Broad Learning System: An Effective and Efficient Incremental Learning System Without the Need for Deep Architecture

C. L. Philip Chen, *Fellow, IEEE*, and Zhulin Liu

***Abstract*— Broad Learning System (BLS) that aims to offer an alternative way of learning in deep structure is proposed in this paper. Deep structure and learning suffer from a time-consuming training process because of a large number of connecting parameters in filters and layers. Moreover, it encounters a complete retraining process if the structure is not sufficient to model the system. The BLS is established in the form of a flat network, where the original inputs are transferred and placed as "mapped features" in feature nodes and the structure is expanded in wide sense in the "enhancement nodes." The incremental learning algorithms are developed for fast remodeling in broad expansion without a retraining process if the network deems to be expanded. Two incremental learning algorithms are given for both the increment of the feature nodes (or filters in deep structure) and the increment of the enhancement nodes. The designed model and algorithms are very versatile for selecting a model rapidly. In addition, another incremental learning is developed for a system that has been modeled encounters a new incoming input. Specifically, the system can be remodeled in an incremental way without the entire retraining from the beginning. Satisfactory result for model reduction using singular value decomposition is conducted to simplify the final structure. Compared with existing deep neural networks, experimental results on the Modified National Institute of Standards and Technology database and NYU NORB object recognition dataset benchmark data demonstrate the effectiveness of the proposed BLS.**

***Index Terms*— Big data, big data modeling, broad learning system (BLS), deep learning, incremental learning, random vector functional-link neural networks (RVFLNN), single layer feedforward neural networks (SLFN), singular value decomposition (SVD).**

I. Introduction

DEEP structure neural networks and learnings have been applied in many fields and have achieved breakthrough successes in a great number of applications [1], [2], particularly in large scale data processing [3], [4]. Among them, the most popular deep networks are the deep belief networks (DBN) [5], [6], the deep Boltzmann machines (DBM) [7], and the convolutional neural networks (CNN) [8], [9]. Although the deep structure has been so powerful, most of networks suffer from the time-consuming training process because a great number of hyperparameters and complicated structures are involved. Moreover, this complication makes it so difficult to analyze the deep structure theoretically that most work span in turning the parameters or stacking more layers for better accuracy. To achieve this mission, more and more powerful computing resource has been involved. Recently, some variations in hierarchical structure [10]–[12] or ensembles [13]–[17], are proposed to improve the training performance.

Single layer feedforward neural networks (SLFN) have been widely applied to solve problems such as classification and regression because of their universal approximation capability [18]–[21]. Conventional methods for training SLFN are gradient-descent-based learning algorithms [22], [23]. The generalization performance of them is more sensitive to the parameter settings such as learning rate. Similarly, they usually suffer from slow convergence and trap in a local minimum. The random vector functional-link neural network (RVFLNN), proposed in [19]–[21], offers a different learning method.

RVFLNN effectively eliminates the drawback of the long training process and also it provides the generalization capability in function approximation [20], [24]. It has been proven that RVFLNN is a universal approximation for continuous functions on compact sets with fast learning property. Therefore, RVFLNN has been employed to solve problems in diverse domains, including the context of modeling and control [25]. Although RVFLNN enhances the performance of the perception significantly, this technique could not work well on remodeling high-volume and time-variety data in modern large data era [26]. To model moderate data size, a dynamic step-wise updating algorithm was proposed in [27] to update the output weights of the RVFLNN for both a new added pattern and a new added enhancement node. This paper paves a path for remodeling the system that has been modeled and encounters a new incoming data.

Nowadays, in addition to the growth of data in size, the data dimensions also increase tremendously. Taking the raw data with high dimension directly to a neural network, the system cannot sustain its accessibility anymore. The challenge for solving high dimensional data problem becomes imperative recently. Two common practices to alleviate this problem are dimension reduction and feature extraction. Feature extraction

Manuscript received March 25, 2017; revised May 20, 2017 and June 11, 2017; accepted June 12, 2017. This work was supported in part by Macao Science and Technology Development Fund under Grant 019/2015/A1, in part by UM Research Grants, and in part by the National Nature Science Foundation of China under Grant 61572540. *(Corresponding author: C. L. Philip Chen.)*

C. L. P. Chen is with the Department of Computer and Information Science, Faculty of Science and Technology, University of Macau, Macau 99999, China; with Dalian Maritime University, Dalian 116026, China; and also with the State Key Laboratory of Management and Control for Complex Systems, Institute of Automation, Chinese Academy of Sciences, Beijing 100080, China (e-mail: philip.chen@ieee.org).

Z. Liu is with the Department of Computer and Information Science, Faculty of Science and Technology, University of Macau, Macau 99999, China (e-mail: zhulinlau@gmail.com).

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/TNNLS.2017.2716952

is to seek, possible, the optimal transformation from the input data into the feature vectors. Common approaches, which have the advantage of easy implementation and outstanding efficiency, for feature selection include variable ranking [28], feature subset selection [29], penalized least squares [30], random feature extractions such as nonadaptive random projections [31] and random forest [32], or convolution-based input mapping, to name a few.

Likewise for the feature extraction, the RVFLNN can take mapped features as its input. The proposed Broad Learning System (BLS) is designed based on the idea of RVFLNN. In addition, the BLS can effectively and efficiently update the systems (or relearn) incrementally when it deems necessary. The BLS is designed as, first, the mapped features are generated from the input data to form the feature nodes. Second, the mapped features are enhanced as enhancement nodes with randomly generated weights. The connections of all the mapped features and the enhancement nodes are fed into the output. Ridge regression of the pseudoinverse is designed to find the desired connection weights. Broad Learning algorithms are designed for the network through the broad expansion in both the feature nodes and the enhancement nodes. And the incremental learning algorithms are developed for fast remodeling in broad expansion without a retraining process if the network deems to be expanded.

It should be noted that once the learning system has completed its modeling, it may consist of some redundancy due to the broad expansion. The system can be simplified more using low-rank approximations. Low-rank approximation has been established as a new tool in scientific computing to address large-scale linear and multilinear algebra problems, which would be intractable by classical techniques. In [33] and [34], comprehensive exposition of the theory, algorithms, and applications of structured low-rank approximations are presented. Among the various algorithms, the singular value decomposition (SVD) and nonnegative matrix factorization (NMF) [35] are widely used for exploratory data analysis. By embedding these classical low-rank algorithms into the proposed broad learning network, we also design an SVD-based structure to simplify broad learning algorithm. This simplification process can be done by compressing the system on-the-fly right after the moment when additional feature nodes are inserted or right after additional enhancement nodes are inserted or after both together. One can also choose to compress the whole structure after the learning is completed. The algorithms are designed so versatile such that one can select number of equivalent neural nodes in the final structure. This approach lays out an excellent approach for model selection.

This paper is organized as follows. In Section II, preliminaries of RVFLNN, ridge regression approach of pseudoinverse, sparse autoencoder, and SVD are given. Section III presents the BLS and gives the details of the proposed broad learning algorithms. Section IV compares the performance in Modified National Institute of Standards and Technology database (MNIST) classification and NYU object recognition benchmark (NORB) classification with those from various deep systems. Also the performance analysis of the proposal

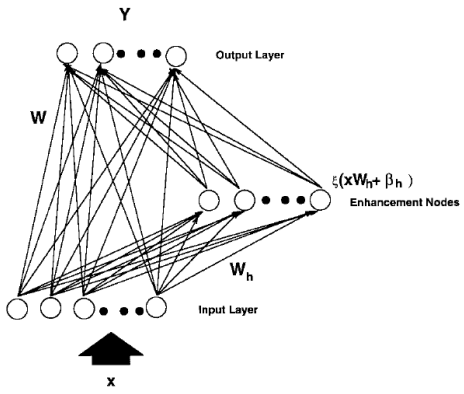

Fig. 1. Functional-link neural network [27].

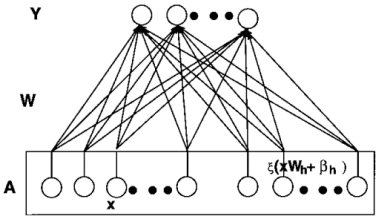

Fig. 2. Same functional-link network redrawn from Fig. 1 [27].

broad learning algorithms is addressed here. Finally, discussions and conclusions are given in section V.

II. Preliminaries

Pao and Takefuji [19] and Igelnik and Pao [21] give a classical mathematic discussion of the advantages of the functional-link network in terms of training speed and its generalization property over the general feedforward networks [20]. The capabilities and universal approximation properties of RVFLNN have been clearly shown and, hence, are omitted in this section. The illustration of the flatted characteristics of the functional-link network is shown again in Figs. 1 and 2. In this section, first, the proposed broad learning is introduced based on the rapid and dynamic learning features of the functional-link network developed by Chen and Wan [27] and Chen [36]. Second, the ridge regression approximation algorithm of the pesudoinverse is recalled. After that, sparse autoencoder and SVD are briefly discussed.

A. Dynamic Stepwise Updating Algorithm for Functional-Link Neural Networks

For a general network in usual classification task, referring to Figs. 1 and 2, denoted by $\boldsymbol{A}$ the matrix $[\boldsymbol{X}|\xi(\boldsymbol{X}\boldsymbol{W}_h+\boldsymbol{\beta}_h)]$, where $\boldsymbol{A}$ is the expanded input matrix consisting of all input vectors combined with enhancement components. The functional-link network model is illustrated in Fig. 2. In [27], a dynamic version model was proposed to update the weights of the system instantly for both a new added pattern and a new added enhancement node. Compared with the classic model, this model is simple, fast, and easy to update. Generally, this model was inspired by the pseudoinverse of a partitioned matrix described in [37] and [38].

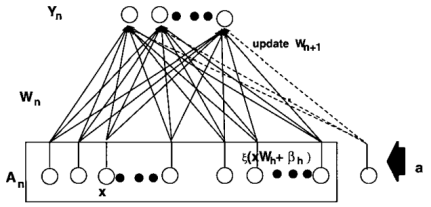

Fig. 3. Illustration of stepwise update algorithm [27].

Denote $\boldsymbol{A}_n$ be the $n \times m$ pattern matrix. In this section, we will only introduce the stepwise updating algorithm for adding a new enhancement node to the network as shown in Fig. 3. In this case, it is equivalent to add a new column to the input matrix $\boldsymbol{A}_n$. Denote $\boldsymbol{A}_{n+1} \triangleq [\boldsymbol{A}_n|\boldsymbol{a}]$. Then the pseudoinverse of the new $\boldsymbol{A}_{n+1}^+$ equals

$$\begin{bmatrix} \boldsymbol{A}_n^+ - \boldsymbol{d}\boldsymbol{b}^T \\ \boldsymbol{b}^T \end{bmatrix}$$

where $\boldsymbol{d} = \boldsymbol{A}_n^+\boldsymbol{a}$

$$\boldsymbol{b}^T = \begin{cases} (\boldsymbol{c})^+ & \text{if } \boldsymbol{c} \neq 0 \\ (\boldsymbol{1} + \boldsymbol{d}^T\boldsymbol{d})^{-1}\boldsymbol{d}^T\boldsymbol{A}_n^+ & \text{if } \boldsymbol{c} = 0 \end{cases}$$

and $\boldsymbol{c} = \boldsymbol{a} - \boldsymbol{A}_n\boldsymbol{d}$.

Again, the new weights are

$$\boldsymbol{W}_{n+1} = \begin{bmatrix} \boldsymbol{W}_n - \boldsymbol{d}\boldsymbol{b}^T\boldsymbol{Y}_n \\ \boldsymbol{b}^T\boldsymbol{Y}_n \end{bmatrix}$$

where $\boldsymbol{W}_{n+1}$ and $\boldsymbol{W}_n$ are the weights after and before new enhancement nodes are added, respectively. In this way, the new weights can be updated easily by only computing the pseudoinverse of the corresponding added node. It should be noted that if $\boldsymbol{A}_n$ is of the full rank, then $\boldsymbol{c} = 0$ which will make a fast updating of the pseudoinverse $\boldsymbol{A}_n^+$ and the weight $\boldsymbol{W}_n$.

B. Pseudoinverse and Ridge Regression Learning Algorithms

In a flatted network, pseudoinverse can be considered as a very convenient approach to solve the output-layer weights of a neural network. Different methods can be used to calculate this generalized inverse, such as orthogonal projection method, orthogonalization method, iterative method, and SVD [39], [40]. However, a direct solution is too expensive, especially the training samples and input patterns are suffered from high volume, high velocity, and/or high variety [26]. Also, pseudoinverse, which is the least square estimator of linear equations, is aimed to reach the output weights with the smallest training errors, but may not true for generalization errors, especially for the ill condition problems. In fact, the following kind of optimal problems is an alternative way to solve the pseudoinverse:

$$\arg\min_{\mathbf{W}} : \|\boldsymbol{A}\boldsymbol{W} - \boldsymbol{Y}\|_v^{\sigma_1} + \lambda\|\boldsymbol{W}\|_u^{\sigma_2} \tag{1}$$

where $\sigma_1 > 0$, $\sigma_2 > 0$, and u, v are typically kinds of norm regularization. By taking $\sigma_1 = \sigma_2 = u = v = 2$, the above optimal problem is setting as regular l_2 norm regularization, which is convex and has a better generalization performance. The value λ denotes the further constraints on the sum of the squared weights, $\boldsymbol{W}$. This solution is equivalent with the ridge regression theory, which gives an approximation to the Moore−Penrose generalized inverse by adding a positive number to the diagonal of $\boldsymbol{A}^T\boldsymbol{A}$ or $\boldsymbol{A}\boldsymbol{A}^T$ [41]. Theoretically, if $\lambda = 0$, the inverse problem degenerates into the least square problem and leads the solution to original pesuedoinverse. On the other hand, if $\lambda \to \infty$, the solution is heavily constrained and tends to $\mathbf{0}$. Consequently, we have

$$\boldsymbol{W} = (\lambda\boldsymbol{I} + \boldsymbol{A}\boldsymbol{A}^T)^{-1}\boldsymbol{A}^T\boldsymbol{Y}. \tag{2}$$

Specifically, we have that

$$\boldsymbol{A}^+ = \lim_{\lambda \to 0}(\lambda\boldsymbol{I} + \boldsymbol{A}\boldsymbol{A}^T)^{-1}\boldsymbol{A}^T. \tag{3}$$

C. Sparse Autoencoder

Supervised learning tasks, such as classifications, usually need a good feature representation of the input to achieve an outstanding performance. Feature representation is not only an efficient way of data representation, but also, more importantly, it captures the characteristics of the data. Usually, intractable mathematic derivation is used or an easy random initialization to generate a set of random features can be populated. However, randomness suffers from unpredictability and needs guidance. To overcome the randomness nature, the sparse autoencoder could be regarded as an important tool to slightly fine-tune the random features to a set of sparse and compact features. Specifically, sparse feature learning models have been attractive that could explore essential characterization [12], [42], [43].

To extract the sparse features from the given training data, $\boldsymbol{X}$ can be considered to solve the optimization problem [44]. Notice that it is equivalent to the optimization problem in (1) if we set $\sigma_2 = u = 1$, and $\sigma_1 = v = 2$

$$\arg\min_{\hat{W}} : \|\boldsymbol{Z}\hat{\boldsymbol{W}} - \boldsymbol{X}\|_2^2 + \lambda\|\hat{\boldsymbol{W}}\|_1 \tag{4}$$

where $\hat{\boldsymbol{W}}$ is the sparse autoencoder solution and $\boldsymbol{Z}$ is the desired output of the given linear equation, i.e., $\boldsymbol{X}\boldsymbol{W} = \boldsymbol{Z}$.

The above problem denoted by *lasso* in [45] is convex. Consequently, the approximation problem in (4) can be solved by dozens of ways, such as orthogonal matching pursuit [46], K-SVD [47], alternating direction method of multipliers (ADMM) [48], and fast iterative shrinkage-thresholding algorithm (FISTA) [49]. Among them, the ADMM method is actually designed for general decomposition methods and decentralized algorithms in the optimization problems. Moreover, it has been shown that many state-of-art algorithms for l_1 norm involved problems could be derived by ADMM [50], such as FISTA. Hence, a brief review of typical approach for *lasso* is presented below.

First, (4) can be equivalently considered as the following general problem:

$$\arg\min_{\mathbf{w}} : f(\boldsymbol{w}) + g(\boldsymbol{w}), \quad \boldsymbol{w} \in \mathbb{R}^n \tag{5}$$

where $f(\boldsymbol{w}) = \|\boldsymbol{Z}\boldsymbol{w} - \boldsymbol{x}\|_2^2$ and $g(\boldsymbol{w}) = \lambda\|\boldsymbol{w}\|_1$. In ADMM form, the above problem could be rewritten as

$$\arg\min_{\mathbf{w}} : f(\boldsymbol{w}) + g(\boldsymbol{o}), \quad \text{s.t. } \boldsymbol{w} - \boldsymbol{o} = 0. \tag{6}$$

Therefore, the proximal problem could be solved by the following iterative steps:

$$\begin{cases} \boldsymbol{w}_{k+1} := (\boldsymbol{Z}^T\boldsymbol{Z} + \rho\boldsymbol{I})^{-1}(\boldsymbol{Z}^T\boldsymbol{x} + \rho(\boldsymbol{o}^k - \boldsymbol{u}^k)) \\ \boldsymbol{o}_{k+1} := S_{\frac{\lambda}{\rho}}(\boldsymbol{w}_{k+1} + \boldsymbol{u}_k) \\ \boldsymbol{u}_{k+1} := \boldsymbol{u}_k + (\boldsymbol{w}_{k+1} - \boldsymbol{o}_{k+1}) \end{cases} \tag{7}$$

where $\rho > 0$ and S is the soft threshholding operator which is defined as

$$S_\kappa(a) = \begin{cases} a - \kappa, & a > \kappa \\ 0, & |a| \le \kappa \\ a + \kappa, & a < -\kappa. \end{cases} \tag{8}$$

D. Singular Value Decomposition

Now comes a highlight of linear algebra. Any real $m \times n$ matrix A can be factored as

$$\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

where $\boldsymbol{U}$ is an $m \times m$ orthogonal matrix whose columns are the eigenvectors of $\boldsymbol{A}\boldsymbol{A}^T$, $\boldsymbol{V}$ is an $n \times n$ orthogonal matrix whose columns are the eigenvectors of $\boldsymbol{A}^T\boldsymbol{A}$, and $\boldsymbol{\Sigma}$ is an $m \times n$ diagonal matrix of the form

$$\boldsymbol{\Sigma} = \text{diag}\{\sigma_1, \ldots, \sigma_r, 0, \ldots, 0\}$$

with $\sigma_1 \geqslant \sigma_2 \geqslant \cdots \geqslant \sigma_r > 0$ and $r = \text{rank}(\boldsymbol{A})$. Moreover, in the above, $\sigma_1, \ldots, \sigma_r$ are the square roots of the eigenvalues of $\boldsymbol{A}^T\boldsymbol{A}$. They are called the singular values of $\boldsymbol{A}$. Therefore, we achieve a decomposition of matrix $\boldsymbol{A}$, which is one of a number of effective numerical analysis tools used to analyze matrices. In our algorithms, two different ways to reduce the size of the matrix are involved. In the first, the threshold parameter η is set as $0 < \eta \le 1$, which means that the components associate with an eigenvalue $\sigma_i \ge \eta\sigma_1$ are kept. The second case is to select a fixed l singularities, while l is smaller than n. Define a threshold value ε, which is η for case 1 and l for case 2. In the real practice, both of the cases may be happened depending on various requirements. An SVD technique is known in its advantage in feature selection.

III. Broad Learning System

In this section, the details of the proposed BLS are given. First, the model construction that is suitable for broad expansion is introduced, and the second part is the incremental learning for the dynamic expansion of the model. The two characteristics result in a complete system. At last, model simplification using SVD is presented.

A. Broad Learning Model

The proposed BLS is constructed based on the traditional RVFLNN. However, unlike the traditional RVFLNN that takes the input directly and establishes the enhancement nodes, we first map the inputs to construct a set of mapped features. In addition, we also develop incremental learning algorithms that can update the system dynamically.

Assume that we present the input data $\boldsymbol{X}$ and project the data, using $\phi_i(\boldsymbol{X}\boldsymbol{W}_{e_i} + \boldsymbol{\beta}_{e_i})$, to become the ith mapped features, $\boldsymbol{Z}_i$, where $\boldsymbol{W}_{e_i}$ is the random weights with the proper dimensions. Denote $\boldsymbol{Z}^i \equiv [\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_i]$, which is the concatenation of all the first i groups of mapping features. Similarly, the jth group of enhancement nodes, $\xi_j(\boldsymbol{Z}^i\boldsymbol{W}_{h_j} + \boldsymbol{\beta}_{h_j})$ is denoted as $\boldsymbol{H}_j$, and the concatenation of all the first j groups of enhancement nodes are denoted as $\boldsymbol{H}^j \equiv [\boldsymbol{H}_1, \ldots, \boldsymbol{H}_j]$. In practice, i and j can be selected differently depending upon the complexity of the modeling tasks. Furthermore, ϕ_i and ϕ_k can be different functions for $i \neq k$. Similarly, ξ_j and ξ_r can be different functions for $j \neq r$. Without loss of generality, the subscripts of the ith random mappings ϕ_i and the jth random mappings ξ_j are omitted in this paper.

In our BLS, to take the advantages of sparse autoencoder characteristics, we apply the linear inverse problem in (7) and fine-tune the initial $\boldsymbol{W}_{e_i}$ to obtain better features. Next, the details of the algorithm are given below.

Assume the input data set $\boldsymbol{X}$, which equips with N samples, each with M dimensions, and $\boldsymbol{Y}$ is the output matrix which belongs to $\mathbb{R}^{N\times C}$. For n feature mappings, each mapping generates k nodes, can be represented as the equation of the form

$$\boldsymbol{Z}_i = \phi(\boldsymbol{X}\boldsymbol{W}_{e_i} + \boldsymbol{\beta}_{e_i}), \quad i = 1, \ldots, n \tag{9}$$

where $\boldsymbol{W}_{e_i}$ and $\boldsymbol{\beta}_{e_i}$ are randomly generated. Denote all the feature nodes as $\boldsymbol{Z}^n \equiv [\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_n]$, and denote the mth group of enhancement nodes as

$$\boldsymbol{H}_m \equiv \xi(\boldsymbol{Z}^n\boldsymbol{W}_{h_m} + \boldsymbol{\beta}_{h_m}). \tag{10}$$

Hence, the broad model can be represented as the equation of the form

$$\begin{aligned} \boldsymbol{Y} &= [\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_n|\xi(\boldsymbol{Z}^n\boldsymbol{W}_{h_1} + \boldsymbol{\beta}_{h_1}), \ldots, \xi(\boldsymbol{Z}^n\boldsymbol{W}_{h_m} + \boldsymbol{\beta}_{h_m})]\boldsymbol{W}^m \\ &= [\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_n|\boldsymbol{H}_1, \ldots, \boldsymbol{H}_m]\boldsymbol{W}^m \\ &= [\boldsymbol{Z}^n|\boldsymbol{H}^m]\boldsymbol{W}^m \end{aligned}$$

where the $\boldsymbol{W}^m = [\boldsymbol{Z}^n|\boldsymbol{H}^m]^+\boldsymbol{Y}$. $\boldsymbol{W}^m$ are the connecting weights for the broad structure and can be easily computed through the ridge regression approximation of $[\boldsymbol{Z}^n|\boldsymbol{H}^m]^+$ using (3). Fig. 4(a) shows the above broad learning network.

B. Alternative Enhancement Nodes Establishment

In the previous section, the broad expansion of enhancement nodes is added synchronously with the connections from mapped features. Here, a different construction can be done by connecting each group of mapped features to a group of enhancement nodes. Details are described below.

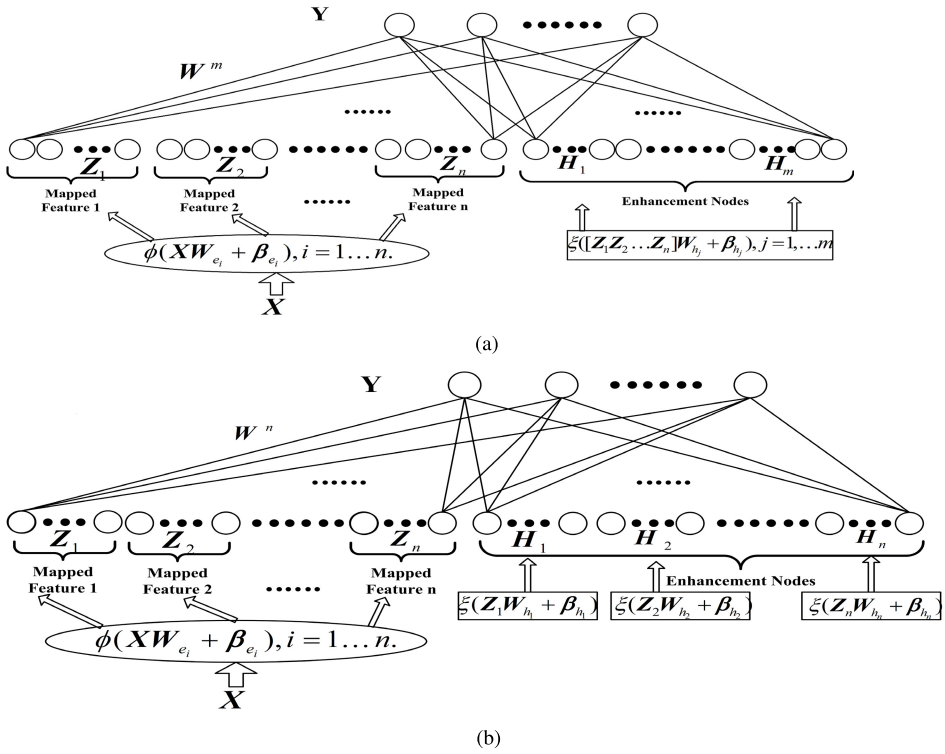

Fig. 4. Illustration of BLS. (a) BLS. (b) BLS with an alternative enhancement nodes establishment.

For input data set, $\boldsymbol{X}$, for n mapped features and for n enhancement groups, the new construction is

$$\boldsymbol{Y} = [\boldsymbol{Z}_1, \xi(\boldsymbol{Z}_1\boldsymbol{W}_{h_1} + \boldsymbol{\beta}_{h_1})| \dots \boldsymbol{Z}_n, \xi(\boldsymbol{Z}_n\boldsymbol{W}_{h_n} + \boldsymbol{\beta}_{h_n})]\boldsymbol{W}^n \quad (11)$$

$$\triangleq [\boldsymbol{Z}_1, \dots, \boldsymbol{Z}_n|\xi(\boldsymbol{Z}_1\boldsymbol{W}_{h_1} + \boldsymbol{\beta}_{h_1}), \dots, \xi(\boldsymbol{Z}_n\boldsymbol{W}_{h_n} + \boldsymbol{\beta}_{h_n})]\boldsymbol{W}^n \quad (12)$$

where $\boldsymbol{Z}_i$, $i = 1, \dots, n$, are $N \times \alpha$ dimensional mapping features achieved by (9) and $\boldsymbol{W}_{h_j} \in \mathbb{R}^{\alpha\times\gamma}$. This model structure is illustrated in Fig. 4(b).

It is obvious that the main difference between two constructions, in Fig. 4(a) and (b), is in the connections of the enhancement nodes. The following theorem proves that the above two different connections in the enhancement nodes are actually equivalent.

Theorem 1: For the model in Section III-A [or Fig. 4(a)], the feature dimension of $\boldsymbol{Z}_i^{(a)}$ is k, for $i = 1, \dots, n$, and the dimension of $\boldsymbol{H}_j^{(a)}$ is q, for $j = 1, \dots, m$. Respectively, for the model in Section III-B [or Fig. 4(b)] the feature dimension of $\boldsymbol{Z}_i^{(b)}$ is k, for $i = 1, \dots, n$, and the dimension of $\boldsymbol{H}_j^{(b)}$ is γ, for $j = 1, \dots, n$. Then if $mq = n\gamma$, and $\boldsymbol{H}^{(a)}$ and $\boldsymbol{H}^{(b)}$ are normalized, the two networks are exactly equivalent.

Consequently, no matter which kind of establishment of enhancement nodes is adapted, the network is essentially the same, as long as the total number of feature nodes and enhancement nodes are equal. Hereby, only the model in Section III-A [or Fig. 4(a)] will be considered in the rest of this paper. The proof is as follows.

Proof: Suppose that the elements of $\boldsymbol{W}_{e_i}$, $\boldsymbol{W}_{h_j}$, $\boldsymbol{\beta}_{e_i}$, and $\boldsymbol{\beta}_{h_j}$ are randomly drawn independently of the same distribution $\rho(w)$. For the first model, treat the nk feature mapping together as $\boldsymbol{Z}^{(a)} \in \mathbb{R}^{N\times nk}$ whose weights are $\boldsymbol{W}_e^{(a)}$, and separately, the mq enhancement nodes are denoted together as $\boldsymbol{H}^{(a)} \in \mathbb{R}^{N\times mq}$ whose weights are $\boldsymbol{W}_h^{(a)}$. Respectively, for the second model, treat the nk feature mapping together as $\boldsymbol{Z}^{(b)} \in \mathbb{R}^{N\times nk}$ whose weights are $\boldsymbol{W}_e^{(b)}$, and separately, the $n\gamma$ enhancement nodes are denoted together as $\boldsymbol{H}^{(b)} \in \mathbb{R}^{N\times n\gamma}$ whose weights are $\boldsymbol{W}_h^{(b)}$. Obviously, $\boldsymbol{W}_e^{(b)}$ and $\boldsymbol{W}_e^{(a)}$, that is with the same dimension, are exactly equivalent because the entrances of the two matrices are generated from the same distribution.

As for the enhancement nodes part, first we have that $\boldsymbol{H}^{(a)}$ and $\boldsymbol{H}^{(b)}$ are of the same size if $mq = n\gamma$. Therefore, we need to prove that their elements are equivalent. For any sample $\boldsymbol{x}_l$ chosen from the data set, denote the columns of $\boldsymbol{W}_e^{(a)}$ as $\boldsymbol{w}_{e_i}^a \in \mathbb{R}^N, i = 1, \dots, nk$ and the columns of $\boldsymbol{W}_h^{(a)}$ as $\boldsymbol{w}_{h_j}^a \in \mathbb{R}^{nk}$, $j = 1, \dots, mq$.

Hence, the jth enhancement node associate with the sample $\boldsymbol{x}_l$ should be

$$\begin{aligned}
\boldsymbol{H}_{lj}^{(a)} &= \xi\big(\phi\big(\boldsymbol{X}\boldsymbol{W}_e^{(a)} + \boldsymbol{\beta}_e^{(a)}\big)\boldsymbol{W}_h^{(a)} + \boldsymbol{\beta}_h^{(a)}\big)_{lj} \\
&= \xi\big(\big[\phi\big(\boldsymbol{x}_l\boldsymbol{w}_{e_1}^a + \beta_{el1}^a\big), \dots, \phi\big(\boldsymbol{x}_l\boldsymbol{w}_{e_{nk}}^a + \beta_{elnk}^a\big)\big]\boldsymbol{w}_{h_j}^a + \beta_{h_j}^a\big) \\
&= \sum_{i=1}^{nk} \xi\big(\phi\big(\boldsymbol{x}_l\boldsymbol{w}_{e_i}^a + \beta_{li}^a\big)w_{h_{li}}^a + \beta_{h_{li}}^a\big) \\
&\approx nkE\big[\xi\big(\phi\big(\boldsymbol{x}_l\boldsymbol{w}_e^a + \beta_e^a\big)w_h^a + \beta_h^a\big)\big] \\
&= nkE\big[\xi\big(\phi\big(\boldsymbol{x}_l\boldsymbol{w}_e + \beta_e\big)w_h + \beta_h\big)\big].
\end{aligned}$$

Here E stands for the expectation of distribution, $\boldsymbol{w}_e$ is the N dimension random vector drawn from the distribution density $\rho(w)$, and w_h is the scaler number sampled from $\rho(w)$.

Similarly, for the columns of $\boldsymbol{W}_e^{(b)}$ as $\boldsymbol{w}_{e_i}^b \in \mathbb{R}^N, i = 1, \ldots, nk$ and the columns of $\boldsymbol{W}_h^{(b)}$ as $\boldsymbol{w}_{h_j}^b \in \mathbb{R}^k, j = 1, \ldots, n\gamma$, it could be deduced that for the second model we have

$$\begin{aligned}\boldsymbol{H}_{lj}^{(b)} &= \xi\big(\phi\big(\boldsymbol{X}\boldsymbol{W}_e^{(b)} + \boldsymbol{\beta}_e^{(b)}\big)\boldsymbol{W}_h^{(b)} + \boldsymbol{\beta}_h^{(b)}\big)_{lj} \\ &= \xi\big(\big[\phi\big(\boldsymbol{x}_{e_1}\boldsymbol{w}_1^b + \beta_{e_{l1}}^b\big), \ldots, \phi\big(\boldsymbol{x}_{e_l}\boldsymbol{w}_{e_k}^b + \beta_{e_{lk}}^b\big)\big]\boldsymbol{w}_{h_j}^b + \beta_j^b\big) \\ &= \sum_{i=1}^{nk} \xi\big(\phi\big(\boldsymbol{x}_l\boldsymbol{w}_{e_i}^b + \beta_{e_{li}}^b\big)w_{h_{li}}^b + \beta_{h_{ij}}^b\big) \\ &\approx kE\big[\xi\big(\phi\big(\boldsymbol{x}_l\boldsymbol{w}_e^b + \beta_e^b\big)w_h{}^b + \beta_h^b\big)\big] \\ &= kE\big[\xi\big(\phi\big(\boldsymbol{x}_l\boldsymbol{w}_e + \beta_e\big)w_h + \beta_h\big)\big].\end{aligned}$$

Since all the $\boldsymbol{w}_e$, w_h and β_e, β_h are drawn from the same distribution, the expectations of the above two composite distributions are obviously the same. Hence, it is clear that

$$\boldsymbol{H}_{lj}^{(b)} \triangleq \frac{1}{n}\boldsymbol{H}_{lj}^{(a)}.$$

Therefore, we could conclude that under the given assumption, $\boldsymbol{H}^{(a)}$ and $\boldsymbol{H}^{(b)}$ are also equivalent if the normalization operator is applied. ■

C. Incremental Learning for Broad Expansion: Increment of Additional Enhancement Nodes

In some cases, if the learning cannot reach the desired accuracy, one solution is to insert additional enhancement nodes to achieve a better performance. Next, we will detail the broad expansion method for adding p enhancement nodes. Denote $\boldsymbol{A}^m = [\boldsymbol{Z}^n|\boldsymbol{H}^m]$ and $\boldsymbol{A}^{m+1}$ as

$$\boldsymbol{A}^{m+1} \equiv [\boldsymbol{A}^m|\xi(\boldsymbol{Z}^n\boldsymbol{W}_{h_{m+1}} + \boldsymbol{\beta}_{h_{m+1}})]$$

where $\boldsymbol{W}_{h_{m+1}} \in \mathbb{R}^{nk\times p}$, and $\boldsymbol{\beta}_{h_{m+1}} \in \mathbb{R}^p$. The connecting weights and biases from mapped features to the p additional enhancement nodes, are randomly generated.

By the discussion in Section II, we could deduce the pseudoinverse of the new matrix as

$$(\boldsymbol{A}^{m+1})^+ = \begin{bmatrix}(\boldsymbol{A}^m)^+ - \boldsymbol{D}\boldsymbol{B}^T \\ \boldsymbol{B}^T\end{bmatrix} \tag{13}$$

where $\boldsymbol{D} = (\boldsymbol{A}^m)^+\xi(\boldsymbol{Z}^n\boldsymbol{W}_{h_{m+1}} + \boldsymbol{\beta}_{h_{m+1}})$

$$\boldsymbol{B}^T = \begin{cases}(\boldsymbol{C})^+ & \text{if } \boldsymbol{C} \neq 0 \\ (\boldsymbol{1} + \boldsymbol{D}^T\boldsymbol{D})^{-1}\boldsymbol{B}^T(\boldsymbol{A}^m)^+ & \text{if } \boldsymbol{C} = 0\end{cases} \tag{14}$$

and $\boldsymbol{C} = \xi(\boldsymbol{Z}^n\boldsymbol{W}_{h_{m+1}} + \boldsymbol{\beta}_{h_{m+1}}) - \boldsymbol{A}^m\boldsymbol{D}$.

Again, the new weights are

$$\boldsymbol{W}^{m+1} = \begin{bmatrix}\boldsymbol{W}^m - \boldsymbol{D}\boldsymbol{B}^T\boldsymbol{Y} \\ \boldsymbol{B}^T\boldsymbol{Y}\end{bmatrix}. \tag{15}$$

The broad learning construction model and learning procedure is listed in Algorithm 1, meanwhile, the structure is illustrated in Fig. 5. Notice that all the peseudoinverse of the involved matrix are calculated by the regularization approach in (3). Specifically, this algorithm only needs to compute the pseudoinverse of the additional enhancement nodes instead of computations of the entire $(\boldsymbol{A}^{m+1})$ and thus results in fast incremental learning.

Algorithm 1 Broad Learning: Increment of p Additional Enhancement Nodes

Input : training samples $\boldsymbol{X}$;
Output: $\boldsymbol{W}$

for $i = 0; i \leq n$ **do**
  Random $\boldsymbol{W}_{e_i}$, $\boldsymbol{\beta}_{e_i}$;
  Calculate $\boldsymbol{Z}_i = [\phi(\boldsymbol{X}\boldsymbol{W}_{e_i} + \boldsymbol{\beta}_{e_i})]$;
end
Set the feature mapping group $\boldsymbol{Z}^n = [\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_n]$;
for $j = 1; j \leq m$ **do**
  Random $\boldsymbol{W}_{h_j}$, $\boldsymbol{\beta}_{h_j}$;
  Calculate $\boldsymbol{H}_j = [\xi(\boldsymbol{Z}^n\boldsymbol{W}_{h_j} + \boldsymbol{\beta}_{h_j})]$;
end
Set the enhancement nodes group $\boldsymbol{H}^m = [\boldsymbol{H}_1, \ldots, \boldsymbol{H}_m]$;
Set $\boldsymbol{A}^m$ and calculate $(\boldsymbol{A}^m)^+$ with Eq. (3);
while *The training error threshold is not satisfied* **do**
  Random $\boldsymbol{W}_{h_{m+1}}$, $\boldsymbol{\beta}_{h_{m+1}}$;
  Calculate $\boldsymbol{H}_{m+1} = [\xi(\boldsymbol{Z}_{m+1}\boldsymbol{W}_{h_{m+1}} + \boldsymbol{\beta}_{h_{m+1}})]$;
  Set $\boldsymbol{A}^{m+1} = [A^m|\boldsymbol{H}_{m+1}]$;
  Calculate $(\boldsymbol{A}^{m+1})^+$ and $\boldsymbol{W}^{m+1}$ by Eq. (13,14,15);
  $m = m + 1$;
end
Set $\boldsymbol{W} = \boldsymbol{W}^{m+1}$;

D. Incremental Learning for Broad Expansion: Increment of the Feature Mapping Nodes

In various applications, with the selected feature mappings, the dynamic increment of the enhancement nodes may not be good enough for learning. This may be caused from the insufficient feature mapping nodes that may not extract enough underlying variation factors which define the structure of the input data.

In popular deep structure networks, when the existing model could not learn the task well, general practices are to either increase the number of the filter (or window) or increase the number of layer. The procedures suffer from tedious learning by resetting the parameters for new structures.

Instead, in the proposed BLS, if the increment of a new feature mapping is needed, the whole structure can be easily constructed and the incremental learning is applied without retraining the whole network from the beginning.

Here, let us consider the incremental learning for newly incremental feature nodes. Assume that the initial structure consists of n groups feature mapping nodes and m groups broad enhancement nodes. Considering that the $(n + 1)$th feature mapping group nodes are added and denoted as

$$\boldsymbol{Z}_{n+1} = \phi(\boldsymbol{X}\boldsymbol{W}_{e_{n+1}} + \boldsymbol{\beta}_{e_{n+1}}). \tag{16}$$

The corresponding enhancement nodes are randomly generated as follows:

$$\boldsymbol{H}_{ex_m} = [\xi(\boldsymbol{Z}_{n+1}\boldsymbol{W}_{ex_1} + \boldsymbol{\beta}_{ex_1}), \ldots, \xi(\boldsymbol{Z}_{n+1}\boldsymbol{W}_{ex_m} + \boldsymbol{\beta}_{ex_m})] \tag{17}$$

where $\boldsymbol{W}_{ex_i}$ and $\boldsymbol{\beta}_{ex_i}$ are randomly generated. Denote $\boldsymbol{A}_{n+1}^m = [\boldsymbol{A}_n^m|\boldsymbol{Z}_{n+1}|\boldsymbol{H}_{ex_m}]$, which is the upgrade of new mapped features and the corresponding enhancement nodes. The relatively

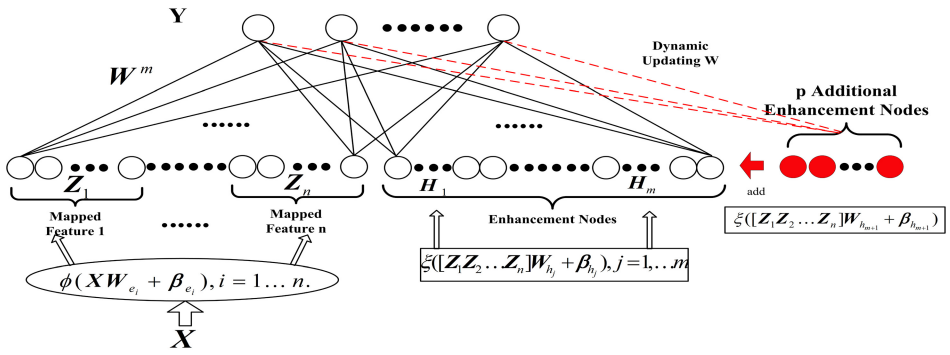

Fig. 5. Broad learning: increment of p additional enhancement nodes.

upgraded pseudoinverse matrix should be achieved as follows:

$$\left(A_{n+1}^m\right)^+ = \begin{bmatrix} (A_n^m)^+ - DB^T \\ B^T \end{bmatrix} \tag{18}$$

where $D = (A_n^m)^+[Z_{n+1}|H_{ex_m}]$

$$B^T = \begin{cases} (C)^+ & \text{if } C \neq 0 \\ (1 + D^T D)^{-1} D^T \left(A_n^m\right)^+ & \text{if } C = 0 \end{cases} \tag{19}$$

and $C = [Z_{n+1}|H_{ex_m}] - A_n^m D$.

Again, the new weights are

$$W_{n+1}^m = \begin{bmatrix} W_n^m - DB^T Y \\ B^T Y \end{bmatrix}. \tag{20}$$

Specifically, this algorithm only needs to compute the pseudoinverse of the additional mapped features instead of computing of the entire A_{n+1}^m, thus resulting in fast incremental learning.

The incremental algorithm of the increment feature mapping is shown in Algorithm 2. And the incremental network for additional $(n+1)$ feature mapping as well as p enhancement nodes is shown in Fig. 6.

Algorithm 2 Broad Learning: Increment of $n+1$ Mapped Features

Input : training samples X;
Output: W

for $i = 0; i \leq n$ **do**
  Random W_{e_i}, β_{e_i};
  Calculate $Z_i = [\phi(XW_{e_i} + \beta_{e_i})]$;
end
Set the feature mapping group $Z^n = [Z_1, \ldots, Z_n]$;
for $j = 1; j \leq m$ **do**
  Random W_{h_j}, β_{h_j};
  Calculate $H_j = [\xi(Z^n W_{h_j} + \beta_{h_j})]$;
end
Set the enhancement nodes group $H^m = [H_1, \ldots, H_m]$;
Set A_n^m and calculate $(A_n^m)^+$ with Eq. (3);
while *The training error threshold is not satisfied* **do**
  Random $W_{e_{n+1}}$, $\beta_{e_{n+1}}$;
  Calculate $Z_n = [\phi(XW_{e_{n+1}} + \beta_{e_{n+1}})]$;
  Random W_{ex_i}, β_{ex_i}, $i = 1, \ldots, m$;
  Calculate $H_{ex_m} = [\xi(Z_{n+1}W_{ex_1} + \beta_{ex_1}), \ldots, \xi(Z_{n+1}W_{ex_m} + \beta_{ex_m})]$;
  Update A_{n+1}^m;
  Update $(A_{n+1}^m)^+$ and W_{n+1}^m by Eq. (18,19,20);
  $n = n + 1$;
end
Set $W = W_{n+1}^m$;

E. Incremental Learning for the Increment of Input Data

Now let us come to the cases that the input training samples keep entering. Often, once a system modeling is completed and if a new input with a corresponding output enters to model, the model should be updated to reflect the additional samples. The algorithm in this section is designed to update the weights easily without an entire training cycle.

Denote X_a as the new inputs added into the neural network, and denote A_n^m as the n groups of feature mapping nodes and m groups of enhancement nodes of the initial network. The respectively increment of mapped feature nodes and the enhancement nodes are formulated as follows:

$$A_x = \left[\phi(X_a W_{e_1} + \beta_{e_1}), \ldots, \phi(X_a W_{e_n} + \beta_{e_n})\right| \tag{21}$$

$$\left.\xi\left(Z_x^n W_{h_1} + \beta_{h_1}\right), \ldots, \xi\left(Z_x^n W_{h_m} + \beta_{h_m}\right)\right] \tag{22}$$

where $Z_x{}^n = [\phi(X_a W_{e_1} + \beta_{e_1}), \ldots, \phi(X_a W_{e_n} + \beta_{e_n})]$ is the group of the incremental features updated by X_a. The W_{e_i}, W_{h_j} and β_{e_i}, β_{h_j} are randomly generated during the initial of the network. Hence, we have the updating matrix

$${}^x A_n^m = \begin{bmatrix} A_n^m \\ A_x^T \end{bmatrix}.$$

The associated pseudoinverse updating algorithm could be deduced as follows:

$$\left({}^x A_n^m\right)^+ = \left[\left(A_n^m\right)^+ - BD^T | B\right] \tag{23}$$

where $D^T = A_x^T A_n^{m+}$

$$B^T = \begin{cases} (C)^+ & \text{if } C \neq 0 \\ (1 + D^T D)^{-1} \left(A_n^m\right)^+ D & \text{if } C = 0 \end{cases} \tag{24}$$

and $C = A_x^T - D^T A_n^m$.

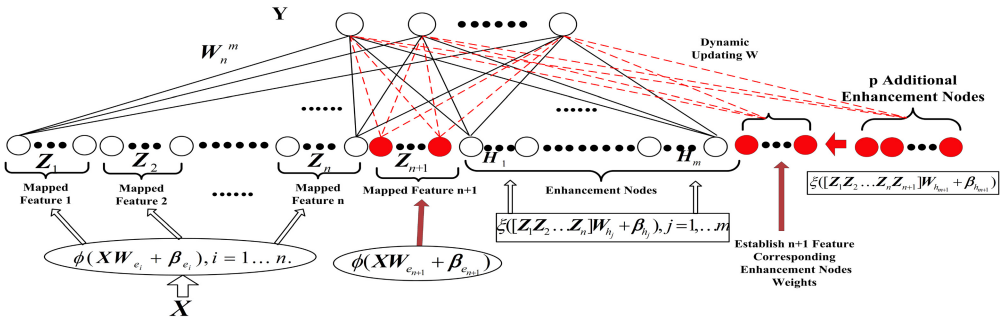

Fig. 6. Broad learning: increment of $n+1$ mapped features.

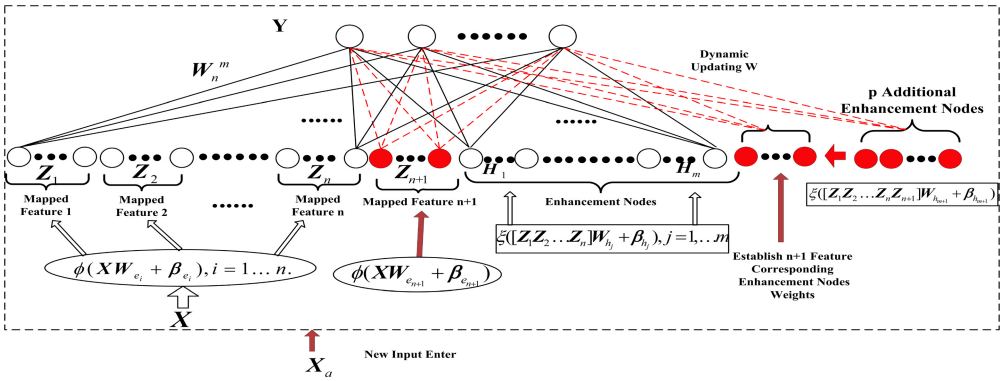

Fig. 7. Broad learning: increment of input data.

Therefore the updated weights are

$$^{x}\boldsymbol{W}_n^m = \boldsymbol{W}_n^m + \left(\boldsymbol{Y}_{\boldsymbol{a}}^T - \boldsymbol{A}_{\boldsymbol{x}}^T \boldsymbol{W}_n^m\right)\boldsymbol{B} \tag{25}$$

where $\boldsymbol{Y_a}$ are the respectiv labels of additional $\boldsymbol{X_a}$.

Similarly, the input nodes updating algorithm is shown in Algorithm 3. And the network illustration is checked in Fig. 7. Again, this incremental learning saves time for only computing necessary pseudoinverse. This particular scheme is perfect for incremental learning for new incoming input data.

Remark 1: So far a general framework of the proposed BLS is presented, while the selection of functions for the feature mapping deserves attention. Theoretically, functions $\phi_i(\cdot)$ have no explicit restrictions, which means that the common choices such as kernel mappings, nonlinear transformations, or convolutional functions are acceptable. Specifically, if the function in the feature mapping uses convolutional functions, the broad learning network structure is very similar to that of classical CNN structure except that the broad learning network has additional connecting links between the convolutional layers and the output layer. Furthermore, additional connections, either laterally or stacking, among the feature nodes can be explored.

Remark 2: The proposed BLS is constructed based on the characteristics of the flatted functional-link networks. In general, such flatted functional extensions and incremental learning algorithms could be used in various networks such as support vector machine or RBF. The pseudoinverse computation is used here. This can be replaced with an iterative algorithm if desired. Gradient descent approach can be used to find the weights of enhancement nodes as well if it is desired.

F. Broad Learning: Structure Simplification and Low-Rank Learning

After the broad expansion with added mapped features and enhancement nodes via incremental learning, the structure may have a risk of being redundant due to poor initialization or redundancy in the input data.

Generally, the structure can be simplified by a series of low-rank approximation methods. In this section, we adapt the classical SVD as a conservative choice to offer the structure simplification for the proposed broad model.

The simplification can be done in different ways: 1) during the generation of mapped features; 2) during the generation of enhancement nodes; or 3) in the completion of broad learning.

1) SVD Simplification of Mapping Features: Let us begin with the random initial network with n groups of feature nodes that can be represented as the equation of the following form:

$$\begin{aligned}\boldsymbol{Y} &= [\phi(\boldsymbol{X}\boldsymbol{W}_{e_1} + \boldsymbol{\beta}_{e_1}), \ldots, \phi(\boldsymbol{X}\boldsymbol{W}_{e_n} + \boldsymbol{\beta}_{e_n})]\boldsymbol{W}_n^0 \\ &= [\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_n]\boldsymbol{W}_n^0.\end{aligned}$$

Similarly, from the previous sections, we denote by $\boldsymbol{A}_n^0 = [\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_n]$, which yields

$$\boldsymbol{Y} = \boldsymbol{A}_n^0 \boldsymbol{W}_n^0.$$

To explore the characteristics of the matrix $\boldsymbol{A}_n^0$, we apply SVD to $\boldsymbol{Z}_i$, $i = 1, \ldots, n$ as follows:

$$\boldsymbol{Z}_i = \boldsymbol{U}_{\boldsymbol{Z}_i} \boldsymbol{\Sigma}_{\boldsymbol{Z}_i}^P {\boldsymbol{V}_{\boldsymbol{Z}_i}}^T \tag{26}$$

$$= \boldsymbol{U}_{\boldsymbol{Z}_i} \cdot \left[\boldsymbol{\Sigma}_{\boldsymbol{Z}_i}^P | \boldsymbol{\Sigma}_{\boldsymbol{Z}_i}^Q\right] \cdot \left[\boldsymbol{V}_{\boldsymbol{Z}_i}^P | \boldsymbol{V}_{\boldsymbol{Z}_i}^Q\right]^T \tag{27}$$

$$= \boldsymbol{U}_{\boldsymbol{Z}_i} \boldsymbol{\Sigma}_{\boldsymbol{Z}_i}^P {\boldsymbol{V}_{\boldsymbol{Z}_i}^P}^T + \boldsymbol{U}_{\boldsymbol{Z}_i} \boldsymbol{\Sigma}_{\boldsymbol{Z}_i}^Q {\boldsymbol{V}_{\boldsymbol{Z}_i}^Q}^T = \boldsymbol{Z}_i^P + \boldsymbol{Z}_i^Q \tag{28}$$

Algorithm 3 Broad Learning: Increment of Feature-Mapping Nodes, Enhancement Nodes, and New Inputs

Input : training sample $\boldsymbol{X}$;
Output: $\boldsymbol{W}$
for $i = 0; i \leq n$ **do**
  Random $\boldsymbol{W}_{e_i}$, $\boldsymbol{\beta}_{e_i}$;
  Calculate $\boldsymbol{Z}_i = [\phi(\boldsymbol{X}\boldsymbol{W}_{e_i} + \boldsymbol{\beta}_{e_i})]$;
end
Set the feature mapping group $\boldsymbol{Z}^n = [\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_n]$;
for $j = 1; j \leq m$ **do**
  Random $\boldsymbol{W}_{h_j}$, $\boldsymbol{\beta}_{h_j}$;
  Calculate $\boldsymbol{H}_j = [\xi(\boldsymbol{Z}^n\boldsymbol{W}_{h_j} + \boldsymbol{\beta}_{h_j})]$;
end
Set the enhancement nodes group $\boldsymbol{H}^m = [\boldsymbol{H}_1, \ldots, \boldsymbol{H}_m]$;
Set $\boldsymbol{A}_n^m$ and calculate $(\boldsymbol{A}_n^m)^+$ with Eq. (3);
while *The training error threshold is not satisfied* **do**
  if *p enhancement nodes are added* **then**
    Random $\boldsymbol{W}_{h_{m+1}}$, $\boldsymbol{\beta}_{h_{m+1}}$;
    Calculate $\boldsymbol{H}_{m+1} = [\xi(\boldsymbol{Z}^n\boldsymbol{W}_{h_{m+1}} + \boldsymbol{\beta}_{h_{m+1}})]$; Update $\boldsymbol{A}_n^{m+1}$;
    Calculate $(\boldsymbol{A}_n^{m+1})^+$ and $\boldsymbol{W}_n^{m+1}$ by Eq. (13,14,15);
    $m = m + 1$;
  else
    if $n + 1$ *feature mapping is added* **then**
      Random $\boldsymbol{W}_{e_{n+1}}$, $\boldsymbol{\beta}_{e_{n+1}}$;
      Calculate $\boldsymbol{Z}_{n+1} = [\phi(\boldsymbol{X}\boldsymbol{W}_{e_{n+1}} + \boldsymbol{\beta}_{e_{n+1}})]$;
      Random $\boldsymbol{W}_{ex_i}$, $\boldsymbol{\beta}_{ex_i}$, $i = 1, \ldots, m$;
      Calculate $\boldsymbol{H}_{ex_{n+1}} = [\xi(\boldsymbol{Z}_{n+1}\boldsymbol{W}_{ex_i} + \boldsymbol{\beta}_{ex_i}), \ldots, \xi(\boldsymbol{Z}_{n+1}\boldsymbol{W}_{ex_m} + \boldsymbol{\beta}_{ex_m})]$;
      Update $\boldsymbol{A}_{n+1}^m$;
      Update $(\boldsymbol{A}_{n+1}^m)^+$ and $\boldsymbol{W}_{n+1}^m$ by Eq. (18,19,20);
      $n = n + 1$;
    else
      New inputs are added as $\boldsymbol{X_a}$;
      calculate $\boldsymbol{A_x}$ by (21),(22), update ${}^x\boldsymbol{A}_n^m$;
      update $({}^x\boldsymbol{A}_n^m)^+$ and $({}^x\boldsymbol{W}_n^m)$ by Eq. (23,24,25);
    end
  end
end
Set $\boldsymbol{W}$;

where $\boldsymbol{\Sigma}^P$ and $\boldsymbol{\Sigma}^Q$ are divided by the order of singularities, under the parameter ε.

Remember that our motivation is to achieve a satisfactory reduction of the numbers of nodes. The idea is to compress $\boldsymbol{Z}_i$ by the principal portion, $\boldsymbol{Z}_i^P$. The equation between $\boldsymbol{Z}_i$ and $\boldsymbol{Z}_i^P$ is derived as follows:

$$\begin{aligned}\boldsymbol{Z}_i^P\boldsymbol{V}_{\boldsymbol{Z}_i}^P &= \boldsymbol{U}_{\boldsymbol{Z}_i}\boldsymbol{\Sigma}_{\boldsymbol{Z}_i}^P{\boldsymbol{V}_{\boldsymbol{Z}_i}^P}^T\boldsymbol{V}_{\boldsymbol{Z}_i}^P \\ &= \boldsymbol{U}_{\boldsymbol{Z}_i}\boldsymbol{\Sigma}_{\boldsymbol{Z}_i}^P{\boldsymbol{V}_{\boldsymbol{Z}_i}^P}^T\boldsymbol{V}_{\boldsymbol{Z}_i}^P + \boldsymbol{U}_{\boldsymbol{Z}_i}\boldsymbol{\Sigma}_{\boldsymbol{Z}_i}^P{\boldsymbol{V}_{\boldsymbol{Z}_i}^Q}^T\boldsymbol{V}_{\boldsymbol{Z}_i}^P \\ &= \boldsymbol{U}_{\boldsymbol{Z}_i}\cdot\left[\boldsymbol{\Sigma}_{\boldsymbol{Z}_i}^P|\boldsymbol{\Sigma}_{\boldsymbol{Z}_i}^Q\right]\cdot\left[\boldsymbol{V}_{\boldsymbol{Z}_i}^P|\boldsymbol{V}_{\boldsymbol{Z}_i}^Q\right]^T\cdot\boldsymbol{V}_{\boldsymbol{Z}_i}^P \\ &= \boldsymbol{Z}_i\boldsymbol{V}_{\boldsymbol{Z}_i}^P.\end{aligned}$$

As for the original model, define

$$\boldsymbol{W}_n^0 \triangleq \left[\boldsymbol{W}_{\boldsymbol{Z}_1}^{\{0,n\}}|\ldots|\boldsymbol{W}_{\boldsymbol{Z}_n}^{\{0,n\}}\right]^T$$

we have that

$$\begin{aligned}\boldsymbol{Y} &= \boldsymbol{A}_n^0\boldsymbol{W}_n^0 \\ &= [\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_n]\boldsymbol{W}_n^0 \\ &= \boldsymbol{Z}_1\boldsymbol{W}_{\boldsymbol{Z}_1}^{\{0,n\}} + \cdots + \boldsymbol{Z}_n\boldsymbol{W}_{\boldsymbol{Z}_n}^{\{0,n\}} \\ &= \boldsymbol{Z}_1\boldsymbol{V}_{\boldsymbol{Z}_1}^P{\boldsymbol{V}_{\boldsymbol{Z}_1}^P}^T\boldsymbol{W}_{\boldsymbol{Z}_1}^{\{0,n\}} + \cdots + \boldsymbol{Z}_n\boldsymbol{V}_{\boldsymbol{Z}_n}^P{\boldsymbol{V}_{\boldsymbol{Z}_n}^P}^T\boldsymbol{W}_{\boldsymbol{Z}_n}^{\{0,n\}} \\ &= \left[\boldsymbol{Z}_1\boldsymbol{V}_{\boldsymbol{Z}_1}^P, \ldots, \boldsymbol{Z}_n\boldsymbol{V}_{\boldsymbol{Z}_n}^P\right]\begin{bmatrix}{\boldsymbol{V}_{\boldsymbol{Z}_1}^P}^T\boldsymbol{W}_{\boldsymbol{Z}_1}^{\{0,n\}} \\ \cdots \\ {\boldsymbol{V}_{\boldsymbol{Z}_n}^P}^T\boldsymbol{W}_{\boldsymbol{Z}_n}^{\{0,n\}}\end{bmatrix} \\ &= \boldsymbol{A}_F^{\{0,n\}}\boldsymbol{W}_F^{\{0,n\}}\end{aligned}$$

where

$$\boldsymbol{W}_F^{\{0,n\}} = \begin{bmatrix}{\boldsymbol{V}_{\boldsymbol{Z}_1}^P}^T\boldsymbol{W}_{\boldsymbol{Z}_1}^{\{0,n\}} \\ \cdots \\ {\boldsymbol{V}_{\boldsymbol{Z}_n}^P}^T\boldsymbol{W}_{\boldsymbol{Z}_n}^{\{0,n\}}\end{bmatrix}.$$

Finally, by solving a least square linear equation, the model is refined to

$$\boldsymbol{Y} = \boldsymbol{A}_F^{\{0,n\}}\boldsymbol{W}_F^{\{0,n\}} \tag{29}$$

where

$$\boldsymbol{W}_F^{\{0,n\}} = \left(\boldsymbol{A}_F^{\{0,n\}}\right)^+\boldsymbol{Y}. \tag{30}$$

Here, $(\boldsymbol{A}_F^{\{0,n\}})^+$ is the pseudoinverse of $\boldsymbol{A}_F^{\{0,n\}}$. In this way, the original $\boldsymbol{A}_n^0$ is simplified to $\boldsymbol{A}_F^{\{0,n\}}$.

2) SVD Simplification of Enhancement Nodes: We are able to simplify the structure after adding a new group of enhancement nodes to the network. Suppose that the n groups of feature mapping nodes and m groups of enhancement nodes have been added, and the network is

$$\boldsymbol{Y} = \boldsymbol{A}_F^{\{m,n\}}\boldsymbol{W}_F^{\{m,n\}}$$

where

$$\boldsymbol{A}_F^{\{m,n\}} = \left[\boldsymbol{Z}_1\boldsymbol{V}_{\boldsymbol{Z}_1}^P, \ldots, \boldsymbol{Z}_n\boldsymbol{V}_{\boldsymbol{Z}_n}^P|\boldsymbol{H}_1\boldsymbol{V}_{\boldsymbol{H}_1}^P, \ldots, \boldsymbol{H}_m\boldsymbol{V}_{\boldsymbol{H}_m}^P\right]$$

and

$$\boldsymbol{H}_j = \xi\left(\left[\boldsymbol{Z}_1\boldsymbol{V}_{\boldsymbol{Z}_1}^P, \ldots, \boldsymbol{Z}_n\boldsymbol{V}_{\boldsymbol{Z}_n}^P\right]\boldsymbol{W}_{h_j} + \boldsymbol{\beta}_{h_j}\right).$$

In the above equations, $\boldsymbol{H}_j^P$s, $j = 1, \ldots, m$, are obtained by the same way as $\boldsymbol{Z}_i^P$, which means

$$\boldsymbol{H}_j = \boldsymbol{U}_{\boldsymbol{H}_j}\boldsymbol{\Sigma}_{\boldsymbol{H}_j}^P{\boldsymbol{V}_{\boldsymbol{H}_j}}^T \tag{31}$$

$$= \boldsymbol{U}_{\boldsymbol{H}_j}\cdot\left[\boldsymbol{\Sigma}_{\boldsymbol{H}_j}^P|\boldsymbol{\Sigma}_{\boldsymbol{H}_j}^Q\right]\cdot\left[\boldsymbol{V}_{\boldsymbol{H}_j}^P|\boldsymbol{V}_{\boldsymbol{H}_j}^Q\right]^T \tag{32}$$

$$= \boldsymbol{U}_{\boldsymbol{H}_j}\boldsymbol{\Sigma}_{\boldsymbol{H}_j}^P{\boldsymbol{V}_{\boldsymbol{H}_j}^P}^T + \boldsymbol{U}_{\boldsymbol{H}_j}\boldsymbol{\Sigma}_{\boldsymbol{H}_j}^Q{\boldsymbol{V}_{\boldsymbol{H}_j}^Q}^T = \boldsymbol{H}_j^P + \boldsymbol{H}_j^Q. \tag{33}$$

Similarly, the simplified structure is obtained by substituting $\boldsymbol{H}_j$ by $\boldsymbol{H}_j\boldsymbol{V}_{\boldsymbol{H}_j}^P$.

Algorithm 4 Broad Learning: SVD Structure Simplification

Input : training sample $\boldsymbol{X}$; threshold ε_e, ε_h, ε
Output: $\boldsymbol{W}$

for $i = 1; i \leq n$ **do**
  Random $\boldsymbol{W}_{e_i}$, $\boldsymbol{\beta}_{e_i}$;
  Calculate $\boldsymbol{Z}_i = [\phi(\boldsymbol{X}\boldsymbol{W}_{e_i} + \boldsymbol{\beta}_{e_i})]$;
  Calculate $\boldsymbol{V}_{\boldsymbol{Z}_i}^P$ by SVD under threshold ε_e;
end
for $j = 1; j \leq m$ **do**
  Random $\boldsymbol{W}_{h_j}$, $\boldsymbol{\beta}_{h_j}$;
  Calculate $\boldsymbol{H}_j = [\xi([\boldsymbol{Z}_1\boldsymbol{V}_{\boldsymbol{Z}_1}^P, \ldots, \boldsymbol{Z}_n\boldsymbol{V}_{\boldsymbol{Z}_n}^P]\boldsymbol{W}_{h_j} + \boldsymbol{\beta}_{h_j})]$;
  Calculate $\boldsymbol{V}_{\boldsymbol{H}_j}^P$ by SVD under threshold ε_h;
end
Set $\boldsymbol{A}^{\{m,n\}} = [\boldsymbol{Z}_1\boldsymbol{V}_{\boldsymbol{Z}_1}^P, \ldots, \boldsymbol{Z}_n\boldsymbol{V}_{\boldsymbol{Z}_n}^P | \boldsymbol{H}_1\boldsymbol{V}_{\boldsymbol{H}_1}^P, \ldots, \boldsymbol{H}_m\boldsymbol{V}_{\boldsymbol{H}_m}^P]$;
calculate $(\boldsymbol{A}^{\{m,n\}})^+$ with Eq. (3);
while *The training error threshold is not satisfied* **do**
  Random $\boldsymbol{W}_{h_{m+1}}$, $\boldsymbol{\beta}_{h_{m+1}}$;
  Calculate $\boldsymbol{H}_{m+1} = [\xi([\boldsymbol{Z}_1\boldsymbol{V}_{\boldsymbol{Z}_1}^P, \ldots, \boldsymbol{Z}_n\boldsymbol{V}_{\boldsymbol{Z}_n}^P]\boldsymbol{W}_{h_{m+1}} + \boldsymbol{\beta}_{h_{m+1}})]$;
  Calculate $\boldsymbol{V}_{\boldsymbol{H}_{m+1}}^P$ by SVD under threshold ε_h;
  Update $\boldsymbol{A}^{\{m+1,n\}}$;
  Calculate $(\boldsymbol{A}_F^{\{m+1,n\}})^+$ and $\boldsymbol{W}_F^{\{m+1,n\}}$ by Eq. (34-36);
  $m = m + 1$;
end
Get $\boldsymbol{V}_F^P$ by SVD under threshold ε;
Calculate $\boldsymbol{A}_F$;
Calculate ${\boldsymbol{A}_F}^+$ and $\boldsymbol{W}_F$ by Eq. (3);
Set $\boldsymbol{W} = \boldsymbol{W}_F$;

3) SVD Simplification of Inserting Additional p Enhancement Nodes: Without loss of generality, based on the above assumptions, we deduce an SVD simplification for inserting additional p enhancement nodes as follows:

$$\boldsymbol{A}^{\{m+1,n\}} = \left[\boldsymbol{A}_F^{\{m,n\}} | \boldsymbol{H}_{m+1}\right]$$

where $\boldsymbol{H}_{m+1} = \xi([\boldsymbol{Z}_1\boldsymbol{V}_{\boldsymbol{Z}_1}^P, \ldots, \boldsymbol{Z}_n\boldsymbol{V}_{\boldsymbol{Z}_n}^P]\boldsymbol{W}_{h_{m+1}} + \boldsymbol{\beta}_{h_{m+1}})$.

Similarly, as the SVD implement in the previous steps, we have that

$$\boldsymbol{A}_F^{\{m+1,n\}} = \left[\boldsymbol{A}_F^{\{m,n\}} | \boldsymbol{H}_{m+1}\boldsymbol{V}_{\boldsymbol{H}_{m+1}}^P\right].$$

To update the pseudoinverse of $\boldsymbol{A}_F^{\{m+1,n\}}$, similar to (13)–(15), we conclude that

$$\left(\boldsymbol{A}_F^{\{m+1,n\}}\right)^+ = \begin{bmatrix} \left(\boldsymbol{A}_F^{\{m,n\}}\right)^+ - \boldsymbol{D}\boldsymbol{B}^T \\ \boldsymbol{B}^T \end{bmatrix} \tag{34}$$

where $\boldsymbol{D} = (\boldsymbol{A}_F^{\{m,n\}})^+ \boldsymbol{H}_{m+1}\boldsymbol{V}_{\boldsymbol{H}_{m+1}}^P$

$$\boldsymbol{B}^T = \begin{cases} (\boldsymbol{C})^+ & \text{if } \boldsymbol{C} \neq 0 \\ (\boldsymbol{1} + \boldsymbol{D}^T\boldsymbol{D})^{-1}\boldsymbol{D}^T\left(\boldsymbol{A}_F^{\{m,n\}}\right)^+ & \text{if } \boldsymbol{C} = 0 \end{cases} \tag{35}$$

and $\boldsymbol{C} = \boldsymbol{H}_{m+1}\boldsymbol{V}_{\boldsymbol{H}_{m+1}}^P - \boldsymbol{A}_F^{\{m,n\}}\boldsymbol{D}$. The new weights are

$$\boldsymbol{W}_F^{\{m+1,n\}} = \begin{bmatrix} \boldsymbol{W}_F^{\{m,n\}} - \boldsymbol{D}\boldsymbol{B}^T\boldsymbol{Y} \\ \boldsymbol{B}^T\boldsymbol{Y} \end{bmatrix}. \tag{36}$$

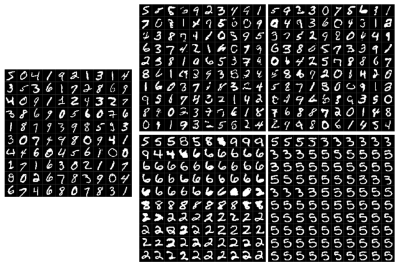

Fig. 8. MNIST data set [51].

Here, $\boldsymbol{W}_F^{\{m+1,n\}}$ is the least square solution of the following model:

$$\boldsymbol{Y} = \boldsymbol{A}_F^{\{m+1,n\}}\boldsymbol{W}_F^{\{m+1,n\}}.$$

4) SVD Simplification of Completion of Broad Learning: Now, it seems that a complete network is built; however, there is a need to simplify more. A possible solution is to cut off more small singular values components.

Therefore, we have that

$$\begin{aligned} \boldsymbol{A}_F^{\{m,n\}} &= \boldsymbol{U}_F\boldsymbol{\Sigma}_F\boldsymbol{V}_F^T \\ &= \boldsymbol{U}_F \cdot \left[\boldsymbol{\Sigma}_F^P | \boldsymbol{\Sigma}_F^Q\right] \cdot \left[\boldsymbol{V}_F^P | \boldsymbol{V}_F^Q\right]^T \\ &= \boldsymbol{U}_F\boldsymbol{\Sigma}_F^P{\boldsymbol{V}_F^P}^T + \boldsymbol{U}_F\boldsymbol{\Sigma}_F^Q{\boldsymbol{V}_F^Q}^T \\ &= \boldsymbol{A}_F^{\{m,n\}^P} + \boldsymbol{A}_F^{\{m,n\}^Q}. \end{aligned}$$

Similar to the beginning of the algorithm, set

$$\boldsymbol{A}_F = \boldsymbol{A}_F^{\{m,n\}}\boldsymbol{V}_F^P$$

we have an approximation of the matrix as follows:

$$\boldsymbol{Y} = \boldsymbol{A}_F\boldsymbol{W}_F \tag{37}$$

where

$$\boldsymbol{W}_F = {\boldsymbol{A}_F}^+\boldsymbol{Y}. \tag{38}$$

Finally, the structure simplified broad learning algorithm is given in Algorithm 4. Generally, the number of the neural nodes could be significantly reduced depending on the threshold values ε_e, ε_h, and ε, for simplifying the feature mapping nodes, the enhancement nodes, and the final structure, respectively.

IV. Experiment and Discussion

In this section, experimental results are given to verify the proposed system. To confirm the effectiveness of the proposed system, classification experiments are applied on popular MNIST and NORB data. To prove the effectiveness of BLS, we will compare the classification ability of our method with existing mainstream methods, including stacked auto encoders (SAE) [6], another version of stacked autoencoder (SDA) [52], DBN [5], multilayer perceptron-based methods (MLP) [53], DBM [7], two kinds of extremely learning machine (ELM)-based multilayer structure, which denoted as MLELM [54] and HELM [10], respectively. The above comparing experiments are tested on MATLAB software platform under a laptop

TABLE I
CLASSIFICATION ACCURACY ON MNIST DATA SET

Method	Accuracy (%)	Traing time (s)
SAE	98.60	36448.40
SDA	98.72	37786.03
DBN	98.87	53219.77
DBM	99.05	121455.69
MLP	97.39	21468.12
MLELM	99.04	475.83
HELM	99.13	281.37
CNN	95.63	21793.93*
FRBM	97.44	577.8
BL	98.74	78.6833
BL	98.74	29.9157*

TABLE II
CLASSIFICATION ACCURACY ON MNIST DATA SET WITH DIFFERENT NUMBERS OF ENHANCEMENT NODES

Feature nodes	Enhancement nodes	Accuracy (%)
100	4000	98.19
100	6000	98.47
100	8000	98.55
100	9500	98.59
100	10500	98.59
100	11000	98.74
100	12000	98.67
200	11000	98.69
400	11000	98.61
1000	11000	98.53
2000	11000	98.45

that equips with Intel-i7 2.4 GHz CPU, 16 GB memory. The classification results of the above methods are cited from [10].

Furthermore, we compare our results with an extended fuzzy restricted Boltzmann machine (FRBM) [13] which offers a more reasonable, solid, and theoretical foundation for establishing FRBMs. The one-layer FRBM and our proposed broad learning results are tested on a 3.40-GHz Intel i7-6700 CPU processor PC with MATLAB platform. It should be noted that duplicate experiments are tested in the sever computer equips with 2.30 GHz Intel Xeon E5-2650 CPU processor and the testing accuracy and the training time are given with a special superscript $*$.

A state-of-art deep CNN [3] has achieved extremely good results even on the ImageNet challenge. However, this is generally done with the help of a very deep structure or the ensembles of various operations. In this paper, only the comparison of the original deep CNN (LeNet5) in [22] is presented here for fair comparison because the proposed BLS only uses linear feature mapping. Related experiments about CNN are tested in the above server computer on Theano platform.

Generally, all the methods we mentioned above, except for HELM and MLELM, are deep structures and the hyper-parameters are tuned based on the back propagation whose initial learning rates are set as 0.1 and the decay rate for each learning epoch is set as 0.95. As for the ELM-based networks, the regularization parameters of MLELM are set as 10^{-1}, 10^3, and 10^8, respectively, while the penalty parameter of HELM is 10^8. Other detailed parameters could be checked in [10]. In our proposed BLS, the regularization parameter λ for ridge regression is set as 10^{-8}, meanwhile, the one layer linear feature mapping with one step fine-tune to emphasize the selected features is adopted. In addition, the associated parameters W_{e_i} and β_{e_i}, for $i = 1, \ldots, n$ are drawn from the standard uniform distributions on the interval $[-1, 1]$. For the enhancement nodes, the sigmoid function is chosen to establish BLS. At the end of this section, experimental results based on the proposed broad learning algorithms are given.

A. MNIST Data

In this section, a series of experiments is focused on the classical MNIST handwritten digital images [8]. This data set consists of 70 000 handwritten digits partitioned into a training set containing 60 000 samples and a test set of 10 000 samples. Every digit is represented by an image with the size of 28×28 gray-scaled pixels. Typically, images are shown in Fig. 8.

To test the accuracy and efficiency of our proposed broad learning structures in the classification, we give *a priori* knowledge about the numbers of feature nodes and enhancement nodes. However, this is exactly the normal practice for building the network in the deep learning neural networks, which is in fact the most challenging task of the whole learning process. In our experiments, the network is constructed by total 10×10 feature nodes and $1 \times 11\,000$ enhancement nodes. For reference, the deep structures of SAE, DBN, DBM, MLELM, and HELM is $1000 - 500 - 25 - 30$, $500 - 500 - 2000$, $500-500-1000$, $700-700-15\,000$, and $300-300-12\,000$, respectively. The testing accuracies of mentioned methods and our proposed method are shown in Table I. Although the 98.74% is not the best one, (in fact, the performance of the broad learning is still better than SAE and MLP), the training time in the server is very fast at a surprising level which is 29.9157 seconds, while the testing time in the server is 1.0783 s. Moreover, it should be noticed that the number of feature mapping nodes is only 100 here. This result accords with scholar's intuition in large scale learning that the information in the practical applications is usually redundancy. More results with different mapped features are given in Table II.

Next, we will show the fast and effectiveness of the incremental learning system. And the associated experiments are implemented in the sever computer mentioned above. Let the final structure consists of 100 feature nodes and 11 000 enhancement nodes. Two different initial networks are used to test the incremental learning here.

First, suppose the initial network is set as 10×10 feature nodes and 9000 enhancement nodes. Then, Algorithm 1 is applied to dynamically increase 500 enhancement nodes each time until it reaches 11 000.

Second, three dynamic increments are used here to increase dynamically: 1) the feature nodes; 2) the corresponding enhancement nodes; and 3) the additional enhancement nodes. Scenario is shown in Fig. 6. The network is initially set to have 10×6 feature nodes and 7000 enhancement nodes at the beginning of the Algorithm 3. Then, the feature nodes are dynamically increased from 60 to 100 at the step of 10 in each

TABLE III
CLASSIFICATION ACCURACY ON MNIST DATA SET USING INCREMENTAL LEARNING

Method	Number of Feature Nodes	Number of Enhancement Nodes	Testing Accuracy (%)	Training Time (s)	Testing Time (s)
BL	100	11000	98.74	29.9157*	1.0783*
IBL	100	9000 $\longrightarrow$ 11000 500	98.66	87.4836*	1.8588*
IBL	60 $\longrightarrow$ 100 10	7000 $\longrightarrow$ 11000 (250+750)	98.77	93.7444*	1.6371*

TABLE IV
SNAPSHOT RESULTS OF MNIST CLASSIFICATION USING INCREMENTAL LEARNING

Number of Feature Nodes	Number of Enhancement Nodes	Testing Accuracy (%)	Each Additional Training Times (s)	Accumulative Training Times (s)	Each Additional Testing Times (s)	Accumulative Testing Times (s)
60	3000	97.83	10.7010*	10.7010*	0.5093*	0.5093*
60$\longrightarrow$70 10	3000$\longrightarrow$ 5000 (750+1250)	98.30	13.8580*	24.5590*	0.3101*	0.8194*
70$\longrightarrow$80 10	5000 $\longrightarrow$7000 (750+1250)	98.48	18.2420*	42.8010*	0.4066*	1.2260*
80 $\longrightarrow$90 10	7000 $\longrightarrow$9000 (750+1250)	98.55	22.2968*	65.0978*	0.2914*	1.5174*
90$\longrightarrow$ 100 10	9000$\longrightarrow$ 11000 (750+1250)	98.73	29.5685*	94.6663*	0.2448*	1.7622*

update, the corresponding enhancement nodes for the additional features are increased at 250 each, and the additional enhancement nodes are increased at 750 each. Or equivalently, at each step, 10 feature nodes and 1000 enhancement nodes are inserted to the network. Table III presents the performance of the above two different dynamic constructions for MNIST classification compared with the results in Table I. The incremental versions have the similar performance as the one-shot construction. It is surprising that the dynamic increments on both feature nodes and enhancement nodes perform the best. This may be caused by the randomness nature of the feature nodes and the enhancement nodes. This implies that dynamic update of the model using incremental learning could present a compatible result; meanwhile, it provides the opportunities to adjust structure and accuracy for the system to match the desired performance.

Additional experiment is designed to test the elapse time of the incremental learning. The initial network is set as 10×6 feature nodes and 3000 enhancement nodes. Similarly as the previous case, the feature nodes are dynamically increased from 60 to 100 at the step of 10 in each update, the corresponding enhancement nodes for the additional features are increased at 750 each, and the additional enhancement nodes are increased at 1250 each. The training times and results of each update are presented in Table IV. The result is very competitive to that of the one-shot solution when the network reaches 100 feature nodes and 11 000 enhancement nodes as shown in Table I. It is proven that the incremental learning algorithms are very effective.

Finally, incremental broad learning algorithms on added inputs are tested. On the one hand, suppose the initial network is trained under the first 10 000 training samples in the MNIST data set. Then, incremental algorithm is applied to add dynamically 10 000 input patterns each time until all the 60 000 training samples are fed. The structure of the tested BLS is set as 10×10 feature nodes and 5000 enhancement nodes, and the snapshot results of each update are shown in Table V. On the other hand, experiments for the increment of input patterns and enhancement nodes are tested. First, the network is initially set to have 10×10 feature nodes and 5000 enhancement nodes. Then, the additional enhancement nodes are increased dynamically at 250 each, and the additional input input patterns are increased at 10 000 each. The attractive results of each update could be checked in Table VI.

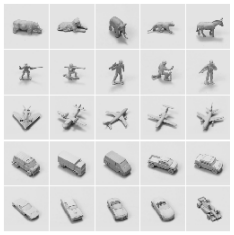

Fig. 9. Examples for training figures.

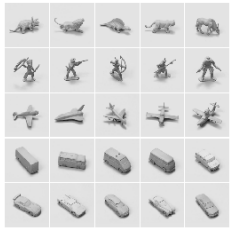

Fig. 10. Examples for testing figures.

B. NORB Data

NORB data set [55] is a more complicated data set compared with MNIST data set; in a total of 48 600 images, each has $2 \times 32 \times 32$ pixels. The NORB contains images of 50 different 3-D toy objects belonging to five distinct categories: 1) animals; 2) humans; 3) airplanes; 4) trucks; and 5) cars,

TABLE V

SNAPSHOT RESULTS OF MNIST CLASSIFICATION USING INCREMENTAL LEARNING: INCREMENT OF INPUT PATTERNS

Number of Input Patterns	Stucture	Testing Accuracy (%)	Each Additional Training Time (s)	Accumulative Training Time (s)	Each Additional Testing Time (s)	Accumulative Testing Time (s)
60000	(100,5000)	98.34	10.3715*	10.3715*	0.7116*	0.7116*
10000	(100,5000)	97.02	4.0495*	4.0495*	0.6675*	0.6675*
$\underset{10000}{10000\longrightarrow 20000}$	(100,5000)	97.72	4.5931*	8.6429*	0.0564*	0.7239*
$\underset{10000}{20000\longrightarrow 30000}$	(100,5000)	97.94	4.8011*	13.4437*	0.0658*	0.7897*
$\underset{10000}{30000\longrightarrow 40000}$	(100,5000)	98.07	4.6459*	18.0896*	0.0638*	0.8535*
$\underset{10000}{40000\longrightarrow 50000}$	(100,5000)	98.11	4.8745*	22.9641*	0.0560*	0.9095*
$\underset{10000}{50000\longrightarrow 60000}$	(100,5000)	98.25	5.9136*	28.8777*	0.0643*	0.9738*

TABLE VI

SNAPSHOT RESULTS OF MNIST CLASSIFICATION USING INCREMENTAL LEARNING: INCREMENT OF INPUT PATTERNS AND ENHANCEMENT NODES

Number of Feature Nodes	Number of Enhancement Nodes	Number of Input patterns	Testing Accuracy (%)	Each Additional Training Time (s)	Accumulative Training Time (s)	Each Additional Testing Time (s)	Accumulative Testing Time (s)
100	3000	10000	97.03	2.4725*	2.4725*	0.5404*	0.5404*
100	$\underset{1600}{3000\longrightarrow 4600}$	$\underset{10000}{10000\longrightarrow 20000}$	97.76	5.8443*	8.3168*	0.2942*	0.8346*
100	$\underset{1600}{4600\longrightarrow 6200}$	$\underset{10000}{20000\longrightarrow 30000}$	97.89	11.0399*	19.3567*	0.1978*	1.0324*
100	$\underset{1600}{6200\longrightarrow 7800}$	$\underset{10000}{30000\longrightarrow 40000}$	97.89	16.9018*	36.2585*	0.3645*	1.3969*
100	$\underset{1600}{7800\longrightarrow 9400}$	$\underset{10000}{40000\longrightarrow 50000}$	97.93	25.4579*	61.7164*	0.2090*	1.6059*
100	$\underset{1600}{9400\longrightarrow 11000}$	$\underset{10000}{50000\longrightarrow 60000}$	98.05	34.0481*	95.7645*	0.2060*	1.8119*

TABLE VII

CLASSIFICATION ACCURACY ON NORB DATA SET

Method	Accuracy (%)	Traing time (s)
SAE	86.28	60504.34
SDA	87.62	65747.69
DBN	88.47	87280.42
DBM	89.65	182183.53
MLP	84.20	34005.470
MLELM	88.91	775.2850
HELM	91.28	432.19
BL	89.27	41.4666
BL	89.27	21.2546*

as shown in Figs. 9 and 10. The sampled objects are imaged under various lighting conditions, elevations, and azimuths. The training set contains 24 300 stereo image of 25 objects (five per class) as shown in Fig. 9, while the testing set contains 24 300 image of the remaining 25 objects, as shown in Fig. 10. In our experiments, the network was constructed by the one-shot model which consists of 100×10 feature nodes and 1×9000 enhancement nodes. Compared with the deep and complex structure of DBN and HELM, which is $4000-4000-4000$ and $3000-3000-15000$, respectively, the proposed BLS presents faster training time. The testing results, shown in Table VII, present a pleasant performance, especially the training time of the proposed broad learning. Similar to the MNIST cases, although the accuracy is not the best one, the performance matches with the previous work with a testing time of 6.0299 s in the server. Considering the superfast speed in computation, which is the best among the existing methods, the proposed broad learning and network is very attractive.

C. SVD-Based Structure Simplification

In this part, we run simulations using SVD to simplify the structure after the model is constructed. The experiments are tested in MNIST data set. During the experiments, the threshold $\varepsilon_e = \varepsilon_h = 1$, and $\varepsilon = N$ are set, which means that there is no simplification on feature nodes and enhancement nodes generation, but only to keep the first N important principle components in the final simplified network, i.e., apply SVD operation to $A_F^{\{m,n\}}$. As shown in Table VIII, N is selected as 500, 600, 800, 1000, 1500, 2000, 2500, and 3000.

The Ω in table denotes the network structures of BLS, where the first is the number of feature nodes and the second is the number of the enhancement nodes. Or specifically, summation of the numbers lie in the Ω column is the total number of nodes in the broad network. In the "BLSVD" column of the table, SVD operation is applied to the network and the network is compressed to the desired N nodes. The tests are to compare with the Restrict Boltzmann Machine (RBM) and the original BLS. Parameters of RBM are referred to [5], i.e., the learning rate is 0.05 and the weight decay is 0.001. All of the three methods are repeated ten times. In the table, the minimal test error (MTE) and the average test error (ATE) under all the ten experiments are shown in percentage.

From the table we could obviously observe that when the number of nodes exceeds 1000, both the BLS models have better results than RBM. Moreover, the models selected by SVD significantly improve the accuracy. Specifically, the RBM is in fact trapped in around 3% accuracy no matter how many nodes are added to the network. In addition, the result of the proposed SVD-based broad learning varies in a narrow range than RBM.

TABLE VIII
NETWORK COMPRESSION RESULT USING SVD BROAD LEARNING ALGORITHM

Numbers of Nodes	RBM		BL			BLSVD		
	MTE (%)	ATE (%)	Ω	MTE(%)	ATE (%)	Ω	MTE(%)	ATE (%)
500	3.38	3.7560	(100, 400)	4.44	4.6624	(100, 11000)→500	4.06	4.1790
600	3.18	3.5800	(100, 500)	4.09	4.3200	(100, 11000)→600	3.72	3.8350
800	3.05	3.3350	(100, 700)	3.56	3.7240	(100, 11000)→800	3.21	3.4500
1000	2.98	3.2890	(100, 900)	3.17	3.3670	(100, 11000)→1000	2.96	3.1250
1500	3.00	3.2050	(100, 1400)	2.75	2.8210	(100, 11000)→1500	2.45	2.6300
2000	3.02	3.3210	(100, 1900)	2.33	2.4430	(100, 11000)→2000	2.20	2.3220
2500	2.92	3.0740	(100, 2400)	2.18	2.2600	(100, 11000)→2500	1.98	2.1180
3000	2.76	3.0900	(100, 2900)	1.98	2.0960	(100, 11000)→3000	1.87	1.9980

D. Performance Analysis

Based on the experiments above, BLS obviously outperforms the existing deep structure neural networks in terms of training speed. Furthermore, compared with other MLP training methods, BLS leads to a promising performance in classification accuracy and learning speed. Compared with hours or days for training and hundreds of epochs of iteration with high-performance computers in deep structure, the BLS can be easily constructed in a few minutes, even in a regular PC.

In addition, it should be mentioned that, from Tables III and IV, the incremental version of broad learning does not lose the accuracy of the classification, even better in the MNIST case.

Furthermore, the broad structure of our system could be simplified by applying series of low-rank approximations. In this paper, only the classical SVD method is discussed and compared with one-layer RBM, the proposed SVD-based broad learning is more stable. A fast structure reduction using different low-rank algorithms can be developed if the SVD is considered as not so efficient.

V. CONCLUSION

BLS is proposed in this paper, which aims to offer an alternative way for deep learning and structure. The establishment of the system is based on the idea of RVFLNN.

The designed model can be expanded in wide fashion when new feature nodes and enhancement nodes are needed. The corresponding incremental learning algorithms are also designed. The incremental learnings are developed for fast remodeling in broad expansion without a retraining process if the network deems to be expanded. From the incremental experimental results presented in Table IV, it is shown that the incremental learnings can rapidly update and remodel the system. It is observed that learning time of a one-shot structure is smaller than step-by-step increments version to reach the final structure. However, this incremental learning offers an approach for system remodeling and for model selection, especially in modeling high-volume time-varying systems.

The experiments on MNIST and NORB data confirm the dynamic update properties of the proposed BLS. Finally, the SVD approach is applied to simplify the structure. It is also indicated that the simplified networks demonstrate promising results.

Lastly, with proper arrangement in the feature nodes, the proposed broad learning algorithms and incremental learnings (see Section III) can be applied to a flat network or to a network that only needs to compute the connecting weights of the last layer, such as ELM.

ACKNOWLEDGMENT

Source codes of the paper can be found in the author's website at: http://www.fst.umac.mo/en/staff/pchen.html.

REFERENCES

[1] M. Gong, J. Zhao, J. Liu, Q. Miao, and L. Jiao, "Change detection in synthetic aperture radar images based on deep neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 1, pp. 125–138, Jan. 2016.

[2] W. Hou, X. Gao, D. Tao, and X. Li, "Blind image quality assessment via deep learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 6, pp. 1275–1286, Jun. 2015.

[3] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. New York, NY, USA: Curran Associates, Inc., 2012, pp. 1097–1105.

[4] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[5] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, pp. 0899–7667, May 2006.

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

[7] R. Salakhutdinov and G. E. Hinton, "Deep boltzmann machines," in *Proc. AISTATS*, vol. 1. 2009, p. 3.

[8] 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.

[9] K. Simonyan and A. Zisserman, (2014). "Very deep convolutional networks for large-scale image recognition." [Online]. Available: https://arxiv.org/abs/1409.1556

[10] J. Tang, C. Deng, and G.-B. Huang, "Extreme learning machine for multilayer perceptron," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 4, pp. 809–821, Apr. 2016.

[11] M. Chen, Z. E. Xu, K. Q. Weinberger, and F. Sha. (2012). "Marginalized denoising autoencoders for domain adaptation." [Online]. Available: https://arxiv.org/abs/1206.4683

[12] M. Gong, J. Liu, H. Li, Q. Cai, and L. Su, "A multiobjective sparse feature learning model for deep neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 12, pp. 3263–3277, Dec. 2015.

[13] S. Feng and C. L. P. Chen, "A fuzzy restricted Boltzmann machine: Novel learning algorithms based on crisp possibilistic mean value of fuzzy numbers," *IEEE Trans. Fuzzy Syst.*, to be published.

[14] C. L. P. Chen, C.-Y. Zhang, L. Chen, and M. Gan, "Fuzzy restricted Boltzmann machine for the enhancement of deep learning," *IEEE Trans. Fuzzy Syst.*, vol. 23, no. 6, pp. 2163–2173, Dec. 2015.

[15] Z. Yu, L. Li, J. Liu, and G. Han, "Hybrid adaptive classifier ensemble," *IEEE Trans. Cybern.*, vol. 45, no. 2, pp. 177–190, Feb. 2015.

[16] Z. Yu, H. Chen, J. Liu, J. You, H. Leung, and G. Han, "Hybrid K-nearest neighbor classifier," *IEEE Trans. Cybern.*, vol. 46, no. 6, pp. 1263–1275, Jun. 2016.

[17] Z. Yu *et al.*, "Incremental semi-supervised clustering ensemble for high dimensional data clustering," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 3, pp. 701–714, Mar. 2016.

[18] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Netw.*, vol. 6, no. 6, pp. 861–867, 1993.

[19] Y.-H. Pao and Y. Takefuji, "Functional-link net computing: Theory, system architecture, and functionalities," *Computer*, vol. 25, no. 5, pp. 76–79, May 1992.

[20] Y.-H. Pao, G.-H. Park, and D. J. Sobajic, "Learning and generalization characteristics of the random vector functional-link net," *Neurocomputing*, vol. 6, no. 2, pp. 163–180, 1994.

[21] B. Igelnik and Y.-H. Pao, "Stochastic choice of basis functions in adaptive function approximation and the functional-link net," *IEEE Trans. Neural Netw.*, vol. 6, no. 6, pp. 1320–1329, Nov. 1995.

[22] Y. LeCun *et al.*, "Handwritten digit recognition with a back-propagation network," in *Proc. Neural Inf. Process. Syst. (NIPS)*, 1990, pp. 396–404.

[23] J. S. Denker *et al.*, "Neural network recognizer for hand-written zip code digits," in *Advances in Neural Information Processing Systems*, D. S. Touretzky, Ed. San Francisco, CA, USA: Morgan Kaufmann, 1989, pp. 323–331.

[24] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," *IEEE Trans. Neural Netw.*, vol. 1, no. 1, pp. 4–27, Mar. 1990.

[25] I. Y. Tyukin and D. V. Prokhorov, "Feasibility of random basis function approximators for modeling and control," in *Proc. IEEE Control Appl. Intell. Control (ISIC) (CCA)*, Jul. 2009, pp. 1391–1396.

[26] C. L. P. Chen and C.-Y. Zhang, "Data-intensive applications, challenges, techniques and technologies: A survey on big data," *Inf. Sci.*, vol. 275, pp. 314–347, Aug. 2014.

[27] C. L. P. Chen and J. Z. Wan, "A rapid learning and dynamic stepwise updating algorithm for flat neural networks and the application to time-series prediction," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 29, no. 1, pp. 62–72, Feb. 1999.

[28] A. Rakotomamonjy, "Variable selection using SVM-based criteria," *J. Mach. Learn. Res.*, vol. 3, pp. 1357–1370, Mar. 2003.

[29] P. M. Narendra and K. Fukunaga, "A branch and bound algorithm for feature subset selection," *IEEE Trans. Comput.*, vol. 26, no. 9, pp. 917–922, Sep. 1977.

[30] J. Fan and R. Li, "Variable selection via nonconcave penalized likelihood and its oracle properties," *J. Amer. Statist. Assoc.*, vol. 96, no. 456, pp. 1348–1360, 2001.

[31] R. G. Baraniuk and M. B. Wakin, "Random projections of smooth manifolds," *Found. Comput. Math.*, vol. 9, no. 1, pp. 51–77, 2009.

[32] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.

[33] L. Grasedyck, D. Kressner, and C. Tobler, "A literature survey of low-rank tensor approximation techniques," *GAMM-Mitteilungen*, vol. 36, no. 1, pp. 53–78, 2013.

[34] I. Markovsky, "Low rank approximation," in *Algorithms, Implementation, Applications* (Communications and Control Engineering). London, U.K.: Springer, 2011.

[35] Z. Yang, Y. Xiang, K. Xie, and Y. Lai, "Adaptive method for nonsmooth nonnegative matrix factorization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 4, pp. 948–960, Apr. 2017.

[36] C. L. P. Chen, "A rapid supervised learning neural network for function interpolation and approximation," *IEEE Trans. Neural Netw.*, vol. 7, no. 5, pp. 1220–1230, Sep. 1996.

[37] C. Leonides, "Control and dynamic systems V18," in *Advances in Theory and Applications* (Control and dynamic systems). Amsterdam, The Netherlands: Elsevier, 2012.

[38] A. Ben-Israel and T. Greville, *Generalized Inverses: Theory and Applications*. New York, NY, USA: Wiley, 1974.

[39] C. R. Rao and S. K. Mitra, "Generalized inverse of a matrix and its applications," in *Proc. 6th Berkeley Symp. Math. Statist. Probab.*, vol. 1. 1972, pp. 601–620.

[40] D. Serre, "Matrices," in *Theory and Applications* (Graduate Texts in Mathematics). New York, NY, USA: Springer-Verlag, 2002.

[41] A. E. Hoerl and R. W. Kennard, "Ridge regression: Biased estimation for nonorthogonal problems," *Technometrics*, vol. 42, no. 1, pp. 80–86, Feb. 2000.

[42] Z. Xu, X. Chang, F. Xu, and H. Zhang, "$L_{1/2}$ regularization: A thresholding representation theory and a fast solver," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 7, pp. 1013–1027, Jul. 2012.

[43] W. Yang, Y. Gao, Y. Shi, and L. Cao, "MRM-lasso: A sparse multiview feature selection method via low-rank analysis," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 11, pp. 2801–2815, Nov. 2015.

[44] B. A. Olshausen and D. J. Field, "Sparse coding with an overcomplete basis set: A strategy employed by V1?" *Vis. Res.*, vol. 37, no. 23, pp. 3311–3325, 1997.

[45] R. Tibshirani, "Regression shrinkage and selection via the lasso," *J. Roy. Statist. Soc., B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.

[46] J. A. Tropp and A. C. Gilbert, "Signal recovery from random measurements via orthogonal matching pursuit," *IEEE Trans. Inf. Theory*, vol. 53, no. 12, pp. 4655–4666, Dec. 2007.

[47] M. Aharon, M. Elad, and A. Bruckstein, "*rmK*-SVD: An algorithm for designing overcomplete dictionaries for sparse representation," *IEEE Trans. Signal Process.*, vol. 54, no. 11, pp. 4311–4322, Nov. 2006.

[48] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, Jan. 2011.

[49] A. Beck and M. Teboulle, "A fast iterative shrinkage-thresholding algorithm for linear inverse problems," *SIAM J. Imag. Sci.*, vol. 2, no. 1, pp. 183–202, 2009.

[50] T. Goldstein and B. O'Donoghue, S. Setzer, and R. Baraniuk, "Fast alternating direction optimization methods," *SIAM J. Imag. Sci.*, vol. 7, no. 3, pp. 1588–1623, 2014.

[51] O. Breuleux, Y. Bengio, and P. Vincent, "Quickly generating representative samples from an RBM-derived process," *Neural Comput.*, vol. 23, no. 8, pp. 2058–2073, Aug. 2011.

[52] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders," in *Proc. 25th Int. Conf. Mach. Learn. (ICML)*, New York, NY, USA, 2008, pp. 1096–1103.

[53] C. M. Bishop, *Pattern Recognition and Machine Learning* (Information Science and Statistics). Secaucus, NJ, USA: Springer-Verlag, 2006.

[54] E. Cambria *et al.*, "Extreme learning machines [trends controversies]," *IEEE Intell. Syst.*, vol. 28, no. 6, pp. 30–59, Nov. 2013.

[55] 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 Recognit. (CVPR)*, vol. 2. Jun. 2004, pp. II-94–II-104.

C. L. Philip Chen (S'88–M'88–SM'94–F'07) received the M.S. degree in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 1985, and the Ph.D. degree in electrical engineering from Purdue University, West Lafayette, IN, USA, in 1988.

He was a tenured professor in the United States for 23 years. He is currently the Dean of the Faculty of Science and Technology, University of Macau, Macau, China, where he is the Chair Professor of the Department of Computer and Information Science. He is also the Department Head and an Associate Dean with two different universities. His current research interests include systems, cybernetics, and computational intelligence.

Dr. Chen is a fellow of AAAS, Chinese Association of Automation, and HKIE. He was the President of IEEE Systems, Man, and Cybernetics Society from 2012 to 2013. He has been the Editor-in-Chief of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS since 2014 and an Associate Editor of several IEEE Transactions. He is also the Chair of TC 9.1 Economic and Business Systems of International Federation of Automatic Control, and also an Accreditation Board of Engineering and Technology Education Program Evaluator for computer engineering, electrical engineering, and software engineering programs.

Zhulin Liu received the bachelor's degree in mathematics from Shandong University, Shandong, China in 2005, and the M.S. degree in mathematics from the University of Macau, Macau, China, in 2009, where she is currently pursuing the Ph.D. degree with the Faculty of Science and Technology.

Her current research interests include computational intelligence, matching learning, and function approximation.