ARTICLE IN PRESS

Available online at www.sciencedirect.com

SCIENCE DIRECT®

NEUROCOMPUTING

Neurocomputing ▮ (▮▮▮▮) ▮▮▮–▮▮▮

www.elsevier.com/locate/neucom

Extreme learning machine: Theory and applications

Guang-Bin Huang*, Qin-Yu Zhu, Chee-Kheong Siew

School of Electrical and Electronic Engineering, NanyangTechnological University, Nanyang Avenue, Singapore 639798, Singapore

Received 27 January 2005; received in revised form 1 December 2005; accepted 3 December 2005
Communicated by J. Tin-Yau Kwok

Abstract

It is clear 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. Unlike these conventional implementations, this paper proposes a new learning algorithm called **e**xtreme **l**earning **m**achine (ELM) for **s**ingle-hidden **l**ayer **f**eedforward neural **n**etworks (SLFNs) which randomly chooses hidden nodes and analytically determines the output weights of SLFNs. In theory, this algorithm tends to provide good generalization performance at extremely fast learning speed. The experimental results based on a few artificial and real benchmark function approximation and classification problems including very large complex applications show that the new algorithm can produce good generalization performance in most cases and can learn thousands of times faster than conventional popular learning algorithms for feedforward neural networks.[1]
© 2006 Elsevier B.V. All rights reserved.

Keywords: Feedforward neural networks; Back-propagation algorithm; Extreme learning machine; Support vector machine; Real-time learning; Random node

1. Introduction

Feedforward neural networks have been extensively used in many fields due to their ability: (1) to approximate complex nonlinear mappings directly from the input samples; and (2) to provide models for a large class of natural and artificial phenomena that are difficult to handle using classical parametric techniques. On the other hand, there lack faster learning algorithms for neural networks. The traditional learning algorithms are usually far slower than required. It is not surprising to see that it may take several hours, several days, and even more time to train neural networks by using traditional methods.

From a mathematical point of view, research on the approximation capabilities of feedforward neural networks has focused on two aspects: universal approximation on compact input sets and approximation in a finite set of training samples. Many researchers have explored the universal approximation capabilities of standard multilayer feedforward neural networks. Hornik [7] proved that if the activation function is continuous, bounded and nonconstant, then continuous mappings can be approximated in measure by neural networks over compact input sets. Leshno [17] improved the results of Hornik [7] and proved that feedforward networks with a nonpolynomial activation function can approximate (in measure) continuous functions. In real applications, the neural networks are trained in finite training set. For function approximation in a finite training set, Huang and Babri [11] shows that a **s**ingle-hidden **l**ayer **f**eedforward neural **n**etwork (**SLFN**) with at most N hidden nodes and with almost any nonlinear activation function can exactly learn N distinct observations. It should be noted that the input weights (linking the input layer to the first hidden layer) and hidden layer biases need to be adjusted in all these previous theoretical research works as well as in almost all practical learning algorithms of feedforward neural networks.

*Corresponding author. Tel.: +65 6790 4489; fax: +65 6793 3318.
E-mail address: egbhuang@ntu.edu.sg (G.-B. Huang).
URL: http://www.ntu.edu.sg/home/egbhuang/.

[1]For the preliminary idea of the ELM algorithm, refer to "Extreme Learning Machine: A New Learning Scheme of Feedforward Neural Networks", Proceedings of International Joint Conference on Neural Networks (IJCNN2004), Budapest, Hungary, 25–29 July, 2004.

0925-2312/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2005.12.126

Traditionally, all the parameters of the feedforward networks need to be tuned and thus there exists the dependency between different layers of parameters (weights and biases). For past decades, gradient descent-based methods have mainly been used in various learning algorithms of feedforward neural networks. However, it is clear that gradient descent-based learning methods are generally very slow due to improper learning steps or may easily converge to local minima. And many iterative learning steps may be required by such learning algorithms in order to obtain better learning performance.

It has been shown [23,10] that SLFNs (with N hidden nodes) with randomly chosen input weights and hidden layer biases (and such hidden nodes can thus be called random hidden nodes) can exactly learn N distinct observations. Unlike the popular thinking and most practical implementations that all the parameters of the feedforward networks need to be tuned, one may not necessarily adjust the input weights and first hidden layer biases in applications. In fact, some simulation results on artificial and real large applications in our work [16] have shown that this method not only makes learning extremely fast but also produces good generalization performance.

In this paper, we first rigorously prove that the input weights and hidden layer biases of SLFNs can be randomly assigned if the activation functions in the hidden layer are infinitely differentiable. After the input weights and the hidden layer biases are chosen randomly, SLFNs can be simply considered as a linear system and the output weights (linking the hidden layer to the output layer) of SLFNs can be *analytically determined* through simple generalized inverse operation of the hidden layer output matrices. Based on this concept, this paper proposes a simple learning algorithm for SLFNs called **e**xtreme **l**earning **m**achine (**ELM**) whose learning speed can be thousands of times faster than traditional feedforward network learning algorithms like back-propagation (BP) algorithm while obtaining better generalization performance. Different from traditional learning algorithms the proposed learning algorithm not only tends to reach the smallest training error but also the smallest norm of weights. Bartlett's [1] theory on the generalization performance of feedforward neural networks states for feedforward neural networks reaching smaller training error, the smaller the norm of weights is, the better generalization performance the networks tend to have. Therefore, the proposed learning algorithm tends to have good generalization performance for feedforward neural networks.

As the new proposed learning algorithm can be easily implemented, tends to reach the smallest training error, obtains the smallest norm of weights and the good generalization performance, and runs extremely fast, in order to differentiate it from the other popular SLFN learning algorithms, it is called the extreme learning machine in the context of this paper.

This paper is organized as follows. Section 2 rigorously proves that the input weights and hidden layer biases of SLFNs can be randomly assigned if the activation functions in the hidden layer are infinitely differentiable. Section 3 further proposes the new ELM learning algorithm for single-hidden layer feedforward neural networks (SLFNs). Performance evaluation is presented in Section 4. Discussions and conclusions are given in Section 5. The Moore–Penrose generalized inverse and the minimum norm least-squares solution of a general linear system which play an important role in developing our new ELM learning algorithm are briefed in the Appendix.

2. Single hidden layer feedforward networks (SLFNs) with random hidden nodes

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

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

where $\mathbf{w}_i = [w_{i1}, w_{i2}, \ldots, w_{in}]^{\mathrm{T}}$ is the weight vector connecting the ith hidden node and the input nodes, $\beta_i = [\beta_{i1}, \beta_{i2}, \ldots, \beta_{im}]^{\mathrm{T}}$ is the weight vector connecting the ith hidden node and the output nodes, and b_i is the threshold of the ith hidden node. $\mathbf{w}_i \cdot \mathbf{x}_j$ denotes the inner product of $\mathbf{w}_i$ and $\mathbf{x}_j$. The output nodes are chosen linear in this paper.

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

$$\sum_{i=1}^{\tilde{N}} \beta_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) = \mathbf{t}_j, \quad j = 1, \ldots, N. \tag{2}$$

The above N equations can be written compactly as

$$\mathbf{H}\beta = \mathbf{T}, \tag{3}$$

where

$$\mathbf{H}(\mathbf{w}_1, \ldots, \mathbf{w}_{\tilde{N}}, b_1, \ldots, b_{\tilde{N}}, \mathbf{x}_1, \ldots, \mathbf{x}_N) = \begin{bmatrix} g(\mathbf{w}_1 \cdot \mathbf{x}_1 + b_1) & \cdots & g(\mathbf{w}_{\tilde{N}} \cdot \mathbf{x}_1 + b_{\tilde{N}}) \\ \vdots & \cdots & \vdots \\ g(\mathbf{w}_1 \cdot \mathbf{x}_N + b_1) & \cdots & g(\mathbf{w}_{\tilde{N}} \cdot \mathbf{x}_N + b_{\tilde{N}}) \end{bmatrix}_{N \times \tilde{N}}, \tag{4}$$

$$\beta = \begin{bmatrix} \beta_1^{\mathrm{T}} \\ \vdots \\ \beta_{\tilde{N}}^{\mathrm{T}} \end{bmatrix}_{\tilde{N} \times m} \quad \text{and} \quad \mathbf{T} = \begin{bmatrix} \mathbf{t}_1^{\mathrm{T}} \\ \vdots \\ \mathbf{t}_N^{\mathrm{T}} \end{bmatrix}_{N \times m}. \tag{5}$$

As named in Huang et al. [11,10], $\mathbf{H}$ is called the hidden layer output matrix of the neural network; the ith column of $\mathbf{H}$ is the ith hidden node output with respect to inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$.

If the activation function g is infinitely differentiable we can prove that the required number of hidden nodes $\tilde{N} \leqslant N$. Strictly speaking, we have[2]

Theorem 2.1. *Given a standard SLFN with N hidden nodes and activation function $g : R \to R$ which is infinitely differentiable in any interval, for N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$, where $\mathbf{x}_i \in \mathbf{R}^n$ and $\mathbf{t}_i \in \mathbf{R}^m$, for any $\mathbf{w}_i$ and b_i randomly chosen from any intervals of $\mathbf{R}^n$ and $\mathbf{R}$, respectively, according to any continuous probability distribution, then with probability one, the hidden layer output matrix $\mathbf{H}$ of the SLFN is invertible and $\|\mathbf{H}\beta - \mathbf{T}\| = 0$.*

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

Following the same proof method of Tamura and Tateishi ([23], p. 252) and our previous work ([10], Theorem 2.1), it can be easily proved by contradiction that vector $\mathbf{c}$ does not belong to any subspace whose dimension is less than N.

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

$$(\alpha, \mathbf{c}(b_i) - \mathbf{c}(a)) = \alpha_1 \cdot g(b_i + d_1) + \alpha_2 \cdot g(b_i + d_2) + \cdots + \alpha_N \cdot g(b_i + d_N) - z = 0, \tag{6}$$

where $d_k = \mathbf{w}_i \cdot \mathbf{x}_k$, $k = 1, \ldots, N$ and $z = \alpha \cdot \mathbf{c}(a)$, $\forall b_i \in (a, b)$. Assume $\alpha_N \neq 0$, Eq. (6) can be further written as

$$g(b_i + d_N) = -\sum_{p=1}^{N-1} \gamma_p g(b_i + d_p) + z/\alpha_N, \tag{7}$$

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

$$g^{(l)}(b_i + d_N) = -\sum_{p=1}^{N-1} \gamma_p g^{(l)}(b_i + d_p), \quad l = 1, 2, \ldots, N, N+1, \ldots, \tag{8}$$

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

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

Such activation functions include the sigmoidal functions as well as the radial basis, sine, cosine, exponential, and many other nonregular functions as shown in Huang and Babri [11].

Furthermore, we have

Theorem 2.2. *Given any small positive value $\varepsilon > 0$ and activation function $g : R \to R$ which is infinitely differentiable in any interval, there exists $\tilde{N} \leqslant N$ such that for N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$, where $\mathbf{x}_i \in \mathbf{R}^n$ and $\mathbf{t}_i \in \mathbf{R}^m$, for any $\mathbf{w}_i$ and b_i randomly chosen from any intervals of $\mathbf{R}^n$ and $\mathbf{R}$, respectively, according to any continuous probability distribution, then with probability one, $\|\mathbf{H}_{N \times \tilde{N}} \beta_{\tilde{N} \times m} - \mathbf{T}_{N \times m}\| < \varepsilon$.*

Proof. The validity of the theorem is obvious, otherwise, one could simply choose $\tilde{N} = N$ which makes $\|\mathbf{H}_{N \times \tilde{N}} \beta_{\tilde{N} \times m} - \mathbf{T}_{N \times m}\| < \varepsilon$ according to Theorem 2.1. □

3. Proposed extreme learning machine (ELM)

Based on Theorems 2.1 and 2.2 we can propose in this section an extremely simple and efficient method to train SLFNs.

3.1. Conventional gradient-based solution of SLFNs

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

$$\|\mathbf{H}(\hat{\mathbf{w}}_1, \ldots, \hat{\mathbf{w}}_{\tilde{N}}, \hat{b}_1, \ldots, \hat{b}_{\tilde{N}})\hat{\beta} - \mathbf{T}\| = \min_{\mathbf{w}_i, b_i, \beta} \|\mathbf{H}(\mathbf{w}_1, \ldots, \mathbf{w}_{\tilde{N}}, b_1, \ldots, b_{\tilde{N}})\beta - \mathbf{T}\| \tag{9}$$

which is equivalent to minimizing the cost function

$$E = \sum_{j=1}^{N} \left(\sum_{i=1}^{\tilde{N}} \beta_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) - \mathbf{t}_j \right)^2. \tag{10}$$

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

$$\mathbf{W}_k = \mathbf{W}_{k-1} - \eta \frac{\partial E(\mathbf{W})}{\partial \mathbf{W}}. \tag{11}$$

Here η is a learning rate. The popular learning algorithm used in feedforward neural networks is the BP learning algorithm where gradients can be computed efficiently by propagation from the output to the input. There are several issues on BP learning algorithms:

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

[2] In fact, the theorem and its proof are also linearly valid for the case $g_i(\mathbf{x}) = g(\|\mathbf{x} - \mathbf{w}_i\|/b_i), \mathbf{w}_i \in \mathbf{R}^n, b_i \in R^+$.

(2) Another peculiarity of the error surface that impacts the performance of the BP learning algorithm is the presence of local minima [6]. It is undesirable that the learning algorithm stops at a local minima if it is located far above a global minima.
(3) Neural network may be over-trained by using BP algorithms and obtain worse generalization performance. Thus, validation and suitable stopping methods are required in the cost function minimization procedure.
(4) Gradient-based learning is very time-consuming in most applications.

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

3.2. *Proposed minimum norm least-squares (LS) solution of SLFNs*

As rigorously proved in Theorems 2.1 and 2.2, unlike the traditional function approximation theories which require to adjust input weights and hidden layer biases, input weights and hidden layer biases can be randomly assigned if only the activation function is infinitely differentiable. It is very interesting and surprising that unlike the most common understanding that all the parameters of SLFNs need to be adjusted, the input weights $\mathbf{w}_i$ and the hidden layer biases b_i are in fact not necessarily tuned and the hidden layer output matrix $\mathbf{H}$ can actually remain unchanged once random values have been assigned to these parameters in the beginning of learning. For fixed input weights $\mathbf{w}_i$ and the hidden layer biases b_i, seen from Eq. (9), to train an SLFN is simply equivalent to finding a least-squares solution $\hat{\beta}$ of the linear system $\mathbf{H}\beta = \mathbf{T}$:

$$\begin{aligned}&\|\mathbf{H}(\mathbf{w}_1,\ldots,\mathbf{w}_{\tilde{N}},b_1,\ldots,b_{\tilde{N}})\hat{\beta}-\mathbf{T}\|\\ &\quad=\min_{\beta}\|\mathbf{H}(\mathbf{w}_1,\ldots,\mathbf{w}_{\tilde{N}},b_1,\ldots,b_{\tilde{N}})\beta-\mathbf{T}\|.\end{aligned} \tag{12}$$

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

However, in most cases the number of hidden nodes is much less than the number of distinct training samples, $\tilde{N} \ll N$, $\mathbf{H}$ is a nonsquare matrix and there may not exist $\mathbf{w}_i, b_i, \beta_i$ $(i=1,\ldots,\tilde{N})$ such that $\mathbf{H}\beta=\mathbf{T}$. According to Theorem 5.1 in the Appendix, the smallest norm least-squares solution of the above linear system is

$$\hat{\beta}=\mathbf{H}^{\dagger}\mathbf{T}, \tag{13}$$

where $\mathbf{H}^{\dagger}$ is the *Moore–Penrose generalized inverse* of matrix $\mathbf{H}$ [22,19].

Remark 1. As discussed in the Appendix, we have the following important properties:

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

$$\|\mathbf{H}\hat{\beta}-\mathbf{T}\|=\|\mathbf{H}\mathbf{H}^{\dagger}\mathbf{T}-\mathbf{T}\|=\min_{\beta}\|\mathbf{H}\beta-\mathbf{T}\|. \tag{14}$$

Although almost all learning algorithms wish to reach the minimum training error, however, most of them cannot reach it because of local minimum or infinite training iteration is usually not allowed in applications.
(2) *Smallest norm of weights*. Further, the special solution $\hat{\beta}=\mathbf{H}^{\dagger}\mathbf{T}$ has the smallest norm among all the least-squares solutions of $\mathbf{H}\beta=\mathbf{T}$:

$$\begin{aligned}&\|\hat{\beta}\|=\|\mathbf{H}^{\dagger}\mathbf{T}\|\leqslant\|\beta\|,\\ &\quad\forall\beta\in\left\{\beta:\|\mathbf{H}\beta-\mathbf{T}\|\leqslant\|\mathbf{H}\mathbf{z}-\mathbf{T}\|,\forall\mathbf{z}\in\mathbf{R}^{\tilde{N}\times N}\right\}.\end{aligned} \tag{15}$$

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

3.3. *Proposed learning algorithm for SLFNs*

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

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

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

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

Step 3: Calculate the output weight β

$$\beta=\mathbf{H}^{\dagger}\mathbf{T}, \tag{16}$$

where $\mathbf{T}=[\mathbf{t}_1,\ldots,\mathbf{t}_N]^{\mathrm{T}}$.

Remark 2. As shown in Theorem 2.1, in theory this algorithm works for any infinitely differential activation function $g(x)$. Such activation functions include the sigmoidal functions as well as the radial basis, sine, cosine, exponential, and many nonregular functions as shown in Huang and Babri [11]. According to Theorem 2.2, the upper bound of the required number of hidden nodes is the number of distinct training samples, that is $\tilde{N}\leqslant N$.

Remark 3. Several works [11,23,10,9,4] have shown that SLFNs with N hidden nodes can exactly learn N distinct observations. Tamura and Tateishi [23] and Huang [10,9] rigorously prove that SLFNs (with N hidden nodes) with *randomly chosen sigmoidal hidden nodes* (with both input weights and hidden biases randomly generated) can exactly learn N distinct observations. Huang et al. [11,9] also rigorously proves that if input weights and hidden biases are allowed to be tuned (as done in most traditional implementations) SLFNs with at most N hidden nodes and with almost any nonlinear activation function can exactly learn N distinct observations and these activation functions include differentiable and nondifferentiable functions, continuous and noncontinuous functions, etc.

This paper rigorously proves that for any infinitely differentiable activation function SLFNs with N hidden nodes can learn N distinct samples exactly and SLFNs may require less than N hidden nodes if learning error is allowed. Different from previous works [11,23,10,9,4] and the ELM algorithm introduced in this paper, Ferrari and Stengel [4] shows that SLFNs with N *sigmoidal* hidden nodes and *with input weights randomly generated but hidden biases appropriately tuned* can exactly learn N distinct observations. Hidden nodes are not randomly generated in the work done by Ferrari and Stengel [4], although the input weights are randomly generated, the hidden biases need to be determined based on the input weights and input training data (cf. Eq. (18) of [4]).

Remark 4. Modular networks have also been suggested in several works [23,10,16,9,4], which partition the training samples into L subsets each learned by an SLFN separately. Suppose that the number of hidden nodes in ith SLFN is s_i. For the methods proposed by Huang [10,16,9], since random hidden nodes (with randomly generated input weights and hidden layer biases) are used in each SLFN these SLFNs can actually share common hidden nodes. That means, the ith hidden node of the first SLFN can also work as the ith hidden node of the rest SLFNs and the total number of hidden nodes required in these L SLFNs is still $\max_i(s_i)$. Although Ferrari and Stengel [4] also randomly generates input weights for these sub-SLFNs but the hidden biases of these sub-SLFNs need to tuned based on the input weights and input training data. And thus, the hidden biases of these SLFNs are different, which means these SLFNs cannot share the common hidden nodes and the total number of hidden nodes required in modular network implementation of Ferrari and Stengel [4] is $\sum_{i=1}^{L} s_i \gg \max_i(s_i)$. One can refer to Tamura and Tateishi [23] and Huang [10,16,9] for details of modular network implementation.

Remark 5. Several methods can be used to calculate the Moore–Penrose generalized inverse of $\mathbf{H}$. These methods may include but are not limited to orthogonal projection, orthogonalization method, iterative method, and singular value decomposition (SVD) [18]. The orthogonalization method and iterative method have their limitations since searching and iteration are used which we wish to avoid in ELM. The orthogonal project method can be used when $\mathbf{H}^{\mathrm{T}}\mathbf{H}$ is nonsingular and $\mathbf{H}^{\dagger} = (\mathbf{H}^{\mathrm{T}}\mathbf{H})^{-1}\mathbf{H}^{\mathrm{T}}$ which is also used in Ferrari and Stengel [4]. However, $\mathbf{H}^{\mathrm{T}}\mathbf{H}$ may not always be nonsingular or may tend to be singular in some applications and thus orthogonal projection method may not perform well in all applications. The SVD can be generally used to calculate the Moore–Penrose generalized inverse of $\mathbf{H}$ in all cases.

4. Performance evaluation

In this section, the performance of the proposed ELM learning algorithm[3] is compared with the popular algorithms of feedforward neural networks like the conventional BP algorithm and support vector machines (SVMs) on quite a few benchmark real problems in the function approximation and classification areas. All the simulations for the BP and ELM algorithms are carried out in MATLAB 6.5 environment running in a Pentium 4, 1.9 GHZ CPU. Although there are many variants of BP algorithm, a faster BP algorithm called Levenberg–Marquardt algorithm is used in our simulations. As mentioned in the HELP of MATLAB package and tested on many benchmark applications among all traditional BP learning algorithms, the Levenberg–Marquardt algorithm appears to be the fastest method for training moderate-sized feedforward neural networks (up to several hundred weights). It has a very efficient implementation of Levenberg–Marquardt algorithm provided by MATLAB package, which has been used in our simulations for BP. The simulations for SVM are carried out using compiled C-coded SVM packages: LIBSVM[4] running in the same PC. The kernel function used in SVM is radial basis function whereas the activation function used in our proposed algorithms is a simple sigmoidal function $g(x) = 1/(1 + \exp(-x))$. In our experiments, all the inputs (attributes) have been normalized into the range $[0, 1]$ while the outputs (targets) have been normalized into $[-1, 1]$. As seen from ELM algorithm, the learning time of ELM is mainly spent on calculating the Moore–Penrose generalized inverse $\mathbf{H}^{\dagger}$ of the hidden layer output matrix $\mathbf{H}$.

4.1. *Benchmarking with regression problems*

4.1.1. *Artificial case: approximation of 'SinC' function with noise*

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

$$y(x) = \begin{cases} \sin(x)/x, & x \neq 0, \\ 1, & x = 0. \end{cases} \tag{17}$$

A training set (x_i, y_i) and testing set (x_i, y_i) with 5000 data, respectively, are created where x_i's are uniformly randomly distributed on the interval $(-10, 10)$. In order to make the regression problem 'real', large uniform noise distributed in $[-0.2, 0.2]$ has been added to all the training samples while testing data remain noise-free.

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

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

Table 1
Performance comparison for learning noise free function: SinC

Algorithms	Time (s)		Training		Testing		No of SVs/nodes
	Training	Testing	RMS	Dev	RMS	Dev	
ELM	0.125	0.031	0.1148	0.0037	0.0097	0.0028	20
BP	21.26	0.032	0.1196	0.0042	0.0159	0.0041	20
SVR	1273.4	5.9087	0.1149	0.0007	0.0130	0.0012	2499.9

There are 20 hidden nodes assigned for our ELM algorithm and BP algorithm. 50 trials have been conducted for all the algorithms and the average results and standard deviations (Dev) are shown in Table 1. It can be seen from Table 1 that ELM learning algorithm spent 0.125 s CPU time obtaining the testing root mean square error (RMSE) 0.0097, however, it takes 21.26 s CPU time for BP algorithm to reach a much higher testing error 0.0159. The new ELM runs 170 times faster than the conventional BP algorithms. We also compare the performance of SVR and our ELM algorithm. The parameter C is tuned and set as $C = 100$ in SVR algorithm. Compared with SVR, the reduction for our ELM algorithm in CPU time is also above 10,000 times, even though the fact that C executable may be faster than MATLAB environment has not be taken into account. Since the number of support vectors obtained by SVR is much larger than the hidden nodes required by ELM, the testing time spent for the obtained SVR is 190 times longer than the testing time for ELM, meaning that after trained the SLFN may response to new external unknown stimuli much faster than SVM in real deployment.

Fig. 1 shows the true and the approximated function of the ELM learning algorithm. Fig. 2 shows the true and the approximated function of the BP and SVM learning algorithms.

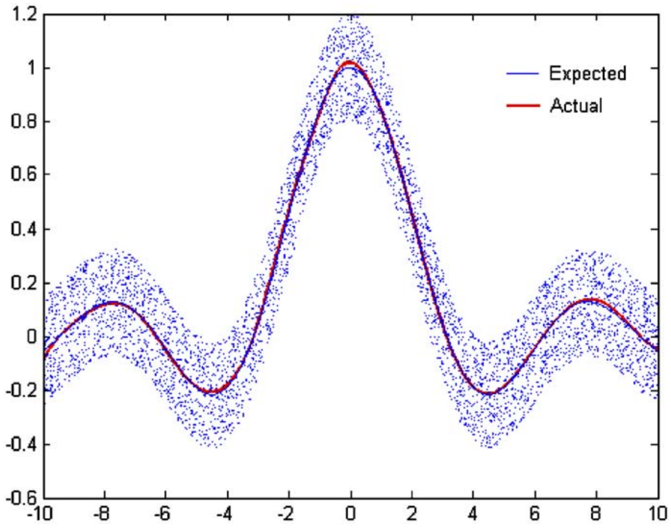


Fig. 1. Outputs of the ELM learning algorithm.

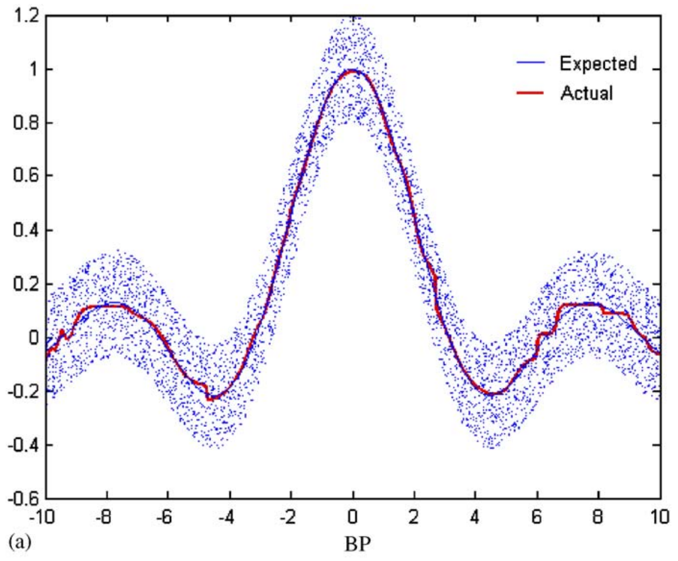
4.1.2. Real-world regression problems

The performance of ELM, BP and SVR are compared on 13 real-world benchmark data sets[5] covering various fields. The specifications of the data sets are listed in Table 2. As in the real applications, the distributions of these data set are unknown and most of them are not noisy-free. For each case, the training data set and testing data set are randomly generated from its whole data set before each trial of simulation.

For BP and ELM, the number of hidden nodes are gradually increased by an interval of 5 and the nearly optimal number of nodes for BP and ELM are then selected based on cross-validation method. In addition, over-stopping criteria is used for BP as well. Average results of 50 trials of simulations for each fixed size of SLFN are obtained and then finally the best performance obtained by BP and ELM are reported in this paper. As proposed by Hsu and Lin [8], for each problem, we estimate the generalized accuracy using different combination of cost parameters C and kernel parameters γ: $C = [2^{12}, 2^{11}, \ldots, 2^{-1}, 2^{-2}]$ and $\gamma = [2^4, 2^3, \ldots, 2^{-9}, 2^{-10}]$. Therefore, for each problem we try $15 \times 15 = 225$ combinations of parameters (C, γ) for SVR. Average results of 50 trials of simulations with each combination of (C, γ) are obtained and the best performance obtained by SVR are shown in this paper as well.

As observed from Tables 3 and 4, general speaking, ELM and SVR obtain similar generalization performance, which is slightly higher than BP's in many cases. If the difference of the two testing RMSE obtained by two algorithms is larger than 0.005 for a case, the winner's testing RMSE will be shown in boldface in Tables 3 and 4. As observed from Table 5, ELM needs more hidden nodes than BP but it is more compact than SVR in most cases. The advantage of the ELM on training time is quite obvious. As shown in Table 6, ELM obtains the fastest learning speed in all cases. ELM learns up to hundreds of times faster than BP. However, out of these three algorithms, BP obtains the shortest testing time (response time to unknown data set for testing) in all cases because BP usually provides the most compact network architectures. Table 7 shows the average standard deviations of training and testing RMSE of the BP, SVR and ELM.

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



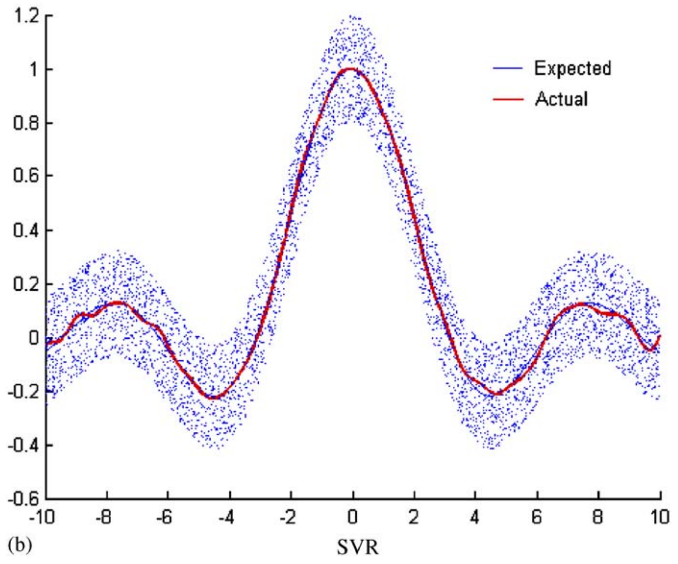


Fig. 2. Outputs of the BP and SVR learning algorithms.

Table 2
Specification of real-world regression cases

Data sets	# Observations		# Attributes	
	Training	Testing	Continuous	Nominal
Abalone	2000	2177	7	1
Delta ailerons	3000	4129	6	0
Delta elevators	4000	5517	6	0
Computer activity	4000	4192	8	0
Census (house8L)	10,000	12,784	8	0
Auto price	80	79	14	1
Triazines	100	86	60	0
Machine CPU	100	109	6	0
Servo	80	87	0	4
Breast cancer	100	94	32	0
Bank	4500	3692	8	0
California housing	8000	12,460	8	0
Stocks	450	500	10	0

Table 3
Comparison of training and testing RMSE of BP and ELM

Data sets	BP		ELM	
	Training	Testing	Training	Testing
Abalone	0.0785	0.0874	0.0803	**0.0824**
Delta ailerons	0.0409	0.0481	0.0423	**0.0431**
Delta elevators	0.0544	0.0592	0.0550	0.0568
Computer activity	0.0273	0.0409	0.0316	0.0382
Census (house8L)	0.0596	0.0685	0.0624	0.0660
Auto price	0.0443	0.1157	0.0754	**0.0994**
Triazines	0.1438	0.2197	0.1897	**0.2002**
Machine CPU	0.0352	0.0826	0.0332	**0.0539**
Servo	0.0794	0.1276	0.0707	**0.1196**
Breast cancer	0.2788	0.3155	0.2470	**0.2679**
Bank domains	0.0342	0.0379	0.0406	0.0366
California housing	0.1046	0.1285	0.1217	0.1267
Stocks domain	0.0179	0.0358	0.0251	0.0348

Table 4
Comparison of training and testing RMSE of SVR and ELM

Data sets	SVR		ELM	
	Training	Testing	Training	Testing
Abalone	0.0759	0.0784	0.0803	0.0824
Delta ailerons	0.0418	0.0429	0.04230	0.0431
Delta elevators	0.0534	0.0540	0.0545	0.0568
Computer activity	0.0464	0.0470	0.0316	**0.0382**
Census (house8L)	0.0718	0.0746	0.0624	**0.0660**
Auto price	0.0652	**0.0937**	0.0754	0.0994
Triazines	0.1432	**0.1829**	0.1897	0.2002
Machine CPU	0.0574	0.0811	0.0332	**0.0539**
Servo	0.0840	0.1177	0.0707	0.1196
Breast cancer	0.2278	0.2643	0.2470	0.2679
Bank domains	0.0454	0.0467	0.0406	**0.0366**
California housing	0.1089	**0.1180**	0.1217	0.1267
Stocks domain	0.0503	0.0518	0.0251	**0.0348**

Table 5
Comparison of network complexity of BP, SVR and ELM

Data sets	BP	SVR		ELM
	# nodes	(C, γ)	# SVs	# nodes
Abalone	10	$(2^4, 2^{-6})$	309.84	25
Delta ailerons	10	$(2^3, 2^{-3})$	82.44	45
Delta elevators	5	$(2^0, 2^{-2})$	260.38	125
Computer activity	45	$(2^5, 2^{-5})$	64.2	125
Census (house8L)	10	$(2^1, 2^{-1})$	810.24	160
Auto price	5	$(2^8, 2^{-5})$	21.25	15
Triazines	5	$(2^{-1}, 2^{-9})$	48.42	10
Machine CPU	10	$(2^6, 2^{-4})$	7.8	10
Servo	10	$(2^2, 2^{-2})$	22.375	30
Breast cancer	5	$(2^{-1}, 2^{-4})$	74.3	10
Bank domains	20	$(2^{10}, 2^{-2})$	129.22	190
California housing	10	$(2^3, 2^1)$	2189.2	80
Stocks domain	20	$(2^3, 2^{-9})$	19.94	110

Table 6
Comparison of training and testing time of BP, SVR and ELM

Data sets	BP[a] (s)		SVR[b] (s)		ELM[a] (s)	
	Training	Testing	Training	Testing	Training	Testing
Abalone	1.7562	0.0063	1.6123	0.3223	0.0125	0.0297
Delta ailerons	2.7525	0.0156	0.6726	0.2231	0.0591	0.0627
Delta elevators	1.1938	0.0125	1.121	0.5868	0.2812	0.2047
Computer activity	67.44	0.0688	1.0149	0.3027	0.2951	0.172
Census (house8L)	8.0647	0.0457	11.251	4.1469	1.0795	0.6298
Auto price	0.2456	$<10^{-4}$	0.0042	0.0465	0.0016	$<10^{-4}$
Triazines	0.5484	$<10^{-4}$	0.0086	0.0540	$<10^{-4}$	$<10^{-4}$
Machine CPU	0.2354	$<10^{-4}$	0.0018	0.0421	0.0015	$<10^{-4}$
Servo	0.2447	$<10^{-4}$	0.0045	0.0394	$<10^{-4}$	$<10^{-4}$
Breast cancer	0.3856	$<10^{-4}$	0.0064	0.0591	$<10^{-4}$	$<10^{-4}$
Bank domains	7.506	0.0466	1.6084	0.3021	0.6434	0.2205
California housing	6.532	0.0469	74.184	10.453	1.1177	0.3033
Stocks domain	1.0487	0.0063	0.0690	0.0637	0.0172	0.0297

[a]Run in MATLAB environment.
[b]Run in C executable environment.

Table 7
Comparison of the standard deviation of training and testing RMSE of BP, SVR and ELM

Data sets	BP		SVR		ELM	
	Training	Testing	Training	Testing	Training	Testing
Abalone	0.0011	0.0034	0.0015	0.0013	0.0049	0.0058
Delta ailerons	0.0015	0.0015	0.0012	0.0010	0.0030	0.0035
Delta elevators	0.0007	0.0003	0.0006	0.0005	0.0028	0.0029
Computer activity	0.0007	0.0007	0.0015	0.0016	0.0005	0.0033
Census (house8L)	0.0011	0.0050	0.0013	0.0013	0.001	0.0017
Auto price	0.0405	0.0231	0.0025	0.0040	0.0119	0.0119
Triazines	0.0656	0.0531	0.0152	0.0163	0.0212	0.0209
Machine CPU	0.0192	0.0715	0.0083	0.0180	0.0060	0.0156
Servo	0.0313	0.0475	0.0406	0.0185	0.0121	0.0113
Breast cancer	0.1529	0.0962	0.0115	0.0151	0.0121	0.0167
Bank domains	0.0006	0.0004	0.0005	0.0008	0.0006	0.0009
California housing	0.0045	0.0026	0.0012	0.0011	0.0021	0.0033
Stocks domain	0.0012	0.0022	0.0016	0.0022	0.0011	0.0016

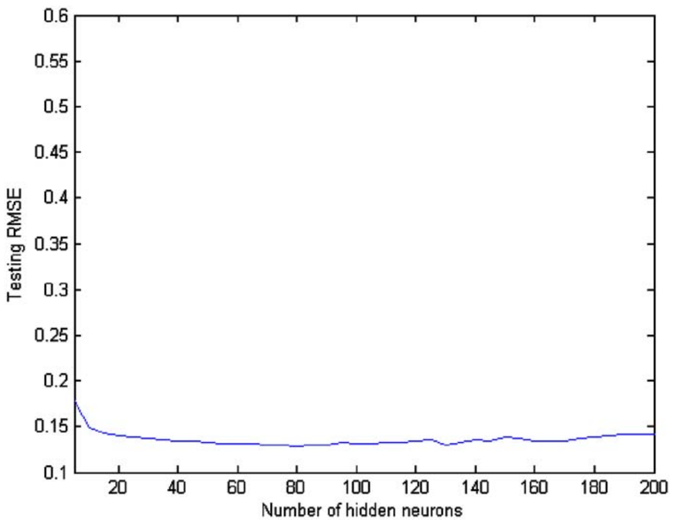


Fig. 3. The generalization performance of ELM is stable on a wide range of number of hidden nodes.

Fig. 3 shows the relationship between the generalization performance of ELM and its network size for the California Housing case. As observed from Fig. 3, the generalization performance of ELM is very stable on a wide range of number of hidden nodes although the generalization performance tends to become worse when too few or too many nodes are randomly generated. It is also true to other cases. Wang and Huang [24] has conducted a good comparison of the performance of ELM versus BP as a function of the number of hidden nodes.

4.2. *Benchmarking with small and medium real classification applications*

4.2.1. *Medical diagnosis application: diabetes*

The performance comparison of the new proposed ELM algorithm and many other popular algorithms has been conducted for a real medical diagnosis

Table 8
Performance comparison in real medical diagnosis application: diabetes

Algorithms	Time (s)		Success rate (%)				# SVs/nodes
	Training	Testing	Training		Testing		
			Rate	Dev	Rate	Dev	
ELM	0.0118	0.0031	78.68	1.18	77.57	2.85	20
BP	3.0116	0.0035	86.63	1.7	74.73	3.2	20
SVM	0.1860	0.0673	78.76	0.91	77.31	2.35	317.16

problem: Diabetes,[6] using the "Pima Indians Diabetes Database" produced in the Applied Physics Laboratory, Johns Hopkins University, 1988. The diagnostic, binary-valued variable investigated is whether the patient shows signs of diabetes according to World Health Organization criteria (i.e., if the 2 h post-load plasma glucose was at least 200 mg/dl at any survey examination or if found during routine medical care). The database consists of 768 women over the age of 21 resident in Phoenix, Arizona. All examples belong to either positive or negative class. All the input values are within [0, 1]. For this problem, as usually done in the literature [20,21,5,25] 75% and 25% samples are randomly chosen for training and testing at each trial, respectively. The parameter C of SVM algorithm is tuned and set as $C = 10$ and the rest parameters are set as default.

Fifty trials have been conducted for all the algorithms and the average results are shown in Table 8. Seen from Table 8, in our simulations SVM can reach the testing rate 77.31% with 317.16 support vectors at average. Rätsch et al. [20] obtained a testing rate 76.50% for SVM which is slightly lower than the SVM result we obtained. However, the new ELM learning algorithm can achieve the average testing rate 77.57% with 20 nodes, which is obviously higher than all the results so far reported in the literature using various popular algorithms such as SVM [20], SAOCIF [21], Cascade-Correlation algorithm [21], bagging and boosting methods [5], C4.5 [5], and RBF [25] (cf. Table 9). BP algorithm performs very poor in our simulations for this case. It can also be seen that the ELM learning algorithm run around 300 times faster than BP, and 15 times faster than SVM for this small problem without considering that C executable environment may run much faster than MATLAB environment.

Table 9
Performance comparison in real medical diagnosis application: diabetes

Algorithms	Testing rate (%)
ELM	77.57
SVM [20]	76.50
SAOCIF [21]	77.32
Cascade-Correlation [21]	76.58
AdaBoost [5]	75.60
C4.5 [5]	71.60
RBF [25]	76.30
Heterogeneous RBF [25]	76.30

4.2.2. *Medium size classification applications*

The ELM performance has also been tested on the Banana database[7] and some other multiclass databases from the Statlog collection [2]: Landsat satellite image (SatImage), Image segmentation (Segment) and Shuttle landing control database. The information of the number of data, attributes and classes of these applications is listed in Table 10.

Table 10
Information of the benchmark problems: Landsat satellite image, Image segmentation, Shuttle landing control database, and Banana

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

For each trial of simulation of SatImage and Segment, the training data set and testing data set are randomly generated from their overall database. The training and testing data sets are fixed for Shuttle and Banana applications, as usually done in the literature. Although there are 40,000 training data in the original Banana database, during our simulations it is found that there are only 5200 distinct training samples and the redundancy of the training data has been removed. Fifty trials have been conducted for the ELM algorithm, and 5 trials for BP since it obviously takes very long time to train SLFNs using BP for these several cases. Seen from Table 11, obviously the ELM algorithm run much faster than BP learning algorithm by a factor up to 4200.

4.3. *Benchmarking with real-world very large complex applications*

We have also tested the performance of our ELM algorithm for very large complex applications such as forest cover-type prediction.

[6]ftp://ftp.ira.uka.de/pub/node/proben1.tar.gz.
[7]http://www.first.gmd.de/~raetsch/data.

Table 11
Performance comparison in more benchmark applications: Satimage, Segment, Shuttle, and Banana

Problems	Algorithms	Time (s)		Success rate (%)				# Nodes
		Training	Testing	Training		Testing		
				Rate	Dev	Rate	Dev	
Satellite image	ELM	14.92	0.34	93.52	1.46	89.04	1.50	500
	BP	12561	0.08	95.26	0.97	82.34	1.25	100
Image segment	ELM	1.40	0.07	97.35	0.32	95.01	0.78	200
	BP	4745.7	0.04	96.92	0.45	86.27	1.80	100
	RBF [25]	N/A	N/A	80.48	N/A	N/A	N/A	N/A
Shuttle	ELM	5.740	0.23	99.65	0.12	99.40	0.12	50
	BP	6132.2	0.22	99.77	0.10	99.27	0.13	50
Banana	ELM	2.19	20.06	92.36	0.17	91.57	0.25	100
	BP	6132.2	21.10	90.26	0.27	88.09	0.70	100

Table 12
Performance comparison of the ELM, BP and SVM learning algorithms in forest-type prediction application

Algorithms	Time (min)		Success rate (%)				# SVs/nodes
	Training	Testing	Training		Testing		
			Rate	Dev	Rate	Dev	
ELM	1.6148	0.7195	92.35	0.026	90.21	0.024	200
SLFN [3]	12	N/A	82.44	N/A	81.85	N/A	100
SVM	693.6000	347.7833	91.70	N/A	89.90	N/A	31,806

Forest cover-type prediction is an extremely large complex classification problem with seven classes. The forest cover type [2] for 30×30 m cells was obtained from US forest service (USFS) region 2 resource information system (RIS) data. There are 581,012 instances (observations) and 54 attributes per observation. In order to compare with the previous work [3], similarly it was modified as a binary classification problem where the goal was to separate class 2 from the other six classes. There are 100,000 training data and 481,012 testing data. The parameters for the SVM are $C = 10$ and $\gamma = 2$.

Fifty trials have been conducted for the ELM algorithm, and 1 trial for SVM since it takes very long time to train SVM for this large complex case.[8] Seen from Table 12, the proposed ELM learning algorithm obtains better generalization performance than SVM learning algorithm. However, the proposed ELM learning algorithm only spent 1.6 min on learning while SVM need to spend nearly 12 h on training. The learning speed has dramatically been increased 430 times. On the other hand, since the support vectors obtained by SVM is much larger than the required hidden nodes in ELM, the testing time spent SVMs for this large testing data set is more than 480 times than the ELM. It takes more than 5.5 h for the SVM to react to the 481,012 testing samples. However, it takes only less than 1 min for the obtained ELM to react to the testing samples. That means, after trained and deployed the ELM may react to new observations much faster than SVMs in such a real application. It should be noted that in order to obtain as good performance as possible for SVM, long time effort has been made to find the appropriate parameters for SVM. In fact the generalization performance of SVM we obtained in our simulation for this case is much higher than the one reported in the literature [3].

We did try to run the efficient optimal BP package provided by MATLAB for this application, however, it always ran out of memory and also showed that there are too many variables (parameters) required in the simulations, meaning that the application is too large to run in our ordinary PC. On the other hand, it has been reported [3] that SLFNs using gradient-based learning algorithm could only reach a much lower testing accuracy (81.85% only) than our ELM (90.21%).

5. Discussions and conclusions

This paper proposed a simple and efficient learning algorithm for **s**ingle-hidden **l**ayer **f**eedforward neural **net**works (SLFNs) called **e**xtreme **l**earning **m**achine (ELM), which has also been rigorously proved in this paper. The proposed ELM has several interesting and significant

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

features different from traditional popular gradient-based learning algorithms for feedforward neural networks:

(1) The learning speed of ELM is extremely fast. In our simulations, the learning phase of ELM can be completed in seconds or less than seconds for many applications. Previously, it seems that there exists a virtual speed barrier which most (if not all) classic learning algorithms cannot break through and it is not unusual to take very long time to train a feedforward network using classic learning algorithms even for simple applications.
(2) The proposed ELM has better generalization performance than the gradient-based learning such as back-propagation in most cases.
(3) The traditional classic gradient-based learning algorithms may face several issues like local minima, improper learning rate and overfitting, etc. In order to avoid these issues, some methods such as weight decay and early stopping methods may need to be used often in these classical learning algorithms. The ELM tends to reach the solutions straightforward without such trivial issues. The ELM learning algorithm looks much simpler than most learning algorithms for feedforward neural networks.
(4) Unlike the traditional classic gradient-based learning algorithms which only work for differentiable activation functions, as easily observed the ELM learning algorithm could be used to train SLFNs with many nondifferentiable activation functions [15]. The performance comparison of SLFNs with different activation functions is beyond the scope of this paper and shall be investigated in the future work. It may also be interesting to further compare ELM with sigmoidal activation functions and ELM with RBF activation functions [13,14] in the future work.

It should be worth pointing out that gradient-based learning algorithms like back-propagation can be used for feedforward neural networks which have more than one hidden layers while the proposed ELM algorithm at its present form is still only valid for single-hidden layer feedforward networks (SLFNs). Fortunately, it has been found in theory that SLFNs can approximate any continuous function and implement any classification application [12]. Thus, reasonably speaking the proposed ELM algorithm can generally be efficiently used in many applications. As observed from most cases, BP achieves shorter testing time than ELM.

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

This paper has demonstrated that ELM can be used efficiently in many applications, however, two more interesting aspects are still open: the universal approximation capability[9] of ELM and the performance of ELM in sparse high-dimensional applications, which are currently under our investigation.

Appendix A

In this section, the Moore–Penrose generalized inverse is introduced. We also consider in this section the minimum norm least-squares solution of a general linear system $\mathbf{Ax}=\mathbf{y}$ in Euclidean space, where $\mathbf{A}\in\mathbf{R}^{m\times n}$ and $\mathbf{y}\in\mathbf{R}^{m}$. As shown in [23,10], the SLFNs are actually a linear system if the input weights and the hidden layer biases can be chosen randomly.

A.1. Moore–Penrose generalized inverse

The resolution of a general linear system $\mathbf{Ax}=\mathbf{y}$, where $\mathbf{A}$ may be singular and may even not be square, can be made very simple by the use of the Moore–Penrose generalized inverse [22].

Definition 5.1 (*Serre [22], Rao and Mitra [19]*). A matrix $\mathbf{G}$ of order $n\times m$ is the Moore–Penrose generalized inverse of matrix $\mathbf{A}$ of order $m\times n$, if

$$\mathbf{AGA}=\mathbf{A},\quad \mathbf{GAG}=\mathbf{G},\quad (\mathbf{AG})^{\mathrm{T}}=\mathbf{AG},\quad (\mathbf{GA})^{\mathrm{T}}=\mathbf{GA}. \tag{18}$$

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

A.2. Minimum norm least-squares solution of general linear system

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

$$\|\mathbf{A}\hat{\mathbf{x}}-\mathbf{y}\|=\min_{\mathbf{x}}\|\mathbf{Ax}-\mathbf{y}\|, \tag{19}$$

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

[9]Readers can refer to Huang et al. [26] for the universal approximation capability of ELM.

Definition 5.2. $\mathbf{x}_0 \in \mathbf{R}^n$ is said to be a minimum norm least-squares solution of a linear system $\mathbf{Ax} = \mathbf{y}$ if for any $\mathbf{y} \in \mathbf{R}^m$

$$\|\mathbf{x}_0\| \leqslant \|\mathbf{x}\|, \quad \forall \mathbf{x} \in \{\mathbf{x} : \|\mathbf{Ax} - \mathbf{y}\| \leqslant \|\mathbf{Az} - \mathbf{y}\|, \forall \mathbf{z} \in \mathbf{R}^n\}. \tag{20}$$

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

Theorem 5.1 (*p. 147 of Serre [22], p. 51 of Rao and Mitra [19]*). *Let there exist a matrix* $\mathbf{G}$ *such that* $\mathbf{Gy}$ *is a minimum norm least-squares solution of a linear system* $\mathbf{Ax} = \mathbf{y}$. *Then it is necessary and sufficient that* $\mathbf{G} = \mathbf{A}^\dagger$, *the* Moore–Penrose generalized inverse *of matrix* $\mathbf{A}$.

Remark. Seen from Theorem 5.1, we can have the following properties key to our proposed ELM learning algorithm:

(1) The special solution $\mathbf{x}_0 = \mathbf{A}^\dagger\mathbf{y}$ is one of the least-squares solutions of a general linear system $\mathbf{Ax} = \mathbf{y}$:

$$\|\mathbf{Ax}_0 - \mathbf{y}\| = \|\mathbf{AA}^\dagger\mathbf{y} - \mathbf{y}\| = \min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{y}\|. \tag{21}$$

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

$$\|\mathbf{x}_0\| = \|\mathbf{A}^\dagger\mathbf{y}\| \leqslant \|\mathbf{x}\|, \quad \forall \mathbf{x} \in \{\mathbf{x} : \|\mathbf{Ax} - \mathbf{y}\| \leqslant \|\mathbf{Az} - \mathbf{y}\|, \forall \mathbf{z} \in \mathbf{R}^n\}. \tag{22}$$

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

References

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

[2] 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.

[3] R. Collobert, S. Bengio, Y. Bengio, A parallel mixtures of SVMs for very large scale problems, Neural Comput. 14 (2002) 1105–1114.

[4] S. Ferrari, R.F. Stengel, Smooth function approximation using neural networks, IEEE Trans. Neural Networks 16 (1) (2005) 24–38.

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

[6] S. Haykin, Neural Networks: A Comprehensive Foundation, Prentice-Hall, New Jersey, 1999.

[7] K. Hornik, Approximation capabilities of multilayer feedforward networks, Neural Networks 4 (1991) 251–257.

[8] C.-W. Hsu, C.-J. Lin, A comparison of methods for multiclass support vector machines, IEEE Trans. Neural Networks 13 (2) (2002) 415–425.

[9] G.-B. Huang, Learning capability of neural networks, Ph.D. Thesis, Nanyang Technological University, Singapore, 1998.

[10] G.-B. Huang, Learning capability and storage capacity of two-hidden-layer feedforward networks, IEEE Trans. Neural Networks 14 (2) (2003) 274–281.

[11] G.-B. Huang, H.A. Babri, Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions, IEEE Trans. Neural Networks 9 (1) (1998) 224–229.

[12] G.-B. Huang, Y.-Q. Chen, H.A. Babri, Classification ability of single hidden layer feedforward neural networks, IEEE Trans. Neural Networks 11 (3) (2000) 799–801.

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

[14] G.-B. Huang, C.-K. Siew, Extreme learning machine with randomly assigned RBF kernels, Int. J. Inf. Technol. 11 (1) (2005).

[15] G.-B. Huang, Q.-Y. Zhu, K.Z. Mao, C.-K. Siew, P. Saratchandran, N. Sundararajan, Can threshold networks be trained directly?, IEEE Trans. Circuits Syst. II 53 (3) (2006) 187–191.

[16] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Real-time learning capability of neural networks, Technical Report ICIS/45/2003, School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, April 2003.

[17] M. Leshno, V.Y. Lin, A. Pinkus, S. Schocken, Multilayer feedforward networks with a nonpolynomial activation function can approximate any function, Neural Networks 6 (1993) 861–867.

[18] J.M. Ortega, Matrix Theory, Plenum Press, New York, London, 1987.

[19] C.R. Rao, S.K. Mitra, Generalized Inverse of Matrices and its Applications, Wiley, New York, 1971.

[20] 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.

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

[22] D. Serre, Matrices: Theory and Applications, Springer, New York, 2002.

[23] S. Tamura, M. Tateishi, Capabilities of a four-layered feedforward neural network: four layers versus three, IEEE Trans. Neural Networks 8 (2) (1997) 251–255.

[24] D. Wang, G.-B. Huang, Protein sequence classification using extreme learning machine, in: Proceedings of International Joint Conference on Neural Networks (IJCNN2005), Montreal, Canada, 31 July–4 August, 2005.

[25] 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.

[26] G.-B. Huang, L. Chen, C.-K. Siew, Universal approximation using incremental networks with random hidden computation nodes, IEEE Trans. Neural Networks 17 (4) 2006.

Guang-Bin Huang received the B.Sc. degree in applied mathematics and M.Eng. degree in computer engineering from Northeastern University, PR China, in 1991 and 1994, respectively, and Ph.D. degree in electrical engineering from Nanyang Technological University, Singapore in 1999. During undergraduate period, he also concurrently studied in Wireless Communication Department of Northeastern University, PR China.

From June 1998 to May 2001, he worked as Research Fellow in Singapore Institute of Manufacturing Technology (formerly known as Gintic Institute of Manufacturing Technology) where he has led/implemented several key industrial projects. From May 2001, he has been working as an Assistant Professor in the Information

Communication Institute of Singapore (ICIS), School of Electrical and Electronic Engineering, Nanyang Technological University. His current research interests include machine learning, computational intelligence, neural networks, and bioinformatics. He serves as an Associate Editor of *Neurocomputing*. He is a senior member of IEEE.

Qin-Yu Zhu received the B.Eng. degree from Shanghai Jiao Tong University, China in 2001. He is currently a Ph.D. student with Information Communication Institute of Singapore, School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore. His research interests include neural networks and evolutionary algorithms. He has published a number of papers in international journals and conferences.

Chee-Kheong Siew is currently an associate professor in the School of EEE, Nanyang Technological University (NTU). From 1995 to 2005, he served as the Head of Information Communication Institute of Singapore (ICIS) after he managed the transfer of ICIS to NTU and rebuilt the institute in the university environment. He obtained his B.Eng. in Electrical Engineering from University of Singapore in 1979 and M.Sc. in Communication Engineering, Imperial College in 1987. After six years in the industry, he joined NTU in 1986 and was appointed as the Head of the Institute in 1996. His current research interests include neural networks, packet scheduling, traffic shaping, admission control, service curves and admission control, QoS framework, congestion control and multipath routing. He is a member of IEEE.