Available online at www.sciencedirect.com

ScienceDirect

NEUROCOMPUTING

Neurocomputing 70 (2007) 3056–3062

www.elsevier.com/locate/neucom

Letters

Convex incremental extreme learning machine

Guang-Bin Huang[a,*], Lei Chen[a,b]

[a]School of Electrical and Electronic Engineering, Nanyang Technological University, Nanyang Avenue, Singapore 639798, Singapore
[b]School of Computing, National University of Singapore, 3 Science Drive 2, Singapore 117543, Singapore

Received 12 June 2006; received in revised form 15 February 2007; accepted 18 February 2007
Communicated by S. Hochreiter
Available online 19 March 2007

Abstract

Unlike the conventional neural network theories and implementations, Huang et al. [Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Transactions on Neural Networks 17(4) (2006) 879–892] have recently 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. This paper shows that while retaining the same simplicity, the convergence rate of I-ELM can 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, we show that given a type of piecewise continuous computational hidden nodes (possibly not neural alike nodes), if SLFNs $f_n(\mathbf{x}) = \sum_{i=1}^{n} \beta_i G(\mathbf{x}, \mathbf{a}_i, b_i)$ can 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.) can actually be randomly generated according to any continuous sampling distribution. In theory, the parameters of these SLFNs can be analytically determined by ELM instead of being tuned.
© 2007 Elsevier B.V. All rights reserved.

Keywords: Generalized feedforward networks; Incremental extreme learning machine; Convergence rate; Random hidden nodes; Convex optimization

1. Introduction

Neural networks have been successfully applied in many applications. Out of many kinds of neural networks single-hidden-layer feedforward networks (SLFNs) have been investigated more thoroughly. SLFN functions with n hidden nodes can be represented by

$$f_n(\mathbf{x}) = \sum_{i=1}^{n} \beta_i g_i(\mathbf{x}) = \sum_{i=1}^{n} \beta_i G(\mathbf{x}, \mathbf{a}_i, b_i), \quad \mathbf{x} \in \mathbf{R}^d, \ \beta_i \in R, \tag{1}$$

where g_i or $G(\mathbf{x}, \mathbf{a}_i, b_i)$ denotes the hidden node output function (with the hidden node parameters $(\mathbf{a}_i, b_i)$) and β_i is the weight of the connection between the ith hidden node and the output node. Seen from the viewpoint of network architectures, two types of hidden nodes have been mainly used in SLFNs:

(1) *Additive hidden nodes.* For additive nodes with activation function g, g_i is defined as

$$g_i(\mathbf{x}) = G(\mathbf{x}, \mathbf{a}_i, b_i) = g(\mathbf{a}_i \cdot \mathbf{x} + b_i), \quad \mathbf{a}_i \in \mathbf{R}^d, \ b_i \in R, \tag{2}$$

where $\mathbf{a}_i$ is the weight vector connecting the input layer to the ith hidden node and b_i is the bias of the ith hidden node. $\mathbf{a}_i \cdot \mathbf{x}$ denotes the inner product of vectors $\mathbf{a}_i$ and $\mathbf{x}$ in $\mathbf{R}^d$.

(2) *RBF* (*radial basis function*) *hidden nodes.* The RBF network can be considered a specific SLFN

*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/.

0925-2312/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.neucom.2007.02.009

which applies RBF nodes in its hidden layer. The output of each RBF node is some radially symmetric function of the distance between the input and the center. For RBF nodes with activation function g, g_i is defined as

$$g_i(\mathbf{x}) = G(\mathbf{x}, \mathbf{a}_i, b_i) = g(b_i\|\mathbf{x} - \mathbf{a}_i\|), \quad \mathbf{a}_i \in \mathbf{R}^d, \ b_i \in R^+, \tag{3}$$

where $\mathbf{a}_i$ and b_i are the center and impact factor of the ith RBF node, β_i the weight connecting the ith RBF hidden node to the output node. R^+ indicates the set of all non-negative real value.

According to the conventional neural network theories [1,9,21,28], SLFNs with additive or RBF hidden nodes are universal approximators when all the parameters of the networks are adjustable. However, as observed in most neural network implementations tuning all the parameters of the networks may render learning complicated and inefficient, and it may be difficult to directly train networks with non-differential activation functions such as threshold networks. Unlike conventional neural network theories, Huang et al. [10] recently rigorously prove in an incremental constructive method that in order to let SLFNs work as universal approximators one may simply randomly choose hidden nodes (with random values for both the hidden biases and input weights linking the input layer to the hidden layer or the random values for centers and impact factors of RBF hidden nodes) and then only need to analytically calculate (instead of tuning) the output weights linking the hidden layer and the output layer. In such SLFNs implementations, the activation functions for additive nodes can be any bounded nonconstant piecewise continuous functions $g : R \to R$ and the activation functions for RBF nodes can be any integrable piecewise continuous functions $g : R \to R$ and $\int_R g(x)\,\mathrm{d}x \neq 0$. The resulting method referred to as incremental extreme learning machine (I-ELM) [10] randomly adds nodes to the hidden layer one by one and freezes the output weights of the existing hidden nodes when a new hidden node is added. I-ELM is not only efficient for SLFN with continuous (including differentiable) activation functions but also for SLFNs with piecewise continuous (such as threshold) activation functions. ELM with fixed network architectures has also been investigated in several earlier works [12–16,22,23].

In this paper Barron's convex optimization learning method [1] is incorporated into I-ELM and the improved method is referred to as convex I-ELM (CI-ELM). Different from I-ELM, CI-ELM recalculates the output weights of the existing hidden nodes after a new hidden node is added. Different from Barron's work [1], CI-ELM adds randomly generated hidden nodes. The proposed algorithm (CI-ELM) can achieve faster convergence rates and more compact network architectures than I-ELM while retaining the I-ELM's simplicity and efficiency.

2. Proposed CI-ELM

For the sake of readability, in this paper we apply the same notations used in our earlier work [10]. Typically, let $L^2(X)$ be a space of functions f on an input space X which is a compact subset in the Euclidean space $\mathbf{R}^d$ such that $\int_X |f(\mathbf{x})|^2\,\mathrm{d}\mathbf{x} < \infty$. Let $L^2(\mathbf{R}^d)$ denoted by L^2. The inner product $\langle u, v\rangle$ is denoted by $\langle u, v\rangle = \int_X u(\mathbf{x})v(\mathbf{x})\,\mathrm{d}\mathbf{x}$ and the norm in $L^2(X)$ space will be denoted as $\|\cdot\|$. The closeness between the network function f_n and the target function f is measured by the $L^2(X)$ distance:

$$\|f_n - f\| = \left[\int_X |f_n(\mathbf{x}) - f(\mathbf{x})|^2\,\mathrm{d}\mathbf{x}\right]^{1/2}. \tag{4}$$

Definition 2.1 (*Voxman et al. [32, P. 334]*). A function $g(x) : R \to R$ is said to be *piecewise continuous* if it has only a finite number of discontinuities in any interval, and its left and right limits are defined (not necessarily equal) at each discontinuity.

Definition 2.2. The function sequence $\{g_n(\mathbf{x}) = G(\mathbf{x}, \mathbf{a}_n, b_n)\}$ is said to be randomly generated if the corresponding parameters $(\mathbf{a}_n, b_n)$ are randomly generated from $\mathbf{R}^d \times R$ based on a continuous sampling distribution probability.

Definition 2.3. A node is called a random node if its parameters $(\mathbf{a}, b)$ are randomly generated based on a continuous sampling distribution probability.

Remark 1. Definitions 2.2 and 2.3 imply that: Different from the earlier works [24] which "randomly" generate only some parameters (RBF centers $\mathbf{a}_i$) of the hidden nodes based on the distribution of the training data or some probability space determined by the target functions f, the random nodes defined in our I-ELM (and ELM) are independent from the target functions f and the training datasets, and all the hidden node parameters (including both $\mathbf{a}_i$ and b_i) are randomly generated. In the theory and algorithms[1] of I-ELM and ELM the function sequence $\{g_n(\mathbf{x}) = G(\mathbf{x}, \mathbf{a}_n, b_n)\}$ could actually be generated randomly (without prior knowledge) before the target functions or the training datasets are presented.

Without loss of generality, we assume that the network has only one linear output node. All the analysis can be easily extended to multi nonlinear output nodes cases. Let $e_n \equiv f - f_n$ denote the residual error function for the current network f_n with n hidden nodes where $f \in L^2(X)$ is the target function. The I-ELM algorithm [10] does not recalculate the output weights of the existing hidden nodes after a new hidden node is added. In other words, once the output weights of hidden nodes are calculated they will remain frozen and will not be changed any more. That means, the output function f_n of the SLFNs of size n

[1]The ELM codes are available at http://www.ntu.edu.sg/home/egbhuang/.

trained by the I-ELM is

$$f_n(\mathbf{x}) = f_{n-1}(\mathbf{x}) + \beta_n g_n(\mathbf{x}). \tag{5}$$

In I-ELM, one randomly generates the newly added hidden node g_n according to any continuous sampling distribution and then analytically calculates the corresponding output weight $\beta_n = \langle e_{n-1}, g_n\rangle / \|g_n\|^2$ of the new hidden node regardless of those previous existing nodes. In theory, the input and output of the target function f need not be normalized. Although hidden node parameters are chosen randomly in I-ELM, as rigorously proved by Huang et al. [10] I-ELM can work as a universal approximator:

Theorem 2.1 (*[10]*). *Given any bounded nonconstant piecewise continuous function $g: \mathbf{R} \to \mathbf{R}$ for additive nodes or any integrable piecewise continuous function $g: \mathbf{R} \to \mathbf{R}$ and $\int_{\mathbf{R}} g(x)\,\mathrm{d}x \neq 0$ for RBF nodes, for any continuous target function f and any function sequence $\{g_n\}$ randomly generated based on any continuous sampling distribution,* $\lim_{n\to\infty}\|f - f_n\| = 0$ *holds with probability* 1 *if*

$$\beta_n = \frac{\langle e_{n-1}, g_n\rangle}{\|g_n\|^2}. \tag{6}$$

Theorem 2.1 not only shows the universal approximation capability of I-ELM but also proposes a method on how to calculate the output weights of the newly added hidden node. In fact, there may exist different ways to find the output weights of such SLFNs with different features. Different from the network updating function (5) used in I-ELM, CI-ELM adopts the Barron's convex optimization concept [1] by keeping the main feature of I-ELM where hidden nodes are randomly generated:

$$f_n = (1-\beta_n) f_{n-1} + \beta_n g_n, \tag{7}$$

where $0 \leqslant \beta_n \leqslant 1$. In fact, we have:

Theorem 2.2. *Given any nonconstant piecewise continuous function $g: \mathbf{R} \to \mathbf{R}$, if $span\{G(\mathbf{x}, \mathbf{a}, b) : (\mathbf{a}, b) \in \mathbf{R}^d \times R\}$ is dense in L^2, then for any continuous target function f and any function sequence $\{g_n(\mathbf{x}) = G(\mathbf{x}, \mathbf{a}_n, b_n)\}$ randomly generated based on any continuous sampling distribution,* $\lim_{n\to\infty}\|f - ((1-\beta_n)f_{n-1} + \beta_n g_n)\| = 0$ *holds with probability* 1 *if*

$$\beta_n = \frac{\langle e_{n-1}, g_n - f_{n-1}\rangle}{\|g_n - f_{n-1}\|^2}, \tag{8}$$

where $G(\mathbf{x}, \mathbf{a}, b)$ is the output of hidden nodes.

Proof. The proof method of I-ELM [10] can be adopted to prove the validity of this theorem without any major modification. The proof consists of two steps: (a) we first prove that the sequence $\{\|e_n\|\}$ converges; (b) we then further prove that the sequence $\{\|e_n\|\}$ converges to zero.

(a) According to formula (7), we have

$$\begin{aligned} e_n &= f - f_n \\ &= f - ((1-\beta_n) f_{n-1} + \beta_n g_n) \\ &= f - f_{n-1} + \beta_n f_{n-1} - \beta_n g_n \\ &= e_{n-1} - \beta_n (g_n - f_{n-1}). \end{aligned} \tag{9}$$

Let $\Delta = \|e_{n-1}\|^2 - \|e_n\|^2$, then we have

$$\begin{aligned} \Delta &= \|e_{n-1}\|^2 - \|e_{n-1} - \beta_n(g_n - f_{n-1})\|^2 \\ &= 2\beta_n \langle e_{n-1}, g_n - f_{n-1}\rangle - \beta_n^2 \|g_n - f_{n-1}\|^2 \\ &= \|g_n - f_{n-1}\|^2 \left(\frac{\langle e_{n-1}, g_n - f_{n-1}\rangle^2}{\|g_n - f_{n-1}\|^4} \right. \\ &\quad \left. - \left(\beta_n - \frac{\langle e_{n-1}, g_n - f_{n-1}\rangle}{\|g_n - f_{n-1}\|^2} \right)^2 \right). \end{aligned} \tag{10}$$

Δ is maximized iff $\beta_n = \langle e_{n-1}, g_n - f_{n-1}\rangle / \|g_n - f_{n-1}\|^2$, meaning that $\|e_n\| = \|f - ((1-\beta_n) f_{n-1} + \beta_n g_n)\|$ achieves its minimum iff

$$\beta_n = \frac{\langle e_{n-1}, g_n - f_{n-1}\rangle}{\|g_n - f_{n-1}\|^2}. \tag{11}$$

Moreover, when $\beta_n = \langle e_{n-1}, g_n - f_{n-1}\rangle / \|g_n - f_{n-1}\|^2$, $\Delta = \Delta_{\max} = \langle e_{n-1}, g_n - f_{n-1}\rangle^2 / \|g_n - f_{n-1}\|^2 \geqslant 0$. So the sequence $\{\|e_n\|\}$ are decreasing and bounded below by zero and the sequence $\{\|e_n\|\}$ converges.

(b) Seen from the proof of the original I-ELM [10], the sequence $\{\|e_n\|\}$ converges to zero as long as the three sufficient conditions are satisfied: (1) $span\{G(\mathbf{x}, \mathbf{a}, b) : (\mathbf{a}, b) \in \mathbf{R}^d \times R\}$ is dense in L^2 (to support b.1 of the proof of I-ELM [10], p. 882); (2) $e_n \perp (e_{n-1} - e_n)$ (to support b.2 of the proof of I-ELM [10], p. 883); and (3) g is a nonconstant piecewise continuous function[2] (to support b.3 of the proof of I-ELM [10], p. 884). Since conditions (1) and (3) have been given as the preconditions of the theorem, in order to prove $\lim_{n\to+\infty}\|e_n\| = 0$ we only need to prove $e_n \perp (e_{n-1} - e_n)$.

From formula (9) we have

$$\begin{aligned} \langle e_n, g_n - f_{n-1}\rangle &= \langle e_{n-1} - \beta_n(g_n - f_{n-1}), g_n - f_{n-1}\rangle \\ &= \langle e_{n-1}, g_n - f_{n-1}\rangle - \beta_n \langle g_n - f_{n-1}, g_n - f_{n-1}\rangle \\ &= \langle e_{n-1}, g_n - f_{n-1}\rangle - \beta_n \|g_n - f_{n-1}\|^2. \end{aligned} \tag{12}$$

According to formula (11), we further have

$$\begin{aligned} &\langle e_n, g_n - f_{n-1}\rangle \\ &\quad = \langle e_{n-1}, g_n - f_{n-1}\rangle - \frac{\langle e_{n-1}, g_n - f_{n-1}\rangle}{\|g_n - f_{n-1}\|^2} \cdot \|g_n - f_{n-1}\|^2 \\ &\quad = \langle e_{n-1}, g_n - f_{n-1}\rangle - \langle e_{n-1}, g_n - f_{n-1}\rangle = 0. \end{aligned} \tag{13}$$

According to formula (9), $e_n - e_{n-1} = -\beta_n(g_n - f_{n-1})$. Thus, from formula (13) we have

$$\langle e_n, e_n - e_{n-1}\rangle = \langle e_n, -\beta_n(g_n - f_{n-1})\rangle = 0, \tag{14}$$

which means $e_n \perp (e_n - e_{n-1})$.

This completes the proof of this theorem. □

[2]It should be noted that Lemma II.6 of [10] is valid for any nonconstant piecewise continuous function g.

Seen from formula (7) and Theorem 2.2, if a new hidden node is added the output weight of the newly added hidden node is set as: $\beta_n = \langle e_{n-1}, g_n - f_{n-1}\rangle / \|g_n - f_{n-1}\|^2$ and the output weights of the existing hidden nodes will be multiplied by $1-\beta_n = 1 - \langle e_{n-1}, g_n - f_{n-1}\rangle / \|g_n - f_{n-1}\|^2$. Similar to Kwok and Yeung [19], a consistent estimate of β_n based on the training set is

$$\beta_n = \frac{E\cdot[E-(F-H)]^{\mathrm{T}}}{[E-(F-H)]\cdot[E-(F-H)]^{\mathrm{T}}} = \frac{\sum_{p=1}^{N} e(p)[e(p)-(f(p)-h(p))]}{\sum_{p=1}^{N}[e(p)-(f(p)-h(p))]^2}, \tag{15}$$

where $h(p)$ is the activation of the new hidden node for the input of pth training sample and $e(p)$ is the corresponding residual error before this new hidden node is added. $H = [h(1),\ldots,h(N)]^{\mathrm{T}}$ is the activation vector of the new node for all the N training samples and $E = [e(1),\ldots,e(N)]^{\mathrm{T}}$ is the residual vector before this new hidden node added. $F = [t_1,\ldots,t_N]^{\mathrm{T}}$ is the target vector of the target function, where t_p is the target output of the pth training data $\mathbf{x}_p$.

In real applications, one may not really wish to get zero approximation error by adding an infinite number of nodes to the network, a maximum number of hidden nodes is usually given. Thus, such an incremental constructive method for SLFNs can be summarized as follows:

Algorithm CI-ELM. Given a training set $\aleph = \{(\mathbf{x}_i, t_i) | \mathbf{x}_i \in \mathbf{R}^n, t_i \in R, i = 1,\ldots,N\}$, activation function $g(x)$, maximum number of hidden nodes $L_{\max}$ and expected learning accuracy ε,

```
Step Initialization: Let the number of hidden nodes $L = 0$
    and the residual error $E = t$, where $t = [t_1,\ldots,t_N]^{\mathrm{T}}$.
Step Learning step:
    while $L < L_{\max}$ and $\|E\| > \varepsilon$
      (a) Increase by 1 the number of hidden nodes $L$:
          $L = L + 1$.
      (b) Randomly assign hidden node parameters $(\mathbf{a}_L, b_L)$
          for new hidden node $L$.
      (c) Calculate the output weight $\beta_L$ for the newly
          added hidden node:
          $\beta_L = \frac{E\cdot[E-(F-H_L)]^{\mathrm{T}}}{[E-(F-H_L)]\cdot[E-(F-H_L)]^{\mathrm{T}}}$.   (16)
      (d) Recalculate the output weight vectors of all
          existing hidden nodes if $L > 1$:
          $\beta_i = (1-\beta_L)\beta_i, \quad i = 1,\ldots,L-1$.   (17)
      (e) Calculate the residual error after adding the new
          hidden node $L$:
          $E = (1-\beta_L)E + \beta_L(F - H_L)$   (18)
    endwhile
```

β_i in the right side of Eq. (17) represents the output weight of ith hidden node before a new hidden is added and β_i in the left side represents the output weight of ith hidden node after a new hidden added. The residual error E in the right side of Eq. (18) represents the residual error vector before the new node added and the E in the left side represents the residual error vector after the new node added, which is consistent to $E_L = f - f_L = ((1-\beta_L)f + \beta_L f) - ((1-\beta_L)f_{L-1} + \beta_L g_L) = (1-\beta_L)E_{L-1} + \beta_L(f - g_L)$.

Remark 2. Seen from Theorem 2.2, when the network architecture is fixed (with fixed n) we have

Theorem 2.3. *Given any nonconstant piecewise continuous function* $g: \mathbf{R} \to \mathbf{R}$, *if span*$\{G(\mathbf{x}, \mathbf{a}, b) : (\mathbf{a}, b) \in \mathbf{R}^d \times R\}$ *is dense in* L^2, *for any continuous target function f and any function sequence* $\{g_n(\mathbf{x}) = G(\mathbf{x}, \mathbf{a}_n, b_n)\}$ *randomly generated based on any continuous sampling distribution*, $\lim_{n\to\infty}\|f - f_n\| = 0$ *holds with probability* 1 *if the output parameters are determined by ordinary least square to minimize* $\|f(\mathbf{x}) - \sum_{i=1}^{n}\beta_i g_i(\mathbf{x})\|$.

Theorem 2.3 means that the ELM with fixed network architectures [12–14,16] where the output parameters are determined by ordinary least square can work as universal approximators if only the activation function g is nonconstant piecewise and *span*$\{G(\mathbf{x}, \mathbf{a}, b) : (\mathbf{a}, b) \in \mathbf{R}^d \times R\}$ is dense in L^2.

Remark 3. Theorems 2.2 and 2.3 are valid for any type of hidden node activation function g as long as g is nonconstant piecewise continuous and *span*$\{G(\mathbf{x}, \mathbf{a}, b) : (\mathbf{a}, b) \in \mathbf{R}^d \times R\}$ is dense in L^2. The output function of hidden node $G(\mathbf{x}, \mathbf{a}, b)$ could be other than the traditional neural nodes such as additive or RBF hidden nodes. For RBF hidden nodes with activation function g and hidden node parameters $(\mathbf{a}, b)$, $G(\mathbf{x}, \mathbf{a}, b)$ is defined as $G(\mathbf{x}, \mathbf{a}, b) = g(b\|\mathbf{x} - \mathbf{a}\|)$. For additive hidden nodes with activation function g and hidden node parameters $(\mathbf{a}, b)$, $G(\mathbf{x}, \mathbf{a}, b)$ is defined as $G(\mathbf{x}, \mathbf{a}, b) = g(\mathbf{a}\cdot\mathbf{x} + b)$. According to [28] given any bounded nonconstant piecewise continuous integrable g *span*$\{g(b(\mathbf{x} - \mathbf{a})) : (\mathbf{a}, b) \in \mathbf{R}^d \times R^+\}$ is dense in L^2. Thus, Theorems 2.2 and 2.3 are valid for any bounded nonconstant piecewise continuous integrable RBF hidden nodes. Furthermore, according to [21] *span*$\{g(\mathbf{a}\cdot\mathbf{x} + b) : (\mathbf{a}, b) \in \mathbf{R}^d \times R\}$ is dense in L^2 if and only if g is not polynomial (almost everywhere), thus, Theorems 2.2 and 2.3 are also valid for nonconstant piecewise continuous nonpolynomial additive hidden nodes.

Remark 4. The conventional learning algorithms for SLFNs always need to adjust the hidden node parameters $(\mathbf{a}, b)$. Interestingly, Theorems 2.2 and 2.3 indicate that if SLFNs can work as universal approximators with adjustable hidden parameters $(\mathbf{a}, b)$ and one does not care much the network size, from a function approximation point of view the hidden node parameters can actually be randomly generated according to any continuous sampling distribution and then fixed instead of being tuned.

Remark 5. It may be worth mentioning the differences between tuning-free ELM and the earlier tuning-based works [24]. The earlier work is valid for RBF hidden nodes only. However, ELM is a unified framework for generalized SLFNs and this paper has shown its universal approximation capability for any type of computational hidden nodes as long as SLFNs with this type of adjustable hidden nodes can be universal approximators, which include additive/RBF hidden nodes, multiplicative nodes, fuzzy rules [11], fully complex nodes [18,22], hinging functions [3], high-order nodes [7], ridge polynomials [30], wavelets [5,6], and Fourier series [8], etc. The hidden nodes in ELM can be combinatorial nodes each consisting of different type of computational nodes. Even for RBF hidden nodes, although superficially similar, ELM and the earlier work are different in essence.

Each RBF hidden node has two important elements: the center $\mathbf{a}_i$ and the impact factor b_i. In Lowe [24] RBF centers $\mathbf{a}_i$ are randomly chosen from the training data [24], but one has to tune and fix the (same) value of impact factors b_i for all the RBF nodes according to the spread of those RBF centers ELM for RBF case randomly generates both centers $\mathbf{a}_i$ and impact factors b_i, respectively, from $\mathbf{R}^d$ and R^+ according to any continuous sampling distribution. In ELM, all the hidden node parameters are totally randomly generated and completely independent from the training data. If RBF centers and impact factors are selected based on the training data, it may give advantages to the training data and thus easily causes over fitting. In addition, whether the earlier work [24] has the universal approximation capability is still an open question, whereas the universal approximation capability of ELM (with any random sequence of hidden nodes) has been investigated and proved.

From a function approximation point of view the hidden nodes of ELMs are not much relevant to the target functions or the training data. All the hidden node parameters ($\mathbf{a}_i$ and b_i) of ELM could randomly be generated according to any given continuous probability distribution without any prior knowledge (even before the data are presented and the ELM learning starts). All the hidden node parameters ($\mathbf{a}_i$ and b_i) of ELM are not only independent from each other but also independent from the training data, which makes ELM learning scheme more efficiently and much simpler.

3. Performance evaluation

The regression performance of CI-ELM, I-ELM, the stochastic gradient descent BP [20], and other well-known incremental learning algorithms like RAN [29] and MRAN [33] are compared in this paper. Table 1 gives the specification of eight real regression problems [2] used in such comparisons. All the inputs of these datasets are normalized into the range $[-1, 1]$ in our experiments. All the simulations are running in MATLAB 6.5 environment and the same PC with Pentium IV 3.0 GHz CPU and 1G RAM.

Table 1
Specification of benchmark regression problems

Problems	# Training data	# Testing data	# Attributes
Abalone	2000	2177	8
Auto price	80	79	15
Boston housing	250	256	13
California housing	8000	12 640	8
Census (House8L)	10 000	12 784	8
Delta ailerons	3000	4129	5
Delta elevators	4000	5517	6
Machine CPU	100	109	6

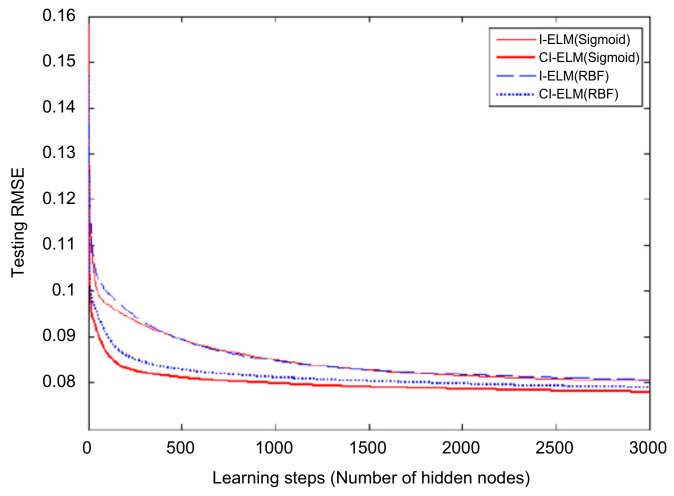

Fig. 1. Performance comparison (testing RMSE) between CI-ELM and I-ELM for abalone case.

The performance of CI-ELM has been tested on two popular activation functions: $g(x) = 1/(1 + \exp(-x))$ (for additive nodes) and $g(\mathbf{x}) = \exp(-b\|\mathbf{x} - \mu\|^2)$ (for RBF nodes). The input weights and hidden biases are randomly chosen from the range $[-1, 1]$ for additive nodes. The centers are randomly chosen from the range $[-1, 1]$ and the impact factor b is chosen from the range $[0, 0.5]$ for RBF nodes.

Fig. 1 shows the performance comparison (testing RMSE) between CI-ELM and I-ELM with up to 3000 hidden nodes for abalone dataset. It can be seen that CI-ELM with additive nodes and RBF nodes obviously converge much faster than I-ELM with additive nodes and RBF nodes, respectively. In fact, the same generalization performance trends have been obtained for all the tested cases.

It is found that without further increasing hidden nodes CI-ELM with 200 hidden nodes can generally obtain good performance for all the tested cases which are comparable

Table 2
Training time (s) comparison between CI-ELM and I-ELM with 200 hidden nodes

Datasets	Sigmoid		RBF		RAN[a]	MRAN[a]	BP[b]
	CI-ELM[a]	I-ELM[a]	CI-ELM[a]	I-ELM[a]			
Abalone	0.2531	0.2214	0.4409	0.5030	39.928	255.84	0.4406
Auto price	0.0300	0.0329	0.0358	0.0468	0.3565	2.5015	0.0154
Boston	0.0311	0.0515	0.0576	0.0657	2.0940	22.767	0.0579
California	0.6902	0.5448	1.2851	1.3656	3301.7	2701.1	2.0307
Census	0.9330	0.8667	1.7113	1.7928	5399.0	3805.3	2.7814
Delta ailerons	0.2078	0.2620	0.3739	0.4327	237.96	175.07	0.6610
Delta elevators	0.2939	0.2708	0.5539	0.6321	661.78	331.75	0.8830
Machine CPU	0.0342	0.0234	0.0376	0.0447	0.1735	0.2454	0.0206

[a]Run in MATLAB environment.
[b]Run in C executable environment which is usually much faster than MATLAB.

Table 3
Performance comparison (testing RMSE) between CI-ELM and I-ELM with 200 hidden nodes

Datasets	Sigmoid		RBF		RAN	MRAN	BP
	CI-ELM	I-ELM	CI-ELM	I-ELM			
Abalone	$\underline{0.0828}$	0.0920	$\underline{0.0858}$	0.0938	0.1183	0.0906	0.1175
Auto price	**0.0927**	0.0977	0.1196	0.1261	0.1418	0.1373	0.2383
Boston	**0.1106**	0.1167	0.1455	0.1320	0.1474	0.1321	0.1882
California	0.1547	0.1683	0.1660	0.1731	0.1506	**0.1480**	0.1579
Census	$\underline{0.0873}$	0.0923	$\underline{0.0860}$	0.0922	0.1061	$\underline{0.0903}$	$\underline{0.0866}$
Delta ailerons	$\underline{0.0480}$	0.0525	$\underline{0.0494}$	0.0632	0.1018	0.0618	$\underline{0.0459}$
Delta elevators	$\underline{0.0604}$	0.0740	$\underline{0.0622}$	0.0790	0.1322	0.0807	$\underline{0.0653}$
Machine CPU	**0.0489**	0.0504	0.0589	0.0674	0.1069	0.1068	0.1988

Table 4
Performance comparison (standard deviation of testing RMSE) between CI-ELM and I-ELM with 200 hidden nodes

Datasets	Sigmoid		RBF		RAN	MRAN	BP
	CI-ELM	I-ELM	CI-ELM	I-ELM			
Abalone	0.0030	0.0046	0.0029	0.0053	0.0076	0.0065	0.0095
Auto price	0.0083	0.0069	0.0177	0.0255	0.0261	0.0381	0.0587
Boston	0.0059	0.0112	0.0076	0.0126	0.0177	0.0140	0.0243
California	0.0049	0.0049	0.0055	0.0081	0.0035	0.0030	0.0033
Census	0.0018	0.0023	0.0023	0.0029	0.0038	0.0042	0.0025
Delta ailerons	0.0058	0.0078	0.0069	0.0116	0.0083	0.0050	0.0033
Delta elevators	0.0067	0.0126	0.0040	0.0123	0.0130	0.0068	0.0019
Machine CPU	0.0084	0.0079	0.0116	0.0177	0.0246	0.0367	0.0429

to the results obtained by RAN, MRAN, and BP. Tables 2 and 3 show the average training time and generalization performance obtained over 20 trials for each case. For the sake of readability, the lowest close generalization performance (testing RMSE) obtained by different algorithms are underlined in Table 3 while the best results among all algorithms are shown in boldface. As observed from Tables 2 and 3 CI-ELM and I-ELM spend almost the same training time for the same network size but CI-ELM generally obtains a better generalization performance. As observed from Fig. 1, CI-ELM has a much faster convergence rate than I-ELM. In order to reach the same generalization performance CI-ELM may need much fewer hidden nodes. Thus CI-ELM may need less training time in order to reach the same generalization performance. Table 4 shows that CI-ELM usually produces smaller standard deviations, which means CI-ELM is usually much stabler.

4. Conclusion

Motivated by the recently proposed learning theory on neural networks with random hidden nodes [10] and the convex optimization method for neural networks [1] this paper further proposes a convex incremental extreme learning machine (CI-ELM). CI-ELM randomly generates and adds computational nodes to the hidden layer and only analytically calculates the output weights of the hidden nodes. Different from I-ELM, based on a convex optimization method CI-ELM recalculates the output weights of the existing hidden nodes after a new hidden node is added. CI-ELM can obtain a faster convergence rate and more compact network architecture while retaining the I-ELM's simplicity and efficiency. The universal approximation capability of CI-ELM has been proved for "generalized" feedforward networks in this paper.

References

[1] A.R. Barron, Universal approximation bounds for superpositions of a sigmoidal function, IEEE Trans. Inf. Theory 39 (3) (1993) 930–945.

[2] C. Blake, C. Merz, UCI Repository of Machine Learning Databases, Department of Information and Computer Sciences, University of California, Irvine, USA, 1998. ⟨http://www.ics.uci.edu/~mlearn/MLRepository.html⟩.

[3] L. Breiman, Hinging hyperplanes for regression, and function approximation, IEEE Trans. Inf. Theory 39 (3) (1993) 999–1013.

[5] I. Daubechies, Orthonormal bases of compactly supported wavelets, Commun. Pure Appl. Math. 41 (1988) 909–996.

[6] I. Daubechies, The wavelet transform, time-frequency localization and signal analysis, IEEE Trans. Inf. Theory 36 (5) (1990) 961–1005.

[7] C.L. Giles, T. Maxwell, Learning, invariance, and generalization in high-order neural networks, Appl. Opt. 26 (23) (1987) 4972–4978.

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

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

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

[11] G.-B. Huang, N.-Y. Liang, H.-J. Rong, P. Saratchandran, N. Sundararajan, On-line sequential extreme learning machine, in: the IASTED International Conference on Computational Intelligence (CI 2005), Calgary, Canada, July 4–6, 2005.

[12] 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), vol. 2, Kunming, China, December 6–9, 2004, pp. 1029–1036.

[13] 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.

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

[15] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Real-time learning capability of neural networks, IEEE Trans. Neural Networks 17 (4) (2006) 863–878.

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

[18] T. Kim, T. Adali, Approximation by fully complex multilayer perceptrons, Neural Comput. 15 (2003) 1641–1666.

[19] T.-Y. Kwok, D.-Y. Yeung, Objective functions for training new hidden units in constructive neural networks, IEEE Trans. Neural Networks 8 (5) (1997) 1131–1148.

[20] Y. LeCun, L. Bottou, G.B. Orr, K.-R. Müller, Efficient BackProp, in: Lecture Notes in Computer Science, vol. 1524, 1998, pp. 9–50.

[21] 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.

[22] M.-B. Li, G.-B. Huang, P. Saratchandran, N. Sundararajan, Fully complex extreme learning machine, Neurocomputing 68 (2005) 306–314.

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

[24] D. Lowe, Adaptive radial basis function nonlinearities and the problem of generalisation, in: Proceedings of the First IEE International Conference on Artificial Neural Networks 1989, pp. 171–175.

[28] J. Park, I.W. Sandberg, Universal approximation using radial-basis-function networks, Neural Comput. 3 (1991) 246–257.

[29] J. Platt, A resource-allocating network for function interpolation, Neural Comput. 3 (1991) 213–225.

[30] Y. Shin, J. Ghosh, Approximation of multivariate functions using ridge polynomial networks, in: Proceedings of the International Joint Conference on Neural Networks (IJCNN'2002), Baltimore, MD, June 2002, pp. 380–385.

[32] W.L. Voxman, J. Roy, H. Goetschel, Advanced Calculus: An Introduction to Modern Analysis, Marcel Dekker, New York, 1981.

[33] L. Yingwei, N. Sundararajan, P. Saratchandran, A sequential learning scheme for function approximation using minimal radial basis function (RBF) neural networks, Neural Comput. 9 (1997) 461–478.

Guang-Bin Huang received the B.Sc. degree in applied mathematics and the M.Eng. degree in computer engineering from Northeastern University, China, in 1991 and 1994, respectively, and the Ph.D. degree in electrical engineering from Nanyang Technological University, Singapore, in 1999. From June 1998 to May 2001, he was a Research Fellow with the Singapore Institute of Manufacturing Technology (formerly known as Gintic Institute of Manufacturing Technology), where he led/implemented several key industrial projects. Since then, he has been an Assistant Professor in the School of Electrical and Electronic Engineering, Nanyang Technological University. His current research interests include machine learning, bioinformatics, and networking. He is a senior member of IEEE.

Dr. Huang is an Associate Editor of *Neurocomputing* and IEEE Transaction on Systems, Man, and Cybernetics—Part B: Cybernetics.

Lei Chen received the B.Sc. degree in applied mathematics and the M.Sc. degree in operational research and control theory from Northeastern University, China, in 1999 and 2002, respectively. He is currently working toward the Ph.D. degree from Nanyang Technological University, Singapore. He is working as a Research Associate in National University of Singapore. His research interests include artificial neural networks, pattern recognition, and machine learning.