Int. J. Mach. Learn. & Cyber. (2011) 2:107–122
DOI 10.1007/s13042-011-0019-y

ORIGINAL ARTICLE

Extreme learning machines: a survey

Guang-Bin Huang · Dian Hui Wang · Yuan Lan

Received: 1 April 2011 / Accepted: 16 April 2011 / Published online: 25 May 2011

Abstract Computational intelligence techniques have been used in wide applications. Out of numerous computational intelligence techniques, neural networks and support vector machines (SVMs) have been playing the dominant roles. However, it is known that both neural networks and SVMs face some challenging issues such as: (1) slow learning speed, (2) trivial human intervene, and/or (3) poor computational scalability. Extreme learning machine (ELM) as emergent technology which overcomes some challenges faced by other techniques has recently attracted the attention from more and more researchers. ELM works for generalized single-hidden layer feedforward networks (SLFNs). The essence of ELM is that the hidden layer of SLFNs need not be tuned. Compared with those traditional computational intelligence techniques, ELM provides better generalization performance at a much faster learning speed and with least human intervene. This paper gives a survey on ELM and its variants, especially on (1) batch learning mode of ELM, (2) fully complex ELM, (3) online sequential ELM, (4) incremental ELM, and (5) ensemble of ELM.

Keywords Extreme learning machine · Support vector machine · ELM kernel · ELM feature space · Ensemble · Incremental learning · Online sequential learning

G.-B. Huang (✉) · Y. Lan
School of Electrical and Electronic Engineering,
Nanyang Technological University, Nanyang Avenue,
Singapore 639798, Singapore
e-mail: egbhuang@ntu.edu.sg

D. H. Wang
Department of Computer Science and Computer Engineering,
La Trobe University, Melbourne, VIC 3086, Australia
e-mail: dh.wang@latrobe.edu.au

1 Introduction

There exist many types of neural networks, however, feedforward neural networks may be one of the most popular neural networks. A feedforward neural network consists of one input layer receiving the stimuli from external environments, one or multi-hidden layers, and one output layer sending the network output to external environments. Three main approaches are usually used in training feedforward networks:

1. Gradient-descent based (e.g. backpropagation (BP) method [1] for multi-layer feedforward neural networks). Additive type of hidden nodes are most often used in such networks. For additive hidden node with the activation function $g(x): R \rightarrow R$ (e.g. sigmoid: $g(x) = 1/(1+\exp(-x))$), the output function of the ith node in the lth hidden layer is given by
$$G(\mathbf{a}_i^{(l)}, b_i^{(l)}, \mathbf{x}^{(l)}) = g(\mathbf{a}_i^{(l)} \cdot \mathbf{x}^{(l)} + b_i^{(l)}), \quad b_i^{(l)} \in R \qquad (1)$$
where $\mathbf{a}_i^{(l)}$ is the weight vector connecting the $(l-1)$th layer to the ith node of the lth layer and $b_i^{(l)}$ is the bias of the ith node of the lth layer. $\mathbf{a}_i^{(l)} \cdot \mathbf{x}^{(l)}$ denotes the inner product of vectors $\mathbf{a}_i^{(l)}$ and $\mathbf{x}^{(l)}$. Gradient-descent based learning algorithms usually run much slower than expected.
2. Standard optimization method based (e.g. support vector machines, SVMs [2], for a specific type of SLFNs, the so-called support vector network). Rosenblatt [3] investigated perceptrons (multi-layer feedforward neural networks) half a century ago. Rosenblatt suggested a learning mechanism where only the weights of the connections from the last hidden layer to the output layer were adjusted. After all the rest weights fixed the input data are actually transformed

into a feature space Z of the last hidden layer (cf. Fig. 1). In this feature space a linear decision function is constructed:

$$f(\mathbf{x}) = \operatorname{sign}\left(\sum_{i=1}^{L} \beta_i z_i(\mathbf{x})\right) \qquad (2)$$

where β_i is the output weight between the output node and the ith neuron in the last hidden layer of a perceptron, and $z_i(\mathbf{x})$ is the output of the ith neuron in the last hidden layer of the perceptron. In order to find an alternative solution of $z_i(\mathbf{x})$, in 1995 Cortes and Vapnik [2] proposed the SVM which maps the data from the input space to some high dimensional feature space Z through some nonlinear mapping chosen a priori. Optimization methods are used to find the separating hyperplane which maximizes the separating margins of two different classes in the feature space.

3. Least-square based (e.g. radial basis function (RBF) network learning [4]). For RBF hidden node with activation function $g(x): R \rightarrow R$ (e.g. Gaussian: $g(x) = \exp(-x^2)$, $G(\mathbf{a}_i, b_i, \mathbf{x})$ is given by

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

where $\mathbf{a}_i$ and b_i are the center and impact factor of the ith RBF hidden node. R^+ indicates the set of all positive real values. The RBF network is a special case of SLFNs with RBF nodes in its hidden layer (cf. Fig. 2). 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. In Lowe's RBF network implementation [4], the centers $\mathbf{a}_i$ of RBF hidden nodes can be randomly selected from the training data or from the region of training data instead of tuning, and all the impact factors b_i of RBF hidden nodes are usually set with the same value (p. 173 of [4]). After RBF hidden nodes parameters $(\mathbf{a}_i, b_i)$ fixed, the output weight vector $\boldsymbol{\beta}_i$ linking the ith RBF hidden node to the output layer becomes the only unknown parameter which can be resolved by least-square method.

Extreme learning machines (ELMs) were originally developed for the SLFNs [5–7] and then extended to the "generalized" SLFNs. Such generalized SLFNs need not be neuron alike [8, 9]. The essence of ELM is that: different from the common understanding of learning, the hidden layer of SLFNs need not be tuned. One of the typical implementation of ELMs is to apply random computational nodes in the hidden layer, which may be independent of the training data. Different from traditional learning algorithms for neural networks ELM not only tends to reach the smallest training error but also the smallest norm of output weights. According to the neural network theory [10], for feedforward neural networks reaching smaller training error the smaller the norm of weights is, the better generalization performance the networks tend to have. Since in ELM the hidden layer need not be tuned and the hidden layer parameters can be fixed, the output weights can then be resolved using the lease-square method.

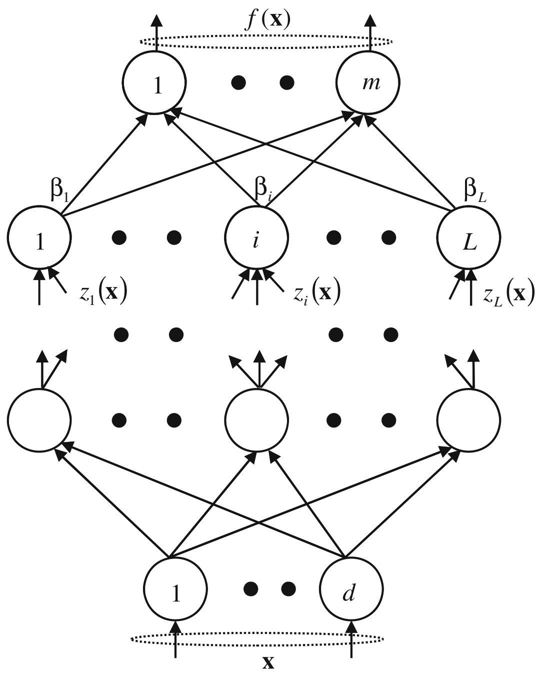

Fig. 1 Multi hidden layers feedforward network

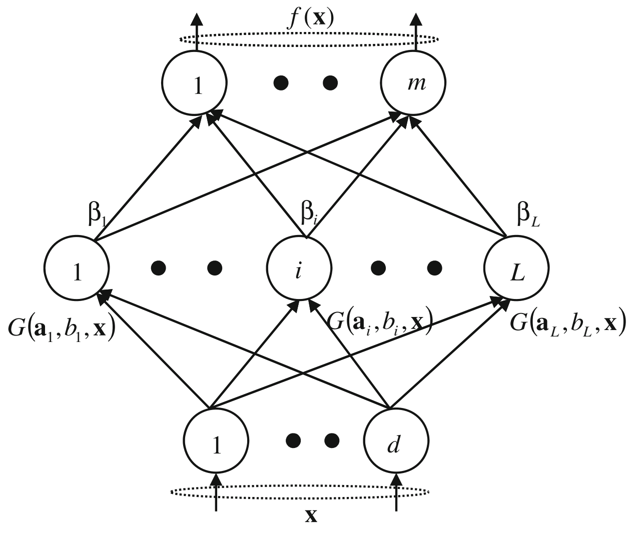
$f(\mathbf{x})$
1
m
β_1
β_i
β_L
1
i
L
$G(\mathbf{a}_1, b_1, \mathbf{x})$
$G(\mathbf{a}_i, b_i, \mathbf{x})$
$G(\mathbf{a}_L, b_L, \mathbf{x})$
1
d
$\mathbf{x}$

Fig. 2 Single-hidden layer feedforward network

2 Learning theories of ELMs

The interpolation capability and universal approximation capability of ELMs have been investigated in Huang et al.

[6–9].The output function of SLFNs with L hidden nodes can be represented by

$$f_L(\mathbf{x}) = \sum_{i=1}^{L} \boldsymbol{\beta}_i g_i(\mathbf{x}) = \sum_{i=1}^{L} \boldsymbol{\beta}_i G(\mathbf{a}_i, b_i, \mathbf{x}), \quad \mathbf{x} \in \mathbf{R}^d,\ \boldsymbol{\beta}_i \in \mathbf{R}^m \tag{4}$$

where g_i denotes the output function $G(\mathbf{a}_i, b_i, \mathbf{x})$ of the ith hidden node. For additive nodes with activation function g, g_i is defined as

$$g_i = G(\mathbf{a}_i, b_i, \mathbf{x}) = g(\mathbf{a}_i \cdot \mathbf{x} + b_i), \quad \mathbf{a}_i \in \mathbf{R}^d,\ b_i \in R \tag{5}$$

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

$$g_i = G(\mathbf{a}_i, b_i, \mathbf{x}) = g(b_i \|\mathbf{x} - \mathbf{a}_i\|),\ \mathbf{a}_i \in \mathbf{R}^d,\ b_i \in R^+ \tag{6}$$

In the past two decades, the interpolation and universal approximation capabilities of SLFNs have been investigated thoroughly. It was proved [11, 12] that N arbitrary distinct samples can be learned precisely by SLFNs with N threshold hidden nodes. Further study [13] gave a more complete answer on the interpolation capability of SLFNs and proved that an SLFN with at most N hidden nodes and with any arbitrary bounded nonlinear activation function which has a limit at one infinity can learn any N arbitrary distinct samples with zero error. Such activation functions include the threshold, ramp and sigmoid functions as well as the radial basis, "cosine squasher" [14] and many non-regular functions. Many researchers [15–21] have rigorously proved in theory that given activation function $g(x)$ satisfying certain mild conditions there exists a sequence of network functions $\{f_L\}$ approximating to any given continuous target function f with any expected learning error $\epsilon > 0$. In all these conventional neural network theories, all the parameters in any f_L of the network sequence (e.g. the hidden layer parameters $(\mathbf{a}_i, b_i)$ and the output weights $\boldsymbol{\beta}_i$) are required freely adjustable. According to these conventional neural network theories, hidden layer parameters $(\mathbf{a}_i, b_i)$ need to be tuned properly and appropriate values of network parameters (e.g. $(\mathbf{a}_i, b_i)$ and $\boldsymbol{\beta}_i$) need to be found for any given target function f. To minimize the effort spent on adjusting hidden layer parameters $(\mathbf{a}_i, b_i)$ has been tried in the past two decades. Instead of adjusting all the parameters of hidden layers in all f_L of the network sequence, some researchers [22–25] suggested incremental methods for SLFNs which adjust the parameters of newly added hidden nodes and then fix them after tuning. The parameters of the existing hidden nodes will remain fixed and never be updated in the further learning procedure. Hidden layer parameters in those conventional learning models need to be adjusted at least once based on the training samples. In contrast, all the parameters of the hidden layer in the ELMs need not be tuned and can be independent of the training samples [6–9]. One of the typical implementation of ELMs is that the hidden node parameters $(\mathbf{a}_i, b_i)$ of ELM can be randomly generated. The learning capability of extreme learning machines have been studied in two aspects: interpolation capability [6] and universal approximation capability [7–9].

2.1 Interpolation theorem

For N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i) \in \mathbf{R}^d \times \mathbf{R}^m$, SLFNs with L hidden nodes are mathematically modeled as

$$\sum_{i=1}^{L} \boldsymbol{\beta}_i g_i(\mathbf{x}_j) = \sum_{i=1}^{L} \boldsymbol{\beta}_i G(\mathbf{a}_i, b_i, \mathbf{x}_j) = \mathbf{o}_j, \quad j = 1, \ldots, N \tag{7}$$

That SLFNs can approximate these N samples with zero error means that $\sum_{j=1}^{L} \|\mathbf{o}_j - \mathbf{t}_j\| = 0$, i.e., there exist $(\mathbf{a}_i, b_i)$ and $\boldsymbol{\beta}_i$ such that

$$\sum_{i=1}^{L} \boldsymbol{\beta}_i G(\mathbf{a}_i, b_i, \mathbf{x}_j) = \mathbf{t}_j, \quad j = 1, \ldots, N. \tag{8}$$

The above N equations can be written compactly as:

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{T} \tag{9}$$

where

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}(\mathbf{x}_1) \\ \vdots \\ \mathbf{h}(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_1) & \cdots & G(\mathbf{a}_L, b_L, \mathbf{x}_1) \\ \vdots & \cdots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_N) & \cdots & G(\mathbf{a}_L, b_L, \mathbf{x}_N) \end{bmatrix}_{N \times L} \tag{10}$$

$$\boldsymbol{\beta} = \begin{bmatrix} \boldsymbol{\beta}_1^T \\ \vdots \\ \boldsymbol{\beta}_L^T \end{bmatrix}_{L \times m} \quad \text{and} \quad \mathbf{T} = \begin{bmatrix} \mathbf{t}_1^T \\ \vdots \\ \mathbf{t}_N^T \end{bmatrix}_{N \times m} \tag{11}$$

$\mathbf{H}$ is called the hidden layer output matrix of the SLFN [13, 26]; the ith column of $\mathbf{H}$ is the ith hidden node output with respect to inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$. $\mathbf{h}(\mathbf{x}) = G(\mathbf{a}_1, b_1, \mathbf{x}), \ldots, g(\mathbf{a}_L, b_L, \mathbf{x})$ is called the hidden layer feature mapping. The ith row of $\mathbf{H}$ is the hidden layer feature mapping with respect to the ith input $\mathbf{x}_i : \mathbf{h}(\mathbf{x}_i)$. It has been proved [6] that from the interpolation capability point of view, if the activation function g is infinitely differentiable in any interval the hidden layer parameters can be randomly generated.

Theorem 2.1 [6] *Given any small positive value $\epsilon > 0$, activation function $g : R \rightarrow R$ which is infinitely differentiable in any interval, and N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i) \in \mathbf{R}^d \times \mathbf{R}^m$, there exists $L \leq N$ such that for any*

$\{\mathbf{a}_i, b_i\}_{i=1}^L$ *randomly generated from any intervals of* $\mathbf{R}^d \times R$, *according to any continuous probability distribution, with probability one,* $\|\mathbf{H}_{N\times L}\boldsymbol{\beta}_{L\times m} - \mathbf{T}_{N\times m}\| < \epsilon$.

From the interpolation point of view the maximum number of hidden nodes required is not larger than the number of training samples. In fact, if $L = N$, the training errors can be zero.

Theorem 2.2 [6] *Given any activation function* $g : R \to R$ *which is infinitely differentiable in any interval and* N *arbitrary distinct samples* $(\mathbf{x}_i, \mathbf{t}_i) \in \mathbf{R}^d \times \mathbf{R}^m$, *for any* $\{(\mathbf{a}_i, b_i)\}_{i=1}^N$ *randomly generated from any intervals of* $\mathbf{R}^d \times R$, *according to any continuous probability distribution, with probability one,* $\|\mathbf{H}_{N\times N}\boldsymbol{\beta}_{N\times m} - \mathbf{T}_{N\times m}\| = 0$.

From interpolation point of view, wide type of activation functions can be used in ELM, which include the sigmoid functions, the radial basis, sine, cosine, exponential, and many other non-regular functions [13]. It may be too strict to request that activation functions of hidden nodes are infinitely differentiable. For example, it may not include some important activation functions such as threshold function: $g(x) = 1_{x\geq 0} + 0_{x<0}$. Threshold networks are very popular in real applications, especially in digital hardware implementation. However, as threshold function is not differentiable, researchers did not manage to find any efficient direct learning algorithms for threshold networks in the past two decades [27–30]. Interestingly, from the universal approximation point of view, the above mentioned interpolation theorem can be extended to almost any type of nonlinear piecewise continuous function including the threshold function, and thus an efficient direct learning algorithm (e.g. ELM) can be applied to those cases which cannot be handled by other learning techniques in the past decades.

2.2 Universal approximation theorem

Huang et al. [7] proved in theory that SLFNs with randomly generated additive or RBF nodes can universally approximate any continuous target functions over any compact subset $X \in \mathbf{R}^d$. Let $L^2(X)$ be a space of functions f on a compact subset X in the d-dimensional Euclidean space $\mathbf{R}^d$ such that $|f|^2$ are integrable, that is, $\int_X |f(\mathbf{x})|^2 d\mathbf{x} < \infty$. Let $L^2(\mathbf{R}^d)$ denoted by L^2. For $u, v \in L^2(X)$, the inner product $\langle u, v\rangle$ is defined by

$$\langle u, v\rangle = \int_X u(\mathbf{x})v(\mathbf{x})\, d\mathbf{x} \tag{12}$$

The norm in $L^2(X)$ space is denoted as $\|\cdot\|$, and the closeness between the network function f_L and the target function f is measured by the $L^2(X)$ distance:

$$\|f_L - f\| = \left[\int_X |f_L(\mathbf{x}) - f(\mathbf{x})|^2 d\mathbf{x}\right]^{1/2} \tag{13}$$

Definition 2.1 (p. 334 of [31]) A function $g(x) : R \to R$ is said to be *piecewise continuous* if it has only a finite number of discontinuities in any interval, and its left and right limits are defined (not necessarily equal) at each discontinuity.

Definition 2.2 A node is called a random node if its parameters $(\mathbf{a}, b)$ are randomly generated based on a continuous sampling distribution probability.

Different from the randomness mentioned in other learning methods [4, 32, 33], all the hidden node parameters $(\mathbf{a}_i, b_i)$ in ELMs can be independent of the training samples and can be randomly generated before the training samples observed. (Refer to [34] for the details of the differences between ELM and Igelnik and Pao [33] and Lowe et al. [4, 32]).

Definition 2.3 The function sequence $\{g_i = G(\mathbf{a}_i, b_i, \mathbf{x})\}$ is said randomly generated if the corresponding parameters $(\mathbf{a}_i, b_i)$ are randomly generated from $\mathbf{R}^d \times R$ or $\mathbf{R}^d \times R^+$ based on a continuous sampling distribution probability.

Lemma 2.1 (Proposition 1 of [16]) *Given* $g : R \to R$, $\mathrm{span}\{g(\mathbf{a}\cdot\mathbf{x} + b) : (\mathbf{a}, b) \in \mathbf{R}^d \times R\}$ *is dense in* L^p *for every* $p \in [1, \infty)$, *if and only if* g *is not a polynomial (almost everywhere).*

Lemma 2.2 [17] *Let* $k : \mathbf{R}^d \to R$ *be an integrable bounded function such that* k *is continuous (almost everywhere) and* $\int_{\mathbf{R}}^d k(\mathbf{x})\, d\mathbf{x} \neq 0$. *Then span*$\{k(\frac{\mathbf{x}-\mathbf{a}}{b}) : (\mathbf{a}, b) \in \mathbf{R}^d \times R^+\}$ *is dense in* L^p *for every* $p \in [1, \infty)$.

Lemmas 2.1 and 2.2 show that feedforward neural networks with additive or RBF hidden nodes can approximate any target continuous function provided that the hidden node parameters $(\mathbf{a}_i, b_i)$ are tuned properly and appropriate values are given. Lemmas 2.1 and 2.2 only show the universal approximation capability of feedforward neural networks with additive or RBF hidden nodes, however, how to find the suitable hidden node parameters $(\mathbf{a}_i, b_i)$ remains open, and many tuning based learning algorithms have been suggested in the past. Huang et al. [7] proved that given any *bounded nonconstant piecewise continuous* activation function $g : R \to R$ for additive nodes or *integrable piecewise continuous* activation function $g : R \to R$ (and $\int_R g(x)\, dx \neq 0$) for RBF nodes, the hidden layer of such SLFN need not be tuned, in fact, all the hidden nodes can be randomly generated. SLFNs with randomly generated hidden nodes can universally approximate any target functions. Let $e_L \equiv f - f_L$ denote the residual error function for the current network f_L with L hidden nodes where $f \in L^2(X)$ is the target function. The output layer may have more than one nodes, $m > 1$, that is, the function f is a multi-output function: $f = [f^{(1)}, \ldots, f^{(m)}]^T$. The corresponding output function of the network with L hidden nodes is $f_L = [f_L^{(1)}, \ldots, f_L^{(m)}]^T$. Let $\beta_L^{(j)}$ denote the output weight between the Lth hidden node and the jth output node, and $e_L^{(j)} \equiv f^{(j)} - f_L^{(j)}$ the residual error function of the jth output node of the network with L hidden nodes, $j = 1, \ldots, m$. In theory, we have

Theorem 2.3 [7] *Given any bounded nonconstant piecewise continuous function $g : R \to R$ for additive nodes or any integrable piecewise continuous function $g : R \to R$ and $\int_R g(x)dx \neq 0$ for RBF nodes, for any continuous target function f and any randomly generated function sequence $\{g_L\}$, $\lim_{L\to\infty} \|f - f_L\| = 0$ holds with probability one if*

$$\beta_L^{(j)} = \frac{\langle e_{L-1}^{(j)}, g_L \rangle}{\|g_L\|^2}, \quad j = 1, \ldots, m. \tag{14}$$

Theorem 2.3 can be further extended from additive or RBF hidden nodes cases to "generalized" SLFNs [8, 9]. Given a type of piecewise computational hidden nodes (possibly not neural alike nodes), if SLFNs can work as universal approximators with adjustable hidden parameters, from a function approximation point of view the hidden node parameters of such "generalized" SLFNs can actually be randomly generated according to any continuous sampling distribution. In theory, the parameters of these SLFNs can be analytically determined by ELM instead of being tuned. Tuning is actually not required in such generalized SLFNs which include sigmoid networks, RBF networks, trigonometric networks, threshold networks, fully complex neural networks, high-order networks, ridge polynomial networks, etc.

Theorem 2.4 [8, 9] *Given any nonconstant piecewise continuous function $g : R \to R$, if $span\{G(\mathbf{a}, b, \mathbf{x}) : (\mathbf{a}, b) \in \mathbf{R}^d \times R\}$ is dense in L^2, for any continuous target function f and any function sequence $\{g_L(\mathbf{x}) = G(\mathbf{a}_L, b_L, \mathbf{x})\}$ randomly generated based on any continuous sampling distribution, $\lim_{L\to\infty} \|f - f_L\| = 0$ holds with probability one if the output weights $\boldsymbol{\beta}_i$ are determined by ordinary least square to minimize $\|f(\mathbf{x}) - \sum_{i=1}^{L} \boldsymbol{\beta}_i g_i(\mathbf{x})\|$.*

Theorem 2.4 means that ELM with fixed network architectures [5, 6, 35, 36] where the output parameters are determined by ordinary least square can work as universal approximators if only the activation function g is nonconstant piecewise and $span\{G(\mathbf{a}, b, \mathbf{x}) : (\mathbf{a}, b) \in \mathbf{R}^d \times R\}$ is dense in L^2.

3 ELM

The essence of ELM is that:

1. The hidden layer of ELM need not be iteratively tuned [5, 6].
2. According to feedforward neural network theory [10], both the training error $\|\mathbf{H}\boldsymbol{\beta} - \mathbf{T}\|$ and the norm of weights $\|\boldsymbol{\beta}\|$ need to be minimized [5, 6].
3. The hidden layer feature mapping need to satisfy the universal approximation condition (Theorems 2.3 and 2.4) [7–9].

According to Theorems 2.1 and 2.4 the hidden nodes can be randomly generated, the only unknown parameters in SLFNs are the output weights vectors $\boldsymbol{\beta}_i$ between the hidden layer and the output layer, which can simply be resolved by ordinary least-square directly.

3.1 Basic ELM [5, 6]

Hidden node parameters $(\mathbf{a}_i, b_i)$ remain fixed after randomly generated. To train an SLFN is simply equivalent to finding a least-squares solution $\hat{\boldsymbol{\beta}}$ of the linear system $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$:

$$\|\mathbf{H}\hat{\boldsymbol{\beta}} - \mathbf{T}\| = \min_{\boldsymbol{\beta}} \|\mathbf{H}\boldsymbol{\beta} - \mathbf{T}\| \tag{15}$$

If the number L of hidden nodes is equal to the number N of distinct training samples, $L = N$, according to Theorem 2.1 matrix $\mathbf{H}$ is square and invertible when hidden node parameters $(\mathbf{a}_i, b_i)$ are randomly chosen, and thus SLFNs can approximate these training samples with zero error.However, in most cases the number of hidden nodes is much less than the number of distinct training samples, $L \ll N$, $\mathbf{H}$ is a nonsquare matrix and there may not exist $\mathbf{a}_i, b_i, \boldsymbol{\beta}_i$ $(i = 1, \ldots, L)$ such that $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$. The smallest norm least-squares solution of the above linear system is:

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

where $\mathbf{H}^\dagger$ is the *Moore–Penrose generalized inverse* of matrix $\mathbf{H}$ [37, 38]. Thus, ELM can be summarized as follows:

Algorithm ELM: Given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbf{R}^d, \mathbf{t}_i \in \mathbf{R}^m, i = 1, \ldots, N\}$, hidden node output function $G(\mathbf{a}_i, b_i, \mathbf{x})$, and hidden node number L,

```
step 1   Randomly generate hidden node parameters $(\mathbf{a}_i, b_i), i = 1, \ldots, L$.
step 2   Calculate the hidden layer output matrix $\mathbf{H}$.
step 3   Calculate the output weight vector $\boldsymbol{\beta}$ :
```

$$\boldsymbol{\beta} = \mathbf{H}^\dagger \mathbf{T} \tag{17}$$

ELM algorithm can work with wide type of activation function. Many popular learning algorithms do not deal with threshold networks directly. Instead some analog networks are used to approximate threshold networks such that gradient-descent method can finally be used [27]. However, ELM can be used to train threshold networks directly [36]. Different methods can be used to calculate Moore–Penrose generalized inverse of a matrix: orthogonal projection method, orthogonalization method, iterative method, and singular value decomposition (SVD) [38].

3.2 Random hidden layer feature mapping based ELM [39]

The orthogonal projection method can be efficiently used in ELM [39]: $\mathbf{H}^\dagger = \left(\mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T$ if $\mathbf{H}^T\mathbf{H}$ is nonsingular or $\mathbf{H}^\dagger = \mathbf{H}^T\left(\mathbf{H}\mathbf{H}^T\right)^{-1}$ if $\mathbf{H}\mathbf{H}^T$ is nonsingular. According to the ridge regression theory [40], it was suggested [39, 41] that a positive value $1/\lambda$ is added to the diagonal of $\mathbf{H}^T\mathbf{H}$ or $\mathbf{H}\mathbf{H}^T$ in the calculation of the output weights $\boldsymbol{\beta}$. The resultant solution is stabler and tends to have better generalization performance. That is, in order to improve the stability of ELM we can have

$$\boldsymbol{\beta} = \mathbf{H}^T\left(\frac{\mathbf{I}}{\lambda} + \mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{T} \tag{18}$$

and the corresponding output function of ELM is:

$$f(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta} = \mathbf{h}(\mathbf{x})\mathbf{H}^T\left(\frac{\mathbf{I}}{\lambda} + \mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{T} \tag{19}$$

Or we can have

$$\boldsymbol{\beta} = \left(\frac{\mathbf{I}}{\lambda} + \mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T} \tag{20}$$

and the corresponding output function of ELM is:

$$f(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta} = \mathbf{h}(\mathbf{x})\left(\frac{\mathbf{I}}{\lambda} + \mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T} \tag{21}$$

Huang et al. [39] shows that the solutions (18) and (20) are actually consistent to minimize $\|\mathbf{H}\boldsymbol{\beta} - \mathbf{T}\|^2 + \lambda\|\boldsymbol{\beta}\|^2$, which is the essential target of ELM as mentioned before. Thus, ELM algorithm can be rewritten as follows: **Algorithm ELM:** Given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i)|\mathbf{x}_i \in \mathbf{R}^d, \mathbf{t}_i \in \mathbf{R}^m, i = 1, \ldots, N\}$, hidden node output function $G(\mathbf{a}_i, b_i, \mathbf{x})$, and hidden node number L,

```
step 1  Randomly generate hidden node parameters $(\mathbf{a}_i, b_i)$, $i = 1, \ldots, L$.
step 2  Calculate the hidden layer output matrix $\mathbf{H}$.
step 3  Calculate the output weight vector $\boldsymbol{\beta}$ :
```

$$\boldsymbol{\beta} = \mathbf{H}^T\left(\frac{\mathbf{I}}{\lambda} + \mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{T} \tag{22}$$

or

$$\boldsymbol{\beta} = \left(\frac{\mathbf{I}}{\lambda} + \mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T} \tag{23}$$

In these implementations, the condition on the number of hidden nodes can be mild, it does not closely depend on the number of training samples N. It works for both the cases $L < N$ or $L \geq N$. This is different from the interpolation theorem which requires $L \leq N$ (Theorem 2.1), but consistent to the universal approximation theorem (Theorem 2.4). Figure 3 shows a classification boundary obtained by ELM for a binary-class case. Formula (18) is used in this testing case and the number of hidden nodes L is much larger than the number of training samples. Toh [41] and Deng et al. [42] studied such regularization enhancement under *sigmoid* additive type of SLFNs. Deng et al. [42] and Man et al. [43] focused on obtaining the analytical solution (21) based on optimization methods. Toh [41] proposed a corresponding total error rate based multi-class solution of ELM (TER-ELM). Miche et al. [44] studied ELM with a cascade of two regularization penalties. Huang et al. [39] further extended this study to *generalized SLFNs* with *different type of hidden nodes (feature mappings)* as well as *kernels* and showed that the simple unified algorithm of ELM can be obtained for regression, binary and multi-label classification cases which, however, have to be handled separately by SVMs and its variants [2, 45–49].

3.3 Kernel based ELM [39]

Huang et al. [39] also studied the kernel based ELM. If the hidden layer feature mapping $\mathbf{h}(\mathbf{x})$ is unknown to users, one can define a kernel matrix for ELM as follows:

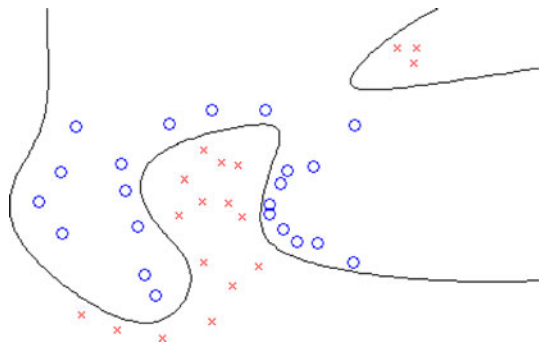

Fig. 3 Classification boundary obtained by ELM for a binary class classification: $L = 10^3$ and $\lambda = 10^5$

$$\mathbf{\Omega}_{ELM} = \mathbf{H}\mathbf{H}^T : \Omega_{ELM\,i,j} = \mathbf{h}(\mathbf{x}_i)\cdot\mathbf{h}(\mathbf{x}_j) = K(\mathbf{x}_i,\mathbf{x}_j) \tag{24}$$

Then the output function of ELM (19) can be written compactly as:

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{h}(\mathbf{x})\mathbf{H}^T\left(\frac{\mathbf{I}}{\lambda}+\mathbf{H}\mathbf{H}^T\right)^{-1}\mathbf{T} \\ &= \begin{bmatrix} K(\mathbf{x},\mathbf{x}_1) \\ \vdots \\ K(\mathbf{x},\mathbf{x}_N) \end{bmatrix}^T \left(\frac{\mathbf{I}}{\lambda}+\mathbf{\Omega}_{ELM}\right)^{-1}\mathbf{T} \end{aligned} \tag{25}$$

In this specific kernel implementation of ELM, the hidden layer feature mapping $\mathbf{h}(\mathbf{x})$ need not be known to users, instead its corresponding kernel $K(\mathbf{u},\ \mathbf{v})$ (e.g. $K(\mathbf{u},\mathbf{v}) = \exp(-\gamma\|\mathbf{u}-\mathbf{v}\|^2)$) is given to users. The number of hidden nodes L (the dimensionality of the hidden layer feature space) need not be specified either. Thus, the ELM algorithm can be rewritten for the kernel case as follows: **Algorithm ELM** (*single-step kernel version*): Given a training set $\aleph = \{(\mathbf{x}_i,\mathbf{t}_i)|\mathbf{x}_i \in \mathbf{R}^d, \mathbf{t}_i \in \mathbf{R}^m, i = 1,\ldots,N\}$, kernel $K(\mathbf{u},\ \mathbf{v})$: Calculate the output function:

$$f(\mathbf{x}) = \begin{bmatrix} K(\mathbf{x},\mathbf{x}_1) \\ \vdots \\ K(\mathbf{x},\mathbf{x}_N) \end{bmatrix}^T \left(\frac{\mathbf{I}}{\lambda}+\mathbf{\Omega}_{ELM}\right)^{-1}\mathbf{T} \tag{26}$$

It can be seen that kernel based ELM algorithm can be implemented in a *single* learning step. Frénay and Verleysen [50, 51] studied the kernel implementation of ELM if $\mathbf{h}(\mathbf{x})$ is known to users. If the hidden layer feature mapping $\mathbf{h}(\mathbf{x})$ is known to users, Frénay and Verleysen [51] defined the ELM kernel as

$$K(\mathbf{u},\mathbf{v}) = \lim_{L\to+\infty}\frac{1}{L}\mathbf{h}(\mathbf{u})\cdot\mathbf{h}(\mathbf{v}) \tag{27}$$

A parameter-insensitive kernel with analytic form can then be obtained for SVM for regression, which significantly reduces the computational complexity. We conjecture that Frénay and Verleysen's ELM kernel [51] can work for SVM and its variants as well as in (25). All the above mentioned can be applied in regression, binary and multi-label classification applications directly. ELMs can be applied to complex space as well.

4 Fully complex ELM

In high speed digital communication systems, equalizers are very often used at receivers to recover the original symbols from the received signals [34, 52]. Two conventional approaches are usually used for solving equalization problems.

1. Real-valued neural network models such as feedforward neural networks, RBF networks and recurrent neural networks.
2. Complex-valued neural networks: this approach has attracted considerable attention in channel equalization applications in the past 15 years [53–55]. Split-complex activation (basis) functions consisting of two real-valued activation functions, one processing the real part and the other processing the imaginary part, have been traditionally employed in these complex-valued neural networks.

Instead of using split-complex activation function, extreme learning machine can use fully complex activation function directly. Li et al [34] proved the universal approximation capability of extreme learning machine with fully complex activation function:

Theorem 4.1 [34] *Given any complex continuous discriminatory or any complex bounded nonlinear piecewise continuous function $\sigma : C \to C$, for any target complex continuous function $f : \mathbf{C}^d \to C$ and any randomly generated function sequence $\{g_L = \prod_{l=1}^{s_n}\sigma(\mathbf{a}_{Ll}\cdot\mathbf{z}+b_L)\}$, $\lim_{L\to\infty}\|f-f_L\| = 0$ holds with probability one if*

$$\beta_L^{(j)} = \frac{\langle e_{L-1}^{(j)}, g_L\rangle}{\|g_L\|^2}, \quad j = 1,\ldots,m. \tag{28}$$

When the network architecture is fixed (with fixed L), from Theorem 4.1 we have

Theorem 4.2 [34] *Given any complex continuous discriminatory or any complex bounded nonlinear piecewise continuous function $\sigma : C \to C$, for any continuous target function $f : \mathbf{C}^d \to C$ and any function sequence $\{g_L = \prod_{l=1}^{s_L}\sigma(\mathbf{a}_{Ll}\cdot\mathbf{z}+b_L)\}$ randomly generated based on any continuous sampling distribution probability, $\lim_{L\to\infty}\|f - f_L\| = 0$ holds with probability one if the output weights $\boldsymbol{\beta}_i$ are determined by ordinary least square to minimize $\left\|f(\mathbf{z}) - \sum_{i=1}^{L}\boldsymbol{\beta}_i g_i(\mathbf{z})\right\|$.*

Thus, the ELM algorithms introduced in Sect. 3 can be linearly extended to the complex domain. Compared to others equalizers, ELM can obtain much lower symbol error rate (SER) and provide parsimonious structures for applications in the complex domain [52].

5 Online sequential ELM (OS-ELM)

ELM algorithms introduced in Sect. 3 learn training samples only after all training samples are ready. In many industrial applications training data may come one by one or chunk by chunk. In these cases, on-line sequential learning algorithms are preferred over batch learning algorithms as sequential learning algorithms do not require retraining whenever a new data is received. Sequential learning is difficult to be implemented for feedforward neural networks with additive (e.g. [56]) or RBF hidden nodes [57–64]. Most of the conventional online sequential learning algorithms have several parameters for users to specify and it is very time-consuming to tune those parameters. OS-ELM [65] is a simple and efficient online sequential learning algorithm that can handle both additive and RBF nodes in a unified framework. 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. The training observations are sequentially presented to the learning algorithm (one-by-one or chunk-by-chunk with varying or fixed chunk length). A single or a chunk of training observations is discarded and may not be used any more as soon as the learning procedure for that particular observation(s) is completed. According to Sect. 3, one of the solutions of the output weight vector $\boldsymbol{\beta}$ is:

$$\boldsymbol{\beta} = (\mathbf{H}^{\mathbf{T}}\mathbf{H})^{-1}\mathbf{H}^{\mathbf{T}}\mathbf{T} \tag{29}$$

Sequential implementation of the *least-squares solution* of Eq. 29 results in the OS-ELM which uses the recursive least squares algorithm [66].

OS-ELM Algorithm: [65]

step 1 **Initialization Phase:** Initialize the learning using a small chunk of initial training data $\aleph_0 = \{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=1}^{N_0}$ from the given training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) | \mathbf{x}_i \in \mathbf{R}^n, \mathbf{t}_i \in \mathbf{R}^m, i = 1, \ldots\}$, $N_0 \geq L$.

(a) Randomly generate the hidden node parameters $(\mathbf{a}_i, b_i), i = 1, \ldots, L$.

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

$$\mathbf{H}_0 = \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_1) & \cdots & G(\mathbf{a}_L, b_L, \mathbf{x}_1) \\ \vdots & \cdots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_{N_0}) & \cdots & G(\mathbf{a}_L, b_L, \mathbf{x}_{N_0}) \end{bmatrix}_{N_0 \times L} \tag{30}$$

(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 $\mathbf{T}_0 = [\mathbf{t}_1, \ldots, \mathbf{t}_{N_0}]^T$.

(d) Set $k = 0$.

step 2 **Sequential Learning Phase:**

(a) Present the $(k+1)$th chunk of new observations: $\aleph_{k+1} = \{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=\left(\sum_{j=0}^{k} N_j\right)+1}^{\sum_{j=0}^{k+1} N_j}$, where N_{k+1} denotes the number of observations in the $(k+1)$th chunk.

(b) Calculate the partial hidden layer output matrix $\mathbf{H}_{k+1}$ for the $(k+1)$th chunk of data $\aleph_{k+1}$:

$$\mathbf{H}_{k+1} = \begin{bmatrix} G\left(\mathbf{a}_1, b_1, \mathbf{x}_{\left(\sum_{j=0}^{k} N_j\right)+1}\right) & \cdots & G\left(\mathbf{a}_L, b_L, \mathbf{x}_{\left(\sum_{j=0}^{k} N_j\right)+1}\right) \\ \vdots & \cdots & \vdots \\ G\left(\mathbf{a}_1, b_1, \mathbf{x}_{\sum_{j=0}^{k+1} N_j}\right) & \cdots & G\left(\mathbf{a}_L, b_L, \mathbf{x}_{\sum_{j=0}^{k+1} N_j}\right) \end{bmatrix}_{N_{k+1} \times L} \tag{31}$$

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

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

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \mathbf{P}_k \mathbf{H}_{k+1}^T (\mathbf{I} + \mathbf{H}_{k+1} \mathbf{P}_k \mathbf{H}_{k+1}^T)^{-1} \mathbf{H}_{k+1} \mathbf{P}_k$$
$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} + \mathbf{P}_{k+1} \mathbf{H}_{k+1}^T (\mathbf{T}_{k+1} - \mathbf{H}_{k+1} \boldsymbol{\beta}^{(k)}) \tag{32}$$

(d) Set $k = k + 1$. Go to *step* 2a.

Seen from the above OS-ELM algorithm, OS-ELM and ELM can achieve the same learning performance (training error and generalization accuracy) when $rank(\mathbf{H}_0) = L$. In addition, if $N_0 = N$, OS-ELM also becomes the batch ELM. In OS-ELM, the chunk size of incoming training data need not be constant. When the training data is received one-by-one instead of chunk-by-chunk, $N_{k+1} \equiv 1$, formula (32) has the following simple format (Sherman-Morrison formula [67]):

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \frac{\mathbf{P}_k \mathbf{h}(\mathbf{x}_{k+1}) \mathbf{h}^T(\mathbf{x}_{k+1}) \mathbf{P}_k}{1 + \mathbf{h}^T(\mathbf{x}_{k+1}) \mathbf{P}_k \mathbf{h}(\mathbf{x}_{k+1})}$$
$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} + \mathbf{P}_{k+1} \mathbf{h}(\mathbf{x}_{k+1}) (\mathbf{t}_{k+1}^T - \mathbf{h}^T(\mathbf{x}_{k+1}) \boldsymbol{\beta}^{(k)}) \tag{33}$$

where $\mathbf{h}(\mathbf{x}_{k+1}) = [G(\mathbf{a}_1, b_1, \mathbf{x}_{k+1}) \cdots G(\mathbf{a}_L, b_L, \mathbf{x}_{k+1})]$. OS-ELM is efficient in time-series prediction which is required in many real-world problems. The chaotic Mackey–Glass differential delay equation [68] is one of the classical benchmark time series problems in literature:

$$\frac{dx(t)}{dt} = \frac{ax(t-\tau)}{1 + x^{10}(t-\tau)} - bx(t) \tag{34}$$

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:

$$x(t+\Delta t) = \frac{2 - b\Delta t}{2 + \Delta t} x(t) + \frac{a\Delta t}{2 + b\Delta t} \left[\frac{x(t+\Delta t - \tau)}{1 + x^{10}(t + \Delta t - \tau)} + \frac{x(t-\tau)}{1 + x^{10}(t-\tau)} \right] \tag{35}$$

The time series is generated under the condition $x(t-\tau) = 0.3$ for $0 \leq t \leq \tau$ and predicted with $\upsilon = 50$

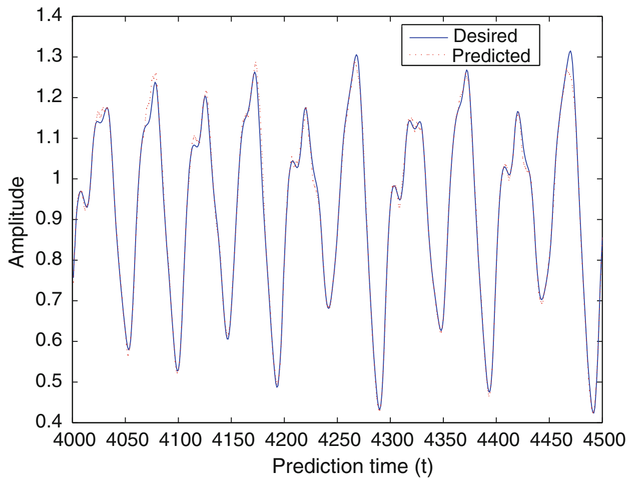

Fig. 4 Time-series prediction: the approximated curve obtained by OS-ELM. OS-ELM is trained by the training observations is from $t = 1$ to $t = 4, 000$, and the predicted period is from $t = 4{,}001$ to $t = 4{,}500$

sample steps ahead using the four past samples: $s_{n-\upsilon}$, $s_{n-\upsilon-6}$, $s_{n-\upsilon-12}$, and $s_{n-\upsilon-18}$. Hence, the nth input–output instance is:

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

Figure 4 shows the approximated curve of OS-ELM in this time-series prediction. In this simulation, $\Delta t = 1$, and the training observations is from $t = 1$ to $t = 4{,}000$ and the testing observations from $t = 4{,}001$ to $t = 4{,}500$. The number of hidden nodes of OS-ELM is $L = 120$.

6 Incremental ELM (I-ELM)

The universal approximation capability of ELMs was proved using incremental learning method where the hidden nodes are added one by one [7–9]. The proof itself is indeed a practical incremental constructive method, which actually shows an efficient way to construct an incremental feed-forward network (referred to as I-ELMs). Different from other incremental learning algorithms which may only work with some type of hidden nodes (e.g. resource allocation network and its variants [57, 58, 60, 63, 64] work only for RBF networks), I-ELM can work well with a wide type of activation functions no matter whether they are sigmoidal or nonsigmoidal, continuous or noncontinuous, and differentiable or non-differentiable. The traditional gradient-descent based learning algorithms cannot be applied to networks with non-differential activation functions such as threshold networks since the required derivatives are not available. These conventional methods may also face local minima issues. Although many other incremental learning algorithms have been proposed in literature [57–60, 63, 64], unlike I-ELM the universal approximation capability of these previous learning algorithms has not been proved. Different from other conventional incremental learning algorithms which may have several parameters for us to specify, I-ELM has no parameters for users to specify except the maximum network architecture and the expected accuracy. Experimental results show that I-ELM outperforms other learning algorithms (including support vector regression (SVR) [69, 70], stochastic gradient-descent BP [56], and incremental RBF networks (RAN [57], RANEKF [58], MRAN [59, 60], GAP-RBF [63], GGAP-RBF [64]) in terms of generalization performance and learning speed. I-ELMs can be implemented in different ways:

1. *Basic I-ELM* Every time only one hidden node is randomly generated and added to the existing network [7, 8].
2. *Enhanced I-ELM* Every time k hidden nodes are randomly generated. However, among the k randomly generated hidden nodes only the most appropriate hidden node will be added to the existing network [9].

Compared to the original I-ELM [7, 8], this enhanced implementation [9] will produce a more compact network architecture and the learning can be completed in a faster convergence rate and learning speed. I-ELM is a specific case of EI-ELM when $k = 1$.

Theorem 6.1 [9] *Given a SLFN with any nonconstant piecewise continuous hidden nodes* $G(\mathbf{a}, b, \mathbf{x})$, *if* span$\{G(\mathbf{a}, b, \mathbf{x}) : (\mathbf{a}, b) \in \mathbf{R}^d \times R\}$ *is dense in* L^2, *for any continuous target function* f *and any randomly generated function sequence* $\{g_L\}$ *and any positive integer* k, $\lim_{L\to\infty} \|f - f_L^*\| = 0$ *holds with probability one if*

$$\beta_L^{(j)*} = \frac{\langle e_{L-1}^{(j)*}, g_L^* \rangle}{\|g_L^*\|^2}, \quad j = 1, \ldots, m \tag{36}$$

where $f_L^{(j)*} = \sum_{i=1}^{L} \beta_i^{(j)*} g_i^*$, $e_L^{(j)*} = f^{(j)} - f_L^{(j)*}$ *and* $g_L^* = \{g_i | \min_{(L-1)k+1 \le i \le Lk} \|(f^{(j)} - f_{L-1}^{(j)*}) - \beta_L^{(j)} g_i\|\}$.

According to formula (36), the weight $\beta_L^{(j)}$ between the Lth newly added node and the jth output node should be chosen as $\frac{\langle e_{L-1}^{(j)*}, g_L^* \rangle}{\|g_L^*\|^2}$. In real applications, only the training samples are available, the target function $f(\mathbf{x})$ is unknown and the exact functional form of $e_{L-1}^{(j)*}$ is not available, thus, formula (36) cannot be calculated explicitly. Instead, formula (36) can be estimated based on the training samples:

$$\beta_L^{(j)} = \frac{\mathbf{E}^{(j)} \cdot \hat{\mathbf{h}}^T}{\hat{\mathbf{h}} \cdot \hat{\mathbf{h}}^T} = \frac{\sum_{p=1}^{N} e^{(j)}(p) G(\mathbf{a}_L, b_L, \mathbf{x}_p)}{\sum_{p=1}^{N} G^2(\mathbf{a}_L, b_L, \mathbf{x}_p)} \tag{37}$$

where $e^{(j)}(p)$ is the corresponding residual error of the jth output node before the Lth new hidden neuron is added. $\hat{\mathbf{h}} = [G(\mathbf{a}_L, b_L, \mathbf{x}_1), \ldots, G(\mathbf{a}_L, b_L, \mathbf{x}_N)]^T$ is the activation vector of the newly added node for all the N training samples and $\mathbf{E}^{(j)} = [e^{(j)}(1), \ldots, e^{(j)}(N)]^T$ is the residual vector the jth output node with respect to all the N training samples before this new hidden node added. Let $\mathbf{E} = [\mathbf{E}^{(1)}, \ldots, \mathbf{E}^{(m)}]$.

EI-ELM Algorithm: Given a training set $\aleph = \{(\mathbf{x}_i, t_i) | \mathbf{x}_i \in \mathbf{R}^d, t_i \in R, i = 1, \ldots, N\}$, hidden node output function $G(\mathbf{a}, b, \mathbf{x})$, maximum number $L_{\max}$ of hidden nodes, maximum number k of trials of assigning random hidden nodes at each step, and expected learning accuracy ϵ,

```
step 1 Initialization: Let $L = 0$ and residual error $\mathbf{E} = \mathbf{T}$.
step 2 Learning step:
    while $L < L_{\max}$ and $\|\mathbf{E}\| > \epsilon$
(a) Increase by 1 the number of hidden nodes $L$: $L = L + 1$.
(b) for $i = 1 : k$
        (i) Assign random parameters $(\mathbf{a}_{(i)}, b_{(i)})$ for the new hidden node $L$ according to any continuous sampling distribution probability.
        (ii) Calculate the output weight $\beta_{(i)}^{(j)}$ for the new hidden node:
             $\beta_{(i)}^{(j)} = \frac{\mathbf{E}^{(j)} \cdot \hat{\mathbf{h}}_{(i)}^T}{\hat{\mathbf{h}}_{(i)} \cdot \hat{\mathbf{h}}_{(i)}^T}, j = 1, \ldots, m$    (38)
        (iii) Calculate the residual error after adding the new hidden node $L$:
             $\mathbf{E}_{(i)}^{(j)} = \mathbf{E}^{(j)} - \beta_{(i)}^{(j)} \hat{\mathbf{h}}_{(i)}, j = 1, \ldots, m$    (39)
    endfor
(c) Let $i^* = \{i | \min_{1 \le i \le k} \|\mathbf{E}_{(i)}\|\}$ where $\mathbf{E}_{(i)} = [\mathbf{E}_{(i)}^{(1)}, \ldots, \mathbf{E}_{(i)}^{(m)}]$. Set $\mathbf{E} = \mathbf{E}_{(i)}$, $\mathbf{a}_L = \mathbf{a}_{(i^*)}$, $b_L = b_{(i^*)}$, and $\boldsymbol{\beta}_L = \boldsymbol{\beta}_{(i^*)}$.
endwhile
```

Before learning, there is no node in the network and the initial residual error is set as the expected target vector $\mathbf{T}$ of the training data set as shown in *step* 1. Learning will stop when the number L of hidden nodes has exceeded the predefined maximum number $L_{\max}$ or the residual error $\mathbf{E}$ is small enough ($\|\mathbf{E}\| < \epsilon$). *step* 2b randomly generates k new hidden nodes and *step* 2c will choose and add the most appropriate hidden node of the k randomly generated hidden nodes. $\hat{\mathbf{h}}_{(i)}$ in formula (38) is the activation vector of the ith trial of hidden node for all the N training samples and $\beta_{(i)}^{(j)}$ is the corresponding output weight between the ith trial of hidden node and the jth output node. $\mathbf{E}_{(i)}^{(j)}$ of formula (39) is the residual error vector of the jth output node if the ith trial of hidden node is added. $\mathbf{E}^{(j)}$ in the right hand of formula (39) represents the earlier residual error vector corresponding to the jth output node before the new node added.

7 ELM ensembles

The idea of neural network ensemble was proposed by Hansen and Salamon [71] . Their work showed that a single network's performance can be expected to improve using an ensemble of neural networks with a plurality consensus scheme. This technique has been spread widely after that. The most prevailing approaches for training neural networks comprised the ensemble are Bagging [72] and Boosting [73–75]. An integration of several ELMs was proposed by Sun et al [76] to predict the future sales amount. Several ELM networks were connected in parallel and the average of the ELMs' outputs was used as the final predicted sales amount. The resulting ensemble has better generalization performance. Heeswijk et al. [77] investigated the adaptive ensemble models of ELM on the application of one-step ahead prediction in (non-)stationary time series. It was verified that the method did work on stationary time series and the capability of the method on non-stationary time series was tested. The empirical studies showed that the adaptive ensemble model achieved an acceptable testing error with good adaptivity. Heeswijk et al. [78] also studied ELM ensemble for large scale regression applications. Furthermore, network ensembles are potentially important methods to perform sequential learning [79, 80]. Network ensemble consists of a few of single networks that may have different adaptabilities to the new data. Some of the networks in the ensemble may adapt faster and better to the new data than others, which could make the ensemble overcome the problem of networks that could not adapt well to the new data. Lan et al. [81] proposed an integrated network structure, which is called ensemble of online sequential ELM (EOS-ELM). EOS-ELM comprised several OS-ELM networks. The average value of outputs of each OS-ELM in the ensemble was used as the final measurement of network performance. The simulation results proved that EOS-ELM is more stable than original OS-ELM in each trial of simulation for most problems.

8 Pruning ELM

Rong et al. [82] presented a pruned ELM (referred to as P-ELM) as a systematic and automated method for ELM classifier network design. It starts with a large network and then eliminates the hidden nodes that have low relevance to the class labels by using statistical criteria, namely, the Chi-squared (χ^2) and information gain (IG) measures. P-ELM mainly focuses on pattern classification applications. Another pruning algorithm called optimally-pruned ELM (referred to as OP-ELM) was proposed by Miche et al. [83]. The OP-ELM methodology has three steps: (1) build the SLFN using the original ELM algorithm; (2) rank the hidden nodes by applying multi-response sparse regression algorithm (MRSR) [84]; and (3) select the hidden nodes through leave-one-out (LOO) validation. OP-ELM is applicable for both regression and classification applications.

9 Constructive model selection of ELM

9.1 Error minimized ELM

Error minimized ELM (EM-ELM) [85] is an error minimization based method in which the number of hidden nodes can grow one-by-one or group-by-group until optimal. The approach can significantly reduce the computational complexity and its convergence was proved as well. In EM-ELM, the hidden nodes are randomly generated and added to the network sequentially. Further study of EM-ELM shows [86] that some newly added hidden nodes may be more efficient in reducing the residual error as compared to other hidden nodes. Hence, an enhancement of EM-ELM (referred to as EEM-ELM) [86] was proposed by applying random search method. In the enhancement of EM-ELM, the hidden node is added to the network one-by-one. At each incremental learning step, k hidden nodes are randomly generated and the hidden node that leads to highest residual error reduction will be added to the network, and then the output weights are updated incrementally in the same way of original EM-ELM.

9.2 Stepwise forward selection based constructive ELM for regression

Instead of using a simple selection method that randomly generates a group of hidden nodes in each step of the training process (i.e. like in EEM-ELM), one could randomly generate a large number of hidden nodes as the candidate reservoir and then pick the hidden node one-by-one via a stepwise forward selection method. The fast construction algorithm (FCA) proposed in [87] is a constructive hidden node selection method for ELM based on orthogonal least squares (OLS). OLS selects a suitable set of variables to form the subset model from a large set of candidates. At each step, the net decrease in the residual error is maximized. The key advantage of the algorithm is that it can explicitly identify the net contribution of the newly added node without solving the whole least-squares problem, which significantly reduces the computational complexity. However, OLS cannot guarantee an optimal solution because it is greedy and on the basis of a local optimization [88]. By modifying the classic forward selection algorithm, a constructive hidden nodes selection method for ELM (CS-ELM) [89] was proposed,which is less greedy and without any matrix decompositions. At each step of CS-ELM, the hidden node with an output that has the highest correlation with the current residual is selected.

9.3 Two-stage ELM for regression

It is found [90] that the parsimonious network structure is probably missed by some greedy selection methods due to the fact that the hidden nodes added earlier may become insignificant when other hidden nodes are added to the network. In FCA [87], the researchers solved this problem by adding a fine tuning phase after the forward selection, which reviewed the hidden nodes selected in forward selection phase and replaced the selected hidden nodes with candidate nodes that achieve more contribution. Inspired by the above mentioned CS-ELM and the FCA algorithm, a two-stage algorithm was proposed and it is called TS-ELM [90]. The first stage attempts to select hidden nodes by forward recursive algorithm and the selection is terminated by the final prediction error (FPE) criterion; while the second stage is a backward refinement phase that removes the insignificant hidden nodes by applying LOO method.

10 SVM with ELM feature mapping

SVM [2] has become one of the most popular classifiers. SVM has been extensively applied in wide type of applications. As explained in Cortes and Vapnik [2], SVM can be seen as a specific type of SLFNs, the so-called support vector networks. A multi-layer feedforward network (cf. Fig. 1) can be considered to transform the input data into a feature space Z of the last hidden layer [2, 3]. In order to find a solution of $z_i(\mathbf{x})$ where $z_i(\mathbf{x})$ is the activation function of the ith node of the last hidden layer, Cortes and Vapnik [2] proposed the support vector machine which maps the data from the input space to some high dimensional feature space Z through some nonlinear mapping $\phi(\mathbf{x}) : \mathbf{x}_i \rightarrow \phi(\mathbf{x}_i)$. Standard optimization methods are used to find the separating hyperplane which maximizes the separating margins of two different classes in the feature space:

$$\begin{aligned}&\text{minimize: } L_P = \frac{1}{2}\|\boldsymbol{\beta}\|^2 + \lambda\sum_{i=1}^{N}\xi_i\\&\text{subject to: } t_i(\boldsymbol{\beta}\cdot\phi(\mathbf{x}_i)+b)\ge 1-\xi_i, \quad i=1,\ldots,N\\&\qquad\qquad \xi_i\ge 0, \quad i=1,\ldots,N\end{aligned}\tag{40}$$

where λ is a user specified parameter and provides a tradeoff between the distance of the separating margin and the training error. Vectors $\mathbf{x}_i$ for which $t_i(\boldsymbol{\beta}\cdot\phi(\mathbf{x}_i)+b)=1$ is termed support vectors. The hyperplane $\mathbf{w}\cdot\phi(\mathbf{x})+b=0$ separates the training data with a maximal margin in the feature space. It maximizes the distance $2/\|\boldsymbol{\beta}\|$ between two different classes in the feature space Z. To train such a SVM is equivalent to solving the following dual optimization problem:

$$\begin{aligned}&\text{minimize: } L_D = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}t_it_j\alpha_i\alpha_j\phi(\mathbf{x}_i)\cdot\phi(\mathbf{x}_j) - \sum_{i=1}^{N}\alpha_i\\&\text{subject to: } \sum_{i=1}^{N}t_i\alpha_i = 0 \quad 0\le\alpha_i\le\lambda,\ i=1,\ldots,N\end{aligned}\tag{41}$$

where each Lagrange multiplier α_i corresponds to a training example $(\mathbf{x}_i, t_i)$. Kernel functions $K(\mathbf{u},\mathbf{v})=\phi(\mathbf{u})\cdot\phi(\mathbf{v})$ are usually used in the implementation of SVM learning algorithm:

$$\begin{aligned}&\text{minimize: } L_D = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}t_it_jK(\mathbf{x}_i,\mathbf{x}_j)\alpha_i\alpha_j - \sum_{i=1}^{N}\alpha_i\\&\text{subject to: } \sum_{i=1}^{N}t_i\alpha_i = 0 \quad 0\le\alpha_i\le\lambda,\ i=1,\ldots,N\end{aligned}\tag{42}$$

The SVM kernel function $K(\mathbf{u}, \mathbf{v})$ needs to satisfy Mercer's condition [2]. The decision function of SVM is:

$$f(\mathbf{x}) = \text{sign}\left(\sum_{s=1}^{N_s}\alpha_st_sK(\mathbf{x},\mathbf{x}_s)+b\right)\tag{43}$$

Liu et al. [91] and Frénay and Verleysen [50] made a significant contribution showing that (random) ELM kernels can be used in SVM and better generalization can be achieved. Their methods keep the same optimization methods as the conventional SVM. Further study [92] showed that SVM's optimization constrains can be milder if ELM kernel is used, and the optimal solution can be obtained more efficiently. ELM is to minimize the training error as well as the norm of the output weights [5, 6]:

$$\begin{aligned}&\text{Minimize: } \sum_{i=1}^{N}\|\boldsymbol{\beta}\cdot\mathrm{h}(\mathbf{x}_i)-t_i\|\\&\text{and}\\&\text{Minimize: } \|\boldsymbol{\beta}\|\end{aligned}\tag{44}$$

For the binary classification applications, the decision function of ELM is: $f(\mathbf{x}) = \text{sign}(\sum_{i=1}^{L}\beta_iG(\mathbf{a}_i,b_i,\mathbf{x})) = \text{sign}(\boldsymbol{\beta}\cdot\mathbf{h}(\mathbf{x}))$. In ELM, to minimize the norm of the output weights $\|\boldsymbol{\beta}\|$ is actually to maximize the distance of the separating margins of the two different classes in the ELM feature space: $2/\|\boldsymbol{\beta}\|$, which is similar to SVM's target. From the standard optimization theory point of view, the objective (44) of ELM in minimizing both the training errors and the output weights can be written as:

$$\begin{aligned}&\text{Minimize: } L_P = \frac{1}{2}\|\boldsymbol{\beta}\|^2 + \lambda\sum_{i=1}^{N}\xi_i\\&\text{Subject to: } t_i\boldsymbol{\beta}\cdot\mathbf{h}(\mathbf{x}_i)\ge 1-\xi_i, \quad i=1,\ldots,N\\&\qquad\qquad \xi_i\ge 0, \quad i=1,\ldots,N\end{aligned}\tag{45}$$

which is very similar to SVM's optimization problem (40) with two main differences:

1. Different from the conventional SVM, the randomness can be adopted in the ELM mapping $\mathbf{h}(\mathbf{x})$, that is, all the parameters of $\mathbf{h}(\mathbf{x})$ are chosen randomly.
2. The bias b is not required in the ELM's optimization constrains since in theory ELM with $\mathbf{h}(\mathbf{x})$ has universal approximation capability and the separating hyperplane in the ELM feature space tends to pass through the origin. In SVM, the feature mapping $\phi(\mathbf{x})$ is unknown and it is not required to satisfy universal approximation condition. However, in ELM, the feature mapping $\mathbf{h}(\mathbf{x})$ is required to satisfy universal approximation conditions (Theorems 2.3 and 2.4).

Based on the Karush–Kuhn–Tucker (KKT) conditions [93], the equivalent dual optimization problem can be obtained:

$$\begin{aligned}&\text{minimize: } L_D = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}t_it_j\alpha_i\alpha_j\mathbf{h}(\mathbf{x}_i)\cdot\mathbf{h}(\mathbf{x}_j) - \sum_{i=1}^{N}\alpha_i\\&\text{subject to: } 0\le\alpha_i\le\lambda, \quad i=1,\ldots,N\end{aligned}\tag{46}$$

As the separating hyperplane tends to pass through the origin in the ELM feature space, the above dual ELM optimization problem does not have the condition $\sum_{i=1}^{N}t_i\alpha_i=0, \quad \forall i$, which is, however, required in the conventional dual SVM optimization problem (41). With the ELM kernel (24) we have:

$$\begin{aligned}&\text{minimize: } L_D = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}t_it_jK(\mathbf{x}_i,\mathbf{x}_j)\alpha_i\alpha_j - \sum_{i=1}^{N}\alpha_i\\&\text{subject to: } 0\le\alpha_i\le\lambda, \quad i=1,\ldots,N\end{aligned}\tag{47}$$

The decision function of ELM is defined as

$$f(\mathbf{x}) = \text{sign}\left(\sum_{s=1}^{N_s}\alpha_st_sK(\mathbf{x},\mathbf{x}_s)\right)\tag{48}$$

Experimental results [92] have shown that the generalization performance of ELM is less sensitive to the user specified parameters especially the number of hidden nodes. Thus, compared to SVM, users can use ELM easily and effectively by avoiding tedious and time-consuming parameter tuning.

11 Conclusions

As a learning technique, ELM has demonstrated good potentials to resolving regression and classification problems. Recently, ELM techniques have received considerable attention in computational intelligence and machine learning communities, in both theoretic study and applications [41–44, 50, 51, 78, 80, 91, 94–119]. Fundamentals of ELM techniques are composed of twofold: universal approximation capability with random hidden layer, and various learning techniques with easy and fast implementations. The following issues on ELM remain open and may be worth investigating in the future.

1. As observed in experimental studies [6, 39], the performance of ELM is stable in a wide range of number of hidden nodes. Compared to the BP learning algorithm, the performance of ELM is not very sensitive to the number of hidden nodes. However, how to prove it in theory remains open.
2. One of the typical implementations of ELM is to use random nodes in the hidden layer and the hidden layer of SLFNs need not be tuned. It is interesting to see that the generalization performance of ELM turns out to be very stable. How to estimate the oscillation bound of the generalization performance of ELM remains open too.
3. It seems that ELM performs better than other conventional learning algorithms in applications with higher noise. How to prove it in theory is not clear.
4. Experimental results [6, 39, 92] show that compared to backpropagation algorithm, SVM and least-square SVM (LS-SVM) ELM usually achieve similar or better generalization in regression and classification applications. How to prove it in theory is still an open problem.
5. ELM provides a batch learning kernel solution (25) which is much simpler than other kernel learning algorithms such as LS-SVM [49]. It is known that it is not straightforward to have an efficient online sequential implementation of SVM and LS-SVM. However, due to the simplicity of ELM, it may be easier to implement the online sequential variant of the kernel based ELM (25).

Acknowledgments This research was sponsored by the grant from Academic Research Fund (AcRF) Tier 1 under project no. RG 22/08 (M52040128).

References

1. Rumelhart DE, Hinton GE, Williams RJ (1986) Learning representations by back-propagation errors. Nature 323:533–536
2. Cortes C, Vapnik V (1995) Support vector networks. Mach Learn 20(3):273–297
3. Rosenblatt F (1962) Principles of neurodynamics: perceptrons and the theory of brain mechanisms. Spartan Books, New York
4. Lowe D (1989) Adaptive radial basis function nonlinearities and the problem of generalisation. In: Proceedings of first IEE international conference on artificial neural networks, pp 171–175
5. Huang G-B, Zhu Q-Y, Siew C-K (2004) Extreme learning machine: a new learning scheme of feedforward neural networks. In: Proceedings of international joint conference on neural networks (IJCNN2004), vol 2, Budapest, Hungary, 25–29 July 2004, pp 985–990
6. Huang G-B, Zhu Q-Y, Siew C-K (2006) Extreme learning machine: theory and applications. Neurocomputing 70:489–501
7. Huang G-B, Chen L, Siew C-K (2006) Universal approximation using incremental constructive feedforward networks with random hidden nodes. IEEE Trans Neural Netw 17(4):879–892
8. Huang G-B, Chen L (2007) Convex incremental extreme learning machine. Neurocomputing 70:3056–3062
9. Huang G-B, Chen L (2008) Enhanced random search based incremental extreme learning machine. Neurocomputing 71:3460–3468
10. Bartlett PL (1998) The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. IEEE Trans Inf Theory 44(2):525–536
11. Huang S-C, Huang Y-F (1991) Bounds on the number of hidden neurons in multilayer perceptrons. IEEE Trans Neural Netw 2(1):47–55
12. Sartori MA, Antsaklis PJ (1991) A simple method to derive bounds on the size and to train multilayer neural networks. IEEE Trans Neural Netw 2(4):467–471
13. Huang G-B, Babri HA (1998) Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions. IEEE Trans Neural Netw 9(1):224–229
14. Gallant A, White H (1992) There exists a neural network that does not make avoidable mistakes. In: White H (ed) Artificial neural networks: approximation and learning theory. Blackwell, Oxford, pp 5–11
15. Hornik K (1991) Approximation capabilities of multilayer feedforward networks. Neural Netw 4:251–257
16. Leshno M, Lin VY, Pinkus A, Schocken S (1993) Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. Neural Netw 6:861–867
17. Park J, Sandberg IW (1991) Universal approximation using radial-basis-function networks. Neural Comput 3:246–257
18. Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Netw 2:359–366
19. Cybenko G (1989) Approximation by superpositions of a sigmoidal function. Math Control Signals Syst 2(4):303–314
20. Funahashi K (1989) On the approximate realization of continuous mappings by neural networks. Neural Netw 2:183–192

21. Stinchcombe M, White H (1992) Universal approximation using feedforward networks with non-sigmoid hidden layer activation functions. In: White H (ed) Artificial neural networks: approximation and learning theory. Blackwell, Oxford, pp 29–40
22. Barron AR (1993) Universal approximation bounds for superpositions of a sigmoidal function. IEEE Trans Inf Theory 39(3):930–945
23. Kwok T-Y, Yeung D-Y (1997) Objective functions for training new hidden units in constructive neural networks. IEEE Trans Neural Netw 8(5):1131–1148
24. Meir R, Maiorov VE (2000) On the optimality of neural-network approximation using incremental algorithms. IEEE Trans Neural Netw 11(2):323–337
25. Romero E (2001) Function approximation with SAOCIF: a general sequential method and a particular algorithm with feed-forward neural networks. Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya. http://www.lsi.upc.es/dept/techreps/html/R01-41.html
26. Huang G-B (2003) Learning capability and storage capacity of two-hidden-layer feedforward networks. IEEE Trans Neural Netw 14(2):274–281
27. Corwin EM, Logar AM, Oldham WJB (1994) An iterative method for training multilayer networks with threshold function. IEEE Trans Neural Netw 5(3):507–508
28. Toms DJ (1990) Training binary node feedforward neural networks by backpropagation of error. Electron Lett 26(21):1745–1746
29. Goodman RM, Zeng Z (1994) A learning algorithm for multi-layer perceptrons with hard-limiting threshold units. In: Proceedings of the 1994 IEEE workshop of neural networks for signal processing, pp 219–228
30. Plagianakos VP, Magoulas GD, Nousis NK, Vrahatis MN (2001) Training multilayer networks with discrete activation functions. In: Proceedings of the IEEE international joint conference on neural networks (IJCNN'2001), Washington, DC, USA
31. Voxman WL, Roy J, Goetschel H (1981) Advanced calculus: an introduction to modern analysis. Marcel Dekker, New York
32. Broomhead DS, Lowe D (1988) Multivariable functional interpolation and adaptive networks. Complex Syst 2:321–355
33. Igelnik B, Pao YH (1995) Stochastic choice of basis functions in adaptive function approximation and the functional-link net. IEEE Trans Neural Netw 6(6):1320–1329
34. Huang G-B, Li M-B, Chen L, Siew C-K (2008) Incremental extreme learning machine with fully complex hidden nodes. Neurocomputing 71:576–583
35. Huang G-B, Siew C-K (2004) Extreme learning machine: RBF network case. In: Proceedings of the eighth international conference on control, automation, robotics and vision (ICARCV 2004), vol 2, Kunming, China, 6–9 Dec 2004, pp 1029–1036
36. Huang G-B, Zhu Q-Y, Mao K-Z, Siew C-K, Saratchandran P, Sundararajan N (2006) Can threshold networks be trained directly?. IEEE Trans Circuits Syst II 53(3):187–191
37. Serre D (2002) Matrices: theory and applications. Springer, New York
38. Rao CR, Mitra SK (1971) Generalized inverse of matrices and its applications. Wiley, New York
39. Huang G-B, Zhou H, Ding X, Zhang R (2010) Extreme learning machine for regression and multi-class classification. IEEE Trans Pattern Anal Mach Intell (submitted)
40. Hoerl AE, Kennard RW (1970) Ridge regression: biased estimation for nonorthogonal problems. Technometrics 12(1):55–67
41. Toh K-A (2008) Deterministic neural classification. Neural Comput 20(6):1565–1595
42. Deng W, Zheng Q, Chen L (2009) Regularized extreme learning machine. In: IEEE symposium on computational intelligence and data mining (CIDM2009), 30 March 2009–2 April 2009, pp 389–395
43. Man Z, Lee K, Wang D, Cao Z, Miao C (2011) A new robust training algorithm for a class of single-hidden layer feedforward neural networks. Neurocomputing (in press)
44. Miche Y, van Heeswijk M, Bas P, Simula O, Lendasse A (2011) TROP-ELM: a double-regularized elm using lars and tikhonov regularization. Neurocomputing (in press)
45. Drucker H, Burges CJ, Kaufman L, Smola A, Vapnik V (1997) Support vector regression machines. In: Mozer M, Jordan J, Petscbe T (eds) Neural information processing systems, vol 9. MIT Press, Cambridge, pp 155–161
46. Hsu C-W, Lin C-J (2002) A comparison of methods for multi-class support vector machines. IEEE Trans Neural Netw 13(2):415–425
47. Lin K-M, Lin C-J (2003) A study on reduced support vector machines. IEEE Trans Neural Netw 14(6):1449–1459
48. Lee Y-J, Mangasarian OL (2001) RSVM: reduced support vector machines. In: Proceedings of the SIAM international conference on data mining, Chicago, USA, 5–7 Apr 2001
49. Suykens JAK, Vandewalle J (1997) Least squares support vector machine classifiers. Neural Process Lett 9(3):293–300
50. Frénay B, Verleysen M (2010) Using SVMs with randomised feature spaces: an extreme learning approach. In: Proceedings of the 18th European symposium on artificial neural networks (ESANN), Bruges, Belgium, 28–30 Apr 2010, pp 315–320
51. Frénay B, Verleysen M (2011) Parameter-insensitive kernel in extreme learning for non-linear support vector regression. Neurocomputing (in press)
52. Li M-B, Huang G-B, Saratchandran P, Sundararajan N (2005) Fully complex extreme learning machine. Neurocomputing 68:306–314
53. Cha I, Kassam SA (1995) Channel equalization using adaptive complex radial basis function networks. IEEE J Sel Areas Commun 13:122–131
54. Jianping D, Sundararajan N, Saratchandran P (2002) Communication channel equalization using complex-valued minimal radial basis function neural networks. IEEE Trans Neural Netw 13:687–696
55. Kim T, Adali T (2003) Approximation by fully complex multilayer perseptrons. Neural Comput 15:1641–1666
56. LeCun Y, Bottou L, Orr GB, Müller K-R (1998) Efficient BackProp. Lect Notes Comput Sci 1524:9–50
57. Platt J (1991) A resource-allocating network for function interpolation. Neural Comput 3:213–225
58. Kadirkamanathan V, Niranjan M (1993) A function estimation approach to sequential learning with neural networks. Neural Comput 5:954–975
59. Yingwei L, Sundararajan N, Saratchandran P (1997) A sequential learning scheme for function approximation using minimal radial basis function (RBF) neural networks. Neural Comput 9:461–478
60. Yingwei L, Sundararajan N, Saratchandran P (1998) Performance evaluation of a sequential minimal radial basis function (RBF) neural network learning algorithm. IEEE Trans Neural Netw 9(2):308–318
61. Salmerón M, Ortega J, Puntonet CG, Prieto A (2001) Improved RAN sequential prediction using orthogonal techniques. Neurocomputing 41:153–172
62. Rojas I, Pomares H, Bernier JL, Ortega J, Pino B, Pelayo FJ, Prieto A (2002) Time series analysis using normalized PG-RBF network with regression weights. Neurocomputing 42:267–285
63. Huang G-B, Saratchandran P, Sundararajan N (2004) An efficient sequential learning algorithm for growing and pruning RBF (GAP-RBF) networks. IEEE Trans Syst Man Cybern Part B 34(6):2284–2292

64. Huang G-B, Saratchandran P, Sundararajan N (2005) A generalized growing and pruning RBF (GGAP-RBF) neural network for function approximation. IEEE Trans Neural Netw 16(1):57–67
65. Liang N-Y, Huang G-B, Saratchandran P, Sundararajan N (2006) A fast and accurate on-line sequential learning algorithm for feedforward networks. IEEE Trans Neural Netw 17(6):1411–1423
66. Chong EKP, Zak SH (2001) An introduction to optimization. Wiley, New York
67. Golub GH, Loan CFV (1996) Matrix computations, 3rd edn. The Johns Hopkins University Press, Baltimore
68. Mackey MC, Glass L (1997) Oscillation and chaos in physiological control systems. Science 197:287–289
69. Vapnik VN (1998) Statistical learning theory. Wiley, New York
70. Smola A, Schölkopf B (1998) A tutorial on support vector regression. NeuroCOLT2 technical report NC2-TR-1998-030
71. Hansen LK, Salamon P (1990) Neural network ensemble. IEEE Trans Pattern Anal Mach Intell 12(10):993–1001
72. Breiman L (1996) Bagging predictor. Mach Learn 24(2):123–140
73. Schapire RE (1990) The strength of weak learnability. Mach Learn 5(2):197–227
74. Freund Y (1995) Boosting a weak algorithm by majority. Inf Comput 121(2):256–285
75. Freund Y, Schapire RE (1997) A decision-theoretic generalization of online learning and an application to boosting. J Comput Syst Sci 55:119–139
76. Sun Z-L, Choi T-M, Au K-F, Yu Y (2008) Sales forecasting using extreme learning machine with applications in fashion retailing. Decis Support Syst 46(1):411–419
77. van Heeswijk M, Miche Y, Lindh-Knuutila T, Hilbers PA, Honkela T, Oja E, Lendasse A (2009) Adaptive ensemble models of extreme learning machines for time series prediction. Lect Notes Comput Sci 5769:305–314
78. van Heeswijk M, Miche Y, Oja E, Lendasse A (2011) Gpu-accelerated and parallelized ELM ensembles for large-scale regression. Neurocomputing (in press)
79. Minku FL, Inoue H, Yao X (2011) Negative correlation in incremental learning. Nat Comp (in press)
80. Sun Y, Yuan Y, Wang G (2011) An OS-ELM based distributed ensemble classification framework in p2p networks. Neurocomputing (in press)
81. Lan Y, Soh YC, Huang G-B (2009) Ensemble of online sequential extreme learning machine. Neurocomputing 72:3391–3395
82. Rong H-J, Ong Y-S, Tan A-H, Zhu Z (2008) A fast pruned-extreme learning machine for classification problem. Neurocomputing 72:359–366
83. Miche Y, Sorjamaa A, Lendasse A (2008) OP-ELM: theory, experiments and a toolbox. Lect Notes Comput Sci 5163:145–154
84. Simila T, Tikka J (2005) Multiresponse sparse regression with application to multidimensional scaling. In: Proceedings in artificial neural networks: formal models and their applications, ICANN 2005, vol 3697, pp 97–102
85. Feng G, Huang G-B, Lin Q, Gay R (2009) Error minimized extreme learning machine with growth of hidden nodes and incremental learning. IEEE Trans Neural Netw 20(8):1352–1357
86. Lan Y, Soh YC, Huang G-B (2010) Random search enhancement of error minimized extreme learning machine. In: European symposium on artificial neural networks (ESANN 2010), Bruges, Belgium, Apr 2010, pp 327–332
87. Li K, Huang G-B, Ge SS (2010) Fast construction of single hidden layer feedforward networks. In: Rozenberg G, Bäck T, Kok JN (eds) Handbook of natural computing. Springer, Berlin, Mar 2010
88. Mao K-Z, Bilings SA (1997) Algorithms for minimal model structure detection in nonlinear dynamic system identification. Int J Control 68(2):311–330
89. Lan Y, Soh YC, Huang G-B (2010) Constructive hidden nodes selection of extreme learning machine for regression. Neurocomputing 73:3191–3199
90. Lan Y, Soh YC, Huang GB (2010) Two-stage extreme learning machine for regression. Neurocomputing 73:3028–3038
91. Liu Q, He Q, Shi Z (2008) Extreme support vector machine classifier. Lect Notes Comput Sci 5012:222–233
92. Huang G-B, Ding X, Zhou H (2010) Optimization method based extreme learning machine for classification. Neurocomputing 74:155–163
93. Fletcher R (1981) Practical methods of optimization. In: Constrained optimization, vol 2. Wiley, New York
94. Handoko SD, Keong KC, Soon OY, Zhang GL, Brusic V (2006) Extreme learning machine for predicting hla-peptide binding. Lect Notes Comput Sci 3973:716–721
95. Sun Z-L, Au K-F, Choi T-M (2008) A neuro-fuzzy inference system through integration of fuzzy logic and extreme learning machines. IEEE Trans Syst Man Cybern Part B Cybern 37(5):1321–1331
96. Tang X, Han M (2009) Partial lanczos extreme learning machine for single-output regression problems. Neurocomputing 72(13-15):3066–3076
97. Miche Y, Sorjamaa A, Bas P, Simula O, Jutten C, Lendasse A (2010) OP-ELM: optimally pruned extreme learning machine. IEEE Trans Neural Netw 21(1):158–162
98. Yeu C-WT, Lim M-H, Huang G-B, Agarwal A, Ong Y-S (2006) A new machine learning paradigm for terrain reconstruction. IEEE Geosci Remote Sens Lett 3(3):382–386
99. Soria-Olivas E, Gomez-Sanchis J, Martin JD, Vila-Frances J, Martinez M, Magdalena JR, Serrano AJ (2011) BELM: Bayesian extreme learning machine. IEEE Trans Neural Netw 22(3):505–509
100. Xu Y, Dong ZY, Meng K, Zhang R, Wong KP (2011) Real-time transient stability assessment model using extreme learning machine. IET Gener Transm Distrib 5(3):314–322
101. Barea R, Boquete L, Rodriguez-Ascariz JM, Ortega S, Lopez E (2011) Sensory system for implementing a human-computer interface based on electrooculography. Sensors 11(1):310–328
102. Chang N-B, Han M, Yao W, Chen L-C, Xu S (2011) Change detection of land use and land cover in an urban region with SPOT-5 images and partial lanczos extreme learning machine. J Appl Remote Sens 4
103. Saraswathi S, Sundaram S, Sundararajan N, Zimmermann M, Nilsen-Hamilton M (2011) ICGA-PSO-ELM approach for accurate multiclass cancer classification resulting in reduced gene sets in which genes encoding secreted proteins are highly represented. IEEE ACM Trans Comput Biol Bioinforma 6(2):452–463
104. Li F-C, Wang P-K, Wang G-E (2009) Comparison of the primitive classifiers with extreme learning machine in credit scoring. In: 2009 IEEE international conference on industrial engineering and engineering management, pp 685–688
105. Choi K, Toh K-A, Byun H (2011) Realtime training on mobile devices for face recognition applications. Pattern Recogn 44(2):386–400
106. Chen FL, Ou TY (2011) Sales forecasting system based on gray extreme learning machine with Taguchi method in retail industry. Expert Syst Appl 38(3):1336–1345
107. Ye Y, Squartim S, Piazza F (2010) Incremental-based extreme learning machine algorithms for time-variant neural networks. Lect Notes Comput Sci 6215:9–16
108. Suresh S, Saraswathi S, Sundararajan N (2010) Performance enhancement of extreme learning machine for multi-category

sparse data classification problems. Eng Appl Artif Intell 23(7):1149–1157

109. Li G, Liu M, Dong M (2010) A new online learning algorithm for structure-adjustable extreme learning machine. Comput Math Appl 60(3):377–389
110. Liu Y, Xu X, Wang C (2009) Simple ensemble of extreme learning machine. In: Proceedings of the 2009 2nd international congress on image and signal processing, pp 2177–2181
111. Deng W, Chen L (2010) Color image watermarking using regularized extreme learning machine. Neural Network World 20(3):317–330
112. Mohammed AA, Wu QMJ, Sid-Ahmed MA (2010) Application of wave atoms decomposition and extreme learning machine for fingerprint classification. Lect Notes Comput Sci 6112:246–256
113. Minhas R, Baradarani A, Seifzadeh S, Wu QMJ (2010) Human action recognition using extreme learning machine based on visual vocabularies. Neurocomputing 73:1906–1917
114. Malathi V, Marimuthu NS, Baskar S (2010) Intelligent approaches using support vector machine and extreme learning machine for transmission line protection. Neurocomputing 73:2160–2167
115. Tang X-L, Han M (2010) Ternary reversible extreme learning machines: the incremental tri-training method for semi-supervised classification. Knowl Inf Syst 22(3):345–372
116. Nizar AH, Dong ZY, Wang Y (2008) Power utility nontechnical loss analysis with extreme learning machine method. IEEE Trans Power Syst 23(3):946–955
117. Cho JS, White H (2011) Testing correct model specification using extreme learning machines. Neurocomputing (in press)
118. Wang Y, Cao F, Yuan Y (2011) A study on effectiveness of extreme learning machine. Neurocomputing (in press)
119. Deng J, Li K, Irwin GW (2011) Fast automatic two-stage nonlinear model identification based on the extreme learning machine. Neurocomputing (in press)