Stacked Extreme Learning Machines

Hongming Zhou, Guang-Bin Huang, Zhiping Lin, Han Wang, and Yeng Chai Soh

***Abstract*—Extreme learning machine (ELM) has recently attracted many researchers' interest due to its very fast learning speed, good generalization ability, and ease of implementation. It provides a unified solution that can be used directly to solve regression, binary, and multiclass classification problems. In this paper, we propose a stacked ELMs (S-ELMs) that is specially designed for solving large and complex data problems. The S-ELMs divides a single large ELM network into multiple stacked small ELMs which are serially connected. The S-ELMs can approximate a very large ELM network with small memory requirement. To further improve the testing accuracy on big data problems, the ELM autoencoder can be implemented during each iteration of the S-ELMs algorithm. The simulation results show that the S-ELMs even with random hidden nodes can achieve similar testing accuracy to support vector machine (SVM) while having low memory requirements. With the help of ELM autoencoder, the S-ELMs can achieve much better testing accuracy than SVM and slightly better accuracy than deep belief network (DBN) with much faster training speed.**

***Index Terms*—Deep learning, eigenvalue, extreme learning machine (ELM), feature mapping, principal component analysis (PCA), support vector machines (SVMs).**

I. Introduction

EXTREME learning machine (ELM) as a type of generalized single-hidden layer feed-forward networks has recently been studied by many researchers in theory and applications [1]. The hidden node parameters of ELM (including input weights and hidden layer biases) are randomly generated and the output weights are then analytically determined by using the generalized inverse method. Huang *et al*. [2] proposed the equality constrained optimization method-based ELM, and provided two solutions for different size of training data. The user can choose to use training data size based (complexity of computation is based on the size of training data) for the small to medium size training data, or hidden nodes size based (complexity of computation is based on the number of hidden nodes). The training data is usually mapped from an original input space to a higher dimensional space, within which the tackled problem can be solved. The number of hidden nodes determines the dimensionality of the mapped space. In general, to solve a big data problem, the more complicated the training instances are, the more hidden nodes would be required for training a generalized model to approximate the target functions. This brings problems to ELM when facing extremely large and complex training datasets. The large number of required hidden nodes makes ELM network very large and the computational task becomes very costly.

There are currently several approaches taken to tackle big dataset problems. Super computer with clusters of CPUs is of course a direct approach to solve such a large dataset problems. With the computing power of super computer, e.g., the Cray Titan super computer at Oak Ridge National Laboratory, with 560 K cores, speed at 17.59 PFLOP/s and 710 K GB memory [3], many large dataset problems can be easily solved without any constraints on memory and speed. However, super computer is not always available to many researchers and it is very costly on both hardware expense and power consumption. Introduced by Google in 2003, the MapReduce model is a parallel computing framework that can process large data in a distributed environment on clusters of computers [4]–[6]. Recently, with the introduction of NVidia CUDA parallel computing platform [7] in 2007, GPU computing [8] becomes popular to accelerate and solve larger-scale problems. There are also several approaches working on computational algorithms to reduce the computational complexity, e.g.: 1) approximation algorithms on original data samples such as greedy approximation [10], reduced support vector machines (SVMs) [9], Nyström method [11] and core vector machines [12]; 2) chunking method [13] or decomposition method [14]; and 3) sequential or incremental learning algorithms such as resource allocation network [15], online sequential ELM [16], and incremental SVM [17]. However, there are some limitations with those algorithms: 1) some of the traditional approximation algorithms on original data may not work well if the original training data is very complicated and 2) none of those algorithms deal with the original whole batch of training data at one time. This may lead to a biased result that misses some of the original information.

To solve large and complex data problems using ELM without incurring a memory problem, one should keep the network size small yet be able to achieve good generalization accuracy. This paper proposes a stacked ELMs (S-ELMs) algorithm that tries to break one large ELM network into multiple sub-ELMs to complete the targeted machine learning task. The multiple sub-ELMs are stacked on multiple layers that are serially connected. As the hidden node parameters of ELM are randomly generated, the importance of different nodes that contribute to the target output may vary a lot. We can choose the most significant few percents of nodes (or combined nodes, such as a few nodes combined together as one node with some nonlinear relationship) to represent all the nodes for a single

Manuscript received December 18, 2013; revised June 27, 2014 and September 29, 2014; accepted October 5, 2014. Date of publication October 28, 2014; date of current version August 14, 2015. This work was supported in part by the grant from Singapore Academic Research Fund (AcRF) Tier 1 under Project RG 22/08 (M52040128), and in part by the Singapore's National Research Foundation under Grant NRF2011NRF-CRP001-090. This paper was recommended by Associate Editor X. Zeng.

The authors are with the School of Electrical and Electronic Engineering, Nanyang Technological University, Nanyang, Singapore 639798 (e-mail: hmzhou@ntu.edu.sg).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCYB.2014.2363492

ELM in each layer. The lower layer can output such nodes to the ELM network in the upper layer. Those nodes are then combined with the new random nodes and function as the total hidden layer ouput of upper layer ELM. In the upper layer, the same procedure as the previous layer can be taken to output the most significant nodes to the higher layer. In this manner, we can keep the whole network size fixed yet be able to learn the data in a higher dimensional ELM feature space. In this paper, we choose the top few significant nodes using their output weights' eigenvalues, and reduce the nodes by multiplying the corresponding eigenvectors. This can be done by performing PCA dimension reduction [18] on the output weights calculated in each small ELM. The principal components are now the output weights of the selected significant nodes which are computed from the basis vectors. Likewise, Deng and Yu [19] also proposed a stacked multiple module learning network that uses both restricted Boltzmann machine and ELM to determine the hidden layer components in each module.

To further improve the testing rate, especially for the unstructured large data without properly selected features, we implemented ELM autoencoder in each iteration of S-ELMs algorithm. ELM autoencoder is an autoencoder that uses ELM algorithm as the training algorithm to reconstruct the input data. The ELM autoencoder can serve as unsupervised pretraining of data and help capture interesting input data structure that may be useful for improving model generalization ability. Thus, ELM autoencoder-based S-ELMs (AE-S-ELMs) may achieve better generalization performance. The simulation results show that the S-ELMs can achieve very close testing accuracy to SVMs result even with the random hidden nodes. The ELM AE-S-ELMs can achieve much better testing accuracy than SVM and slightly better accuracy than deep belief network (DBN) with much faster training speed. Both of them (S-ELMs and ELM AE-S-ELMs) are capable of solving very large data problems without incurring the out of memory problem.

This paper is organized as follows: in Section II, the basics of ELM and its solutions are revised, the concept of principal component analysis (PCA) and the autoencoder network are also briefly introduced; in Sections III and IV, the details of proposed S-ELMs and ELM AE-S-ELMs are presented; in Section V, the generalization capability and computational complexity are discussed; in Section VI, we will show the performance of the S-ELMs and ELM AE-S-ELMs on four large datasets and discuss the effects of different parameters; in Section VII, we conclude this paper.

II. Related Works

This section briefly introduces ELM (including the preliminary ELM and equality constrained optimization method-based ELM), PCA, and autoencoder networks.

A. ELM

1) Preliminary ELM: ELM was originally proposed by Huang *et al.* [25]–[27] for the single-hidden-layer feedforward neural networks and then extended to the generalized single-hidden-layer feedforward networks where the hidden layer need not be neuron alike [28], [29]. The hidden layer parameters are randomly generated without tuning and are independent of the training data. The input data is mapped from the input space to the L-dimensional hidden layer feature space (ELM feature space). The output of ELM can be written as

$$f_L(\mathbf{x}) = \sum_{i=1}^{L} \beta_i h_i(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta} \tag{1}$$

where $\boldsymbol{\beta} = [\beta_1, \ldots, \beta_L]^T$ is the matrix of the output weights from the hidden nodes to the output nodes. $\mathbf{h}(\mathbf{x}) = [h_1(\mathbf{x}), \ldots, h_L(\mathbf{x})]$ is the row vector representing the outputs of L hidden nodes with respect to the input $\mathbf{x}$. $\mathbf{h}(\mathbf{x})$ actually maps the data from the d-dimensional input space to the L-dimensional hidden layer feature space (*ELM feature space*) $\mathbf{H}$, and thus, $\mathbf{h}(\mathbf{x})$ is indeed a feature mapping.

The above linear equations can be written in the matrix form

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{T} \tag{2}$$

where $\mathbf{H}$ is the hidden layer output matrix

$$\mathbf{H} = \begin{bmatrix} \mathbf{h}(\mathbf{x}_1) \\ \vdots \\ \mathbf{h}(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} h_1(\mathbf{x}_1) & \cdots & h_L(\mathbf{x}_1) \\ \vdots & \vdots & \vdots \\ h_1(\mathbf{x}_N) & \vdots & h_L(\mathbf{x}_N) \end{bmatrix} \tag{3}$$

and $\mathbf{T} = [\mathbf{t}_1, \ldots, \mathbf{t}_N]^T$ is a matrix of target labels. The solution of above equation is given as: $\boldsymbol{\beta} = \mathbf{H}^{\dagger}\mathbf{T}$, where $\mathbf{H}^{\dagger}$ is the Moore–Penrose generalized inverse [30], [31] of matrix $\mathbf{H}$.

2) Equality Constrained Optimization Method-Based ELM: The equality constrained optimization method-based ELM [2] is proposed from the standard optimization point of view to solve ELMs linear equation (2) as well as minimize the output weights $\|\boldsymbol{\beta}\|$.

According to ELM learning theory, widespread type of feature mappings $\mathbf{h}(\mathbf{x})$ can be used in ELM so that ELM can approximate any continuous target functions [27]–[29]. That is, given any target continuous function $f(\mathbf{x})$ there exist a series of β_i such that

$$\lim_{L\to+\infty} \|f_L(\mathbf{x}) - f(\mathbf{x})\| = \lim_{L\to+\infty} \left\| \sum_{i=1}^{L} \beta_i h_i(\mathbf{x}) - f(\mathbf{x}) \right\| = 0. \tag{4}$$

With this universal approximation capability, the classification problem of the proposed equality constrained optimization-based ELM can be formulated as

$$\begin{aligned} \text{Minimize: } & L_{P_{\text{ELM}}} = \frac{1}{2}\|\boldsymbol{\beta}\|^2 + C\frac{1}{2}\sum_{i=1}^{N}\xi_i^2 \\ \text{Subject to: } & \mathbf{h}(\mathbf{x}_i)\boldsymbol{\beta} = t_i - \xi_i, \ i = 1, \ldots, N. \end{aligned} \tag{5}$$

Different solutions to the above optimization equation can be obtained according to the size of training datasets, thus to minimize the computational complexity.

In the case that the training dataset is relatively small, the output function of ELM classifier can be obtained as

$$\mathbf{f}(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta} = \mathbf{h}(\mathbf{x})\mathbf{H}^T \left(\frac{\mathbf{I}}{C} + \mathbf{H}\mathbf{H}^T \right)^{-1} \mathbf{T}. \tag{6}$$

If a feature mapping $\mathbf{h}(\mathbf{x})$ is unknown to users, one can apply Mercer's conditions on ELM and define the kernel matrix for ELM as follows:

$$\mathbf{\Omega}_{\text{ELM}} = \mathbf{H}\mathbf{H}^T : \Omega_{\text{ELM}i,j} = h(\mathbf{x}_i) \cdot h(\mathbf{x}_j) = K\left(\mathbf{x}_i, \mathbf{x}_j\right). \quad (7)$$

Then the output function of ELM classifier (6) can be written compactly as

$$\mathbf{f}(\mathbf{x}) = \begin{bmatrix} K(\mathbf{x}, \mathbf{x}_1) \\ \vdots \\ K(\mathbf{x}, \mathbf{x}_N) \end{bmatrix}^T \left(\frac{\mathbf{I}}{C} + \mathbf{\Omega}_{\text{ELM}}\right)^{-1} \mathbf{T}. \quad (8)$$

In this specific case, similar to SVM, LS-SVM, and PSVM, the 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 dimensionality L of the feature space (the number of hidden nodes) need not be given either.

In another case that the size of the training dataset is much larger than the dimensionality of the feature space, $N \gg L$, an alternative solution is given for simpler matrix calculation. In this case, the output function of ELM classifier is

$$\mathbf{f}(\mathbf{x}) = \mathbf{h}(\mathbf{x})\boldsymbol{\beta} = \mathbf{h}(\mathbf{x}) \left(\frac{\mathbf{I}}{C} + \mathbf{H}^T\mathbf{H}\right)^{-1} \mathbf{H}^T\mathbf{T}. \quad (9)$$

In theory, the above two solutions can be used in any size of applications. However, different approaches have different computational cost and their efficiency may vary in different applications. During the implementation of ELM, it is found that the generalization performance of ELM is not very sensitive to the dimensionality of the feature space (L) and good performance can be reached as long as L is large enough [2]. Thus, if the training datasets are very large $N \gg L$, one may prefer to apply solutions (9) in order to reduce computational costs. However, for extremely large and complex data problems, when using solutions (9), the L required is usually a very large number as well. The computational cost will still be very high.

B. PCA Dimension Reduction

PCA [18] is a widely used statistical technique for unsupervised dimension reduction. It converts a set of possibly correlated variables into a set of linearly uncorrelated variables called principal components. The greatest variance by any projection of the data set comes to lie on the first axis (first principal component), and the second greatest variance on the second axis, and so on. The number of chosen principal components may be much less than the number of original variables. PCA can be done by using eigenvalue decomposition of a data covariance matrix or singular value decomposition of a data matrix. For a given input matrix $\mathbf{A} \in R^{m \times n}$, the goal of PCA is to find a matrix $\mathbf{Y} \in R^{m \times n'}$, where $\mathbf{Y}$ is the Karhunen–Loève transform of matrix $\mathbf{A}$: $\mathbf{Y} = \mathbb{KLT}\{\mathbf{A}\}$. The following are the major steps taken when using covariance method [32].

1) Normalize matrix $\mathbf{A}$ along its columns by subtracting the mean of each column to get the mean-subtracted matrix $\mathbf{B}$, where each column of $\mathbf{B}$ satisfies

$$\mathbf{B}_i = \mathbf{A}_i - \frac{1}{m}\sum_{j=1}^{m} \mathrm{A}_{i,j}, \ \ i = 1, \ldots, n. \quad (10)$$

2) Find the covariance matrix: $\mathbf{C} = cov(\mathbf{B})$.
3) Find the eigenvectors $\mathbf{V}$ and eigenvalues $\mathbf{D}$ of the covariance matrix $\mathbf{C}$ from

$$\mathbf{V}^{-1}\mathbf{C}\mathbf{V} = \mathbf{D} \quad (11)$$

where $\mathbf{D}$ is the diagonal matrix of eigenvalues of $\mathbf{C}$.
4) Sort the eigenvalues $\mathbf{D}$ in descending order as well as matching the corresponding eigenvectors. The reordered eigenvectors are now $\tilde{\mathbf{V}}$.
5) Choose the top $n' \leq n$ significant principal components according to their eigenvalues. This can be done by multiplying the top n' rows of $\tilde{\mathbf{V}}$ to the normalized matrix of $\mathbf{A}$. We then can obtain

$$\mathbf{Y} = \mathbf{B}\tilde{\mathbf{V}}\left(1 : n'\right) = \mathbb{KLT}\{\mathbf{A}\}. \quad (12)$$

C. Autoencoder Networks

Autoencoder/auto-associative neural networks are neural networks that encode the input $\mathbf{X} \in \mathbb{R}^d$ in certain representation $\mathbf{H} \in \mathbb{R}^L$ to reconstruct the original input $\mathbf{X}$. That is, the desired output of an autoencoder $\mathbf{Y} \in \mathbb{R}^d$ is set to be the same as the input: $\mathbf{Y} = \mathbf{X}$. The autoencoder networks usually apply back-propagation algorithm for training. A simple one hidden layer autoencoder network diagram is presented in Fig. 1. The number of hidden units L can be smaller or larger than the data dimension d. Both can help us to discover interesting structure about the data (compressed or sparse representation of the input data). If the hidden layer is linear and mean square error criterion is used for training, the autoencoder functions similarly to PCA: the hidden units learn to represent the principal components of the data [20]. If the hidden layer is nonlinear, the autoencoder is capable of capture multimodel aspects of input data distribution [21].

Recently, autoencoders have been working as a key stage in many deep architecture approaches [22]–[24]. In those deep-learning algorithms, autoencoders perform unsupervised pretraining of data, which leads to state-of-the-art performance on many challenging classification and regression problems.

III. S-ELMs

A. Problem With Existing ELM Solution for Very Large and Complex Dataset

The equality constrained optimization method-based ELM provides two solutions for datasets of different sizes. The solution for the very large dataset is given in (9). In [2], for a large dataset, the number of hidden nodes is chosen as 1000. The generalization performance is fairly good compared with SVM and least square SVM. However, this number may not be the optimal number for the best generalization performance of ELM. For example, for the MNIST database of handwritten digits [35], we found that the testing accuracy increases when the number of hidden nodes increases beyond 1000. Due to the physical memory limitation, we can only test up

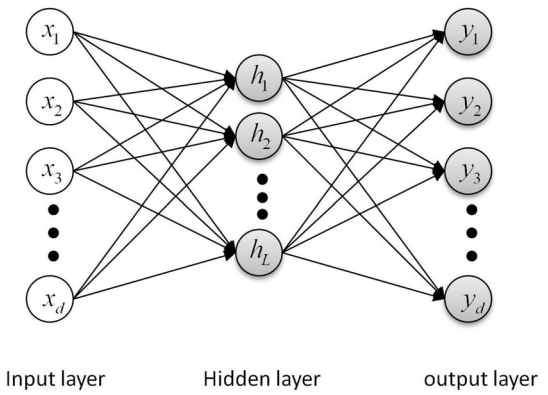

Fig. 1. Network structure of one hidden layer autoencoder.

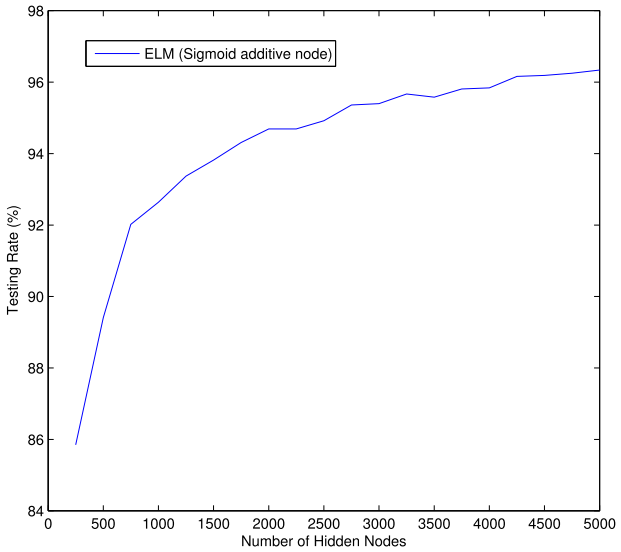

Fig. 2. Relationship between the testing accuracy and the number of hidden nodes for MNIST dataset using the equality optimization method-based ELMs solution for very large dataset.

to 5000 hidden nodes for the MNIST dataset. Fig. 2 shows the relationship between the testing accuracy and the number of hidden nodes. The testing accuracy increases when increasing the number of hidden nodes. It is reasonable to believe that ELM can achieve a better accuracy with more hidden nodes for this dataset. However, due to the physical memory limitation, it is very hard to test more nodes to find out the optimal number of hidden nodes for the best performance. In this paper, we propose a S-ELMs that can help solve this problem. Instead of having one huge ELM that requires many hidden nodes, we break it into multiple small ELMs, each stacking on top of another and forming a S-ELMs learning network. The S-ELMs learning algorithm is an iterative algorithm for which the computational cost is fixed during each iteration. The details of S-ELMs are given in the next subsection.

B. S-ELMs

The S-ELMs learning network consists of multiple small ELMs located at different layers. Each of the small ELMs

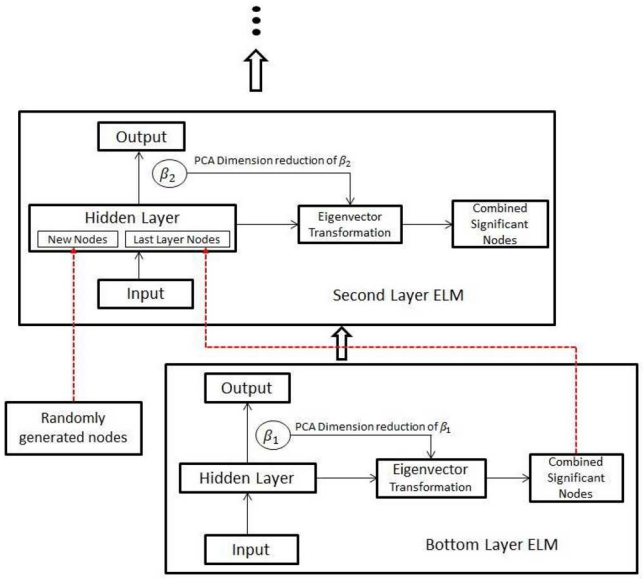

Fig. 3. Block diagram showing the information flow of the first two layers of the S-ELMs learning network.

is serially connected and their hidden layer outputs are propagated to next layer. Here, it is not the full hidden layer outputs that are propagated, but the "reduced" hidden layer outputs are passed to next layer.

Fig. 3 illustrates the operations of the first two bottom layers of S-ELMs. In the bottom layer, firstly we start with a simple ELM network with L hidden nodes. The hidden layer output can be recorded as $\mathbf{H}_1$, and the output weight of the bottom layer ELM $\boldsymbol{\beta}_1$ can be obtained as

$$\boldsymbol{\beta}_1 = \left(\frac{\mathbf{I}}{C} + {\mathbf{H}_1}^T\mathbf{H}_1\right)^{-1} {\mathbf{H}_1}^T\mathbf{T} \tag{13}$$

where $\boldsymbol{\beta}_1$ represents the weights of all connections between the hidden nodes and the output nodes. Some of these connections may not be linearly independent or there exists a basis that we can use to combine them to reduce the number of the hidden nodes required. This can be done by performing a PCA dimension reduction on $\boldsymbol{\beta}_1^T$. The dimension of $\boldsymbol{\beta}_1^T$ is reduced from L to L', where L' can be a user specified value, with $L' \leq L$. In the PCA dimension reduction process, there will be a set of eigenvectors generated based on the orders of their corresponding eigenvalues. The top L' number of eigenvectors are recorded as $\tilde{\mathbf{V}} \in \mathbf{R}^{L\times L'}$. The reduced output weight is then obtained as: ${\boldsymbol{\beta}_1}' = {\boldsymbol{\beta}_1}^T\tilde{\mathbf{V}}$. The original L random hidden nodes can now be replaced by L' combined significant nodes, and the reduced hidden layer output ${\mathbf{H}_1}'$ becomes

$${\mathbf{H}_1}' = \mathbf{H}_1\tilde{\mathbf{V}}. \tag{14}$$

As shown in Fig. 4, the "fat" ELM with L hidden nodes is reduced to a "slim" ELM with L' hidden nodes by performing a PCA dimension reduction on the hidden layer outputs. The reduced hidden layer output ${\mathbf{H}_1}'$ is propagated to the second layer to represent the total hidden nodes information of the first layer's ELM.

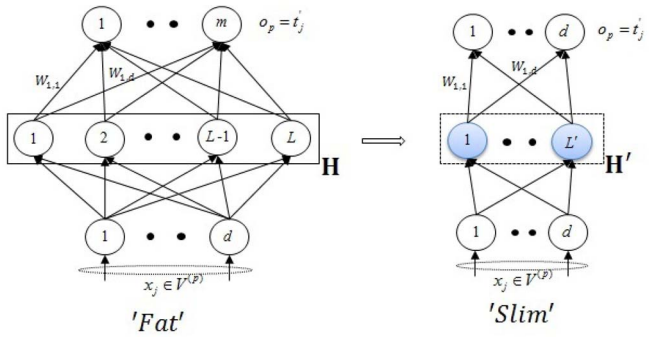

Fig. 4. Fat ELM is reduced to a slim ELM using PCA dimension reduction method.

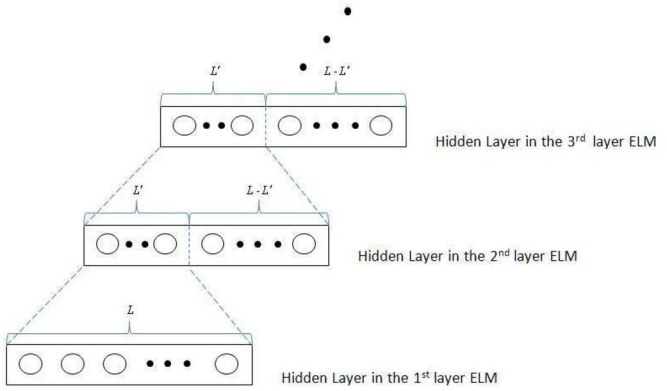

Fig. 5. Lower layer ELM propagates its hidden nodes output to the upper layer ELM and functions as a part of hidden nodes in the upper layer ELM.

As shown in Fig. 5, in the second layer there will be $(L - L')$ new random hidden nodes generated, and the output of these random hidden nodes $\mathbf{H}_{2\text{new}}$ is

$$\mathbf{H}_{2\text{new}} = \begin{bmatrix} \mathbf{h}_{2\text{new}}(\mathbf{x}_1) \\ \vdots \\ \mathbf{h}_{2\text{new}}(\mathbf{x}_N) \end{bmatrix} = \begin{bmatrix} h_1(\mathbf{x}_1) & \cdots & h_{L-L'}(\mathbf{x}_1) \\ \vdots & \vdots & \vdots \\ h_1(\mathbf{x}_N) & \vdots & h_{L-L'}(\mathbf{x}_N) \end{bmatrix}. \quad (15)$$

The total hidden layer output in the second layer of S-ELMs is combined as

$$\mathbf{H}_2 = \left[\mathbf{H}_1{}', \mathbf{H}_{2\text{new}}\right]. \quad (16)$$

In the second layer of Fig. 3, the same procedure as the previous layer can be taken. Firstly, we calculate the output weight $\boldsymbol{\beta}_2$ based on the $\mathbf{H}_2$ obtained in (13) and (16), then perform PCA dimension reduction to ${\boldsymbol{\beta}_2}^T$ and use the eigenvectors obtained during the PCA dimension reduction process to calculate the reduced hidden layer output $\mathbf{H}_2{}'$ and output the corresponding significant nodes to upper layer to function as a part of the hidden nodes and so on until the last layer. In the last layer, there is no need of reducing the number of hidden nodes. Equation (9) is used to calculate the output of the whole network. The above PCA dimension reduction and ELM feature mapping combination process can be described as the following steps:

$$\begin{aligned} \mathbf{H}_1 &\rightarrow \mathbf{H}_1{}' \\ \left[\mathbf{H}_1{}', \mathbf{H}_{2\text{new}}\right] &\rightarrow \mathbf{H}_2{}' \\ \left[\mathbf{H}_2{}', \mathbf{H}_{3\text{new}}\right] &\rightarrow \mathbf{H}_3{}' \\ &\vdots \\ \left[\mathbf{H}_{N-2}{}', \mathbf{H}_{N-1\text{new}}\right] &\rightarrow \mathbf{H}_{N-1}{}' \\ &\left[\mathbf{H}_{N-1}{}', \mathbf{H}_{N\text{new}}\right]. \end{aligned}$$

The algorithm for S-ELMs is summarized in Algorithm 1.

Algorithm 1 Stacked ELMs Learning Network

Given a large training dataset $\Re = \{(\mathbf{x}_i, \mathbf{t}_i) | x_i \in \mathbf{R}^n, \mathbf{t}_i \in \mathbf{R}^m, i = 1, \ldots, N\}$, activation function $g(x)$, number of hidden nodes in each layer L, number of targeted combined nodes L', regularization coefficient C, and number of total layers S:

Step 1) **Apply extreme learning machine algorithm for layer** 1**:**

a) Randomly generate the hidden layer parameters: input weight $\mathbf{w}_i$ and bias b_i, $i = 1, \ldots, L$.
b) Calculate the hidden layer output matrix $\mathbf{H}$.
c) Calculate the output weight $\boldsymbol{\beta}$

$$\boldsymbol{\beta} = \left(\frac{\mathbf{I}}{C} + \mathbf{H}^T\mathbf{H}\right)^{-1} \mathbf{H}^T\mathbf{T}. \quad (17)$$

Step 2) **PCA dimension reduction of** $\boldsymbol{\beta}^T$. Reduce the dimension of $\boldsymbol{\beta}^T$ from L to L'. The reordered eigenvectors produced in this step is recorded as $\tilde{\mathbf{V}} \in \mathbf{R}^{L,L'}$.

Step 3) **Learning step for layer** $2 \rightarrow S$:
for $p = 1$ to $(S - 1)$

a) Randomly generate $(L - L')$ hidden nodes and calculate corresponding $\mathbf{H}_{\text{new}}$.
b) Combined significant nodes $\mathbf{H}'$

$$\mathbf{H}' = \mathbf{H}\tilde{\mathbf{V}}. \quad (18)$$

c) Hidden layer output matrix: $\mathbf{H} = [\mathbf{H}', \mathbf{H}_{\text{new}}]$.
d) repeat step 1(c) and step 2.
Remark: step 2 is not needed in the last layer S.

endfor

Step 4) **Output of the S-ELMs learning network in the final layer**

$$\mathbf{F}_{\text{out}} = \mathbf{H}\boldsymbol{\beta}. \quad (19)$$

IV. ELM AE-S-ELMs

The S-ELMs is designed to tackle the large and complex data problems. Most of such problems are unstructured large data without properly selected features. Thus, an autoencoder could be helpful to learn a certain representation of the data and feed to S-ELMs for training and testing. In this section, we implement an ELM algorithm

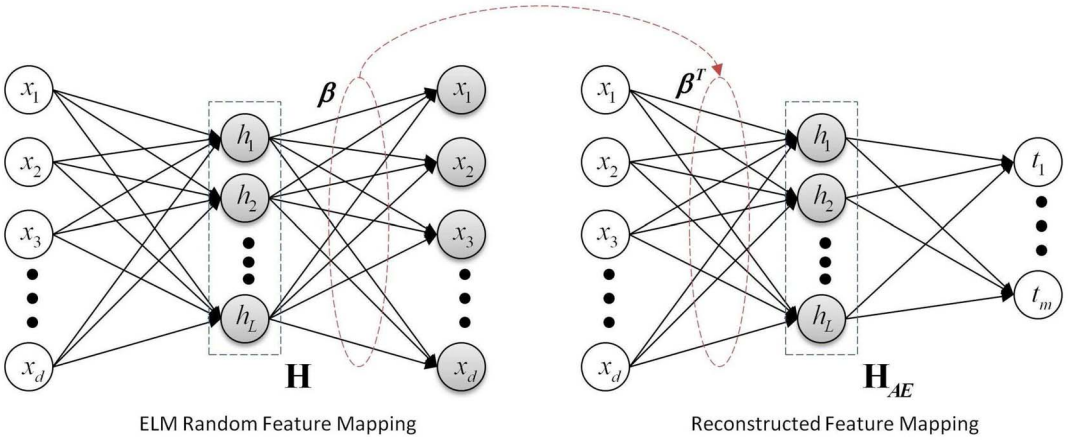

Fig. 6. In ELM autoencoder, ELM is used as the training algorithm for the network, the transpose of output weight $\boldsymbol{\beta}$ (reconstruction matrix) is used as the input weight of a normal ELM.

trained autoencoder in each iteration of S-ELMs algorithm. The ELM autoencoder can serve as unsupervised pretraining of data and help capture interesting input data structure that may be useful for improving model generalization ability. Thus, ELM AE-S-ELMs may achieve better generalization performance.

A. ELM Autoencoder

Unlike conventional autoencoders that usually apply backpropagation algorithm for training to obtain the identity function [33], we use ELM as the training algorithm for an autoencoder. Since the network structure of ELM and one hidden layer autoencoder are exactly the same, ELM algorithm could be immediately implemented to train the autoencoder by setting the desired output equal to its input. We name the ELM algorithm-based autoencoder as ELM autoencoder. Kasun *et al*. [34] also proposed an ELM-based autoencoder. In their approach, the hidden layer parameters are not purely randomly generated but orthogonally randomly generated. In our approach to using ELM autoencoder, the data $\mathbf{X} \in \mathbb{R}^d$ is firstly mapped to ELM feature space $\mathbf{H} \in \mathbb{R}^l$ (ELM space representation of the data). Then the ELM space representation $\mathbf{H}$ is reconstructed to the original input $\mathbf{X}$ through the reconstruction matrix $\boldsymbol{\beta} \in \mathbb{R}^{l,d}$, that is: $\mathbf{X} = \mathbf{H}\boldsymbol{\beta}$. As shown in Fig. 6, the reconstruction matrix $\boldsymbol{\beta}$ that may contain interesting input data distributions is then retained for the usage of data pretraining in a fresh ELM, where instead of randomly generating input weights, $\boldsymbol{\beta}^T$ is used as the input weights. As demonstrated in [34], the output weight $\boldsymbol{\beta}$ learns to represent the input data via singular values and it performs better than manually calculated SVD basis. Thus, $\boldsymbol{\beta}^T$ can be used for unsupervised pretraining of the data, and will likely result in better generalization performance when solving large unstructured data problems.

In an ELM autoencoder with l number of hidden units, the input data with N arbitrary samples $(\mathbf{x}_i, \mathbf{t}_i)$, where $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{id}]^T \in \mathbb{R}^d$ and $\mathbf{t}_i = [t_{i1}, t_{i2}, \ldots, t_{im}]^T \in \mathbb{R}^m$, is reconstructed at the output layer through the function

$$\sum_{i=1}^{l} \beta_i g\left(\mathbf{a}_i \cdot \mathbf{x}_j + b_i\right) = \mathbf{x}_j \qquad j = 1, \ldots, N \tag{20}$$

where $\mathbf{a}_i = [a_{i1}, a_{i2}, \ldots, a_{id}]^T$ is the randomly generated input weight, $\boldsymbol{\beta}_i = [\beta_{i1}, \beta_{i2}, \ldots, \beta_{id}]^T$ is the reconstruction matrix, and $g(\cdot)$ is the activation function of the hidden units. Equation (20) can be written in the matrix format

$$\mathbf{H}\boldsymbol{\beta} = \mathbf{X}. \tag{21}$$

The reconstruction matrix $\boldsymbol{\beta}$ can be simply obtained as: $\boldsymbol{\beta} = \mathbf{H}^{\dagger}\mathbf{X}$. Or similar to the equality constrained optimization method-based ELM [2], we can add a regularization term in the solution, which makes the solution likely to be nonsingular. Then the reconstruction matrix $\boldsymbol{\beta}$ becomes: $\boldsymbol{\beta} = ((\mathbf{I}/C) + \mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{X}$. When we use the transpose of reconstruction matrix of ELM encoder as the input weight of another ELM, the hidden layer output $\mathbf{H}_{\mathrm{AE}}$ can be obtained as

$$\mathbf{H}_{\mathrm{AE}} = \begin{bmatrix} g\left(\boldsymbol{\beta}_1^T \cdot \mathbf{x}_1 + b_1\right) & \cdots & g\left(\boldsymbol{\beta}_l^T \cdot \mathbf{x}_1 + b_l\right) \\ \vdots & \vdots & \vdots \\ g\left(\boldsymbol{\beta}_1^T \cdot \mathbf{x}_N + b_1\right) & \vdots & g\left(\boldsymbol{\beta}_l^T \cdot \mathbf{x}_N + b_l\right) \end{bmatrix}_{N \times l} \tag{22}$$

where $\boldsymbol{\beta} = ((\mathbf{I}/C) + \mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{X}$.

Remark: If we set the total number of hidden nodes in one layer ELM as L and number of targeted combined nodes as L', for the first layer of ELM AE-S-ELMs, we choose $l = L$; for layers $2 \rightarrow S$, we choose $l = L - L'$.

B. Implementation of ELM Autoencoder in S-ELMs

The ELM autoencoder can be implemented at each layer of the S-ELMs. In the S-ELMs, the first layer ELMs hidden layer contains all random nodes and from layers $2 \rightarrow S$, the hidden layer output of each ELM consists of output from the previous layer and the new random nodes. Here, we can replace the random nodes with the "pretrained" sigmoid

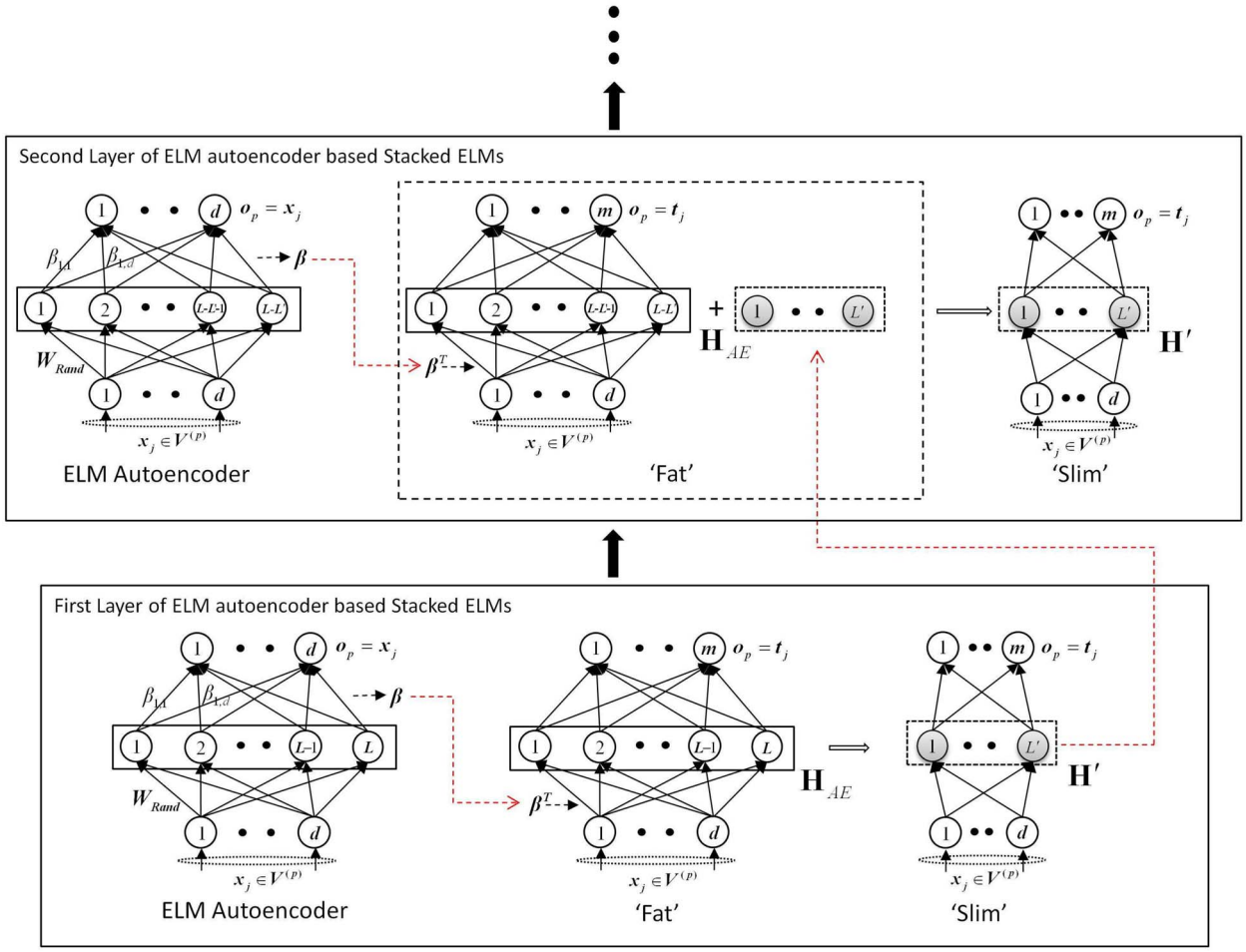

Fig. 7. Network diagram of ELM AE-S-ELMs.

nodes. The "pretrained" sigmoid nodes are obtained from ELM autoencoder.

Fig. 7 explains the network structure of ELM AE-S-ELMs. As shown in Fig. 7, ELM autoencoder is used in every layer and helps generate the reconstruction matrix $\boldsymbol{\beta}$. The transpose of $\boldsymbol{\beta}$ is used as the input weight to generate $\mathbf{H}_{\text{AE}}$. In the first layer, $\mathbf{H}_{\text{AE}} \in \mathbb{R}^{N\times L}$ is reduced to $\mathbf{H}' \in \mathbb{R}^{N\times L'}$ using PCA dimension reduction method. $\mathbf{H}'$ is propagated to the second layer and functions as part of the hidden layer matrix. In the second layer, ELM autoencoder with $L - L'$ hidden nodes is created, then the hidden layer output in second layer $\mathbf{H}_{\text{AE}} \in \mathbb{R}^{N\times(L-L')}$ is generated according to (22). $\mathbf{H}_{\text{AE}}$ is then combined with the PCA reduced hidden layer output $\mathbf{H}'$ from previous layer to form the whole hidden layer matrix. $\mathbf{H}_{\text{AE}}$ plays the same role as $\mathbf{H}_{\text{new}}$ in (16). Similarly to the S-ELMs, such hidden layer output propagation process will repeat until the last layer. The detailed algorithm of ELM AE-S-ELMs is shown in Algorithm 2.

V. Discussions

A. Generalization Capability

The proposed S-ELMs learning network breaks the large single hidden layer ELM network to a total number of U smaller ELM networks. From the second layer onward, there are $(L - L')$ new randomly generated nodes in each layer. Suppose, during the PCA dimension reduction of output weight $\boldsymbol{\beta}$ and eigenvector transformation of hidden layer output $\mathbf{H}$ step, it retains 100% of the original information. The S-ELMs learning network can simulate a large single hidden layer ELM network with a total number of $\tilde{L}$ hidden nodes

$$\begin{aligned}\tilde{L} &= L + \left(L - L'\right) \times (S-1) \\ &= L \times (1 + \sigma(S-1))\end{aligned} \tag{30}$$

where $\sigma = L - L'$ is between 0 and 1 and indicates the degree of new information added during each iteration. However, due to the information loss during the PCA dimension reduction step in every layer, the actual number of total hidden nodes that the S-ELMs learning network can simulate is smaller than $\tilde{L}$. Suppose after one PCA dimension reduction process, the remaining nodes contain a portion of η of original information, where η is a number between 0 and 1. Equation (30) becomes

$$\tilde{L} = \eta^{S-1}L + \eta^{S-2}\left(L - L'\right) + \cdots + \left(L - L'\right). \tag{31}$$

Equation (31) indicates that the more number of layers, the more information will be lost for the lower layers. Fig. 8 shows an example on the relationship between the number of hidden nodes in each layer L', the total number of layer S and the total

Algorithm 2 ELM AE-S-ELMs Algorithm

Given a large training dataset $\Re = \{(\mathbf{x}_i, \mathbf{t}_i)|x_i \in \mathbb{R}^d, \mathbf{t}_i \in \mathbb{R}^m, i = 1, \ldots, N\}$, activation function $g(\cdot)$, number of hidden nodes in each layer L, number of targeted combined nodes L', regularization coefficient C and number of total layers S:

Step 1) **ELM autoencoder on layer 1:**

a) Randomly generate the hidden layer parameters: input weight $\mathbf{a}_i$ and bias b_i, $i = 1, \ldots, L$.

b) Calculate hidden layer output matrix $\mathbf{H}$.

c) Calculate reconstruction matrix $\boldsymbol{\beta}_{rc}$

$$\boldsymbol{\beta}_{rc} = \left(\frac{\mathbf{I}}{C} + \mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{X}. \tag{23}$$

d) Use the reconstruction matrix as input weight of another ELM to generate hidden layer output matrix $\mathbf{H}$

$$\mathbf{H} = \begin{bmatrix} g\left(\boldsymbol{\beta}_{rc1} \cdot \mathbf{x}_1 + b_1\right) & \cdots & g\left(\boldsymbol{\beta}_{rcL} \cdot \mathbf{x}_1 + b_L\right) \\ \vdots & \vdots & \vdots \\ g\left(\boldsymbol{\beta}_{rc1} \cdot \mathbf{x}_N + b_1\right) & \vdots & g\left(\boldsymbol{\beta}_{rcL} \cdot \mathbf{x}_N + b_L\right) \end{bmatrix}_{N \times L}. \tag{24}$$

e) Calculate output weight of ELM $\boldsymbol{\beta}$

$$\boldsymbol{\beta} = \left(\frac{\mathbf{I}}{C} + \mathbf{H}^T\mathbf{H}\right)^{-1}\mathbf{H}^T\mathbf{T}. \tag{25}$$

Step 2) **PCA dimension reduction of $\boldsymbol{\beta}^T$.** Reduce the dimension of $\boldsymbol{\beta}^T$ from L to L'. The reordered eigenvectors produced in this step is recorded as $\tilde{\mathbf{V}} \in \mathbf{R}^{L,L'}$.

Step 3) **Learning step for layer $2 \rightarrow S$:**
for $p = 1$ to $(S - 1)$

a) Randomly generate $(L - L')$ hidden nodes and calculate corresponding $\mathbf{H}_{\text{new}}$.

b) Calculate reconstruction matrix $\boldsymbol{\beta}_{rc}$

$$\boldsymbol{\beta}_{rc} = \left(\frac{\mathbf{I}}{C} + \mathbf{H}_{\text{new}}^T\mathbf{H}_{\text{new}}\right)^{-1}\mathbf{H}_{\text{new}}^T\mathbf{X}. \tag{26}$$

c) Calculate the additional hidden layer output $\mathbf{H}_{\text{AE}}$ using the reconstruction matrix

$$\mathbf{H}_{\text{AE}} = \begin{bmatrix} g\left(\boldsymbol{\beta}_{rc1} \cdot \mathbf{x}_1 + b_1\right) & \cdots & g\left(\boldsymbol{\beta}_{rcl} \cdot \mathbf{x}_1 + b_l\right) \\ \vdots & \vdots & \vdots \\ g\left(\boldsymbol{\beta}_{rc1} \cdot \mathbf{x}_N + b_1\right) & \vdots & g\left(\boldsymbol{\beta}_{rcl} \cdot \mathbf{x}_N + b_l\right) \end{bmatrix}_{N \times l} \tag{27}$$

where $l = L - L'$

d) Calculate the combined significant nodes $\mathbf{H}'$

$$\mathbf{H}' = \mathbf{H}\tilde{\mathbf{V}}. \tag{28}$$

e) Hidden layer output matrix: $\mathbf{H} = [\mathbf{H}', \mathbf{H}_{\text{AE}}]$.

f) repeat step 1(e) and step 2.

endfor

Step 4) **Output of the S-ELMs learning network in the final layer**

$$\mathbf{F}_{\text{out}} = \mathbf{H}\boldsymbol{\beta}. \tag{29}$$

number of nodes $\tilde{L}$ on the MNIST dataset. We can observe that in order to obtain the same testing accuracy as a single ELM with 3000 hidden nodes, the S-ELMs requires different number of layers for different number of hidden nodes in each layer. When the number of hidden nodes L in each layer is 200, 500, and 1000, and the targeted combined nodes L' is chosen as $10\% \times L$, it requires about 27.46, 10.58, and 4.62 layers, respectively, from Fig. 8. However, from (30), we can calculate the ideal number of total layers as 16.56, 6.56, and 3.22, so the scale-up factor (measures the information loss, the higher the more information is lost) is: $27.46/6.56 = 1.66$, $10.58/6.56 = 1.61$, and $4.62/3.22 = 1.43$. In other words, compared with the ideal number of layers, a smaller number of hidden nodes in each layer requires a larger number of layers to cover the information loss in each layer. Hence, it is advisable to choose a not too small number of hidden nodes L for each layer. Yet, L cannot be so large that is beyond the computing machine's memory capability.

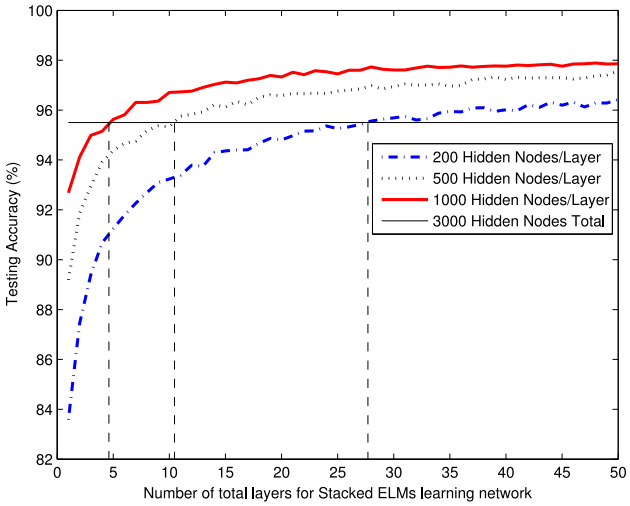

Fig. 8. Testing accuracy of S-ELMs learning network when the number of hidden nodes is chosen as 200, 500, and 1000, respectively. The horizontal line shows the testing accuracy of a single ELM with 3000 hidden nodes.

For the ELM AE-S-ELMs, however, it will be slightly different in determining the size of each small ELMs. Since we implement the ELM autoencoder during each iteration, the size of ELM autoencoder reflects different types of feature mappings (e.g., sparse, equal dimension, or compressed) which may have different impacts on the generalization performance. Hence, the complexity of the small ELM on each layer need to be properly determined for different applications.

B. Computational Complexity and Memory Requirement

In a single ELM network, when solving a large-scale dataset application, the number of hidden nodes $\tilde{L}$ required is usually a large number so that ELM can map the data to a high enough dimensional space and achieve a good generalization performance. In the solution given by (9), matrix calculation involves multiplication $\mathbf{H}^T\mathbf{H}$ of size $\tilde{L} \times \tilde{L}$ and inversion $((\mathbf{I}/C) + \mathbf{H}^T\mathbf{H})^{-1}$ of size $\tilde{L} \times \tilde{L}$ as well. When $\tilde{L}$ becomes larger, the computational cost increases dramatically. In the S-ELMs learning network, matrix in each layer is of much smaller size $L \times L$. In the previous subsection, we have discussed the relationship between $\tilde{L}$ and L. In a rough estimation, $\tilde{L} = \sigma S \times L$, and $\sigma S \gg 1$. In this sense, the computational complexity of single ELM network can be estimated as $\sigma^2 S^2 \times L^2$ while the S-ELMs learning network's computational

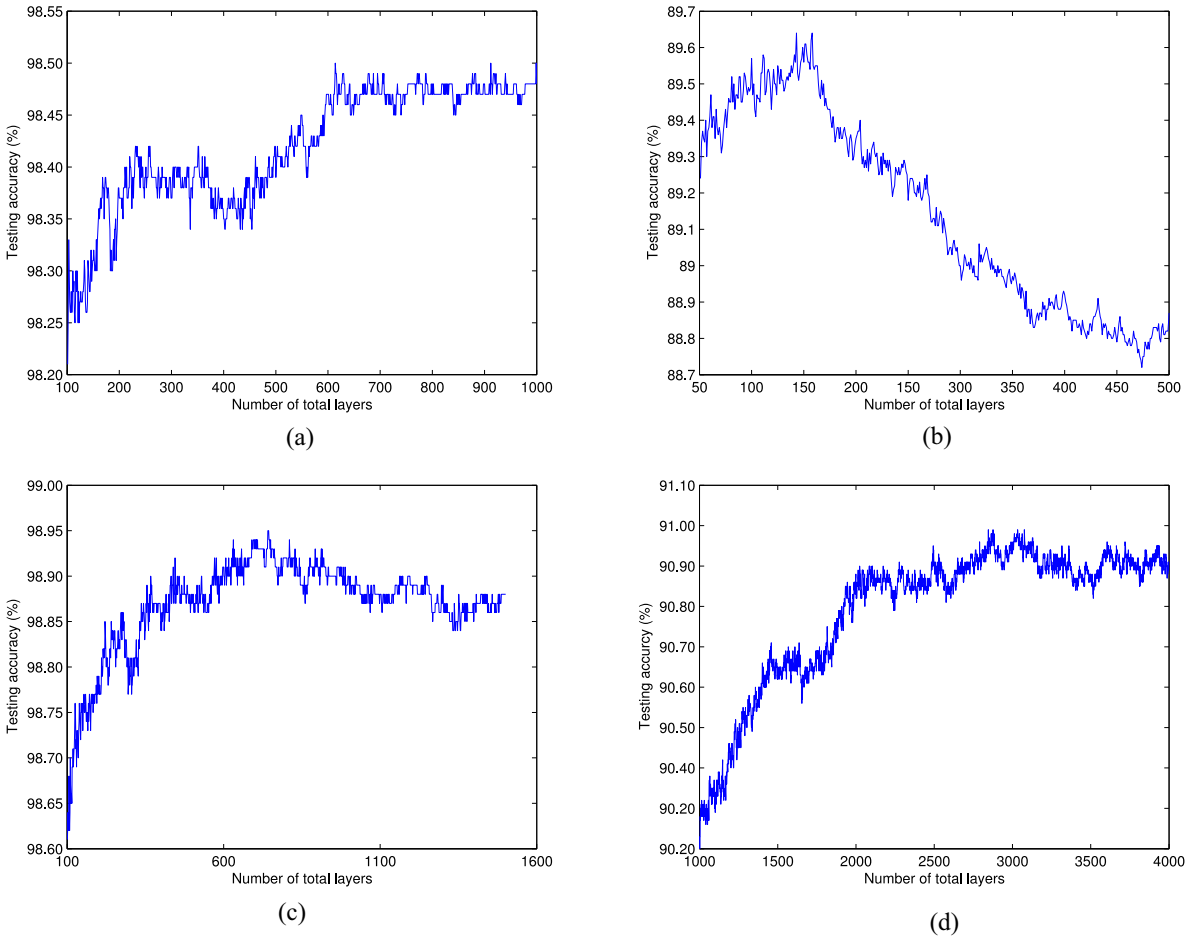

Fig. 9. Relationship between testing accuracy and number of total layers (iterations). (a) S-ELMs on MNIST. (b) S-ELMs on OCR. (c) ELM-AE-based S-ELMs on MNIST. (d) ELM-AE-based S-ELMs on OCR.

complexity is only $S \times L^2$. In our implementation, S is usually much less than $\sigma^2 S^2$.

If ELM autoencoder is employed as an unsupervised data pretraining step for both single large ELM network and S-ELMs, according to (23) and (25), the computational cost can be estimated as $2 \times (\alpha^2 S^2 \times L^2)$ and $S \times (L^2 + \sigma^2 \times L^2)$, respectively. The same conclusion can be made here: the ELM AE-S-ELMs is much less computationally complex than the ELM autoencoder-based single ELM.

VI. Performance Verification

A. Datasets and Simulation Environment Description

In this section, we tested the performance of proposed S-ELMs and ELM AE-S-ELMs on four large datasets: MNIST, OCR Letters, NORB, and USPS data. The MNIST database of handwritten digits [35] is a popular database for many researchers to try different machine-learning techniques. It consists of 60 000 training samples, 10 000 testing samples, and 784 features for each sample. Many methods have been tested on this dataset including SVMs, neural networks and many other classification algorithms [36]. OCR Letters dataset [37] is a public available dataset that corresponds to the handwritten words recognition problem. It contains 42 152 training samples and 10 000 testing samples with 128 features. The NORB dataset is for experiments in 3-D object recognition from shape [38]. It contains 23 000 training samples and 23 000 testing samples with 2048 features. The USPS dataset refers to numeric data obtained from the scanning of handwritten digits from envelopes by the U.S. Postal Service [39]. It contains 7291 training instances and 2007 testing instances with 256 features. The simulations are carried out in the MATLAB 2013a environment running on a Intel Xeon E5-2650 2.00 GHz 256 G RAM computer.

B. Effects of Parameters on the Generalization Performance

There are four parameters the user can adjust to achieve the best testing accuracy of the S-ELMs and ELM AE-S-ELMs: the number of hidden nodes in each layer L, the number of combined nodes L', the total number of layers S, and the regularization parameter C. The hidden nodes activation function used in our simulation is the Sigmoid function.

As discussed in Section V-A, from (30) and (31), we can see that L, L', and S are correlated parameters for S-ELMs. They determine the complexity of the whole network. As shown in Fig. 8, different numbers of L would have effects on the total number of layers S required to simulate the same size ELM network. In our simulation, we firstly tuned the best combination of L and L' in approximating a certain size ELM

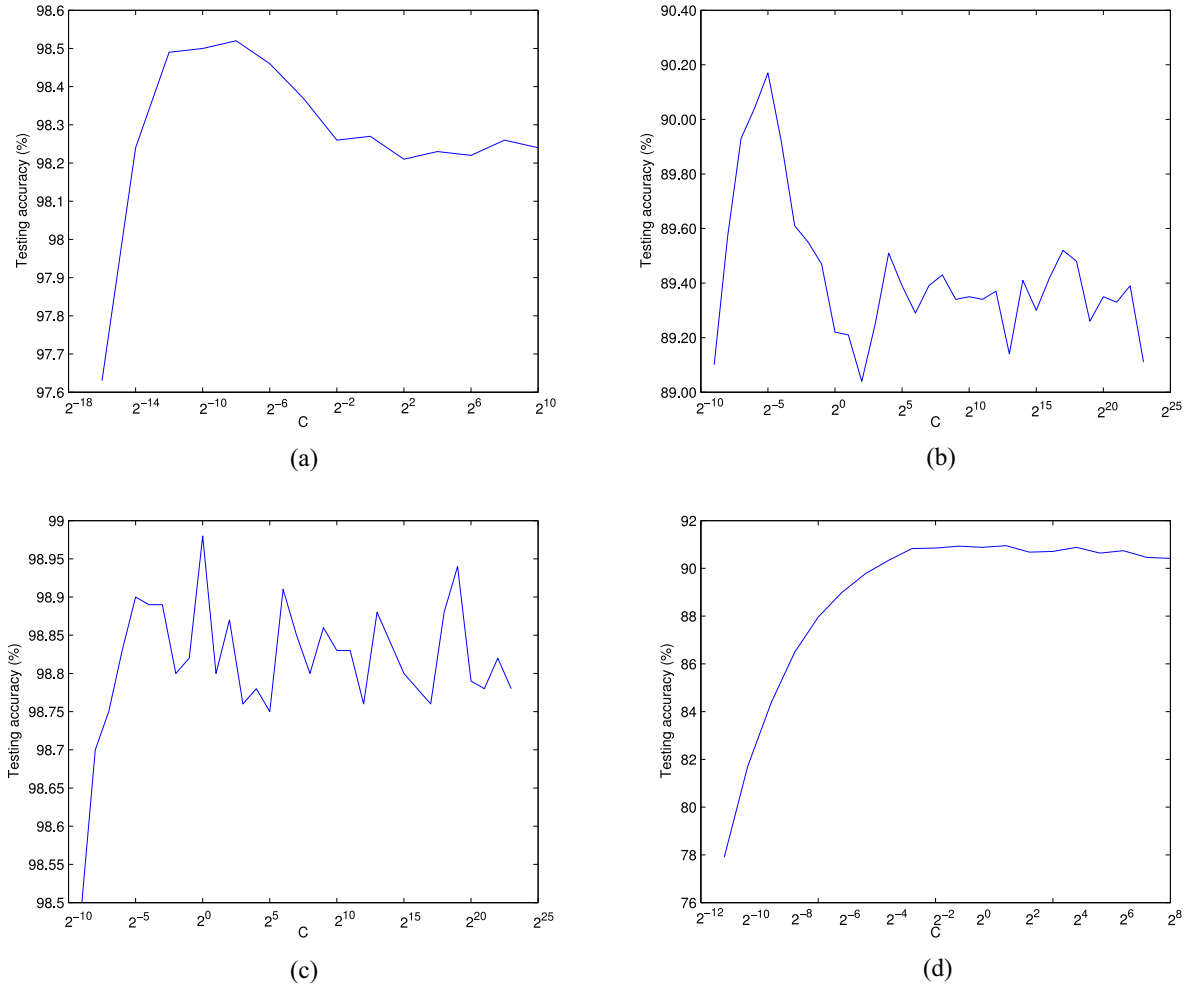

Fig. 10. Relationship between testing accuracy and the regularization parameter C. (a) S-ELMs on MNIST. (b) S-ELMs on OCR. (c) ELM-AE-based S-ELMs on MNIST. (d) ELM-AE-based S-ELMs on OCR.

network, then determined S to obtain the best approximation error. As observed from the two examples on MNIST and OCR data shown in Fig. 9, the testing accuracy will go up when the total number of layers increases until some saturate point. It will stay the same or start to drop once the network becomes so complex that it will overfit the problem.

After the network complexity was determined, we tuned the regularization parameter C subsequently. One will be able to find the best C by conducting a search from $\{2^{-20}, 2^{-19}, \ldots, 2^{19}, 2^{20}\}$. Fig. 10 shows the relationship between the testing accuracy and the regularization factor C.

C. Generalization Performance

The performance of both S-ELMs and ELM AE-S-ELMs on the four datasets are shown in Table I. We also did simulations on single ELM, SVM, and DBN to compare the generalization capability and effectiveness of different algorithms.

1) S-ELMs Versus Single ELM Versus AE-S-ELMs: To check how well the S-ELMs can simulate a single large ELM, we tested the performance of the single ELM on a computer with 256 G memory in order to select the optimal number of hidden nodes. The large number of hidden nodes used in the single ELM will cost heavily during computation. Hence, it requires a large memory space for computation. Simulation results are shown in Table I. The maximum number of hidden nodes per layer shown in Table I indicates the minimum working memory space required to cope with such network. The accuracy of both single ELM and S-ELMs is the average of 20 runs. As we can observe from Table I, S-ELMs can always obtain better testing accuracy than a single ELM. Though the maximum number of hidden nodes per layer for S-ELMs is much smaller than single ELM, the training time taken for S-ELMs is similar to or higher than single ELM. This is because stack ELMs is an iterative learning algorithm, and the accuracy increases when the number of total layer increase until it reaches a saturate point. As shown in Table II, the S-ELMs requires hundreds of iterations in computation. The ELM AE-S-ELMs requires even smaller network size compared with S-ELMs, and the testing accuracy is higher than S-ELMs with similar or more training time.

2) S-ELMs/AE-S-ELMs Versus SVM: We tested SVMs performance on the four datasets using LibSVM [41]. During the simulation of LibSVM, we did a threefold cross-validation with grid search for the optimal cost parameter C and kernel

TABLE I
PERFORMANCE COMPARISON OF SINGLE ELM, S-ELMS, ELM AE-S-ELMS, SVM, AND DBN

Datasets	Training Methods	Training Time(S)	Testing Rate(%)	Testing Dev(%)	Max # of hidden units per layer
MNIST	ELM	4127	98.04	0.22	60000
	S-ELMs	4621	98.51	0.08	1000
	AE-S-ELMs	4347	**98.89**	0.06	**500**
	SVM	**2433**	98.51	0	18746
	DBN	38448	98.87	-	2000
OCR Letters	ELM	**172.83**	87.36	0.18	15000
	S-ELMs	835.62	89.83	0.08	1000
	AE-S-ELMs	4833	**90.96**	0.03	**400**
	SVM	612.7	90.07	0	29146
	DBN	48636	90.59	-	2000
NORB	ELM	**40.38**	90.01	0.23	5000
	S-ELMs	1078	90.31	0.17	**1000**
	AE-S-ELMs	2799	**91.24**	0.13	**1000**
	SVM	678.3	90.04	0	4766
	DBN	72h	89.60	-	4000
USPS	ELM	**5.17**	96.55	0.31	3000
	S-ELMs	151.02	98.34	0.27	**300**
	AE-S-ELMs	53.6	**98.91**	0.11	**300**
	SVM	21.86	98.75	0	2744
	DBN	168	98.01	-	500

TABLE II
PARAMETERS OF SINGLE ELM, S-ELMS, ELM AE-S-ELMS, AND SVM ON MNIST AND OCR LETTERS DATASET

Datasets	ELM		S-ELMs				AE-S-ELMs				SVM	
	L	C	L	L'	C	S	L	L'	C	S	σ	C
MNIST	60000	$2^{(-6)}$	1000	100	$2^{(-8)}$	650	500	50	2^2	700	0.01	10
OCR Letters	15000	2^0	1000	100	$2^{(-5)}$	160	400	280	2^0	3000	0.01	2
NORB	5000	$2^{(-6)}$	1000	100	2^6	120	1000	100	2^6	300	0.1	10
USPS	3000	$2^{(-4)}$	300	60	$2^{(-3)}$	150	300	60	$2^{(-3)}$	80	10	0.01

parameter γ from 15 different values: {0.0001, 0.001, 0.01, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100, 1000, and 10 000}. From simulation results, we can observe that both the training time and testing accuracy for S-ELMs and SVM are at the similar level. However, the maximum number of hidden nodes for SVM (number of support vectors) is much larger than S-ELMs. Hence, SVM requires a larger memory working space for computation. The ELM AE-S-ELMs can achieve much better testing accuracy than SVM and with much smaller network size as well. Both S-ELMs and ELM AE-S-ELMs can address the potential memory issue faced by SVM on large dataset problems.

3) AE-S-ELMs Versus DBN: We also tested the performance of DBN on these four datasets. The DBN codes are taken from [42]. Though in [24], the reported testing accuracy for MNIST data using the 784-500-500-2000-10 network is 98.8%, we managed to obtain a better accuracy at 98.87% as shown in Table I. For the OCR letter data, the reported testing accuracy is 90.32 with the 128-2000-2000-2000-26 network in [43], while we managed to obtain the testing accuracy at 90.59% with the 128-1000-1000-2000-26 network. The network used for NORB and USPS are 2048-1000-1000-2000-5 and 256-200-200-500-10, respectively. Similarly to DBN, the ELM AE-S-ELMs employs the unsupervised data pretraining step during each iteration of S-ELMs. The simulation results in Table I show that AE-S-ELMs can achieve similar or better testing accuracy than DBN. The maximum number of hidden nodes per layer in AE-S-ELMs is smaller than DBN as well. The greater advantage of AE-S-ELMs compared with DBN is its training speed. Thus with better accuracy, smaller network size and faster training speed, AE-S-ELMs would have greater potential in solving many large-scale unstructured raw data problems.

VII. CONCLUSION

In this paper, we have proposed an iterative learning algorithm called S-ELMs. The network consists of multiple ELMs with a small number of hidden nodes in each layer to substitute a single ELM with a large number of hidden nodes. It is especially useful to solve very large complex dataset problems because single ELM usually incurs difficulty in computation when its number of hidden nodes is very large. The proposed S-ELMs employs the PCA dimension reduction method to reduce the number of hidden nodes in each layer. The retained nodes (combined significant nodes) from the previous layer are carried forward to the next layer to merge with new random nodes in that layer and so on. The users can determine the number of layers of the S-ELMs to make it deep enough for the given machine learning tasks. To enhance the generalization performance, especially on the large unstructured raw datasets, we added the ELM autoencoder into each layer

of S-ELMs. The ELM autoencoder functions as the unsupervised data pretraining step for each small ELM in the S-ELMs network.

Simulation results have shown that: 1) compared with original ELM, S-ELMs can achieve better testing accuracy with much smaller network size, and ELM AE-S-ELMs can increase the testing accuracy further; 2) S-ELMs has similar generalization performance compared with the popular used LibSVM, but with much smaller network size; and 3) compared with DBN, the ELM AE-S-ELMs can achieve better testing accuracy with smaller network size and much faster training speed. The proposed S-ELMs and ELM AE-S-ELMs have a great potential to solve many large-scale data problems without bearing the physical memory limitation. In the future, we could further study the multiple hidden layer autoencoders, and implement such models in each iteration of S-ELMs. For more complicated big data problems, such implementation may help improve the testing accuracy further.

REFERENCES

[1] G.-B. Huang. (2010). *Extreme Learning Machines (ELM)*. School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore. [Online]. Available: http://www.extreme-learning-machines.org/

[2] 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.

[3] (2013). *Directory Page for Top500 Lists. Result for Each List Since June 1993*. [Online]. Available: http://www.top500.org

[4] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *Proc. 19th ACM Symp. Operating Syst. Principles*, New York, NY, USA, 2003, pp. 29–43.

[5] D. Borthakur, *The Hadoop Distributed File System: Architecture and Design*. Dover, DE, USA: The Apache Software Foundation, 2007.

[6] J. Dean and S. Ghemawat, "MapReduce: Simplified data processing on large clusters," in *Proc. 6th Symp. Operating Syst. Design Implementation*, San Francisco, CA, USA, 2004, p. 10.

[7] (2007). *NVidia CUDA Zone*. [Online]. Available: http://www.nvidia.com/object/cuda_home_ new.html

[8] (2000). *What is GPU Computing*. [Online]. Available: http://www.nvidia.com/object/what–is–gpu–computing.html

[9] Y.-J. Lee and O. L. Mangasarian, "RSVM: Reduced support vector machines," in *Proc. 1st SIAM Int. Conf. Data Mining*, Chicago, IL, USA, Apr. 2001, p. 7.

[10] A. J. Smola and B. Schölkopf, "Sparse greedy matrix approximation for machine learning," in *Proc. 7th Int. Conf. Mach. Learn.*, San Francisco, CA, USA, 2000, pp. 911–918.

[11] C. Williams and M. Seeger, "Using the Nyström method to speed up kernel machines," in *Advances in Neural Information Processing Systems 13*. Cambridge, MA, USA: MIT Press, 2001, pp. 682–688.

[12] I. W. Tsang, J. T. Kwok, P.-M. Cheung, and N. Cristianini, "Core vector machines: Fast SVM training on very large data sets," *J. Mach. Learn. Res.*, vol. 6, pp. 363–392, Apr. 2005.

[13] V. Vapnik, *Statistical Learning Theory*. New York, NY, USA: Wiley, 1998.

[14] J. C. Platt, "Fast training of support vector machines using sequential minimal optimization," in *Advances in Kernel Methods C Support Vector Learning*. Cambridge, MA, USA: MIT Press, 1999, pp. 185–208.

[15] J. C. Platt, "A resource-allocating network for function interpolation," *Neural Comput.*, vol. 3, no. 2, pp. 213–225, 1991.

[16] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate on-line sequential learning algorithm for feedforward networks," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1411–1423, Nov. 2006.

[17] G. Fung and O. L. Mangasarian, "Incremental support vector machine classification," in *Proc. 7th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2001, pp. 77–86.

[18] I. T. Jolliffe, *Principal Component Analysis*. New York, NY, USA: Springer-Verlag, 1986.

[19] L. Deng and D. Yu, "Deep convex net: A scalable architecture for speech pattern classification," in *Proc. INTERSPEECH*, 2011, pp. 2285–2288.

[20] H. Bourlard and Y. Kamp, "Auto-association by multilayer perceptrons and singular value decopmposition," *Biol. Cybern.*, vol. 59, nos. 4–5, pp. 291–294, 1988.

[21] N. Japkowicz, S. J. Hanson, and M. A. Bluck, "Nonlinear autoassociation is not equivalent to PCA," *Neural Comput.*, vol. 12, no. 3, pp. 531–545, 2000.

[22] D. Erhan *et al.*, "Why does unsupervised pre-training help deep learning?" *J. Mach. Learn. Res.*, vol. 11, pp. 625–660, Feb. 2010.

[23] G. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, 2006.

[24] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

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

[26] 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.

[27] 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.

[28] G.-B. Huang and L. Chen, "Convex incremental extreme learning machine," *Neurocomputing*, vol. 70, nos. 16–18, pp. 3056–3062, 2007.

[29] G.-B. Huang and L. Chen, "Enhanced random search based incremental extreme learning machine," *Neurocomputing*, vol. 71, nos. 16–18, pp. 3460–3468, 2008.

[30] D. Serre, *Matrices: Theory and Applications*. New York, NY, USA: Springer-Verlag, 2002.

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

[32] L. I. Smith, *A Tutorial on Principal Components Analysis*. Ithaca, NY, USA: Cornell Univ. Press, 2002.

[33] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Parallel Distributed Processing*, vol. 1. Cambridge, MA, USA: MIT Press, 1986, pp. 318–362.

[34] L. L. C. Kasun, H. Zhou, G.-B. Huang, and C. M. Vong, "Representational learning with extreme learning machine," *IEEE Intell. Syst.*, vol. 28, no. 6, pp. 31–34, Dec. 2013.

[35] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998.

[36] Y. LeCun and C. Cortes. (1998). *THE MNIST DATABASE of Handwritten Digits*. The Courant Institute of Mathematical Sciences, New York University, NY, USA. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[37] R. Kassel. (2013, Jan.). *OCR Dataset*. MIT Spoken Language Systems Group, Cambridge, MA, USA. [Online]. Available: http://ai.stanford.edu/~btaskar/ocr/

[38] Y. LeCun, F. J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," in *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recog.*, Washington, DC, USA, Jun./Jul. 2004, pp. II:97–II:104.

[39] J. J. Hull, "A database for handwritten text recognition research," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 16, no. 5, pp. 550–554, May 1994.

[40] N.-Y. Liang, G.-B. Huang, P. Saratchandran, and N. Sundararajan, "A fast and accurate on-line sequential learning algorithm for feedforward networks," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1411–1423, Nov. 2006.

[41] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," in *Proc. ACM Trans. Intell. Syst. Technol.*, 2011, pp. 27:1–27:27.

[42] R. Salakhutdinov and G. Hinton. (2013, Jan.). *Training a Deep Autoencoder or a Classifier on MNIST Digits*. Department of Computer Science, University of Toronto, ON, Canada. [Online]. Available: http://www.cs.toronto.edu/∼hinton/ MatlabForSciencePaper.html

[43] R. Salakhutdinov and H. Larochelle, "Efficient learning of deep Boltzmann machines," in *Proc. 13th Int. Conf. Artif. Intell. Statist.*, vol. 9. Sardinia, Italy, May 2010, pp. 693–700.

Hongming Zhou received the B.Eng. and Ph.D. degrees from Nanyang Technological University, Singapore, in 2009 and 2014, respectively.

He is currently a Research Fellow with the School of Electrical and Electronic Engineering, Nanyang Technological University. His current research interests include classification and regression algorithms such as ELMs, neural networks, and support vector machines as well as their applications including HVAC system control applications, biometrics identification, image retrieval, and financial index prediction.

Guang-Bin Huang received the B.Sc. degree in applied mathematics and the M.Eng. degree in computer engineering from Northeastern University, Shenyang, China, in 1991 and 1994, respectively, and the Ph.D. degree in electrical engineering from Nanyang Technological University, Singapore, in 1999.

From 1998 to 2001, he was a Research Fellow at the Singapore Institute of Manufacturing Technology, Nanyang, Singapore (formerly known as the Gintic Institute of Manufacturing Technology), where he has led/implemented several key industrial projects, including the Chief Designer and Technical Leader of Singapore Changi Airport Cargo Terminal Upgrading Project. From 2001, he was an Assistant Professor and an Associate Professor at the School of Electrical and Electronic Engineering, Nanyang Technological University. His current research interests include machine learning, computational intelligence, and ELMs.

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

Zhiping Lin received the B.Eng. degree in control engineering from the South China Institute of Technology, Canton, China, in 1982, and the Ph.D. degree in information engineering from the University of Cambridge, Cambridge, U.K., in 1987.

He was with the University of Calgary, Calgary, AB, Canada, from 1987 to 1988, Shantou University, Shantou, China, from 1988 to 1993, and DSO National Laboratories, Singapore, from 1993 to 1999. Since 1999, he has been an Associate Professor at Nanyang Technological University (NTU), Singapore. He is also the Program Director of Distributed Healthcare, Valens Centre of Excellence, NTU. His current research interests include multidimensional systems and signal processing, statistical and biomedical signal processing, and machine learning.

Dr. Lin was the recipient of the 2007 Young Author Best Paper Award from the IEEE Signal Processing Society. He is currently an Editor-in-Chief of *Multidimensional Systems and Signal Processing*. He was an Editorial Board Member from 1993 to 2004 and a Co-Editor from 2005 to 2010. He is also an Associate Editor of the journal of the Franklin Institute and the Lead Guest Editor of a special issue in Mathematical Problems in Engineering. He was an Associate Editor of *Circuits, Systems and Signal Processing* from 2000 to 2007 and the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—II from 2010 to 2011. He also served as a Reviewer for *Mathematical Reviews* from 2011 to 2013. He is a General Chair of the 9th International Conference on Information, Communications, and Signal Processing (ICICS), 2013. He is a Distinguished Lecturer of the IEEE Circuits and Systems Society from 2007 to 2008 and the Chair of the IEEE Circuits and Systems Singapore Chapter from 2007 to 2008.

Han Wang received the B.Eng. degree from Northeast Heavy Machinery Institute (Yan Shan University), Qinhuangdao, China, in 1982, and the Ph.D. degree from Leeds University, Leeds, U.K., in 1989.

He has been a Teacher at Shanghai, China, a Research Associate at Oxford, U.K., a Lecturer, Senior Lecturer, and an Associate Professor at Nanyang Technological University, Singapore, since 1992. His current research interests include 3-D computer vision, recognition, and robot navigation. He has published over 200 papers in international conferences and journals.

Prof. Wang is currently a Chairman for the IEEE, RAS, Singapore chapter.

Yeng Chai Soh received the B.Eng. (Hons. I) degree in electrical and electronic engineering from the University of Canterbury, Christchurch, New Zealand, and the Ph.D. degree in electrical engineering from the University of Newcastle, Callaghan, NSW, Australia.

He joined Nanyang Technological University, Singapore, where he is currently a Professor with the School of Electrical and Electronic Engineering. His current research interests include robust control and applications, robust estimation and filtering, optical signal processing, and energy efficient systems. He has published over 230 refereed journal papers in the above areas.

Dr. Soh was the recipient of several national and international awards for his research achievements in optical signal processing. He has served as the Head of the Control and Instrumentation Division, the Associate Dean (Research and Graduate Studies), and the Associate Dean (Research) at the College of Engineering. He has served as panel members of several national grants and scholarships evaluation committees.