IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON ADVANCED DATA ANALYTICS FOR LARGE-SCALE COMPLEX DATA ENVIRONMENTS

Received August 21, 2017, accepted September 18, 2017, date of publication October 4, 2017, date of current version October 25, 2017.

Digital Object Identifier 10.1109/ACCESS.2017.2758645

Incremental and Decremental Extreme Learning Machine Based on Generalized Inverse

BO JIN[1], ZHONGLIANG JING[2], AND HAITAO ZHAO[3]

[1]School of Computer Science and Software Engineering, East China Normal University, Shanghai 200062, China
[2]School of Aeronautics and Astronautics, Shanghai Jiaotong University, Shanghai 200240, China
[3]School of Information Science and Engineering, East China University of Science and Technology, Shanghai 200237, China

Corresponding author: Bo Jin (bjin@sei.ecnu.edu.cn)

This work was supported in part by the Key Program of Shanghai Science and Technology Commission under Grant 15JC1401700, in part by the NSFC-Zhejiang Joint Fund for the Integration of Industrialization and Information under Grant U1609220 and Grant U1509219, and in part by the Municipality Projects of Shanghai Science and Technology Commission under Grant 15511104700 and Grant 16DZ1100600.

ABSTRACT In online sequential applications, a machine learning model needs to have a self-updating ability to handle the situation, which the training set is changing. Conventional incremental extreme learning machine (ELM) and online sequential ELM are usually achieved in two approaches: directly updating the output weight and recursively computing the left pseudo inverse of the hidden layer output matrix. In this paper, we develop a novel solution for incremental and decremental ELM (DELM), via recursively updating and downdating the generalized inverse of the hidden layer output matrix. By preserving the global optimality and best generalization performance, our approach implements node incremental ELM (N-IELM) and sample incremental ELM (S-IELM) in a universal form, and overcomes the problem of self-starting and numerical instability in the conventional online sequential ELM. We also propose sample DELM (S-DELM), which is the first decremental version of ELM. The experiments on regression and classification problems with real-world data sets demonstrate the feasibility and effectiveness of the proposed algorithms with encouraging performances.

INDEX TERMS Extreme learning machine, online sequential ELM, incremental ELM, decremental ELM, generalized inverse.

I. INTRODUCTION

Extreme learning machine (ELM) [1] has been a popular topic in the fields of machine learning [2] and neural networks [3] in this decade. As the estimator in regression problems or the classifier in classification problems, ELM has been applied in many applications, such as face recognition [4] and microarray gene expression [5].

From the point of view of network structure, ELM belongs to the single-hidden layer feedforward networks (SLFNs), which consists of one input layer responding the external stimuli, one solo hidden layer, and one output layer transmitting the network output to external environments. For a conventional SLFNs, parameters of neurons are tuned by iterative methods, such as back-propagation algorithm [6]. This results in several problems, such as local minima [3], overfitting, time-consuming. Huang *et al.* [7] propose a simple and efficient solution for SLFNs, i.e. ELM, which doesn't need any iterative techniques. To determine parameters of networks, ELM generalizes SLFNs in two aspects. First, hidden nodes and their parameters in ELM are randomly generalized, and the activation function can be chosen as any bounded non-constant piecewise continuous function [8]. Secondly, output weights of output neurons are calculated via the least square problem between the hidden layer output matrix H and labels of the training set T, which grants ELM an extremely fast speed, the global optimality, and the best generalization performance [9].

Although the batch-mode ELM has been demonstrated to be an effective method, it may not satisfy the demand in many sequential applications, where training data points come one by one or chunk by chunk. Online and sequential learning has drawn a great deal of attention in the machine learning society [10]–[13]. In this paper, we focus on handling three types of ELM problems with a unified solution, i.e. adding samples, adding hidden nodes, and deleting invalid samples.

To solve the problem of retraining the network in the batch-mode ELM when new samples are available, Liang *et al.* [14] propose an online sequential algorithm,

known as online sequential extreme learning machine (OS-ELM). In OS-ELM, the generalized inverse $H^{\dagger}$ of the hidden layer output matrix is replaced by its left pseudo inverse, and its sequential computation is achieved by recursively updating $(H^TH)^{-1}$ via Woodbury formula [15]. The invertibility of H^TH is guaranteed by the interpolation theorem [16], i.e. for a SLFNs, when the number N of training samples is greater than the number L of hidden nodes , the hidden layer output matrix H is full column rank with probability one. OS-ELM, inheriting the advantage of ELM, can learn sequential training data in both one-by-one and chunk-by-chunk modes. However, OS-ELM has two drawbacks: the algorithm needs at least L samples to initialize, which might be embarrassing at the beginning of learning, when the number L of hidden nodes is very large; the variable for iteration is $(H^TH)^{-1}$, which is unreliable from the numerical stability view of point.

Another major issue for ELM is that the learning ability of an ELM with a fixed number of hidden nodes will decrease after adding a large mount of new training instances. For this reason, there is a demand for the ability of adding new hidden nodes for ELM in online applications. A theoretic analysis shows that the sequence of network functions $\{f_n\}$ of a SLFNs with incremental randomly generated additive or RBF hidden nodes can universally converge to any continuous objective function f over any compact subset of Euclidean space [8]. Based on this theory, incremental extreme learning machine (I-ELM) and its improved version, i.e. enhanced incremental extreme learning machine (EI-ELM) [17], are proposed. They directly compute the output weight of the new hidden node by minimizing the approximation error. With the same scheme, EI-ELM takes a fact into consideration that the learning abilities of random hidden nodes are different, hence k hidden node candidates are randomly generated, and the one with the strongest ability, i.e. the minimum approximation error, is chosen as the new hidden node. Although the competition mechanism can select better hidden nodes, the scheme of generating k candidate hidden nodes will increase the computation complexity, especially when the input dimension is huge. A common disadvantage of I-ELM and EI-ELM is that output weights of previous hidden nodes are not updated when adding a new hidden node, so the resulting output vector is no longer the optimal solution of the least square problem in ELM, which results in a non-optimal performance and a non-optimal generalization ability. Therefore, a further improvement about I-ELM is made in convex incremental extreme learning machine (CI-ELM) [18]. Its output weight of the new hidden node is also directly computed via minimizing the approximation error, meanwhile output weights of former hidden nodes are updated according to the Barron's convex optimization concept [19]. Although CI-ELM can achieve a better performance than EI-ELM ($k = 1$), the convex optimization scheme can't gain the optimal solution of the least square problem either. Feng *et al.* [20] propose the error minimized extreme learning machine (EM-ELM), which can minimize the estimation error by adding hidden nodes in a one-by-one or chunk-by-chunk mode. EM-ELM also recursively computes the left pseudo inverse. Because the hidden layer output matrix H is column full rank with probability one [21], $(H^TH)^{-1}$ is incrementally updated by computing the inverse of a 2×2 block matrix [22]. It is proved that EM-ELM can obviously reduce the computation complexity and convergence. Lan *et al.* [23] make a further study about EM-ELM, where a random searching method, similar as EI-ELM, is introduced to reduce the residual error. Xu *et al.* [24] propose the incremental regularized extreme learning machine (IR-ELM) to update ELM with LASSO and Tikhonov regularization.

The major computations of ELM are the hidden layer mapping and computing the least square problem, where the former depends on the input dimension and is usually the same for different ELM methods, unless the random searching scheme of multiple hidden nodes is used. Hence, the scale of H is the major fact when considering the computation complexity of ELM. As numbers of training samples and hidden nodes grow, the hidden layer output matrix becomes larger, therefore the computational efficiency of ELM declines. Besides, some expired, invalid or mock data may be adulterated in the training set in practical applications. For example, some sensors expire in environment monitoring; some old games are dropped in mobile game recommendation [25]. Keeping their invalid samples in the training set will sacrifice the approximation accuracy on the valid class and also decrease the efficiency of the ELM in both time and memory. Moreover, the leave-one-out estimation is usually used to evaluate the generalization performance. Decremental unlearning of each training sample will make the whole procedure more convenient and efficient. Thus, there is a demand for developing a decremental version of ELM, which has not been taken into account in the literature to our best knowledge. In other domains of machine learning, there have been only a few works about decremental learning. Hall *et al.* [26] propose an eigenspace splitting algorithm, which can subtract the eigenspace of a spectacular subset from the eigenspace of the whole training set. Jin *et al.* [27] propose an evd-dualdating based online subspace learning algorithm, i.e. EVDD-IDPCA, which can optimally update the subspace model after simultaneously adding and deleting samples. Pang *et al.* [28] propose a LDA splitting method, which can split a learning LDA model into two independent sub-models. Cauwenberghs and Poggio [29] propose an incremental decremental support vector machine method for adding and/or deleting a single sample, and Karasuyama and Takeuchi [30] expands it to the case of multiple instances. Tsai *et al.* [31] discuss the incremental and decremental algorithm for quickly updating linear logistic regression and linear SVM model.

The major contribution of this paper is proposing a universal solution based on generalized inverse for increment and decremental ELM. As shown in Figure 1, via the hidden layer mapping, adding or deleting samples is equivalent to

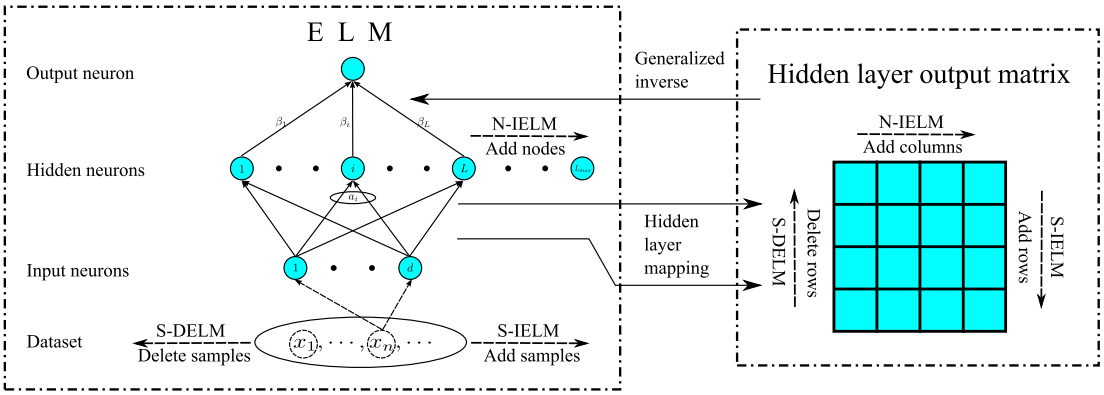

FIGURE 1. Framework of incremental and decremental ELMs based generalized inverse.

adding or deleting rows in the hidden layer output matrix H; adding new hidden nodes is equivalent to adding columns. Different from previous approaches, which avoid computing the generalized inverse $H^\dagger$ by approximately estimating output weights β, or recursively updating the inverse of H^TH, we treat incremental or decremental ELM as a problem of updating or downdating the generalize inverse. The major benefit is that our solution can obtain exactly the same model as the batch-mode ELM, which preserves the optimality of ELM, i.e. the minimum approximation error, the minimum norm, and best generalization performance. In details, based on generalized inverse, we propose three algorithms, node incremental ELM (N-IELM), sample incremental ELM (S-IELM), sample decremental ELM (S-DELM). N-IELM can add new hidden nodes one by one. S-IELM can add new training samples one by one or chunk by chunk, overcomes the numerical unstable problem of OS-ELM, and have the ability of self-starting. S-DELM can delete arbitrary sample from a trained ELM, which is the first decremental version of ELM. Plenty of experiments, including regression and classification, are presented to evaluate the performance of the proposed algorithms on various public dataset, compared with other incremental and online sequential methods. Experimental results demonstrate that: N-IELM achieves better performance on both training and testing sets than EI-ELM, CIELM, and EM-ELM, and also have a comparable efficiency; S-IELM runs a little slower than OS-ELM, but it is much more numerically stable than OS-ELM and is not restricted to the number of training samples in the initial phase; the efficiency and accuracy of ELM are obviously enhanced after deleting expired classes with S-DELM.

The remaining of this paper is organized as follows. Section II briefly reviews Moore-Penrose inverse and ELM. Section III-A introduces the universal formula of updating the generalized inverse of the hidden layer output matrix. Then, N-IELM and S-IELM are proposed based on this form in Section III-B, III-C, respectively. The derivation of S-DELM is presented in Section IV. The performance evaluation of the proposed algorithms is shown in Section V. Finally, a conclusion is given in section VI to finish this paper.

II. PRELIMINARIES

A. MOORE-PENROSE INVERSE

Penrose [32] proves that for every finite matrix A (square or rectangle) of real or complex elements, there exists a unique matrix X satisfying the following four equation (so-called Penrose equations):

$$\begin{aligned} AXA &= A, \\ XAX &= X, \\ (AX)^T &= AX, \\ (XA)^T &= XA, \end{aligned}$$

where A^T denotes the transpose of A. The unique generalized inverse X is commonly known as the Moore-Penrose inverse, and is denoted by $A^\dagger$.

Moore-Penrose inverse has a lot of important properties, which can be found with a systematic introduction in [33]. Here, we list a few of them, which are used in this paper.

1) For arbitrary column vector $a \neq 0$, $a^\dagger = \frac{a^T}{a^T a}$,
2) $(A^T)^\dagger = (A^\dagger)^T$,
3) $A^\dagger = A^\dagger(A^\dagger)^T A^T = A^T(A^\dagger)^T A^\dagger$,
4) $A^\dagger = (A^T A)^\dagger A^T = A^T(AA^T)^\dagger$,
5) $A^\dagger A, AA^\dagger, I_n - A^\dagger A, I_m - AA^\dagger$ are symmetric,
6) $A^\dagger B^\dagger = 0 \Longleftrightarrow BA = 0$,
7) $AA^\dagger = P_A$, where P_A denotes the projection matrix of $\Re(A)$, and $\Re(A)$ is the column space of A,
8) $A^\dagger A = P_{A^T}$.

B. EXTREME LEARNING MACHINE

The essence of extreme learning machine (ELM) [1] is a least-square based generalized single-hidden layer feedforward networks (SLFNs), whose hidden nodes need not to be neuron alike [8]. The output function of ELM can be represented as:

$$f_L(x) = \mathbf{h}(x)\beta, \tag{1}$$

where $\mathbf{h}(x) = \left[G(a_1, b_1, x) \cdots G(a_L, b_L, x)\right]$ is the hidden layer feature mapping from the n-dimension input space to the L-dimension hidden-layer feature space, i.e. ELM feature space; $G(a_i, b_i, x)$ is the activation function;

$\beta = [\beta_1, \beta_2, \cdots, \beta_L]^T$ is the output weight vector between L hidden nodes and output nodes.

From the point of view of theory, if an ELM satisfies the universal approximation capability theorem, the activation function $G(\cdot)$ can be any nonlinear piecewise continuous function [8]. Usually, the hidden layer mapping function $G(a_i, b_i, x)$ can be chosen as the following types [14]:

- For additive hidden nodes with the activation function $g(x) : \mathbb{R} \to \mathbb{R}$, where $g(x)$ can be sigmoid or threshold, $G(a_i, b_i, x)$ is given by

$$G(a_i, b_i, x_i) = g(a_i \cdot x + b_i), a_i \in \mathbb{R}^n, b_i \in \mathbb{R}; \quad (2)$$

- For RBF hidden nodes with the activation function $g(x) : \mathbb{R} \to \mathbb{R}$, where $g(x)$ can be Gaussian or multiquadrics, $G(a_i, b_i, x_i)$ is given by

$$G(a_i, b_i, x_i) = g(\|x - a_i\|, b_i), a_i \in \mathbb{R}^n, b_i \in \mathbb{R}^+. \quad (3)$$

Given a data set with N distinct samples $\aleph = \{(x_i, t_i)\}_{i=1}^N$, where $x_i \in \mathbb{R}^n$, $t_i \in \mathbb{R}^m$ in the regression problem, and $t_i \in \mathbb{N}$ in the classification problem, if the output of the SLFN with L hidden nodes, whose parameters are randomly generated $\{(a_i, b_i)\}_{i=1}^L$, can approximate these training samples with zero estimation error, then there exists β, satisfying

$$H\beta = T, \quad (4)$$

where

$$H = \begin{bmatrix} G(a_1, b_1, x_1) & \cdots & G(a_L, b_L, x_1) \\ \vdots & \ddots & \vdots \\ G(a_1, b_1, x_N) & \cdots & G(a_L, b_L, x_N) \end{bmatrix} \in \mathbb{R}^{N \times L},$$

$$T = \begin{bmatrix} t_1^T \\ \vdots \\ t_N^T \end{bmatrix} \in \mathbb{R}^{N \times m}.$$

H is called the hidden output matrix [34]. The i-th column of H is the i-th hidden node's output vector with respect to inputs $\{x_1, x_2, \cdots, x_N\}$, and the j-th row of H is the output vector of the hidden layer with respect to input x_j.

The solution of ELM can be viewed as an optimization problem under the constraint of minimum the sum of squares of the residuals and minimum norm:

$$\min\{\|H\beta - T\|^2, \quad \|\beta\|\}. \quad (5)$$

A simple solution of (5) can be analytically described via Moore-Penrose inverse [21]:

$$\beta = H^\dagger T. \quad (6)$$

To summarize, there are three core steps in the typical implementation of ELM: randomly generate parameters of hidden nodes, compute the hidden layer output matrix H, compute the weight vector of the output layer $\beta = H^\dagger T$.

As an important conclusion in the theory of ELM, interpolation theorem, presented in Lemma 1, demonstrates that for a SLFNs with N hidden nodes can learn N distinct training samples with zero error. Lemma 1 also indicates that columns of the hidden layer output matrix H satisfy column full rank when the number N of training samples is greater than the number L of hidden nodes, with probability one.

Lemma 1 (Huang et al. [21]): Given a SLFNs with N hidden nodes and any activation function $g : \mathbb{R} \to \mathbb{R}$, which is infinitely differentiable in any interval, for N arbitrary distinct samples (x_i, t_i), where $x_i \in \mathbb{R}^n$ and $t_i \in \mathbb{R}^m$, for any a_i and b_i randomly chosen from any intervals of $\mathbb{R}^n$ and $\mathbb{R}$, respectively, according to any continuous probability distribution, then with probability one, the hidden layer output matrix H is invertible, and $\|H\beta - T\| = 0$.

III. INCREMENTAL EXTREME LEARNING MACHINE

As introduced in Section I, existing incremental and online sequential ELMs are usually achieved in one of the following two ways: progressively updating the output weight vector β, and updating the inverse of $H^T H$. In this section, we present our novel incremental and online sequential ELM via recursively calculating the generalized inverse matrix $H^\dagger$. From the transpose point of view, incremental and online sequential ELM are symmetrical, i.e. adding columns and rows to the hidden layer matrix. So in our methods, we don't distinguish incremental ELM from online sequential ELM any more, and we call them node and sample incremental ELM.

We first introduce a universal formula to update the generalized inverse of the hidden layer output matrix H. Then, based on this form the node incremental ELM (N-IELM) is proposed in Section III-B, and the sample incremental ELM (S-IELM) is proposed in Section III-C, respectively.

A. UNIVERSAL FORMULA FOR NODE AND SAMPLE INCREMENTAL ELM

In our approach, the core of incremental ELM is to update the generalized inverse $H^\dagger$ of the hidden layer output matrix when new hiddens or samples are added into ELM, i.e. to obtain the renewal of $H^\dagger$ without recomputing when adding new columns or rows into H . The recursive formula of the generalized inverse is a complicated problem. Although there already has a universal form in mathematics [33], [35], its computation complexity is very high, and can't be applied to incremental ELM directly. Here, we first consider the basic form of updating $H^\dagger$, then based on this form and the particularity of H, we present our incremental ELM algorithms for adding hidden nodes and samples.

Considering the hidden layer output matrix, the operation of adding samples can be achieved by the transpose of adding samples. Therefore, we only need deduce the universal formula of incremental ELM in the column form, which is presented in Lemma 2, and its row form can be simply obtained by transpose.

Lemma 2: Given a hidden layer output matrix $H_1 \in \mathbb{R}^{N \times L}$, its generalized inverse $H_1^\dagger$, and new output columns $\delta_L H \in \mathbb{R}^{N \times \delta L}$ of new hidden nodes, the new hidden layer output matrix is $H_2 = \begin{bmatrix} H_1 & \delta_L H \end{bmatrix} \in \mathbb{R}^{N \times (L + \delta L)}$. Then,

the generalized inverse $H_2^\dagger$ satisfies the following form:

$$H_2^\dagger = \begin{bmatrix} H_1^\dagger - H_1^\dagger \delta_L H B \\ B \end{bmatrix},$$

where B is a matrix relative to H_1 and $\delta_L H$.

Proof: Partition rows of $H_2^\dagger$ into two parts, with respect to columns of H_1 and $\delta_L H$,

$$H_2^\dagger = \begin{bmatrix} A \\ B \end{bmatrix}, \tag{7}$$

where $A \in \mathbb{R}^{L\times N}$ and $B \in \mathbb{R}^{\delta L\times N}$. Then, it is clearly established that

$$H_2\, H_2^\dagger = H_1\, A + \delta_L H B, \tag{8}$$

$$H_2^\dagger H_2 = \begin{bmatrix} AH_1 & A\delta_L H \\ BH_1 & B\delta_L H \end{bmatrix}. \tag{9}$$

According to the definition of Penrose-Moore inverse, $H_2\, H_2^\dagger H_2 = H_2$, it follows

$$H_2 H_2^\dagger H_1 = H_1, \tag{10}$$

Transposing (10) and left multiplying it by $(H_1^T H_1)^\dagger$, according to the property (4), we have

$$H_1^\dagger H_2 H_2^\dagger = (H_1^T H_1)^\dagger H_1^T H_2 H_2^\dagger = (H_1^T H_1)^\dagger H_1^T = H_1^\dagger. \tag{11}$$

According to the property (3), $H_2^\dagger = H_2^T (H_2^\dagger)^T H_2^\dagger = (H_2^\dagger H_2)^T H_2^\dagger$, substituting $H_2^\dagger H_2$ with (9) ,we have

$$A = H_1^T A^T A + H_1^T B^T B. \tag{12}$$

Since $H_1^\dagger H_1$ is the projection matrix of $\Re(H_1^T)$, $H_1^\dagger H_1\, H_1^T = H_1^T$. Left multiplying (12) with $H_1^\dagger H_1$, it holds

$$H_1^\dagger H_1 A = H_1^T A^T A + H_1^T B^T B = A. \tag{13}$$

Left multiplying (8) with $H_1^\dagger$ and using (13), we have

$$H_1^\dagger H_2 H_2^\dagger = H_1^\dagger H_1 A + H_1^\dagger \delta_L H B, \tag{14}$$

Finally, A can be written as

$$A = H_1^\dagger - H_1^\dagger \delta_L H B, \tag{15}$$

and $H_2^\dagger$ satisfies the proposed form. □

According to the property (2), i.e. the convertibility between transpose and generalized inverse, a symmetric form of updating the generalized inverse of the hidden layer output matrix when adding samples can be obtained via transposing the derivation above, described in Lemma 3.

Lemma 3: Given a hidden layer output matrix $H_1 \in \mathbb{R}^{N\times L}$, its generalized inverse $H_1^\dagger$, and new output rows $\delta_N H \in \mathbb{R}^{\delta N\times L}$ of new hidden samples, the new hidden layer output matrix is $H_2 = \begin{bmatrix} H_1 \\ \delta_N H \end{bmatrix} \in \mathbb{R}^{(N+\delta N)\times L}$. Then, the generalized inverse $H_2^\dagger$ satisfies the following form:

$$H_2^\dagger = \begin{bmatrix} H_1^\dagger - B\delta_N H H_1^\dagger & B \end{bmatrix}, \tag{16}$$

where B is a matrix relative to H_1 and $\delta_N H$.

B. NODE INCREMENTAL ELM

We consider how to calculate B in Lemma 2 in the case of adding hidden nodes one by one, and propose the node incremental ELM (N-IELM) algorithm. We assume the total number of hidden nodes is not greater than the number N of training samples, since Lemma 1 indicates that the maximum number L_{max} of needed hidden nodes to approximate the training data with zero error is equal to N.

Given a training set $\aleph = \{(x_i, t_i)\}_{i=1}^N$, a trained ELM with L hidden nodes $\{(a_i, b_i)\}_{i=1}^L$, the hidden layer output matrix is $H_1 \in \mathbb{R}^{N\times L}$, its Moore-Penrose inverse is $H_1^\dagger \in \mathbb{R}^{L\times N}$, and the output weight vector is $\beta_1 = H_1^\dagger T$. To add an additional hidden node to the existing ELM, we first randomly generate a pair of parameters for the new hidden node (a_{i+1}, b_{i+1}), then compute the new hidden layer output matrix $H_2 = \begin{bmatrix} H_1 & \delta_L H \end{bmatrix} \in \mathbb{R}^{N\times (L+1)}$, where

$$\delta_L H = \begin{bmatrix} G(a_{L_1+1}, b_{L+1}, x_1) & \cdots & G(a_{L_1+1}, b_{L+1}, x_N) \end{bmatrix}^T. \tag{17}$$

is the hidden layer output vector of the new hidden node.

For convenience sake, we denote two vectors: $d = H_1^\dagger \delta_L H$, and $c = \delta_L H - H_1\, d$. According to Lemma 2, (8) is equivalent to

$$H_2 H_2^\dagger = H_1 H_1^\dagger + cB. \tag{18}$$

In (18), because $H_1\, H_1^\dagger$, $H_2\, H_2^\dagger$ are both symmetric, so cB is also symmetric. Thus, it holds

$$B = \lambda c^T, \tag{19}$$

where λ is a real number.

By definition, c is the residual of $\delta_L H$ on the column space $\Re(H_1)$, so we divide the problem into two cases according whether c is zero or not. If $\delta_L H$ is not in $\Re(H_1)$, $c \neq 0$. According to the definition of c and $H_2\, H_2^\dagger H_2 = H_2$, we have

$$H_2 = \begin{bmatrix} H_1 + cBH_1 & \delta_L H + B\delta_L H c - c \end{bmatrix}. \tag{20}$$

Thus, $B\delta_L H c - c = 0$. Because $c \neq 0$, it leads to

$$B\delta_L H = 1. \tag{21}$$

Note that c can be written as

$$c = (I - H_1 H_1^\dagger)\delta_L H = P\delta_L H. \tag{22}$$

where $P = I - H_1\, H_1^\dagger$ is idempotent and symmetric. Repeatedly using (19) and (22), (21) leads to

$$\begin{aligned} 1 = B\delta_L H &= \lambda c^T \delta_L H = \lambda \delta_L H^T P \delta_L H \\ &= \lambda \delta_L H^T P P \delta_L H = \lambda c^T c. \end{aligned} \tag{23}$$

So, we have

$$B = \lambda c^T = \frac{1}{\lambda}(c^T c)^\dagger \lambda c^T = (c^T c)^\dagger c^T = c^\dagger. \tag{24}$$

If $\delta_L H$ is in $\Re(H_1)$, $c = 0$. In (8), because $H_2^\dagger H_2$ is symmetric, we have

$$BH_1 = (A\delta_L H)^T = (1 - B^T \delta_L H) d^T = (1 - \rho) d^T, \tag{25}$$

where $\rho = B^T \delta_L H$ is a scalar. Since $c = 0$ and $H_2 H_2^\dagger$ is the projection matrix of $R(H_2^{\dagger T})$, we have

$$B = BH_2H_2^\dagger = BH_1H_1^\dagger = (1-\rho)d^T H_1^\dagger. \tag{26}$$

Then, ρ can be written as

$$\rho = (1-\rho)d^T H_1^\dagger \delta_L H = (1-\rho)d^T d. \tag{27}$$

From (27), it can be solved that

$$1-\rho = \frac{1}{1+d^T d}. \tag{28}$$

Finally, we can compute B by

$$B = \frac{d^T H_1^\dagger}{1+d^T d}. \tag{29}$$

Thus, we eventually obtain the approach to determine B according as $\delta_L H$ is in or not in $\Re(H_1)$. After obtaining B, the new output weights β_2 with the new hidden node can be computed by Lemma 2,

$$\beta_2 = H_2^\dagger T = \begin{bmatrix} \beta_1 - dBT \\ BT \end{bmatrix}. \tag{30}$$

Then, we propose a fast incremental learning mechanism for ELM with growing hidden nodes, called N-IELM, which can preserve the optimality and is presented in Algorithm 1.

C. SAMPLE INCREMENTAL ELM

In this section, we discuss how to update the Moore-Penrose inverse $H^\dagger$ of the hidden layer output matrix with growing samples based on Lemme 3, and propose a novel online sequential ELM algorithm, called sample incremental ELM (S-IELM). Assuming that the number of hidden nodes is fixed, according to Lemme 1, we divide the incremental sample learning problem of ELM into two subproblems: how to update a trained ELM when H is full column rank, i.e. when $N \geq L$, and how to initialize the learning procedure, i.e. when $N < L$. First, We solve the first subproblem via determining B in the universal recursive form of the generalized inverse $H^\dagger$ in Lemma 3.

Given a training set $\aleph = \{(x_i, t_i)\}_{i=1}^N$, where $N \geq L$, an trained ELM with L hidden nodes $\{(a_i, b_i)\}_{i=1}^L$, the hidden layer output matrix is $H_1 \in \mathbb{R}^{N\times L}$, its Moore-Penrose inverse is $H_1^\dagger \in \mathbb{R}^{L\times N}$, and the output weight vector is $\beta_1 = H_1^\dagger T$. Now, new samples $\delta\aleph = \{(x_i, t_i)\}_{i=N+1}^{N+\delta N}$ appear, the new hidden layer output matrix becomes $H_2 = \begin{bmatrix} H_1 \\ \delta_N H \end{bmatrix} \in \mathbb{R}^{(N+\delta N)\times L}$, where

$$\delta_N H = \begin{bmatrix} G(a_1, b_1, x_{N+1}) & \cdots & G(a_L, b_L, x_{N+1}) \\ \vdots & \ddots & \vdots \\ G(a_1, b_1, x_{N+\delta N}) & \cdots & G(a_L, b_L, x_{N+\delta N}) \end{bmatrix} \in \mathbb{R}^{\delta N\times L}, \tag{31}$$

is the hidden layer output of new samples.

According to Lemma 1, if the activation function $g(\cdot)$ is infinitely differentiable and the number of samples is greater than or equal to the number of hidden nodes, columns in hidden layer output matrices H_1 and H_2 are both full column rank with probability one, i.e. $rank(H_1) = rank(H_2) = L$. In other words, when $N \geq L$, adding samples will not change the row space of the hidden layer output matrix with probability one. Thus, the row spaces of H_1 and $\delta_N H$ satisfy

$$\Re(\delta_N H^T) \subset \Re(H_1^T). \tag{32}$$

Moreover, according to property (8), $H_1^\dagger H_1$ is the projection matrix of $\Re(H_1^T)$, so we have $\delta_N H H_1^\dagger H_1 = \delta_N H$. Using the universal updating formula in Lemme 3, it holds

$$H_2^\dagger H_2 = H_1^\dagger H_1 + B\delta_N H(I - H_1^\dagger H_1) = H_1^\dagger H_1. \tag{33}$$

According to the property (5), $H_2 H_2^\dagger$ is symmetric. Thus, it can be observed that

$$\begin{aligned}(H_1 B)^T &= \delta_N H A \\ &= (I - \delta_N H B)\delta_N H H_1^\dagger \\ &= \delta_N H H_1^\dagger - \delta_N H H_1^\dagger H_1 B \delta_N H H_1^\dagger. \end{aligned} \tag{34}$$

Algorithm 1 Node Incremental Extreme Learning Machine (N-IELM)

Input: Given a set of training data $\aleph = \{(x_i, t_i)\}_{i=1}^N$, the activation function g, the maximum of hidden nodes $L_{max} \leq N$.
Output: The parameters of L_{max} hidden nodes $\{(a_L, b_L)\}_{l=L}^{L_{max}}$, the hidden output matrix H, its generalized inverse $H^\dagger$, the output weight vector β.
For $L = 1{:}L_{max}$

1) Randomly generate a pair of parameter for the L-th hidden node (a_L, b_L), and compute its hidden layer output vector

$$\delta_L H = \begin{bmatrix} G(a_L, b_L, x_1) & \cdots & G(a_L, b_L, x_N) \end{bmatrix}^T.$$

2) **If** L = 1, compute ELM,

$$H = \delta_L H, \quad H^\dagger = \delta_L H^\dagger, \quad \beta = H^\dagger T.$$

Else

a) Compute $d = H^\dagger \delta_L H$, $c = \delta_L H - Hd$.
b) Compute B,

$$B = \begin{cases} c^\dagger & \text{if } c \neq 0 \\ \frac{d^T H^\dagger}{1+d^T d} & \text{if } c = 0 \end{cases}.$$

c) Update ELM,

$$H^\dagger \leftarrow \begin{bmatrix} H^\dagger - dB \\ B \end{bmatrix},$$
$$\beta \leftarrow \begin{bmatrix} \beta - dBT \\ BT \end{bmatrix},$$
$$H \leftarrow \begin{bmatrix} H & \delta_L H \end{bmatrix}.$$

End if
End For

Since $(H_1\ B\delta_N HH_1^\dagger)^T = (\delta_N HH_1^\dagger)^T(I - \delta_N HB)(\delta_N HH_1^\dagger)$ is symmetric, transposing (34), we have

$$H_1 B = (\delta_N HH_1^\dagger)^T - H_1 B\delta_N HH_1^\dagger(\delta_N HH_1^\dagger)^T, \quad (35)$$

which leads to

$$H_1 B[I + \delta_N HH_1^\dagger(\delta_N HH_1^\dagger)^T] = (\delta_N HH_1^\dagger)^T. \quad (36)$$

Because $H_2^\dagger H_2$ is the projection matrix of $\Re(H_2^\dagger)$, $H_1^\dagger H_1\ B = H_2^\dagger H_2\ B = B$. Left multiplying (36) with $H_1^\dagger$, we have

$$B[I + \delta_N HH_1^\dagger(\delta_N HH_1^\dagger)^T] = H_1^\dagger(\delta_N HH_1^\dagger)^T. \quad (37)$$

Therefore, B can be solved from (37)

$$B = H_1^\dagger(\delta_N HH_1^\dagger)^T[I + \delta_N HH_1^\dagger(\delta_N HH_1^\dagger)^T]^{-1}. \quad (38)$$

In previous argument, the method to compute B in Lemma 2 when $N \geq L$ is introduced. Next, we consider the problem of initialization in online sequential learning, i.e. when there are not enough training samples. OS-ELM [14] uses a batch-mode initialization method, which can't start the learning process, until enough samples (L) have been collected. Instead, we present a sequential initialization algorithm. In the sense of transpose, the operation of adding samples is symmetric to the one of adding nodes. Since transpose and generalized inverse is convertible, we can adopt a symmetric approach to initialize, where the computation is the transpose of Algorithm 1.

After obtaining $H_2^\dagger$, the new output weights β_2 can be updated by

$$\beta_2 = H_2^\dagger \begin{bmatrix} T_1 \\ \delta_N T \end{bmatrix} = \beta_1 - B\delta_N H\beta_1 + B\delta_N T \quad (39)$$

where $\delta_N T$ is the label vector of new samples.

In the content above, the problems to incrementally update the generalized inverse of the hidden layer output matrix in both the initialization and sequential learning stage, have been solved. Hence, we propose a novel sample incremental ELM (S-IELM) method described in Algorithm 2, where the initialization phase can learn samples one by one, and the sequential learning phase can learning samples chunk by chunk.

IV. DECREMENTAL EXTREME LEARNING MACHINE

In this section, we discuss the situation that some training samples are expired, invalid or mock. As analyzed before, the complexity of ELM depends on the dimension of the hidden layer output matrix H, so keeping features of invalid samples in the H will increase the unnecessary computation. Besides, expired data will cause a sacrifice of performance on the normal data. So, we propose a novel algorithm to eliminate the negative influence of these samples, namely sample decremental ELM (S-DELM), which is the first decremental version of ELM in the literature.

Algorithm 2 Sample Incremental Extreme Learning Machine (S-IELM)

Input The number L of hidden nodes, and parameters of hidden nodes $\{(a_i, b_i)\}_{i=1}^L$, and the sequential data $\aleph = \{(x_i, t_i), i = 1, 2, \cdots\}$.

Output: The new hidden output matrix H, its generalized inverse $H^\dagger$, the new output weight vector β.

Phase I — Initialization

1) $i = 1$, compute $H, H^\dagger, \beta$,

$$H = \left[G(a_1, b_1, x_1)\ \cdots\ G(a_L, b_L, x_1)\right],$$
$$T = t_1, \quad H^\dagger = \frac{H^T}{HH^T}, \quad \beta = H^\dagger T.$$

2) **For** $i = 2 : L$ a)

a) Compute the output vector of hidden layer with respect to x_i, and update the hidden layer output matrix, the label vector δH.

b) Compute d and c

$$d = \delta HH^\dagger, \quad c = \delta H - dH.$$

c) Compute b

$$b = \begin{cases} c^\dagger & \text{if } c \neq 0 \\ \frac{H^\dagger d^T}{1+dd^T} & \text{if } c = 0 \end{cases}.$$

d) Update the Moore-Penrose inverse and ELM

$$H^\dagger \leftarrow \left[H^\dagger - bd\ b\right], \quad H \leftarrow \begin{bmatrix} H \\ \delta H \end{bmatrix},$$
$$\beta \leftarrow \beta - bdT + bt_i, \quad T \leftarrow \begin{bmatrix} T \\ t_i \end{bmatrix}.$$

End For

Phase II — Sequential Learning

When the chunk of new observations $\delta\aleph = \{(x_i, t_i)\}_{i=N+1}^{N+\delta N}$ come,

1) Compute the hidden layer output matrix $\delta_N H$ with respect to new observation $\delta\aleph$.

2) Compute

$$B = H^\dagger(\delta_N HH^\dagger)^T(I + \delta_N HH^\dagger(\delta_N HH^\dagger)^T)^{-1}.$$

3) Update the Moore-Penrose inverse,

$$H^\dagger \leftarrow \left[H^\dagger - B\delta_N HH^\dagger\ B\right].$$

4) Update ELM

$$\beta \leftarrow \beta - B\delta_N H\beta + B\delta_N T,$$
$$H \leftarrow \begin{bmatrix} H \\ \delta_N H \end{bmatrix}, \quad T \leftarrow \begin{bmatrix} T \\ \delta_N T \end{bmatrix}.$$

Given a training set $\aleph = \{(x_i, t_i)\}_{i=1}^N$, a trained ELM with L hidden nodes $\{(a_i, b_i)\}_{i=1}^L$, the hidden layer output matrix is $H_1 \in \mathbb{R}^{N\times L}$, its Moore-Penrose inverse is $H_1^\dagger \in \mathbb{R}^{L\times N}$, and the output weight vector is $\beta_1 = H_1^\dagger T$. Now,

we delete an arbitrary sample from the training set, whose hidden layer feature vector is δH, and the remaining hidden layer output matrix is $H_2 = \begin{bmatrix} H_1 \\ \underline{\delta H} \end{bmatrix} \in \mathbb{R}^{(N-1)\times L}$, where the underline means deleting and the position of deleted sample is arbitrary. How to update $H_2^\dagger$ and β_2 without recomputing the generalized inverse?

First, we propose Lemma 4, which indicates that the order of columns in the generalized inverse matrix is tied to the order of rows in its original matrix. If we change the order of rows in a matrix, it does not need to recompute its generalized inverse, and only needs to make the same change to the order of column in the original generalized inverse. Lemma 4 can be easily proofed by verifying four Penrose equations [32].

Lemma 4: Given a permutation matrix $P \in \mathbb{R}^{m\times m}$, for arbitrary matrix $H \in \mathbb{R}^{m\times n}$, and its generalized inverse $H^\dagger \in \mathbb{R}^{n\times m}$, then the generalized inverse of the permutated matrix PH is

$$(PH)^\dagger = H^\dagger P^T.$$

With Lemma 4, we can assume that the sample to be delete is the first one in the training data set without losing any generality. Then, the problem of decremental ELM is transformed into the problem of updating the generalized inverse $H^\dagger$, after deleting the first row in the hidden layer output matrix H. We partition $H_1^\dagger$ into two parts, $H_1^\dagger = \begin{bmatrix} d & D \end{bmatrix} \in \mathbb{R}^{L\times N}$, where $d \in \mathbb{R}^{L\times 1}$ is the first column of $H_1^\dagger$. Similar as the deduction in Lemma 1, it can be proofed that

$$(I - d\delta H)H_2^\dagger = D. \tag{40}$$

Then, $H_2^\dagger$ can be obtained by solving (40). We denote $\lambda = 1 - \delta Hd$. when $\lambda \neq 0$, according to Jacobson's lemma [36], $I - d\delta H$ is invertible, and

$$(I - d\delta H)^{-1} = I + \frac{d\delta H}{1 - \delta Hd}. \tag{41}$$

Thus, we have

$$H_2^\dagger = D + \frac{d\delta H}{1 - \delta Hd} D. \tag{42}$$

When $\lambda = 0$, $\delta Hd = 1$. According to the property (1), $\delta H = d^\dagger$. It can be noted that $I - d\delta H = I - dd^\dagger$ is an idempotent matrix. Then, it obviously establishes that

$$(I - dd^\dagger)H_2^\dagger = (I - dd^\dagger)D, \tag{43}$$

which leads to

$$dd^\dagger D = 0. \tag{44}$$

Hence, for d and D, their column spaces satisfy $\Re(d) \perp \Re(D)$. Then, by verifying the four Penrose equation, it is easy to obtain the generalized inverse of $\begin{bmatrix} d & D \end{bmatrix}$,

$$\begin{bmatrix} d & D \end{bmatrix}^\dagger = \begin{bmatrix} d^\dagger \\ D^\dagger \end{bmatrix}. \tag{45}$$

Then, we have

$$H_2^\dagger = D. \tag{46}$$

Via Lemma 4 and the argument above, we can fast renew the generalized inverse of the hidden layer output matrix after deleting arbitrary sample. Based on this conclusion, we propose the sample decremental ELM (S-DELM) algorithm in Algorithm 3.

Algorithm 3 Sample Decremental Extreme Learning Machine (S-DELM)

Input: Given a trained ELM, the hidden layer output matrix H, its Moore-Penrose inverse $H^\dagger$, the output weight vector β, the label vector T, the index of the delete sample k.

Output: The new hidden output matrix H, its generalized inverse $H^\dagger$, the new output weight vector β.

1) Collate the hidden layer output matrix H, its Moore-Penrose inverse $H^\dagger$ and the label vector T, move the k-th rows of H and T to the top, move the k-th column of $H^\dagger$ to the left, and write them as

$$H = \begin{bmatrix} \delta H \\ \hat{H} \end{bmatrix}, \quad H^\dagger = \begin{bmatrix} d & D \end{bmatrix}, T = \begin{bmatrix} \delta T \\ \hat{T} \end{bmatrix}.$$

2) Compute $\lambda = 1 - \delta Hd$.

3) Update ELM

$$H \leftarrow \hat{H}, \quad H^\dagger \leftarrow \begin{cases} D + \frac{1}{\lambda} d\delta HD \\ D \end{cases},$$

$$\beta \leftarrow \begin{cases} (I + \frac{1}{\lambda} d\delta H)(\beta - d\delta T) & \text{if } \lambda \neq 0 \\ \beta - d\delta T & \text{if } \lambda = 0 \end{cases}.$$

V. EXPERIMENTAL EVALUATION

In this section, we evaluate the proposed incremental and decremental ELM algorithms is evaluated, compared with other conventional incremental and online sequential ELM methods. The simulation platform is Matlab running on a computer with 3GHz CPU and 4G RAM.

A. DATASETS AND EXPERIMENT SETTINGS

Many public datasets are used in our verification experiments, including 5 regression cases, 9 classification cases (binary or multiple classes). Baseball, Cloud, Strike, Bodyfat and Space-ga, are taken from Statlib. Pima, Spice, Segment, Secom, and Isolet, are taken from UCI Machine Learning Repository. YaleB, ORL, Feret, and Coil-100, are four standard object or human face database, which include images shot under different environments, e.g. poses, viewpoints, illumination, occlusion. Details of these datasets are listed by the first 6 columns in Table 1, including numbers of samples for training and testing, feature dimensions, numbers of classes, and experiment types.

In our experiments, sigmoid additive hidden nodes or multiquadric RBF hidden nodes are adopted according to the performance of ELM on these datasets. The activation function is $G(a, b, x) = 1/(1 + exp(-(a \cdot x + b)))$ for sigmoid additive hidden nodes, and $G(a, b, x) = (\|x - a\|^2 + b^2)^{1/2}$ for

TABLE 1. Specification of datasets and experiment setting.

	# Train	# Test	# Feature	# Class	Type	# Node	Act
Cloud	76	32	4	-	Regression	30	sig
Baseball	226	96	4	-	Regression	50	sig
Strike	438	187	6	-	Regression	100	rbf
Bodyfat	177	75	14	-	Regression	50	rbf
Space-ga	2175	932	6	-	Regression	300	rbf
Segment	1540	770	19	7	Classification	300	sig
Secom	1097	470	590	2	Classification	100	sig
Spice	600	400	60	2	Classification	200	sig
Isolet	3746	2495	617	26	Classification	500	sig
Pima	538	230	8	2	Classification	200	rbf
YaleB	2700	1350	750	10	Classification	500	sig
Feret	480	240	644	120	Classification	200	sig
Coil-100	4200	3000	625	100	Classification	500	sig
ORL	200	200	644	40	Classification	100	sig

TABLE 2. Performance comparison of EI-ELM, CI-ELM, EM-ELM, N-IELM in regression problems, where green, orange, red, and blue stand for the least training time, the second least training time, the least training RMSE, the least testing RMSE, respectively.

	EI-ELM(k=5)			EI-ELM(k=10)			CI-ELM			EM-ELM			N-IELM		
	Time	Train	Test	Time	Train	Test	Time	Train	Test	Time	Train	Test	Time	Train	Test
Cloud	0.0179	0.6333	0.3658	0.0310	0.5726	0.3775	0.0027	0.9784	0.6588	0.0157	0.2233	0.7438	0.0075	0.2354	0.2921
Baseball	0.0304	0.0496	0.0576	0.0575	0.0506	0.0587	0.0052	0.0529	0.0628	0.0180	0.0380	0.0663	0.0157	0.0385	0.0526
Strike	0.0774	0.8436	0.9304	0.1496	0.8245	0.9252	0.0135	0.9252	0.9946	0.1356	0.6687	0.8285	0.0562	0.6751	0.7689
Bodyfat	0.0304	0.7832	0.7551	0.0568	0.7761	0.7634	0.0051	0.8805	0.8638	0.0167	0.6582	0.8693	0.0146	0.5807	0.6514
Space-ga	0.4694	0.6800	0.7793	0.9037	0.6666	0.7654	0.0988	0.8189	0.9167	3.1112	0.4315	0.8321	2.0982	0.4304	0.5248

multi-quadric RBF hidden nodes, where all the hidden-node parameters $\{(a_i, b_i)\}_{i=1}^{L}$ are randomly generated by uniform distribution. Huang *et al.* [1] demonstrate that the performance of ELM with sigmoid additive hidden nodes and multi-quadric RBF hidden nodes is not sensitive to the number L of hidden nodes , as long as L is large enough. But too many hidden nodes will decrease the computation efficiency of both the batch-mode ELM and incremental decremental ELMs. Mhaskar and Micchelli [37] discuss how to choose activation function for a feedforward neural network by smooth derivatives. In our experiment, numbers of hidden nodes are determined empirically by compromising the performance with the efficiency of ELM on reference datasets, and they are chosen the same to all compared methods for a fair competition. The detailed experiment settings are presented by the last 2 columns in Table 1, including activation function types and numbers of hidden nodes. In our experiments, sigmoid additive hidden nodes normally provide better performances on reference datasets, except Strike, Bodyfat, Space-ga, Pima, where RBF hidden nodes are more suitable.

In the following, three experiments are presented to demonstrate the performance of the proposed incremental and decremental ELM algorithms: adding hidden nodes in section V-B, adding samples in Section V-C, deleting samples in Section V-D, with respect to N-IELM, S-IELM, S-DLEM.

B. ADDING HIDDEN NODES

We first report the experiment of adding hidden nodes, and compare the proposed N-IELM with other hidden node incremental ELM methods: EI-ELM, CI-ELM, EM-ELM. Because the number of candidate hidden nodes have an impact on the performance of EI-ELM, we try two different parameters ($k = 5,\ 10$) for EI-ELM. In our experiment, hidden nodes are increased one by one, until the number L reaches the maximum L_{max}. Each method is given 50 trials on every dataset, and then the average training time, the average training and testing RMSE for regression problems, the average training and testing classification rate for classification problems are recorded for performance comparison.

1) REGRESSION PROBLEMS

Five benchmark datasets are used for simulation, i.e. Cloud, Baseball, Strike, Bodyfat, Space-ga. Table 2 presents the results of regression problems on these datasets in terms of the training time, training RMSE, and testing RMSE. From the first column of every method's contents in Table 2, it can be observed that N-IELM has the second least training time on all datasets except Space-ga. Although CI-ELM achieves the fastest training speed, it obtains the worst training and testing RMSE. As we mentioned before, without calculating the generalized inverse $H^{\dagger}$, the computation complexity of EI-ELM and CI-ELM mainly is generating the new random hidden node; the computation complexity of EM-ELM and N-IELM are updating the generalized inverse $H^{\dagger}$. Therefore, CI-ELM can achieve the best efficiency. When the input dimension and the size of dataset are both small, such as Baseball, Cloud, Strike and Bodyfat, our N-IELM is faster than EI-ELM ($k = 5$). But, when the input dimension is small and the size of dataset is large, such as Space-ga, the computation of updating the generalized inverse of a large hidden layer output matrix H is more than the computation of generating 5 times candidate hidden nodes, which results in N-IELM slower than EI-ELM ($k = 5$) on Space-ga.

TABLE 3. Performance comparison of EI-ELM, CI-ELM, EM-ELM, N-IELM in classification problem, where green, orange, red, and blue stand for the least training time, the second least training time, the best training classification rate, the best testing classification rate, respectively.

	EI-ELM(k=5)			EI-ELM(k=10)			CI-ELM			EM-ELM			N-IELM		
	Time	Train	Test	Time	Train	Test	Time	Train	Test	Time	Train	Test	Time	Train	Test
Segment	0.9270	0.8647	0.8840	1.8014	0.8780	0.8889	0.1933	0.7393	0.7407	3.5224	0.9793	0.9506	1.6430	0.9813	0.9568
Secom	0.1347	0.9353	0.9319	0.2517	0.9371	0.9319	0.0228	0.9335	0.9340	0.3982	0.9371	0.9277	0.1289	0.9380	0.9340
Spice	0.2118	0.8983	0.7350	0.4009	0.9217	0.7050	0.0381	0.7467	0.6475	0.3936	0.8900	0.6625	0.2684	0.9100	0.7425
Isolet	13.388	0.8907	0.8681	25.239	0.8910	0.8637	2.4865	0.8015	0.7844	16.551	0.9613	0.9138	12.629	0.9613	0.9170
Pima	0.2276	0.8457	0.7130	0.4129	0.8569	0.6826	0.0408	0.7546	0.7174	0.3837	0.9052	0.6870	0.2198	0.9071	0.7261
YaleB	47.006	0.8444	0.7074	94.739	0.8385	0.6993	6.0456	0.6789	0.5615	17.015	0.9844	0.7526	12.467	0.9904	0.7585
Feret	5.7434	0.9729	0.7500	11.385	0.9604	0.7458	0.8512	0.8833	0.6375	0.4443	1.0000	0.7708	0.3127	1.0000	0.7875
Coil-100	93.423	0.6710	0.6290	182.01	0.6721	0.6400	10.128	0.5681	0.5450	26.152	0.9760	0.9053	20.128	0.9774	0.9007
ORL	0.3941	0.9500	0.8100	0.7832	0.9700	0.8350	0.1061	0.8950	0.7900	0.0442	1.0000	0.8700	0.0436	1.0000	0.8850

From the last two columns of every method's contents in Table 2, we can see that N-IELM has the best or second best training RMSE, the best testing RMSE in all compared methods. The reason is that N-IELM updates the generalized inverse $H^{\dagger}$, which assures the optimal approximation ability and the optimal generalization performance.

2) CLASSIFICATION PROBLEMS

Table 3 presents the performance comparison of classification experiments for EI-ELM, CI-ELM, EM-ELM and N-IELM, in terms of the average training time, average training classification rate, and average testing classification rate. Observing this table, it can be noted that N-IELM generally achieve a comparable efficiency among the methods considered. On most datasets, N-IELM has the second least training time, only slower than CI-ELM. On Spice and Segment, it spends more training time than both CI-ELM and EI-ELM ($k = 5$), and the reason is similar as we analyzed in the regression problems. Besides, it is worth noted that N-IELM cost the least training time in all methods on Feret and ORL. The reason is that these two datasets have a high feature dimension, a small size, and many classes (40 and 120). The computation of generating new hidden nodes is the same for CI-ELM and N-IELM. The computation of update the generalized inverse $H^{\dagger}$ is small for N-IELM. CI-ELM is proposed for ELM with one single output node, so it has to be repeated to update the output weight of each output node for ELM with multiple output nodes. Consequently, CI-ELM becomes lower efficiency than N-IELM on Feret and ORL.

From the last two columns of every method's contents in Table 3, it can be observed that N-IELM achieve better classification rates on both training and testing on most datasets than other methods, except Spice and Coil-100. It worths being noticed that because of maintaining the generalized inverse $H^{\dagger}$ and the global optimality, N-IELM can obtain nearly perfect training results. For example, training classification rates are greater than 0.99 on YaleB, Feret, and ORL.

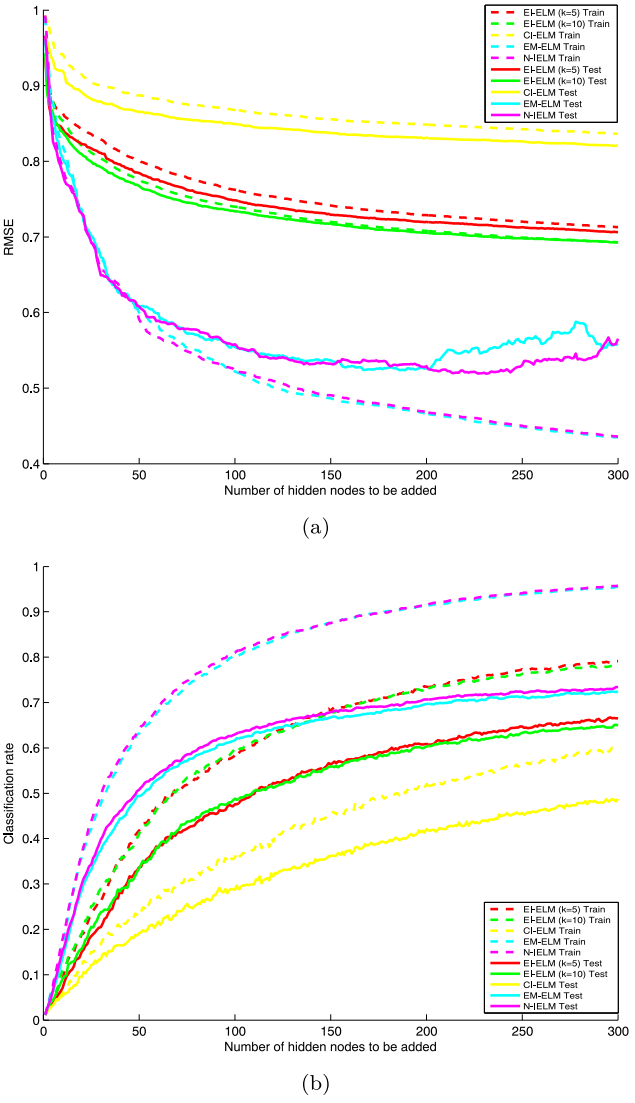

FIGURE 2. Learning updating curves of EI-ELM, CI-ELM, EM-ELM, N-IELM on different datasets, where different colors stand for different datasets, dash lines stand for the training results, solid lines stand for the testing results. (a) Space-ga. (b) Yaleb.

3) PERFORMANCE CURVES WITH INCREASING HIDDEN NODES

Figure 2 shows the learning updating curves of training and testing results during the learning process of increasing hidden nodes one by one for EI-ELM, CI-ELM, EM-ELM, and N-IELM on Space-ga and Yaleb. For Space-ga, presented by Figure 2(a), the activation function is RBF, the number of max hidden nodes is 300, and the evaluating indicator is

TABLE 4. Performance comparison of OS-ELM, S-IELM in regression problem when the chunk size is 10, where green, red, blue and cyan stand for the least training time, the least training RMSE, the least testing RMSE, and the least numerical crash times, respectively.

	OS-ELM				S-IELM			
	Time	Train	Test	# Fault	Time	Train	Test	# Fault
Cloud	0.0049	0.2020	0.6843	1	0.0027	0.1959	0.5959	0
Baseball	0.0116	1.3072	2.6139	10	0.0114	0.0400	0.0522	0
Strike	0.0302	0.7193	0.9610	1	0.0327	0.7137	0.9471	0
Bodyfat	0.0072	0.6604	0.8807	0	0.0067	0.6457	0.8908	0
Space-ga	0.3750	0.4499	0.5731	7	0.8060	0.4413	0.5657	0

TABLE 5. Performance comparison of OS-ELM, S-IELM in classification problem when the chunk size is 10, where green, red, blue and cyan stand for the least training time, the best training classification rate, the best testing classification rate, and the least numerical crash times, respectively.

	OS-ELM				S-IELM			
	Time	Train	Test	# Fault	Time	Train	Test	# Fault
Segment	0.2769	0.9498	0.9415	8	0.4518	0.9497	0.9484	0
Secom	0.0780	0.9338	0.9337	0	0.2270	0.9340	0.9335	0
Spice	0.1306	0.8791	0.7360	0	0.1567	0.8620	0.7233	0
Isolet	6.4191	0.9618	0.9112	29	9.5939	0.9616	0.9199	0
Pima	0.1050	0.8835	0.6867	0	0.1099	0.8835	0.6872	0
YaleB	4.5519	0.9866	0.7593	1	5.4742	0.9869	0.7608	0
Feret	0.0980	1.0000	0.7894	1	0.0905	1.0000	0.7893	0
Coil-100	7.4032	0.9763	0.8973	10	11.5350	0.9761	0.8987	0
ORL	0.0214	0.9986	0.8725	0	0.0165	0.9999	0.8730	0

average training and testing RMSE. For Yaleb, presented by Figure 2(b), the activation function is sigmoid, the number of max hidden nodes is also 300, and the evaluating indicator is average training and testing classification rate. From Figure 2, it can be found that the performance of N-IELM is much better than EI-ELM and CI-ELM on both training and testing, and the training results of N-IELM is very close to EM-ELM, and its testing outcome is better than EM-ELM. From these figures, it is worth to note that to achieve the same performance, the number of hidden nodes needed for N-IELM is much smaller than the numbers for EI-ELM and CI-ELM, even if EI-ELM has a scheme of multiple candidate hidden nodes.

C. ADDING SAMPLES

In this section, we evaluate the sequential learning performance of the proposed S-IELM, compared with OS-ELM. For S-IELM, the experiment is executed in two steps: the first L samples are appended one by one in the initial phase, and then the following samples are added chunk by chunk in the sequential learning phase. For OS-ELM, which can't self-start, the experiment is started by a batch-mode ELM with the first L samples, and then the remaining sample are added chunk by chunk in the sequential learning phase, where the chunk size is the same as S-IELM.

The experiment settings on different datasets, such as types and numbers of hidden nodes, are the same as the settings in the former experiment of adding nodes, which are listed in Table 1. Both methods are repeated 50 times on every dataset, and the average results are taken down for performance comparison. Four evaluating indicators are used: the first three are the same as the last experiment, i.e. the average training time, average training RMSE or classification rate, and average testing RMSE or classification rate; because OS-ELM has the problem of numerical instability, we add the forth indicator, namely Fault, which counts the times that numerical collapses occur. The principle to decide if a numerical collapse occurs is that if there is any element with an absolute value greater than 10^{10} in the inverse or generalized inverse matrix, which should not exist in the normal situation.

1) REGRESSION PROBLEMS

Table 4 presents the results of OS-ELM and S-IELM in online sequential regression experiments on 5 datasets, where the chunk size is 10. From columns of time in Table 4, it can be seen that S-IELM costs less training times than OS-ELM when the size of dataset is small, such as, Cloud, Baseball, and Bodyfat, but when the size of dataset grows, such as Strike and Space-ga, it costs more training times than OS-ELM. There are two reasons for this: first, when recording training time, the time of initialization in S-IELM is taken into account; second, S-IELM directly updates the generalized inverse $H^{\dagger}$, which is more complex than the inverse matrix used in OS-ELM.

The second and third columns of each method's content in Table 4 present the average training and testing RMSE, where those trials with numerical collapse have been removed before statistics. From these results, it can be observed that S-ELM achieve better approximation performance on both training and testing. The reason is that both OS-ELM and S-IELM calculate the generalized inverse $H^{\dagger}$, but the result of OS-ELM is more possible to be disturbed because of numerical instability. The column of Fault in Table 4 records times that OS-ELM and S-IELM break down. OS-ELM has totally 19 times algorithm collapse in 250 trials on 5 datasets, but the number for S-IELM is 0.

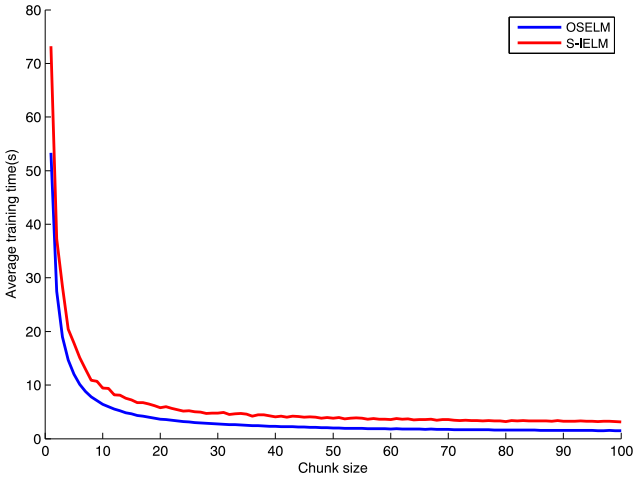

FIGURE 3. Training time of OS-ELM, S-IELM by chunk size (1-100) on Isolet, where the number of hidden nodes is 500.

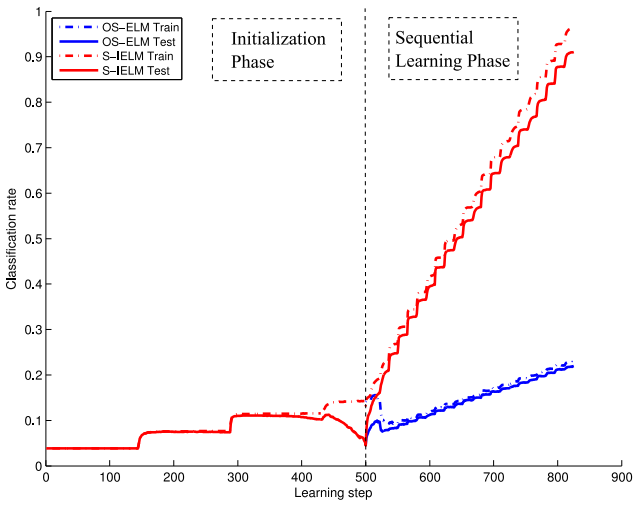

FIGURE 4. Classification rate of OS-ELM, S-IELM by learning step on Isolet, where the number of hidden nodes is 500. In the first 500 step of initialization phase the chunk size is 1, and in the sequential learning phase the chunk size is 10.

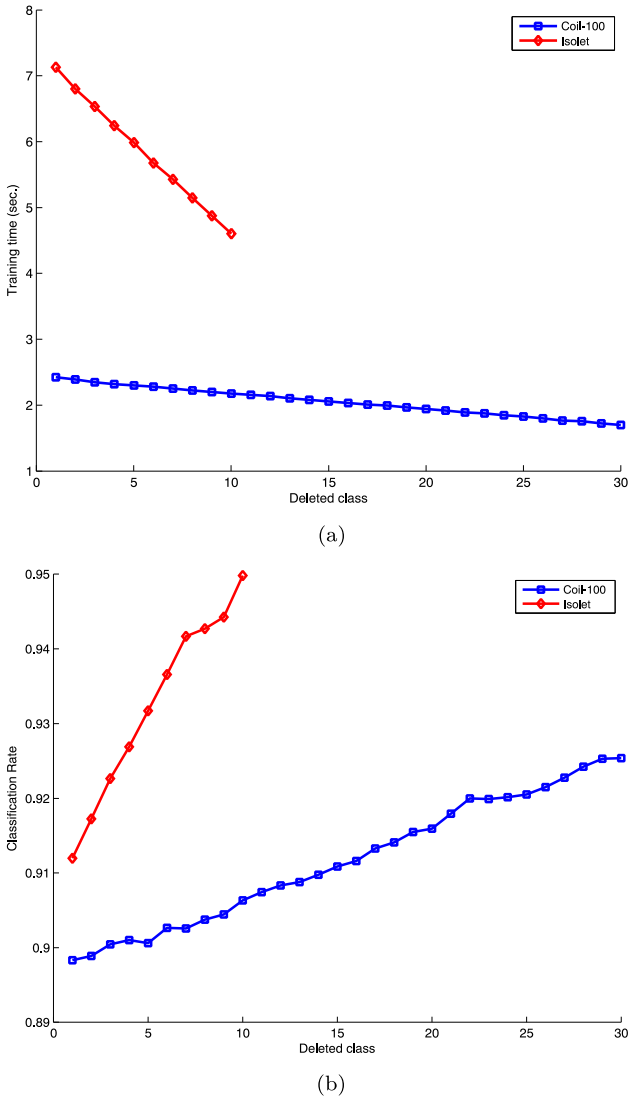

FIGURE 5. Performance of S-DELM on Isolet and Coil-100, where the type and number of hidden nodes are sigmoid 500. On Isolet, samples of 10 classes in 26 are deleted; on Coil-100, samples of 30 classes in 100 are deleted. (a) Cpu time. (b) Classification rate.

2) CLASSIFICATION PROBLEMS

Table 5 shows the results of OS-ELM and S-IELM in online sequential classification experiments, where the experiment settings are listed in Table 1, the chunk size is 10, and both methods are given 50 trials on each dataset. From the column of time in Table 5, similar results can be found as regression experiments that S-IELM spends more training times than OS-ELM. The average training and testing classification rates shown in Table 5, where only results in normal trials are taken into account to compute the equalizing value, indicate that the performance of S-IELM is better than OS-ELM. However, from the column of Fault in Table 5, it can be seen that S-IELM is more stable than OS-ELM. In the most severe situation, i.e. on Isolet, OS-ELM bursts up 29 times, but S-IELM always works normally.

The similar conclusion can also be obtained from Figure 3 and 4. Figure 3 shows the training time curve by chunk size (0-100) of OS-ELM and S-IELM on Isolet when the number of hidden nodes is 500. The red curve of S-IELM is above the blue one of OS-ELM a little, i.e. it is a little slower than OS-ELM. Figure 4 presents the training and testing classification rate updating curve of OS-ELM and S-IELM on Isolet. The cross axle stands for the index of sequential learning round in our experiment. The number of hidden nodes is 500 in this experiment, so the 500 steps on the left side of the black dash line is the initialization phase, where samples are learned one by one for S-IELM. Because OS-ELM is started with a batch-mode ELM, the blue lines have no content on this side. The right side of the black dash line is the sequential learning phase. The longitudinal axis stands for the average training and testing results, where all trials are taken into account, including normal and abnor-

mal trials. From Figure 4, it can be observed that S-IELM achieves much higher training and testing classification rate than OS-ELM, due to the effect of numerical instability on OS-ELM.

D. DELETING SAMPLES

In this section, we demonstrate the effectiveness of the novel application of ELM, i.e. sample decremental ELM (S-DELM). For a better presentation, our criterion to select a dataset in the decremental experiment is a classification dataset with a large size and multiple classes, and finally Isolet and Coil-100 are chosen, which have 6241 samples of 26 classes, and 7200 samples of 100 classes, respectively. In this experiment, we first learn the whole training set by the batch-mode ELM, then randomly select some classes as the expired class and delete their samples with our S-DELM algorithm (10 in 26 classes for Isolet, 30 in 100 classes for Coil-100). The same types and numbers of hidden nodes are used here, as listed in Table 1. S-DELM is repeated 10 times on both data sets. The time of deleting samples of one expired class, and the instantaneous classification rate on the remaining testing samples are recorded, and their means are used for performance evaluation.

Figure 5(a) shows the average training time curve of S-DELM when deleting classes, which demonstrates that in every step of the decremental learning process the operating time is decreasing, because the computation complexity of ELM is leveled down. On Isolet, the running time has decreased from 7.2s to 4.5s, after deleting 10 classes. On Coil, it has decreased from 2.4s to 1.8s, after deleting 30 classes. Figure 5(b) shows the average testing classification rate of S-DELM when deleting classes. From this figure, we can see that after deleting the expired samples from the training set, the classification rate has been increased, from 0.91 to 0.95 on Isolet, and from 0.89 to 0.92 on Coil-100.

VI. CONCLUSION

In this paper, a universal solution based on generalized inverse is proposed for incremental and decremental ELM. Unlike other incremental and online sequential ELM methods, the problem of adding nodes, adding/deleting samples are turned into the problem of updating/downdating the generalized inverse $H^{\dagger}$ of the hidden layer output matrix when adding/deleting rows or columns. As a major benefit, our solution preserves the optimality of ELM, i.e. the best approximation error, and the best generalization performance. Three algorithms, namely node incremental ELM (N-IELM), sample incremental ELM (S-IELM), sample decremental ELM (S-DELM), are proposed to achieve adding nodes, adding samples, deleting samples for ELM. S-IELM also overcomes the initialization and numerical instability problems of OS-ELM. The simulation results on plenty of real-world datasets demonstrate that our methods have better accuracy, better numerical stability, and comparable efficiency, compared to other conventional methods.

REFERENCES

[1] G.-B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 42, no. 2, pp. 513–529, Apr. 2012.
[2] C. M. Bishop, *Pattern Recognition and Machine Learning* (Information Science and Statistics). New York, NY, USA: Springer-Verlag, 2007.
[3] S. S. Haykin, *Neural Networks: A Comprehensive Foundation*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1994.
[4] W. Zong and G.-B. Huang, "Face recognition based on extreme learning machine," *Neurocomputing*, vol. 74, no. 16, pp. 2541–2551, 2011.
[5] R. Zhang, G. B. Huang, N. Sundararajan, and P. Saratchandran, "Multicategory classification using an extreme learning machine for microarray gene expression cancer diagnosis," *IEEE/ACM Trans. Comput. Biol. Bioinf.*, vol. 4, no. 3, pp. 485–495, Jul. 2007.
[6] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, Oct. 1986.
[7] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: A new learning scheme of feedforward neural networks," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, vol. 2. Jul. 2004, pp. 985–990.
[8] G.-B. Huang, L. Chen, and C.-K. Siew, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Trans. Neural Netw.*, vol. 17, no. 4, pp. 879–892, Jul. 2006.
[9] P. L. Bartlett, "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*, vol. 44, no. 2, pp. 525–536, Mar. 1998.
[10] J. C. Yan, C. H. Tian, J. Huang, and F. Albertao, "Incremental dictionary learning for fault detection with applications to oil pipeline leakage detection," *Electron. Lett.*, vol. 47, no. 21, pp. 1198–1199, 2011.
[11] L. Lan, D. Tao, C. Gong, N. Guan, and Z. Luo, "Online multi-object tracking by quadratic pseudo-boolean optimization," in *Proc. IJCAI*, 2016, pp. 3396–3402.
[12] C. Li, J. Yan, F. Wei, W. Dong, Q. Liu, and H. Zha, "Self-paced multi-task learning," in *Proc. AAAI*, 2017, pp. 2175–2181.
[13] C. Li, F. Wei, J. Yan, X. Zhang, Q. Liu, and H. Zha, "A self-paced regularization framework for multilabel learning," *IEEE Trans. Neural Netw. Learn. Syst.*, to be published, doi: 10.1109/TNNLS.2017.2697767.
[14] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate online sequential learning algorithm for feedforward networks," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1411–1423, Nov. 2006.
[15] G. H. Golub and C. F. Van Loan, *Matrix Computations*, vol. 3. Baltimore, MD, USA: Johns Hopkins Univ. Press, 2012.
[16] G.-B. Huang, D. H. Wang, and Y. Lan, "Extreme learning machines: A survey," *Int. J. Mach. Learn. Cybern.*, vol. 2, no. 2, pp. 107–122, Jun. 2011.
[17] G.-B. Huang and L. Chen, "Enhanced random search based incremental extreme learning machine," *Neurocomputing*, vol. 71, nos. 16–18, pp. 3460–3468, Oct. 2008.
[18] G.-B. Huang and L. Chen, "Convex incremental extreme learning machine," *Neurocomputing*, vol. 70, nos. 16–18, pp. 3056–3062, 2007.
[19] A. R. Barron, "Universal approximation bounds for superpositions of a sigmoidal function," *IEEE Trans. Inf. Theory*, vol. 39, no. 3, pp. 930–945, May 1993.
[20] G. Feng, G.-B. Huang, Q. Lin, and R. Gay, "Error minimized extreme learning machine with growth of hidden nodes and incremental learning," *IEEE Trans. Neural Netw.*, vol. 20, no. 8, pp. 1352–1357, Aug. 2009.
[21] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, nos. 1–3, pp. 489–501, 2006.
[22] S. P. Boyd, *Linear Matrix Inequalities in System and Control Theory*, vol. 15. Philadelphia, PA, USA: SIAM, 1994.
[23] Y. Lan, Y. C. Soh, and G.-B. Huang, "Random search enhancement of error minimized extreme learning machine," in *Proc. Eur. Symp. Artif. Neural Netw. (ESANN)*, 2010, pp. 327–332.
[24] Z. Xu, M. Yao, Z. Wu, and W. Dai, "Incremental regularized extreme learning machine and it's enhancement," *Neurocomputing*, vol. 174, pp. 134–142, Jan. 2016.
[25] C. Hou and Z.-H. Zhou. (May 2016). "One-pass learning with incremental and decremental features." [Online]. Available: https://arxiv.org/abs/1605.09082
[26] P. Hall, D. Marshall, and R. Martin, "Merging and splitting eigenspace models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, no. 9, pp. 1042–1049, Sep. 2000.

[27] B. Jin, Z. Jing, and H. Zhao, "EVD dualdating based online subspace learning," *Math. Problems Eng.*, vol. 2014, Jul. 2014, Art. no. 429451.
[28] S. Pang, T. Ban, Y. Kadobayashi, and N. Kasabov, "Incremental and decremental LDA learning with applications," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jun. 2010, pp. 1–8.
[29] G. Cauwenberghs and T. Poggio, "Incremental and decremental support vector machine learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2001, pp. 409–415.
[30] M. Karasuyama and I. Takeuchi, "Multiple incremental decremental learning of support vector machines," *IEEE Trans. Neural Netw.*, vol. 21, no. 7, pp. 1048–1059, Jul. 2010.
[31] C.-H. Tsai, C.-Y. Lin, and C.-J. Lin, "Incremental and decremental training for linear classification," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2014, pp. 343–352.
[32] R. Penrose, "A generalized inverse for matrices," *Math. Proc. Cambridge Philos. Soc*, vol. 51, no. 3, pp. 406–413, 1955.
[33] A. Ben-Israel and T. N. Greville, *Generalized Inverses Theory and Applications*, vol. 15. New York, NY, USA: Springer-Verlag, 2003.
[34] G.-B. Huang, "Learning capability and storage capacity of two-hidden-layer feedforward networks," *IEEE Trans. Neural Netw.*, vol. 14, no. 2, pp. 274–281, Mar. 2003.
[35] R. E. Cline, "Representations for the generalized inverse of a partitioned matrix," *J. Soc. Ind. Appl. Math.*, vol. 12, no. 3, pp. 588–600, 1964.
[36] G. Zhuang, J. Chen, and J. Cui, "Jacobson's lemma for the generalized drazin inverse," *Linear Algebra Appl.*, vol. 436, no. 3, pp. 742–746, 2012.
[37] H. N. Mhaskar and C. A. Micchelli, "How to choose an activation function," in *Proc. Adv. Neural Inf. Process. Syst.*, 1994, pp. 319–326.

BO JIN received the B.S. degree in electronic engineering and the Ph.D. degree in control theory and control engineering from Shanghai Jiao Tong University, China, in 2004 and 2014, respectively. He is currently a Lecturer with the School of Computer Science and Software Engineering, East China Normal University. His major research interests include machine learning, online learning, face recognition, visual tracking, and medical diagnosis modeling.

ZHONGLIANG JING received the B.S., M.S., and Ph.D. degrees in electronics and information technology from Northwestern Polytechnical University, Xi'an, China, in 1983, 1988, and 1994, respectively. He is currently the Cheung Kong Professor and the Executive Dean with the School of Aeronautics and Astronautics, Shanghai Jiao Tong University, China. His major research interests include multi-source information acquisition, processing and fusion, target tracking, and aerospace control. He is an Editorial Board Member of the *Science China: Information Sciences*, *Chinese Optics Letters*, and the *International Journal of Space Science and Engineering*.

HAITAO ZHAO received the M.S. degree in applied mathematics and the Ph.D. degree in pattern recognition and artificial intelligence from the Nanjing University of Science and Technology, Nanjing, China, in 2000 and 2003, respectively. He is currently a Full Professor with the School of Information Science and Engineering, East China University of Science and Technology, Shanghai, China. His major research interests include machine learning, pattern recognition, and computer vision.