Optimization Approximation Solution for Regression Problem Based on Extremal Learning Machine

Yubo Yuan Yuguang Wang Feilong Cao*

Department of Mathematics, China Jiliang University,
Hangzhou 310018, Zhejiang Province, P R China

Abstract

Extreme learning machine (*ELM*) is one of the most popular and important learning algorithms. It comes from single-hidden layer feed-forward neural networks. It has been proved that *ELM* can achieve better performance than support vector machine(*SVM*) in regression and classification. In this paper, mathematically, with regression problem, the step 3 of *ELM* is studied. First of all, the equation $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$ are reformulated as an optimal model. With the optimality, the necessary conditions of optimal solution are presented. The equation $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$ is replaced by $\mathbf{H}^T\mathbf{H}\boldsymbol{\beta} = \mathbf{H}^T\mathbf{T}$. We can prove that the latter must have one solution at least. Secondly, optimal approximation solution is discussed in cases of $\mathbf{H}$ is column full rank, row full rank, neither column nor row full rank. In the last case, the rank-1 and rank-2 methods are used to get optimal approximation solution. In theory, this paper present a better algorithm for *ELM*.

Extreme learning machine, Regression, Optimization, Matrix Theory.

*Corresponding author: Feilong Cao, E-mail: feilongcao@gmail.com

1 Advanced Extreme Learning Machine

In 2004, Huang, et al[1] have shown that the learning speed of feedforward neural networks is in general far slower than required and it has been a major bottleneck in their applications for past decades. Two key reasons behind may be: 1) the slow gradient-based learning algorithms are extensively used to train neural networks, and 2) all the parameters of the networks are tuned iteratively by using such learning algorithms. They proposed a new learning algorithm called *extreme learning machine (ELM)* for single-hidden layer feedforward neural networks (SLFNs) which randomly chooses the input weights and analytically determines the output weights of SLFNs. In theory, the proposed algorithm has a extremely fast learning speed. They have done some experimental based on real-world benchmarking function approximation and classification problems. The results have shown that *ELM* could produce better generalization performance in some cases and learn much faster than traditional popular learning algorithms for feedforward neural networks. Moreover, they [2] showed that *ELM* could be extended to radial basis function (RBF) network case, which allows the centers and impact widths of RBF kernels to be randomly generated and the output weights to be simply analytically calculated instead of iteratively tuned.

In 2005, Huang , et al[3] extended the *ELM* algorithm from the real domain to the complex domain, and then applied the fully *complex extreme learning machine (C-ELM)* for nonlinear channel equalization applications. The simulation results showed that the *ELM* equalizer significantly outperforms other neural network equalizers such as the complex minimal resource allocation network (CMRAN), complex radial basis function (CRBF) network and complex backpropagation (CBP) equalizers. At the same time, they [4] indicated that *ELM* might need higher number of hidden neurons due to the random determination of the hidden nodes parameters and hidden biases. They proposed a hybrid learning algorithm to select the input

weights and Moore-Penrose (MP) generalized inverse to analytically determine the output weights. Experimental results showed that it was able to achieve good generalization performance with much more compact networks.

ELM [5] has been tested on a few artificial and real benchmark function approximation and classification problems including very large complex applications. In 2006, Huang et al. [6] have proposed a new theory to show that single-hidden-layer feedforward networks (SLFNs) with randomly generated additive or radial basis function (RBF) hidden nodes (according to any continuous sampling distribution) can work as universal approximators and the resulting *incremental extreme learning machine (I-ELM)* outperforms many popular learning algorithms. *I-ELM* randomly generates the hidden nodes and analytically calculates the output weights of SLFNs, however, *I-ELM* does not recalculate the output weights of all the existing nodes when a new node is added. In 2007, Huang et al. [7] showed that while retaining the same simplicity, the convergence rate of *I-ELM* could be further improved by recalculating the output weights of the existing nodes based on a convex optimization method when a new hidden node is randomly added. Furthermore, they showed that given a type of piecewise continuous computational hidden nodes (possibly not neural alike nodes), if SLFNs

$$\hat{f}_L(\mathbf{x}) = \sum_{k=1}^{L} \boldsymbol{\beta}_k G(\mathbf{w}_k \cdot \mathbf{x} + b_k)$$

could work as universal approximators with adjustable hidden node parameters, from a function approximation point of view the hidden node parameters of such *generalized* SLFNs (including *sigmoid networks, RBF networks, trigonometric networks, threshold networks, fuzzy inference systems, fully complex neural networks, high-order networks, ridge polynomial networks, wavelet networks, etc.*) could be randomly generated according to any continuous sampling distribution. Zhang et al. [8] used *ELM* for directing multicategory classification problems in the cancer diagnosis area. They evaluated the multicategory classification performance of ELM on three benchmark mi-

croarray data sets for cancer diagnosis, namely, the GCM data set, the Lung data set, and the Lymphoma data set. The results indicated that *ELM* produced comparable or better classification accuracies with reduced training time and implementation complexity compared to artificial neural networks methods like conventional back-propagation ANN , Linder's SANN, and Support Vector Machine methods like SVM-OVO and Ramaswamy's SVM-OVA [8].

Huang et al. [9] extended *I-ELM* from the real domain to the complex domain. They showed that *I-ELM* could approximate any target functions in the complex domain. In the same year [10], they proposed an enhanced method for *I-ELM* (referred to as *EI-ELM*). At each learning step, several hidden nodes were randomly generated and among them the hidden node leading to the largest residual error decreasing would be added to the existing network and the output weight of the network would be calculated in a same simple way as in the original *I-ELM*. In the same year, Lan at al. [11] introduced *ELM* to predict the subcellular localization of proteins based on the frequent subsequences. They showed that *ELM* was extremely fast and could provide good generalization performance.

In 2009, Feng et al. [12] proposed a simple and efficient approach to automatically determine the number of hidden nodes in generalized single-hidden-layer feedforward networks (SLFNs) which need not be neural alike. They named it as *error minimized extreme learning machine (EM-ELM)*. It could add random hidden nodes to SLFNs one by one or group by group (with varying group size). Simulation results demonstrated and verified that *EM-ELM* was much faster than other *sequential/incremental/growing* algorithms with good generalization performance. Rong et al [13] proposed *an online sequential fuzzy extreme learning machine (OS-fuzzy-ELM)* for function approximation and classification problems. The learning in *OS-fuzzy-ELM* could be done with the input data coming in the one-by-one mode or chunk-by-chunk (a block of data) mode with fixed or varying chunk size. In

OS-fuzzy-ELM, all the antecedent parameters of membership functions are randomly assigned first, and then, the corresponding consequent parameters are determined analytically. Performance comparisons of *OS-fuzzy-ELM* with other existing algorithms were made on real-world benchmark problems in the areas of nonlinear system identification, regression, and classification. The results showed that the proposed *OS-fuzzy-ELM* produced similar or better accuracies with at least an order-of-magnitude reduction in the training time.

In 2010, Cao et al. [14] introduced a new structure of wavelet neural networks (WNN) with *extreme learning machine (ELM)*. They used the composite functions at the hidden nodes and the learning was done using *ELM*. Experimental results on the regression of some nonlinear functions and real-world data, the prediction of a chaotic signal and classifications on serval benchmark real-world data sets showed that the proposed neural networks could achieve better performances in most cases than some relevant neural networks and learn much faster than neural networks training with the traditional back-propagation (BP) algorithm. At the same time, Huang et al. [15] studied *ELM* for classification in the aspect of the standard optimization method and extended *ELM* to a specific type of generalized SLFNs–support vector network. They said that: (1) under the *ELM* learning framework, SVM's maximal margin property and the minimal norm of weights theory of feedforward neural networks are actually consistent; (2) from the standard optimization method point of view ELM for classification and SVM are equivalent but ELM has less optimization constraints due to its special separability feature; (3) as analyzed in theory and further verified by the simulation results, *ELM* for classification tends to achieve better generalization performance than traditional SVM. *ELM* for classification is less sensitive to user specified parameters and can be implemented easily.

Actually, experiment results show that *Extremal Learning Machine* takes some advantages of training time. Originally, the solution from *ELM* algo-

rithm is got by solving the moore-penrose inverse of hidden layer matrix $\mathbf{H}$. In many cases, there is no any solution for the equation $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$. The baseline is to look for optimal solution. Therefore, some theoretical works should be done to support this powerful algorithm. This is the motivation of this paper.

In this paper, mathematically, we study the most important step of *ELM* and will present some useful theory to support this algorithm. The proposed implementation will make sure that the *ELM* will keep good effectiveness.

In next section, we will briefly introduce the regression problem and the single hidden layer neural network, at the same time, will list the basic *ELM* algorithm. In Section 3, the optimal solution is discussed. In Section 4, we will give a more powerful *ELM* algorithm. Discussions and conclusions will be presented in Section 5.

2 Brief Review of Regression Problem and Single Hidden Layer Feedforward Networks

Regression problem is one of the most fundamental problems of machine learning. This problem can be formalized as follows: We are given empirical data:

$$\{(\mathbf{x}_1, t_1), \cdots, (\mathbf{x}_N, t_N)\} \tag{1}$$

Here $(\mathbf{x}_i, t_i) \in \mathcal{X} \times \mathcal{T}$, $\mathcal{X}$ is some nonempty set from which the instances(inputs) are taken, usually referred to as the samples space. Mathematically, $\mathbf{x}_i$ is always referred as an independent point, in this case, $\mathcal{X} \subset \mathbb{R}^n$ is named as the *domain* and $\mathbf{x}_i = (x_{i1}, x_{i2}, \cdots, x_{in})^T$. $t_i \in \mathcal{T} \subset \mathbb{R}$ is the output with regard to the input $\mathbf{x}_i$.

The regression basis functions set are denoted as

$$H = \{h_1, h_2, \cdots, h_m\},$$

where $h_i : \mathbb{R}^n \to \mathbb{R}, i = 1, \cdots, m$. The regression function is taken the form as a linear combination of the basis functions

$$\hat{f}(\mathbf{x}) = \sum_{j=1}^{m} \beta_j h_j(\mathbf{x}) = (h_1(\mathbf{x}), h_2(\mathbf{x}), \cdots, h_m(\mathbf{x}))(\beta_1, \beta_2, \cdots, \beta_m)^T. \tag{2}$$

By calculating the values of $\hat{f}$ in the data points $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N$, we expect to determine the coefficients β_j.

If there is no error, the regression problem is equivalent to solving the following equations:

$$\begin{pmatrix} h_1(\mathbf{x}_1) & h_2(\mathbf{x}_1) & \cdots & h_m(\mathbf{x}_1) \\ h_1(\mathbf{x}_2) & h_2(\mathbf{x}_2) & \cdots & h_m(\mathbf{x}_2) \\ \cdots & \cdots & \cdots & \cdots \\ h_1(\mathbf{x}_N) & h_2(\mathbf{x}_N) & \cdots & h_m(\mathbf{x}_N) \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \\ \cdots \\ \beta_N \end{pmatrix} = \begin{pmatrix} t_1 \\ t_2 \\ \cdots \\ t_N \end{pmatrix} \tag{3}$$

It can de briefly denoted as follows

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{T}. \tag{4}$$

Where $\mathbf{H}_{ij} = h_j(\mathbf{x}_i), \boldsymbol{\beta} = (\beta_1, \beta_2, \cdots, \beta_m)^T, \mathbf{T} = (t_1, t_2, \cdots, t_N)^T$.

The solution of (4) is impacted by the number of samples points N and basis functions m.

Example 1. The given data set is $\{(0,0),(1,1),(2,4)\}$,$E = \{1, x\}$, the equations are as follows:

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 4 \end{pmatrix}. \tag{5}$$

There is no any solution because that they are over-determined or over-constrained. In another word, if the regression function is selected as $\hat{f}(x) = \beta_1 + \beta_2 x$, no any line can go through the given three points($\{(0,0),(1,1),(2,4)\}$), this can be seen in the illustration.

In this case, we need to get a *good* solution to minimize the error

$$\min_{\boldsymbol{\beta}} \|\mathbf{A}\boldsymbol{\beta} - \mathbf{Y}\|_2. \tag{6}$$

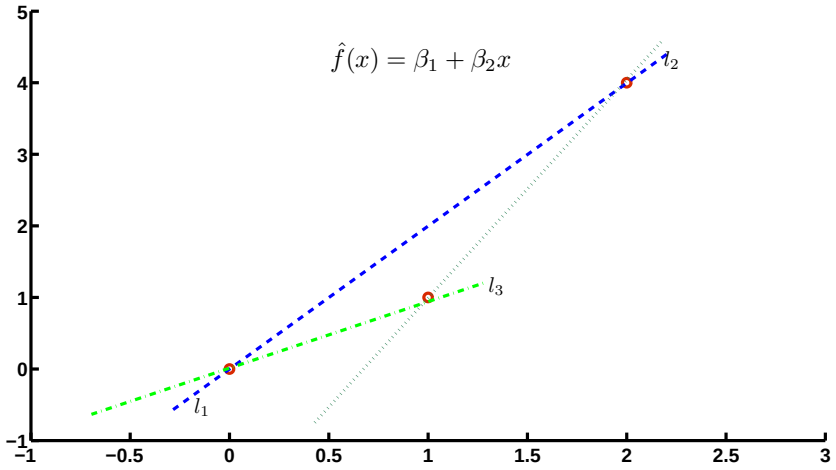

Fig. 1. Example 1

After solving the optimization model (6), we can obtain the *good* solution and get the regression function.

From the view of neural networks, the regression problem has the same solution method with the single-hidden-layer feedforward neural networks (SLFNs)[16, 17, 18, 19].

If we let $\mathbf{w}_i$ denote the weight vector connecting the input layer to the $i-th$ hidden node, b_i is the threshold of the $i-th$ hidden node. β_i is the weight connecting the $i-th$ hidden node to the output node, and g is the hidden node activation function. $\mathbf{w}_i x$ denotes the inner product of vectors $\mathbf{w}$ and x in $\mathbb{R}^n$, the single-hidden-layer feedforward neural networks for regression problem is

$$\hat{f}(\mathbf{x}) = \sum_{k=1}^{L} \beta_k G(\mathbf{w}_k \cdot \mathbf{x} + b_k). \tag{7}$$

Where L is the number of hidden nodes.

If we let

$$o_j = \hat{f}(\mathbf{x}_j), j = 1, 2, \cdots, N, \tag{8}$$

the neural networks can derive the coefficients $\mathbf{w}_i, b_i, \beta_i, i = 1, 2, \cdots, L$, by learning method. Some publications had shown that SLFNs can achieve zero error with these given N samples, it means that

$$\sum_{k=1}^{N} |o_k - t_k| = 0. \tag{9}$$

In another word, there are $\mathbf{w}_i^*, b_i^*, \boldsymbol{\beta}_i^*, i = 1, 2, \cdots, L$, such that

$$\sum_{j=1}^{N} |\sum_{k=1}^{L} \boldsymbol{\beta}^*{}_k G(\mathbf{w}^*{}_k \cdot \mathbf{x}_j + b_k^*) - t_j| = 0. \tag{10}$$

As developed by Huang et al. [5, 6, 7, 10], let

$$\mathbf{H} = \begin{pmatrix} G(\mathbf{w}_1 \cdot \mathbf{x}_1 + b_1) & G(\mathbf{w}_2 \cdot \mathbf{x}_1 + b_2) & \cdots & G(\mathbf{w}_L \cdot \mathbf{x}_1 + b_L) \\ G(\mathbf{w}_1 \cdot \mathbf{x}_2 + b_1) & G(\mathbf{w}_2 \cdot \mathbf{x}_2 + b_2) & \cdots & G(\mathbf{w}_L \cdot \mathbf{x}_2 + b_L) \\ \cdots & \cdots & \cdots & \cdots \\ G(\mathbf{w}_1 \cdot \mathbf{x}_N + b_1) & G(\mathbf{w}_2 \cdot \mathbf{x}_N + b_2) & \cdots & G(\mathbf{w}_L \cdot \mathbf{x}_N + b_L) \end{pmatrix}. \tag{11}$$

The equation (10) is equivalent to the following equations

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{T}. \tag{12}$$

Let us take a look at the equations (4) and (12), we can find that there is no any difference except for $\mathbf{A}$ and $\mathbf{H}$. So, we can say that the solution methods of the single-hidden-layer feedforward neural networks can be used to regression problem.

The *ELM* algorithm is summarized as follows:

Algorithm ELM*: Given a training set*

$$\mathcal{S} = \{(\mathbf{x}_1, t_1), \cdots, (\mathbf{x}_N, t_N)\},$$

active function $G(x)$, and hidden node number L;

Step 1: Randomly assign input weight $\mathbf{w}_i$ *and* b_i, $i = 1, 2, \cdots, L$;

Step 2: Calculate the hidden layer output matrix $\mathbf{H}$;

Step 3: Calculate the output weight $\boldsymbol{\beta} = \mathbf{H}^{\dagger}\mathbf{T}$.

Among of them, the matrix $\mathbf{H}^{\dagger}$ is the *Moore-Penrose generalized inverse* of matrix $\mathbf{H}$.

Actually, it is a very simple algorithm and take less time to training the data set. From this view, it is a *extremal* learning machine.

3 Optimal Approximation Solution for *ELM*

In this section, we will discuss some important issues in order to make sure that the step 3 in *ELM* will get the optimal solution.

The primal idea of the step 3 is equivalent to solve the optimal problem.

$$\min_{\boldsymbol{\beta}} \|\mathbf{H}\boldsymbol{\beta} - \mathbf{Y}\|_2^2. \tag{13}$$

Where $\mathbf{H} \in \mathbb{R}^{N \times L}$ is given by(11), $\mathbf{Y} \in \mathbb{R}^N$ is given. Let us pay attention to the notations N and L, there are the numbers of samples(instances or data points) and hidden nodes.

$$\min_{\boldsymbol{\beta}} (\mathbf{H}\boldsymbol{\beta} - \mathbf{Y})^T(\mathbf{H}\boldsymbol{\beta} - \mathbf{Y}) = \min_{\boldsymbol{\beta}} \boldsymbol{\beta}^T\mathbf{H}^T\mathbf{H}\boldsymbol{\beta} - 2\mathbf{Y}^T\mathbf{H}\boldsymbol{\beta} - \mathbf{Y}^T\mathbf{Y}. \tag{14}$$

It is a normal quadratical programming model without any constrains. The objective function can be denoted as

$$Q(\boldsymbol{\beta}) = \boldsymbol{\beta}^T\mathbf{H}^T\mathbf{H}\boldsymbol{\beta} - 2\mathbf{T}^T\mathbf{H}\boldsymbol{\beta} - \mathbf{Y}^T\mathbf{Y}. \tag{15}$$

If $\boldsymbol{\beta}^*$ is the optimal solution of (14), then

$$\boldsymbol{\beta}^* = arg\{\min_{\boldsymbol{\beta}} Q(\boldsymbol{\beta})\}. \tag{16}$$

The first optimal condition is

$$C_1: \quad \nabla Q(\boldsymbol{\beta}^*) = \mathbf{0}. \tag{17}$$

The second optimal condition is

$$C_2: \quad \mathbf{H}^T\mathbf{H} \ \ is \ \ positive \ \ definite. \tag{18}$$

These two conditions are very important to verify that $\boldsymbol{\beta}^*$ is the optimal solution. Among of them, (18) is a very strong condition. If (18) is hold, we can get the optimal solution from (17). Because that $\mathbf{H}^T\mathbf{H} \in \mathbb{R}^{L\times L}$ is a symmetric matrix, the condition (17) is equivalent to

$$2\mathbf{H}^T\mathbf{H}\boldsymbol{\beta}^* - 2\mathbf{H}^T\mathbf{T} = \mathbf{0}. \tag{19}$$

It is

$$\mathbf{H}^T\mathbf{H}\boldsymbol{\beta}^* = \mathbf{H}^T\mathbf{T}. \tag{20}$$

If the condition (18) is hold, the optimal solution

$$\boldsymbol{\beta}^* = \mathbf{H}^T\mathbf{H}^{-1}\mathbf{c} = \mathbf{H}^T\mathbf{T} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{T}. \tag{21}$$

3.1 H is column or row full rank matrix

In general, the number of hidden nodes is less than the number of samples(given data points or instances), means that $L \leq N$. In fact, it is a very challenge topic that how many hidden nodes should be given to make sure that the neural network can work well.

First of all, let us assume that $L \leq N$. In this case, in order to discuss the optimal solution of (13), we need to prove the following results:

Theorem 3.1. *Let* $\mathbf{H} \in \mathbb{R}^{N\times L}$ *and* $rank(\mathbf{H}) = r$*, then we have*

$$rank(\mathbf{H}^T\mathbf{H}) = rank(\mathbf{H}\mathbf{H}^T) = r, \tag{22}$$

moreover, there must be one solution $\boldsymbol{\beta}$ *for equations*

$$\mathbf{H}^T\mathbf{H}\boldsymbol{\beta} = \mathbf{H}^T\mathbf{T}. \tag{23}$$

Proof. It is easy to verify that $rank(\mathbf{H}^T\mathbf{H}) = rank(\mathbf{H}\mathbf{H}^T)$ because that $rank(\mathbf{A}) = rank(\mathbf{A}^T)$ is always hold for any matrix $\mathbf{A} \in \mathbb{R}^{n\times m}$. We need to prove that $rank(\mathbf{H}^T\mathbf{H}) = r$.

i):It is obvious that $rank(\mathbf{H}^T\mathbf{H}) \leq r$ because that

$$rank(\mathbf{H}^T\mathbf{H}) \leq min\{rank(H^T), rank(H)\} = r; \tag{24}$$

ii): If $rank(\mathbf{H}^T\mathbf{H}) = s$ and $rank(\mathbf{H}) = r$, the spaces

$$\mathcal{S}_1 = \{\boldsymbol{\beta}|\mathbf{H}^T\mathbf{H}\boldsymbol{\beta} = \mathbf{0}, \boldsymbol{\beta} \in \mathbb{R}^L\}, \mathcal{S}_2 = \{\boldsymbol{\beta}|\mathbf{H}\boldsymbol{\beta} = \mathbf{0}, \boldsymbol{\beta} \in \mathbb{R}^L\}. \tag{25}$$

The dimensions of $\mathcal{S}_1$ and $\mathcal{S}_2$ are $L - s$ and $L - r$.

For any $\boldsymbol{\beta} \in \mathcal{S}_1$, we have

$$\|\mathbf{H}\boldsymbol{\beta}\|_2^2 = (\mathbf{H}\boldsymbol{\beta})^T(\mathbf{H}\boldsymbol{\beta}) = \boldsymbol{\beta}^T\mathbf{H}^T\mathbf{H}\boldsymbol{\beta} = 0.$$

Then, we have

$$\mathbf{H}\boldsymbol{\beta} = 0.$$

So, we have $\boldsymbol{\beta} \in \mathcal{S}_2$. It indicates that $L - s \leq L - r$. It shows that

$$s = rank(\mathbf{H}^T\mathbf{H}) \geq r = rank(\mathbf{H}). \tag{26}$$

From (24) and (26), we can get the result that $rank(\mathbf{H}^T\mathbf{H}) = rank(\mathbf{H})$.

Because that $(\mathbf{H}^T\mathbf{H}, \mathbf{H}^T\mathbf{T}) = \mathbf{H}^T(\mathbf{H}, \mathbf{T})$, we have

$$rank(\mathbf{H}^T\mathbf{H}, \mathbf{H}^T\mathbf{T}) = rank(\mathbf{H}^T(\mathbf{H}, \mathbf{T})) \leq rank(\mathbf{H}^T) = rank(\mathbf{H}), \tag{27}$$

with the upper result $rank(\mathbf{H}^T\mathbf{H}) = rank(\mathbf{H})$, we have

$$rank(\mathbf{H}^T\mathbf{H}, \mathbf{H}^T\mathbf{T}) \leq rank(\mathbf{H}^T\mathbf{H}). \tag{28}$$

Let us pay attention to the fact

$$rank(\mathbf{H}^T\mathbf{H}, \mathbf{H}^T\mathbf{T}) \geq rank(\mathbf{H}^T\mathbf{H}), \tag{29}$$

we have

$$rank(\mathbf{H}^T\mathbf{H}, \mathbf{H}^T\mathbf{T}) = rank(\mathbf{H}^T\mathbf{H}). \tag{30}$$

So, according to the solution theorem of linear equations, there must be one solution for equations (23). The proof is ended. □

Theorem 3.2. *Let $\mathbf{H} \in \mathbb{R}^{N\times L}$ and $rank(\mathbf{H}) = L$, the optimization problem (13) has unique optimal solution who can be checked out by*

$$\boldsymbol{\beta} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{T}. \tag{31}$$

Proof. If $\mathbf{H} \in \mathbb{R}^{N\times L}$ and $rank(\mathbf{H}) = L$, according to Theorem 3.1, we have $rank(\mathbf{H}^T\mathbf{H}) = L$ and $det(\mathbf{H}^T\mathbf{H}) \neq 0$, the matrix $\mathbf{H}^T\mathbf{H} = \mathbf{H}^T\mathbf{H}$ is invertible. Furthermore, the quadratical programming (14) is non-concave, the extremal value of the objective $Q(\boldsymbol{\beta})$ is zero. From the first order optimal condition, we have the result $\boldsymbol{\beta} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{T}$. □

Theorem 3.1 and 3.2 show that we can get the out weights by (31) if the hidden layer matrix $\mathbf{H}$ is column full rank.

Example 2. The given data set is

$$\mathbf{X} = \begin{pmatrix} \mathbf{X}_1^T \\ \mathbf{X}_2^T \\ \mathbf{X}_3^T \\ \mathbf{X}_4^T \\ \mathbf{X}_5^T \end{pmatrix} = \begin{pmatrix} 0.8147 & 0.0975 & 0.1576 & 0.1419 & 0.6557 \\ 0.9058 & 0.2785 & 0.9706 & 0.4218 & 0.0357 \\ 0.1270 & 0.5469 & 0.9572 & 0.9157 & 0.8491 \\ 0.9134 & 0.9575 & 0.4854 & 0.7922 & 0.9340 \\ 0.6324 & 0.9649 & 0.8003 & 0.9595 & 0.6787 \end{pmatrix}, \mathbf{T} = \begin{pmatrix} 3 \\ 1 \\ 4 \\ 5 \\ 2 \end{pmatrix}. \tag{32}$$

We set $L = 3$ and randomly select $\mathbf{w}_k, b_k$ as

$$\begin{pmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \mathbf{w}_3^T \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix}, \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}. \tag{33}$$

The activation function is given as $G(x) = e^{-x^2}$. Then according to the basic procedure of the single-hidden-layer feedforward neural networks, we have

the hidden layer matrix as follows

$$\mathbf{H} = \begin{pmatrix} 0.0371 & 0.2998 & 0.2618 \\ 0.0265 & 0.1950 & 0.0206 \\ 0.2808 & 0.0914 & 0.0217 \\ 0.0257 & 0.0217 & 0.1101 \\ 0.0696 & 0.0211 & 0.0391 \end{pmatrix}. \tag{34}$$

Because $rank(\mathbf{H}) = 3 = L$, so we can get the output weight as

$$\boldsymbol{\beta} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{T} = \begin{pmatrix} 15.8698 \\ -7.3131 \\ 21.8435 \end{pmatrix}. \tag{35}$$

When the number of the samples (given data points) is less than the number of hidden nodes (means that $N \leq L$), the procedure to get the out weights is same as the upper one. The following theorem is presented to show that how to get the out weights.

Theorem 3.3. *Let $\mathbf{H} \in \mathbb{R}^{N\times L}$ and $rank(\mathbf{H}) = N, (N \leq L)$, the optimization problem (13) has unique optimal solution who can be checked out by*

$$\boldsymbol{\beta} = \mathbf{H}^T(\mathbf{H}\mathbf{H}^T)^{-1}\mathbf{T}. \tag{36}$$

Example 3. The given data set is

$$\mathbf{X} = \begin{pmatrix} \mathbf{X}_1^T \\ \mathbf{X}_2^T \\ \mathbf{X}_3^T \end{pmatrix} = \begin{pmatrix} 0.9501 & 0.4860 & 0.4565 & 0.4447 & 0.9218 \\ 0.2311 & 0.8913 & 0.0185 & 0.6154 & 0.7382 \\ 0.6068 & 0.7621 & 0.8214 & 0.7919 & 0.1763 \end{pmatrix}, \mathbf{T} = \begin{pmatrix} 4 \\ 1 \\ 3 \end{pmatrix}. \tag{37}$$

We set $L = 5$ and randomly select $\mathbf{w}_k, b_k$ as

$$\begin{pmatrix} \mathbf{w}_1^T \\ \mathbf{w}_2^T \\ \mathbf{w}_3^T \\ \mathbf{w}_4^T \\ \mathbf{w}_5^T \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}, \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}. \tag{38}$$

The activation function is given as $G(x) = e^{-x^2}$. Then according to the basic procedure of the single-hidden-layer feedforward neural networks, we have the hidden layer matrix as follows

$$\mathbf{H} = \begin{pmatrix} 0.4055 & 0.7896 & 0.8119 & 0.8206 & 0.4275 \\ 0.9480 & 0.4518 & 0.9997 & 0.6847 & 0.5799 \\ 0.6919 & 0.5595 & 0.5093 & 0.5341 & 0.9694 \end{pmatrix}. \tag{39}$$

Because $rank(\mathbf{H}) = N = 3 < L = 5$, according to the (36), we can get the output weight as

$$\boldsymbol{\beta} = \mathbf{H}^T(\mathbf{H}\mathbf{H}^T)^{-1}\mathbf{T} = \begin{pmatrix} -2.6860 \\ 4.0978 \\ -0.8854 \\ 2.1223 \\ 1.9428 \end{pmatrix}. \tag{40}$$

3.2 Rank-1 Approximation Optimal Solution

If $\mathbf{H}$ is neither column full rank nor row full rank, it is difficult to check out the optimal solution of (13). In this case, the general method is to get the optimal solution by approximation method.

First, we introduce the rank-1 method. We denote

$$\mathcal{G}^+ = \{\mathbf{G} \in \mathbb{R}^{L\times L} | \mathbf{G} \textit{ is a symmetric and positive definite matrix.}\} \tag{41}$$

Lemma 3.1. *Let* $\mathbf{M} = \mathbf{H}^T\mathbf{H}, c = \mathbf{H}^T\mathbf{T}$, *and* $rank(\mathbf{H}) = s < L$, *there is a symmetric and definite matrix* $\mathbf{B}^* \in \mathcal{G}^+$, *such that*

$$\mathbf{B}^* = arg\{\min_{\mathbf{B}\in\mathcal{G}^+} \|\mathbf{M} - \mathbf{B}\|\}, \tag{42}$$

here $\|\cdot\|$ *refers to the matrix norm.*

Lemma 3.2. *Let* $\mathbf{M} = \mathbf{H}^T\mathbf{H}, c = \mathbf{H}^T\mathbf{T}$, *and* $rank(\mathbf{H}) = s < L$, $\mathbf{B}^* \in \mathcal{G}^+$ *is determined by Lemma 3.1. There is a vector* $\boldsymbol{\beta}^* \in \mathbb{R}^L$ *such that*

$$\boldsymbol{\beta}^* = arg\{\min_{\boldsymbol{\beta}\in\mathbb{R}^L} \|\mathbf{B}^*\boldsymbol{\beta} - \mathbf{c}\|\}, \tag{43}$$

here $\|\cdot\|$ *refers to the vector norm.*

Theorem 3.4. *Let* $\mathbf{M} = \mathbf{H}^T\mathbf{H}, c = \mathbf{H}^T\mathbf{T}$, *and* $rank(\mathbf{H}) = s < L$,$\mathbf{B}^* \in \mathcal{G}^+$ *and* $\boldsymbol{\beta}^* \in \mathbb{R}^L$ *are determined by Lemma 3.1 and 3.2. There is a symmetric and definite matrix* $\mathbf{B} \in \mathcal{G}^+$, *such that*

$$\mathbf{B} = \mathbf{B}^* + \frac{(\mathbf{c} - \mathbf{B}^*\boldsymbol{\beta}^*)(\mathbf{c} - \mathbf{B}^*\boldsymbol{\beta}^*)^T}{(\mathbf{c} - \mathbf{B}^*\boldsymbol{\beta}^*)^T\boldsymbol{\beta}^*}, \tag{44}$$

and

$$\mathbf{B}\boldsymbol{\beta}^* = \mathbf{c}. \tag{45}$$

Proof. Let

$$\mathbf{B} = \mathbf{B}^* + \lambda\mathbf{u}\mathbf{u}^T, \tag{46}$$

$\mathbf{u} \in \mathbb{R}^L$ is an unknown vector, we call $\mathbf{B}$ is the $rank - 1$ modification of $\mathbf{B}^*$. For given $\boldsymbol{\beta}^* \in \mathbb{R}^L$, we require that

$$\mathbf{B}\boldsymbol{\beta}^* = \mathbf{c}, \tag{47}$$

this indicates that

$$(\mathbf{B}^* + \lambda\mathbf{u}\mathbf{u}^T)\boldsymbol{\beta}^* = \mathbf{c}, \tag{48}$$

then

$$\lambda\mathbf{u}\mathbf{u}^T\boldsymbol{\beta}^* = \mathbf{c} - \mathbf{B}^*\boldsymbol{\beta}^*. \tag{49}$$

Because $\mathbf{u}^T\boldsymbol{\beta}^* \in \mathbb{R}$ is a scale, (49) shows that $\mathbf{u}$ and $\mathbf{c} - \mathbf{B}^*\boldsymbol{\beta}^*$ are same direction vectors, there exists a real number α such that

$$\mathbf{u} = \alpha(\mathbf{c} - \mathbf{B}^*\boldsymbol{\beta}^*). \tag{50}$$

With (49) and (50), we have

$$\lambda\alpha^2(\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*)(\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*)^T\boldsymbol{\beta}^* = (\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*). \tag{51}$$

Multiplying with $(\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*)^T$ on the left side of (51), we have

$$\lambda\alpha^2(\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*)^T(\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*)(\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*)^T\boldsymbol{\beta}^* = (\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*)^T(\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*). \tag{52}$$

Because $(\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*)^T(\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*)$ is also a scale, we can get the result from (52)

$$\lambda\alpha^2(\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*)^T\boldsymbol{\beta}^* = 1. \tag{53}$$

Then,

$$\lambda\alpha^2 = \frac{1}{(\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*)^T\boldsymbol{\beta}^*}. \tag{54}$$

Then,

$$\lambda\mathbf{u}\mathbf{u}^T = \lambda\alpha^2(\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*)(\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*)^T = \frac{(\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*)(\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*)^T}{(\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*)^T\boldsymbol{\beta}^*}. \tag{55}$$

With (46) and (55), we have

$$\mathbf{B} = \mathbf{B}^* + \frac{(\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*)(\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*)^T}{(\mathbf{c}-\mathbf{B}^*\boldsymbol{\beta}^*)^T\boldsymbol{\beta}^*}. \tag{56}$$

□

3.3 Rank-2 Approximation Optimal Solution

In this section, we introduce the rank-2 method.

Theorem 3.5. *Let* $\mathbf{M} = \mathbf{H}^T\mathbf{H}, c = \mathbf{H}^T\mathbf{T}$, *and* $rank(\mathbf{H}) = s < L$, $\mathbf{B}^* \in \mathcal{G}^+$ *and* $\boldsymbol{\beta}^* \in \mathbb{R}^L$ *are determined by Lemma 3.1 and 3.2. There is a symmetric and definite matrix* $\mathbf{B} \in \mathcal{G}^+$, *such that*

$$\mathbf{B} = \mathbf{B}^* + \frac{\mathbf{c}\mathbf{c}^T}{\mathbf{c}^T\boldsymbol{\beta}^*} - \frac{\mathbf{B}^*\boldsymbol{\beta}^*(\boldsymbol{\beta}^*)^T\mathbf{B}^*}{(\boldsymbol{\beta}^*)^T\mathbf{B}^*\boldsymbol{\beta}^*}, \tag{57}$$

and

$$\mathbf{B}\boldsymbol{\beta}^* = \mathbf{c}. \tag{58}$$

Proof. Let

$$\mathbf{B} = \mathbf{B}^* + \lambda_1 \mathbf{u}\mathbf{u}^T + \lambda_2 \mathbf{v}\mathbf{v}^T, \tag{59}$$

$\mathbf{u}, \mathbf{v} \in \mathbb{R}^L$ are an unknown vectors, we call $\mathbf{B}$ is the $rank - 2$ modification of $\mathbf{B}^*$. For given $\boldsymbol{\beta}^* \in \mathbb{R}^L$, we require that

$$\mathbf{B}\boldsymbol{\beta}^* = \mathbf{c}, \tag{60}$$

this indicates that

$$(\mathbf{B}^* + \lambda_1 \mathbf{u}\mathbf{u}^T + \lambda_2 \mathbf{v}\mathbf{v}^T)\boldsymbol{\beta}^* = \mathbf{c}, \tag{61}$$

then

$$(\lambda_1 \mathbf{u}\mathbf{u}^T + \lambda_2 \mathbf{v}\mathbf{v}^T)\boldsymbol{\beta}^* = \mathbf{c} - \mathbf{B}^*\boldsymbol{\beta}^*. \tag{62}$$

Difference from Theorem 3.4, there are infinity number of solution for (62), among of them, we let

$$\begin{aligned} \lambda_1 \mathbf{u}\mathbf{u}^T \boldsymbol{\beta}^* &= \mathbf{c}, \\ \lambda_2 \mathbf{v}\mathbf{v}^T \boldsymbol{\beta}^* &= -\mathbf{B}^*\boldsymbol{\beta}^*. \end{aligned} \tag{63}$$

Using the same idea of Theorem 3.4, (63) shows that there exist two real numbers α, γ such that

$$\begin{aligned} \mathbf{u} &= \alpha \mathbf{c}, \\ \mathbf{v} &= -\gamma \mathbf{B}^*\boldsymbol{\beta}^*. \end{aligned} \tag{64}$$

With (63) and (64), we have

$$\begin{aligned} \lambda_1 \alpha^2 \mathbf{c}\mathbf{c}^T \boldsymbol{\beta}^* &= \mathbf{c}, \\ \lambda_2 \gamma^2 \mathbf{B}^*\boldsymbol{\beta}^* (\mathbf{B}^*\boldsymbol{\beta}^*)^T \boldsymbol{\beta}^* &= -\mathbf{B}^*\boldsymbol{\beta}^*. \end{aligned} \tag{65}$$

Multiplying with $\mathbf{c}^T$ and $(\mathbf{B}^*\boldsymbol{\beta}^*)^T$ on the left side of (65), we have

$$\begin{aligned} \lambda_1 \alpha^2 \mathbf{c}^T \boldsymbol{\beta}^* &= 1, \\ \lambda_2 \gamma^2 (\boldsymbol{\beta}^*)^T \mathbf{B}^*\boldsymbol{\beta}^* &= -1. \end{aligned} \tag{66}$$

Then,

$$\begin{aligned} \lambda_1 \alpha^2 &= \tfrac{1}{\mathbf{c}^T \boldsymbol{\beta}^*}, \\ \lambda_2 \gamma^2 &= \tfrac{-1}{(\boldsymbol{\beta}^*)^T \mathbf{B}^* \boldsymbol{\beta}^*}. \end{aligned} \tag{67}$$

With (59) and (67), we have

$$\mathbf{B} = \mathbf{B}^* + \frac{\mathbf{c}\mathbf{c}^T}{\mathbf{c}^T\boldsymbol{\beta}^*} - \frac{\mathbf{B}^*\boldsymbol{\beta}^*(\boldsymbol{\beta}^*)^T\mathbf{B}^*}{(\boldsymbol{\beta}^*)^T\mathbf{B}^*\boldsymbol{\beta}^*}. \tag{68}$$

□

4 Algorithm

With the discussions in section 3, we will propose a new algorithm for *ELM*.

New Algorithm ELM*:*

Given a training set

$$\mathcal{S} = \{(\mathbf{x}_1, t_1), \cdots, (\mathbf{x}_N, t_N)\},$$

active function $G(x)$*, and hidden node number* L*;* $\boldsymbol{\beta} = (\beta_1, \beta_2, \cdots, \beta_L)^T, Y = (t_1, t_2, \cdots, t_N)^T$*;*

Step 1: Randomly assign input weight $\mathbf{w}_i$ *and* b_i*,* $i = 1, 2, \cdots, L$*;*

Step 2: Calculate the hidden layer output matrix $\mathbf{H}$ *by (11),* $r = rank(\mathbf{H})$*;*

Step 3:

if $r = L$*, Calculate the output weight by* $\boldsymbol{\beta}^\dagger = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{T}$*;*

if $r = N$*, Calculate the output weight by* $\boldsymbol{\beta}^\dagger = \mathbf{H}^T(\mathbf{H}\mathbf{H}^T)^{-1}\mathbf{T}$*;*

Otherwise, let $\mathbf{M} = \mathbf{H}^T\mathbf{H}, c = \mathbf{H}^T\mathbf{T}, \varepsilon = 10^{-4}$*, solve optimal models*

$$\min_{\mathbf{B}\in\mathcal{G}^+} \|\mathbf{M} - \mathbf{B}\|,$$

and

$$\min_{\boldsymbol{\beta}\in\mathbb{R}^L} \|\mathbf{B}^*\boldsymbol{\beta} - \mathbf{c}\|,$$

get optimal solutions $\mathbf{B}^*, \boldsymbol{\beta}^*$*;*

if $\|\mathbf{c} - \mathbf{B}^*\boldsymbol{\beta}^*\| > \varepsilon$*, calculate*

$$\mathbf{B} = \mathbf{B}^* + \frac{(\mathbf{c} - \mathbf{B}^*\boldsymbol{\beta}^*)(\mathbf{c} - \mathbf{B}^*\boldsymbol{\beta}^*)^T}{(\mathbf{c} - \mathbf{B}^*\boldsymbol{\beta}^*)^T\boldsymbol{\beta}^*},$$

then Calculate the output weight by $\boldsymbol{\beta}^{\dagger} = \mathbf{B}^{-1}\mathbf{c}$*;*
else
calculate

$$\mathbf{B} = \mathbf{B}^{*} + \frac{\mathbf{c}\mathbf{c}^{T}}{\mathbf{c}^{T}\boldsymbol{\beta}^{*}} - \frac{\mathbf{B}^{*}\boldsymbol{\beta}^{*}(\boldsymbol{\beta}^{*})^{T}\mathbf{B}^{*}}{(\boldsymbol{\beta}^{*})^{T}\mathbf{B}^{*}\boldsymbol{\beta}^{*}},$$

then Calculate the output weight by $\boldsymbol{\beta}^{\dagger} = \mathbf{B}^{-1}\mathbf{c}$.

5 Experimental Performances

From the above discussion, the proposed new ELM is as fast as ELM and the extra time is likely to be spent on the third step of the algorithm when the hidden layer output matrix $\mathbf{H}$ is neither row full rank nor column full rank. But in fact, the extra time spent in the third phase of the algorithm is $\mathcal{O}(L^2)$, which is short in comparison with that spent in the third phase of the ELM algorithm that calculates the output weights.

In the rest part of this section, the comparison of the performance of the proposed new ELM algorithm and the ELM algorithm is conducted. All the simulations for ELM and the new ELM algorithms are both carried out in the Matlab 7.0 environment running in Intel Celeron 743 CPU with the speed of 1.30 GHz and in Intel Core 2 Duo CPU. The activation function used in both algorithm is the sigmoidal function $g(x) = 1/(1 + e^{-x})$.

5.1 Benchmarking with a regression problem: approximation of 'SinC' function with noise

First, we use the 'Sinc' function to measure the performance of the original and the new algorithms. The target function is as follows.

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

Table 1. Performance comparison for learning function: SinC

Algorithms	Time (s)		Accuracy		No. of nodes
	Training	Testing	Training	Testing	
ELM	3.1250×10^{-4}	0.0013	0.1913	0.1544	5
New ELM	9.3750×10^{-4}	9.3750×10^{-4}	0.1890	0.1506	5

A training set (X_i, t_i) and testing set (X_i, t_i) with 300 data, respectively, are created where X_i in both the training data and the testing data are uniformly randomly chosen from $[-10, 10]$. Large uniform noise distributed in $[-0.2, 0.2]$ has been used in all training data to obtain a 'real' regression problem. The experiment is carried out on these data as follows. There are 5 hidden nodes assigned for both the original ELM and the new ELM algorithms. 50 trials have been conducted for the ELM algorithm to eliminate the random error and the results shown are the average performance. Results shown in Table 1 include training time, testing time, training accuracy, testing accuracy and the number of nodes of both algorithms.

It can be seen from Table 1 that the new ELM algorithm spent 9.3750×10^{-4}s CPU time obtaining testing accuracy of 0.1506 with training error of 0.1890 while the original ELM algorithm takes 3.1250×10^{-4}s CPU time to reach a larger testing error 0.1544 with training error of 0.1913. Fig. 2 shows the expected and approximated results of ELM algorithm and Fig. 3 shows the real and the approximated results of the new ELM algorithm. The results show that our new ELM algorithm not only takes less time to train samples but also has a higher accuracy than the original ELM algorithm.

5.2 Benchmarking with real-world applications

In this section, the ELM and the new ELM performances have been tested on 7 real-world applications, including 5 classification tasks and 2 regression tasks. All the examples are from UCI repository of machine learning

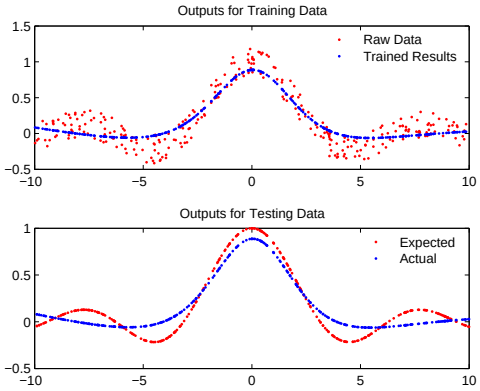

Fig. 2. Outputs of the original ELM algorithm

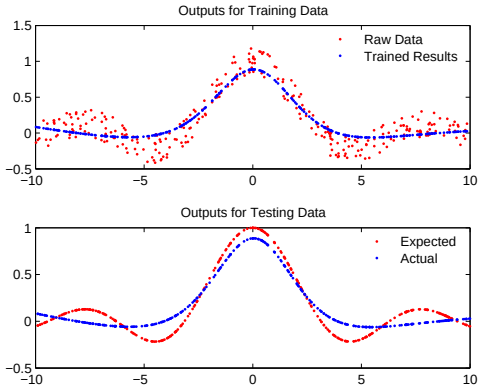

Fig. 3. Outputs of the new ELM algorithm

databases [20]. The speculations of the data sets are listed on the Table 3 and Table 7. In each case that has only one data table, 75% and 25% of samples in the problem are randomly chosen for training and testing respectively before each trial.

5.2.1 Benchmarking with real classification applications

First, we conducted the performance comparison of the new proposed ELM and the original ELM algorithms for a real medical diagnosis problem: Diabetes, using "Pima Indians Diabetes Database" produced in the Applied Physics Laboratory, Johns Hopkins University, 1988. The diagnostic, binary-valued variable investigated is used to show whether the patient shows the signs of diabetes in accordance with World Health Organization criteria. In the database, there are 768 women over 21 resident in Phoenix, Arizona. All samples belong to either positive or negative class and all values of input are drawn from $[0, 1]$. As conducted in [21, 22, 23, 24], 75% and 25% of samples in the problem are randomly chosen for training and testing respectively, at each trial.

Fifty trials were conducted for the two algorithms and the average results are given in Table 2, which shows that in our simulation, the ELM can

Table 2. Performance comparison for Diabetes

Algorithms	Time (s)		Success Rate (%)		No. of nodes
	Training	Testing	Training	Testing	
ELM	0.0059	0.0013	78.99	76.73	20
New ELM	0.0047	0.0025	78.78	77.14	20

reach the testing rate 76.73% with the training time of 0.0059s with 20 nodes whereas the new ELM algorithm with the equal number of nodes can achieve a higher rate of 77.14% with the less time consumption 0.0047s on samples training. The Fig. 4 and Fig. 5 respectively show the varying of training accuracy and testing accuracy of two algorithms in 50 trials.

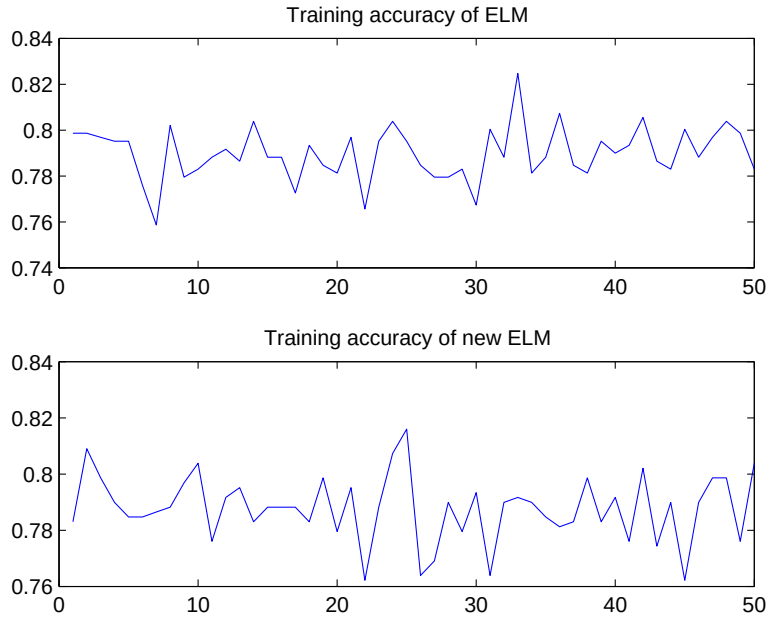

Fig. 4. Training accuracy for Diabetes

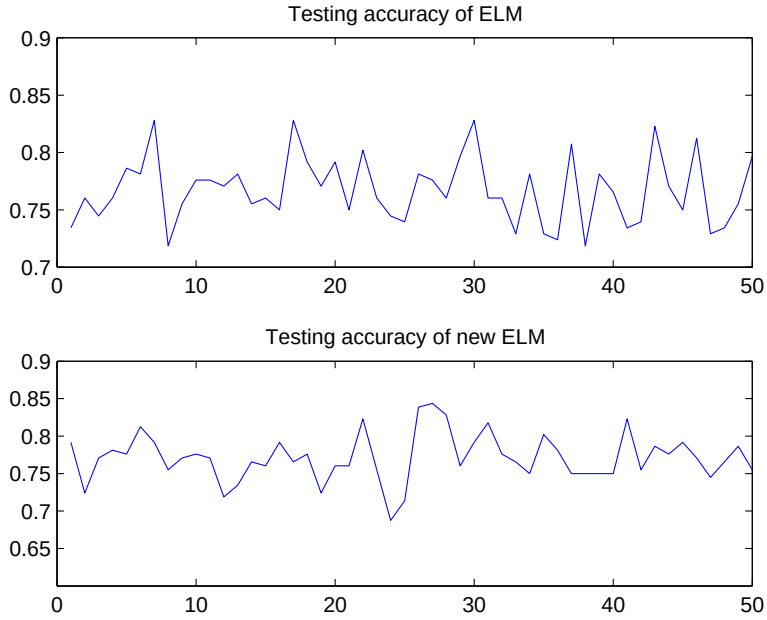

Fig. 5. Testing accuracy for Diabetes

The ELM and the new ELM performances have been tested on other 4 multiclass databases including Statlog (Landsat Satellite), Pendigits, Glass Identification (Glass ID) and Teaching Assistant Evaluation (Teaching). These cases range in size from small to large. The training data sets and testing data sets are randomly generated from the original database for each application. Fifty trials have been conducted for both algorithms. From Table 4, Table 5 and Table 6, we can see that for classification cases, generally, the same training and testing accuracy of the new ELM is the same to that of ELM, its learning speed is as fast as that of ELM and the new ELM and ELM have similar robustness property.

Table 3. Speculations of real-world applications and the number of nodes for each

Data sets	# Observations		# Attributes	Associated Tasks	#Nodes
	Training	Testing	Continuous		
Statlog	4435	2000	36	Classification	20
Pendigits	7494	3498	16	Classification	20
Glass ID	160	54	9	Classification	10
Teaching	113	38	5	Classification	5

Table 4. Comparison training and testing accuracy (error) in classification cases

Data sets	ELM		new ELM	
	Training	Testing	Training	Testing
Statlog	0.8159	0.8125	0.8164	0.8117
Pendigits	0.8589	0.8550	0.8602	0.8573
Glass ID	0.9463	0.4433	0.9438	0.4630
Teaching	0.7968	0.9937	0.7972	0.9832

Table 5. Comparison of training and testing RMSE in classification cases

Data sets	ELM		new ELM	
	Training	Testing	Training	Testing
Statlog	0.0114	0.0104	0.0086	0.0086
Pendigits	0.0152	0.0156	0.0100	0.0108
Glass ID	0.0031	0.0281	8.9720×10^{-16}	1.1214×10^{-16}
Teaching	0.0103	0.0210	0.0069	0.0290

Table 6. Comparison of average training and testing time in classification cases

Data sets	ELM(s)		new ELM(s)	
	Training	Testing	Training	Testing
Statlog	0.0208	0.0208	0.0188	0.0198
Pendigits	0.0396	0.0313	0.0344	0.0333
Glass ID	0.0012	3.1200×10^{-4}	0.0019	0.0012
Teaching	9.3601×10^{-4}	3.1200×10^{-4}	0.0016	0

5.2.2 Benchmarking with real regression applications

The performances of ELM and the new ELM were also compared on 2 real-world regression cases: Housing and Concrete Slump (Slump). The specifications of the data sets are listed in Table 7. Average results of 50 trials of simulations for each algorithm are showed in Table 8, Table 9 and Table 10. The results suggest that in both cases, the new ELM has a better performance than ELM. The new ELM has a much better robustness property in regression cases than ELM.

Table 7. Speculations of real-world applications and the number of nodes for each

Data sets	# Observations		# Attributes	Associated Tasks	#Nodes
	Training	Testing	Continuous		
Housing	378	126	14	Regression	20
Slump	76	27	10	Regression	10

Table 8. Comparison training and testing accuracy (error) in regression cases

Data sets	ELM		new ELM	
	Training	Testing	Training	Testing
Housing	8.2177	2.4302×10^8	437.2138	400.0774
		7.5124(best)		
Slump	7.9927	3.2439×10^5	154.8178	153.4486
		8.8406(best)		132.3626 (best)

Table 9. Comparison of training and testing RMSE in regression cases

Data sets	ELM		new ELM	
	Training	Testing	Training	Testing
Housing	0.2181	1.0854×10^9	20.1151	42.3212
Slump	0.1394	2.1991×10^6	8.0451	11.1593

Table 10. Comparison of average training and testing time in regression cases

Data sets	ELM(s)		new ELM(s)	
	Training	Testing	Training	Testing
Housing	0.0019	0.0031	0.0031	0.0012
Slump	6.2500×10^{-4}	0.0022	3.1250×10^{-4}	0.0034

6 Conclusions and Future Works

In this paper, with regression problem, *ELM* is studied again. Finally, a better and complete algorithm is presented. The key idea comes from the step 3 of *ELM*. The equations($\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$) are reformulated as a optimal model (13). With the optimality, another equations is given $\mathbf{H}^T\mathbf{H}\boldsymbol{\beta} = \mathbf{H}^T\mathbf{T}$. It must have one solution at least with Theorem 3.1. If $\mathbf{H}$ is column full rank, optimal approximation solution is solved by (31); If $\mathbf{H}$ is row full rank, optimal approximation solution is solved by (36); If $\mathbf{H}$ is neither column nor row full rank, the rank-1 (44) and rank-2 (57) methods are used to get optimal approximation solution. In future, we will have some experiments on benchmark data sets from UCI (University of California, Irvine) repository of machine learning databases for regression.

Acknowledgment

This research has been supported by the National Natural Science Foundation under Grants (Nos. 90818020,60873206) and Natural Science Foundation and Education Department of Zhejiang under Grants (Nos. Y7080235, Y200805339). This research has been supported partly by Hangzhou Public Transport Corporation.

References

[1] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: A new learning scheme of feedforward neural networks," in *Proceedings of International Joint Conference on Neural Networks (IJCNN2004)*, vol. 2, (Budapest, Hungary), pp. 985–990, 25-29 July, 2004.

[2] G.-B. Huang and C.-K. Siew, "Extreme learning machine: RBF network case," in *Proceedings of the Eighth International Conference on Control, Automation, Robotics and Vision (ICARCV 2004)*, vol. 2, (Kunming, China), pp. 1029–1036, 6-9 Dec, 2004.

[3] M.-B. Li, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "Fully complex extreme learning machine," *Neurocomputing*, vol. 68, pp. 306–314, 2005.

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

[5] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, pp. 489–501, 2006.

[6] G.-B. Huang, L. Chen, and C.-K. Siew, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Transactions on Neural Networks*, vol. 17, no. 4, pp. 879–892, 2006.

[7] G.-B. Huang and L. Chen, "Convex incremental extreme learning machine," *Neurocomputing*, vol. 70, pp. 3056–3062, 2007.

[8] R. Zhang, G.-B. Huang, N. Sundararajan, and P. Saratchandran, "Multi-category classification using an extreme learning machine for microarray gene expression cancer diagnosis," *IEEE/ACM Transactions on*

Computational Biology and Bioinformatics, vol. 4, no. 3, pp. 485–495, 2007.

[9] L. C. C.-K. S. Guang-Bin Huanga, Ming-Bin Lia, "Incremental extreme learning machine with fully complex hidden nodes," *Neurocomputing*, vol. 71, pp. 576–583, 2008.

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

[11] Y. Lan, Y. C. Soh, and G.-B. Huang, "Extreme learning machine based bacterial protein subcellular localization prediction," in *Neural Networks, 2008. IJCNN 2008. (IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pp. 1859 –1863, 1-8 2008.

[12] G. Feng, G.-B. Huang, Q. Lin, and R. Gay, "Error minimized extreme learning machine with growth of hidden nodes and incremental learning," *IEEE Transactions on Neural Networks*, vol. 20, no. 8, pp. 1352–1357, 2009.

[13] H.-J. Rong, G.-B. Huang, N. Sundararajan, and P. Saratchandran, "Online sequential fuzzy extreme learning machine for function approximation and classification problems," *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, vol. 39, no. 4, pp. 1067–1072, 2009.

[14] J. Cao, Z. Lin, and G. bin Huang, "Composite function wavelet neural networks with extreme learning machine," *Neurocomputing*, vol. 73, no. 7-9, pp. 1405 – 1416, 2010. Advances in Computational Intelligence and Learning - 17th European Symposium on Artificial Neural Networks 2009, 17th European Symposium on Artificial Neural Networks 2009.

[15] G.-B. Huang, X. Ding, and H. Zhou, "Optimization method based extreme learning machine for classification," *Neurocomputing*, vol. In Press, Accepted Manuscript, pp. –, 2010.

[16] G.-B. Huang and H. A. Babri, "Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions," *IEEE Transactions on Neural Networks*, vol. 9, no. 1, pp. 224–229, 1998.

[17] G.-B. Huang, Y.-Q. Chen, and H. A. Babri, "Classification ability of single hidden layer feedforward neural networks," *IEEE Transactions on Neural Networks*, vol. 11, no. 3, pp. 799–801, 2000.

[18] G.-B. Huang, "Learning capability and storage capacity of two-hidden-layer feedforward networks," *IEEE Transactions on Neural Networks*, vol. 14, no. 2, pp. 274–281, 2003.

[19] G.-B. Huang, L. Chen, and C.-K. Siew, "Universal approximation using incremental feedforward networks with arbitrary input weights," in *Technical Report ICIS/46/2003*, (School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore), Oct. 2003.

[20] C. Blake, C. Merz, UCI repository of machine learning databases, in: http://www.ics.uci.edu/˜mlearn/MLRepository.html, Department of Information and Computer Sciences, University of California, Irvine, USA, 1998.

[21] Y. Freund, R.E. Schapire, Experiments with a new boosting algorithm, in: International Conference on Machine Learning, 1996, pp. 148-156.

[22] G. Rätsch, T. Onoda, K.R. Müller, An improvement of AdaBoost to avoid overfitting, in: Proceedings of the Fifth International Conference on Neural Information Processing (ICONIP' 1998), 1998.

[23] E. Romero, R. Alquézar, A new incremental method for function approximation using feed-forward neural networks, in: Proceedings of INNS-IEEE International Joint Conference on Neural Networks (IJCNN2002), 2002, pp. 1968-1973.

[24] D.R. Wilson, T.R. Martinez, Heterogeneous radial basis function networks, in: Proceedings of the International Conference on Neural Networks (ICNN 96), June 1996, pp. 1263-1267.