Computers and Mathematics with Applications 60 (2010) 377–389

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

A new online learning algorithm for structure-adjustable extreme learning machine

Guohu Li, Min Liu *, Mingyu Dong

Department of Automation, Tsinghua University and TNLIST, Beijing, 100084, PR China

ARTICLE INFO

Article history:
Received 20 January 2010
Accepted 16 March 2010

Keywords:
Online learning
Extreme learning machine (ELM)
Adjustable structure
Neural network
Modelling

ABSTRACT

In actual industrial fields, data for modelling are usually generated gradually, which requires that the data-based prediction model has the online learning capability. Although many online learning algorithms have been proposed, the generalization performance needs to be improved further. In this paper, a structure-adjustable online learning neural network (SAO-ELM) based on the extreme learning machine (ELM) with quicker learning speed and better generalization performance is proposed. Firstly, ELM is changed into a structure-adjustable learning machine, in which the number of nodes in its single hidden layer can be adjusted. Then, a special strategy is developed to handle the difficulty that the new added hidden nodes' outputs corresponding to the discarded training data cannot be obtained. After that, an iterative equation is presented to update the output matrix when hidden nodes are added. Results of numerical comparison based on data from the real world benchmark problems and an actual continuous casting process show that the performance of SAO-ELM has significant advantages over that of the typical online learning algorithms on generalization performance. In addition, SAO-ELM retains the merit of quick learning characteristic of ELM.

1. Introduction

Neural network is an important nonlinear modelling method, in which the performance relies heavily on the availability of the training data. However, in many practical applications, acquisition of representative training data is expensive and time consuming. Actually, the training data are usually generated gradually in practical industrial fields. In such settings, it is necessary to update the existing neural network in an online fashion to accommodate new data without compromising its performance on old data [1]. To overcome such difficulties, the online learning neural networks are presented [2–7]. From the literature discussing online learning, the requirements for online learning algorithms can be summarized as follows:

(1) All training data are sequentially (one by one) presented to the model.
(2) At any time, only current training data are seen and learned.
(3) A training datum is discarded as soon as the learning procedure on it is completed.
(4) The model does not know how many data will be presented in its learning procedure.

The requirements of the online learning algorithm lead to the so-called *stability–plasticity dilemma* [8]. The stability means the ability of the neural network to preserve the existing knowledge, while the plasticity indicates the capability of accommodating any new information. A typical approach to deal with this dilemma is to save all the arrived data and retrain the neural network using them. In this method, amount of the training data increases as the learning process goes on. On the

* Corresponding author.
E-mail address: lium@tsinghua.edu.cn (M. Liu).

0898-1221/$ – see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.camwa.2010.03.023

one hand, this will result in extra storing space consumption; on the other hand, it makes the time of the learning process becomes longer and longer, which is not suitable for online applications.

In order to deal with the above dilemma more effectively, various online learning algorithms have been proposed. RAN [2] is the first neural network with the ability of online learning. It starts with no hidden nodes and grows through allocating new hidden nodes based on the novelty of the training data that arrive sequentially. A drawback of RAN is that it uses the least mean square method (LMS) to regulate the output weights, which results in long time convergence. Moreover, RAN has no pruning strategy. Consequently, other online learning algorithms, such as RANEKF [3], MRAN [4], GAP-RBF [5], GGAP-RBF [6], are proposed to improve the performance of RAN. The common deficiency of the above algorithms is that too many parameters need to be determined and the training time is too long. These heavily limit their practical applications.

OS-ELM [7], in which only the number of hidden nodes need to be determined, uses the idea of ELM [9–13] and changes the updating way for the output weights in ELM into an online fashion and possesses the online learning ability. It is a promising one for online learning because it originates from ELM, which has been shown to be extremely fast with generalization performance better than other batch learning methods, and holds those merits of ELM. However, the structure of OS-ELM cannot be changed once the learning process starts. This results in relatively weak capability to accommodate new information because there is a paradox between the fixed structure and the various unknown training samples that will arrive in future. Subsequently, GART [14] is proposed and obtains better performance than OS-ELM.

In this paper, a structure-adjustable online learning neural network (SAO-ELM) based on ELM is proposed. In SAO-ELM, the basic network structure is the same as ELM, but the number of the hidden nodes can be adjusted. The main challenge of adding hidden nodes is that the output of these new added nodes on the old data, which have been discarded because of the requirements of the online learning, is unknown. However, the objective of the learning is corresponding to all the arrived training data. Therefore, a sphere is introduced to surround the arrived data, in which the center and radius of the sphere are recorded, and the RBF node is selected as the hidden node. Then the outputs of the new hidden nodes on the discarded data can be treated as zeros if the center and width of the new added RBF nodes are properly determined to satisfy a condition. The condition is that the output of the RBF node for a special point, which locates on the bound of the sphere and is nearest to the center of the RBF node, should be small enough. After that, the above challenge is solved. Then, if the new coming datum is in the outside of this sphere and the training accuracy cannot satisfy the training requirements, a new hidden node is added and its parameters are properly determined. Subsequently, the output weights are regulated with a new iterative method in which the objective is to minimize the empirical risk corresponding to all arrived data. Otherwise, only the output weights are updated. Finally, the center and radius of the sphere are updated according to the new data in order to make the learning process continue.

This paper is organized as follows: Section 2 gives a brief review of ELM and OS-ELM. Section 3 presents the derivation of SAO-ELM, including the methods to select the parameters for the new added hidden node and the equations to update the output weights. Performance evaluation of SAO-ELM is shown in Section 4 based on the benchmark problems in the areas of regression and classification and a practical problem. Conclusions based on the study are highlighted in Section 5.

2. Review of ELM and OS-ELM

OS-ELM is an online learning version of ELM and the idea of SAO-ELM originates from OS-ELM. In order to provide the necessary background for the development of SAO-ELM in Section 3, this section will briefly review the batch ELM and OS-ELM. The mathematical description of Single Hidden Layer Feed-forward Network (SLFN) with RBF hidden nodes, ELM, and OS-ELM will be introduced one by one. Details about ELM and OS-ELM are given in [9–13,7], respectively.

2.1. Description of SLFN with RBF hidden nodes

If an SLFN has n input nodes, M hidden RBF nodes, and m output nodes, the output vector and its components can be represented by

$$F = [f_1, f_2, \ldots, f_m]^T \tag{1}$$

$$f_j = \sum_{i=1}^{M} \beta_{ji} G(a_i, b_i, x), \quad x \in R^n, a_i \in R^n, b_i \in R^+, j = 1, \ldots, m \tag{2}$$

where a_i and b_i are the center and impact factor of the ith RBF hidden node respectively and β_{ji} is the weight connecting the jth output node to the ith hidden node. R^+ indicates the set of all positive real values. $G(a_i, b_i, x)$ is the output of the ith hidden node with respect to the input x. For the case of RBF nodes, the function G is a radially symmetric function of the distance between the input and the center, namely it can be represented by a function $g(x) : R \to R$ as $g(b_i \|x - a_i\|)$. If the Gaussian function is used as the activation function of the hidden node, $G(a_i, b_i, x)$ can be written as

$$G(a_i, b_i, x) = e^{b_i \|x - a_i\|}. \tag{3}$$

2.2. Review of ELM

ELM is a special kind of SLFN. In ELM, the hidden nodes can be additive nodes or RBF nodes. The parameters of the hidden nodes are randomly generated. The output weights are analytically determined. For N arbitrary distinct samples $(x_i, t_i) \in R^n \times R^m$, in which x_i is a $n \times 1$ input vector and t_i is a $m \times 1$ output vector, the jth output for the ith sample is

$$f_{ij} = \sum_{k=1}^{M} \beta_{jk} G(a_k, b_k, x_i), \quad i = 1, \ldots, N, j = 1, \ldots, m. \tag{4}$$

Eq. (4) can be written compactly as

$$H_0 \beta = F \tag{5}$$

where

$$H_0 = \begin{bmatrix} G(a_1, b_1, x_1) & \ldots & G(a_M, b_M, x_1) \\ \vdots & \ddots & \vdots \\ G(a_1, b_1, x_N) & \cdots & G(a_M, b_M, x_N) \end{bmatrix}_{N \times M} \tag{6}$$

$$\beta = \begin{bmatrix} \beta_{11} & \ldots & \beta_{1m} \\ \vdots & \ddots & \vdots \\ \beta_{M1} & \cdots & \beta_{Mm} \end{bmatrix} = \begin{bmatrix} \beta_1^T \\ \cdots \\ \beta_M^T \end{bmatrix}_{M \times m} \tag{7}$$

$$F = \begin{bmatrix} f_{11} & \ldots & f_{1m} \\ \vdots & \ddots & \vdots \\ f_{N1} & \cdots & f_{Nm} \end{bmatrix} = \begin{bmatrix} f_1^T \\ \cdots \\ f_N^T \end{bmatrix}_{N \times m}. \tag{8}$$

H_0 is called the hidden layer output matrix of the network [7,15]. The ith column of H_0 is the ith hidden node's output vector with respect to all of the N inputs and the jth row is the outputs of all the hidden nodes to the jth sample.

If the empirical risk minimization (ERM) principle is used, the objective function for training the neural network can be written as

$$\min (\|F - T_0\|) = \min \left(\left\| H_{0(N \times M)} \beta_{(M \times m)} - T_{0(N \times m)} \right\| \right) \tag{9}$$

where T_0 is the matrix of training target, namely

$$T_0 = \begin{bmatrix} t_{11} \cdots t_{1m} \\ \cdots \\ t_{N1} \cdots t_{Nm} \end{bmatrix} = \begin{bmatrix} t_1^T \\ \cdots \\ t_N^T \end{bmatrix}_{N \times m}. \tag{10}$$

In ELM, the number of hidden nodes is determined firstly and so the structure is fixed. Then the widths and centers of those nodes are selected randomly. According to Eq. (9), the network output weight matrix, which denotes as β, becomes the only parameter that needs to be determined. The following two theorems provide possible method to calculate β. These theorems are formally stated in [12].

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

Theorem 2. *Given any small positive value $\varepsilon > 0$ and activation function $g(x) : R \to R$ which is infinitely differentiable in any interval, there exists $M \leq N$ such that for N arbitrary distinct input vectors $\{x_i | x_i \in R^n, i = 1, \ldots, N\}$, for any $\{(a_i, b_i)\}_{i=1}^{M}$ randomly generated according to any continuous probability distribution $\|H_{N \times M} \beta_{M \times m} - T_{N \times m}\| < \varepsilon$ with probability one.*

According to Theorem 1, if $N = M$ in Eq. (9), we can select the parameters of the hidden nodes and regulate the output weight matrix to make the training error be zero. The optimal value of β can be given by

$$\beta_0 = H_0^{-1} T_0. \tag{11}$$

However, in most practical applications, it is unnecessary to allocate so many hidden nodes as the training samples. So the size of training data set, i.e. N, is usually larger than the number of hidden nodes, i.e. M. In such settings, Theorem 2 guarantees that any small training error ε can be obtained with some network that its hidden nodes are less than the number of training samples. The optimal output weight matrix can be estimated as [7]

$$\beta_0 = H_0^{\dagger} T_0 \tag{12}$$

where $H_0^{\dagger}$ is the Moore–Penrose generalized inverse. If the condition of $rank(H) = M$ is satisfied, Eq. (12) can be rewritten as

$$\beta_0 = (H_0^T H_0)^{-1} H_0^T T_0. \tag{13}$$

If Eq. (13) is used, a smaller network size M or a larger initial training data set should be chosen to cater for such condition.

2.3. Review of OS-ELM

As seen from Eq. (12), ELM is a batch learning algorithm which assumes that all of the training data are available before training. In order to handle the problem of online learning in many real applications, OS-ELM is proposed.

Given a chunk of initial training data set $X_0 = \{(x_i, t_i)\}_{i=1}^{N}$, the hidden layer output matrix H_0 and the training target matrix T_0 are the same as Eqs. (6) and (10), respectively. If the number of hidden nodes M is less than the number training samples N, i.e. $M \leq N$, the output matrix β_0 can be given by

$$\beta_0 = K_0^{-1} H_0^T T_0 \tag{14}$$

where

$$K_0 = H_0^T H_0 \tag{15}$$

according to Eq. (13).

If a new chunk of data set $X_1 = \{(x_i, t_i)\}_{i=N+1}^{N+N_1}$ arrives, the hidden layer output matrix and the training target for this set can be written as

$$H_1 = \begin{bmatrix} G(a_1, b_1, x_{N+1}) & \dots & G(a_M, b_M, x_{N+1}) \\ \vdots & \ddots & \vdots \\ G(a_1, b_1, x_{N+N_1}) & \cdots & G(a_M, b_M, x_{N+N_1}) \end{bmatrix}_{N_1 \times M} \tag{16}$$

and

$$T_1 = \begin{bmatrix} t_{(N+1)1} \cdots t_{(N+1)m} \\ \cdots \\ t_{(N+N_1)1} \cdots t_{(N+N_1)m} \end{bmatrix} = \begin{bmatrix} t_{N+1}^T \\ \cdots \\ t_{N+N_1}^T \end{bmatrix}_{N_1 \times m}. \tag{17}$$

Then, the objective function for old data set X_0 and new coming data set X_1 is

$$\min \left(\left\| \begin{bmatrix} H_0 \\ H_1 \end{bmatrix} \beta - \begin{bmatrix} T_0 \\ T_1 \end{bmatrix} \right\| \right). \tag{18}$$

The solution for the minimization problem (18) becomes

$$\beta_1 = (K_1)^{-1} \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}_0^T \begin{bmatrix} T_0 \\ T_1 \end{bmatrix} \tag{19}$$

where

$$K_1 = \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}^T \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}. \tag{20}$$

In order to satisfy the four conditions of online learning defined in Section 1, the matrix β_1 and K_1 should be expressed as a function of β_0, K_0, H_1, and T_1. By some mathematical derivation, such equation can be got as

$$\beta_1 = \beta_0 + K_1^{-1} H_1^T (T_1 - H_1 \beta_0) \tag{21}$$

where

$$K_1 = K_0 + H_1^T H_1. \tag{22}$$

(21) and (22) are the output matrix updating equation for online learning in OS-ELM. Apparently, they satisfy all the requirements in online learning defined in Section 1. Therefore OS-ELM is a real online learning algorithm.

3. Derivation of SAO-ELM

ELM and OS-ELM can accommodate the information in training data using a fixed structure neural network in a batch learning manner and an online sequential learning manner, respectively. However, in online learning circumstances, there is a paradox between the fixed structure and the various unknown training samples that will arrive in future. When the structure of the neural network cannot accommodate the new information in the coming data through regulating its output weights, it is necessary to change the structure. This section will give an online learning algorithm based on ELM in the case that the structure of the network can be adjusted.

3.1. Problem description

Suppose the initial training data set is $X_0 = \{(x_i, t_i)\}_{i=1}^{N}$ and the new coming data set is $X_1 = \{(x_i, t_i)\}_{i=N+1}^{N+N_1}$. If not only the output weights should be updated but also L hidden nodes $\{(a_i, b_i)\}_{i=M+1}^{M+L}$ are added to the network, the training objective function will become

$$\min\left(\left\|\begin{bmatrix} H_0 & H_{01} \\ H_1 & H_{11} \end{bmatrix}\beta - \begin{bmatrix} T_0 \\ T_1 \end{bmatrix}\right\|\right) \tag{23}$$

where H_0 and T_0 are shown in Eqs. (6) and (10) while H_1 and T_1 have been given in Eqs. (16) and (17). In Eq. (23), H_{01} and H_{11} are the output matrices of the new added hidden nodes for data sets X_0 and X_1, respectively. They can be deployed as

$$H_{01} = \begin{bmatrix} G(a_{M+1}, b_{M+1}, x_1) & \dots & G(a_{M+L}, b_{M+L}, x_1) \\ \vdots & \ddots & \vdots \\ G(a_{M+1}, b_{M+1}, x_N) & \cdots & G(a_{M+L}, b_{M+L}, x_N) \end{bmatrix}_{N\times L} \tag{24}$$

and

$$H_{11} = \begin{bmatrix} G(a_{M+1}, b_{M+1}, x_{N+1}) & \dots & G(a_{M+L}, b_{M+L}, x_{N+1}) \\ \vdots & \ddots & \vdots \\ G(a_{M+1}, b_{M+1}, x_{N+N_1}) & \cdots & G(a_{M+L}, b_{M+L}, x_{N+N_1}) \end{bmatrix}_{N_1\times L}. \tag{25}$$

In order to get the solution of β in Eq. (23) satisfying the online learning definition, a corresponding method must be developed.

3.2. Method deduction

Before deduction, we should notice that adding some new properly defined hidden nodes can do benefit to the training. This can be stated formally in Theorem 3 as follows.

Theorem 3. *Suppose the initial training set is $X_0 = \{(x_i, t_i)\}_{i=1}^{N}$ and the new coming data set is $X_1 = \{(x_i, t_i)\}_{i=N+1}^{N+N_1}$. If the ERM principle is used, an ELM with M hidden nodes can get minimal training error*

$$E_0 = \min\left(\left\|\begin{bmatrix} H_0 \\ H_1 \end{bmatrix}\beta - \begin{bmatrix} T_0 \\ T_1 \end{bmatrix}\right\|\right)$$

while an ELM with $M + L$ hidden nodes can realize minimal training error

$$E_1 = \min\left(\left\|\begin{bmatrix} H_0 & H_{01} \\ H_1 & H_{11} \end{bmatrix}\beta - \begin{bmatrix} T_0 \\ T_1 \end{bmatrix}\right\|\right).$$

Then the conclusion $E_1 \le E_0$ can be obtained.

Proof. Denote

$$H = \begin{bmatrix} H_0 \\ H_1 \end{bmatrix} \quad \text{and} \quad H' = \begin{bmatrix} H_0 & H_{01} \\ H_1 & H_{11} \end{bmatrix}.$$

If the matrix β is divided into two blocks from the row direction as the matrix H' has been divided from the column direction, it can be written as

$$\beta = \begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix}.$$

Meanwhile, we should notice that the solution of an optimization problem found in some set must be better than the solution searched in the subset of that particular set. Then

$$\begin{aligned} E_1 &= \min\left(\left\|\begin{bmatrix} H_0 & H_{01} \\ H_1 & H_{11} \end{bmatrix}\beta - \begin{bmatrix} T_0 \\ T_1 \end{bmatrix}\right\|\right) = \min\left(\left\|\begin{bmatrix} H_0 & H_{01} \\ H_1 & H_{11} \end{bmatrix}\begin{bmatrix} \beta_0 \\ \beta_1 \end{bmatrix} - \begin{bmatrix} T_0 \\ T_1 \end{bmatrix}\right\|\right) \le \min\left(\left\|\begin{bmatrix} H_0 & H_{01} \\ H_1 & H_{11} \end{bmatrix}\begin{bmatrix} \beta_0 \\ 0 \end{bmatrix} - \begin{bmatrix} T_0 \\ T_1 \end{bmatrix}\right\|\right) \\ &= \min\left(\left\|\begin{bmatrix} H_0 \\ H_1 \end{bmatrix}\beta_0 - \begin{bmatrix} T_0 \\ T_1 \end{bmatrix}\right\|\right) = E_0. \quad \square \end{aligned} \tag{26}$$

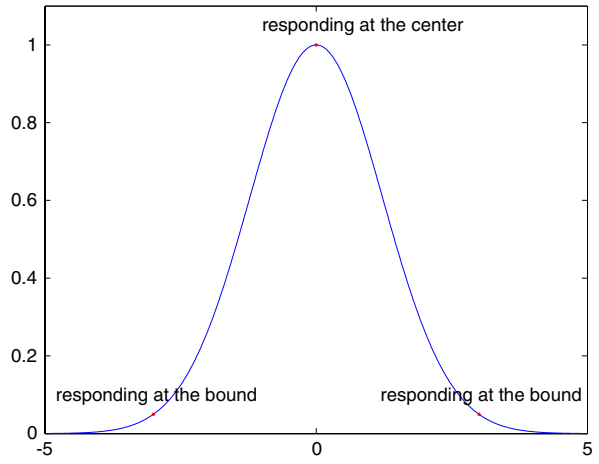

Fig. 1. Response curve of the RBF node.

Theorem 3 assures that adding some more hidden nodes to the network can lessen the training error. But the "less than or equal to" mark in Eq. (26) implies that adding hidden nodes cannot improve the training performance without limit..

Although the benefit of adding hidden nodes is guaranteed by Theorem 3, an iterative output weights updating method must be developed correspondingly. According to Eqs. (12) and (13), the solution for problem (23) should be

$$\beta_1 = \begin{bmatrix} H_0 & H_{01} \\ H_1 & H_{11} \end{bmatrix}^{\dagger} \begin{bmatrix} T_0 \\ T_1 \end{bmatrix} = \left(\begin{bmatrix} H_0 & H_{01} \\ H_1 & H_{11} \end{bmatrix}^{T} \begin{bmatrix} H_0 & H_{01} \\ H_1 & H_{11} \end{bmatrix} \right)^{-1} \begin{bmatrix} H_0 & H_{01} \\ H_1 & H_{11} \end{bmatrix}^{T} \begin{bmatrix} T_0 \\ T_1 \end{bmatrix}. \tag{27}$$

Notice that H_0 and T_0 are changed into β_0 while H_1, H_{11} and T_1 can be figured out from current data set X_1. As stated before, H_{01} is the output of the new added hidden nodes for the old data set X_0. At the time X_1 arrives and the new hidden nodes are added, X_0 has been discarded. Then H_{01} is unknown if an online learning algorithm is used. In order to deal with this challenge, a special method that can make H_{01} approximate a zero matrix is proposed in the following.

Consider the characteristic of the RBF hidden node. The activation function of the RBF node is a radially symmetric function of the distance between the center of the node and the input. So the output of the RBF node will become smaller and smaller as the input goes far away from the center of that node more and more. This phenomenon can be called as local responding property. The sketch map of the responding of an RBF node activation function is shown in Fig. 1.

Suppose a sphere S_0 is used to surround the old data set X_0 and the center of the new added node is outside that sphere as depicted in Fig. 2. In that figure, point "O" is the center of S_0 and point "A" is the center of the new hidden node. Point "B" and "C" are the apogee and perigee of S_0 corresponding to "A", respectively. Suppose the coordinate of "C" is x_c. According to the local responding property of the RBF node, if the output of the new hidden node for x_c is smaller than an arbitrary small positive value ε, the output will be even small for the points inside the sphere. That means all of the component in H_{01} will be trivial because each component in H_{01}, as shown in Eq. (24), is the output of a new added hidden node corresponding to a point inside S_0. In that case, H_{01} can be treated as a zero matrix and the deduction can move on smoothly.

So some special methods must be developed to make the output of the new added hidden node in point "C" be close to zero. This can be done through letting the center of that node be far away from the point "C" if the width is fixed or selecting a small width for that node if the center cannot be changed. In our settings, the center of the node is fixed in "A". So the width must be determined to satisfy the requirement that the response of the new added hidden node in "C" approximates zero. Suppose the center and width of the new hidden node are denoted as a and b. If the Gaussian function is chosen as the activation function of the hidden node, the requirement can be written as

$$e^{\frac{-\|x_c - a\|}{b}} \leq \varepsilon \Rightarrow b \leq -\frac{\|x_c - a\|}{\ln \varepsilon} \tag{28}$$

where ε is an arbitrary small positive value. Any width b that satisfies Eq. (28) is a reasonable choice. However, the coordinate of point "C", i.e. x_c, must be determined first.

Suppose the coordinate of point "A", "B" and "O" are denoted as x_a, x_b, and x_o. There must be a λ_1, such that

$$x_c = x_o + \lambda_1 (x_a - x_o). \tag{29}$$

Then λ_1 can be obtained as

$$\lambda_1 = \frac{\|x_c - x_o\|}{\|x_a - x_o\|} = \frac{R}{\|x_a - x_o\|} \tag{30}$$

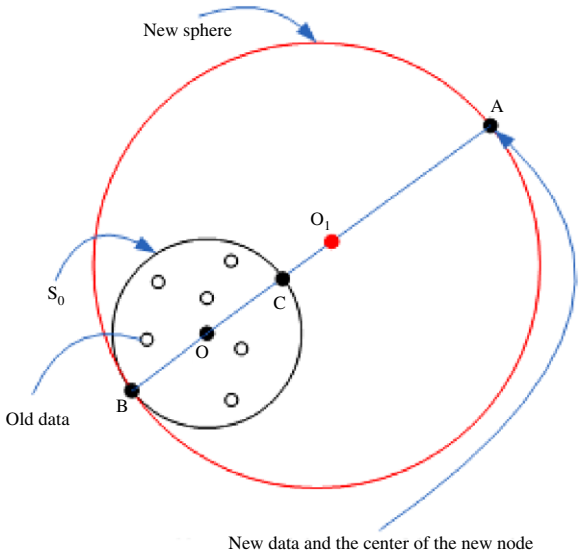

Fig. 2. The old data set and a new added node.

where R is the radius of S_0 and x_o is the coordinate of the center of S_0. If x_o and R are recorded, x_c can be calculated by Eqs. (29) and (30). Then a width b can be chosen from Eq. (28).

When the matrix H_{01} can be treated as a zero matrix, Eq. (27) can be rewritten as

$$\beta_1 = \left(\begin{bmatrix} H_0 & 0 \\ H_1 & H_{11} \end{bmatrix}^T \begin{bmatrix} H_0 & 0 \\ H_1 & H_{11} \end{bmatrix} \right)^{-1} \begin{bmatrix} H_0 & 0 \\ H_1 & H_{11} \end{bmatrix}^T \begin{bmatrix} T_0 \\ T_1 \end{bmatrix}. \tag{31}$$

Denote $H = \begin{bmatrix} H_0 \\ H_1 \end{bmatrix}$ and $\delta H = \begin{bmatrix} 0 \\ H_{11} \end{bmatrix}$, then

$$\beta_1 = \left([H\ \delta H]^T [H\ \delta H] \right)^{-1} [H\ \delta H]^T \begin{bmatrix} T_0 \\ T_1 \end{bmatrix}. \tag{32}$$

Denote $K_1 = \begin{bmatrix} H_0 & 0 \\ H_{11} & H_{12} \end{bmatrix}^T \begin{bmatrix} H_0 & 0 \\ H_{11} & H_{12} \end{bmatrix}$, then

$$K_1 = [H\ \delta H]^T [H\ \delta H] = \begin{bmatrix} H^T \\ \delta H^T \end{bmatrix} [H\ \delta H]. \tag{33}$$

Denote $K_1^{-1} = A = \begin{bmatrix} A_{11} A_{12} \\ A_{21} A_{22} \end{bmatrix} = \left(\begin{bmatrix} H^T \\ \delta H^T \end{bmatrix} [H\ \delta H] \right)^{-1}$. Accordingto [16], the components of A should be

$$\begin{aligned} A_{11} &= \left(H^T H\right)^{-1} + \left(H^T H\right)^{-1} \left(H^T \delta H\right) \times R^{-1} \left(\delta H^T H\right) \left(H^T H\right)^{-1} \\ A_{12} &= -(H^T H)^{-1} (H^T \delta H) R^{-1}, \qquad A_{21} = A_{12}^T, \qquad A_{22} = R^{-1} \end{aligned} \tag{34}$$

where

$$R = \delta H^T \delta H - \left(\delta H^T H\right) \left(H^T H\right)^{-1} \left(H^T \delta H\right). \tag{35}$$

Substitute the expression of H and δH into Eqs. (34) and (35), then

$$\begin{aligned} H^T H &= H_0^T H_0 + H_1^T H_1 = K_0 + H_1^T H_1 \\ H^T \delta H &= H_1^T H_{11}, \qquad \delta H^T \delta H = H_{11}^T H_{11}. \end{aligned} \tag{36}$$

Using Eqs. (36), (34) and (35) can be rewritten as

$$\begin{aligned} A_{11} &= \left(K_0 + H_1^T H_1\right)^{-1} + \left(K_0 + H_1^T H_1\right)^{-1} \left(H_1^T H_{11}\right) R^{-1} \left(H_{11}^T H_1\right) \left(K_0 + H_1^T H_1\right)^{-1} \\ A_{12} &= -(K_0 + H_1^T H_1)^{-1} (H_1^T H_{11}) R^{-1}, \qquad A_{21} = A_{12}^T, \qquad A_{22} = R^{-1} \\ R &= H_{11}^T H_{11} - \left(H_{11}^T H_1\right) \left(K_0 + H_1^T H_1\right)^{-1} \left(H_1^T H_{11}\right). \end{aligned} \tag{37}$$

A useful equation can be taken from [17] for the fast iterative implementation of matrix inversion, that is

$$\left(K_0 + H_1^T H_1\right)^{-1} = K_0^{-1} - K_0^{-1} H_1^T (I + H_1 K_0^{-1} H_1^T)^{-1} H_1 K_0^{-1}. \tag{38}$$

Combining Eqs. (31)–(38), the quick iterative updating method of K, K^{-1}, and β can be written as follows in the case of adding hidden nodes when the new training data arrives:

$$\begin{aligned}
&K_{n+1} = \begin{bmatrix} K_n + H_1^T H_1 & H_1^T H_{11} \\ H_{11}^T H_1 & H_{11}^T H_{11} \end{bmatrix}, \qquad \beta_{n+1} = P_{n+1} \begin{bmatrix} K_n \beta_n + H_1^T T_1 \\ H_{11}^T T_1 \end{bmatrix}, \qquad P_{n+1} = K_{n+1}^{-1} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \\
&A_{11} = P'_{n+1} + P'_{n+1} \left(H_1^T H_{11}\right) R^{-1} \left(H_{11}^T H_1\right) P'_{n+1}, \qquad A_{12} = -P'_{n+1} (H_1^T H_{11}) R^{-1}, \qquad A_{21} = A_{12}^T, A_{22} = R^{-1} \\
&R = H_{11}^T H_{11} - \left(H_{11}^T H_1\right) P'_{n+1} \left(H_1^T H_{11}\right) \\
&P'_{n+1} = \left(K_n + H_1^T H_1\right)^{-1} = P_n - P_n H_1^T (I + H_1 P_n H_1^T)^{-1} H_1 P_n, \qquad P_n = K_n^{-1}.
\end{aligned} \tag{39}$$

If there is no need to add hidden nodes to the network, the quick iterative updating method of K, K^{-1}, and β can be written as [7]:

$$\begin{aligned}
K_{n+1} &= K_n + H_1^T H_1 \\
P_{n+1} &= K_{n+1}^{-1} = P_n - P_n H_1^T (I + H_1 P_n H_1^T)^{-1} H_1 P_n \\
\beta_{n+1} &= \beta_n + P_{n+1} H_1^T (T_1 - H_1 \beta_n).
\end{aligned} \tag{40}$$

Finally, the radius and center of the sphere S_0 should be updated accordingly when new samples arrive in order to make the online learning process continue. Consider the case of one sample firstly. The old data set X_0, the new coming datum x_1, and the sphere S_0 surrounding X_0 are shown in Fig. 2. Suppose x_1 locates at point "A". In order to get the smallest sphere to surround X_0 and x_1, the center of the new sphere should be point "O_1", which is the midpoint of "A" and "B", and the new radius should be half of the length of the line segment "AB". Therefore the coordinate of "B" must be acquired first. As shown in Eq. (29), there must be a λ_2, such that

$$x_b = x_o + \lambda_2 (x_o - x_a) \tag{41}$$

and λ_2 can be derived as

$$\lambda_2 = \frac{\|x_b - x_o\|}{\|x_o - x_a\|} = \frac{R}{\|x_o - x_a\|} \tag{42}$$

where R and x_o are the same as that in Eq. (29). Then the updating equation for R and x_o are:

$$\begin{aligned}
R_{new} &= \frac{\|x_a - x_b\|}{2} \\
x_{o_new} &= \frac{x_a + x_b}{2}.
\end{aligned} \tag{43}$$

If the number of new coming data is more than one, the above method can be used to update R and x_o for each data one by one and then the final center and radius of the new sphere can be obtained.

3.3. The algorithm

Based on the solid foundation presented above, an algorithm can be summarized. For the sake of simplicity, only the case that the data are arriving one by one is considered. If the data arrive chunk by chunk, the following algorithm can be used for each data in the chunk sequentially.

Step (1): Select the activation function and the number M for the hidden nodes. Initialize the learning using a small chunk of initial training data $X_0 = \{(x_i, t_i)\}_{i=1}^N$. Assign random center a_i and impact factor b_i, $i = 1, \ldots M$. Calculate the initial hidden layer output matrix H_0 which is shown in Eq. (6). Estimate the initial output weight $\beta_0 = P_0 H_0^T T_0$ where $P_0 = K_0^{-1}$ and $K_0 = H_0^T H_0$. T_0 is shown in Eq. (10). Determine the center x_0 and radius R_0 of the smallest sphere S_0 that surrounds X_0.

Step (2): If there is no new training data coming, the current network is used for prediction. Otherwise, go to step (3) to start the online learning procedure to accommodate the new information in datum $x_1 = (x_{N+1}, t_{N+1})$.

Step (3): Keep the structure of the network unchanged and try to update output weights and related parameters using Eq. (40). Calculate the mean square training error for new arrived datum. Judge whether x_1 is inside or outside the sphere S_0. If the training error for x_1 cannot satisfy the requirements and x_1 is in the outside of S_0, discard all of the update and go to step (4). Otherwise, go to step (5).

Table 1
Specification of data sets for performance evaluation.

Data set	#Attributes	#Classes	#Training data	#Testing data
Auto-Mpg	7	–	320	72
Abalone	8	–	3000	1 177
California housing	8	–	8000	12 640
Image segment	19	7	1500	810
Satellite image	36	6	4435	2 000
DNA	180	3	2000	1 186
Practical application	84	2	1056	508

Step (4): Add a hidden node to the network. Set the center of that node the same as the coordinate of x_1.Use Eq. (28) to determine the width. Update output weights and related parameters using Eq. (39) instead. Go to step (5).

Step (5): Update the parameters of S_0, i.e. center and radius, using Eq. (43) and go to step (2).

Remark 1. In step (1), the number of the initial hidden nodes M should not be less than the size of initialization data set X_0 in order to make $rank(H_0) = M$, which is the foundation of all of the deduction.

Remark 2. The presented algorithm is similar to the recursive least-square (RLS) [18] algorithm and OS-ELM. So its convergence can be ensured by all of the results of RLS algorithm which has been indicated by [7].

4. Performance evaluation of SAO-ELM

The performance of SAO-ELM is evaluated on the benchmark problems described in Table 1 which includes three regression applications (auto-MPG, abalone, California housing), three classification problems (image segment, satellite image, DNA) and a practical application (continuous casting quality prediction). SAO-ELM is firstly compared with other popular online learning algorithms, such as OS-ELM, GART, GGAP-RBF, and MRAN. Then the performance evaluation of SAO-ELM is conducted fully on a practical problem. All the simulations have been conducted in MATLAB 2008A environment running on an ordinary PC with 3.2 GHZ CPU. The Gaussian RBF activation function $G(a, b, x) = \exp(-\|x - a\|^2 / b)$ is the only selection for all the 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].

4.1. Benchmark applications

Before simulation, we should estimate the optimal architecture of the network and the optimal leaning parameters of the learning algorithm. That is called model selection in the literature. For SAO-ELM, only the optimal number of hidden units needs to be determined. As stated in [7], the number can be determined using the cross-validation method. However, for benchmark problems in this simulation, in order to compare SAO-ELM and OS-ELM in the same circumstance, the number of the initial hidden nodes M is the same as the one stated in [7]. The number of training data N_0 for initialization should be about $M + 50$ for regression problems and about $M + 100$ for classification problems. The centers of the initial hidden nodes are randomly chosen from the range [−1, 1]. Similarly, the widths are also randomly chosen from the range [0, 1] except for "image segment" and DNA case. For these two cases, the range should be [3, 11] and [20, 60] respectively in order to make H_0 nonsingular. The performance is the average result on 50 trials. The average training time, the average training and testing RMSE for regression problems, and the average training and testing classification rate for classification applications are represented in the following.

(1) *Regression problems:* auto-MPG, abalone, and California housing are the three regression problems selected from [19]. The auto-MPG problem is to predict the fuel consumption of different models of cars. The abalone problem is the estimation of the age of abalone from physical measurements. 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. The training and testing data are randomly selected for all the regression problems.

Table 2 summarizes the results for regression problems in terms of training time, training RMSE, testing RMSE, and the number of hidden units for each algorithm. The initial number of hidden nodes for SAO-ELM is the same as OS-ELM so as to make a consistent condition for comparison. The results of other online learning algorithm are cited from the literature directly.

As observed from Table 2, the performance of SAO-ELM is better than others. Although the best training RMSE is obtained by GART, the testing RMSE of SAO-ELM can be always optimal. This indicates that SAO-ELM has the best generalization performance, which is very important for practical application. The training time of SAO-ELM is larger than that of OS-ELM, which is due to the fact that the SAO-ELM has to update the output weights for trial before adding a node to the hidden layer while OS-ELM updates the output weights directly. However, as we can see from Table 2, SAO-ELM still hold the fast

Table 2
Comparison between SVOS-ELM and other sequential algorithms on regression problems.

Data sets	Algorithms	Training time	RMSE		#Nodes	
			Training	Testing	Start	End
Auto-MPG	SAO-ELM	0.2784	0.0639	0.0607	25	30.42
	OS-ELM(RBF) [7]	0.0915	0.0696	0.0759	25	
	GART [14]	–	0.0417	0.0739	–	
	GAP-RBF [5]	0.4520	0.1144	0.1404	3.12	
Abalone	SAO-ELM	3.3794	0.0738	0.0734	25	34.24
	OS-ELM(RBF) [7]	1.2478	0.0759	0.0783	25	
	GART [14]	–	0.0646	0.0800	–	
	GAP-RBF [5]	83.784	0.0963	0.0966	23.62	
California housing	SAO-ELM	11.3356	0.1296	0.1307	50	60.18
	OS-ELM(RBF) [7]	6.9629	0.1321	0.1341	50	
	GART [14]	–	0.0683	0.1316	–	
	GGAP-RBF [6]	115.34	0.1417	0.1386	18	

Table 3
Comparison between SVOS-ELM and other sequential algorithms on classification applications.

Data sets	Algorithms	Training time	Accuracy		#Nodes	
			Training	Testing	Start	End
Image segmentation	SAO-ELM	9.1644	0.9725	0.9516	180	191
	OS-ELM(RBF) [7]	9.9981	0.9700	0.9488	180	
	GART [14]	–	0.9909	0.9680	–	
	GAP-RBF [5]	1724.3	–	0.8993	44.2	
	MRAN [4]	7004.5	–	0.9330	53.1	
Satellite image	SAO-ELM	211.0347	0.9480	0.9139	400	413
	OS-ELM(RBF) [7]	319.14	0.9318	0.8901	400	
	GART [14]	–	0.9804	0.9053	–	
	MRAN [4]	2469.4	–	0.8636	20.4	
DNA	SAO-ELM	14.9844	0.9640	0.9454	200	214
	OS-ELM(RBF) [7]	20.9510	0.9612	0.9437	200	
	GART [14]	–	1.0000	0.8831	–	
	MRAN [4]	6079	–	0.8685	5	

learning characteristic of ELM. So it can still be used for online application. Finally we can see that the number of hidden nodes at the end of the simulation is larger than the initial number which indicates that SAO-ELM has the ability to adjust its structure as what we expect.

(2) *Classification Problems:* Three classification problems are chosen from [19], namely: image segmentation, satellite image, and DNA. The image segmentation problem consists of 2310 instances. The instances were drawn randomly from a database of 7 outdoor images. Each image is a 3×3 region. The aim is to recognize each region into one of the seven categories, i.e. brick facing, sky, foliage, cement, window, path, and grass using 19 attributes extracted from each square region.

The satellite image problem consists of a database generated from landsat multispectral scanner. The sample database was generated taking a small section from the original data. Each data in the database corresponds to a region of 3×3 pixels. The aim is to predict the classification, given the multi-spectral values. There are 6 classes and each class is coded as a number.

The DNA problem is the database "Primate splice-junction gene sequences (DNA) with associated imperfect domain theory". Splice junctions are points on a DNA sequence at which "superfluous" DNA is removed during the process of protein creation in higher organisms. The problem posed in this data set is to recognize, given a sequence of DNA, 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 problem consists of two subtasks: recognizing exon/intron boundaries (EI site), and recognizing intron/exon boundaries (IE site). Every symbolic variables in the DNA sequence is coded as three binary indicator variables as it was done in [7] resulting 180 binary attributes. The character string indicating the category is coded as a number.

Table 3 summarizes the results for classification problems in terms of training time, training RMSE, testing RMSE, and the number of hidden units for each algorithm. The initial number of hidden nodes for SAO-ELM is the same as OS-ELM so as to make a consistent condition for comparison. The results for other online learning algorithms are cited from the literature directly.

As we can see from Table 3, SAO-ELM obtains the best generalization performance except for "image segmentation" case. For that case, the GART is the best. But the training time and network structure are unknown from the literature presenting

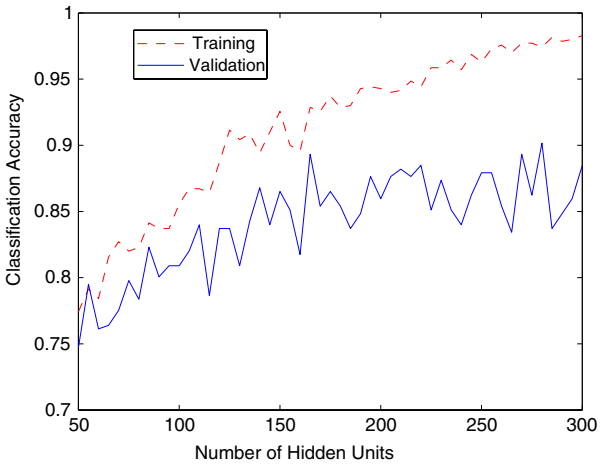

Fig. 3. Cross-validation.

Table 4
Comparison between SAO-ELM and other neural network on real application.

Data sets	Algorithms	Training time	Accuracy		#Nodes	
			Training	Testing	Start	End
Practical problem	SAO-ELM	7.7397	0.8904	0.8165	200	210
	OS-ELM(RBF)	7.3006	0.8762	0.8073	200	
	BP NN	77.000	0.8520	0.7720	80	

GART. We can also conclude that SAO-ELM is better than OS-ELM in all aspects including the training time, training accuracy, and testing accuracy. All of these advantages are originated from the structure-adjustable property. By contrast, the structure of OS-ELM is fixed. The number of hidden nodes of SAO-ELM at the end of the simulation is larger than the initial one which also indicates the successful implementation of adjusting structure when learning.

4.2. Practical problem

The practical problem comes from the continuous casting process of steelmaking. The aim is to predict the continuous casting quality using various parameters about continuous casting process. It is an important task because it can ensure process continuity, improve product quality and reduce production cost. There are 84 attributes used. The quality statuses, i.e. normal and abnormal, are coded as binary number. We use 1056 samples for training and 508 samples for testing. The testing data are gathered some days later after the training data have been collected.

As stated before, we should do model selection before modelling. That is, we should determine the initial number of hidden units and the parameters for those units. The centers of the initial hidden nodes are randomly selected from the range $[-1, 1]$. The widths are randomly chosen from the range $[10, 20]$. In order to determine the optimal initial hidden unit number, a cross-validation procedure is used. The training data are divided into two non-overlapping groups: one for training and another for validation. The optimal number is selected as the one which results in the highest classification accuracy for the validation data set. The result of cross-validation is shown in Fig. 3. The red dashed curve corresponds to the trend of the classification accuracy as the number of hidden units increase for the training data while the blue solid one is for the validation data. Although the classification accuracy for the training data is increasing all along as the number of hidden units goes up, the one for the validation data cannot be improved after the number of hidden node reaches 200. Therefore we choose the initial hidden units number for the practical problem as 200. The number of initial training data is 300 (i.e. $200 + 100$, as stated in Section 4.1) as this is also a classification problem.

The learning evolution is shown in Fig. 4. In that figure, the curve above shows the trend of the classification accuracy for validation data with the increasing of the number of samples and the one below sets out the changes of the number of hidden units as the training samples provided to the learning machine continuously. As we can see from Fig. 4, the improvement of classification accuracy for validation data is consistent with the increasing of hidden nodes. This demonstrates the validity of the proposed method of adding nodes.

Table 4 compares the results of SAO-ELM with OS-ELM and BP neural network using the various indices used for benchmark problems. The results show that SAO-ELM obtains the best performance and has the quick and online learning capability.

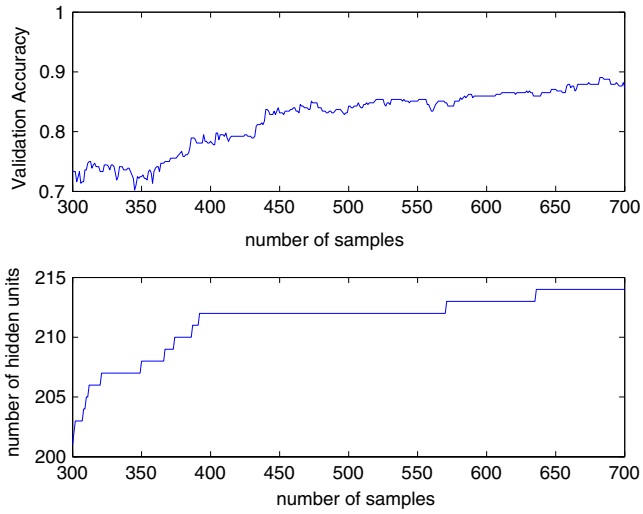

Fig. 4. Learning evolution.

5. Conclusion

In this paper, a new online learning algorithm (SAO-ELM) has been developed for single hidden layer neural network. The RBF hidden nodes are used because of their local responding property. Using the idea of ELM proposed by Huang et al. developed for batch learning, the parameters of the hidden nodes are randomly selected and the output weights are analytically determined. Apart from determining the number of hidden nodes using the cross-validation method, no other parameters have to be chosen. Hidden nodes can be added to the network in the middle of learning if necessary. A new fast iterative equation for updating the output weights when a hidden unit is added to the network is deduced. The performance of SAO-ELM is compared with other well-known online learning algorithms on real world benchmark problems including regression problems and classification problems. In addition, a numerical comparison based on practical continuous casting process data is made. Results indicate that SAO-ELM produces better generalization performance and keeps the fast learning characteristic of ELM.

Acknowledgements

This work was supported in part by National Basic Research Program of China (973 Program) (2002CB312202, 2009CB320602), in part by National High Technology Research and Development Program of China (863 Program) (2006AA04Z163), in part by National Natural Science Foundation of China (60834004, 60721003), and in part by Program for New Century Excellent Talents in University.

References

[1] R. Polikar, L. Udpa, S. Udpa, V. Honavar, Learn++: an incremental learning algorithm for supervised neural networks, IEEE Trans. Syst., Man, Cybern. C, Appl. Rev. 31 (2001) 497–508.
[2] J. Platt, A resource-allocating network for function interpolation, Neural Comput. 3 (1991) 213–225.
[3] V. Kadirkamanathan, M. Niranjan, A function estimation approach to sequential learning with neural networks, Neural Comput. 5 (1993) 954–975.
[4] L. Yingwei, N. Sundararajan, P. Saratchandran, A sequential learning scheme for function approximation using minimal radial basis function (RBF) neural networks, Neural Comput. 9 (1997) 461–478.
[5] G.-B. Huang, P. Saratchandran, N. Sundararajan, An efficient sequential learning algorithm for growing and pruning RBF (GAP-RBF) networks, IEEE Trans. Syst., Man, Cybern. 34 (2004) 2284–2292.
[6] G.-B. Huang, P. Saratchandran, N. Sundararajan, A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation, IEEE Trans. Neural Netw. 16 (2005) 57–67.
[7] N.-Y. Liang, G.-B. Huang, P. Saratchandran, N. Sundararajan, A fast and accurate online sequential learning algorithms for feedforward network, IEEE Trans. Neural Netw. 17 (2006) 1411–1423.
[8] S. Grossberg, Nonlinear neural network: principles, mechanisms and architectures, IEEE Trans. Neural Netw. 1 (1988) 17–61.
[9] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: A new learning scheme of feedforward neural networks, in: Proc. Int. Joint Conf. Neural Netw., IJCNN2004, Budapest, Hungary 2, 2004, pp. 985–990.
[10] G.-B. Huang, C.-K. Siew, Extreme learning machine: RBF network case, in: Proc. 8th Int. Conf. Control, Autom., Robot., Vis., ICARCV 2004, Kunming, China, 2004 pp. 1029–1036.
[11] G.-B. Huang, Q.-Y. Zhu, K.Z. Mao, C.-K. Siew, P. Saratchandran, N. Sundararajan, Can threshold networks be trained directly? IEEE Trans. Circuits Syst. II, Exp. Briefs. 53 (2006) 187–191.
[12] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (2006) 489–501.
[13] G.-B. Huang, L. Chen, C.-K. Siew, Universal approximation using incremental constructive feedforward networks with random hidden nodes, IEEE Trans. Neural Netw. 17 (2006) 879–892.

[14] Keem Siah Yap, Chee Peng Lim, Izham Zainal Abidin, A hybrid ART-GRNN online learning neural network with a ε-insensitive loss function, IEEE Trans. Neural Netw. 19 (2008) 1641–1646.
[15] G.-B. Huang, Learning capability and storage capacity of two-hidden-layer feedforward networks, IEEE Trans. Neural Netw. 14 (2003) 274–281.
[16] Guorui Feng, Guang-Bin Huang, Qingping Lin, Robert Gay, Error minimized extreme learning machine with growth of hidden nodes and incremental learning, IEEE Trans. Neural Netw. 20 (2009) 1352–1357.
[17] G.H. Golub, C.F.V. Loan, Matrix Computations, 3rd ed., The Johns Hopkins Univ. Press, Baltimore, MD, 1996.
[18] E.K.P. Chong, S.H. Zak, An Introduction to Optimization, Wiley, New York, 2001.
[19] C. Blake, C. Merz, UCI repository of machine learning databases, Dept. Inf. Comput. Sci., Univ. California, Irvine, CA, 1998 [Online]. Available: http://www.ics.uci.edu/~mlearn/MLRepository.html.