Universal Approximation Using Incremental Constructive Feedforward Networks With Random Hidden Nodes

Guang-Bin Huang, *Senior Member, IEEE*, Lei Chen, and Chee-Kheong Siew, *Member, IEEE*

***Abstract*—According to conventional neural network theories, single-hidden-layer feedforward networks (SLFNs) with additive or radial basis function (RBF) hidden nodes are universal approximators when all the parameters of the networks are allowed adjustable. However, as observed in most neural network implementations, tuning all the parameters of the networks may cause learning complicated and inefficient, and it may be difficult to train networks with nondifferential activation functions such as threshold networks. Unlike conventional neural network theories, this paper proves in an incremental constructive method that in order to let SLFNs work as universal approximators, one may simply randomly choose hidden nodes and then only need to adjust 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 $\int_R g(x)dx \neq 0$. The proposed incremental method is efficient not only for SFLNs with continuous (including nondifferentiable) activation functions but also for SLFNs with piecewise continuous (such as threshold) activation functions. Compared to other popular methods such a new network is fully automatic and users need not intervene the learning process by manually tuning control parameters.**

***Index Terms*—Ensemble, feedforward network, incremental extreme learning machine, radial basis function, random hidden nodes, support vector machine, threshold network, universal approximation.**

I. Introduction

THE widespread popularity of neural networks in many fields is mainly due to their ability to approximate complex nonlinear mappings directly from the input samples. Neural networks can provide models for a large class of natural and artificial phenomena that are difficult to handle using classical parametric techniques. Out of many kinds of neural networks, single-hidden-layer feedforward neural networks (SLFNs) have been investigated more thoroughly.

Seen from the viewpoint of network architectures, two main SLFN network architectures have been investigated: 1) the SLFNs with additive hidden nodes and 2) radial basis function (RBF) networks which apply RBF nodes in the hidden layer. The output function of an SLFN with n additive nodes can be represented by

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

It should be: Manuscript received October 29, 2003; revised May 8, 2005 and October 24, 2005.
Manuscript received May 8, 2005; revised October 24, 2005.

The authors are with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798, Singapore (e-mail: gbhuang@ieee.org).

Digital Object Identifier 10.1109/TNN.2006.875977

where $\mathbf{a}_i$ is the weight vector connecting the input layer to the ith hidden node, β_i is the weight connecting the ith hidden node to the output node, and g is the hidden node activation function. $\mathbf{a}_i \cdot \mathbf{x}$ denotes the inner product of vectors $\mathbf{a}_i$ and $\mathbf{x}$ in $\mathbf{R}^d$. The RBF network is considered a specific SLFN which applies RBF nodes in its hidden layer. Each RBF node has its own centroid and impact factor, and its output is some radially symmetric function of the distance between the input and the center. The output function of an SLFN with n RBF nodes can be represented by

$$f_n(\mathbf{x}) = \sum_{i=1}^{n} \beta_i g\left(\frac{\|\mathbf{x} - \mathbf{a}_i\|}{b_i}\right) \quad \mathbf{a}_i \in \mathbf{R}^d, \quad b_i \in R^+, \quad \beta_i \in R \quad (2)$$

where $\mathbf{a}_i$ and b_i are the center and impact factor of ith RBF node and β_i is the weight connecting the ith RBF hidden node to the output node. R^+ indicates the set of all positive real value. From a mathematical point of view, research on the approximation capabilities of SLFNs has focused on two aspects: universal approximation on compact input sets and approximation in a set consisting of finite number of training samples.

Hornik [1] proved that if the activation function is continuous, bounded, and nonconstant, then continuous mappings can be approximated by SLFNs with additive hidden nodes over compact input sets. Leshno [2] improved the results of Hornik [1] and proved that SLFNs with additive hidden nodes and with a nonpolynomial activation function can approximate any continuous target functions. Regarding the universal approximation capability of RBF network, Park and Sandberg [3] has proved that RBF network with the same impact factor for suitably chosen kernels can approximate any continuous target function. For function approximation in a set consisting of N training samples $\{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in X \subset \mathbf{R}^d, \mathbf{t}_i \in \mathbf{R}^m, i = 1, 2, \ldots, N\}$, some researchers have proposed constructive methods for SLFNs. For instance, Huang and Babri [4] show that an SLFN with at most N hidden nodes and with almost any nonlinear activation function can learn N distinct observations with zero error. It should be noted that all the network parameters (including input weights and biases of additive nodes or centers and impact factor of RBF nodes) need to be adjusted in all these previous research works.

It has been shown in our recent work [5] that with the input weights chosen randomly, both SLFNs (with N additive hidden nodes) and two-hidden-layer feedforward neural networks (TLFNs) (with at most $O(\sqrt{N})$ additive hidden nodes) can learn N distinct observations with arbitrarily small error. One

may only simply adjust the rest parameters except for the input weights and first hidden layer biases in applications [6]–[10]. Baum [11] has claimed that (seen from simulations for SLFN with additive hidden nodes) one may fix the weights of the connections on one level and simply adjust the connections on the other level and no gain is possible by using an algorithm able to adjust the weights on both levels simultaneously. Furthermore, Huang *et al.* [12], [13] show that the centers and impact factors of the RBF nodes of RBF networks can also be randomly chosen. In fact, some simulation results on artificial and real large-scale complex cases [6]–[10], [12], [13] have shown that this method (with fixed network architectures) may be dramatically efficient and have good generalization performance. Some researchers [14]–[16] have proved in theory that for SLFN with additive hidden nodes, one may freeze input weights and biases of hidden nodes once they have been *tuned* and need not further adjust them when new nodes are added.

These research results may imply that in the many applications of SLFNs, input weights and biases of additive hidden nodes or centers and impact factors of RBF hidden nodes need not be adjusted at all. However, so far there lacks a strict theoretical justification for this viewpoint, and it is not clear what functions can be chosen as activation functions of hidden layers in such SLFNs with randomly generated nodes.

This paper mainly proves in theory that given any *bounded nonconstant piecewise continuous* activation function $g : R \to R$ for additive nodes or any *integrable piecewise continuous* activation function $g : R \to R$ (and $\int_R g(x)dx \neq 0$) for RBF nodes, the network sequence $\{f_n\}$ with randomly generated hidden nodes can converge to any continuous target function f by only properly adjusting the output weights linking the hidden layer to the output nodes. Although a systematically investigation of the performance of such networks is not the aim of this paper, the constructive methods have been tested on both artificial and real-world benchmark problems. As observed from the simulation results, the proposed constructive methods can actually achieve good performance. Compared to other popular algorithms such as support vector regression (SVR) [17], [18], stochastic gradient descent backpropagation (BP) [19], and incremental RBF networks (RAN [20], RANEKF [21], MRAN [22], [23], GAP-RBF [24], and GGAP-RBF [25]), the proposed incremental SLFN is fully automatic in the sense that except for target errors and the allowed maximum number of hidden nodes, no control parameters need to be manually tuned by users.

This paper is organized as follows. Section II proves in theory that SLFNs with random additive or RBF hidden nodes are universal approximators for any continuous functions on any compact sets. Section III introduces the incremental algorithm with random additive or RBF hidden nodes, which is directly derived from our new universal approximation theory. We will also extend the case of incremental SLFNs with random additive nodes to the case of TLFNs with random additive nodes and further show that in many implementations, like ensemble computing and local experts mixture, the different learning machines may in general share the additive hidden nodes and thus much more compact network architectures could be obtained. Performance evaluation is presented in Section IV. Discussions and conclusions are given in Section V.

II. Universal Approximation Capability of Single-Hidden-Layer Feedforward Networks (SLFNs) With Random Nodes

We first prove in theory that SLFNs with randomly generated additive or RBF nodes and with a broad type of activation functions can universally approximate any continuous target functions in any compact subset X of the Euclidean space $\mathbf{R}^d$.

Without loss of generality for universal approximation, assume that the network has only one linear output node. It can be easily seen that the extension of all the analysis conducted in this paper to multinonlinear output nodes cases is straightforward. SLFN network functions with n hidden nodes can be represented by

$$f_n(\mathbf{x}) = \sum_{i=1}^{n} \beta_i g_i(\mathbf{x}), \qquad \mathbf{x} \in \mathbf{R}^d, \qquad \beta_i \in R \tag{3}$$

where g_i denotes the hidden node output function. For additive nodes with activation function g, g_i is defined as

$$g_i = g(\mathbf{a}_i \cdot \mathbf{x} + b_i), \qquad \mathbf{a}_i \in \mathbf{R}^d, \qquad b_i \in R \tag{4}$$

and for RBF nodes with activation function g, g_i is defined as

$$g_i = g\left(\frac{\|\mathbf{x} - \mathbf{a}_i\|}{b_i}\right), \qquad \mathbf{a}_i \in \mathbf{R}^d, \qquad b_i \in R^+. \tag{5}$$

It has been shown [1]–[3], [26]–[29] that given activation function $g(x)$ satisfying certain mild conditions, there exists a sequence of network functions $\{f_n\}$ approximating any given continuous target function f. It should be noted that all parameters in any f_n in the sequence are required freely adjustable in all these previous works. Some researchers [14]–[16] proposed incremental methods for SLFNs with additive nodes which can allow input weights and biases of new hidden nodes to be tuned when added and then fixed after tuned. Romero [30] also proposes an incremental algorithm for SLFNs with additive and RBF nodes and shows that the input weights and biases or centers and impact factors of hidden nodes are tuned when added and can then be kept fixed.

The major objective of this paper is to show that given any *bounded nonconstant piecewise continuous* activation function $g : R \to R$ for additive nodes or any *integrable piecewise continuous* activation function $g : R \to R$ (and $\int_R g(x)dx \neq 0$) for RBF nodes, the network sequence $\{f_n\}$ with randomly generated hidden nodes can converge to any continuous target function f by only properly adjusting the output weights. Different from other increment algorithms for SLFNs with additive nodes [14]–[16], [30], [31] and incremental RBF networks [20]–[25], [30], new nodes need not be tuned at the time they are added. Thus, the efficiency of SLFNs could be considerably increased.

Let $L^2(X)$ be a space of functions f on a compact subset X in the d-dimensional Euclidean space $\mathbf{R}^d$ such that $|f|^2$ are integrable, that is, $\int_X |f(\mathbf{x})|^2 d\mathbf{x} < \infty$. Let $L^2(\mathbf{R}^d)$ be denoted

by L^2. Similar to [15], for $u, v \in L^2(X)$, the inner product $\langle u, v\rangle$ is defined by

$$\langle u, v\rangle = \int_X u(\mathbf{x})v(\mathbf{x})d\mathbf{x}. \tag{6}$$

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

$$\|f_n - f\| = \left[\int_X |f_n(\mathbf{x}) - f(\mathbf{x})|^2 d\mathbf{x}\right]^{1/2}. \tag{7}$$

Definition II.1 [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 II.2: The function sequence $\{g_n = g(\mathbf{a}_n\cdot\mathbf{x}+b_n)\}$ or $\{g_n = g(\|\mathbf{x}-\mathbf{a}_n\|/b_n)\}$ is said to be randomly generated if the corresponding parameters $(\mathbf{a}_n, b_n)$ are randomly generated from $\mathbf{R}^d \times R$ or $\mathbf{R}^d \times R^+$ based on a continuous sampling distribution probability.

Remark 1: As done in our simulations, one may randomly generate sequence $\{g_n\}$ based on a uniform sampling distribution probability.

Definition II.3: A node is called a random node if its parameters $(\mathbf{a}, b)$ are randomly generated based on a continuous sampling distribution probability.

A. Necessary Lemmas

Some lemmas that are used to prove our main Theorem II.1 are provided in this section.

Lemma II.1 [33, p. 81]: The space of L^2 is complete.

Lemma II.2 [33, p. 80]: The sum of two functions of L^2 is an element of L^2.

Lemma II.3 [2, Proposition 1]: Given $g : R \to R$, $span\{g(\mathbf{a}\cdot\mathbf{x}+b) : (\mathbf{a}, b) \in \mathbf{R}^d \times R\}$ is dense in L^p for every $p \in [1, \infty)$, if and only if g is not a polynomial (almost everywhere).

Lemma II.4 [3]: Let $k : \mathbf{R}^d \to R$ be an integrable bounded function such that k is continuous (almost everywhere) and $\int_{\mathbf{R}^d} k(\mathbf{x})d\mathbf{x} \neq 0$. Then $span\{k((\mathbf{x}-\mathbf{a})/b) : (\mathbf{a}, b) \in \mathbf{R}^d \times R^+\}$ is dense in L^p for every $p \in [1, \infty)$.

Remark 2: As mentioned by Park and Sandberg [3, p. 252], there is no requirement of radial symmetry of the kernel function k in Lemma II.4. The radial basis functions case considered in this paper is a specific case of Lemma II.4: $k((\mathbf{x}-\mathbf{a})/b) = g(\|\mathbf{x}-\mathbf{a}\|/b)$.

Lemma II.5: Given a bounded nonconstant piecewise continuous function $g : R \to R$, we have

$$\lim_{(\mathbf{a},b)\to(\mathbf{a}_0,b_0)} \|g(\mathbf{a}\cdot\mathbf{x}+b) - g(\mathbf{a}_0\cdot\mathbf{x}+b_0)\| = 0 \quad \forall(\mathbf{a}_0, b_0) \in \mathbf{R}^d \times R \tag{8}$$

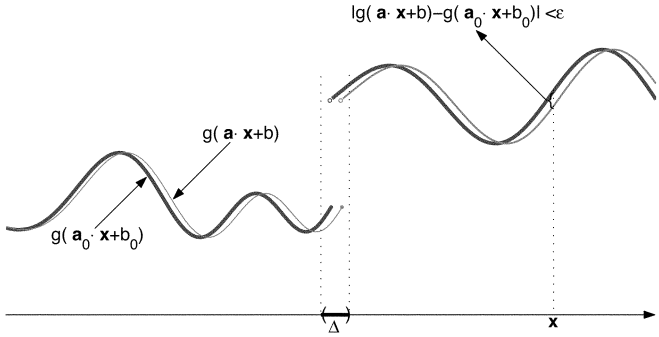

Fig. 1. Bounded piecewise continuous functions $g(\mathbf{a}\cdot\mathbf{x}+b)$ and $g(\|\mathbf{x}-\mathbf{a}\|/b)$ can, respectively, be as close to $g(\mathbf{a}_0\cdot\mathbf{x}+b_0)$ and $g(\|\mathbf{x}-\mathbf{a}_0\|/b_0)$ as possible by adjusting $(\mathbf{a}, b)$.

and

$$\lim_{(\mathbf{a},b)\to(\mathbf{a}_0,b_0)} \left\| g\left(\frac{\|\mathbf{x}-\mathbf{a}\|}{b}\right) - g\left(\frac{\|\mathbf{x}-\mathbf{a}_0\|}{b_0}\right)\right\| = 0 \quad \forall(\mathbf{a}_0, b_0) \in \mathbf{R}^d \times R^+. \tag{9}$$

Proof: Let $\hat{g}_0$ and $\hat{g}$ denote $g(\mathbf{a}_0\cdot\mathbf{x}+b_0)$ and $g(\mathbf{a}\cdot\mathbf{x}+b)$ (or $g(\|\mathbf{x}-\mathbf{a}_0\|/b_0)$ and $g(\|\mathbf{x}-\mathbf{a}\|/b)$), respectively. Since g is bounded, suppose that $|g(x)| < M$ for all x. Given any small positive value ϵ, set $\gamma = (1/2)(\epsilon/8M^2)^{1/d}$. Without loss of generality, suppose that $g(x)$ has only one discontinuity $\breve{x} \in R$ and $\hat{g}_0$ and $\hat{g}$ have single discontinuity points $\hat{\mathbf{x}}_0 \in \mathbf{R}^d$ and $\hat{\mathbf{x}} \in \mathbf{R}^d$, respectively. Thus, for given $(\mathbf{a}_0, b_0)$ and $(\mathbf{a}, b)$, we have $\mathbf{a}_0\cdot\hat{\mathbf{x}}_0+b_0 = \mathbf{a}\cdot\hat{\mathbf{x}}+b = \breve{x}$ (or $(\|\hat{\mathbf{x}}_0-\mathbf{a}_0\|/b_0) = (\|\hat{\mathbf{x}}-\mathbf{a}\|/b) = \breve{x}$). Since $\mathbf{a}\cdot\mathbf{x}+b$ (or $\|\mathbf{x}-\mathbf{a}\|/b$) is a continuous function of $\mathbf{a}$ and b, there exists $\gamma_1 > 0$ such that when $\|(\mathbf{a}, b) - (\mathbf{a}_0, b_0)\| \leq \gamma_1$, we have $\|\hat{\mathbf{x}} - \hat{\mathbf{x}}_0\| < \gamma$. Let Δ denote a neighborhood of point $\hat{\mathbf{x}}_0$ (as shown in Fig. 1)

$$\Delta = \{\mathbf{x} : \|\mathbf{x} - \hat{\mathbf{x}}_0\| < \gamma\}. \tag{10}$$

Obviously, the discontinuous points ($\hat{\mathbf{x}}_0$ and $\hat{\mathbf{x}}$) of $\hat{g}_0$ and $\hat{g}$ are then located in the small subset Δ. Thus, $\hat{g} - \hat{g}_0$ shall be continuous in the subset $X - \Delta$. Further, there exists $\gamma_2 < \gamma_1$ when $\|(\mathbf{a}, b) - (\mathbf{a}_0, b_0)\| \leq \gamma_2$; we have $(\hat{g} - \hat{g}_0)^2 \leq (\epsilon/2\int_X d\mathbf{x})$ for all $\mathbf{x} \in X - \Delta$. Thus, when $\|(\mathbf{a}, b) - (\mathbf{a}_0, b_0)\| \leq \gamma_2$, we have

$$\int_X (\hat{g} - \hat{g}_0)^2 d\mathbf{x} \leq \int_{X-\Delta} (\hat{g}-\hat{g}_0)^2 d\mathbf{x} + \int_\Delta (\hat{g}-\hat{g}_0)^2 d\mathbf{x} < \frac{\epsilon}{2\int_X d\mathbf{x}} \cdot \int_X d\mathbf{x} + (2\gamma)^d \cdot 4M^2 = \epsilon. \tag{11}$$

This completes the proof. ∎

Lemma II.6: Given a bounded nonconstant piecewise continuous function $g : R \to R$ and any given positive value $\hat{\theta} < \pi/2$ and any given g_0, for any randomly generated function sequence $\{g_n\}$, there exists a positive integer M such that for any continuous segment $G_{(n,M)} = \{g_n, g_{n+1}, \ldots, g_{n+M-1}\}$ $(n = 1, 2, \ldots)$, with probability as close to one as desired, there exists $g_i \in G_{(n,M)}$ satisfying

$$\theta_{(g_i,g_0)} < \hat{\theta} \tag{12}$$

where $\theta_{(g_i,g_0)}$ denotes the angle formed by g_i and g_0.

Proof: Given a positive value $\hat{\theta} < \pi/2$, according to Lemma II.5, there exists $\delta > 0$ such that $\forall(\mathbf{a}, b) \in Q_0 = \{(\mathbf{a}, b) : \|(\mathbf{a}, b) - (\mathbf{a}_0, b_0)\| < \delta\}$, $\|g - g_0\| < \|g_0\| \sin(\hat{\theta})$. Suppose that the continuous sampling distribution probability in $\mathbf{R}^d \times R$ (for the additive node case or $\mathbf{R}^d \times R^+$ for the RBF node case) is $p(\mathbf{x})$ and $\int_{\mathbf{R}^d \times R} p(\mathbf{x})d\mathbf{x} = 1$ (or $\int_{\mathbf{R}^d \times R^+} p(\mathbf{x})d\mathbf{x} = 1$). Then for any continuous segment $G_{(n,M)} = \{g_n, g_{n+1}, \ldots, g_{n+M-1}\}$, the probability that among all the M elements the parameters $(\mathbf{a}, b)$ of some elements are sampled from the area Q_0 is $\int_{Q_0} p(\mathbf{x})d\mathbf{x}$. That is, from the probability point of view, there probably are $\sum_{i=0}^{M-1}(1 - \int_{Q_0} p(\mathbf{x})d\mathbf{x})^i \cdot \int_{Q_0} p(\mathbf{x})d\mathbf{x}$ elements $g_i \in G_{(n,M)}$ such that $\|g_i - g_0\| < \|g_0\| \sin(\hat{\theta})$. Thus, M can be selected to make $\sum_{i=0}^{M-1}(1 - \int_{Q_0} p(\mathbf{x})d\mathbf{x})^i \cdot \int_{Q_0} p(\mathbf{x})d\mathbf{x} \approx 1$; then for any continuous segment $G_{(n,M)} = \{g_n, g_{n+1}, \ldots, g_{n+M-1}\}$ at least there exists an element $g_i \in G_{n,M}$ and $(\mathbf{a}_i, b_i) \in Q_0$. That is, there exists M such that $\exists g_i \in G_{(n,M)}$ $(\forall n)$ satisfying $\sin(\theta_{(g_i,g_0)}) \leq \|g_i - g_0\|/\|g_0\| < \sin(\hat{\theta})$ where $\theta_{(g_i,g_0)} < \pi/2$, that is, $\theta_{(g_i,g_0)} < \hat{\theta}$. ∎

B. Universal Approximation Using SLFNs With Random Nodes

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. We can now prove that given any *bounded nonconstant piecewise continuous* activation function $g : R \to R$ for additive nodes or *integrable piecewise continuous* activation function $g : R \to R$ (and $\int_R g(x)dx \neq 0$) for RBF nodes, for any continuous target function f and any randomly generated sequence $\{g_n\}$, $\lim_{n\to\infty} \|e_n\| = 0$ holds if $\beta_n = \langle e_{n-1}, g_n\rangle/\|g_n\|^2$.

Theorem II.1: Given any bounded nonconstant piecewise continuous function $g : R \to R$ for additive nodes or any integrable piecewise continuous function $g : R \to R$ and $\int_R g(x)dx \neq 0$ for RBF nodes, for any continuous target function f and any randomly generated function sequence $\{g_n\}$, $\lim_{n\to\infty} \|f - f_n\| = 0$ holds with probability one if

$$\beta_n = \frac{\langle e_{n-1}, g_n\rangle}{\|g_n\|^2}. \tag{13}$$

Proof: We first prove that the expression $\|e_n\| = \|f - (f_{n-1} + \beta g_n)\|$ achieves its minimum iff $\beta = \beta_n = \langle e_{n-1}, g_n\rangle/\|g_n\|^2$ and the sequence $\{\|e_n\|\}$ converges. Then we further prove $\lim_{n\to+\infty} e_n = 0$.

a) Since g is bounded nonconstant piecewise continuous and f is continuous, we have g_n, $f \in L^2(X)$, and $\|g_n\| = \int_X g_n^2 d\mathbf{x} \neq 0$. According to Lemma II.2, $e_n \in L^2(X)$. Let $\Delta = \|e_{n-1}\|^2 - \|e_n\|^2$; then we have

$$\begin{aligned}\Delta &= \|e_{n-1}\|^2 - \|f - (f_{n-1} + \beta_n g_n)\|^2 \\ &= \|e_{n-1}\|^2 - \|e_{n-1} - \beta_n g_n\|^2 \\ &= 2\beta_n\langle e_{n-1}, g_n\rangle - \beta_n^2\|g_n\|^2 \\ &= \|g_n\|^2\left(\frac{\langle e_{n-1}, g_n\rangle^2}{\|g_n\|^4} - \left(\beta_n - \frac{\langle e_{n-1}, g_n\rangle}{\|g_n\|^2}\right)^2\right). \end{aligned} \tag{14}$$

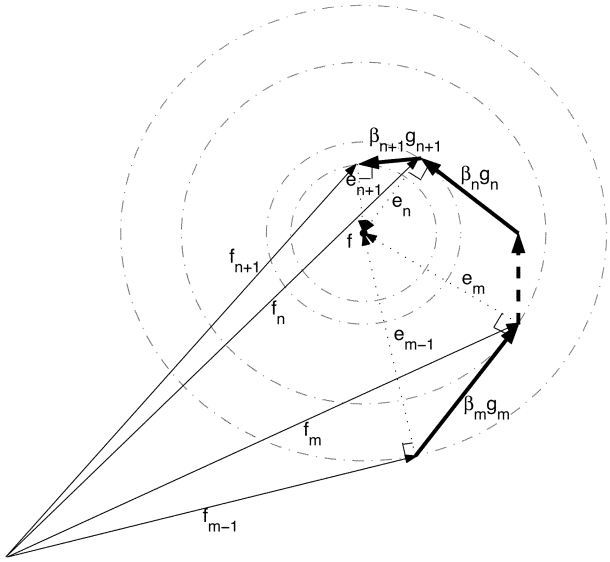

Fig. 2. Error residual function e_n is always orthogonal to g_n: $e_n \perp g_n$. The sequence $\{\|e_n\|\}$ is decreasing and bounded below by zero.

Δ is maximized iff $\beta_n = \langle e_{n-1}, g_n\rangle/\|g_n\|^2$, meaning that $\|e_n\| = \|f - (f_{n-1} + \beta_n g_n)\|$ achieves its minimum iff $\beta = \beta_n = \langle e_{n-1}, g_n\rangle/\|g_n\|^2$. From the function space point of view, $e_n \perp g_n$ when $\|e_n\| = \|f - (f_{n-1} + \beta_n g_n)\|$ achieves its minimum (see Fig. 2). In fact, when $\beta_n = \langle e_{n-1}, g_n\rangle/\|g_n\|^2$, we have

$$\begin{aligned}\langle e_n, g_n\rangle &= \langle e_{n-1} - \beta_n g_n, g_n\rangle \\ &= \langle e_{n-1}, g_n\rangle - \beta_n\langle g_n, g_n\rangle \\ &= 0. \end{aligned} \tag{15}$$

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

b) We can now prove $\lim_{n\to+\infty} \|e_n\| = 0$ by contradiction method.

b.1) Since the sequence $\{\|e_n\|\}$ converges, there exists $r \geq 0$ such as $\lim_{n\to+\infty} \|e_n\| = r$. Suppose $r > 0$; then the sequence $\{\|e_n\|\}$ is decreasing and bounded below by r, that is, $\|e_n\| \geq r$ for all positive integer n. Thus, $\forall\delta > 0$, $\exists N_1 > 0$, such that when $n > \tilde{N}_1$, we have $r \leq \|e_n\| < r + \delta$, which implies an infinite number of e_n $(\forall n > \tilde{N}_1)$ covered by a compact set. Thus, there exists a subsequence $\{e_{n_k}\}$ which converges to a limit denoted by e^* and $\|e^*\| = \lim_{k\to+\infty} \|e_{n_k}\| = r$. Since $e_{n_k} \in L^2(X)$ and $L^2(X)$ is complete (see Lemma II.1), we have $e^* \in L^2(X)$.

Furthermore, there should exist $g^* = g(\mathbf{a}^* \cdot \mathbf{x} + b^*)$ or $g^* = g(\|\mathbf{x} - \mathbf{a}^*\|/b^*)$ such that g^* is not orthogonal to e^*. Otherwise, e^* is orthogonal to $\text{span}\{g(\mathbf{a} \cdot \mathbf{x} + b) : (\mathbf{a}, b) \in \mathbf{R}^d \times R\}$ or $\text{span}\{g(\|\mathbf{x} - \mathbf{a}\|/b) : (\mathbf{a}, b) \in \mathbf{R}^d \times R^+\}$, which is contradictory to the fact that $\text{span}\{g(\mathbf{a} \cdot \mathbf{x} + b) : (\mathbf{a}, b) \in \mathbf{R}^d \times R\}$ (see Lemma II.3) or

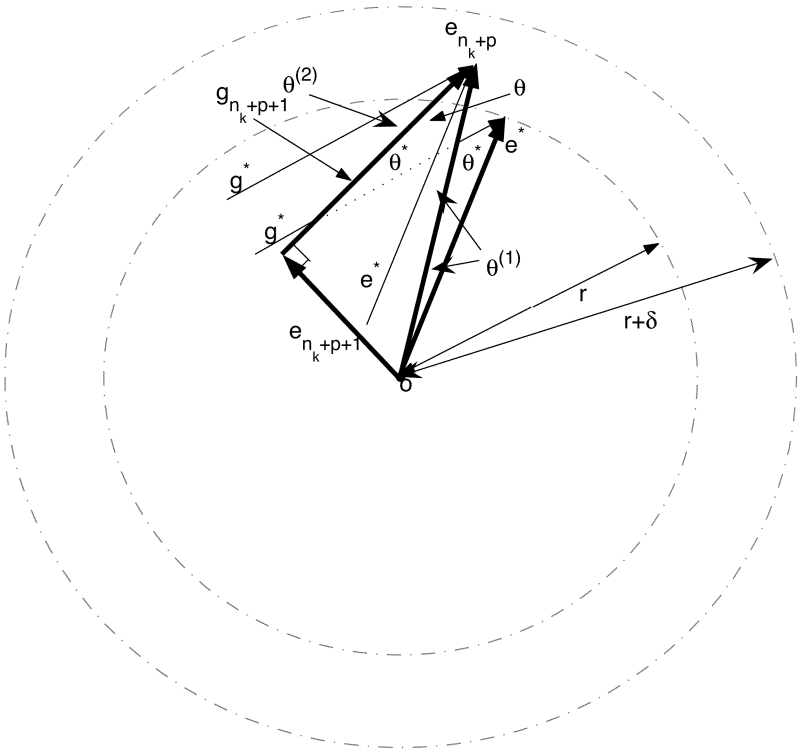

Fig. 3. For the limit e^* of a subsequence of $\{e_n\}$, if $\|e^*\| = r \neq 0$ there exists a g^* which is not orthogonal to e^*, then there exists e_{n_k+p+1} such that $\|e_{n_k+p+1}\| < \lim_{n\to+\infty}\|e_n\| = r$, which is contradictory to the fact that $\|e_{n_k+p+1}\| \geq r$. Thus r must be zero. For the sake of simplicity, $\theta_{(g^*,e^*)}$, $\theta_{(e_{n_k+p},e^*)}$, $\theta_{(g_{n_k+p+1},g^*)}$, and $\theta_{(e_{n_k+p},g_{n_k+p+1})}$ are denoted by θ^*, $\theta^{(1)}$, $\theta^{(2)}$, and θ, respectively. $\theta \leq \theta^* + \theta^{(1)} + \theta^{(2)}$.

$\operatorname{span}\{g(\|\mathbf{x}-\mathbf{a}\|/b) : (\mathbf{a},b) \in \mathbf{R}^d \times R^+\}$ is dense in $L^2(X)$ (see Lemma II.4).

Let $\theta_{(g^*,e^*)}$ denote the angle formed by g^* and e^* and we have $0 \leq \theta_{(g^*,e^*)} < \pi/2$. (see Fig. 3).

b.2) Let $\theta_{(e_n,e^*)}$ be the angle formed by e_n and e^*, with $0 \leq \theta_{(e_n,e^*)} \leq \pi/2$. Choose any small positive value ξ such that $\xi < 1 - \sin(\theta_{(g^*,e^*)})$. Since $\lim_{k\to+\infty}\|e_{n_k}\| = \|e^*\| = r$, $\exists \tilde{N}_2$ when $k > \tilde{N}_2$

$$\|e_{n_k}\| < \frac{r}{1-\xi}$$
$$\theta_{(e_{n_k},e^*)} < \arcsin\left(\frac{1-\xi-\sin\left(\theta_{(g^*,e^*)}\right)}{2}\right). \quad (16)$$

Since for all positive integer n, $e_n \perp g_n$ and $e_n = e_{n-1} - \beta_n g_n$ [see (15)], we have $e_n \perp (e_{n-1} - e_n)$. Furthermore, we have

$$\begin{aligned}\|e_{n-1}-e_n\|^2 &= \|e_{n-1}\|^2 + \|e_n\|^2 - 2\langle e_{n-1}, e_n\rangle \\ &= \|e_{n-1}\|^2 + \|e_n\|^2 \\ &\quad - 2\left(\langle e_{n-1}, e_n\rangle - \langle e_{n-1}-e_n, e_n\rangle\right) \\ &= \|e_{n-1}\|^2 - \|e_n\|^2. \end{aligned} \quad (17)$$

Thus, we have $\lim_{n\to+\infty}\|\beta_n g_n\|^2 = \lim_{n\to+\infty}(\|e_{n-1}\|^2 - \|e_n\|^2) = r^2 - r^2 = 0$. Then, for any given positive integer m, we have $\lim_{n\to+\infty}\sum_{p=n+1}^{n+m}\|\beta_p g_p\| = 0$. Since $\|e_{n_k+m} - e_{n_k}\| = \|\sum_{p=n_k+1}^{n_k+m}\beta_p g_p\| \leq \sum_{p=n_k+1}^{n_k+m}\|\beta_p g_p\|$, for any fixed positive integer m, we have

$$\lim_{k\to+\infty}\|e_{n_k} - e_{n_k+m}\| = 0. \quad (18)$$

We, then, further have

$$\begin{aligned}\lim_{k\to+\infty}\|e_{n_k+m} - e^*\| &= \lim_{k\to+\infty}\|e_{n_k+m} - e_{n_k} + e_{n_k} - e^*\| \\ &\leq \lim_{k\to+\infty}\|e_{n_k+m} - e_{n_k}\| \\ &\quad + \lim_{k\to+\infty}\|e_{n_k} - e^*\| \\ &= 0 \end{aligned} \quad (19)$$

where e_{n_k+m} need not be an element of the subsequence $\{e_{n_k}\}$. According to (16) and (19), there exists $\tilde{N}_3 > \tilde{N}_2$, $\forall k > \tilde{N}_3$ we have (see Fig. 3)

$$\|e_{n_k+p}\| < \frac{r}{1-\xi}$$
$$\theta_{(e_{n_k+p},e^*)} < \arcsin\left(\frac{1-\xi-\sin\left(\theta_{(g^*,e^*)}\right)}{2}\right), \quad 0 \leq p \leq L(n_k) \quad (20)$$

where $L(n_k)$ is a function of k : $L(n_k) = \max_m\{m : \|e_{n_k+p}\| < r/(1-\xi)$ and $\theta_{(e_{n_k+p},e^*)} < \arcsin(1-\xi-\sin(\theta_{(g^*,e^*)})/2), 0 \leq p \leq m\}$. Obviously for any given positive

integer M, $\exists k > \tilde{N}_3 L(n_k) > M$ since for any given positive integer m we have $\lim_{k\to+\infty} \sum_{p=n_k+1}^{n_k+m} \|\beta_p g_p\| = 0$.

b.3) Let $\theta_{(g_n,g^*)}$ denote the angle formed by g_n and g^*, with $0 \le \theta_{(g_n,g^*)} \le \pi/2$. Since function sequences $\{g_n\}$ are randomly generated at a continuous sampling distribution probability, according to Lemma II.6, there exists a positive integer M such that for any continuous segment, $G_{(n,M)} = \{g_n, g_{n+1}, \ldots, g_{n+M-1}\}$ $(n = 1, 2, \ldots)$ of function sequence $\{g_n\}$ $\exists g_{n+p} \in G_{(n,M)}$ satisfying $\theta_{(g_{n+p},g^*)} < \arcsin(1 - \xi - \sin(\theta_{(g^*,e^*)})/2)$. As $\exists k > \tilde{N}_3$ such that $L(n_k) + 1 > M$, according to Lemma II.6 for continuous segment $G_{(n_k+1,L(n_k)+1)} = \{g_{n_k+1}, \ldots, g_{n_k+L(n_k)+1}\}$ $\exists p$, $0 \le p \le L(n_k)$, satisfying (see Fig. 3)

$$\theta_{\left(g_{n_k+p+1},g^*\right)} < \arcsin\left(\frac{1-\xi-\sin\left(\theta_{(g^*,e^*)}\right)}{2}\right). \quad (21)$$

b.4) Let $\theta_{(e_{n_k+p},g_{n_k+p+1})}$ denote the angle formed by e_{n_k+p} and g_{n_k+p+1}. As $\theta_{(e_{n_k+p},g_{n_k+p+1})} \le \theta_{(e_{n_k+p},e^*)} + \theta_{(e^*,g_{n_k+p+1})}$ and $\theta_{(e^*,g_{n_k+p+1})} \le \theta_{(g^*,e^*)} + \theta_{(g_{n_k+p+1},g^*)}$, where $\theta_{(e^*,g_{n_k+p+1})}$ is the angle formed by e^* and g_{n_k+p+1}, we have $\theta_{(e_{n_k+p},g_{n_k+p+1})} \le \theta_{(g^*,e^*)} + \theta_{(e_{n_k+p},e^*)} + \theta_{(g_{n_k+p+1},g^*)}$ (see Fig. 3). Thus, according to (20) and (21), we have $\sin(\theta_{(e_{n_k+p},g_{n_k+p+1})}) \le \sin(\theta_{(g^*,e^*)}) + \sin(\theta_{(e_{n_k+p},e^*)}) + \sin(\theta_{(g_{n_k+p+1},g^*)}) < 1 - \xi$. Furthermore, as $\|e_{n_k+p}\| < r/(1-\xi)$ [see (20)], we have $\|e_{n_k+p+1}\| = \|e_{n_k+p}\| \sin(\theta_{(e_{n_k+p},g_{n_k+p+1})}) < (r/(1-\xi)) * (1-\xi) = r$, which is contradictory to the fact that all $\|e_n\| \ge r$. Thus $r = 0$, that is, $\lim_{n\to+\infty} \|e_n\| = r = 0$. This completes the proof. ■

Remark 3: Seen from Theorem II.1, in order to let the incremental network f_n with random hidden nodes converge to any continuous target function f, one may only need to choose a bounded nonconstant piecewise continuous activation function $g : R \to R$ for additive nodes (such as sigmoidal, threshold, Gaussian, sine, cosine functions) or an integrable piecewise continuous activation function $g : R \to R$ and $\int_R g(x)dx \neq 0$ for RBF nodes (such as Gaussian function). Such activation functions also include some nonregular functions as shown in [4]. It should be noted that all the previous results [1]–[4], [14]–[16], [20]–[29], [34]–[44] are based on the conventional network model where the parameters of hidden nodes need to be tuned during training phase. Furthermore, these previous theoretical results may not be implementable either.

Remark 4: Barron [14] proposed an incremental algorithm taking the form $f_n = \alpha_n f_n + (1-\alpha_n) g_n$ which tries to achieve the nearly minimal value for $\|\alpha_n f_n + (1-\alpha_n) g_n - f\|$ by adjusting the input weights and the bias of the newly added node and then fixing them after tuning. Meir and Maiorov [16] also proposed an incremental algorithm for the Sobolev class of target functions $f \in W_p^r(K)$ instead of any continuous function f discussed in this paper. The Sobolev class of functions $W_p^r(K)$ for any nonnegative integer r is defined as $W_p^r(K) = \{f : \max_{0\le|\vec{k}|\le r} \|(\partial^{|\vec{k}f|}/\partial x_1^{k_1} \cdots \partial x_d^{k_d})\|_{L^p(K)<\infty}\}$ where $|\vec{k}| = k_1 + \cdots + k_d$. The incremental algorithm proposed by Meir and Maiorov [16] takes the form $f_n = (n-1)/n f_n + Q/n h_n$ or $f_n = (n-1)/n f_n + 1/n g_n$ and minimizes the value of $\|f_n - f\|$ by optimizing the input weights and the bias of the newly added node and then fixing them after tuning. Similarly, Romero [30], [31] proposed an incremental algorithm taking the form $f_n = \sum_{i=1}^n \beta_i^n g_i$. β_i^n $(i = 1, \ldots, n)$ and g_n are chosen to minimize $\|f - (f_{n-1} + \mu g)\| + \alpha_n$ ($\forall \mu$ and $\forall g$) where $\alpha_n \ge 0$. Its universal approximation capability was only proved under the hypothesis $\lim_{n\to+\infty} \alpha_n = 0$. It should be noted that all these previous incremental algorithms [14], [16], [30], [31] need to recalculate the output weights of all the existing nodes when a new node is added. Different from these methods, Kwok and Yeung [15] only adjust the parameters (including input weights, bias, and output weights) of the newly added node and then fix all of them. All the parameters (including output weights) of the previous existing nodes will be no longer tuned when a new node is added. Our incremental method randomly generates input weights and biases for newly added nodes instead of tuning them, and the output weights remain fixed once tuned. Unlike the theoretical works done by Barron [14] and Meir and Maiorov [16], which do not give the explicit methods on how to tune all the parameters, our theory indeed gives an efficient learning implementation by obtaining the value for the only parameters (output weights) directly: $\beta_n = \langle e_{n-1}, g_n \rangle / \|g_n\|^2$.

Remark 5: Different from other incremental learning theories [14]–[16] that are proposed for SLFNs with additive nodes, our main Theorem II.1 is also valid for RBF networks. The algorithm proposed by Romero [30], [31] works for both additive nodes and RBF nodes; however, all the parameters need to be tuned in that algorithm.

III. Proposed Incremental Algorithms With Random Additive or RBF Nodes

A. Incremental Algorithm for SLFNs With Random Additive or RBF Nodes

The main objective of this paper is to provide a theoretical justification for SLFNs with randomly generated nodes. To systematically investigate the performance of such networks is basically beyond the scope of this paper. However, we can simply show that the incremental constructive method provided in the above analysis actually works and performs efficiently in real applications.

According to Theorem II.1, when the nth hidden node is added, the weight β_n linking the new node to the output node should be chosen as $\langle e_{n-1}, g_n \rangle / \|g_n\|^2$. In practice, it could not

be calculated since the exact functional form of e_{n-1} is unavailable. Similar to [15, p. 1134], a consistent estimate of the weight β_n based on the training set is

$$\beta_n = \frac{E \cdot H^T}{H \cdot H^T} = \frac{\sum_{p=1}^{N} e(p)h(p)}{\sum_{p=1}^{N} h^2(p)} \tag{22}$$

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. $H = [h(1), \ldots, h(N)]^T$ is the activation vector of the new node for all the N training samples and $E = [e(1), \ldots, e(N)]^T$ is the residual vector before this new hidden node added. In real applications, one may not really wish to get zero approximation error by adding infinite nodes to the network; a maximum number of hidden nodes is normally given. Thus, such an incremental constructive method for SLFNs can be summarized as follows[1]:

Algorithm 1

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 node number $\tilde{N}_{\max}$, and expected learning accuracy ϵ:

Step 1) **Initialization:** Let $\tilde{N} = 0$ and residual error $E = t$, where $t = [t_1, \ldots, t_N]^T$.

Step 2) **Learning step:**

while $\tilde{N} < \tilde{N}_{\max}$ and $\|E\| > \epsilon$

a) increase by one the number of hidden nodes $\tilde{N} : \tilde{N} = \tilde{N} + 1$;

b) assign random input weight $\mathbf{a}_{\tilde{N}}$ and bias $b_{\tilde{N}}$ (or random center $\mathbf{a}_{\tilde{N}}$ and impact factor $b_{\tilde{N}}$) for new hidden node $\tilde{N}$;

c) calculate the output weight $\beta_{\tilde{N}}$ for the new hidden node

$$\beta_{\tilde{N}} = \frac{E \cdot H_{\tilde{N}}^T}{H_{\tilde{N}} \cdot H_{\tilde{N}}^T}; \tag{23}$$

d) calculate the residual error after adding the new hidden node $\tilde{N}$:

$$E = E - \beta_{\tilde{N}} \cdot H_{\tilde{N}}. \tag{24}$$

endwhile

Before learning, there is no node in the network and the residual error is initially set as the expected target vector of the training data set, as shown in Step 1). Learning will stop when the number $\tilde{N}$ of hidden nodes has exceeded the predefined maximum number $\tilde{N}_{\max}$ or the residual error E becomes less than the expected one. The E in the right side of (24) 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 with $E_{\tilde{N}} = f - f_{\tilde{N}} = E_{\tilde{N}-1} - \beta_{\tilde{N}} g_{\tilde{N}}$.

[1]The network with the fixed architecture and randomly assigned hidden nodes is called extreme learning machine (ELM) [7]–[10], [12], [13] where the output parameters are determined by ordinary least square and according to Theorem II.1 $\lim_{n\to\infty} \|f - f_n\| = 0$ holds for ELM. For the sake of consistency, Algorithm 1 directly derived from Theorem II.1 can then be referred as incremental ELM (I-ELM) accordingly as the hidden nodes are increased one by one.

Remark 6: Different from most popular learning algorithms, which require the activation function of hidden nodes differentiable, the proposed incremental learning algorithm is suitable not only for differentiable activation function such as the sigmoidal, Gaussian, sine, and cosine functions but also for nondifferentiable functions such as threshold functions.

B. Incremental Algorithm for TLFNs With Additive Nodes

In order to learn finite training samples, Huang [5] constructs a specific class of TLFNs which need much fewer additive nodes than SLFNs by letting separate learning machines (SLFNs locally trained on different data regions) share common additive nodes in the first hidden layer. In that implementation, the input weights and the biases of the first hidden layers are also randomly given. TLFNs proposed in [5] can approximate those finite training samples with arbitrarily small errors; however, it is still an open issue whether it can universally approximate any continuous functions. Section III-A of this paper proposes an incremental implementation for SLFNs that makes SLFNs universally approximate any continuous functions in any compact subset X of $\mathbf{R}^d$. During this incremental implementation, the input weights and hidden layer biases need only to be randomly given. Based on the constructive method in [5] and the introduced incremental implementation of SLFN in Section III-A, naturally we can construct very compact TLFNs similar to the one proposed by Huang [5] but with the important universal approximation feature.

For simplicity, we first construct a subnetwork which can approximate the following target:

$$\bar{f} = f/C + 0.5 \tag{25}$$

which is a scaled function of the continuous target f and C is a scale factor to make $0 < \bar{f} < 1$.

Similar to [5], the bounded sample input set X can be divided into L subregions $V^{(1)}, \ldots, V^{(L)}$ by setting

$$V^{(p)} = \begin{cases} \{\mathbf{x} | \mathbf{x} \in X \text{ and } b_{p-1} \le \mathbf{a} \cdot \mathbf{x} < b_p\}, & 1 \le p < L \\ \{\mathbf{x} | \mathbf{x} \in X \text{ and } b_{L-1} \le \mathbf{a} \cdot \mathbf{x} \le b_L\}, & p = L \end{cases} \tag{26}$$

where $\mathbf{a}$ is a random vector and $b_0 = \min_{\mathbf{x}\in X}\{\mathbf{a} \cdot \mathbf{x}\}$, $b_L = \max_{\mathbf{x}\in X}\{\mathbf{a} \cdot \mathbf{x}\}$, and $b_p = b_0 + p \cdot ((b_L - b_0)/L)$ is the pth equal partition point of the interval $[b_0, b_L]$. Given a bounded, nonconstant and piecewise continuous activation function $g(x)$ and any small positive value ϵ, for randomly chosen sequence $\{(\mathbf{a}_n, b_n)\}_{n=1}^{\infty}$, according to Theorem II.1, we can construct an SLFN denoted by M_p such that for subregion $V^{(p)}$, when $\beta_n^{(p)}$ between the hidden layer and the output node of M_p is selected as $\langle e_{n-1}, g_n \rangle / \|g_n\|^2$ and the number of additive hidden nodes $\tilde{N}_p$ of M_p is large enough, we have $\|\bar{f}_{\tilde{N}_p} - \bar{f}\|_{\mathbf{x}\in V^{(p)}} < \epsilon$. Obviously, the sigmoidal activation function can be selected for the output layer of all these L SLFNs: $\{M_p\}_{p=1}^{L}$.

We can construct a subnetwork. Let the hidden nodes of the L separate SLFNs (learning machines) M_p link to the same input nodes and the output nodes of these L learning machines construct the output layer of the subnetwork. The activation function used in the output layer is sigmoidal (i.e., $g_{\text{output}}(x) = 1/(1 + e^{-x})$). Furthermore, since the input weights and hidden

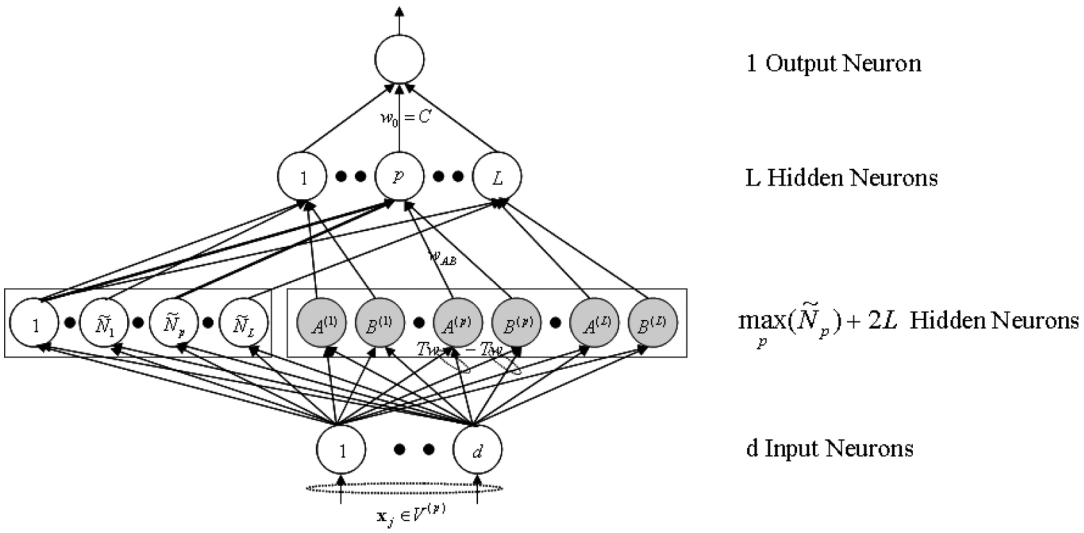

Fig. 4. Compact TLFN constructed by L quantizers and L local learning machines which share common hidden nodes.

layer biases of all the L learning machines (SLFNs) M_p can be chosen randomly, these L learning machines can share hidden nodes. For example, without loss of generality, suppose $\tilde{N}_1 \leq \cdots \leq \tilde{N}_L$. Then we can provide $\tilde{N}_L$ nodes shared by the L learning machines where the first $\tilde{N}_p$ nodes are linked to the pth node in the output layer. Thus, the pth learning machine M_p (pth sub-SLFN) comprises the input nodes, the first $\tilde{N}_p$ nodes in the first hidden layer and the pth nodes in the second hidden layer (see Fig. 4).

Similar to [5], now we can construct the expected TLFN by introducing a quantizer neural module that consists of two sigmoid nodes A and B. First, we can add $2L$ nodes in the hidden layer. These $2L$ nodes are labeled as $A^{(p)}$ and $B^{(p)}$, where $p = 1, \ldots, L$. All the newly added L $A^{(p)}$ nodes and L $B^{(p)}$ nodes are fully connected to the input layer with weights $\mathbf{a}_{A^{(p)}}$ and $\mathbf{a}_{B^{(p)}}, p = 1, \ldots, L$, respectively. However, for each output node, there are only two newly added nodes linking to it, that is, only two nodes $A^{(p)}$ and $B^{(p)}$ of the $2L$ newly added nodes are linked to the pth output node, where $p = 1, \ldots, L$.

The biases of nodes $A^{(p)}$ and $B^{(p)}$ are set as $\overline{b}_{A^{(p)}} = -b_p$ and $\overline{b}_{B^{(p)}} = b_{p-1}$, respectively. The weight vectors of the connections linking the input nodes to the newly added hidden nodes $A^{(p)}$ and $B^{(p)}$ are chosen as $\overline{\mathbf{a}}_{A^{(p)}} = T \cdot \mathbf{a}$ and $\overline{\mathbf{a}}_{B^{(p)}} = -T \cdot \mathbf{a}$, respectively, where neural quantizer factor T is a given positive value. All the weights w_{AB} of the connections linking these newly added $2L$ hidden nodes to their corresponding output nodes are chosen as the same value $w_{AB} = -U$. The neural quantizer modules can inhibit the outputs of those SLFNs and make them nearly zero by adjusting its quantizer factor U if the input does not come from their specialized subregions. Thus, the pth node in the second layer of the TLFN (the output of the pth SLFN) produces the target output if and only if the input comes from the pth subregion, whereas the outputs of the other nodes are inhibited by neural quantizer modules and can be neglected.

We add an output layer with a linear node which links to all those L output nodes (now the second hidden layer nodes) of the above constructed subset network. The scale factor C is simply chosen as the weights between the second hidden layer and the output layer of the finally constructed TLFN, and the bias of the new output layer is $b_o = -0.5C$ (see Fig. 4). The output of the final constructed TLFN can universally approximate the target function f.

Similar to the incremental algorithm for SLFNs, the incremental method for TLFNs can be summarized as follows:

Algorithm 2

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 $g(x)$, number of groups (subregions) L, maximum node number $\tilde{N}_{\max}$, and expected learning accuracy ϵ. Let quantizer factors T and U large enough.

Step 1) **Dividing training sample space into subregions:**

a) Randomly generate a vector $\mathbf{a}$ and reindex the training samples such that $\mathbf{a} \cdot \mathbf{x}_1 < \cdots < \mathbf{a} \cdot \mathbf{x}_N$.

b) Divide the sorted training samples into L groups $G^{(p)}$ each consisting of about N/L training samples.

Step 2) **Initialization:** Set the parameters queue $W = \emptyset$, which stores the input weight vector and hidden bias sequence $\{(\mathbf{a}_1, b_1), (\mathbf{a}_2, b_2), \ldots\}$. Let $|W|$ denote the total number of elements in W.

Step 3) **Learning step:**

for $p = 1$ to L

Let $\tilde{N}_p = 0$ and residual error E initialized with the target vector of the training samples of the pth training group $G^{(p)}$.

while $\tilde{N}_p < \tilde{N}_{\max}$ and $\|E\| > \epsilon$

a) Increase by one the number of hidden nodes $\tilde{N}_p$: $\tilde{N}_p = \tilde{N}_p + 1$.

b) If $\tilde{N}_p \leq |W|$, use the $\tilde{N}_p$th parameters $(\mathbf{a}_{\tilde{N}_p}, b_{\tilde{N}_p})$ in W as the input weight vector and bias of the new hidden node. Otherwise, assign random input weight vector $\mathbf{a}_{\tilde{N}_p}$ and bias $b_{\tilde{N}_p}$ for new hidden node $\tilde{N}_p$, and add the parameter $(\mathbf{a}_{\tilde{N}_p}, b_{\tilde{N}_p})$ into W.

TABLE I
SPECIFICATION OF BENCHMARK DATA SETS

Problems	# Training Data	# Testing Data	# Attributes	
			Continuous	Normal
Abalone	2000	2177	7	1
Auto Price	80	79	14	1
Boston Housing	250	256	13	0
California Housing	8000	12640	8	0
Census (House8L)	10000	12784	8	0
Delta Ailerons	3000	4129	5	0
Delta Elevators	4000	5517	6	0
Machine CPU	100	109	6	0

c) Calculate the output weight $\beta_{\tilde{N}_p}^{(p)}$ for the newly added hidden node of the pth learning SLFN M_p

$$\beta_{\tilde{N}_p}^{(p)} = \frac{E \cdot H_{\tilde{N}_p}^T}{H_{\tilde{N}_p} \cdot H_{\tilde{N}_p}^T}. \tag{27}$$

d) Calculate the residual error after adding the new hidden node $\tilde{N}_p$

$$E = E - \beta_{\tilde{N}_p}^{(p)} \cdot H_{\tilde{N}_p}. \tag{28}$$

endwhile
endfor

In real applications, quantizer factors T and U can be simply set as a very large value which can be considered infinity in the specified computing environments. For example, in many ordinary PCs, T and U can be set as 10^{128} as done in our simulations.

Compared with [5], where only infinite differential activation functions like sigmoidal function have been rigorously proved for such TLFNs, the activation functions that could be used in such TLFNs have been extended very widely in this paper, include almost all popular activation functions which can be used in the application of neural networks.

IV. PERFORMANCE EVALUATION

In this section, the performance of the proposed incremental learning algorithms is compared with other popular learning algorithms on benchmark problems in the function approximation area: 1) approximation of a SinC function and 2) eight real-world function approximation problems from the UCI database [45].[2] The popular learning algorithms to be compared include support vector machine for regression (SVR) [17], [18], stochastic gradient descent backpropagation (BP) [19], and other popular incremental learning algorithms such as RAN [20] and MRAN [22], [23]. The sigmoidal activation function with gain parameter λ : $g(x) = (1/1 + e^{-\lambda x})$ is used in the stochastic gradient BP algorithm. The kernel function used in SVM and our SLFN with RBF nodes are the radial basis function $\phi(\mathbf{x}) = \exp(-\gamma\|\mathbf{x}-\boldsymbol{\mu}\|^2)$. The specification of these benchmark problems is shown in Table I. In our experiments, all the inputs (attributes) have been normalized into the range $[-1, 1]$ while the outputs (targets) have been normalized into [0, 1]. For the case of SLFN with sigmoidal and Sin additive hidden nodes, the input weights $\mathbf{a}_i$ and hidden biases b_i are randomly chosen from the range $[-1, 1]$. For the case of SLFN with hard-limit additive hidden nodes ($g(x) = -1_{x\leq 0} + 1_{x>0}$), the input weights $\mathbf{a}_i$ and hidden biases b_i are randomly chosen from the range $[-0.1, 0.1]$ and $[-1, 1]$, respectively. For SLFN with RBF activation function $\phi(\mathbf{x}) = \exp(-\gamma\|\mathbf{x}-\mu\|^2)$, the centers μ_i are randomly chosen from the range $[-1, 1]$ whereas the impact factor γ is chosen from the range (0, 0.5). For SLFN, our target error is $\epsilon = 0.01$.

All the simulations for BP, RAN, MRAN, and our new incremental learning algorithms are carried out in MATLAB 6.5 environment running in a Pentium 4 2.99 GHz CPU. The simulations for SVM are carried out using compiled C-coded SVM packages: LIBSVM [46] running in the same PC. The basic algorithm of this C-coded SVM packages is a simplification of three works: original SMO by Platt [47], SMO modification by Keerthi *et al.* [48], and SVM$^{\text{Light}}$ by Joachims [49]. It should be noted that a C implementation would be faster than MATLAB by up to 10–50 times for the same applications [50]. So a comparison with LIBSVM gives SVM an advantage. Twenty trials have been conducted for each problem, with training and testing data sets randomly generated for each trial. Simulation results including the average testing root mean square error (RMSE) and the corresponding standard deviation (Dev) are given in this section.

A. Comparison Between the SLFN and Other Popular Incremental Learning Algorithms

The performance of the proposed incremental learning algorithm is first compared with other popular incremental learning algorithms such as RAN and MRAN. Some parameters for BP, RAN, and MRAN need to be manually tuned by trial and error. For both RAN and MRAN, $\eta_{\max} = 1.15$, $\eta_{\min} = 0.04$, $\epsilon = 0.0001$, and $\gamma = 0.999$. For MRAN, some additional parameters are set as $Q = 0.0005$, and growing and pruning thresholds are set as 0.0001. In addition, for RAN and MRAN algorithms, distance parameter $\kappa = 0.3$ is set for Abalone, California Housing, and Census (House8L) and $\kappa = 0.8$ is set for other cases. The allowed maximum number of nodes of SLFNs is $\tilde{N}_{\max} = 300$.

Fig. 5 shows the average testing RMSE decreasing trends and the spent training time of the SLFN with RBF nodes for some data sets. As observed from Fig. 5(a), the testing RMSE is decreasing with the increase of hidden nodes. Seen from Fig. 5(b), the spent learning time is linearly increased with the learning steps since the learning time spent for each step (a new node added) is almost fixed (three fixed-length vector multiplication operations only). For the sake of readability, only partial simulation results are shown in Fig. 5. In fact, similar learning curves can also be plotted for the SLFN with a different type of nodes trained on different data sets.

During our simulations it is found that without further increasing hidden nodes, the SLFN with 200 hidden nodes can generally obtain good performance for all the tested cases which are comparable to the results obtained by RAN and MRAN. Table II shows the average testing RMSE and the corresponding Dev of the proposed SLFN with 200 hidden nodes, RAN and MRAN. As observed from Table II, except for the California Housing data set, the generalization performance of the proposed SLFN algorithm with different activation functions is generally better than or comparable to RAN and MRAN. The

[2]Auto Price data set, which was originally from [45], can be downloaded from http://www.liacc.up.pt/~ltorgo/Regression/price.html

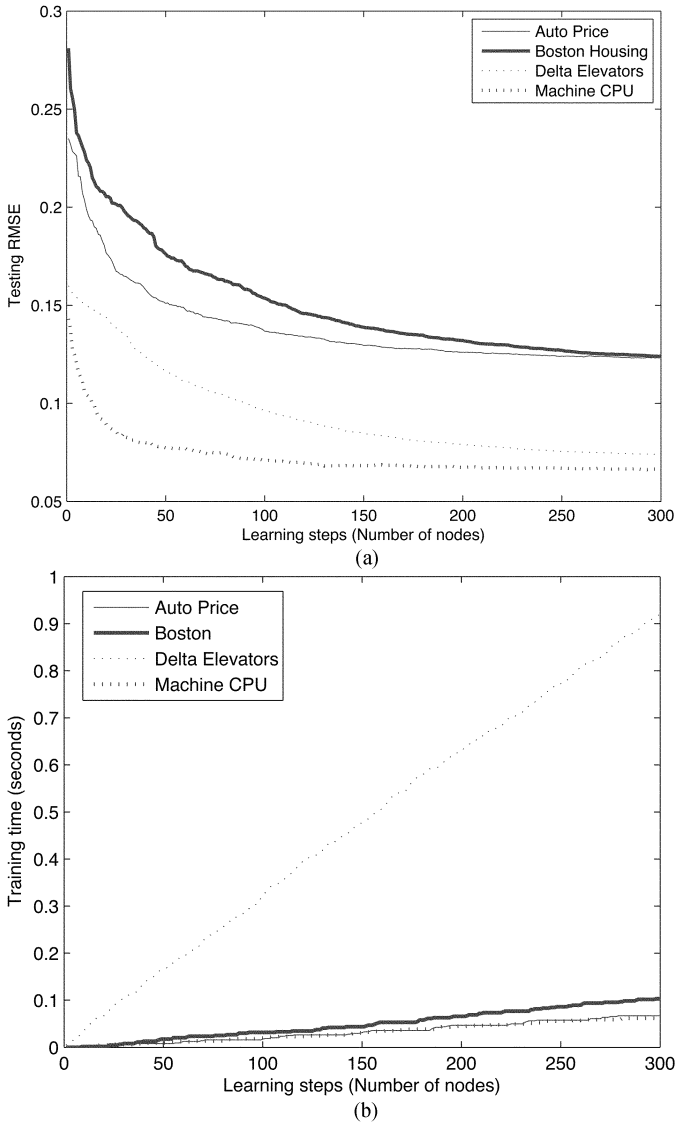

Fig. 5. Performance of the proposed SLFN learning algorithm with RBF nodes on some benchmark problems: (a) Average testing RMSE and (b) training time (seconds).

training time spent for each learning algorithm is shown in Table III. It can be seen that the proposed SLFN algorithm runs much faster than other sequential learning algorithms. The network architectures of RAN and MRAN are automatically determined through their own internal growing and/or pruning mechanism, which is also shown in Table III. Different from RAN and our SLFN algorithms, MRAN can prune insignificant nodes based on some criterion. Thus, among all algorithms, MRAN obtains the most parsimonious network architectures. Among all the algorithms, RAN obtains the largest size of networks for many tested cases excepted for small data sets.

B. Comparison Between the SLFN Algorithm With SVR and Stochastic Gradient Descent BP

The proposed SLFN algorithm is also compared with the popular function regression algorithms such as SVR and stochastic gradient descent BP. Compared with other methods, SVR can usually obtain higher generalization performance. Thus, the allowed maximum number of nodes for SLFN is increased to 500 in this comparison. The sigmoidal activation function is used in the SLFN. In order to get the best generalization performance, the cost parameter C and kernel parameter γ of SVR need to be chosen appropriately. Similar to [51], we estimate the generalized accuracy using different combinations of cost and kernel parameters $(C,\gamma) : C \in \{2^{12}, 2^{11}, \ldots, 2^{-1}, 2^{-2}\}$ and $\gamma \in \{2^{4}, 2^{3}, \ldots, 2^{-9}, 2^{-10}\}$. Therefore, for each problem we try $15 \times 15 = 225$ combinations of parameters (C,γ) for SVM and the average performances over 20 trials are obtained for each combination of parameters (C,γ). The average performance (testing RMSE, the corresponding Dev, and the training time) of SLFN and SVR are shown in Table IV. Table IV also shows the number of support vectors obtained by SVR and the optimal values of the cost and kernel parameters (C,γ). The learning rate η and the momentum constant m of BP are set as $\eta = 0.06$ and $m = 0.8$. The number of nodes assigned to stochastic gradient descent BP algorithm is decided by trial and error in order to obtain as good a generalization performance as possible. During our simulations, 15 nodes and 30 nodes are assigned to BP when tested on Auto Price and Census (House8L) data sets, respectively, and ten nodes are assigned to BP tested on other data sets. It can be seen that the generalization performance of SLFN is close to that of SVR in most cases. Among eight data sets, the proposed SLFN obtains much better generalization performance than BP in four data sets and almost same generalization performance in the other four data sets. Compared to the results of RAN and MRAN shown in Table II, the proposed SLFN with 500 sigmoidal nodes produces better generalization performance than RAN and MRAN in all cases except the California Housing data set. For medium or large applications, SVR spent much more time on training than SLFN.

C. Performance Evaluation of the SLFN Algorithm for Threshold Networks

The widely BP learning algorithm and its variants cannot be used to train the threshold neural networks (SLFN with threshold hidden activation function) directly as the threshold functions are nondifferentiable. One method is to approximate threshold activation function using sigmoid activation function with higher gain parameters λ. For example, a threshold network may be approximated by a sigmoid network with $\lambda = 10$. One may then train such an approximated threshold network using BP instead of training the threshold network directly [10]. The performance comparison between our SLFN algorithm for threshold network and the stochastic gradient descent BP for sigmoid network with $\lambda = 10$ has been conducted. There are 500 threshold nodes assigned to the SLFN. The average performance of our SLFN and stochastic gradient descent BP for threshold networks is shown in Table V. As observed from Table V, our proposed SLFN algorithm overall tends to have better generalization performance than the approximated threshold network trained by BP. It should be noted that the stochastic gradient descent BP just trains an approximated threshold network instead of a true threshold network, whereas our SLFN algorithm provides a method to train the true threshold network directly. Fig. 6 shows the testing RMSE decreasing trends for some data sets.

TABLE II
AVERAGE TESTING RMSE AND THE CORRESPONDING STANDARD DEVIATIONS (DEV) OF DIFFERENT ALGORITHMS: SLFN (WITH 200 NODES) WITH DIFFERENT ACTIVATION FUNCTIONS (SIGMOID, SIN, AND RBF), AND TWO POPULAR SEQUENTIAL LEARNING ALGORITHMS: RAN AND MRAN

Problems	SLFN						RAN		MRAN	
	Sigmoid		RBF		Sin					
	Mean	Dev	Mean	Dev	Mean	Dev	Mean	Dev	Mean	Dev
Abalone	0.0920	0.0046	0.0938	0.0053	**0.0886**	0.0049	0.1183	0.0076	0.0906	0.0065
Auto Price	**0.0977**	0.0069	0.1261	0.0255	0.1162	0.0179	0.1418	0.0261	0.1373	0.0381
Boston Housing	**0.1167**	0.0112	0.1320	0.0126	0.1404	0.0114	0.1474	0.0177	0.1321	0.0140
California Housing	0.1683	0.0049	0.1731	0.0081	0.1550	0.0052	0.1506	0.0035	**0.1480**	0.0030
Census (House8L)	0.0923	0.0023	0.0922	0.0029	**0.0842**	0.0015	0.1061	0.0038	0.0903	0.0042
Delta Ailerons	**0.0525**	0.0078	0.0632	0.0116	0.0635	0.0090	0.1018	0.0083	0.0618	0.0050
Delta Elevators	0.0740	0.0126	0.0790	0.0123	**0.0739**	0.0065	0.1322	0.0130	0.0807	0.0068
Machine CPU	**0.0504**	0.0079	0.0674	0.0177	0.0665	0.0278	0.1069	0.0246	0.1068	0.0367

TABLE III
TRAINING TIME (SECONDS) AND NETWORK COMPLEXITY COMPARISON OF DIFFERENT ALGORITHMS: SLFN WITH DIFFERENT ACTIVATION FUNCTIONS (SIGMOID, SIN, AND RBF) AND TWO POPULAR SEQUENTIAL LEARNING ALGORITHMS: RAN AND MRAN. IN ALL THESE CASE, 200 NODES ARE INCREMENTALLY ADDED TO SLFNS

Problems	Training time of SLFN			Training Time		# nodes	
	Sigmoid	RBF	Sin	RAN	MRAN	RAN	MRAN
Abalone	0.2214	0.5030	0.1778	39.928	255.84	186.3	67.7
Auto Price	0.0329	0.0468	0.0188	0.3565	2.5015	23.8	22.5
Boston Housing	0.0515	0.0657	0.0470	2.0940	22.767	40.5	36.2
California Housing	0.5448	1.3656	0.3872	3301.7	2701.1	4883.0	93.0
Census (House8L)	0.8667	1.7928	0.5194	5399.0	3805.3	6393.2	77.3
Delta Ailerons	0.2620	0.4327	0.1715	237.96	175.07	1118.1	76.6
Delta Elevators	0.2708	0.6321	0.2261	661.78	331.75	2417.4	76.8
Machine CPU	0.0234	0.0447	0.0297	0.1735	0.2454	6.9	7.0

TABLE IV
PERFORMANCE COMPARISON OF SLFN WITH INCREMENTALLY ADDED 500 NODES, STOCHASTIC GRADIENT DESCENT BP, AND SVR. THE CLOSE GENERALIZATION PERFORMANCE (TESTING RMSE) OBTAINED BY DIFFERENT ALGORITHMS IS UNDERLINED AND THE BEST RESULTS AMONG THREE ALGORITHMS ARE SHOWN IN BOLDFACE

Problems	SLFN[a]			BP[a] ($\lambda = 1$)			SVR[b]				
	Testing RMSE		Training	Testing RMSE		Training	Testing RMSE		Training	# SVs	(C, γ)
	Mean	Dev	Time (s)	Mean	Dev	Time (s)	Mean	Dev	Time (s)		
Abalone	0.0878	0.0032	0.5560	0.1175	0.0095	0.4406	0.0846	0.0013	1.6123	309.84	$(2^{4}, 2^{-6})$
Auto Price	**0.0883**	0.0036	0.0954	0.2383	0.0587	0.0154	0.1052	0.0040	0.0042	21.25	$(2^{8}, 2^{-5})$
Boston Housing	**0.1095**	0.0090	0.1419	0.1882	0.0243	0.0579	0.1155	0.0079	0.0494	46.44	$(2^{4}, 2^{-3})$
California Housing	0.1555	0.0021	1.3763	0.1579	0.0033	2.0307	**0.1311**	0.0011	74.184	2189.2	$(2^{3}, 2^{1})$
Census (House8L)	0.0871	0.0021	1.7295	0.0866	0.0025	2.7814	**0.0683**	0.0013	11.251	810.24	$(2^{1}, 2^{-1})$
Delta Ailerons	0.0472	0.0049	0.7058	0.0459	0.0033	0.6610	0.0467	0.0010	0.6726	82.44	$(2^{3}, 2^{-3})$
Delta Elevators	0.0639	0.0067	0.7296	0.0653	0.0019	0.8830	0.0603	0.0005	1.1210	260.38	$(2^{0}, 2^{-2})$
Machine CPU	**0.0491**	0.0089	0.0765	0.1988	0.0429	0.0206	0.0620	0.0180	0.0018	7.8	$(2^{6}, 2^{-4})$

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

TABLE V
PERFORMANCE COMPARISON BETWEEN THE APPROXIMATED THRESHOLD NETWORK ($\lambda = 10$) TRAINED BY STOCHASTIC GRADIENT DESCENT BP AND THE TRUE THRESHOLD NETWORKS TRAINED BY THE PROPOSED SLFN WITH 500 THRESHOLD NODES: $g(x) = -1_{x<0} + 1_{x\geq 0}$

Problems	Testing RMSE		Dev of Testing RMSE		Training Time (s)	
	Threshold	BP($\lambda = 10$)	Threshold	BP($\lambda = 10$)	Threshold	BP($\lambda = 10$)
Abalone	**0.0951**	0.1332	0.0142	0.0102	0.2908	0.4313
Auto Price	**0.1141**	0.3209	0.0130	0.0665	0.0735	0.0172
Boston Housing	**0.1346**	0.2196	0.0104	0.0279	0.0907	0.0548
California Housing	0.1828	0.1806	0.0179	0.0226	0.8186	1.9548
Census (House8L)	**0.0941**	0.1032	0.0062	0.0068	1.0117	2.7359
Delta Ailerons	0.0790	**0.0400**	0.0397	0.0055	0.3550	0.6375
Delta Elevators	**0.0713**	0.0895	0.0110	0.0090	0.5102	0.8970
Machine CPU	**0.0739**	0.2281	0.0140	0.0479	0.0658	0.0215

D. Comparison Between the SLFN and TLFN Algorithms

The performance of the proposed SLFN and TLFN algorithms has been compared on medium or large problems such as Abalone, California Housing, Census (House8L), Delta Ailerons, and Delta Elevators. The performance of these two algorithms is also evaluated in the approximation of the SinC function

$$y(x) = \begin{cases} \sin(x)/x, & x \neq 0 \\ 1, & x = 0 \end{cases}. \tag{29}$$

TABLE VI
PERFORMANCE COMPARISON OF TLFN WITH 100 SIN NODES AND SLFN WITH 200 SIN NODES. THE TRAINING TIMES OF TLFNS ARE THE TOTAL TIME SPENT ON SEPARATE SLFNS TRAINED ON DIFFERENT SUBREGIONS AND DO NOT INCLUDE THE TIME SPENT ON SORTING TRAINING DATA

Problems	TLFN					SLFN		
	Testing RMSE		Spent Time (s)		L	Testing RMSE		Training
	Mean	Dev	Training	Sorting		Mean	Dev	Time (s)
SinC	**0.0091**	0.0010	0.5078	0.0078	20	0.0096	0.0049	0.6297
Abalone	**0.0866**	0.0040	0.3141	0.0016	20	0.0886	0.0049	0.1778
California Housing	**0.1527**	0.0026	0.2453	0.0031	2	0.1550	0.0052	0.3872
Census (House8L)	**0.0813**	0.0031	0.3937	0.0094	10	0.0842	0.0015	0.5194
Delta Ailerons	**0.0528**	0.0108	0.0891	0.0016	2	0.0635	0.0090	0.1715
Delta Elevators	**0.0719**	0.0039	0.1125	0.0016	2	0.0739	0.0065	0.2261

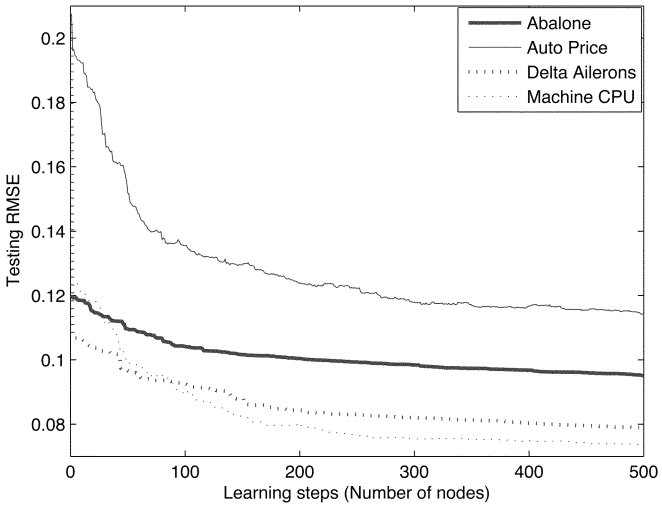

Fig. 6. Performance of the SLFN learning algorithm with 500 hard-limit additive nodes on some benchmark problems: average testing RMSE.

A training set (x_i, y_i) and testing set (x_i, y_i) with 10 000 and 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.

The maximum number of hidden nodes is set as 200 for the SLFN learning algorithm and the maximum number of hidden nodes is set as 100 for the TLFN learning algorithm. The target error is set 0.001 for SinC problem and 0.01 for the rest of the problems. Table VI shows the average performance of these two algorithms. It can be observed that TLFN with 100 nodes can get better generalization performance than SLFN with 200 nodes. The TLFN algorithm runs faster than SLFN in these cases. The training time shown in Table VI is the total time spent for training all SLFNs in all subregions but does not include the time spent for sorting the input data into different L subregions. It can be seen that the sorting time spent in TLFN is only 0.5–2.5% of the training time, which could be ignored.

V. DISCUSSION AND CONCLUSION

This paper first proves in an incremental constructive method that the input weights and hidden layer biases of SLFNs with additive hidden nodes or centers and impact factors of RBF nodes of RBF networks need not be tuned at all. In order to make the incremental network f_n with randomly generated hidden nodes converge to any continuous target function f in a compact subset of $\mathbf{R}^d$, one may only need to choose any bounded nonconstant piecewise continuous activation function $g : R \to R$ for additive nodes or any integrable piecewise continuous activation function $g : R \to R$ and $\int_R g(x)dx \neq 0$ for RBF nodes. It is also noted that all the previous results [1]–[4], [14]–[16], [20]–[29], [34]–[44] are based on the conventional network model where additive or RBF hidden nodes need to be tuned during training. These previous proof methods may not be implementable either. For example, some previous constructive proof methods [14], [37] need to find enough appropriate interpolation or partition points for network construction purposes; however, in real applications, only a finite number of training samples is available and they may not be the required interpolation points for such universal approximation problems. Some incremental algorithms proposed for SLFNs with additive hidden nodes in the literature [14]–[16], [30], [31] require one to find the best input weights and hidden node biases for newly added nodes based on complicated methods as well.

The main aim of this paper is to prove in theory that the SLFNs with randomly generated hidden nodes are actually universal approximators. In fact, the proof itself is a practical incremental constructive method, which actually shows an efficient way to construct an incremental feedforward network. This simple incremental constructive method with different activation functions has been compared with other popular learning algorithms on some benchmark artificial and real-world problems. The simulation results show that our proposed method can reach good generalization performance in a very efficient and simple way. Interestingly, theoretically the learning algorithms derived from our constructive method can be applied to a wide of activation functions no matter whether they are sigmoidal or nonsigmoidal, continuous or noncontinuous, or differentiable or nondifferentiable. The traditional gradient-descent-based learning algorithms cannot be applied to networks with nondifferential activation functions such as threshold networks since the required derivatives are not available. These methods may also face local minima issues. Many popular learning algorithms do not deal with threshold networks directly either and instead use some analog networks to approximate them such that gradient-descent method can finally be used. However, the proposed learning algorithm can be used to train threshold networks directly. A thorough performance evaluation of the proposed algorithms using different activation functions for different applications should be further systematically conducted in future. Although many other incremental learning algorithms

have been proposed in the literature for training RBF networks [20]–[25], unlike our incremental learning algorithms, the universal approximation capability of these previous learning algorithms has not been proved yet. The universal approximation capability of these incremental RBF learning algorithms could be proved based on our proposed theory.

A special TLFN network architecture has also been proposed in this paper, which consists of many learning machines (SLFNs) with additive nodes each locally trained on one subregion of the whole sample input space. Since all these local learning machines can use the same input weights and hidden layer biases, these local learning machines can actually share the same hidden nodes. Compared to[5], which only shows that infinite differentiable sigmoidal functions can be used as activation function in such TLFNs, it has been extended in this paper that a large broad type of activation functions can be used if only they are bounded nonconstant piecewise continuous. It is not difficult to see that for modular networks such as ensemble computing and mixture of local experts [52], [53], one may actually be able to obtain compact networks by allowing those local learning machines to share common hidden nodes. Some researchers [14]–[16], [30], [31] proposed incremental constructive methods for SLFNs that will fix parameters of existing nodes after trained and will not be adjusted when new nodes are added. However, similar compact TLFNs cannot be constructed based on such incremental methods [14]–[16]. Based on these constructive methods [14]–[16], [30], [31], the input weights of each SLFN need to be trained at least one time and different SLFNs have different input weights and hidden biases. Thus different SLFNs cannot share common hidden nodes.

ACKNOWLEDGMENT

The author would like to thank the anonymous associate editor and reviewers for their invaluable suggestions, which have been incorporated to improve the quality of this paper dramatically.

REFERENCES

[1] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Netw.*, vol. 4, pp. 251–257, 1991.
[2] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Netw.*, vol. 6, pp. 861–867, 1993.
[3] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Comput.*, vol. 3, pp. 246–257, 1991.
[4] G.-B. Huang and H. A. Babri, "Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions," *IEEE Trans. Neural Netw.*, vol. 9, no. 1, pp. 224–229, Jan. 1998.
[5] G.-B. Huang, "Learning capability and storage capacity of two-hidden-layer feedforward networks," *IEEE Trans. Neural Netw.*, vol. 14, no. 2, pp. 274–281, Mar. 2003.
[6] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Real-time learning capability of neural networks," *IEEE Trans. Neural Netw.*, vol. 17, no. 4, pp. 863–878, Jul. 2006.
[7] ——, "Extreme learning machine: A new learning scheme of feedforward neural networks," in *Proc. Int. Joint Conf. Neural Networks (IJCNN2004)*, Budapest, Hungary, Jul. 25–29, 2004, vol. 2, pp. 985–990.
[8] ——, "Extreme learning machine: Theory and applications," *Neurocomput.*, to be published.
[9] M.-B. Li, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "Fully complex extreme learning machine," *Neurocomput.*, vol. 68, pp. 306–314, 2005.
[10] G.-B. Huang, Q.-Y. Zhu, K. Z. Mao, C.-K. Siew, P. Saratchandran, and N. Sundararajan, "Can threshold networks be trained directly?," *IEEE Trans. Circuits Syst. II*, vol. 53, no. 3, pp. 187–191, Mar. 2006.
[11] E. Baum, "On the capabilities of multilayer perceptrons," *J. Complex.*, vol. 4, pp. 193–215, 1988.
[12] G.-B. Huang and C.-K. Siew, "Extreme learning machine: {RBF} network case," in *Proc. 8th Int. Conf. Control, Automation, Robotics Vision (ICARCV 2004)*, Kunming, China, Dec. 6–9, 2004, vol. 2, pp. 1029–1036.
[13] ——, "Extreme learning machine with randomly assigned RBF kernels," *Int. J. Inf. Technol.*, vol. 11, no. 1, pp. 16–24, 2005.
[14] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Trans. Inf. Theory*, vol. 39, no. 3, pp. 930–945, Mar. 1993.
[15] T.-Y. Kwok and D.-Y. Yeung, "Objective functions for training new hidden units in constructive neural networks," *IEEE Trans. Neural Netw.*, vol. 8, no. 5, pp. 1131–1148, Sep. 1997.
[16] R. Meir and V. E. Maiorov, "On the optimality of neural-network approximation using incremental algorithms," *IEEE Trans. Neural Netw.*, vol. 11, no. 2, pp. 323–337, Mar. 2000.
[17] V. N. Vapnik, *Statistical Learning Theory*. New York: Wiley, 1998.
[18] A. Smola and B. Schölkopf, A tutorial on support vector regression NeuroCOLT2 Tech. Rep. NC2-TR-1998-030, 1998.
[19] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient BackProp," *Lecture notes in computer science*, vol. 1524, pp. 9–50, 1998.
[20] J. Platt, "A resource-allocating network for function interpolation," *Neural Comput.*, vol. 3, pp. 213–225, 1991.
[21] V. Kadirkamanathan and M. Niranjan, "A function estimation approach to sequential learning with neural networks," *Neural Comput.*, vol. 5, pp. 954–975, 1993.
[22] L. Yingwei, N. Sundararajan, and P. Saratchandran, "A sequental learning scheme for function approximation using minimal radial basis function (RBF) neural networks," *Neural Comput.*, vol. 9, pp. 461–478, 1997.
[23] ——, "Performance evaluation of a sequental minimal radial basis function (RBF) neural network learning algorithm," *IEEE Trans. Neural Netw.*, vol. 9, no. 2, pp. 308–318, Mar. 1998.
[24] G.-B. Huang, P. Saratchandran, and N. Sundararajan, "An efficient sequential learning algorithm for growing and pruning RBF (GAP-RBF) networks," *IEEE Trans. Syst., Man, Cybern. B*, vol. 34, no. 6, pp. 2284–2292, Dec. 2004.
[25] ——, "A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 57–67, Jan. 2005.
[26] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, pp. 359–366, 1989.
[27] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control, Signals Syst.*, vol. 2, no. 4, pp. 303–314, 1989.
[28] K. Funahashi, "On the approximate realization of continuous mappings by neural networks," *Neural Netw.*, vol. 2, pp. 183–192, 1989.
[29] M. Stinchcombe and H. White, "Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions," in *Artificial Neural Networks: Approximation and Learning Theory*, H. White, Ed. Oxford, U.K.: Blackwell, 1992, pp. 29–40.
[30] E. Romero, Function Approximation with SAOCIF: A general sequential method and a particular algorithm with feed-forward neural networks [Online]. Available: http://www.lsi.upc.es/dept/techreps/html/R01-41.html Dept. de Llenguatges i Sistemes Informàtics, Univ. Politècnica de Catalunya, Spain, 2001
[31] ——, "A new incremental method for function approximation using feed-forward neural networks," in *Proc. INNS-IEEE Int. Joint Conf. Neural Networks (IJCNN'2002)*, 2002, pp. 1968–1973.
[32] W. L. Voxman, J. Roy, and H. Goetschel, *Advanced Calculus: An Introduction to Modern Analysis*. New York: Marcel Dekker, 1981.
[33] A. N. Kolmogorov and S. V. Fomin, *Elements of the Theory of Functions and Functional Analysis, Volume 2: Measure, The Lebesgue Integral, Hilbert Space*. Baltimore, MD: Graylock, 1961.
[34] Y. Ito, "Approximation of continuous functions on $\mathbf{R}^d$ by linear combinations of shifted rotations of a sigmoid function with and without scaling," *Neural Netw.*, vol. 5, pp. 105–115, 1992.
[35] A. Gallant and H. White, "There exists a neural network that does not make avoidable mistakes," in *Artificial Neural Networks: Approximation and Learning Theory*, H. White, Ed. Oxford, U.K.: Blackwell, 1992, pp. 5–11.

[36] F. Girosi and T. Poggio, Networks and the best approximation property A.I. Memo 1164, Artificial Intell. Lab., Massachusetts Inst. Tech., 1989.
[37] T. Chen, H. Chen, and R.-W. Liu, "Approximation capability in $C(\overline{\mathbf{R}}^n)$ by multilayer feedforward networks and related problems," *IEEE Trans. Neural Netw.*, vol. 6, no. 1, pp. 25–30, Jan. 1995.
[38] T. Poggio and F. Girosi, A theory of networks for approximation and learning A.I. Memo No. 1140, Artificial Intell. Lab., Massachusetts Inst. Tech., 1989.
[39] V. Kurková, "Kolmogorov's theorem and multilayer neural networks," *Neural Netw.*, vol. 5, pp. 501–506, 1992.
[40] V. Y. Kreinovich, "Arbitrary nonlinearity is sufficient to represent all functions by neural networks: A theorem," *Neural Netw.*, vol. 4, pp. 381–383, 1991.
[41] C.-H. Choi and J. Y. Choi, "Constructive neural networks with piecewise interpolation capabilities for function approximations," *IEEE Trans. Neural Netw.*, vol. 5, no. 6, pp. 936–944, 1994.
[42] Y. Ito, "Approximation of functions on a compact set by finite sums of a sigmoid function without scaling," *Neural Netw.*, vol. 4, pp. 817–826, 1991.
[43] J. Wray and G. G. Green, "Neural networks, approximation theory and finite precision computation," *Neural Netw.*, vol. 8, no. 1, pp. 31–37, 1995.
[44] R. Hecht-Nielsen, "Kolmogorov's mapping neural network existence theorem," in *Proc. Int. Conf. Neural Networks*, 1987, pp. 11–14.
[45] C. Blake and C. Merz, UCI repository of machine learning databases [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html Dept. Inf. Comp. Sci., Univ. California. Irvine, 1998
[46] C.-C. Chang and C.-J. Lin, LIBSVM—A library for support vector machines [Online]. Available: http://www.csie.ntu.edu.tw/~cjlin/libsvm/ Dept. Comp. Sci. Inf. Eng., Nat. Taiwan Univ., Taiwan, R.O.C., 2003
[47] J. Platt, Sequential minimal optimization: A fast algorithm for training support vector machines Microsoft Research Tech. Rep. MSR-TR-98-14, 1998.
[48] S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy, "Improvement to Platt's SMO algorithm for SVM classifier design," *Neural Comput.*, vol. 13, pp. 637–649, 2001.
[49] T. Joachims, SVM$^{\mathrm{light}}$—Support vector machine [Online]. Available: http://svmlight.joachims.org/ Dept. of Computer Science, Cornell Univ. Ithaca, NY, 2003
[50] MathWorks, Compiler execution speed comparison [Online]. Available: http://www.mathworks.com/products/compiler/examples/example2.shtml MathWorks, Inc., 2003
[51] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Trans. Neural Netw.*, vol. 13, no. 2, pp. 415–425, 2002.
[52] M. I. Jordan and R. A. Jacobs, "Hierarchical mixtures of experts and the EM algorithm," *Neural Comput.*, vol. 6, pp. 181–214, 1994.
[53] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive mixtures of local experts," *Neural Comput.*, vol. 3, pp. 79–87, 1991.

Guang-Bin Huang (M'98–SM'04) 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.

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.

His research interests include artificial neural networks, pattern recognition, and machine learning.

Chee-Kheong Siew (M'92) received the B.Eng. degree in electrical engineering from the University of Singapore, Singapore, in 1979, and the M.Sc. degree in communication engineering from Imperial College, London, U.K., in 1987.

He is currently an Associate Professor in the School of Electrical and Electronics Engineering, Nanyang Technological University (NTU), Singapore. From 1995 to 2005, he was Head of the Information Communication Institute of Singapore (ICIS) after he managed the transfer of ICIS to NTU and rebuilt the institute in the university environment. After six years in industry, he joined NTU in 1986 and became 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.