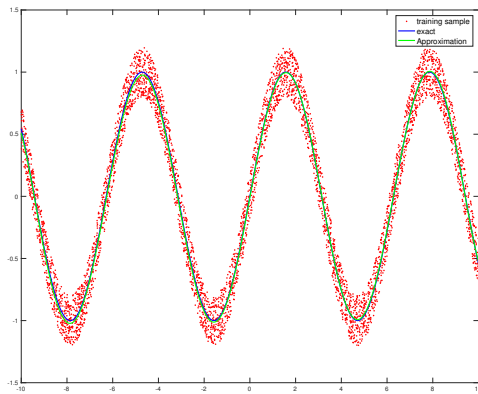
	Training Set			Testing Set	
without outliers	distinct samples	outliers	noisy	distinct samples	noisy
	5000	0	yes	5000	no
with outliers	distinct samples	outliers	noisy	distinct samples	noisy
	4900	100	yes	5000	no

In both situations, we train the original ELM and the A-optimal design regularized one with the training set to approximate the sine function respectively. Moreover, the performance of both algorithms will be evaluated on the corresponding testing set in each situation. Specifically, RMSE for the testing set of both algorithms will be computed and compared in each situation. The following table shows us the comparison of the RMSE between both algorithms.

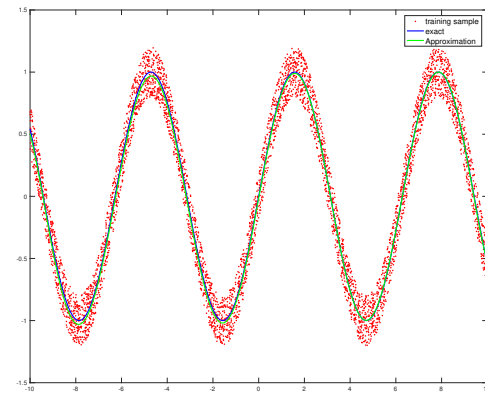
Table 5.2: Comparison of RMSE of the original ELM and the ELM with A-optimal design regularization in both situations

RMSE	without outliers		with outliers	
	Original ELM	Regularized ELM	Original ELM	Regularized ELM
Training Set	0.1150	0.1157	0.2566	0.2405
Testing Set	0.0145	0.0148	0.1006	0.0426

From the table above, it is obvious that there is no significant difference between both algorithms, when there is no outliers. Both algorithms have good performances and the RMSE for the testing set of both algorithms are very low. But for the situation, where there is outliers in the training set, the ELM with A-optimal design regularization performs much better than the original one. It implies that the performance of the original ELM is seriously influenced by the outliers. While for the A-optimal regularized ELM, honestly, the performance is more or less affected by the outliers as well. However, the RMSE for the testing set is still very low. In other words, the the ELM with A-optimal design regularization is much more robust against the outliers than the original ELM, it can approximate the sine function in high precision even when there are outliers in the training set. In figure 5.2, the difference of the performances of both algorithms is illustrated much more apparent.

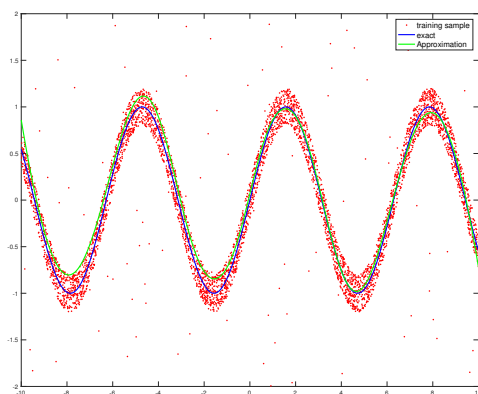


Original ELM

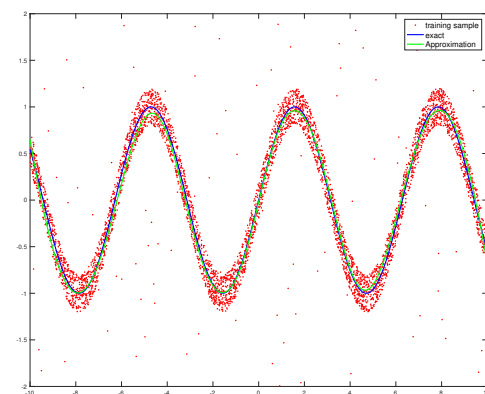


The A-optimal regularized ELM

Figure 5.1: Approximation of the sine function with both algorithms without outliers



Original ELM



The A-optimal regularized ELM

Figure 5.2: Approximation of the sine function with both algorithms with outliers

Figure 5.1 and 5.2 illustrate the results of approximation of the sine function. All red points in the figure 5.1 and 5.2 refer to as the distinct training samples. In figure 5.2, the red speckle symbolize the outliers. The blue curves imply the standard sine function, while the green curves represent our results of approximation of the sine function. In figure 5.1, it is illustrated very obviously that, the original ELM and the ELM with A-optimal design regularization can both approximate the sine function with rather high precision. There are only some delicate differences between the curve of the sine function and the curve of our approximation, no matter which algorithm we use. However, in the situation where there are outliers in the training set, the ELM with A-optimal design regularization is able to approximate the sine function with higher accuracy. In figure 5.2, it is legible that, the difference between the approximation by the original

ELM and the standard sine function cannot be neglected at all, especially for the inflection points of the sine function, while the approximation error of the ELM with A-optimal design regularization is much less. We can also find some differences at the inflection points, though it is within the tolerance range of error and has only subtle influences on the holistic performance.

5.2 Regression Analysis in Real-world Problems

In the second case study, we apply both of the original ELM and the one with A-optimal design regularization in analyzing and figuring out real-world regression problems. All the datasets are downloaded from the website of a statistical library: <http://www.liaad.up.pt/ltorgo/Regression/DataSets.html>. The specification of all the datasets used in this case study is enumerated in the table 5.3.

Table 5.3: Specification of the datasets used for regression analysis

Dataset	Number of training samples	Number of testing samples	Number of features
Bank domains	4500	3692	32
Puma	4499	3693	32
Triazines	124	62	60
Pyrim	49	25	27
Machine CPU	139	70	6
Kinematic	5461	2731	8
California housing	13760	6880	8
Stocks domain	633	317	9
Fried_delve	27179	13589	10

From the table above, we can be aware of the information of each dataset roughly. Each dataset describes a regression problem in real life. For instance, the dataset of bank domains conclude a family of data generated from a simulation of how bank-customers choose their banks. In this application, tasks are based on predicting the fraction of bank customers who leave the bank because of full queues. The bank family of the data is generated from a simplistic simulator, which simulates the queues in a series of banks. Customers come from several residential areas, choose their preferred bank depending on distances and have tasks of varying complexity, and various levels of patience. Each bank has several queues, that whether open or close is according to demand. The tellers have various influences, and customers may change queue, if their patience expires. The object is to predict the rate of rejections, i.e. the fraction of customers that are turned away from the bank because all the open tellers have full queues.

Similarly, other datasets listed in table 5.3 contain the information about the corresponding regression applications. In each dataset, the samples are described with some corresponding numerical features. In addition, for each sample, there is a numerical target as well.

For each dataset, we adopt both the original ELM and the one with A-optimal design regularization to train the neural network in the training set. In the input layer of the neural network, the input neurons are filled with the numerical features. Since there is only single target of each sample in regression applications, the output layer is filled with single output neuron, which involves the target of the training sample. With all the training samples, we can estimate the solution of the output weight by using both algorithms.

Afterwards, we used the trained neural network to predict the output of the samples in the testing set. The performance of the trained neural network is inspected on the root mean square error (RMSE) of the samples in the testing set.

$$RMSE = \sqrt{\frac{\sum_{i=1}^N (o_i - y_i)^2}{n}} \quad (5.1)$$

o_i is output predicted by the trained neural network, y_i is the numerical target of the i^{th} sample. By comparing the prediction to the corresponding target, we can calculate the RMSE of the samples in the testing set. The lower the RMSE is, the more accurate is the prediction. Beyond that, we the trace of MSE matrix of the estimated output weight is calculated as well, so that we can figure it out how the A-optimal design regularization contributes to improve the accuracy of predictions for ELM in regression analysis. Before experiments, all numerical features and targets are already normalized to the interval $(-1,1)$.

The table 5.4 presents the RMSE of the predictions for the training data as well as the trace of MSE matrix of the output weight. The RMSE of predictions for all the testing data is listed in the table 5.5.

Table 5.4: Results for training data: RMSE and trace(MSE($\hat{\mathbf{B}}$))

Training data	RMSE		trace(MSE($\hat{\mathbf{B}}$))	
	Original ELM	A-optimal Regularized ELM	Original ELM	A-optimal Regularized ELM
Bank domains	0.0795	0.0806	169.74	141.36
Puma	0.0245	0.0248	148.99	140.16
Triazines	0.1479	0.1494	69.34	62.29
Pyrim	0.0776	0.0780	59.96	51.79
Machine CPU	0.0461	0.0506	46.74	42.93
Kinematic	0.0891	0.0903	54.36	48.89
California housing	0.1221	0.1246	1699.76	1564.39
Stocks domain	0.0297	0.0311	1348.24	1289.01
Fried_delve	0.1976	0.2011	6491.16	6211.84

Table 5.5: Results for testing data: RMSE

Testing data	RMSE	
	Origin ELM	A-optimal regularized ELM
Bank domains	0.0901	0.0819
Puma	0.0296	0.0251
Triazines	0.1661	0.1391
Pyrim	0.1004	0.0876
Machine CPU	0.0594	0.0511
Kinematic	0.1021	0.0968
California housing	0.1256	0.1251
Stocks domain	0.0396	0.0316
Fried_delve	0.3169	0.2466

From both tables, we can find that the training data can be approximated with a higher accuracy by using the algorithm of the original ELM to train the neural network. While for the testing data, the original ELM performs worse than the regularized ELM, the predictions computed by the original ELM have higher RMSE.

As it is already alluded in the previous chapter, the generalization ability of the original ELM is barely satisfactory. When we apply the algorithm of the original ELM to train the neural networks in the training set, the objective is minimizing the training error, namely, the trained neural network is aimed to approximate the data in the training set as perfectly as possible. However, as it is discussed in the chapter 3, minimizing the training error may affect the performance in the testing set. In other words, the generalization ability is influenced, which will result in lower accuracy of the predictions in the testing data or some other con-generic data in real life.

By applying the A-optimal design regularization, the neural network is designed not only to minimize the training error in the training set, but to minimize the norm of the output weight as well. We do not aim to compute the predictions of the training data as accurate as possible, otherwise, we propose to train the neural network to possess good generalization ability. If we check the trace of MSE matrix of the estimated output weight in the table 5.4, we can find that the trace of MSE matrix of the estimated output weight derived by the A-optimal regularized ELM is lower, i.e. we can estimate the output weight with higher precision by using the A-optimal regularized ELM. Such output weight with higher precision improves the accuracy of the predictions of the testing data. As it is listed in the table 5.5, we can predict the output of the data in the testing set with lower RMSE. It consists with the aforementioned opinion that it can refine the predictions of the testing data if we estimate the output weight with higher precision.

5.3 Multi-class Classification in Satellite Images

The last case study is about classification applications in satellite images. We apply both of the original ELM and the ELM with A-optimal design regularization to carry out the classification in two study areas of different geographic features, respectively, Wuhan in China and Karlsruhe in Germany.

Case Study 1: Classification in Wuhan

Initially, we talk about our study in Wuhan. Our study area is a part of Wuhan, which is demonstrated in the figure 5.3. The light pink area within the legible boundary is Wuhan. The highlight red rectangle indicate our study area in Wuhan.

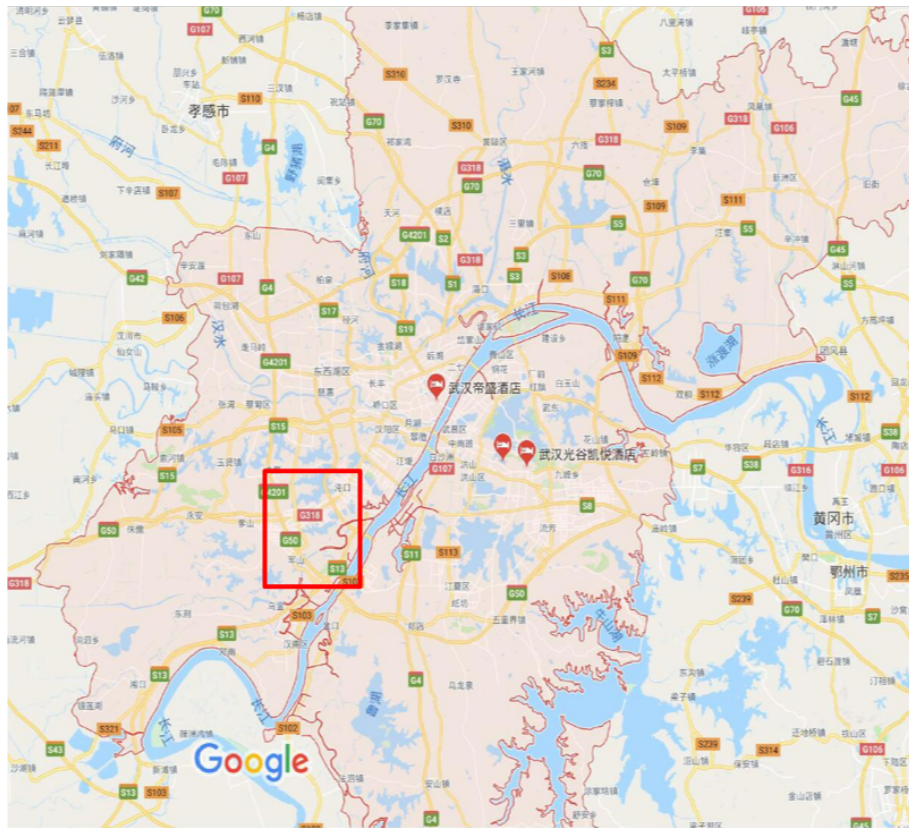


Figure 5.3: Position of the study area in Wuhan; source: Google Earth

In addition, we can observe the geographic features in figure 5.4, which is a satellite image from Google Earth with high resolution, about 10 meters. This satellite image with high resolution also helps us in labeling the pixels and generating distinct samples for training and testing set.



Figure 5.4: Satellite image of the study area in Wuhan; source: Google Earth

The study area is a plain and countryside area, where people cultivate paddies. As it is shown in the figure 5.4, around the land of the study area, there are many lakes in all size. In the land of the study area, most are clay instead of concrete. Based on such geographic features, we aim to classify the study area into 5 classes, respectively, bare land, water, trees, grass, and buildings. The roads in the study area in included in the bare land.

The following introduces the data we used for classification in detail. The satellite image to be classified is shot by Landsat 8 and downloaded from the website of USGS GLOVIS. The resolution of this satellite image is 30 meter and the satellite image of this study area contains 598×597 pixels. For this study area, the 7 spectral bands

are directly employed as the features for classification. The specification of the data information is listed in the following table.

Table 5.6: Specification of data information of the study area in Wuhan

Data Information	
Data source	https://glovis.usgs.gov
Image category	Landsat 8 satellite image
Image resolution	30 meters
Image size	598 × 597 pixels
Feature	7 spectral bands

In order to train the original ELM and the ELM with A-optimal design regularization and inspect the performance, we select 3550 pixels in the satellite image and label them as distinct samples. The number of the labeled pixels for each class is specified as follows

Table 5.7: Specification of labeled pixels of the study are in Wuhan

Number of labeled pixels for each class					
Building	Grass	Trees	Bare land	Water	Total
750	569	512	989	730	3550

In addition, 100 labeled pixels from each class are randomly chosen training samples to generate the training set. While the testing set is composed of the other 3050 labeled pixels. For the evaluation of the performance of both algorithms in this study area, we will firstly compute the confusion matrix in the testing set, which reveals the classification accuracy. The table 5.8 and 5.9 demonstrate the confusion matrix for the classification of the original ELM and the regularized ELM in the testing set respectively.

Table 5.8: Confusion matrix of the classification in the study are in Wuhan by using the original ELM

		Class from the reference data				
		Water	Bare land	Trees	Grass	Buildings
Class from the classification	Water	730	0	0	0	0
	Bare Land	80	890	1	7	11
	Trees	32	2	156	323	0
	Grass	0	0	0	569	0
	Buildings	31	7	0	0	712

In the confusion matrix, the diagonal elements represent the number of the correctly classified pixels, while the off-diagonals imply the miss-classification. From the results of classification displayed in the tables above, there is no doubt that the performance of the original ELM is not good enough at all to satisfy the standard requirement of the accuracy of classification. For the class of bare land, there are 80 pixels, which are classified incorrectly as water, in the total 989 pixels. In other words, there are about 10 percents are miss-classified, which can be counted as a classification error. The reason for such classification error may lie in that the roads and some bare lands under the shadow of trees have similar reflection with water, which results in similar spectral features with water. Therefore, some pixels of bare land are classified as water. Besides the miss-classification of bare land, the error of classification for the class of trees is much more serious. More than half of the pixels, which stand for trees, are miss-classified as grass. Such error of classification is completely not tolerant. The similarity between trees and grass in the satellite image may be responsible for the error. On the one hand, it is difficult for us to distinguish between trees and grass and label the pixels without mistakes in the satellite image. On the other hand, it is also tough for the neural networks to classify the pixels into the corresponding class correctly, because the spectral features of trees and grass are very similar to each other.

Table 5.9: Confusion matrix of the classification in the study area in Wuhan by using the regularized ELM

Class from the classification	Class from the reference data					
	Water	Bare land	Trees	Grass	Buildings	
Water	730	0	0	0	0	
Bare Land	2	946	31	3	7	
Trees	0	2	497	12	1	
Grass	0	0	4	565	0	
Buildings	2	18	3	12	715	

The confusion matrix of the classification with the regularized ELM is presented in the table 5.6. It is obvious that the ELM with A-optimal design regularization has much better performance. The A-optimal design regularization helps to improve the classification accuracy of ELM especially for the class of trees. From the table 5.6, we can find that only 2 pixels of bare land are classified incorrectly as water, when we apply the regularized ELM. Although there are more pixels of bare land are miss-classified as trees, the amount of miss-classification is still reduced. In particular, almost all the pixels of trees are classified correctly, which is a huge amelioration.

According to the confusion matrix, we can calculate the corresponding accuracy and cohens kappa coefficient of the classification by both algorithms respectively. The accuracy and the cohens kappa coefficient are the intuitionistic characters to evaluate the performance of the classifiers. The higher is the value of the accuracy and the cohens kappa coefficient, the better the classifiers perform.

Table 5.10: Accuracy and Cohens Kappa coefficient of the classification in the study area in Wuhan

	Accuracy	Cohens kappa coefficient
Original ELM	84.11%	0.7596
The A-optimal regularized ELM	97.27%	0.9504

From the table 5.10, it is conspicuous that we can classify the pixels in the testing set with much better performance when we apply the algorithm of the regularized ELM. There are dramatic improvements, especially in the cohens kappa coefficient.

On the side, we use the trained neural network by both algorithms to classify the whole image. The figure 5.5 demonstrate the satellite image to be classified. As it is already introduced, it is a Landsat 8 satellite image of low resolution.

**Figure 5.5:** Satellite image to be classified for study area in Wuhan

The figure 5.6 and 5.7 display the results of classification classified by both algorithms respectively.

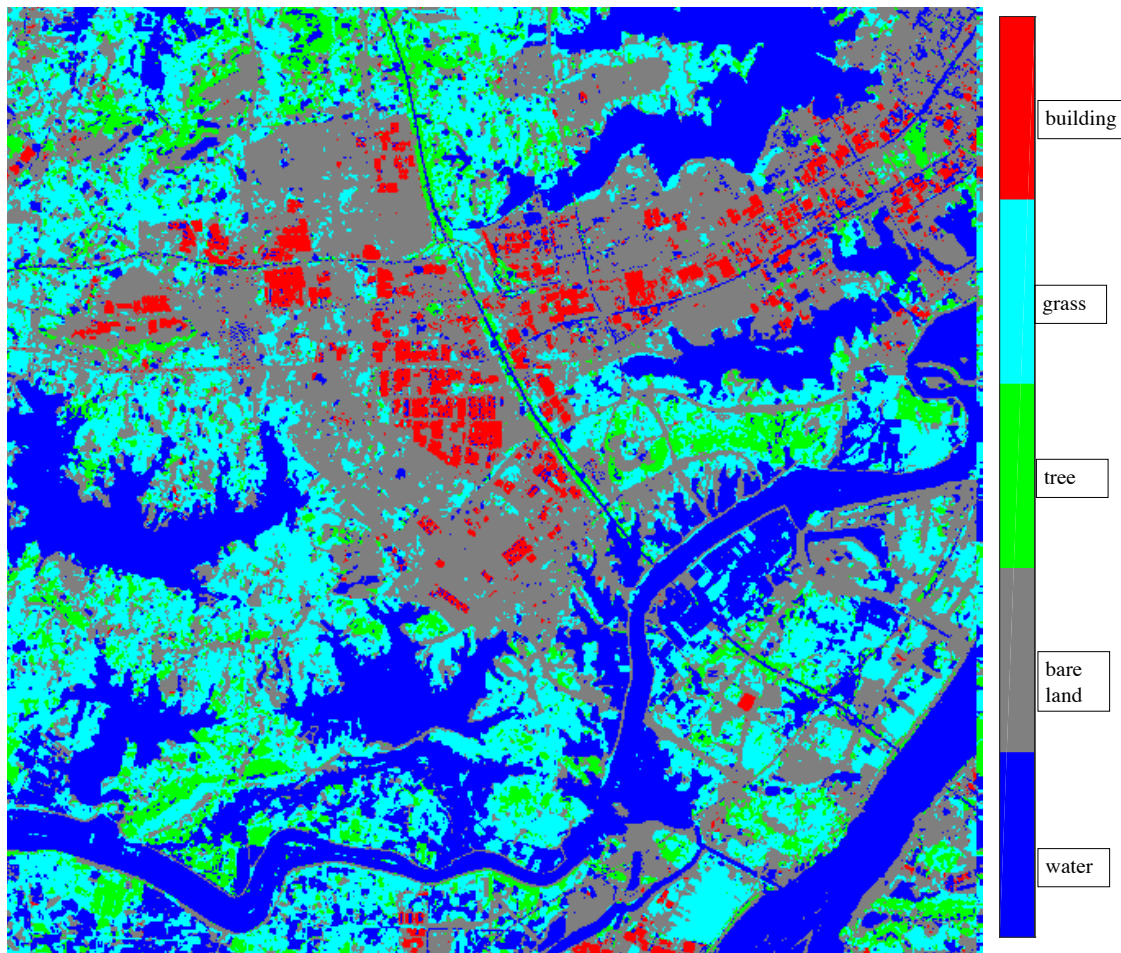


Figure 5.6: Classified study area in Wuhan by the original ELM

In the figure 5.6, most pixels, which represent trees, are classified incorrectly as grass, when we use the neural network trained by the algorithm of the original ELM. Besides, at the right top corner, many pixels of bare land are classified as water incorrectly, especially for some obvious main roads. However, in the figure 5.7, we can see the improvement unambiguously that most pixels in the satellite image of the study area in Wuhan are classified accurately, according to the figure 5.4, which is of high resolution. The errors, which appear in the classification by the algorithm of the original ELM, are amended when we apply the A-optimal regularization in the original ELM.

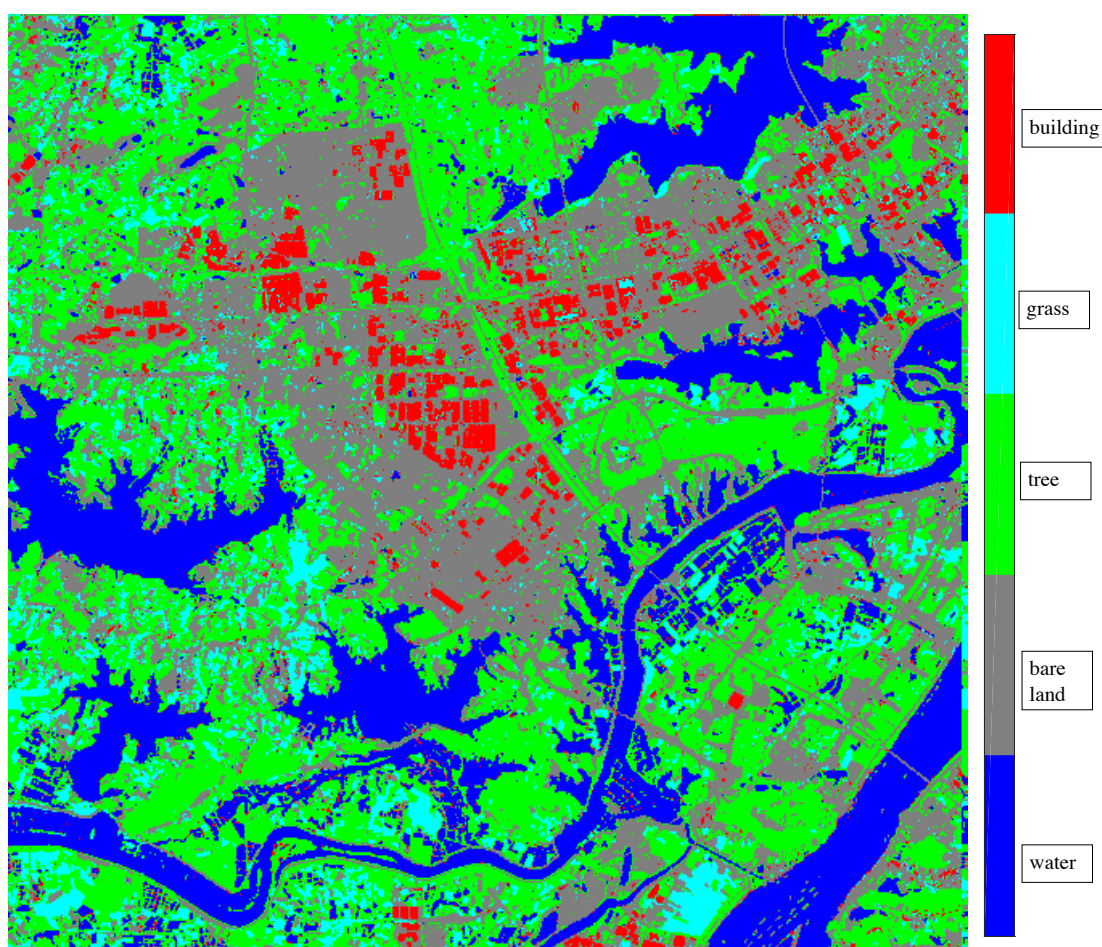


Figure 5.7: Classified study area in Wuhan by the ELM with A-optimal design regularization

When we apply the algorithm of the original ELM to train the SLFN, the estimate of the output weight $\hat{\mathbf{B}} = \mathbf{H}^{\dagger} \mathbf{Y} = (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T \mathbf{Y}$. In the application of classification, the condition number of the normal matrix is very large, which signifies that the normal matrix is ill-posed. When the normal matrix is ill-posed, we cannot estimate a stable and reliable solution of the output weight (Hansen, 1998). The solution of the output weight is very sensitive to the training set, some small errors in the training set can affect the performance of the classifier seriously. Specifically, we may make mistakes in labeling the pixels. In particular, it is in all probability that some pixels, which are on behalf of trees are labeled as grass by mistake and vice versa, because we cannot distinguish trees and grass in the satellite image of low resolution. As we can see in the figure 5.5, it is not feasible to label the pixels absolutely correctly, even if we have the satellite image of high resolution from Google Earth shoot in a same time period as an assistant. Therefore, in the training set, there are some pixels, which are labeled incorrectly and can be counted as outliers. Such outliers influence the performance of the classifier seriously when we adopt the algorithm of the original ELM. Whereas to the original ELM, when the ELM with A-optimal design regularization is adopted, it helps circumvent the difficulties and control the general instability of ill-posed prob-

lems. Hence, we can still classify the satellite image of the study area in Wuhan with high accuracy even though there are outliers in the training set.

Case Study 2: Classification in Karlsruhe

In addition to Wuhan, another study on classification is carried out in Karlsruhe in Germany. Other than Wuhan, Karlsruhe is a hilly area. Karlsruhe lies completely to the east of the Rhine, and almost completely on the Upper Rhine Plain. It contains the Turmberg in the east, and also lies on the borders of the Kraichgau leading to the Northern Black Forest. Our study area in Karlsruhe is mainly composed of a river, forest, cultivated land and urban region. Hence, we aim to classify the satellite image of the study area in Karlsruhe in six classes, respectively, bare land, buildings, forest, cultivated land, concrete land, and water. Here, the bare land implies only the soil area, where there are no vegetation. The cultivated land means the soil area with vegetation. The figure () demonstrates a satellite image from Google Earth of high resolution, which reveals the topography and landforms of the study area in Karlsruhe.



Figure 5.8: Satellite image of study area in Karlsruhe from Google Earth

The satellite image to be classified is downloaded from USGS Glovis as well. As same as previously, it is also shoot by Landsat 8. However, in such case study in Karlsruhe, principle components analysis (PCA) and Kautlr-Thomas Transformation (Kauth and Thomas, 1976) are adopted for building the feature space. We use PCA in the first spectral band of the satellite image shoot by Landsat 8 to extract the texture information, which helps the classifier to distinguish trees and grass. In this case study, it is conducive to distinguish the forest and the vegetation on the cultivated land. Similarly, Kautlr-Thomas Transformation is employed to extract features like soil brightness, greenness, yellow stuff and etc. Such features contribute to build a better feature space then the original spectral bands, which can conduce to a better result of classification. The data information is listed in the table 5.11.

Table 5.11: Specification of data information of the study area in Karlstuhe

Data Information	
Data source	https://glovis.usgs.gov
Image category	Landsat 8 satellite image
Image resolution	30 meters
Image size	641 × 661 pixels
Feature	15-dimensional feature space

With the same method, we label 2983 pixels in the satellite image to be classified of Landsat 8, respectively, 553 pixels of bare land, 520 pixels of buildings, 600 pixels of forest, 472 pixels of cultivated land, 426 pixels of concrete land and 416 pixels of water, which is specified in the following table.

Table 5.12: Specification of labeled pixels of the study area in Karlsruhe

Number of labeled pixels for each class						
Bare Land	Buildings	Forest	Cultivated Land	Concrete Land	Water	Total
553	520	600	472	426	416	2983

As same as the case study in Wuhan, 100 labeled pixels from each class are randomly selected to generate the training set, the testing set consist of the other 2383 labeled pixels. The table 5.13 and 5.14 reveal the confusion matrix of the classification in the testing set by using the algorithm of the original ELM and the regularized ELM respectively.

Table 5.13: Confusion matrix of the classification in the study are in Karlsruhe by using the original ELM

		Class from the reference data					
		Bare Land	Buildings	Forest	Cultivated Land	Concrete Land	Water
Class from the classification	Bare Land	486	64	2	0	1	0
	Buildings	23	472	0	1	24	0
	Forest	0	0	600	0	0	0
	Cultivated Land	0	0	1	471	0	0
	Concrete Land	66	5	150	6	189	0
	Water	0	0	0	0	20	402

From the confusion matrix numerated in the table above, all the pixels, which represent the forest, in the testing set are classified infallibly when the algorithm of the original ELM is used. Nevertheless, for the other classes, the classifier of the original ELM performs barely satisfactory, especially for the class of the concrete land. As we can see in the table 5.13, the amount of the correctly classified pixels of the concrete land is less than 50 percent. Specifically, only 189 pixels among all 426 pixels of the concrete land are classified correctly.

Table 5.14: Confusion matrix of the classification in the study are in Karlsruhe by using the regularized ELM

		Class from the reference data					
		Bare Land	Buildings	Forest	Cultivated Land	Concrete Land	Water
Class from the classification	Bare Land	534	9	0	0	10	0
	Buildings	8	495	0	0	17	0
	Forest	0	0	592	5	3	0
	Cultivated Land	0	0	2	470	0	0
	Concrete Land	2	7	6	0	401	0
	Water	0	0	1	0	0	421

Then we can check the accuracy of the classification, when the A-optimal design regularization is applied, from the table 5.14. For some single class, such as the forest, the pixels of the forest are classified with lower accuracy, but from an overall perspective, the pixels of each class are classified with more than 95 percent accuracy. There are only few off-diagonals, which means almost all the pixels in the testing set are classified correctly.

Table 5.15: Accuracy and Cohens Kappa coefficient of the classification in the study area in Karlsruhe

	Accuracy	Cohens kappa coefficient
Original ELM	87.83%	0.8528
The A-optimal regularized ELM	97.65%	0.9717

In addition, the table 5.15 shows us the accuracy and the cohens kappa coefficient clearly. It is obvious that building the 15-dimensional feature space is conducive to improve the performance the classifiers. Compared to the study on classification in Wuhan, the performance of neural network trained by the original ELM is meliorate a lot, when the 15-dimensional feature space is build. However, the performance of the neural network trained by the regularized ELM is still much better, about 10 percent higher in the accuracy and 12 percent high in the cohens kappa coefficient. Complex feature spaces cannot help to circumvent the ill-pose in the classification mentioned previously. In order to stabilize the solution of the output matrix in the neural network, when we use the ELM, it is necessary to apply the A-optimal design regularization.

The figure 5.9 is the satellite image of Landsat 8, which is to be classified.

**Figure 5.9:** Satellite image to be classified for study area in Karlsruhe

The figure 5.10 and 5.11 demonstrate the results of the classification in the study area in Karlsruhe by using different learning algorithms.

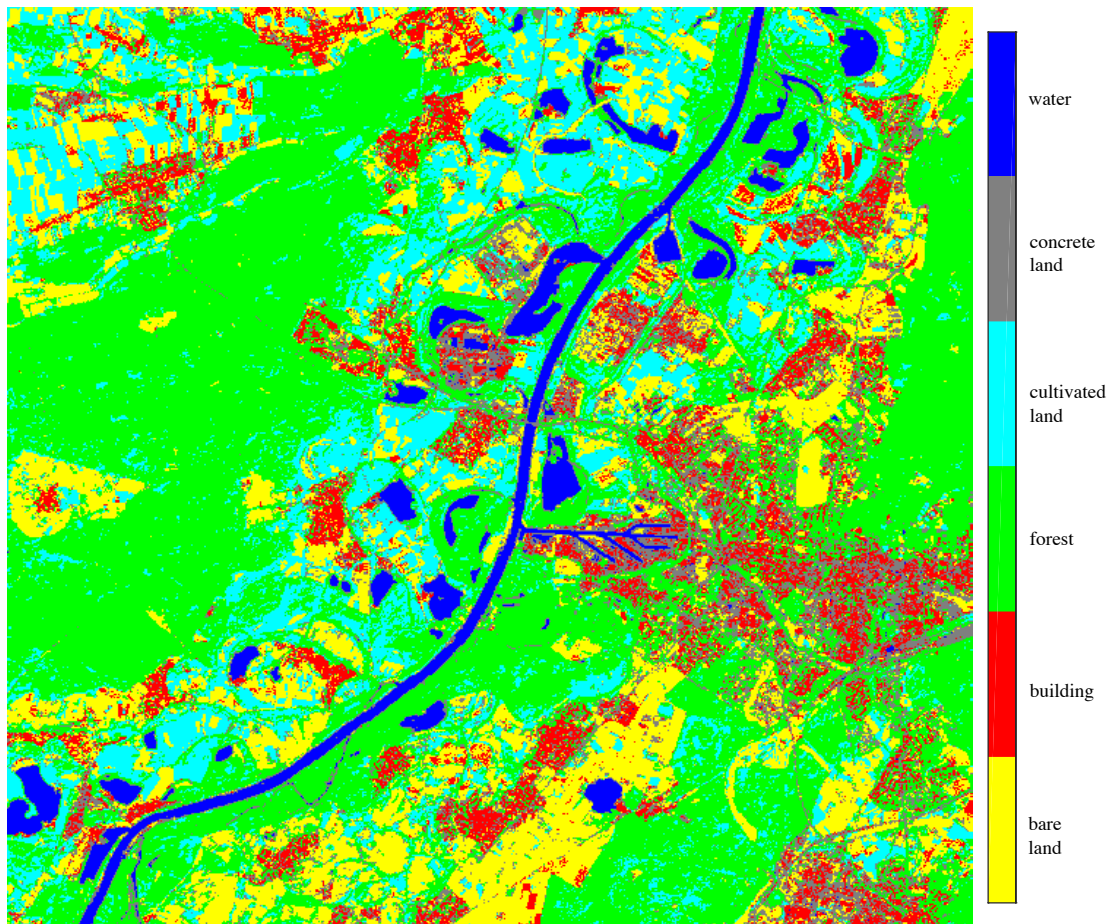


Figure 5.10: Classified study area in karlsruhe by the original ELM

By comparing the figure 5.10 to the figure 5.8 from Google Earth, it is not difficult to find that the roads cannot be distinguished when we use the neural network trained by the original ELM to classify the satellite image. Moreover, many pixels of the concrete land are classified into the class of the forest and the bare land, which is as same as indicated in the table 5.13. At the bottom right corner of the figure 5.10, it is near the city center of Karlsruhe, which should be mainly composed of the concrete land and buildings. But many pixels in this area are classified into the forest by the neural network trained by the original ELM, which is irrational and does not conform to the facts.

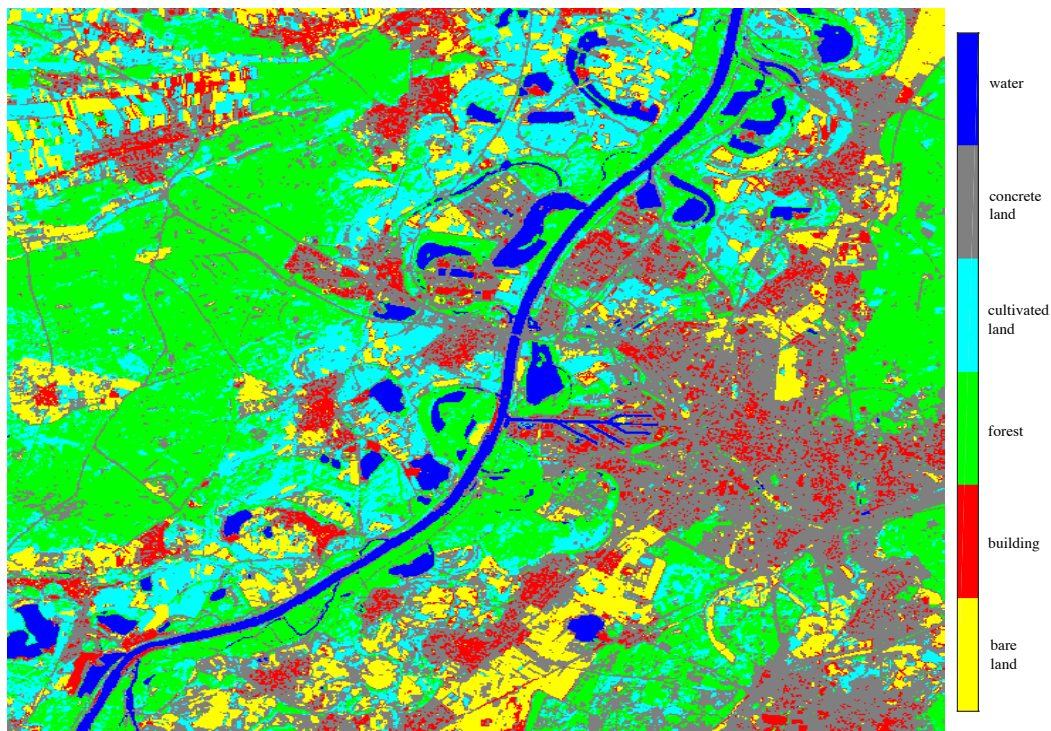


Figure 5.11: Classified study area in Karlsruhe by the regularized ELM

In the figure 5.11, almost all roads are recognized, except some narrow roads in the forest and fields. Since the satellite image to be classified is of low resolution, a pixel implies an area of 30×30 square meters. If a road is not wide enough, then the corresponding pixels may not be classified as roads. That is the reason why there are many discontinuous gray lines in the figure 5.11. For the urban area of Karlsruhe, the classifier trained by the ELM with A-optimal design regularization performs obviously much better, which can be ascertained by comparing the figure 5.11 to the satellite image of high resolution from Google Earth.

5.4 Conclusions and Future Works

Through the above three case studies, respectively, approximation of the sine function, regression analysis of real-world problems and the multi-class classification for satellite images, the advantages of the ELM with A-optimal design regularization manifest themselves.

After applying the A-optimal design regularization to the original ELM, the trained neural network holds strong anti-noise ability. The performance of the proposed A-optimal regularized ELM is not significantly affected by the outliers in the training set. From the first case study, namely the approximation of the sine function, it is obvious that the proposed A-optimal regularized ELM works despite the outliers.

Compared to the original ELM, our proposed A-optimal regularized ELM not only keeps the advantage of extremely fast training speed but also has the improved generalization ability. With the A-optimal design regularization, a proper or even optimal regularization parameter can be determined after a small amount of iterations rather than hundreds or even more of traverses, which are proposed by Deng et al. (2009) and discussed in the chapter 3.

Furthermore, the A-optimal design regularization stabilizes the estimated solution of the output weight in the neural network, thus improves the performance of the trained neural network. In the case study of multi-class classification for satellite images, the accuracy of the classification is significantly higher using our proposed A-optimal regularized ELM. Besides, the training speed is not seriously affected after introducing the A-optimal design regularization. In both case studies involving classification, the training time of the proposed A-optimal regularized ELM is about 0.5 second. Compared to the training time of the original ELM, which is approximately 0.1 second, the A-optimal design regularization impedes the training speed seldom. A research team from Tongji University implements the classification for a study area in Karlsruhe with another algorithm—the kernel ELM proposed by Huang et al. (2012). Using the kernel ELM, they classify the satellite image also with high accuracy and strong positive correlation, where the relative accuracy and the Cohens kappa coefficient are 96.91% and 0.9614 respectively. However, the computational effort is so heavy that it took almost a whole day to train the neural network. It goes against the intention—fast training speed—of using ELM.

In summary, the application of the A-optimal ELM in the ELM not only keeps the advantage of extremely fast training speed of the ELM, but also determines an optimal regularization for the regularized ELM, which is conducive to improve the performance of the trained neural network.

In the future, we plan to adopt the A-optimal regularization in other related fields in machine learning, for example, the least squares support vector machine (LS-SVM). In usual cases, people just select an empirical value as the regularization parameter for avoiding over-fitting in the training procedure. By applying the A-optimal design regularization, we aim to choose the proper regularization parameter, which is pertinent to the real-world applications and conduces to generate a model with optimal performance.

Moreover, we will study the prospect of the A-optimal design regularization in deep learning. Nowadays, the most common method for regularization in deep learning is dropout. However, dropout is slow to train (2-3 times slower than training without dropout). In addition, when there is an extremely large amount of data, dropout does not help much. Hence, we attend to apply the A-optimal design regularization as L2-regularization in deep learning.

Bibliography

- Amari, S. (1996), 'A new learning algorithm for blind signal separation', in 'Proc. 1995 Conference NIPS', pp. 757–763.
- Cai, J. (2004), 'Statistical inference of the eigenspace components of a symmetric random deformation tensor', Deutsche Geodätische Kommission (DGK) Reihe C Heft, Nr. 577, 138 pages, Munich.
- Cai, J., Grafarend, E. W. and Schaffrin, B. (2004), 'The α -optimal regularization parameter in uniform tykhonov-phillips regularization α -weighted ble', IAG Symposia 127 "V Hotline-Marussi Symposium on Mathematical Geodesy" edited by F. Sanco, Springer, pp. 309–324.
- Deng, W., Zheng, Q. and Chen, L. (2009), 'Regularized extreme learning machine', IEEE Symposium on Computational Intelligence & Data Mining pp. 389–395.
- Fletcher, R. (1980), 'Practical methods of optimization', Vol. 2 Constrained Optimization, J. Wiley.
- Gurney, K. N. (1997), 'An introduction to neural networks.', Morgan Kaufmann.
- Hansen, P. C. (1998), 'Rank-deficient and discrete ill-posed problems', Society for Industrial and Applied Mathematics, SIAM.
- Haykin, S. S. (2009), 'Neural networks and learning machines', third edn, Pearson Education.
- Hierons, R. (2015), 'Machine learning', Software Testing Verification & Reliability 9(3), 191–193.
- Huang, G. B. and Siew, C. K. (2006), 'Real-time learning capability of neural networks', IEEE Transactions on Neural Networks 17(4), 863–78.
- Huang, G. B., Zhou, H., Ding, X. and Zhang, R. (2012), 'Extreme learning machine for regression and multiclass classification', IEEE Trans Syst Man Cybern B Cybern 42(2), 513–529.
- Huang, G.-B., Zhu, Q.-Y. and Siew, C.-K. (2006), 'Extreme learning machine: Theory and applications', Neurocomputing 70(1), 489 – 501. Neural Networks.
URL: <http://www.sciencedirect.com/science/article/pii/S0925231206000385>

- Kauth, R. J. and Thomas, G. (1976), 'The tasselled cap – a graphic description of the spectral-temporal development of agricultural crops as seen by landsat', *Machine Processing of Remotely Sensed Data 1*.
- Mitchell, T. M. (1997), 'Machine Learning', McGraw-Hill, New York.
- Phillips, D. L. (1962), "A technique for the numerical solution of certain integral equations of the first kind", *Journal of the Acm* 9(1), 84–97.
- Rao, C. R. and Mitra, S. K. (1972), 'Generalized inverse of a matrix and its applications', in 'Proceedings of the Sixth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Theory of Statistics', University of California Press, Berkeley, Calif., pp. 601–620.
URL: <https://projecteuclid.org/euclid.bsmsp/1200514113>
- Rosenblatt, F. (1988), 'The perceptron: A probabilistic model for information storage and organization in the brain.', MIT Press.
- Rumelhart, D. E., Hinton, G. E. and Williams, R. J. (1988), 'Learning representations by back-propagating errors.', MIT Press.
- Tamura, S. and Tateishi, M. (1997), 'Capabilities of a four-layered feedforward neural network: four layers versus three', *IEEE Trans Neural Netw* 8(2), 251–255.
- Tikhonov, A. N. (1963), 'On the solution of ill-posed problems and the method of regularization', in 'Dokl. Akad. Nauk SSSR', pp. 501–504.
- Zell, A., Mache, N., Hüttel, M. and Vogt, M. (1993), 'Simulation neuronaler netze auf massiv parallelen rechnern', 1998, 495–502.

Appendix A

Proof of the equation (4.13), refers to Cai (2004), pages 65

Theorem 3 (Cayley matrix inverse differentiation).

$$d(A^T \Sigma_y^{-1} A + \alpha S^{-1})^{-1} = -(A^T \Sigma_y^{-1} A + \alpha S^{-1})^{-1} S^{-1} (A^T \Sigma_y^{-1} A + \alpha S^{-1})^{-1} d\alpha \quad (\text{A.1})$$

Proof. Assume that M is a matrix with rank n . There is no doubt that

$$MM^{-1} = I_n \quad (\text{A.2})$$

Differentiate both sides of the equation (5.16)

$$\begin{aligned} dMM^{-1} &= dI_n \Rightarrow \\ (dM)M^{-1} + M(dM^{-1}) &= 0 \Rightarrow \\ dM^{-1} &= -M^{-1}dM \end{aligned} \quad (\text{A.3})$$

For $M = (A^T \Sigma_y^{-1} A + \alpha S^{-1})^{-1}$, we can get the result of type (5.15), namely Cayley matrix inverse differentiation.

Theorem 4 (Differentiation of trace of a matrix). Assume that A, B, X are matrix with rank n .

$$\text{trace}(A + B) = \text{trace}(A) + \text{trace}(B) \quad (\text{A.4})$$

$$d(\text{trace}(A + B)) = \text{trace}(dA) + \text{trace}(dB) \quad (\text{A.5})$$

$$d(\text{trace}(XAX^T)) = \text{trace}(A + A^T)X^T dX \quad (\text{A.6})$$

Theorem 5 (Cayley inverse: sum of two matrices).

$$\begin{aligned} &(A^T \Sigma_y^{-1} A + \alpha S^{-1})^{-1} A^T \Sigma_y^{-1} A \\ &= [I_n + \alpha (A^T \Sigma_y^{-1} A)^{-1} S^{-1}]^{-1} \\ &= I_n - \alpha (A^T \Sigma_y^{-1} A + \alpha S^{-1})^{-1} S^{-1} \\ &= I_n - \alpha (SA^T \Sigma_y^{-1} A + \alpha I_n)^{-1} \end{aligned} \quad (\text{A.7})$$

Theorem 6.

$$\text{trace}(\beta\beta') = \beta'\beta \quad (\text{A.8})$$

$$d(\beta'\beta) = (d\beta')\beta + \beta'(d\beta) = 2\beta'(d\beta) \quad (\text{A.9})$$

For $\beta = -\alpha(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1}S^{-1}x$, we can use the lemmas above to compute that

$$\begin{aligned} d\beta &= -d\alpha(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1}S^{-1}x - \alpha d[(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1}S^{-1}]S^{-1}x \\ &= -d\alpha(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1}S^{-1}x + \alpha(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1}S^{-1}(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1}d\alpha S^{-1}x \\ &= -(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1}[I_n - \alpha S^{-1}(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1}]S^{-1}d\alpha x \quad (\text{A.10}) \\ &= -(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1}A^T\Sigma_y^{-1}A(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1}S^{-1}d\alpha x \\ &= -(SA^T\Sigma_y^{-1}A + \alpha I_n)^{-1}SA^T\Sigma_y^{-1}A(SA^T\Sigma_y^{-1}A + \alpha I_n)^{-1}d\alpha x \end{aligned}$$

$$\begin{aligned} d(\beta'\beta) &= 2\beta'(d\beta) \\ &= 2[-\alpha(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1}S^{-1}x]^T \cdot \\ &[-(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1}A^T\Sigma_y^{-1}A(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1}S^{-1}d\alpha x] \quad (\text{A.11}) \\ &= 2\alpha x'S^{-1}(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-2}A^T\Sigma_y^{-1}A(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1}S^{-1}xd\alpha \\ &= 2\alpha x'(SA^T\Sigma_y^{-1}A + \alpha I_n)^{-2}SA^T\Sigma_y^{-1}ASA^T\Sigma_y^{-1}A + \alpha I_n)^{-1}xd\alpha \end{aligned}$$

With such results, we can prove the correctness of A-optimal design

Proof. Take the differentiation of the trace of the mean square error matrix $MSE_{\alpha,S}(\hat{x})$ of α -homBLE

$$d \text{trace}(MSE_{\alpha,S}(\hat{x})) = \text{trace}(d[(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1}A^T\Sigma_y^{-1}A(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1}S^{-1}]) + d\beta'\beta \quad (\text{A.12})$$

"the first term"

$$\begin{aligned} &\text{trace}(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1}A^T\Sigma_y^{-1}A(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1}S^{-1} \\ &= \text{trace}(2A^T\Sigma_y^{-1}A(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1}d(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1}) \quad (\text{A.13}) \\ &= -2\text{trace}(A^T\Sigma_y^{-1}A(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-2}S^{-1}(A^T\Sigma_y^{-1}A + \alpha S^{-1})^{-1})d\alpha \end{aligned}$$

"the second term"

$$d(\beta'\beta) = 2\alpha x'(SA^T\Sigma_y^{-1}A + \alpha I_n)^{-2}SA^T\Sigma_y^{-1}ASA^T\Sigma_y^{-1}A + \alpha I_n)^{-1}xd\alpha \quad (\text{A.14})$$

"the derivative of $MSE_{\alpha,S}(\hat{x})$ to the weight factor α "

$$\begin{aligned}
& \frac{d}{d\alpha} \text{trace}(MSE_{\alpha,S}(\hat{x})) \\
&= -2 \text{trace}(A^T \Sigma_y^{-1} A (A^T \Sigma_y^{-1} A + \alpha S^{-1})^{-2} S^{-1} (A^T \Sigma_y^{-1} A + \alpha S^{-1})^{-1}) \\
&+ 2\alpha x' (S A^T \Sigma_y^{-1} A + \alpha I_n)^{-2} S A^T \Sigma_y^{-1} A S A^T \Sigma_y^{-1} A + \alpha I_n)^{-1} x
\end{aligned} \tag{A.15}$$

$$\frac{d}{d\alpha} \text{trace}(MSE_{\alpha,S}(\hat{x})) = 0 \Rightarrow$$

$$\hat{\alpha} = \frac{\text{trace}(A^T \Sigma_y^{-1} A (A^T \Sigma_y^{-1} A + \alpha S^{-1})^{-2} S^{-1} (A^T \Sigma_y^{-1} A + \alpha S^{-1})^{-1})}{x' S^{-1} (A^T \Sigma_y^{-1} A + \alpha S^{-1})^{-2} A^T \Sigma_y^{-1} A (A^T \Sigma_y^{-1} A + \alpha S^{-1})^{-1} S^{-1} x}$$