A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks

Nan-Ying Liang, Guang-Bin Huang, *Senior Member, IEEE*, P. Saratchandran, *Senior Member, IEEE*, and N. Sundararajan, *Fellow, IEEE*

Abstract—In this paper, we develop an online sequential learning algorithm for single hidden layer feedforward networks (SLFNs) with additive or radial basis function (RBF) hidden nodes in a unified framework. The algorithm is referred to as online sequential extreme learning machine (OS-ELM) and can learn data one-by-one or chunk-by-chunk (a block of data) with fixed or varying chunk size. The activation functions for additive nodes in OS-ELM can be any bounded nonconstant piecewise continuous functions and the activation functions for RBF nodes can be any integrable piecewise continuous functions. In OS-ELM, the parameters of hidden nodes (the input weights and biases of additive nodes or the centers and impact factors of RBF nodes) are randomly selected and the output weights are analytically determined based on the sequentially arriving data. The algorithm uses the ideas of ELM of Huang *et al.* developed for batch learning which has been shown to be extremely fast with generalization performance better than other batch training methods. Apart from selecting the number of hidden nodes, no other control parameters have to be manually chosen. Detailed performance comparison of OS-ELM is done with other popular sequential learning algorithms on benchmark problems drawn from the regression, classification and time series prediction areas. The results show that the OS-ELM is faster than the other sequential algorithms and produces better generalization performance.

Index Terms—Extreme learning machine (ELM), growing and pruning RBF network (GAP-RBF), GGAP-RBF, minimal resource allocation network (MRAN), online sequential ELM (OS-ELM), resource allocation network (RAN), resource allocation network via extended kalman filter (RANEKF), stochastic gradient descent back-propagation (SGBP).

I. INTRODUCTION

OVER THE past two decades, single hidden layer feedforward neural networks (SLFNs) have been discussed thoroughly by many researchers [1]–[10]. Two main architectures exist for SLFN, namely: 1) those with additive hidden nodes, and 2) those with radial basis function (RBF) hidden nodes. For many of the applications using SLFNs, training methods are usually of batch-learning type. Batch learning is usually a time consuming affair as it may involve many iterations through the training data. In most applications, this may take several minutes to several hours and further the learning parameters (i.e., learning rate, number of learning epochs, stopping criteria, and other predefined parameters) must be properly chosen to ensure convergence. Also, whenever a new data is received batch

learning uses the past data together with the new data and performs a retraining, thus consuming a lot of time. There are many industrial applications where online sequential learning algorithms are preferred over batch learning algorithms as sequential learning algorithms do not require retraining whenever a new data is received.

The back-propagation (BP) algorithm and its variants have been the backbone for training SLFNs with additive hidden nodes. It is to be noted that BP is basically a batch learning algorithm. Stochastic gradient descent BP (SGBP)[11] is one of the main variants of BP for sequential learning applications. In SGBP, network parameters are learned at each iteration on the basis of first-order information of instantaneous value of the cost function using the current training pattern. SGBP suffers from slow training error convergence as large number of training data may be required. To overcome this deficiency, researchers have proposed to use second order information in the network parameter learning process, such as the recursive Levenberg–Marquardt algorithm [12], [13]. Even though second-order methods can shorten the overall convergence time, they may need more time for processing each data and this may pose problems in sequential learning if the data arrives quickly. The network size of SGBP needs to be predefined and fixed.

Sequential learning algorithms have also become popular for feedforward networks with RBF nodes. These include resource allocation network (RAN) [14] and its extensions [15]–[19]. Different from SGBP, the number of RBF hidden nodes in RAN [14] and its variants [15]–[19] is not predefined. RAN [14] and RANEKF [15] determines whether to add a new node based on the novelty of incoming data. Besides growing nodes based on the novelty, MRAN [16], [17], GAP-RBF [18], and GGAP-RBF [19] can also prune insignificant nodes from the networks. RAN, RANEKF, and MRAN require many control parameters to be tuned and in the case of large problems the learning speed may be slow [18], [19]. Although GAP-RBF [18] and GGAP-RBF [19] have tried to simplify the sequential learning algorithms and increase the learning speed, they need the information about the input sampling distribution or input sampling range, and the learning may still be slow for large applications. Also, all the aforementioned sequential learning algorithms handle data one by one only and cannot handle data on a chunk (block of data) by chunk basis. It is worth noting that all the BP-based and RAN-based sequential learning algorithms can only handle specific types of hidden (additive or RBF) nodes and not both.

In this paper, a sequential learning algorithm referred to as online sequential extreme learning machine (OS-ELM) that can handle both additive and RBF nodes in a unified framework

is introduced. OS-ELM can learn the training data not only one-by-one but also chunk-by-chunk (with fixed or varying length) and discard the data for which the training has already been done. It is a versatile sequential learning algorithm in the following sense.

1) The training observations are *sequentially* (one-by-one or chunk-by-chunk with varying or fixed chunk length) presented to the learning algorithm.

2) At any time, only the *newly* arrived single or chunk of observations (instead of the entire past data) are seen and learned.

3) A single or a chunk of training observations is *discarded* as soon as the learning procedure for that particular (single or chunk of) observation(s) is completed.

4) The learning algorithm has no *prior* knowledge as to how many training observations will be presented.

OS-ELM originates from the batch learning extreme learning machine (ELM) [20]–[22], [27], [30] developed for SLFNs with additive and RBF nodes. The performance of ELM has been evaluated on a number of benchmark problems from the function regression and classification areas. Results show that compared with other gradient–descent-based learning algorithms (including BP algorithms) ELM provides better generalization performance at higher learning speed and the learning phase in many applications is completed within seconds [20]–[23].

In OS-ELM with additive nodes, the input weights (of the connections linking the input nodes to hidden nodes) and biases are randomly generated and based on this the output weights are analytically determined. Similarly, in OS-ELM with RBF nodes, the centers and widths of the nodes are randomly generated and fixed and then, based on this, the output weights are analytically determined. Unlike other sequential learning algorithms which have many control parameters to be tuned, OS-ELM only requires the number of hidden nodes to be specified.

The performance of the proposed OS-ELM is evaluated by comparing it with other sequential learning algorithms such as SGBP, RAN, RANEKF, MRAN, GAP-RBF, and GGAP-RBF. Experimental results on benchmark problems from regression, classification, and time-series prediction problems show that the proposed OS-ELM produces better generalization performance at a very fast learning speed. For regression, the benchmark problems considered are three higher dimensional real-world problems from the University of California at Irvine (UCI) machine learning repository [24], i.e.: 1) abalone—determination of the age of abalone using the abalone database; 2) Auto-MPG—determination of the fuel consumption of different models of cars using the Auto-MPG database; 3) California housing—estimation of the median house prices in the California area using California housing database. For classification, the comparison is done based on the following real world benchmark problems [24], i.e.: 1) image segment problem, 2) satellite image problem; and 3) DNA Problem. For time-series prediction, the Mackey–Glass chaotic time series [25] is used.

The paper is organized as follows. Section II gives a brief review of the batch ELM. Section III presents the derivation of OS-ELM. Section IV highlights the difference between OS-ELM and other popular sequential learning algorithms such as SGBP, GAP-RBF, GGAP-RBF, RAN, RANEKF, and MRAN. Performance evaluation of OS-ELM is shown in Section V based on the benchmark problems in the areas of regression, classification, and time-series prediction. Conclusions based on the study are highlighted in Section VI.

II. REVIEW OF ELM

This section briefly reviews the batch ELM developed by Huang *et al.* [20]–[22], [27], [30] to provide the necessary background for the development of OS-ELM in Section III. A brief mathematical description of SLFN incorporating both additive and RBF hidden nodes in a unified way is given first.

A. Mathematical Description of Unified SLFN

The output of an SLFN with $\tilde{N}$ hidden nodes (additive or RBF nodes) can be represented by

$$f_{\tilde{N}}(\mathbf{x}) = \sum_{i=1}^{\tilde{N}} \beta_i G(\mathbf{a}_i, b_i, \mathbf{x}), \qquad \mathbf{x} \in \mathbf{R}^n, \qquad \mathbf{a}_i \in \mathbf{R}^n \quad (1)$$

where $\mathbf{a}_i$ and b_i are the learning parameters of hidden nodes and β_i the weight connecting the ith hidden node to the output node. $G(\mathbf{a}_i, b_i, \mathbf{x})$ is the output of the ith hidden node with respect to the input $\mathbf{x}$. For additive hidden node with the activation function $g(x) : R \to R$ (e.g., sigmoid and threshold), $G(\mathbf{a}_i, b_i, \mathbf{x})$ is given by

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

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 $\mathbf{R}^n$.

For RBF hidden node with activation function $g(x) : R \to R$ (e.g., Gaussian), $G(\mathbf{a}_i, b_i, \mathbf{x})$ is given by

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

where $\mathbf{a}_i$ and b_i are the center and impact factor of ith RBF node. R^+ indicates the set of all positive real values. The RBF network is a special case of SLFN with RBF nodes in its hidden layer. Each RBF node has its own centroid and impact factor, and its output is given by a radially symmetric function of the distance between the input and the center.

B. ELM

In supervised batch learning, the learning algorithms use a finite number of input–output samples for training. For N, arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i) \in \mathbf{R}^n \times \mathbf{R}^m$. Here, $\mathbf{x}_i$ is a $n \times 1$ input vector and $\mathbf{t}_i$ is a $m \times 1$ target vector. If an SLFN with $\tilde{N}$

hidden nodes can approximate these N samples with zero error, it then implies that there exist $\boldsymbol{\beta}_i$, $\mathbf{a}_i$, and b_i such that

$$f_{\tilde{N}}(\mathbf{x}_j) = \sum_{i=1}^{\tilde{N}} \boldsymbol{\beta}_i G(\mathbf{a}_i, b_i, \mathbf{x}_j) = \mathbf{t}_j, \qquad j = 1, \ldots, N. \quad (4)$$

Equation (4) can be written compactly as

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{T} \quad (5)$$

where

$$\mathbf{H}(\mathbf{a}_1, \ldots, \mathbf{a}_{\tilde{N}}, b_1, \ldots, b_{\tilde{N}}, \mathbf{x}_1, \ldots, \mathbf{x}_N)$$
$$= \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_1) & \cdots & G(\mathbf{a}_{\tilde{N}}, b_{\tilde{N}}, \mathbf{x}_1) \\ \vdots & \cdots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_N) & \cdots & G(\mathbf{a}_{\tilde{N}}, b_{\tilde{N}}, \mathbf{x}_N) \end{bmatrix}_{N \times \tilde{N}} \quad (6)$$

$$\boldsymbol{\beta} = \begin{bmatrix} \boldsymbol{\beta}_1^T \\ \vdots \\ \boldsymbol{\beta}_{\tilde{N}}^T \end{bmatrix}_{\tilde{N} \times m} \quad \text{and} \quad \mathbf{T} = \begin{bmatrix} \mathbf{t}_1^T \\ \vdots \\ \mathbf{t}_N^T \end{bmatrix}_{N \times m}. \quad (7)$$

$\mathbf{H}$ is called the hidden layer output matrix of the network [26]; the ith column of $\mathbf{H}$ is the ith hidden node's output vector with respect to inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$ and the jth row of $\mathbf{H}$ is the output vector of the hidden layer with respect to input $\mathbf{x}_j$.

The ELM algorithm is based on the following two principles.

1) When the number of training samples equals the number of hidden nodes, i.e., $N = \tilde{N}$, one can randomly assign the parameters of hidden nodes (the input weights and biases for additive hidden nodes or the centers and impact factors for RBF) and based on this analytically calculate the output weights by simply inverting $\mathbf{H}$ and realize zero training error. Calculation of the output weights is done in a single step here. There is no need for any lengthy training procedure where the network parameters are adjusted interactively with appropriately chosen control parameters (learning rate and learning epochs, etc.).

2) When the number of training samples is greater than the number of hidden nodes, i.e., $N > \tilde{N}$, one can still randomly assign the parameters of hidden nodes and calculate the output weights by using a pseudoinverse of $\mathbf{H}$ to give a small nonzero training error $\epsilon > 0$. Here also the output weights' calculation is done in a single step and does not need lengthy training procedure. These have been formally stated in the following theorems [27].

Theorem II.1: Let an SLFN with $\tilde{N}$ additive or RBF hidden nodes and an activation function $g(x)$ which is infinitely differentiable in any interval of R be given[1] Then, for $\tilde{N}$ arbitrary distinct input vectors $\{\mathbf{x}_i \,|\, \mathbf{x}_i \in \mathbf{R}^n, i = 1, \ldots, \tilde{N}\}$, and $\{(\mathbf{a}_i, b_i)\}_{i=1}^{\tilde{N}}$ randomly generated with any continuous probability distribution, respectively, the hidden layer output matrix $\mathbf{H}$ is invertible with probability one.

Proof: Instead of repeating the rigorous proof provided by Huang *et al.* [27], the basic idea of the proof can be summarized as follows.

[1]Detail discussions on threshold networks have been given in [22].

Let us consider a vector $\mathbf{c}(b_i) = [G(\mathbf{a}_i, b_i, \mathbf{x}_1), \ldots, G(\mathbf{a}_i, b_i, \mathbf{x}_{\tilde{N}})]^T$, the ith column of $\mathbf{H}$, in Euclidean space $\mathbf{R}^{\tilde{N}}$, where $b_i \in (d, f)$ and (d, f) is any interval of $\mathbf{R}$. Following the same proof method of Tamura and Tateishi [28, p. 252] and Huang [26, Th. 2.1], it can be easily proved that vector $\mathbf{c}$ does not belong to any subspace whose dimension is less than $\tilde{N}$. Hence, from any interval (d, f) it is possible to choose $\tilde{N}$ bias values $b_1, \ldots, b_N$ for the $\tilde{N}$ hidden neurons such that the corresponding vectors $\mathbf{c}(b_1), \mathbf{c}(b_2), \ldots, \mathbf{c}(b_{\tilde{N}})$ span $\mathbf{R}^{\tilde{N}}$. This means that for any randomly generated $\mathbf{a}_i$ and b_i based on any continuous probability distribution, $\mathbf{H}$ can be made full-rank with probability one. ∎

Theorem II.1 implies that the SLFNs with $\tilde{N}$ randomly generated additive or RBF hidden nodes can learn $\tilde{N}$ distinct samples with zero error. In real applications, the number of hidden nodes $\tilde{N}$ will always be less than the number of training samples N and, hence, the training error cannot be made exactly zero but can approach a nonzero training error ϵ. The following theorem formally states this fact [27].

Theorem II.2: Given any small positive value $\epsilon > 0$ and activation function $g(x) : R \to R$ which is infinitely differentiable in any interval, there exists $\tilde{N} \leq N$ such that for N arbitrary distinct input vectors $\{\mathbf{x}_i \,|\, \mathbf{x}_i \in \mathbf{R}^n, i = 1, \ldots, \tilde{N}\}$, for any $\{(\mathbf{a}_i, b_i)\}_{i=1}^{\tilde{N}}$ randomly generated according to any continuous probability distribution $\|\mathbf{H}_{N \times \tilde{N}} \boldsymbol{\beta}_{\tilde{N} \times m} - \mathbf{T}_{N \times m}\| < \epsilon$ with probability one.

Proof: According to Theorem II.1, for any $\tilde{N} \geq N$, we have $\|\mathbf{H}_{N \times \tilde{N}} \boldsymbol{\beta}_{\tilde{N} \times m} - \mathbf{T}_{N \times m}\| = 0$. Thus, there should exist $\tilde{N} \leq N$ which makes $\|\mathbf{H}_{N \times \tilde{N}} \boldsymbol{\beta}_{\tilde{N} \times m} - \mathbf{T}_{N \times m}\| < \epsilon$. ∎

According to Theorems II.1 and II.2, the hidden node parameters $\mathbf{a}_i$ and b_i (input weights and biases or centers and impact factors) of SLFNs *need not be tuned during training and may simply be assigned with random values.* Equation (5) then becomes a linear system and the output weights $\boldsymbol{\beta}$ are estimated as

$$\hat{\boldsymbol{\beta}} = \mathbf{H}^\dagger \mathbf{T} \quad (8)$$

where $\mathbf{H}^\dagger$ is the Moore–Penrose generalized inverse [29] of the hidden layer output matrix $\mathbf{H}$. There are several ways to calculate the Moore–Penrose generalized inverse of a matrix, including orthogonal projection method, orthogonalization method, iterative method, and singular value decomposition (SVD) [29]. The orthogonal projection method can be used when $\mathbf{H}^T\mathbf{H}$ is nonsingular and $\mathbf{H}^\dagger = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T$. However, $\mathbf{H}^T\mathbf{H}$ may tend to become singular in some applications. Thus orthogonal projection method may not perform well in all applications. The orthogonalization method and iterative method have limitations since searching and iterations are used. The SVD can always be used to calculate the Moore–Penrose generalized inverse of $\mathbf{H}$, and thus is used in the most implementations of ELM.

It should be noted that the aforementioned theorems assume the availability of complete training data and use (8) for calculating the output weights. ELM is thus a batch learning method. Universal approximation capability of ELM has been analyzed

in [30] in an incremental method[2] and it has been shown that single SLFNs with randomly generated additive or RBF nodes with a widespread of piecewise continuous activation functions can universally approximate any continuous target function on any compact subspace of the Euclidean space $\mathbf{R}^n$. In the implementation of ELM, the activation functions for additive nodes can be any bounded nonconstant piecewise continuous functions and the activation functions for RBF nodes can be any integrable piecewise continuous functions.

III. OS-ELM

The batch ELM described previously assumes that all the training data (N samples) is available for training. However, in real applications, the training data may arrive chunk-by-chunk or one-by-one (a special case of chunk) and, hence, the batch ELM algorithm has to be modified for this case so as to make it online sequential.

The output weight matrix $\hat{\boldsymbol{\beta}}$ ($\hat{\boldsymbol{\beta}} = \mathbf{H}^{\dagger}\mathbf{T}$) given in (8) is a least-squares solution of (5). Here, we consider the case where rank($\mathbf{H}$) $= \tilde{N}$ the number of hidden nodes. Under this condition, $\mathbf{H}^{\dagger}$ of (8) is given by

$$\mathbf{H}^{\dagger} = (\mathbf{H}^{\mathbf{T}}\mathbf{H})^{-1}\mathbf{H}^{\mathbf{T}}. \tag{9}$$

This is also called the *left pseudoinverse* of $\mathbf{H}$ from the fact that $\mathbf{H}^{\dagger}\mathbf{H} = \mathbf{I}_{\tilde{\mathbf{N}}}$. If $\mathbf{H}^{\mathbf{T}}\mathbf{H}$ tends to become singular, one can make it nonsingular by choosing smaller network size $\tilde{N}$ or increasing data number N in the initialization phase of OS-ELM. Substituting (9) into (8), $\hat{\boldsymbol{\beta}}$ becomes

$$\hat{\boldsymbol{\beta}} = (\mathbf{H}^{\mathbf{T}}\mathbf{H})^{-1}\mathbf{H}^{\mathbf{T}}\mathbf{T}. \tag{10}$$

Equation (10) is called the *least-squares solution* to $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$. Sequential implementation of the *least-squares solution* of (10) results in the OS-ELM.

Given a chunk of initial training set $\aleph_0 = \{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=1}^{N_0}$ and $N_0 \geq \tilde{N}$, if one considers using the batch ELM algorithm, one

---

[2]Huang *et al.* [30] learn the data in batch mode but the hidden nodes of the network grow one by one. However, the sequential learning algorithm proposed in this paper has fixed network architecture and learns training data in sequential mode.

needs to consider only the problem of minimizing $\|\mathbf{H}_0\boldsymbol{\beta} - \mathbf{T}_0\|$ where

$$\mathbf{H}_0 = \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_1) & \cdots & G(\mathbf{a}_{\tilde{N}}, b_{\tilde{N}}, \mathbf{x}_1) \\ \vdots & \cdots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_{N_0}) & \cdots & G(\mathbf{a}_{\tilde{N}}, b_{\tilde{N}}, \mathbf{x}_{N_0}) \end{bmatrix}_{N_0 \times \tilde{N}}$$

and

$$\mathbf{T}_0 = \begin{bmatrix} \mathbf{t}_1^T \\ \vdots \\ \mathbf{t}_{N_0}^T \end{bmatrix}_{N_0 \times m} . \tag{11}$$

By Theorem II.1, the solution to minimizing $\|\mathbf{H}_0\boldsymbol{\beta} - \mathbf{T}_0\|$ is given by $\boldsymbol{\beta}^{(0)} = \mathbf{K}_0^{-1}\mathbf{H}_0^T\mathbf{T}_0$ where $\mathbf{K}_0 = \mathbf{H}_0^T\mathbf{H}_0$.

Suppose now that we are given another chunk of data $\aleph_1 = \{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=N_0+1}^{N_0+N_1}$, where N_1 denotes the number of observations in this chunk; the problem then becomes minimizing

$$\left\| \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix} \boldsymbol{\beta} - \begin{bmatrix} \mathbf{T}_0 \\ \mathbf{T}_1 \end{bmatrix} \right\| \tag{12}$$

where (13), shown at the bottom of the page, holds.

Considering both chunks of training data sets $\aleph_0$ and $\aleph_1$, the output weight $\boldsymbol{\beta}$ becomes

$$\boldsymbol{\beta}^{(1)} = \mathbf{K}_1^{-1} \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{T}_0 \\ \mathbf{T}_1 \end{bmatrix} \tag{14}$$

where

$$\mathbf{K}_1 = \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix}. \tag{15}$$

For sequential learning, we have to express $\boldsymbol{\beta}^{(1)}$ as a function of $\boldsymbol{\beta}^{(0)}$, $\mathbf{K}_1$, $\mathbf{H}_1$, and $\mathbf{T}_1$ and not a function of the data set $\aleph_0$. Now $\mathbf{K}_1$ can be written as

$$\mathbf{K}_1 = \begin{bmatrix} \mathbf{H}_0^T & \mathbf{H}_1^T \end{bmatrix} \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix}$$
$$= \mathbf{K}_0 + \mathbf{H}_1^T\mathbf{H}_1 \tag{16}$$

and

$$\mathbf{H}_1 = \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_{N_0+1}) & \cdots & G(\mathbf{a}_{\tilde{N}}, b_{\tilde{N}}, \mathbf{x}_{N_0+1}) \\ \vdots & \cdots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_{N_0+N_1}) & \cdots & G(\mathbf{a}_{\tilde{N}}, b_{\tilde{N}}, \mathbf{x}_{N_0+N_1}) \end{bmatrix}_{N_1 \times \tilde{N}}$$

and

$$\mathbf{T}_1 = \begin{bmatrix} \mathbf{t}_{N_0+1}^T \\ \vdots \\ \mathbf{t}_{N_0+N_1}^T \end{bmatrix}_{N_1 \times m} \tag{13}$$

$$
\begin{aligned}
\begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{T}_0 \\ \mathbf{T}_1 \end{bmatrix} &= \mathbf{H}_0^T \mathbf{T}_0 + \mathbf{H}_1^T \mathbf{T}_1 \\
&= \mathbf{K}_0 \mathbf{K}_0^{-1} \mathbf{H}_0^T \mathbf{T}_0 + \mathbf{H}_1^T \mathbf{T}_1 \\
&= \mathbf{K}_0 \boldsymbol{\beta}^{(0)} + \mathbf{H}_1^T \mathbf{T}_1 \\
&= \left(\mathbf{K}_1 - \mathbf{H}_1^T \mathbf{H}_1 \right) \boldsymbol{\beta}^{(0)} + \mathbf{H}_1^T \mathbf{T}_1 \\
&= \mathbf{K}_1 \boldsymbol{\beta}^{(0)} - \mathbf{H}_1^T \mathbf{H}_1 \boldsymbol{\beta}^{(0)} + \mathbf{H}_1^T \mathbf{T}_1. \quad (17)
\end{aligned}
$$

Combining (14) and (17), $\boldsymbol{\beta}^{(1)}$ is given by

$$
\begin{aligned}
\boldsymbol{\beta}^{(1)} &= \mathbf{K}_1^{-1} \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{T}_0 \\ \mathbf{T}_1 \end{bmatrix} \\
&= \mathbf{K}_1^{-1} (\mathbf{K}_1 \boldsymbol{\beta}^{(0)} - \mathbf{H}_1^T \mathbf{H}_1 \boldsymbol{\beta}^{(0)} + \mathbf{H}_1^T \mathbf{T}_1) \\
&= \boldsymbol{\beta}^{(0)} + \mathbf{K}_1^{-1} \mathbf{H}_1^T (\mathbf{T}_1 - \mathbf{H}_1 \boldsymbol{\beta}^{(0)}) \quad (18)
\end{aligned}
$$

where $\mathbf{K}_1$ is given by

$$
\mathbf{K}_1 = \mathbf{K}_0 + \mathbf{H}_1^T \mathbf{H}_1. \quad (19)
$$

Generalizing the previous arguments, as new data arrives, a recursive algorithm for updating the least-squares solution, which is similar to the recursive least-squares algorithm [31], can be written as follows. When $(k + 1)$th chunk of data set

$$
\aleph_{k+1} = \{ (\mathbf{x}_i, \mathbf{t}_i) \}_{i=\left(\sum_{j=0}^{k} N_j\right)+1}^{\sum_{j=0}^{k+1} N_j}
$$

is received, where $k \geq 0$ and N_{k+1} denotes the number of observations in the $(k + 1)$th chunk, we have

$$
\begin{aligned}
\mathbf{K}_{k+1} &= \mathbf{K}_k + \mathbf{H}_{k+1}^T \mathbf{H}_{k+1} \\
\boldsymbol{\beta}^{(k+1)} &= \boldsymbol{\beta}^{(k)} + \mathbf{K}_{k+1}^{-1} \mathbf{H}_{k+1}^T (\mathbf{T}_{k+1} - \mathbf{H}_{k+1} \boldsymbol{\beta}^{(k)})
\end{aligned}
\quad (20)
$$

where (21), shown at the bottom of the page, and

$$
\mathbf{T}_{k+1} = \begin{bmatrix} \mathbf{t}_{\left(\sum_{j=0}^{k} N_j\right)+1}^T \\ \vdots \\ \mathbf{t}_{\sum_{j=0}^{k+1} N_j}^T \end{bmatrix}_{N_{k+1} \times m} \quad (22)
$$

both hold.

$\mathbf{K}_{k+1}^{-1}$ rather than $\mathbf{K}_{k+1}$ is used to compute $\boldsymbol{\beta}^{(k+1)}$ from $\boldsymbol{\beta}^{(k)}$ in (20). The update formula for $\mathbf{K}_{k+1}^{-1}$ is derived using the Woodbury formula [32]

$$
\begin{aligned}
\mathbf{K}_{k+1}^{-1} &= \left(\mathbf{K}_k + \mathbf{H}_{k+1}^T \mathbf{H}_{k+1} \right)^{-1} \\
&= \mathbf{K}_k^{-1} - \mathbf{K}_k^{-1} \mathbf{H}_{k+1}^T (\mathbf{I} + \mathbf{H}_{k+1} \mathbf{K}_k^{-1} \mathbf{H}_{k+1}^T)^{-1} \\
&\quad \times \mathbf{H}_{k+1} \mathbf{K}_k^{-1}. \quad (23)
\end{aligned}
$$

Let $\mathbf{P}_{k+1} = \mathbf{K}_{k+1}^{-1}$, then the equations for updating $\boldsymbol{\beta}^{(k+1)}$ can be written as

$$
\begin{aligned}
\mathbf{P}_{k+1} &= \mathbf{P}_k - \mathbf{P}_k \mathbf{H}_{k+1}^T \left(\mathbf{I} + \mathbf{H}_{k+1} \mathbf{P}_k \mathbf{H}_{k+1}^T \right)^{-1} \mathbf{H}_{k+1} \mathbf{P}_k \\
\boldsymbol{\beta}^{(k+1)} &= \boldsymbol{\beta}^{(k)} + \mathbf{P}_{k+1} \mathbf{H}_{k+1}^T (\mathbf{T}_{k+1} - \mathbf{H}_{k+1} \boldsymbol{\beta}^{(k)}). \quad (24)
\end{aligned}
$$

Equation (24) gives the recursive formula for $\boldsymbol{\beta}^{(k+1)}$.

Remark 1: From (24), it can be seen that the sequential implementation of the least-squares solution (10) is similar to recursive least-squares algorithm in [31]. Hence, all the convergence results of recursive least-squares (RLS) can be applied here.

Remark 2: From the derivation of OS-ELM, it can be seen that OS-ELM and ELM can achieve the same learning performance (training error and generalization accuracy) when $\text{rank}(\mathbf{H}_0) = \tilde{N}$. In order to make $\text{rank}(\mathbf{H}_0) = \tilde{N}$ and $\text{rank}(\mathbf{K}_0) = \tilde{N}$ the number of initialization data N_0 should not be less than the hidden node number $\tilde{N}$.

Now, the OS-ELM can be summarized as follows.

Proposed OS-ELM Algorithm: First, select the type of node (additive or RBF), the corresponding activation function g, and the hidden node number $\tilde{N}$. The data $\aleph = \{ (\mathbf{x}_i, \mathbf{t}_i) \,|\, \mathbf{x}_i \in \mathbf{R}^n, \mathbf{t}_i \in \mathbf{R}^m, i = 1, \ldots \}$ arrives sequentially.

OS-ELM consists of two phases, namely an initialization phase and a sequential learning phase. In the initialization phase, the appropriate matrix $\mathbf{H}_0$ is filled up for use in the learning phase. The number of data required to fill up $\mathbf{H}_0$ should be at least equal to the number of hidden nodes. According to Theorem II.1, $\text{rank}(\mathbf{H}_0) = \tilde{N}$ if the first $\tilde{N}$ training data are distinct. For example, if there are ten nodes, ten training samples are enough. If the first $\tilde{N}$ training data are not distinct, more training data may be required. However, in most cases the number of training data required in the initialization phase can be equal or close to $\tilde{N}$. Following the initialization phase, learning phase commences either on a one-by-one or chunk-by-chunk (with fixed or varying size) basis as desired. Once a data is used, it is discarded and not used any more.

    Step 1) **Initialization Phase:** Initialize the learning using a small chunk of initial training data

$$
\mathbf{H}_{k+1} = \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_{\left(\sum_{j=0}^{k} N_j\right)+1}) & \cdots & G(\mathbf{a}_{\tilde{N}}, b_{\tilde{N}}, \mathbf{x}_{\left(\sum_{j=0}^{k} N_j\right)+1}) \\ \vdots & \cdots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_{\sum_{j=0}^{k+1} N_j}) & \cdots & G(\mathbf{a}_{\tilde{N}}, b_{\tilde{N}}, \mathbf{x}_{\sum_{j=0}^{k+1} N_j}) \end{bmatrix}_{N_{k+1} \times \tilde{N}} \quad (21)
$$

$\aleph_0 = \{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=1}^{N_0}$ from the given training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) \,|\, \mathbf{x}_i \in \mathbf{R}^n, \mathbf{t}_i \in \mathbf{R}^m, i = 1, \dots\}$, $N_0 \geq \tilde{N}$.

  a) Assign random input weights $\mathbf{a}_i$ and bias b_i (for additive hidden nodes) or center $\mathbf{a}_i$ and impact factor b_i (for RBF hidden nodes), $i = 1, \dots, \tilde{N}$.

  b) Calculate the initial hidden layer output matrix $\mathbf{H}_0$

$$\mathbf{H}_0 = \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_1) & \cdots & G(\mathbf{a}_{\tilde{N}}, b_{\tilde{N}}, \mathbf{x}_1) \\ \vdots & \cdots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_{N_0}) & \cdots & G(\mathbf{a}_{\tilde{N}}, b_{\tilde{N}}, \mathbf{x}_{N_0}) \end{bmatrix}_{N_0 \times \tilde{N}}.$$
(25)

  c) Estimate the initial output weight $\boldsymbol{\beta}^{(0)} = \mathbf{P}_0 \mathbf{H}_0^T \mathbf{T}_0$, where $\mathbf{P}_0 = (\mathbf{H}_0^T \mathbf{H}_0)^{-1}$ and $\mathbf{T}_0 = [\mathbf{t}_1, \dots, \mathbf{t}_{N_0}]^T$.

  d) Set $k = 0$.

Step 2) **Sequential Learning Phase:** Present the $(k+1)$th chunk of new observations

$$\aleph_{k+1} = \{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=(\sum_{j=0}^{k} N_j)+1}^{\sum_{j=0}^{k+1} N_j}$$

where N_{k+1} denotes the number of observations in the $(k+1)$th chunk.

  a) Calculate the partial hidden layer output matrix $\mathbf{H}_{k+1}$ for the $(k+1)$th chunk of data $\aleph_{k+1}$, as shown in (26), at the bottom of the page.

  b) Set $\mathbf{T}_{k+1} = [\mathbf{t}_{(\sum_{j=0}^{k} N_j)+1}, \dots, \mathbf{t}_{\sum_{j=0}^{k+1} N_j}]^T$.

  c) Calculate the output weight $\boldsymbol{\beta}^{(k+1)}$

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \mathbf{P}_k \mathbf{H}_{k+1}^T \left(\mathbf{I} + \mathbf{H}_{k+1} \mathbf{P}_k \mathbf{H}_{k+1}^T\right)^{-1} \mathbf{H}_{k+1} \mathbf{P}_k$$
$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} + \mathbf{P}_{k+1} \mathbf{H}_{k+1}^T \left(\mathbf{T}_{k+1} - \mathbf{H}_{k+1} \boldsymbol{\beta}^{(k)}\right). \quad (27)$$

  d) Set $k = k + 1$. Go to Step 2).

Remark 3: The chunk size does not need to be constant, i.e., the number N_{k+1} of training observations in the $(k+1)$th chunk does not need to be the same as N_k. When the training data is received one-by-one instead of chunk-by-chunk $N_{k+1} \equiv 1$, (27) has the following simple format (Sherman–Morrison formula [32]):

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \frac{\mathbf{P}_k \mathbf{h}_{k+1} \mathbf{h}_{k+1}^T \mathbf{P}_k}{1 + \mathbf{h}_{k+1}^T \mathbf{P}_k \mathbf{h}_{k+1}}$$

$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} + \mathbf{P}_{k+1} \mathbf{h}_{k+1} \left(\mathbf{t}_{k+1}^T - \mathbf{h}_{k+1}^T \boldsymbol{\beta}^{(k)}\right) \quad (28)$$

where $\mathbf{h}_{k+1} = [G(\mathbf{a}_1, b_1, \mathbf{x}_{(k+1)}) \cdots G(\mathbf{a}_{\tilde{N}}, b_{\tilde{N}}, \mathbf{x}_{(k+1)})]$, which is the case analyzed and demonstrated in [33].

Remark 4: In order to handle the case where $\mathbf{H}_0$ and/or $\mathbf{I} + \mathbf{H}_{k+1} \mathbf{P}_k \mathbf{H}_{k+1}^T$ are singular or near singular and to make OS-ELM more robust, $\boldsymbol{\beta}^{(k)}$ and $\mathbf{P}_k$ $(k = 0, \dots)$ can be calculated using SVD as done in all our implementations of OS-ELM.

Remark 5: If $N_0 = N$, then OS-ELM becomes the batch ELM. Thus, batch ELM can be considered as a special case of OS-ELM when all the training observations are present in one learning iteration.

IV. COMPARISON OF OS-ELM WITH OTHER SEQUENTIAL LEARNING ALGORITHMS

In this section, the similarities and differences between OS-ELM and some of the other well-known sequential learning algorithms are presented.

A. One-by-One Versus Chunk-by-Chunk Learning Modes

In real applications, the data presented to the learning algorithm may be one-by-one or chunk-by-chunk where the chunk size may vary. Sequential learning algorithms like SGBP [11], RAN [14], RANEKF [15], MRAN [16], [17], GAP-RBF [18], GGAP-RBF [19], and OS-ELM can be used only in the one-by-one mode. SGBP can in principle work for chunk-by-chunk but difficulty may arise as their learning speed is too slow for them to complete the first chunk before the next one arrives. However, SGBP works well for one-by-one learning mode. OS-ELM works well for one-by-one and also for chunk-by-chunk learning modes. Further, the chunk can be varying and need not be fixed.

B. Selection of Parameters

The control parameters used in the sequential learning algorithms RAN, RANEKF, MRAN, GAP-RBF, and GGAP-RBF include distance parameters $(\epsilon_{\max}, \gamma, \epsilon_{\min})$ and impact factor adjustment parameter κ. Besides these, MRAN uses some growing and pruning parameters. GAP-RBF and GGAP-RBF need an estimate of the input sampling distributions or ranges of sampling areas. (Refer to [19] for details.)

For SGBP, the algorithm control parameters are network size, learning rate and momentum constant, and they need to be adjusted depending on each problem. The only control parameter to be selected for OS-ELM is the size $\tilde{N}$ of the network.

$$\mathbf{H}_{k+1} = \begin{bmatrix} G\left(\mathbf{a}_1, b_1, \mathbf{x}_{\left(\sum_{j=0}^{k} N_j\right)+1}\right) & \cdots & G\left(\mathbf{a}_{\tilde{N}}, b_{\tilde{N}}, \mathbf{x}_{\left(\sum_{j=0}^{k} N_j\right)+1}\right) \\ \vdots & \cdots & \vdots \\ G\left(\mathbf{a}_1, b_1, \mathbf{x}_{\sum_{j=0}^{k+1} N_j}\right) & \cdots & G\left(\mathbf{a}_{\tilde{N}}, b_{\tilde{N}}, \mathbf{x}_{\sum_{j=0}^{k+1} N_j}\right) \end{bmatrix}_{N_{k+1} \times \tilde{N}} \quad (26)$$

TABLE I
SPECIFICATION OF BENCHMARK DATA SETS

Dataset	# Attributes	# Classes	# Training Data	# Testing Data
Auto-MPG	7	-	320	72
Abalone	8	-	3,000	1,177
California Housing	8	-	8,000	12,640
Image Segment	19	7	1,500	810
Satellite Image	36	6	4,435	2,000
DNA	180	3	2,000	1,186
Mackey Glass	4	-	4,000	500

C. Activation Function Types

RAN, RANEKF, MRAN, GAP-RBF, and GGAP-RBF use only RBF nodes and SGBP uses only additive hidden nodes. However, OS-ELM can work for both additive and RBF hidden nodes. Unlike SGBP which is gradient–descent-based and can only work for differentiable activation function, OS-ELM can work for nondifferentiable activation functions as well. In OS-ELM, feedforward networks with additive hidden nodes and RBF networks have been unified in the sense that they can be trained sequentially with the same learning algorithm.

V. PERFORMANCE EVALUATION OF OS-ELM

The performance of OS-ELM is evaluated on the benchmark problems described in Table I which includes three regression applications (auto-MPG, abalone, California housing) [24], three classification applications (image segment, satellite image, DNA) [24] and one time series prediction application [25]. OS-ELM is first compared with other popular sequential learning algorithms such as SGBP, RAN, RANEKF, MRAN, GAP-RBF, and GGAP-RBF in one-by-one learning mode and then the performance evaluation of OS-ELM in chunk-by-chunk learning mode is conducted. All the simulations have been conducted in MATLAB 6.5 environment running on an ordinary PC with 3.0 GHZ CPU. Both the Gaussian RBF activation function $G(\mathbf{a}, b, \mathbf{x}) = \exp(-\|\mathbf{x} - \mathbf{a}\|^2/b)$ and the sigmoidal additive activation function $G(\mathbf{a}, b, \mathbf{x}) = (1)/(1 + \exp(-(\mathbf{a} \cdot \mathbf{x} + b)))$ have been used in the simulations of ELM[3] and OS-ELM. The Gaussian RBF activation function is also used in the simulations of RAN, RANEKF, MRAN, GAP-RBF, and GGAP-RBF and the sigmoidal activation function used in SGBP. In our simulations, the input and output attributes of regression applications are normalized into the range [0,1] while the input attributes of classification applications are normalized into the range [-1,1]. For both ELM and OS-ELM with additive hidden nodes, the input weights and biases are randomly chosen from the range [-1,1]. For both ELM and OS-ELM with RBF hidden nodes the centers are randomly chosen from the range [-1,1]. The impact width b is chosen from the range [0.2,4.2] for all problems except for OS-ELM in image segment and DNA cases. For these two cases in order to make $\mathbf{H}_0$ nonsingular the range should be [3, 11] and [20, 60], respectively.

A. Model Selection

The estimation of optimal architecture of the network and the optimal learning parameters of the learning algorithm is called

[3]Source codes and some references of ELM can be found at www.ntu.edu.sg/home/egbhuang/

model selection in the literature. It is problem specific and has to be predetermined.

For OS-ELM, only the parameter of the optimal number of hidden units needs to be determined. SGBP requires determination of the optimal number of the hidden units, learning rate, and momentum constant. For RAN, RANEKF, MRAN, GAP-RBF, and GGAP-RBF, control parameters including distance parameters ($\epsilon_{\max}, \gamma, \epsilon_{\min}$), impact factor adjustment parameter κ have to be determined. For MRAN, the growing and pruning thresholds need to be determined. For GGAP-RBF, the input sampling distribution has to be estimated.

For the sake of simplicity, we mainly discuss the procedure of selecting the optimal number of hidden nodes for the proposed OS-ELM algorithm and SGBP. The procedure makes use of the training and validation set methods. The training data set is separated into two nonoverlapped subsets: One for training and the other for validation. The optimal number of hidden units is selected as the one which results in the lowest validation error. An example of model selection of optimal hidden unit number for OS-ELM with sigmoidal activation function [OS-ELM (sigmoid)] and SGBP is shown in Fig. 1 in auto-MPG case. In that figure, the top two curves correspond to training and validation error (averaged over 50 trials) for SGBP and the bottom two curves are for OS-ELM (sigmoid). As observed from Fig. 1, the lowest validation error is achieved when the number of hidden nodes of OS-ELM (sigmoid) and SGBP are within the ranges [15, 35] and [13, 24], respectively. Therefore, one can choose the optimal hidden unit numbers for OS-ELM and SGBP (in auto-MPG case) from these ranges. It can also be seen that RMS error curves for OS-ELM are quite smooth compared to SGBP. It implies that OS-ELM is less sensitive to the network size. Fig. 2 details such behavior in a single trial for OS-ELM with 25 hidden units and for SGBP with 13 hidden units. In addition, the number of training data N_0 for initialization has been taken as $N_0 = \tilde{N} + 50$ for regression problems, $N_0 = \tilde{N} + 100$ for classification problems, and $N_0 = \tilde{N} + 1500$ for times series problems where $\tilde{N}$ is the network size.

In our paper, the optimal number of hidden units for OS-ELM and SGBP has been selected for all benchmark problems. The optimal learning parameters for SGBP and the optimal control parameters for the RAN-based sequential algorithms are also selected.

B. Performance Evaluation of OS-ELM: One-by-One Case

We first evaluate and compare the performance of the proposed OS-ELM with other one-by-one learning mode algorithms: SGBP, RAN, RANEKF, MRAN GAP-RBF, and GGAP-RBF. For each problem, the results are averaged over
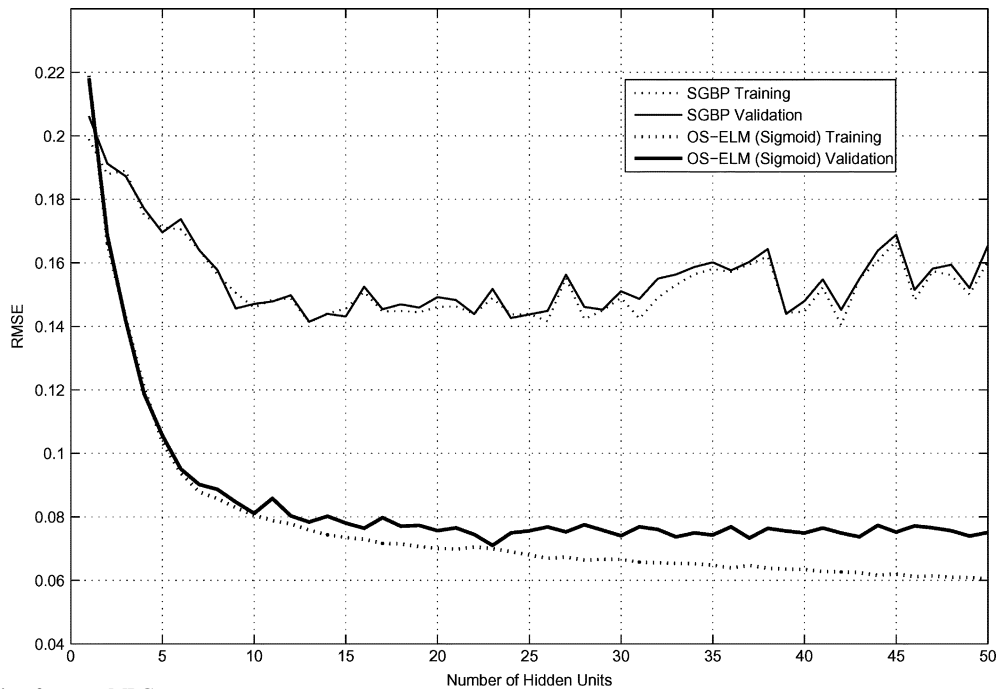
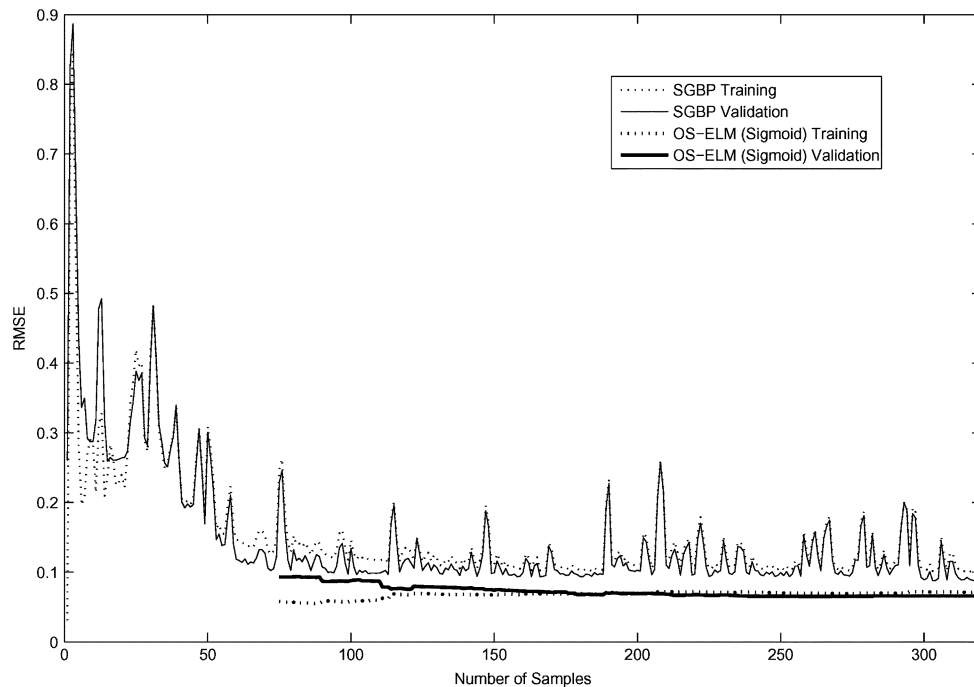Fig. 1.   Model selection for auto-MPG case.

Fig. 2.   Learning evolution for auto-MPG case.

50 trials. The average training time, the average training and testing RMSE for regression and time-series prediction applications and the average training and testing classification rate for classification problems are shown.

1) Regression Problems: Three benchmark problems have been studied here, namely: auto-MPG, abalone, and California housing [24]. The auto-MPG problem is to predict the fuel consumption (miles per gallon) of different models of cars. The abalone problem is the estimation of the age of abalone from the physical measurements and the California housing problem is to predict the median California housing price

based on the information collected using all the block groups in California from the 1990 census. For all the problems studied here, the training and testing data are randomly selected for each trial.

Table II summarizes the results for regression problems in terms of the training time, training RMSE, testing RMSE, and the number of hidden units for each algorithm. The number of hidden units for OS-ELM (sigmoid), OS-ELM (RBF), and SGBP was determined based on the model selection procedure while for RAN, RANEKF, MRAN, and GGAP-RBF it is generated automatically by the algorithms.

TABLE II
COMPARISON BETWEEN OS-ELM AND OTHER SEQUENTIAL ALGORITHMS ON REGRESSION APPLICATIONS

Datasets	Algorithms	Time (seconds)	RMSE		# nodes
			Training	Testing	
Auto-MPG	OS-ELM (Sigmoid)	0.0444	0.0680	0.0745	25
	OS-ELM (RBF)	0.0915	0.0696	0.0759	25
	Stochastic BP	0.0875	0.1112	0.1028	13
	GAP-RBF[18]	0.4520	0.1144	0.1404	3.12
	MRAN[18]	1.4644	0.1086	0.1376	4.46
	RANEKF[18]	1.0103	0.1088	0.1387	5.14
	RAN[18]	0.8042	0.2923	0.3080	4.44
Abalone	OS-ELM (Sigmoid)	0.5900	0.0754	0.0777	25
	OS-ELM (RBF)	1.2478	0.0759	0.0783	25
	Stochastic BP	0.7472	0.0996	0.0972	11
	GAP-RBF[18]	83.784	0.0963	0.0966	23.62
	MRAN[18]	1500.4	0.0836	0.0837	87.571
	RANEKF[18]	90806	0.0738	0.0794	409
	RAN[18]	105.17	0.0931	0.0978	345.58
California Housing	OS-ELM (Sigmoid)	3.5753	0.1303	0.1332	50
	OS-ELM (RBF)	6.9629	0.1321	0.1341	50
	Stochastic BP	1.6866	0.1688	0.1704	9
	GGAP-RBF[19]	115.34	0.1417	0.1386	18
	MRAN[19]	2891.5	0.1598	0.1586	64
	RANEKF[19]	14181	0.0736	0.1495	200
	RAN[19]	3505.2	0.1083	0.1531	3552

TABLE III
COMPARISON BETWEEN OS-ELM AND OTHER SEQUENTIAL ALGORITHMS ON CLASSIFICATION APPLICATIONS

Datasets	Algorithms	Time (seconds)	Accuracy (%)		# nodes
			Training	Testing	
Image Segmentation	OS-ELM (Sigmoid)	9.9981	97.00	94.88	180
	OS-ELM (RBF)	12.197	96.65	94.53	180
	Stochastic BP	2.5776	83.71	82.55	80
	GAP-RBF	1724.3	-	89.93	44.2
	MRAN	7004.5	-	93.30	53.1
Satellite Image	OS-ELM (Sigmoid)	302.48	91.88	88.93	400
	OS-ELM (RBF)	319.14	93.18	89.01	400
	Stochastic BP	3.1415	85.23	83.75	25
	MRAN	2469.4	-	86.36	20.4
DNA	OS-ELM (Sigmoid)	16.742	95.79	93.43	200
	OS-ELM (RBF)	20.951	96.12	94.37	200
	Stochastic BP	1.0840	85.64	82.11	12
	MRAN	6079.0	-	86.85	5

As observed from Table II, the performance of OS-ELM (sigmoid) and OS-ELM (RBF) is similar to each other except that OS-ELM (RBF) requires twice training time taken by OS-ELM (sigmoid). Comparing with other algorithms, we can see that the training time taken by both OS-ELMs and SGBP is much less than RAN, RANEKF, MRAN, GAP-RBF, and GGAP-RBF. However, out of all learning algorithms, OS-ELMs obtained the lowest testing root-mean-square error (RMSE).

2) Classification Problems: For classification studies, three benchmark problems have been considered, namely: image segmentation, satellite image, and DNA [24]. The image segmentation problem consists of a database of images drawn randomly from seven outdoor images and consists of 2310 regions of 3×3 pixels. The aim is to recognize each region into one of the seven categories, namely: brick facing, sky, foliage, cement, window, path, and grass using 19 attributes extracted from each square region. Training and testing data sets are randomly drawn from the database.

The satellite image problem consists of a database generated from landsat multispectral scanner. One frame of landsat multispectral scanner imagery consists of four digital images of the same scene in four different spectral bands. The database is a (tiny) subarea of a scene, consisting of 82×100 pixels. Each data in the database corresponds to a region of 3×3 pixels. The aim is to classify of the central pixel in a region into the six categories, namely: red soil, cotton crop, grey soil, damp grey soil, soil with vegetation stubble, and very damp grey soil using 36 spectral values for each region. The training and test sets are fixed according to [24], but the order of training set is randomly shuffled for each trial.

The database "Primate splice-junction gene sequences (DNA) with associated imperfect domain theory" is known as the DNA problem. Splice junctions are points on a DNA sequence at which "superfluous" DNA is removed during the process of protein creation in higher organisms. The aim of the DNA problem is, given a sequence of DNA, to recognize the boundaries between exons (the parts of the DNA sequence retained after splicing) and introns (the parts of the DNA sequence that are spliced out). This consists of three subtasks: recognizing exon/intron boundaries (referred to as EI sites), intron/exon boundaries (IE sites), and neither (*n* sites). A given sequence of DNA consists of 60 elements (called "nucleotides" or "base-pairs"). The symbolic variables representing nu-

TABLE IV
COMPARISON BETWEEN OS-ELM AND OTHER SEQUENTIAL ALGORITHMS ON MACKEY–GLASS TIME-SERIES APPLICATION

Algorithms	Time (seconds)	Training RMSE	Testing RMSE	# nodes
OS-ELM (Sigmoid)	7.1148	0.0177	0.0183	120
OS-ELM (RBF)	10.0603	0.0184	0.0186	120
GGAP-RBF[19]	24.326	0.0700	0.0368	13
MRAN[19]	57.205	0.1101	0.0337	16
RANEKF[19]	62.674	0.0726	0.0240	23
RAN[19]	58.127	0.1006	0.0466	39

TABLE V
PERFORMANCE COMPARISON OF ELM AND OS-ELM ON REGRESSION APPLICATIONS

Datasets	Activation Functions	Algorithms	Learning Mode	Time (seconds)	RMSE Training	RMSE Testing	# nodes
Auto - MPG	Sigmoid	ELM	Batch	0.0053	0.0697	0.0694	25
		OS-ELM	1-by-1	0.0444	0.0680	0.0745	25
			20-by-20	0.0150	0.0684	0.0738	25
			[10,30]	0.0213	0.0680	0.0765	25
	RBF	ELM	Batch	0.0100	0.0691	0.0694	25
		OS-ELM	1-by-1	0.0915	0.0696	0.0759	25
			20-by-20	0.0213	0.0686	0.0769	25
			[10,30]	0.0250	0.0692	0.0746	25
Abalone	Sigmoid	ELM	Batch	0.0497	0.0763	0.0771	25
		OS-ELM	1-by-1	0.5900	0.0754	0.0777	25
			20-by-20	0.1622	0.0755	0.0780	25
			[10,30]	0.2221	0.0757	0.0778	25
	RBF	ELM	Batch	0.0972	0.0761	0.0770	25
		OS-ELM	1-by-1	1.2478	0.0759	0.0783	25
			20-by-20	0.2253	0.0761	0.0778	25
			[10,30]	0.3057	0.0764	0.0779	25
California Housing	Sigmoid	ELM	Batch	0.5122	0.1306	0.1333	50
		OS-ELM	1-by-1	3.5753	0.1303	0.1332	50
			20-by-20	0.6500	0.1297	0.1333	50
			[10,30]	0.8338	0.1302	0.1327	50
	RBF	ELM	Batch	1.0210	0.1292	0.1312	50
		OS-ELM	1-by-1	6.9629	0.1321	0.1341	50
			20-by-20	0.9794	0.1312	0.1333	50
			[10,30]	1.3241	0.1305	0.1326	50

cleotides were replaced by three binary indicator variables, thus resulting in 180 binary attributes. The training and test sets are also fixed according to [24], but order of training set is randomly shuffled for each trial. During our simulations, RAN and RANEKF produced large number of hidden nodes for problems, which resulted in system memory overflow or large training time. It is also found that for these cases, it is difficult to estimate input sampling distribution for GGAP-RBF algorithm. For the problem satellite image and DNA, no study has been done with GAP-RBF due to the complexity in estimating the input sampling ranges of higher dimensional and binary input cases. Thus, the results of some RAN-based algorithms could not be provided for these cases. As observed from Table III, OS-ELM achieves the best generalization performance with extremely fast learning speed compared with MRAN. Although SGBP complete training at fastest speed in these cases, its generalization performance are much worse than OS-ELM.

3) Time-Series Prediction Problem: The need to time series prediction arises in many real-world problems such as detecting arrhythmia in heartbeats, stock market indices, etc. One of the classical benchmark problems in literature is the chaotic Mackey–Glass differential delay equation given by Mackey and Glass [25]:

$$\frac{dx(t)}{dt} = \frac{ax(t-\tau)}{1+x^{10}(t-\tau)} - bx(t) \qquad (29)$$

for $a = 0.2$, $b = 0.1$, and $\tau = 17$. Integrating the equation over the time interval $[t, t + \Delta t]$ by the trapezoidal rule yields

$$x(t+\Delta t) = \frac{2-b\Delta t}{2+\Delta t}x(t) + \frac{a\Delta t}{2+b\Delta t}\left[\frac{x(t+\Delta t-\tau)}{1+x^{10}(t+\Delta t-\tau)} + \frac{x(t-\tau)}{1+x^{10}(t-\tau)}\right]. \qquad (30)$$

The time series is generated under the condition $x(t-\tau) = 0.3$ for $0 \le t \le \tau$ and predicted with $v = 50$ sample steps ahead using the four past samples: s_{n-v}, s_{n-v-6}, s_{n-v-12}, and s_{n-v-18}. Hence, the nth input–output instance is

$$\mathbf{x}_n = [s_{n-v}, s_{n-v-6}, s_{n-v-12}, s_{n-v-18}]^T$$
$$y_n = s_n.$$

In this simulation, $\Delta t = 1$ and the training observations is from $t = 1$ to 4000 and the testing observations from $t = 4001$ to $t = 4500$.

Performance comparison results are given in Table IV. For this problem, SGBP is excluded because model selection procedure indicated that it is unsuitable for this application. The validation errors were high even after trying on a wide number of hidden units. As observed from Table IV, OS-ELMs achieves the lowest training and testing RMSE as well as the training time.

TABLE VI
PERFORMANCE COMPARISON OF ELM AND OS-ELM ON CLASSIFICATION APPLICATIONS

Datasets	Activation Functions	Algorithms	Learning Mode	Time (seconds)	Accuracy (%)		# nodes
					Training	Testing	
Image Segmentation	Sigmoid	ELM	Batch	0.6384	96.75	95.07	180
		OS-ELM	1-by-1	9.9981	97.00	94.88	180
			20-by-20	1.0922	97.05	94.60	180
			[10,30]	0.9881	97.00	94.92	180
	RBF	ELM	Batch	1.6300	96.22	94.91	180
		OS-ELM	1-by-1	12.197	96.65	94.53	180
			20-by-20	1.4275	96.70	94.55	180
			[10,30]	1.4456	96.75	94.60	180
Satellite Image	Sigmoid	ELM	Batch	7.1816	91.95	88.97	400
		OS-ELM	1-by-1	302.48	91.88	88.93	400
			20-by-20	21.748	91.92	88.86	400
			[10,30]	21.811	91.93	88.90	400
	RBF	ELM	Batch	24.809	92.94	89.03	400
		OS-ELM	1-by-1	319.14	93.18	89.01	400
			20-by-20	23.433	93.19	89.98	400
			[10,30]	24.756	93.16	89.00	400
DNA	Sigmoid	ELM	Batch	0.9748	96.90	94.30	200
		OS-ELM	1-by-1	16.743	95.79	93.43	200
			20-by-20	1.7322	95.87	93.46	200
			[10,30]	1.7875	95.81	93.42	200
	RBF	ELM	Batch	8.2998	95.87	92.33	200
		OS-ELM	1-by-1	20.951	96.12	94.37	200
			20-by-20	2.6538	96.19	94.30	200
			[10,30]	2.8814	96.17	94.25	200

TABLE VII
PERFORMANCE COMPARISON OF ELM AND OS-ELM ON MACKEY–GLASS TIME SERIES-APPLICATION

Activation Functions	Algorithms	Learning Mode	Time (seconds)	RMSE		# nodes
				Training	Testing	
Sigmoid	ELM	Batch	1.1664	0.0183	0.0187	120
	OS-ELM	1-by-1	7.1184	0.0177	0.0183	120
		20-by-20	0.9894	0.0177	0.0183	120
		[10,30]	1.0440	0.0185	0.0190	120
RBF	ELM	Batch	2.1794	0.0185	0.0180	120
	OS-ELM	1-by-1	10.060	0.0184	0.0186	120
		20-by-20	1.5574	0.0183	0.0186	120
		[10,30]	1.7441	0.0184	0.0187	120

C. Performance Evaluation of OS-ELM: Chunk-by-Chunk

Tables V–VII show the performance comparison of ELM implemented in the chunk-by-chunk learning mode. For the chunk-by-chunk learning mode, we have considered both a fixed chunk size of 20 as well as a randomly varying chunk size between 10–30, indicated by [10, 30] in these tables. For the purpose of comparisons, results of chunk size 1 and also the chunk size of the entire training set (the original batch ELM) have also been presented.

For regression problems, it can be seen from Table V that the accuracies obtained by ELM and OS-ELM when implemented in different chunk size modes using different activation functions are nearly the same. This is consistent with the earlier theoretical analysis in Section III. As far as the training time is concerned, the sequential operation of one-by-one takes the highest time followed by the chunk of [10, 30], 20×20, and finally the batch mode. This implies that the one-by-one sequential mode takes the longest while batch the shortest time and any chunk mode operation falls in between. If the chunk size is large, it approaches the time taken for batch mode operation.

Table VI shows similar results for the classification problems. The classification accuracies for ELM and OS-ELM using different activation functions and training in the different modes are similar. It can also be seen that the training time of OS-ELM reduces with increase in the chunk size. For example, in the case of satellite image problem, the training time reduces from 300 s (using one-by-one learning mode) to 20 s (using twenty-by-twenty learning mode). One interesting exception to this is the DNA problem. For this problem, the training time in the chunk mode of twnty-by-twenty and [10,30] is even smaller than batch mode. The reason for this is that: For the batch mode operation, the training data set is large and needs more RAM space. When this space exceeds a limit (for example, cache limit), the operations slow down. This has been verified by the actual experiments executed on higher performance computers.

Table VII shows the comparison results for time series prediction problem. The prediction accuracies for ELM and OS-ELM using different activation function and training in the different modes are close to each other. It should also be noted that the batch mode takes more time than the chunk mode which may be due to the same reasons as seen in DNA problem.

In summary, the comparison results indicate that OS-ELM can be implemented to suit the way the data arrives without sacrificing the accuracy.

VI. CONCLUSION

In this paper, a fast and accurate online sequential learning algorithm (OS-ELM) has been developed for SLFNs with both

additive and RBF hidden nodes in a unified way. The activation functions for additive nodes can be any bounded nonconstant piecewise continuous functions and the activation functions for RBF nodes can be any integrable piecewise continuous functions. Also, the algorithm can handle data arriving or chunk-by-chunk with varying chunk size. Apart from selecting the number of hidden nodes, no other control parameter has to be chosen. Performance of OS-ELM is compared with other well-known sequential learning algorithms on real world benchmark regression, classification and time-series problems. The results indicate that OS-ELM produces better generalization performance with lower training time. Under the mild condition $\text{rank}(\mathbf{H}_0) = \tilde{N}$ the generalization performance of the OS-ELM approaches that of the batch ELM.

ACKNOWLEDGMENT

REFERENCES

[1] D. E. Rumellhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," *Parallel Distrib. Process.: Explanations Microstructure of Cognition*, vol. 1, pp. 318–362, 1986.

[2] S. Ferrari and R. F. Stengel, "Smooth function approximation using neural networks," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 24–38, Jan. 2005.

[3] C. Xiang, S. Q. Ding, and T. H. Lee, "Geometrical interpretation and architecture selection of {MLP}," *IEEE Trans. Neural Netw.*, vol. 16, no. 1, pp. 84–96, Jan. 2005.

[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, Y.-Q. Chen, and H. A. Babri, "Classification ability of single hidden layer feedforward neural networks," *IEEE Trans. Neural Netw.*, vol. 11, no. 3, pp. 799–801, May 2000.

[6] K. Z. Mao and G.-B. Huang, "Neuron selection for RBF neural network classifier based on data structure preserving criterion," *IEEE Trans. Neural Netw.*, vol. 16, no. 6, pp. 1531–1540, Nov. 2005.

[7] J. Park and I. W. Sandberg, "Universal approximation using radial-basis-function networks," *Neural Comput.*, vol. 3, pp. 246–257, 1991.

[8] 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.

[9] 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.

[10] 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.

[11] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, "Efficient backprop," *Lecture Notes Comput. Sci.*, vol. 1524, pp. 9–50, 1998.

[12] L. S. Ngia, J. Sjöberg, and M. Viberg, "Adaptive neural nets filter using a recursive levenberg-marquardt search direction," in *Proc. Asilomar Conf. Signals, Syst., Comput.*, Nov. 1998, vol. 1–4, pp. 697–701.

[13] V. S. Asirvadam, S. F. McLoone, and G. W. Irwin, "Parallel and separable recursive Levenberg-Marquardt training algorithm," in *Proc. 12th IEEE Workshop Neural Netw. Signal Process.*, Sep. 2002, no. 4–6, pp. 129–138.

[14] J. Platt, "A resource-allocating network for function interpolation," *Neural Comput.*, vol. 3, pp. 213–225, 1991.

[15] V. Kadirkamanathan and M. Niranjan, "A function estimation approach to sequential learning with neural networks," *Neural Comput.*, vol. 5, pp. 954–975, 1993.

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

[17] L. Yingwei, N. Sundararajan, and P. Saratchandran, "Performance evaluation of a sequential minimal radial basis function (RBF) neural network learning algorithm," *IEEE Trans. Neural Netw.*, vol. 9, no. 2, pp. 308–318, Mar. 1998.

[18] 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, Cybern.*, vol. 34, no. 6, pp. 2284–2292, Nov. 2004.

[19] G.-B. Huang, P. Saratchandran, and N. Sundararajan, "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.

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

[21] G.-B. Huang and C.-K. Siew, "Extreme learning machine: RBF network case," in *Proc. 8th Int. Conf. Control, Autom., Robot., Vis. (ICARCV 2004)*, Kunming, China, Dec. 6–9, 2004, vol. 2, pp. 1029–1036.

[22] 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, Exp. Briefs*, vol. 53, no. 3, pp. 187–191, Mar. 2006.

[23] 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.

[24] C. Blake and C. Merz, UCI Repository of Machine Learning Databases Dept. Inf. Comp. Sci., Univ. California, Irvine, CA, 1998 [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html

[25] M. C. Mackey and L. Glass, "Oscillation and chaos in physiological control systems," *Science*, vol. 197, pp. 287–289, 1977.

[26] 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.

[27] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theorey and applications," *Neurocomput.*, 2006, to be published.

[28] S. Tamura and M. Tateishi, "Capabilities of a four-layered feedforward neural network: Four layers versus three," *IEEE Trans. Neural Netw.*, vol. 8, no. 2, pp. 251–255, Mar. 1997.

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

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

[31] E. K. P. Chong and S. H. Żak, *An Introduction to Optimization*. New York: Wiley, 2001.

[32] G. H. Golub and C. F. V. Loan, *Matrix Computations*, 3rd ed. Baltimore, MD: The Johns Hopkins Univ. Press, 1996.

[33] G.-B. Huang, N.-Y. Liang, H.-J. Rong, P. Saratchandran, and N. Sundararajan, "On-line sequential extreme learning machine," presented at the IASTED Int. Conf. Comput. Intell. (CI 2005), Calgary, AB, Canada, Jul. 4–6, 2005.

Nan-Ying Liang received the B.Eng. degree in biomedical engineering from Jilin University, Changchun, China, in July 2002. Currently, she is working toward the Ph.D. degree at the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore.

Her research interests include neural networks, biomedical signal processing, and brain computer interfaces.

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, Shenyang, P. R. China, in 1991 and 1994, respectively, and the Ph.D. degree in electrical engineering from Nanyang Technological University, Singapore, in 1999. During undergraduate period, he also concurrently studied in the Wireless Communication Department at Northeastern University.

From June 1998 to May 2001, he worked as a Research Fellow in Singapore Institute of Manufacturing Technology (formerly known as Gintic Institute of Manufacturing Technology), Singapore, where he led/implemented several key industrial projects. From May 2001, he has been working as an Assistant Professor at the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore. His current research interests include machine learning, bioinformatics, and networking.

Dr. Huang is an Associate Editor of *Neurocomputing* and IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS—PART B: CYBERNETICS.

P. Saratchandran (M'87–SM'96) received the Ph.D. degree in the area of control engineering from Oxford University, London, U.K.

He is an Associate Professor with the Nanyang Technological University, Singapore. He has several publications in refereed journals and conferences and has authored four books titled *Fully Tuned RBF Networks for Flight Control* (Kluwer: Norwell, MA, 2002), *Radial Basis Function Neural Networks with Sequential Learning* (World Scientific: Singapore, 1999) *Parallel Architectures for Artificial Neural Networks* (IEEE Press: Piscataway, NJ, 1998), and *Parallel Implementations of Backpropagation Neural Networks* (World Scientific: Singapore, 1996). His interests are in neural networks, bioinformatics, and adaptive control.

Dr. Saratchandran is an Editor for *Neural Parallel and Scientific Computations*. He is listed in the Marquis *Who's Who in the World* and in the *Leaders in the World*, International Biographics Centre, Cambridge, U.K.

N. Sundararajan (S'73–M'74–SM'84–F'96) received the B.E. degree in electrical engineering with first-class honors from the University of Madras, Chepauk, Chennai, India, in 1966, the M.Tech. degree from the Indian Institute of Technology, Madras, India, in 1968, and the Ph.D. degree in electrical engineering from the University of Illinois at Urbana-Champaign, Urbana, in 1971.

From 1972 to 1991, he was working in the Indian Space Research Organization, Trivandrum, India, starting from Control System Designer to Director, Launch Vehicle Design Group, contributing to the design and development of the Indian satellite launch vehicles. He has also worked as a NRC Research Associate at NASA-Ames, Moffett Field, CA, in 1974 and as a Senior Research Associate at NASA-Langley, Hampton, VA, in 1981–86. From February 1991, he has been working at the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore, first as an Associate Professor (from 1991 to August 1999), and presently as a Professor. He was a "Prof. I.G. Sarma Memorial ARDB Professor" (an endowed visiting professor) from November 2002 to February 2003, at the School of Computer Science and Automation, Indian Institute of Science, Bangalore, India. He has published more than 130 papers and also four books titled *Fully Tuned Radial Basis Function neural networks for flight control* (Kluwer: Norwell, MA, 2001), *Radial Basis Function Neural Networks with Sequential Learning* (World Scientific: Singapore, 1999) *Parallel Architectures for Artificial Neural Networks* (IEEE Press: Piscataway, NJ, 1998), and *Parallel Implementations of Backpropagation Neural Networks* (World Scientific: Singapore, 1996). He is listed in *Marquis Who's Who in Science and Engineering*, and *Men of Achievement*, International Biographical Center, Cambridge, U.K. His research interests are in the areas of aerospace control, neural networks, and parallel implementations of neural networks.

Dr. Sundararajan is a an Associate Fellow of American Institute of Aeronautics and Astronautics (AIAA) and also a Fellow of the Institution of Engineers, (IES) Singapore. He was an Associate Editor for IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY, *IFAC Journal on Control Engineering Practice (CEP)*, IEEE ROBOTICS AND AUTOMATION MAGAZINE, and for *Control-Theory and Advanced Technology (C-TAT)*, Japan. He was also a member of the Board of Governors (BoG) for the IEEE Control System Society (CSS) for 2005. He has contributed as a program committee member in a number of international conferences and was the General Chairman for the 6th International Conference on Automation, Robotics, Control, and Computer Vision (ICARCV 2000) held in Singapore in December 2000.