Information Sciences 649 (2023) 119648

Contents lists available at ScienceDirect

Information Sciences

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

Pseudo inverse versus iterated projection: Novel learning approach and its application on broad learning system

Faliang Yin [a], Weiguo Li [b], Kai Zhang [c], Jian Wang [b,*], Nikhil R. Pal [d]

[a] Department of Engineering, King's College London, London, United Kingdom
[b] College of Science, China University of Petroleum (East China), Qingdao, China
[c] College of Civil Engineering, Qingdao University of Technology, Qingdao, China
[d] Electronics and Communication Sciences Unit and the Center for Artificial Intelligence and Machine Learning of Indian Statistical Institute, Calcutta, India

ARTICLE INFO

Keywords:
Broad learning system
Incremental learning
Kaczmarz method
Least squares estimation
Hyperparameter optimization

ABSTRACT

Broad learning system (BLS) has attracted widespread attention owing to its concise structure and efficient incremental learning based on ridge regression approximation of pseudo-inverse. However, BLS struggles with expensive computational costs and high memory usage when processing complex problems and training large networks due to the inverse operation of a large matrix. There has been some attempts to address this issue. In this work, we attempt to ameliorate the computational efficiency motivated by the state-of-the-art randomized iterative least squares solver. First, we improve the iteration manner of randomized extended Kaczmarz (REK) and then propose the iterated projection learning for training BLS without pseudo-inverse. The iterated projection learning is suitable for both offline and online learning scenarios and can adjust the model structure by incremental learning. Thanks to its linear convergence of least norm in expectation, the obtained model is expected to have better generalization ability without additional regularization procedure. Our model is found to maintain stable performance during incremental learning. Moreover, instead of grid search, we present an evolutionary bilevel programming (EBP) method with dispersion operator to further optimize the hyperparameters of the network structure. Numerical experiments indicate that the proposed methods can improve the efficiency, robustness, and generalization ability compared with pseudo-inverse and grid search based methods.

1. Introduction

Broad learning system (BLS) is a burgeoning neural network structure proposed by Chen et al [1]. Serving as an effective and efficient alternative to deep learning, BLS enables rapid completion of training with ridge regression approximation of pseudo-inverse and exhibits remarkable performance in a wide variety of classification and regression problems. Theoretical proofs have demonstrated the universal approximation capability of BLS [2]. In contrast to currently popular deep learning models, BLS offers distinct advantages in the following three aspects: 1) with few parameters and hyperparameters, BLS is a simple double-hidden layer system connected across layers; 2) only the weights from extended input to output layer are involved in training, and such weights can be quickly calculated by pseudo-inverse rather than the time-consuming back propagation of the gradient; 3) when the input or network

* Corresponding author.
E-mail address: wangjiannl@upc.edu.cn (J. Wang).

https://doi.org/10.1016/j.ins.2023.119648
Received 21 October 2022; Received in revised form 28 August 2023; Accepted 29 August 2023
Available online 3 September 2023

structure needs to be extended, BLS can learn incrementally without training from scratch. In this context, it is worth noting the work by Liu et al. [3], which made an extensive study to demonstrate the efficiency of BLS for traffic prediction. The authors used a large number of state-of-the art methods for comparison and demonstrated superior performance of the BLS systems over others.

The design of BLS is inspired by random vector functional-link neural network (RVFLNN) [4–6] which introduces the sparse feature extraction from input layer to the feature mapping nodes. BLS employs a flat neural network structure that enhances the nonlinear approximation ability by extending the breadth (number of nodes). The original input is first mapped randomly to the feature mapping groups and the sparse weights are obtained with the help of a sparse autoencoder. Then, the output of the feature mapping groups is nonlinearly activated at the enhancement nodes after another random mapping. Next, the extended input is obtained by concatenating the output of feature mapping nodes and the enhancement nodes. The final output of the system is the weighted extended input, and the output weights are the only parameter that needs to be trained.

In recent years, the efficient and flexible nature of BLS has attracted extensive attention, and various variants have enriched the application prospect of BLS [7,8]. Cheng et al. [2] introduced the cascade connection between enhancement nodes and feature mapping nodes. Zhang et al. [9] presented a variant with a dense cascade of enhancement nodes to avoid redundant information. To make BLS useful for big data scenario, authors in [10] have proposed a recursive implementation of BLS that can avoid storing and inverting of large matrices for big data and thereby reduces memory requirement as well as computation. Like some other variants of BLS, this hybrid algorithm avoids retraining of the already trained portion of the network when the width of the network is expanded. The CNN-based Broad learning Systems proposed by Yang [11] used convolution and max pooling for feature extraction. Feng et al. [12] presented a broad and deep architecture which cascaded the convolutional feature mappings and enhancement mappings for representation learning. Moreover, BLS in recurrent form can capture dynamic features [13], which exhibits obvious advantages in dealing with time series problems. In the recent past, the improvement in feature extraction by BLS has been attempted by augmenting it with fuzzy neural systems [13]. For example, Feng et al.[14] replaced the feature mapping groups with a group of Takagi-Sugeno fuzzy subsystems and proposed the fuzzy broad learning system (FBLS). Compact fuzzy broad learning system (CFBLS) [15] balancing precision and complexity was proposed to achieve better interpretability through a compact fuzzy inference system. Hu et al. [16] introduced attention mechanism to further improve the ability of the BLS to deal with time series data. Fan et al. [17], on the other hand, adopted a weighting strategy for the least squares error term to solve the imbalanced classification problem effectively. Liu et al. [18] utilized Cauchy loss instead of least squares loss to improve the performance of the BLS in a noisy environment. Jin et al. [19] added the class-wise sparsity constraint to the objective function of BLS to obtain more vital discriminative ability.

For BLS and its variants, once the network structure is determined, the mapping of the system input to the extended input is well-determined. Since the extended input and the system output are linearly related, the learning of output weights can be regarded as solving a system of linear equations. Thus, the training of BLS can be achieved by the least squares estimation method. Classical methods, including direct method and iterative method, can be used to solve such linear systems. Present works adopt the ridge regression approximation of pseudo-inverse for the learning of BLS. This confers the advantage of BLS over deep learning models, i.e., no back-propagation algorithm is needed in the learning process of BLS, which is computationally expensive for networks with a large number of parameters to train. Furthermore, it also eliminates the tedious and time-consuming derivation of gradient propagation in the process of neural network construction.

However, the training using the pseudo-inverse may not be ideal in all cases, especially when dealing with big data or high-dimensional problems requiring a large number of nodes. The inverse operation of a large matrix can consume considerable computing resources and may face the problem of insufficient computer memory. Also, there may be redundant information in the samples, making it inefficient to process all the samples at once. Therefore, iterative methods may be another approach worth exploring to solve the least squares estimation of the weights. The Kaczmarz method [20] is an impactful class of iterative methods that work on iterative projection of rows for solving linear systems. Later, a similar approach, algebraic reconstruction technique (ART), is rediscovered by Gordon et al. [21] in the field of image reconstruction and has been applied to computerized tomography [22]. Empirical evidence has demonstrated that it is more effective to have the probability of selecting rows follow a non-uniform distribution [23,24]. Strohmer et al. [25] have proposed the random variant of Kaczmarz method and proved its linear convergence rate, which generated new research interest in the past decade. Whereas the above methods are only applicable to consistent linear systems (systems that have at least one solution). For inconsistent systems, Zouzias et al.[26] have proposed randomized extended Kaczmarz (REK) method by introducing the randomized orthogonal projection (ROP), which approximates the least norm least squares solution by establishing a consistent linear system of the target. REK has the ability of solving linear systems similar to pseudo-inverse, which has motivated us to explore use of REK to train BLS via an iterative method.

Generalization performance is another potential advantage of training the output weights with an iterative method. At present, the generalization performance of BLS based on ridge regression can be critical when an incremental version of the algorithm is used. The stepwise updating algorithm of ridge regression for a partitioned matrix [27,28] enables BLS to perform incremental learning efficiently, but the performance of incremental learning may not be stable without the correct regularization parameter. Min et al. [29] found that if the regularization parameter in ridge regression is not set properly, the accuracy will become worse with incremental learning. As different models require different sets of appropriate regularization parameters, a fixed set of regularization parameters may not be the best choice for the dynamic requirements of incremental learning. However, the proper setting of regularization parameters cannot be obtained through prior knowledge. To optimize the regularization parameters, Jin et al. [30,31] used a grid search method, while Min et al. [29] proposed a regularization method using weighted generalized cross validation, which significantly improved the efficiency of regularization parameter optimization. Nevertheless, existing methods inevitably require additional training cost to accomplish the regularization. While for an iterative method, Ma et al. [32] have analyzed the property that REK can converge to the least norm solution starting from the zero vector. Since no component orthogonal to the row span is added in each

iteration, the convergence result is exactly the least norm solution. As the iterative method is an approximation of the required solution, weights with less magnitude are expected, which will be discussed later. This effect is similar to regularization but without introducing additional optimization of any regularization term or training cost. It promotes the construction of a model which is expected to have better generalization performance.

Based on the above analysis and conjecture, we formulate a novel learning approach for the BLS. Our contributions are summarized as follows: 1) we first analyze the feasibility and potential generalization advantage of iterative least squares solver for training BLS, and propose an efficient variant that is suitable for incremental learning; 2) based on the modified method, iterated projection learning is presented for training BLS for offline, online, and incremental learning scenarios without pseudo-inverse; 3) to avoid the time-consuming process of grid search, evolutionary bilevel programming method based on dispersion operator is proposed to optimize the structural hyperparameters of BLS.

The rest of the paper is organized as follows. In section 2, we introduce the typical BLS and the applicable iterative methods. In Section 3, three modes of iterated projection learning are formulated for training BLS. An evolutionary bilevel programming approach is presented in Section 4 for structure optimization of BLS. In Section 5, numerical experiments demonstrate the performance of the proposed methods. The last section provides the conclusion and discusses the prospect of the proposed methods.

2. Preliminaries

2.1. Broad learning system

In this section, we first introduce the network structure of broad learning system. For the symbols covered in this paper, the superscript indicates the sequence number of the variable, and the subscript indicates the index of the element. As shown in Fig. 1, a typical BLS contains a feature mapping node layer and an enhancement node layer. For a given training data set $\{\boldsymbol{X}, \boldsymbol{Y}\}$, $\boldsymbol{X} = [\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_M}]^T \in \mathbb{R}^{M\times D}$ denotes the input matrix, and $\boldsymbol{Y} = [\boldsymbol{y_1}, \boldsymbol{y_2}, \ldots, \boldsymbol{y_M}]^T \in \mathbb{R}^{M\times C}$ denotes the corresponding target. The input matrix is first transformed and passed on to the feature mapping groups. Assuming that each group of feature maps has N_f nodes, the nth group of feature mapping matrix $\boldsymbol{Z}_n \in \mathbb{R}^{M\times N_f}$ is expressed as

$$\boldsymbol{Z}_n = \phi_n(\boldsymbol{X}\boldsymbol{w}_{en} + \boldsymbol{\beta}_{en}), n = 1, 2, \ldots, N_m \tag{1}$$

where $\boldsymbol{w}_{en} \in \mathbb{R}^{D\times N_f}$ and $\boldsymbol{\beta}_{en} \in \mathbb{R}^{D\times 1}$ are the weights and biases of the nth feature mapping ϕ_n respectively. The weights of each feature group are first randomly initialized and further adjusted by a sparse autoencoder to obtain the sparse feature $\boldsymbol{Z}_n$ [1]. The output of N_m groups of feature mapping is expressed as $\boldsymbol{Z} = [\boldsymbol{Z}_1, \boldsymbol{Z}_2, \ldots, \boldsymbol{Z}_{N_m}]$, and serves as the input of the enhancement node layer for nonlinear transformation. The enhancement output matrix $\boldsymbol{H} \in \mathbb{R}^{M\times N_e}$ is calculated by

$$\boldsymbol{H} = f(\boldsymbol{Z}\boldsymbol{w}_h + \boldsymbol{\beta}_h) \tag{2}$$

where f is the activation function introducing the nonlinearity to the system. The enhancement weights and biases are denoted as $\boldsymbol{w}_h \in \mathbb{R}^{N_f N_m \times N_e}$ and $\boldsymbol{\beta}_h \in \mathbb{R}^{N_f N_m \times 1}$, which are randomly initialized. Similar to feature mapping nodes, enhancement nodes can also be grouped by different nonlinear transformations. The feature mapping matrix is then concatenated with the enhancement output matrix as the extended input matrix

$$\boldsymbol{A} = [\boldsymbol{Z} \quad \boldsymbol{H}] \tag{3}$$

The output of the system $\widehat{\boldsymbol{Y}}$ is obtained after weighting,

$$\widehat{\boldsymbol{Y}} = \boldsymbol{A}\boldsymbol{W} \tag{4}$$

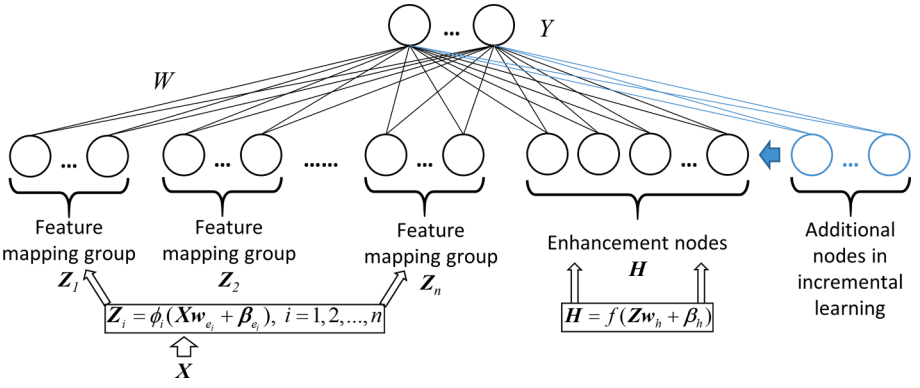

Fig. 1. Structure of typical BLS.

where $\boldsymbol{W} \in \mathbb{R}^{(N_f N_m + N_e) \times C}$ is the output weight matrix between the extended input and system output. Different from the random weights above, the output weights are obtained by training. The most common form of training output weights is the least squares estimation with regularization, which is considered to enhance the generalization performance of the model. The estimation problem can be expressed as the following optimization problem:

$$\underset{W}{\operatorname{argmin}} \|\boldsymbol{A}\boldsymbol{W} - \boldsymbol{Y}\|_v^{\sigma_1} + \lambda \|\boldsymbol{W}\|_u^{\sigma_2} \tag{5}$$

In general setting, $\sigma_1 = \sigma_2 = u = v = 2$. Eq. (5) is equivalent to the ridge regression approximation of pseudo-inverse, where λ is the regularization parameter used to balance the training error and generalization ability. Although the solution based on pseudo-inverse can obtain the result with minimum error, for the linear system with a large number of unknowns, this method suffers from the expensive calculation and memory consumption for the inversion operation. Furthermore, the performance of the model is sensitive to the regularization parameter, but the appropriate setting is not easy to find. Different data sets are expected to have different requirements for the regularization parameter, while networks of different sizes for the same data set also have different requirements. Obviously, fixed regularization parameter cannot necessarily guarantee desirable solutions for diverse problems as well as for incremental learning. Hence, the additional computational cost of optimizing the regularization parameter cannot be ignored for ridge regression.

2.2. Iterated projection

In this paper, another method without the regularization term is adopted to avoid the above problems. If $\lambda = 0$ in the optimization problem (5), the learning of $\boldsymbol{W}$ is transformed into solving a multi-output linear system $\boldsymbol{A}\boldsymbol{W} = \boldsymbol{Y}$ with the extended input $\boldsymbol{A}$ as the coefficient matrix. Iterative method is another approach for solving this problem. It can approximate the target solution recursively with the designed iterative process. The iterative method takes up less memory and can flexibly adjust the calculation time according to the accuracy requirement.

Kaczmarz method [20] is an impactful class of iterative method that works on rows of a linear system. Its convergence rate for a linear system with a unique solution has been proved, and the study of the convergence rate boundary can be found in [25]. Its basic principle is that in each iteration, it orthogonally projects the current estimation point onto a selected row of the coefficient matrix $\boldsymbol{A}$, as shown in Fig. 2(a). The classical Kaczmarz method circularly selects the rows of $\boldsymbol{A}$ in turn. Namely, for the projection at kth step, the index of the row is selected by $i = k \bmod M$, where mod denotes the remainder operation. Considering a single-output problem $\boldsymbol{A}\boldsymbol{w} = \boldsymbol{y}$ with $\boldsymbol{y} \in \mathbb{R}^{M \times 1}$ as an example, Kaczmarz method iteratively adjusts the estimation point $\boldsymbol{w}^k$ in the following manner:

$$w^k = w^{k-1} + \alpha A_{i\cdot}^T \tag{6}$$

where $\boldsymbol{A}_{i\bullet}$ denotes the ith row of the coefficient matrix $\boldsymbol{A}$. Thus, $\boldsymbol{A}_{i\bullet}$ represents the ith row of extended input matrix $\boldsymbol{A}$ in the context of BLS, i.e., the input point $\boldsymbol{x}_i$ after the transformation of feature mapping nodes and enhancement nodes. With the step size α, project the estimation point to the selected ith equation (row) by

$$\boldsymbol{A}_{i\cdot}\boldsymbol{w}^k = \boldsymbol{y}_i \tag{7}$$

where $\boldsymbol{y}_i$ is ith element of vector $\boldsymbol{y}$. The expression of α can be obtained by combining (6) and (7),

$$\alpha = \frac{\boldsymbol{y}_i - \boldsymbol{A}_{i\cdot}\boldsymbol{w}^{k-1}}{\|\boldsymbol{A}_{i\cdot}\|_2^2} \tag{8}$$

Thus, the iterative formula of classical Kaczmarz method is

$$\boldsymbol{w}^k = \boldsymbol{w}^{k-1} + \frac{\boldsymbol{y}_i - \boldsymbol{A}_{i\cdot}\boldsymbol{w}^{k-1}}{\|\boldsymbol{A}_{i\cdot}\|_2^2} \boldsymbol{A}_{i\cdot}^T \tag{9}$$

Strohmer et al. [25] proposed a more efficient variant, the randomized Kaczmarz (RK) method. Unlike the sequential selection of rows, RK randomly selects the rows of $\boldsymbol{A}$ according to a non-uniform probabilities computed by

$$\boldsymbol{P}_i = \frac{\|\boldsymbol{A}_{i\cdot}\|_2^2}{\|\boldsymbol{A}\|_F^2} \tag{10}$$

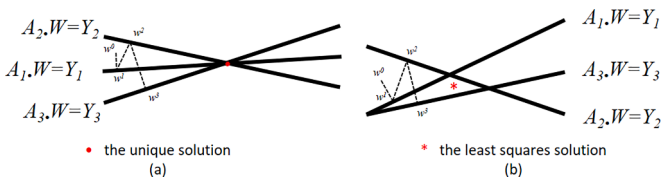

Fig. 2. Projection of Kaczmarz method. (a) Consistent system (b) Inconsistent system.

where $\|\cdot\|_F$ denotes the Frobenius norm. RK has the linear convergence for the consistent linear system in expectation:

Lemma 1. [25] *Assume that $\boldsymbol{A}\boldsymbol{w}=\boldsymbol{y}$ has a solution, and the maximum iteration number $k_m>1$. Denote $\boldsymbol{w}_{LS}:=\boldsymbol{A}^{\dagger}\boldsymbol{y}$ and $\kappa_F^2(\boldsymbol{A}):=\|\boldsymbol{A}\|_F^2\|\boldsymbol{A}^{\dagger}\|_2^2$. It can be shown that RK converges to $\boldsymbol{w}_{LS}$ in expectation:*

$$\mathbb{E}\left\|\boldsymbol{w}^{(k)}-\boldsymbol{w}_{LS}\right\|_2^2 \leq \left(1-\frac{1}{\kappa_F^2(\boldsymbol{A})}\right)^k \left\|\boldsymbol{w}^{(0)}-\boldsymbol{w}_{LS}\right\|_2^2 \quad \forall k>0$$

However, the above methods are applicable only to the consistent systems. For an inconsistent system, RK cannot converge to the least squares solution outside the solution space via projecting to the selected equations. It will approach a solution with an additional error from the least squares solution [33]. This phenomenon, as shown in Fig. 2(b), is known as the convergence horizon. In order to reduce this error, randomized extended Kaczmarz (REK) method [26] introduces the randomized orthogonal projection (ROP) to approximate of the residual of normal equations, and collaborate with RK to constitute an intertwined iterative manner [26],

$$\boldsymbol{r}^k=\boldsymbol{r}^{k-1}-\frac{\boldsymbol{A}_{\cdot j}\boldsymbol{r}^{k-1}}{\left\|\boldsymbol{A}_{\cdot j}\right\|_2^2}\boldsymbol{A}_{\cdot j} \tag{11}$$

$$\boldsymbol{w}^k=\boldsymbol{w}^{k-1}+\frac{\boldsymbol{y}_i-\boldsymbol{r}_i^k-\boldsymbol{A}_{i\cdot}\boldsymbol{w}^{k-1}}{\left\|\boldsymbol{A}_{i\cdot}\right\|_2^2}\boldsymbol{A}_{i\cdot}^T \tag{12}$$

where $\boldsymbol{r}^k$ is the approximation of the residual in the kth step. Similarly, the ROP selects the jth column of $\boldsymbol{A}$ with probability

$$\boldsymbol{P}_j=\frac{\left\|\boldsymbol{A}_{\cdot j}\right\|_2^2}{\|\boldsymbol{A}\|_F^2} \tag{13}$$

The main mechanism of REK is that the approximate normal equation $\tilde{\boldsymbol{y}}_i^k=\boldsymbol{y}_i-\boldsymbol{r}_i^k$ is obtained by ROP and the least norm solution of the system $\boldsymbol{A}\boldsymbol{w}=\tilde{\boldsymbol{y}}^k$ is obtained by RK in kth iteration step. In other words, REK can transform the inconsistent system to a consistent one whose solution is the target least squares solution. Numerical experiments [26] show that REK is significantly more efficient than the direct method based on pseudo-inverse in solving large-scale linear systems. Thus, the iterative solution method has the potential to improve the training efficiency of BLS.

Subsequent research modified the efficiency of intertwined iterative behavior. Dumitrescu [34] proposed the idea of segregating ROP and RK, i.e., first use (11) to obtain the approximate normal equation $\boldsymbol{A}\boldsymbol{w}=\tilde{\boldsymbol{y}}$ via k_m iterations, and then use (9) to solve the least norm solution via the same number of iterations. Compared with the classical iterations of (11) and (12), this approach directly applies RK to a nearly consistent system instead of an inchoate one, which is intuitively more efficient. We adopt the direct way in this work.

Remark 1. *Another significance of Kaczmarz methods is its convergence property. A model with parameters having lower magnitude is generally considered to have better generalization ability, so this property is particularly important to improve the performance of a trained model.*

Lemma 2. [32] *For an underdetermined system $\boldsymbol{A}\boldsymbol{w}=\boldsymbol{y}$, assume that $\boldsymbol{A}$ has full row rank. If the initial iteration point $\boldsymbol{w}^0$ is set in the row space $C(\boldsymbol{A}^T)$, the solution $\boldsymbol{w}_{LS}$ that RK converges to is the least norm solution $\boldsymbol{w}_{LN}$.*

Proof We will give a simple proof of Lemma 2 by contradiction. An underdetermined system is consistent with infinite solutions, but only has a unique least norm solution $\boldsymbol{w}_{LN}=\boldsymbol{A}^T(\boldsymbol{A}\boldsymbol{A}^T)^{-1}\boldsymbol{y}$. It can also be denoted as the product of the matrix $\boldsymbol{A}^T$ and the vector $\boldsymbol{\theta}=(\boldsymbol{A}\boldsymbol{A}^T)^{-1}\boldsymbol{y}$, implying that $\boldsymbol{w}_{LN}$ is in the row space $C(\boldsymbol{A}^T)$. Then, the general solution is denoted as $\boldsymbol{w}=\boldsymbol{w}_{LN}+\boldsymbol{r}$, where $\boldsymbol{r}=(\boldsymbol{I}-\boldsymbol{A}^{\dagger}\boldsymbol{A})\boldsymbol{z}$, $\boldsymbol{z}\in\mathbb{R}^n$ is in the null space $N(\boldsymbol{A})$ (i.e., the solution space of $\boldsymbol{A}\boldsymbol{r}=0$) and orthogonal to $C(\boldsymbol{A}^T)$. Since $\boldsymbol{r}\perp\boldsymbol{w}_{LN}$, the norm of the general solution meets $||\boldsymbol{w}||^2=||\boldsymbol{w}_{LN}||^2+||\boldsymbol{r}||^2$. Thus, $\boldsymbol{w}_{LN}$ differs from other consistent solutions in that the proportion of $\boldsymbol{r}$ in $\boldsymbol{w}_{LN}$ is zero; in other words, its projection on $N(\boldsymbol{A})$ is zero.

Assuming that the solution $\boldsymbol{w}_{LS}$ which RK converges according to Lemma 1 is not $\boldsymbol{w}_{LN}$, then its projection on $N(\boldsymbol{A})$ is not zero. However, if the initial iteration point $\boldsymbol{w}^0$ is set in $C(\boldsymbol{A}^T)$, RK only adds multiples of rows to the estimate point in each iteration as (8). Since $C(\boldsymbol{A}^T)$ is orthogonal to $N(\boldsymbol{A})$, the projection of the obtained solution on $N(\boldsymbol{A})$ cannot increase, which contradicts the earlier assumption. Therefore, if $\boldsymbol{w}^0$ is set in $C(\boldsymbol{A}^T)$, RK can converge to the solution whose projection is zero on $N(\boldsymbol{A})$, and that is exactly the least norm solution $\boldsymbol{w}_{LS}$.

In contrast, if the projection of $\boldsymbol{w}^0$ on $N(\boldsymbol{A})$ is not zero, it cannot be eliminated in RK iterations, resulting in the failure to converge to $\boldsymbol{w}_{LS}$. Hence, the set of $\boldsymbol{w}^0$ is crucial to the convergence property.

In this work, we make the following three improvements to the iterative method to meet the needs of BLS training.

Remark 2. *Reducing the training error to zero is usually not the ultimate goal of training a model. For example, ridge regression with regularization term* (5) *providing results that balance accuracy and model complexity is preferred. In this work, the initial iteration point is strictly set as a zero vector with a view to improving the generalization ability although it is only required to be set in the row space $C(\boldsymbol{A}^T)$ in the original RK method. We give the following intuitive analysis of this improvement.*

According to Lemma 1 and Lemma 2, the Euclidean distance between the approximate solution $\boldsymbol{w}^k$ and the target $\boldsymbol{w}_{LN}$ decreases linearly with iteration in expectation. On the one hand, it shows that the error is inversely related to the number of iterations.

Moreover, this shows that if the target $\boldsymbol{w}_{LN}$ is approached iteratively from zero, the norm of the approximate solution $||\boldsymbol{w}^k||$ will gradually increase from zero to $||\boldsymbol{w}_{LN}||$ in expectation. On the contrary, if the initial iteration point is randomly selected in the row space, the trend of the norm of the approximate solution $||\boldsymbol{w}^k||$ is uncertain because the magnitudes of $\boldsymbol{w}^0$ and $\boldsymbol{w}_{LN}$ are unknown. Therefore, if the initial iteration point is set as the zero vector, the approximate solution $\boldsymbol{w}^k$ can be regarded as a balance between accuracy and complexity (magnitude). From the perspective of reducing the model complexity, this improved method can achieve an effect similar to that of ridge regression, but it completes in the process of convergence to the target $\boldsymbol{w}_{LN}$, which is different from ridge regression that approximates the point outside the solution space by pseudo-inverse. The advantages lie in lower computational cost and the need of no regularization parameter.

Remark 3. *Most of the current iterative methods only consider that the system contains one output. Whereas for the classification and multi-objective regression problems, we usually have to deal with linear systems with multiple outputs. Since different outputs can form multiple independent linear systems, this problem can of course be directly solved by C independent systems, but this will cost redundant repeated algebraic operations. Hence, for a multi-output system $\boldsymbol{AW} = \boldsymbol{Y}$, we respectively formulate the update of the residual matrix $\boldsymbol{R}$ in ROP and estimation matrix $\boldsymbol{W}$ in RK at the kth step as.*

$$\boldsymbol{R}^k = \boldsymbol{R}^{k-1} - \boldsymbol{A}_{\cdot j}\frac{\boldsymbol{A}_{\cdot j}\boldsymbol{R}^{k-1}}{\left\|\boldsymbol{A}_{\cdot j}\right\|_2^2} \tag{14}$$

$$\boldsymbol{W}^k = \boldsymbol{W}^{k-1} + \boldsymbol{A}_{i\cdot}^T\frac{\boldsymbol{Y}_i^k - \boldsymbol{A}_{i\cdot}\boldsymbol{W}^{k-1}}{\|\boldsymbol{A}_{i\cdot}\|_2^2} \tag{15}$$

Moreover, prior variables will be introduced to reduce redundant calculation in iterations and incremental learning, which will be discussed in Section 3. We call the modified method Iterated Projection, with steps shown in **Algorithm 1**.

3. Iterated projection learning

The above preliminaries only represent half the story. Without calculating the pseudo-inverse, iterated projection learning is proposed in this work to avoid the expensive matrix inversion operation in complex problems requiring large networks. This section will introduce two modes of learning for the proposed BLS as well as for the incremental learning of structure.

Algorithm 1 Iterated Projection ($\boldsymbol{A}$, $\boldsymbol{Y}$, $\boldsymbol{W^0}$, k_m).

Input: coefficient matrix $\boldsymbol{A}$, right-handed matrix $\boldsymbol{Y}$, initial point $\boldsymbol{W^0}$, and maximum iteration number k_m;
Output: estimation matrix $\boldsymbol{W}$;
1:Initialize $\boldsymbol{W^0} = \boldsymbol{0}$, and $\boldsymbol{R^0} = \boldsymbol{Y}$.
2:**for** $k = 1$ to k_m **do**
3: Select i according to (10).
4: Update the residual matrix $\boldsymbol{R^k}$ by (14).
5:**end for**
6:Obtain the normal equation by $\widetilde{\boldsymbol{Y}} = \boldsymbol{Y} - \boldsymbol{R}$.
7:**for** $k = 1$ to k_m **do**
8: Select j according to (13).
9: Update the estimation matrix $\boldsymbol{W^k}$ by (15).
10:**end for**
11:**Output** $\boldsymbol{W} = \boldsymbol{W^k}$

3.1. Offline learning mode

The offline learning mode, also known as full-batch learning, is generally suitable for one-shot training requirement of a given data set. The extended input matrix $\boldsymbol{A}$ is calculated by (1) – (3). Offline learning can be expressed as solving the linear system:

$$\boldsymbol{AW} = \boldsymbol{Y} \tag{16}$$

The extended input matrix $\boldsymbol{A}$ and the target $\boldsymbol{Y}$ serve as the coefficient matrix and right-handed matrix (or vector for single-output problems) of the linear system, and iterated projection is used to solve the unknowns, i.e., the output weights $\boldsymbol{W}$. However, the norm operation is repeated a large number of times in RK and ROP. All the norms are calculated once to compute the probabilities for selecting the row and column before starting the iteration. Besides, the norms of the rows and columns, the Frobenius norm of the matrix also have repeated square operations. These require considerable computational resources for large matrices or for more iterations. In order to meet the efficiency requirements of neural network training, we improve the iterative algorithm first. Create a square matrix $\boldsymbol{As}$ to store the square of the elements in $\boldsymbol{A}$,

$$\boldsymbol{As} = \begin{bmatrix} \boldsymbol{A}_{11}^2 & \boldsymbol{A}_{12}^2 & \cdots & \boldsymbol{A}_{1N}^2 \\ \boldsymbol{A}_{21}^2 & \boldsymbol{A}_{22}^2 & \cdots & \boldsymbol{A}_{2N}^2 \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{A}_{M1}^2 & \boldsymbol{A}_{M1}^2 & \cdots & \boldsymbol{A}_{MN}^2 \end{bmatrix} \tag{17}$$

Then use the vectors $\boldsymbol{Ac}$ and $\boldsymbol{Ar}$ to store the square of norm of columns and rows of $\boldsymbol{A}$ respectively,

$$\boldsymbol{Ac} = [\sum \boldsymbol{As}_{1\cdot} \quad \sum \boldsymbol{As}_{2\cdot} \quad \cdots \quad \sum \boldsymbol{As}_{N\cdot}] \tag{18}$$

$$\boldsymbol{Ar} = [\sum \boldsymbol{As}_{1\cdot} \quad \sum \boldsymbol{As}_{2\cdot} \quad \cdots \quad \sum \boldsymbol{As}_{M\cdot}]^T \tag{19}$$

where $\boldsymbol{As}_{ij}$ denotes the element of matrix $\boldsymbol{As}$ at row i, column j. $\Sigma\boldsymbol{As}_{i\cdot}$ and $\Sigma\boldsymbol{As}_{\cdot j}$ denote the sum of the elements in the ith row and jth column, respectively. In this way, the calculation of probability and norm operation in iterated projection only needs to read the corresponding elements from $\boldsymbol{Ar}$ and $\boldsymbol{Ac}$:

$$\|\boldsymbol{A}\|_F^2 = \sum \boldsymbol{As} = \sum \boldsymbol{Ac} = \sum \boldsymbol{Ar} \tag{20}$$

$$\|\boldsymbol{A}_{i\cdot}\|_2^2 = \boldsymbol{Ar}_i \tag{21}$$

$$\|\boldsymbol{A}_{\cdot j}\|_2^2 = \boldsymbol{Ac}_j \tag{22}$$

where $\boldsymbol{Ar}_i$ and $\boldsymbol{Ac}_j$ represent the ith and jth elements of $\boldsymbol{Ar}$ and $\boldsymbol{Ac}$, respectively. $\boldsymbol{Ac}$ and $\boldsymbol{Ar}$ are collectively referred to the prior variables and used in the iterated projection. To avoid extra memory requirement, $\boldsymbol{As}$ is deleted after prior variables are established. Iterative methods can be easily restarted if the prior variables are retained. When the accuracy is not sufficient, the iteration can be restarted with the current result as the start point. This format not only improves computation speed, but more importantly will later serve as the basis of incremental learning.

With the assistance of (17) – (22), the linear system (16) is solved by iterated projection. Considering the generalization ability and training error, the maximum iteration number k_m can be set as a fixed value, such as one or two times of the data set size. The iterated projection learning algorithm for the offline version is summarized as **Algorithm 2**.

3.2. Online learning mode

Online learning mode is suitable for data stream scenarios where samples are added one by one. The training of BLS in online mode can be expressed by solving the linear system,

$$\boldsymbol{A}^t \boldsymbol{W}^t = \boldsymbol{Y}^t \tag{23}$$

where $\boldsymbol{A}^t$ and $\boldsymbol{W}^t$ are the extended input matrix and output weights at the tth step of online learning respectively. Traditional iterative methods only consider the static linear system with a fixed number of equations. However, online learning involves a dynamic linear system with increasing number of rows of $\boldsymbol{A}^t$. Thus, we made further improvements based on the prior variables to meet the specific requirement of online learning.

For the newly added sample $\boldsymbol{x}^t$ at the tth step, the nth feature mapping $\boldsymbol{Z}_n^t \in \mathbb{R}^{1\times N_f}$ is calculated by

$$\boldsymbol{Z}_n^t = \phi_n(\boldsymbol{x}^t \boldsymbol{w}_{en} + \boldsymbol{\beta}_{en}), n = 1, 2, ..., N_m \tag{24}$$

Algorithm 2 Iterated Projection Learning of BLS (Offline).

Input: training data $\boldsymbol{X} \in \mathbb{R}^{M\times D}$ and targets $\boldsymbol{Y} \in \mathbb{R}^{M\times C}$;
Output: random parameters $\boldsymbol{w}_e$, $\boldsymbol{\beta}_e$, $\boldsymbol{w}_h$, $\boldsymbol{\beta}_h$, output weight matrix $\boldsymbol{W}$;
1:Generate $\boldsymbol{w}_e$, $\boldsymbol{\beta}_e$, $\boldsymbol{w}_h$, $\boldsymbol{\beta}_h$ randomly;
2:**for** $i = 1$ to N_m **do**
3: Obtain $\boldsymbol{w}_{ei}$, $\boldsymbol{\beta}_{ei}$ by sparse autoencoder;
4: Calculate Z_i according to (1);
5:**end for**
6:Obtain $\boldsymbol{Z} = [\boldsymbol{Z}_1, \boldsymbol{Z}_2, ..., \boldsymbol{Z}_{Nm}]$;
7:Calculate $\boldsymbol{H}$ according to (2);
8:Obtain $\boldsymbol{A}$ by Eq. (3);
9:Obtain $\boldsymbol{Ac}$ and $\boldsymbol{Ar}$ by Eq. (17) – (19);
9:$\boldsymbol{W}$ = Iterated projection ($\boldsymbol{A}$, $\boldsymbol{Y}$, $\boldsymbol{0}$, k_m);

where $\boldsymbol{w}_{en}$ and $\boldsymbol{\beta}_{en}$ represent the weights and biases of the nth group feature mapping. The output of N_m group feature mapping of the tth sample is,

$$\boldsymbol{Z}^t = [\boldsymbol{Z}_1^t, \boldsymbol{Z}_2^t, ..., \boldsymbol{Z}_{N_m}^t] \tag{25}$$

The enhancement output $\boldsymbol{H} \in \mathbb{R}^{1\times N_e}$ of the tth sample is

$$\boldsymbol{H}^t = f(\boldsymbol{Z}^t \boldsymbol{w}_h + \boldsymbol{\beta}_h) \tag{26}$$

where $\boldsymbol{w}_h$ and $\boldsymbol{\beta}_h$ represent enhancement weights and biases. The above weights and biases are randomly initialized, and the subsequent online learning continues the results of the previous steps. Then, the extended input vector at tth step is obtained by

$$\boldsymbol{a}^t = [\,\boldsymbol{Z}^t \quad \boldsymbol{H}^t\,] \tag{27}$$

Similarly, the corresponding square vector is calculated by,

$$\boldsymbol{as}^t = \left[\,(\boldsymbol{a}_1^t)^2 \quad (\boldsymbol{a}_2^t)^2 \quad \dots \quad (\boldsymbol{a}_N^t)^2\,\right] \tag{28}$$

where $\boldsymbol{a}_j^t$ represents the jth term of the extended input vector $\boldsymbol{a}^t$. In the feed forward process of online learning, the new sample reflects as the new row of the extended input matrix $\boldsymbol{A}^t$ and square matrix $\boldsymbol{As}^t$, so $\boldsymbol{A}^t$ and $\boldsymbol{As}^t$ are updated as,

$$\boldsymbol{A}^t = \begin{bmatrix} \boldsymbol{A}^{t-1} \\ \boldsymbol{a}^t \end{bmatrix} \tag{29}$$

$$\boldsymbol{As}^t = \begin{bmatrix} \boldsymbol{As}^{t-1} \\ \boldsymbol{as}^t \end{bmatrix} \tag{30}$$

where $\boldsymbol{A}^{t-1}$ and $\boldsymbol{As}^{t-1}$ are the extended input matrix and square matrix of the previous step. In particular, if $t = 1$, then $\boldsymbol{A}^1 = \boldsymbol{a}^1$, and $\boldsymbol{As}^1 = \boldsymbol{as}^1$. Also, we update the prior variables $\boldsymbol{Ac}^t$ and $\boldsymbol{Ar}^t$ as

$$\boldsymbol{Ac}^t = \left[\,\boldsymbol{Ac}_1^{t-1} + \boldsymbol{as}_1^t \quad \boldsymbol{Ac}_2^{t-1} + \boldsymbol{as}_2^t \quad \dots \quad \boldsymbol{Ac}_N^{t-1} + \boldsymbol{as}_N^t\,\right] \tag{31}$$

$$\boldsymbol{Ar}^t = \begin{bmatrix} \boldsymbol{Ar}^{t-1} \\ \sum \boldsymbol{as}^t \end{bmatrix} \tag{32}$$

where $\boldsymbol{Ac}_j^{t-1}$ represents the jth term of the column norm vector in the previous step. Then, with the assistance of (29) – (32) and (20) – (22), iterated projection is used to solve the dynamic linear system (23). The square matrix and the prior variables provide great convenience for online or incremental learning. We only need to concatenate the new rows for the matrix $\boldsymbol{A}$, and update the vectors $\boldsymbol{Ac}$, $\boldsymbol{Ar}$ and probability of selecting rows, so that we can continue to train the system on the basis of the previous step. For the first arriving sample, iterated projection starts with the zero vector; while for the subsequent samples, the result of the previous step is inherited as the initial point.

Algorithm 3 Iterated Projection Learning of BLS (Online).

Input Training data point $\boldsymbol{x}^t \in \mathbb{R}^{1\times M}$ and target $\boldsymbol{y}^t \in \mathbb{R}^{1\times C}$, total step M;
Output: random parameters $\boldsymbol{w}_e$, $\boldsymbol{\beta}_e$, $\boldsymbol{w}_h$, $\boldsymbol{\beta}_h$, output weight $\boldsymbol{W}$;
1:Generate w_e, β_e, w_h, β_h randomly, and initialize $\boldsymbol{W}^0 = \boldsymbol{0}$;
2: **while** $t \le M$ **do**
3: **for** i to N_m **do**
4: Obtain $\boldsymbol{w}_{ei}^t$, $\boldsymbol{\beta}_{ei}^t$ by sparse autoencoder;
5: Calculate $\boldsymbol{Z}_i^t$ according to (24);
6: **end for**
7: Obtain $\boldsymbol{Z}^t$ by (25);
8: Calculate $\boldsymbol{H}^t$ according to (26);
9: Obtain $\boldsymbol{A}^t$ by (29);
Update $\boldsymbol{As}^t$, $\boldsymbol{Ac}^t$, $\boldsymbol{Ar}^t$ according to (30) – (32);
$\boldsymbol{W}^t$ = Iterated projection ($\boldsymbol{A}^t$, $\boldsymbol{Y}$, $\boldsymbol{W}^{t-1}$, k_m^t);
18: Update $\boldsymbol{W}^t$ according to the sliding window filter;
19: **end while**

We note here that the extended input matrix involved in each step increases linearly and we propose the number of iterations can be assigned adaptively according to

$$k_m^t = \lceil 2M \cdot \tanh(t/M) \rceil \tag{33}$$

k_m^t is a number between (0,2M), which increases with step number t. If the total sample size of the data set in online learning is

unknown, M in the formula can be the approximate magnitude of the total sample size. In order to avoid the infinite growth of iteration number, the hyperbolic tangent function is introduced; however, other function can also be used.

Moreover, online learning will fine-tune the output weights of the model for fitting the incoming new sample point. Nevertheless, this adjustment may not improve the generalization error or test error, and may cause the test error fluctuation with iteration, which is a common problem in online learning. Optionally, this fluctuation can be eliminated using the training error of adjacent steps. For example, after a few steps of online learning, we set up a sliding window of size n_w as a filter to dynamically record the training error from (t-n_w-1)th step to (t-1)th step of online learning. In each step, the current training error is compared with the record in the filter. If the error of the current step is the minimum, update the output weights; otherwise, the result of the previous step is retained. The algorithmic version of the iterated projection learning for the online mode is presented as **Algorithm 3**.

3.3. Incremental learning via structure augmentation

Incremental learning is suitable for adding data or nodes on an existing system. The online learning mentioned in the previous part is a particular case of the incremental learning of input data. It can be extended to add multiple samples at a time and for this we just need to add the corresponding rows in the square matrix. We shall not repeat it here. In this part we focus on incremental learning of structure augmentation. When the current system cannot meet the accuracy requirements, BLS can add feature mapping nodes and enhancement nodes to improve the approximation performance. Each step of incremental learning can be realized by solving the following linear system,

$$\boldsymbol{A}^s \boldsymbol{W}^s = \boldsymbol{Y} \tag{34}$$

The incremental learning of nodes is reflected in the increase of columns in the extended input matrix $\boldsymbol{A}^s$, so we need to solve a linear system with an increased number of columns. Similar to the increment of the input sample, we calculate the additional feed forward information first. Assuming that at the sth incremental learning step, incremental learning adds c nodes, of which there are c_e feature mapping nodes and c_h enhancement nodes. The output of additional feature mapping $\boldsymbol{Z}^s \in \mathbb{R}^{M\times c_e}$ is

$$\boldsymbol{Z}^s = \phi_s(\boldsymbol{X}\boldsymbol{w}_e^s + \boldsymbol{\beta}_e^s) \tag{35}$$

and for the new enhanced nodes $\boldsymbol{H}^s \in \mathbb{R}^{M\times c_h}$ is

$$\boldsymbol{H}^s = f(\boldsymbol{Z}\boldsymbol{w}_h^s + \boldsymbol{\beta}_h^s) \tag{36}$$

The above weights and biases are generated randomly. Then the additional extended input matrix is defined as

$$\boldsymbol{a}^s = [\boldsymbol{Z}^s \quad \boldsymbol{H}^s] \tag{37}$$

Similarly, every element in $\boldsymbol{a}^s$ is squared to obtain the newly added square matrix, $\boldsymbol{as}^s$. We update $\boldsymbol{A}^s$, $\boldsymbol{As}^s$, $\boldsymbol{Ac}^s$ and $\boldsymbol{Ar}^s$ according to the following equations:

$$\boldsymbol{A}^s = \left[\boldsymbol{A}^{s-1} \quad \boldsymbol{a}^s\right] \tag{38}$$

$$\boldsymbol{As}^s = \left[\boldsymbol{As}^{s-1} \quad \boldsymbol{as}^s\right] \tag{39}$$

$$\boldsymbol{Ac}^s = \left[\boldsymbol{Ac}^{s-1} \quad \sum \boldsymbol{as}_{\cdot 1}^s \quad \dots \quad \sum \boldsymbol{as}_{\cdot c}^s\right] \tag{40}$$

$$\boldsymbol{Ar}^s = \left[\boldsymbol{Ar}_1^{s-1} + \sum \boldsymbol{as}_{1\cdot}^s \quad \boldsymbol{Ar}_2^{s-1} + \sum \boldsymbol{as}_{2\cdot}^s \quad \dots \quad \boldsymbol{Ar}_M^{s-1} + \sum \boldsymbol{as}_{M\cdot}^s\right]^T \tag{41}$$

where $\Sigma \boldsymbol{as}_{i\bullet}^s$ and $\Sigma \boldsymbol{as}_{\bullet j}^s$ denote the sum of the ith row and jth column of $\boldsymbol{as}^s$, respectively. For each step of incremental learning, iterated projection is used to solve the dynamic system (34).

Since the additional columns of $\boldsymbol{A}^s$ involves c new unknowns, it is necessary to initialize the corresponding estimate of new unknowns while updating. For each incremental learning step, the existing nodes inherit the result of the previous step as the starting point of iteration, and the weights of the new nodes are initialized to 0.

$$\boldsymbol{W}^0 = [\boldsymbol{W}^{s-1} | \boldsymbol{0}^c]^T \tag{42}$$

where $\boldsymbol{0}^c$ represents the zero vector (or matrix) of length c. The schematic description of the incremental iterated projection learning via addition of nodes is summarized as **Algorithm 4**.

Algorithm 4 Iterated Projection Learning of BLS (Incremental).

Input: Training data $X \in \mathbb{R}^{M\times D}$, targets $Y \in \mathbb{R}^{M\times C}$, incremental step S, number of incremental feature nodes and enhancement nodes per step c_f, c_h
Output: output weight matrix W;
1: construct initial model by Algorithm 2
2: **while** $s \le S$ **do**
3: **if** $c_e \neq 0$ **then**
4: Obtain $\boldsymbol{w}_e^s$, $\boldsymbol{\beta}_e^s$ by sparse autoencoder;

(*continued on next page*)

(*continued*)

```
    Calculate Z^s according to (35);
  end if
  if c_h ≠ 0 then
    Generate w_h^s, β_h^s randomly;
    Calculate H^s according to (36);
  end if
  Obtain A^s by (38);
  Update As^s, Ac^s, Ar^s according to (39) – (41);
  W^s = Iterated projection (A^s, Y, W^{s-1}, k_m);
end while
```

4. Evolutionary bilevel programming

In addition to iterated projection learning of weights for BLS, we also propose an evolutionary bilevel programming (EBP), a structural hyperparameter optimization method based on evolutionary operators. The hyperparameter optimization of BLS is different from that of other types of feedforward neural networks. On the basis of an appropriate number of nodes, nodes need to be grouped to obtain the best feature representation. Therefore, the hyperparameter optimization of BLS is a typical bilevel optimization programming problem. Existing studies of BLS usually utilize the grid search method to explore all possible combinations of numbers and groups [1,14,15,29], which obviously ignores (does not exploit) the dependency between hyperparameters leading to unnecessary computation. Grouping serves to optimize feature representation at the existing structure scale, while optimization of grouping is completely unnecessary at an inappropriate structure. In order to reduce computational cost, the proposed method subdivides the hyperparameter optimization into two levels: the lower level optimizes the number of nodes, and the upper level optimizes the grouping strategy. In this context we introduce the evolutionary operation to replace the grid search method.

The upper level problem serves as the ultimate goal, which can be expressed as minimization of an objective function, $loss_3$, with three optimizing variables,

$$\min_{N_f,N_m} loss_3(N_f,N_m,N_h), s.t.(N_f,N_m)_{N_h} \in S(N_{fm},N_h) \tag{43}$$

where $loss_3$ is defined as the training error on a data set under the corresponding hyperparameter settings (for classification problems, maximizing the accuracy can be an alternative). N_{fm} represents the total number of feature mapping nodes, and (N_f, N_m) represents one of the integer factorizations of N_{fm} (i.e. $N_{fm} = N_f \times N_m$). S is a subset of the feasible solutions containing the optimal schemes of node numbers. The hyperparameters may have multiple optimum and all of them are included in set S, which is derived from the following

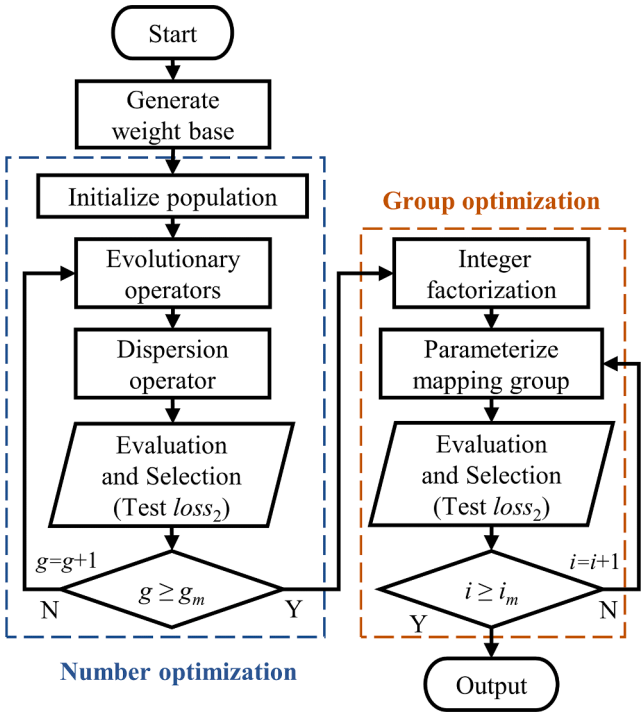

Fig. 3. Flowchart of evolutionary bilevel programming.

lower level problem,

$$S(N_{fm}, N_h) := \underset{N_{fm}, N_h}{\operatorname{argmin}} loss_2(N_{fm}, N_h) \tag{44}$$

Intuitively, the upper level problem needs to be processed on the results of the lower level problem. The objective function of the lower level problem $loss_2$ focuses on the number of nodes, and is defined as the training error on the data set with a fixed single group. In order to solve the problems (43) and (44), the EBP method proposed in this section successively performs optimization of the lower level variables N_{fm} and N_h and group optimization of the upper level variables N_f and N_m. The procedure of EBP is shown in Fig. 3, where g_m and i_m are the maximum number of generations in the Number Optimization and Group Optimization phases respectively. We keep the involved random weights and biases consistent throughout the optimization process by a random seed to stabilize the result.

Number optimization for feature mapping nodes and enhancement nodes is the first part of EBP (shown in the blue box in Fig. 3). The lower level problem (44) can be optimized by population-based *meta*-heuristic methods such as evolutionary algorithms or swarm-techniques [35,36] that involve operators like mutation, crossover, and selection. In this work we use the differential evolution (DE) [37,38] for the optimization.

The initial population with NP individuals is first generated by random sampling in the feasible region. Before the stopping criterion (typically set as the maximum generation g_m) arrives, the evolutionary operators perform on the population in each generation and retain the elite solution as the offspring following the elitist principle. The diversity of offspring is derived from the randomness introduced by mutation operator. The ith mutant individual in the gth generation can be expressed as

$$\boldsymbol{v}_i^g = \boldsymbol{x}_{r1}^g + F \cdot (\boldsymbol{x}_{r2}^g - \boldsymbol{x}_{r3}^g) \tag{45}$$

where $\boldsymbol{x}_{r1}^g$, $\boldsymbol{x}_{r2}^g$ and $\boldsymbol{x}_{r3}^g$ are the randomly selected individuals from the parent population, and the index $r1$, $r2$, $r3 \in [1,NP]$. F is the scaling factor, which controls the amplitude of differential variation. The crossover operator is then performed on each dimension on the basis of the mutant individual $\boldsymbol{v}_i^g$ and the current individual $\boldsymbol{x}_i^g$. The jth dimension of the ith individual after crossover can be expressed as,

$$\boldsymbol{u}_{ij}^g = \begin{cases} \boldsymbol{v}_{ij}^g & if rand(0,1) < r_c or j = j_{rand} \\ \boldsymbol{x}_{ij}^g & otherwise \end{cases} \tag{46}$$

where r_c is the crossover rate in (0,1], $rand(0,1)$ represents a random number in [0,1], and j_{rand} is a random integer, ensuring that at least one of the attribute is mutated.

Since optimizing the number of nodes is a discrete optimization problem, its feasible region consists of integers within a given range. Whereas evolutionary algorithms rely on the differences between individuals in the population. New individuals cannot be generated if all individuals in the population have the same value in one dimension. If the solution at this time is not globally optimal, the population can never escape from the local optimum, which weakens the exploration ability of the algorithm. Therefore, we propose a probabilistic dispersion operator to surmount this drawback. The dispersion probability is calculated for each dimension of the crossover population, and the probability of the dth dimension is

$$r_d = 1 - \frac{\sigma_d}{\mu_d + \varepsilon_0} \tag{47}$$

where σ_d and μ_d represent the standard deviation and mean value of the dth dimension computed over all individuals in the crossover population. σ_d/μ_d is the coefficient of variation, which is a dimensionless indicator in statistics between (0,1] to measure dispersion degree of a set of data. Since μ_d could be zero, a tiny number ε_0 is added to prevent the denominator from being 0. Based on the probability r_d, the dth dimension of the ith trial individual is obtained by

$$\boldsymbol{\delta}_{id}^g = \begin{cases} \boldsymbol{u}_{id}^g + \boldsymbol{i}_{rand} rand(0,1) < r_d \\ \boldsymbol{u}_{id}^g otherwise \end{cases} \tag{48}$$

where $\boldsymbol{\delta}$ represents the offspring after adding disturbance, and the integer i_{rand} represents the random integer perturbation. Dispersion operator can adaptively add perturbation to the dimension tending to convergence without affecting other dimensions, so as to ensure generating new individuals in discrete problems and improve the global search performance and population diversity. Finally, the evaluation and selection operator compares the fitness value of the ith current individual and the ith trial individual to obtain the offspring individual

$$\boldsymbol{x}_i^{g+1} = \begin{cases} \boldsymbol{\delta}_i^g & if loss_2(\delta_i^g) < loss_2(x_i^g) \\ \boldsymbol{x}_i^g & otherwise \end{cases} \tag{49}$$

Group optimization for feature mapping nodes is the second part of EBP (shown in the orange box in Fig. 3) which is activated after number optimization phase terminates. The upper level problem (43) is optimized via integer factorization on the numbers of feature mapping nodes of the best n_e results obtained in the first step. For example, by integer factorization on the number of feature mapping nodes of the solution $(N_{fm},N_h)=(6,3)$, three strategies (1,6,3), (2,3,3) and (3,2,3) can be obtained besides the single grouping.

A variety of algorithms can be used for integer factorization, and a simple example is provided in the Supplementary Materials.

Assuming a total of i_m grouping strategies are obtained, these strategies are all parameterized with the corresponding number of enhancement nodes and then greedily selected for the optimal result with the least error. This grouping strategy together with the number of enhancement nodes is considered as the optimum of hyperparameters to output.

Remark 4. *Optimization of hyperparameters will inevitably demand additional computational costs. Next we analyse the computational overhead of the BEP and grid search scheme. For simplicity, we use an ensemble representation for the computational complexity, i.e., we denote by* $O(M)$ *the computational complexity of one time training of BLS (more specifically, the learning of output weights* $\boldsymbol{W}$ *by pseudo inverse or iterated projection), and by* $O(E)$ *the computational complexity of the evolutionary operators* (45) – (49) *in each generation. For the grid search, all the possible combinations of hyperparameters need to be examined, so the computational complexity of a complete optimization process is* $(N_f \times N_m \times N_h) \times O(M)$. *While for the proposed EBP method, only* $(g_m + i_m) \times O(M) + g_m \times O(E)$ *is involved in the cost of hyperparameter optimization. Since the evolutionary operators adopt only a few linear algebraic operations on vectors, it is sensible to omit the additional term* $O(E)$ *compared with the training cost* $O(M)$. *Hence, our method can significantly reduce the computational cost of hyperparameter optimization with a limited budget of* $(g_m + i_m) \times O(M)$ *instead of the exhausted grid search with a computational overhead of* $(N_f \times N_m \times N_h) \times O(M)$. *Empirical evidence is presented in the next section to prove the efficiency of EBP.*

5. Experiments

In this section, numerical experiments are designed to compare the performance of iterated projection (IP) and pseudo-inverse (PI) in parameter training, and evolutionary bilevel programming (EBP) and grid search in hyperparameter optimization, respectively, so as to demonstrate the accuracy, efficiency and robustness of the proposed methods. So far, pseudo-inverse and grid search have been the primary methods of state-of-the-art BLS variants for parameter training and hyperparameter optimization [1,14,15,29]. Therefore, our experiments will focus on the comparison with methods of parameter training and hyperparameter optimization rather than on structure improvement. Herein, a typical BLS is chosen as the framework for universality, but note that the proposed methods are applicable to all variants of the learning systems trained by pseudo-inverse.

Ten regression data sets taken from the UCI machine learning repository [39] and ten classification data sets obtained from the KEEL [40] repository are used to compare the performance in practical problems in the fields of nature, engineering, medicine, and finance. In order to demonstrate the effectiveness of the proposed method, the selected data sets cover a wide variety having small, medium, and high-dimensional data. The training set size also vary widely. More details of the data sets are listed in Table 1. The code of BLS is provided by [41] and the programs are implemented on the MATLAB 2020a platform with 3.20 GHz CPU and 16 GB memory. All results were averaged from 10 repeated trials. The error and accuracy mentioned in the results refer to the RMSE and the proportion of correct classification respectively, and the training time refers to the time required to train the model. All times in experimental results are in seconds and the magnitude of weights $\boldsymbol{W}$ is measured by entry wise $l1$ norm. We note that each numerical experiment is independent of each other.

5.1. Regression: Iterated projection learning

We first test three modes of iterated projection learning on 10 regression datasets and compare them with ridge regression approximation of pseudo-inverse. For simplicity, the proposed iterated projection learning method is represented by IP and the pseudo-inverse based method is represented by PI. For the hyperparameters of the two training methods, the regularization parameter of PI is

Table 1
Detailed Information for Data Sets.

Data set	Abbr.	Task	Training set size	Test set size	Attributes	Output/Class Number
Leukemia	LE	Classification	65	7	7129	2
Fertility	FE	Classification	90	10	9	2
Iris plants	IR	Classification	135	15	4	3
Sonar, Mines vs. Rocks	SO	Classification	187	21	60	2
Stat Log (Heart)	ST	Classification	243	27	13	2
Musk	MU	Classification	428	48	166	2
Breast Cancer Wisconsin (Diagnostic)	BR	Classification	512	57	30	2
Balance Scale	BA	Classification	563	62	4	3
Pima Indians	PI	Classification	691	77	8	2
Waveform	WA	Classification	4500	500	21	3
Heart Failure	HE	Regression	239	60	12	1
Residential Building	RE	Regression	273	99	103	2
South German Credit	SO	Regression	800	200	20	1
South German Credit Update	SU	Regression	800	200	20	1
Garments Worker Productivity	GA	Regression	958	239	12	1
Wine Quality (red)	WR	Regression	1280	319	11	1
Wine Quality (white)	WW	Regression	3920	978	11	1
Energy Data Complete	EN	Regression	15,788	3947	26	1
Blog Feedback	BL	Regression	52,397	7624	280	1
Transcoding Measurement	TR	Regression	55,027	13,757	18	1

Table 2
Comparison of Offline Mode Performance Between PI and IP for Regression.

Data set	Training error		Testing error		Training time		Magnitude of $\boldsymbol{W}$	
	PI	IP	PI	IP	PI	IP	PI	IP
HE	3.240E-01	4.558E-01	5.202E + 00	**4.883E-01**	0.018	**0.008**	1.094E + 05	**2.189E + 00**
RE	3.183E + 02	8.655E + 02	3.822E + 05	**1.421E + 03**	0.018	**0.013**	7.560E + 08	**5.284E + 03**
SO	3.580E-01	4.549E-01	8.041E + 01	**4.356E-01**	0.028	**0.011**	2.706E + 05	**2.864E + 00**
SU	3.625E-01	4.522E-01	5.998E-01	**4.447E-01**	0.029	**0.011**	2.231E + 05	**3.418E + 00**
GA	1.314E-01	1.727E-01	5.945E-01	**1.758E-01**	0.032	**0.015**	6.640E + 04	**1.982E + 00**
WR	5.322E-01	7.593E-01	8.806E + 00	**7.908E-01**	0.046	**0.021**	2.028E + 05	**2.429E + 01**
WW	6.789E-01	8.517E-01	4.211E + 02	**7.435E-01**	0.104	**0.049**	3.400E + 05	**1.788E + 01**
EN	1.057E + 01	1.163E + 01	3.063E + 01	**1.158E + 01**	0.362	**0.112**	3.915E + 06	**1.165E + 02**
BL	2.671E + 01	3.474E + 01	9.918E + 02	**2.798E + 01**	1.104	**0.196**	5.945E + 07	**2.498E + 02**
TR	1.135E + 01	1.602E + 01	4.222E + 02	**1.595E + 01**	1.186	**0.204**	2.395E + 07	**5.281E + 01**

set as 10^{-8}, and the iteration number of the proposed method is set as 2800 for all data sets.

For offline learning mode, two sets of experiments are designed to analyze the efficiency and robustness of the proposed method respectively. The first set of experiments aims to compare the accuracy and efficiency of the two training methods through a fixed system size, where the structure of the model is fixed as $N_f = 25$, $N_m = 20$, and $N_h = 500$. As shown in Table 2, training errors for PI are lower than the corresponding training errors for IP. However, the test errors for IP are lower than the corresponding test errors for PI. At the first sight it may appear counterintuitive, but it is not. There could be at least two possible reasons for better generalization of the IP scheme. Given a training data set, PI finds the optimal model which depends on the coefficient of the regularization also. Moreover, minimum training error does not necessarily imply better generalization as the system may even memories some data. The IP system, on the other hand, is an approximate system and the trained model has much lower norm of the weight parameters. Hence it is expected to show a better generalization. The proposed method has better efficiency than PI in terms of training time, which is more visible for large data sets. This is because PI processes the information of all data points at one time, while the proposed method only selects one point for projection within the set iteration number, which avoids wasting computational cost in redundant information. For the same system size, the $l1$ norm of output weights obtained by the proposed method is several orders of magnitude smaller than that by PI, which is also consistent with our expectation to obtain a lightweight model without regularization parameter tuning.

The second set of offline learning experiments is designed to demonstrate the robustness of the two methods for hyperparameter setting through 400 groups of experiments of different system sizes. We set the number of nodes in each feature mapping node group and enhancement node group as 25, and tested the model error for grouping from 1 to 20, and thus obtain 20×20 combinations. Experimental results are presented in mesh graphs (the results of the HE data set are shown in Fig. 4, and the others are in the Supplementary Materials). From the mesh graphs, it is evident that the choice of node number has a significant impact on PI performance, and an inappropriate grid size will lead to a large test error. In contrast, the performance of the proposed method is relatively stable under the change in hyperparameters. The mean and standard deviation of test error and those of the norm (magnitude) of $\boldsymbol{W}$ for all data sets are listed in Table 3. Table 3 reveals that the proposed method enjoys stronger robustness. But note that this does not mean that the performance of proposed method is not dependent on the setting of hyperparameters. From the second set of experiments, we can conclude that although both methods aim at minimizing errors, the proposed method can obtain weights with lower magnitude, so it is expected to have better generalization ability and is not likely to be affected strongly by the size of the system.

The online learning mode is implemented by randomly arranging data points to form the data stream for training the models. To enhance the readability of the visualized results, we only present the results of the first two thousand steps of online learning for large data sets. The online training results of the two methods are shown in Table 4. Obviously, compared with the training time of offline mode shown in Table 2, both proposed method and PI inevitably increase computational costs because of the need to progressively revise the model. According to the final test error, the proposed method achieves an error similar to that of the offline mode in Table 2. However, PI shows a tendency of invalidation in all data sets except WW. It can be observed from the blue curve in Fig. 5 that the proposed method can achieve effective data incremental learning, while the accuracy of PI is attenuated. One might wonder why the accuracy of PI decreases with incremental learning in most data sets, which clearly contradicts the common sense. As we have analyzed

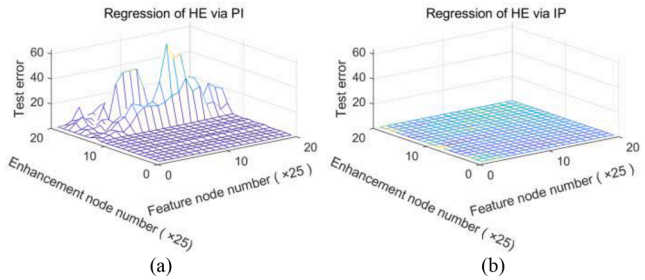

Fig. 4. Mesh graphs of test error and node number.

Table 3
Statistics of the Test Errors of Robustness Experiments for Regression.

Data set	Mean (STD) of testing error		Mean (STD) of magnitude of $\boldsymbol{W}$	
	PI	IP	PI	IP
LE	5.054E + 00 (8.792E + 00)	**4.868E-01 (1.213E-03)**	1.144E + 05 (5.419E + 04)	**9.371E-01 (1.499E-01)**
FE	9.399E + 04 (8.967E + 04)	**1.528E + 03 (1.990E + 01)**	5.943E + 08 (2.041E + 08)	**5.819E + 03 (2.851E + 02)**
IR	1.088E + 02 (1.024E + 02)	**4.369E-01 (4.998E-04)**	2.070E + 05 (5.023E + 04)	**9.686E-01 (5.361E-02)**
SO	4.890E-01 (8.441E-02)	**4.424E-01 (9.381E-04)**	2.000E + 05 (3.865E + 04)	**9.514E-01 (4.719E-02)**
ST	6.306E-01 (5.493E-01)	**1.759E-01 (9.544E-04)**	5.129E + 04 (1.167E + 04)	**1.109E + 00 (1.046E-01)**
MU	1.884E + 00 (2.247E + 00)	**7.944E-01 (1.713E-02)**	1.702E + 05 (1.160E + 05)	**1.077E + 01 (5.527E-01)**
BR	2.468E + 02 (2.067E + 02)	**7.441E-01 (6.481E-03)**	2.960E + 05 (1.728E + 05)	**9.748E + 00 (7.433E-01)**
BA	1.769E + 01 (2.085E + 01)	**1.164E + 01 (2.347E-02)**	2.329E + 06 (1.477E + 06)	**6.491E + 01 (4.448E + 00)**
PI	1.567E + 02 (1.881E + 02)	**2.824E + 01 (2.062E-01)**	4.823E + 07 (2.387E + 07)	**1.673E + 02 (4.301E + 01)**
WA	3.552E + 02 (2.973E + 02)	**1.596E + 01 (5.717E-03)**	2.906E + 07 (2.259E + 07)	**2.628E + 01 (3.466E + 00)**

Table 4
Comparison of Online Mode Performance Between PI and IP for Regression.

Data set	Testing error		Training time		Magnitude of $\boldsymbol{W}$	
	PI	IP	PI	IP	PI	IP
HE	1.557E + 01	**5.226E-01**	10.041	**0.467**	1.632E-01	**1.745E-04**
RE	6.772E + 06	**1.518E + 03**	11.555	**0.719**	6.649E + 01	**1.962E-02**
SO	5.608E + 01	**4.208E-01**	32.358	**5.619**	3.962E + 00	**1.040E-01**
SU	1.074E + 02	**4.158E-01**	32.354	**5.807**	2.394E + 00	**1.161E-01**
GA	8.358E + 01	**2.352E-01**	38.880	**8.704**	3.493E + 00	**2.621E-01**
WR	3.441E + 02	**7.008E-01**	28.161	**20.075**	2.735E + 02	**4.754E + 00**
WW	3.121E + 02	**7.112E-01**	43.824	**59.093**	1.272E + 02	**4.023E + 00**
EN	1.849E + 03	**1.163E + 01**	83.872	**60.729**	1.392E + 02	**1.057E + 00**
BL	7.403E + 03	**2.789E + 01**	83.029	**58.721**	7.161E + 02	**2.123E + 00**
TR	1.749E + 07	**1.325E + 01**	86.331	**59.214**	6.183E + 00	**2.230E-04**

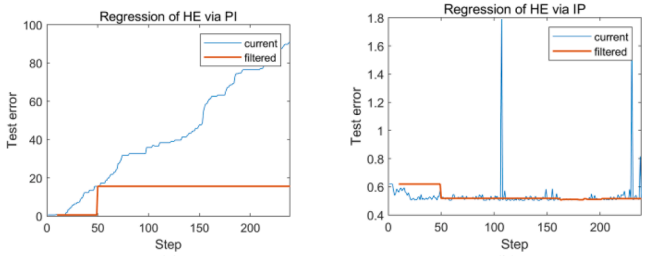

Fig. 5. Plots of the test error and online learning step.

in the introduction, a fixed regularization parameter cannot meet the dynamic requirements of online learning based on PI, and the rapid growth of $l1$ norm of output weight leads to the increase of error. This phenomenon of attenuation in accuracy has already been discussed in [29] and the experimental results are similar to those of ours. Different from the existing work, the proposed method can avoid this problem without introducing additional computational costs. In the case of PI, the attenuation of accuracy may be caused not only because of improper regularization, but also because only one row of the matrix is processed in each step, which makes the matrix close to the singular value and thus leads to an inaccurate solution. The proposed method does not need to consider this problem due to the manner of projection by rows. In addition, the orange curve in Fig. 5 shows the effect of the sliding window filter with size n_w as 50 on stabilizing the system. The results demonstrate that the adjacent training error significantly eliminates the fluctuation of the test error.

In order to test the effect of model adjustment by incremental learning mode, a five-step incremental learning is performed on the initial model. For each step, we add 25 feature mapping nodes, 20 enhancement nodes, and 20 enhancement nodes connected to the new feature mapping nodes. And the iteration number is set to min(M,1000). Table 5 lists the intermediate and final results of incremental learning via two training methods, where time represents the sum of training time of all incremental steps. The weight increment calculating by $\sum W^S - \sum W^0$, represents the magnitude difference between the weights of the final model W^S and the weights of the initial model W^0. Similar to online learning, learning with incremental nodes based on pseudo-inverse of partitioned matrix also exhibits attenuation of accuracy. It can be observed that the weights of the model trained by PI increase rapidly in incremental learning, indicating that regularization is not performed properly possibly because of the use of a fixed parameter. On the contrary, the

Table 5
Comparison of Incremental Mode Performance Between PI and IP for Regression.

Data set		Initial	Step 1	Step 2	Step 3	Step 4	Step 5	Time	Magnitude increment
HE	PI	2.148E + 00	1.911E + 00	2.484E + 00	4.042E + 00	8.031E + 00	1.014E + 01	0.048	4.648E + 06
	IP	**4.871E-01**	**4.873E-01**	**4.876E-01**	**4.875E-01**	**4.875E-01**	**4.876E-01**	**0.020**	**1.036E-01**
RE	PI	2.240E + 05	1.966E + 05	1.871E + 05	2.751E + 05	6.691E + 05	1.702E + 06	0.040	9.036E + 09
	IP	**1.486E + 03**	**1.530E + 03**	**1.542E + 03**	**1.521E + 03**	**1.555E + 03**	**1.544E + 03**	**0.031**	**4.832E + 02**
SO	PI	1.041E + 02	8.257E + 01	1.886E + 02	3.673E + 02	6.989E + 02	7.499E + 02	0.083	1.488E + 06
	IP	**4.359E-01**	**4.359E-01**	**4.362E-01**	**4.354E-01**	**4.359E-01**	**4.357E-01**	**0.029**	**1.340E-01**
SU	PI	5.112E-01	5.241E-01	1.172E + 00	3.427E + 00	6.777E + 00	1.056E + 01	0.079	4.976E + 06
	IP	**4.437E-01**	**4.433E-01**	**4.433E-01**	**4.435E-01**	**4.436E-01**	**4.431E-01**	**0.026**	**2.129E-01**
GA	PI	4.553E-01	6.587E-01	6.635E-01	2.112E + 00	3.475E + 00	4.652E + 00	0.098	1.607E + 06
	IP	**1.761E-01**	**1.755E-01**	**1.759E-01**	**1.757E-01**	**1.759E-01**	**1.759E-01**	**0.034**	**3.810E-01**
WR	PI	8.172E + 00	2.123E + 01	2.206E + 01	3.987E + 01	3.766E + 01	5.463E + 01	0.132	5.808E + 06
	IP	**7.672E-01**	**8.372E-01**	**7.670E-01**	**7.643E-01**	**7.690E-01**	**7.651E-01**	**0.043**	**4.680E + 00**
WW	PI	3.079E + 01	1.510E + 02	6.278E + 02	6.671E + 02	5.931E + 02	2.238E + 02	0.360	3.600E + 06
	IP	**7.448E-01**	**7.415E-01**	**7.417E-01**	**7.459E-01**	**7.416E-01**	**7.386E-01**	**0.130**	**2.514E + 00**
EN	PI	2.052E + 01	5.731E + 01	1.916E + 01	5.708E + 01	7.660E + 01	1.718E + 03	1.398	1.142E + 10
	IP	**1.160E + 01**	**1.159E + 01**	**1.159E + 01**	**1.158E + 01**	**1.156E + 01**	**1.156E + 01**	**0.400**	**1.880E + 01**
BL	PI	1.585E + 03	2.549E + 02	5.093E + 02	1.454E + 03	2.717E + 03	5.231E + 03	4.666	4.384E + 08
	IP	**2.806E + 01**	**2.806E + 01**	**2.805E + 01**	**2.802E + 01**	**2.808E + 01**	**2.799E + 01**	**0.963**	**8.603E + 01**
TR	PI	7.103E + 02	5.172E + 03	8.332E + 03	1.193E + 04	2.047E + 04	2.822E + 04	4.802	1.297E + 09
	IP	**1.595E + 01**	**1.595E + 01**	**1.595E + 01**	**1.594E + 01**	**1.594E + 01**	**1.594E + 01**	**0.981**	**8.557E + 00**

proposed method can keep the weight magnitude stable and complete the incremental learning process in less time. The test errors of the two training methods demonstrate that the proposed method can accomplish structure increment effectively without any tuning of the regularization parameter.

5.2. *Classification: Iterated projection learning*

By setting the same hyperparameters as the regression problems, we also compare the performance the three modes of PI and the proposed method on 10 classification data sets. The difference between the classification problem and the regression problems is that the accuracy is affected not only by the prediction error (square error), but also by the distribution of magnitude the classification outputs. Thus, the minimum square error may not necessarily lead to the minimum classification error even for the training data. In this part we will focus on the differences between the results of the two methods in classification and regression.

The efficiency and robustness of the proposed method are verified by two sets of experiments designed in offline mode identical to the regression problems. First, the test results under a fixed system structure are shown in Table 6. The performance of the two methods is consistent with that for the regression problems. The proposed method can obtain more accurate and lightweight models with the given hyperparameters, and is computationally efficient.

Similarly, 20×20 groups of experiments consistent with the regression problems are designed to show the robustness through a mesh graph of test accuracy (again, we show the FE data set in Fig. 6, and the others in the Supplementary Materials). The average test errors, test accuracies, and norm of the weights along with their respective standard deviations are included in Table 7. For most data sets, the proposed method exhibits higher or at least similar accuracies. For all data sets, the IP method obtains lower norm of the model weights. However, on BA and WA data sets, we observe that although the proposed method achieves better performance in terms of test error, it fails to make more accurate classification. We believe this is related to the least square loss function. Reducing errors is more of an indirect measure than directly improving accuracy because systems make judgment based on the maximum output over the classes.

The results of the online mode for the classification problems are shown in Table 8. The first data set is chosen to illustrate the

Table 6
Comparison of Offline Mode Performance Between PI and IP for Classification.

Data set	Training error		Training accuracy		Testing error		Testing accuracy		Training time		Magnitude of $\boldsymbol{W}$	
	PI	IP	PI	IP	PI	IP	PI	IP	PI	IP	PI	IP
LE	0.000	0.132	100.00%	100.00%	0.419	**0.373**	98.57%	**100.00%**	0.025	**0.022**	1.159E + 01	**9.134E + 00**
FE	0.149	0.393	98.89%	95.56%	1.041	**0.839**	77.00%	**80.00%**	0.026	**0.021**	3.068E + 01	**8.041E + 00**
IR	0.000	0.387	100.00%	98.22%	0.355	**0.295**	**100.00%**	**100.00%**	0.027	**0.025**	2.181E + 02	**8.131E + 00**
SO	0.000	0.515	100.00%	94.44%	1.009	**0.761**	83.33%	**87.62%**	0.029	**0.021**	6.007E + 01	**1.284E + 01**
ST	0.003	0.834	100.00%	74.77%	21.063	**0.893**	53.70%	**64.81%**	0.029	**0.022**	3.492E + 04	**5.961E + 00**
MU	0.000	0.669	100.00%	86.26%	3.467	**0.678**	65.63%	**85.42%**	0.035	**0.023**	3.500E + 03	**1.219E + 01**
BR	0.260	0.491	98.55%	92.48%	150.423	**0.556**	86.49%	**87.02%**	0.037	**0.030**	1.871E + 05	**4.823E + 00**
BA	0.460	0.890	98.76%	87.67%	20.499	**0.816**	81.77%	**90.65%**	0.038	**0.033**	1.892E + 05	**9.678E + 00**
PI	0.497	0.774	93.04%	78.10%	25.667	**0.815**	65.19%	**75.06%**	0.042	**0.035**	1.406E + 05	**5.734E + 00**
WA	0.778	0.897	89.14%	84.81%	1.005	**0.891**	84.96%	**85.66%**	0.166	**0.161**	1.119E + 05	**1.068E + 01**

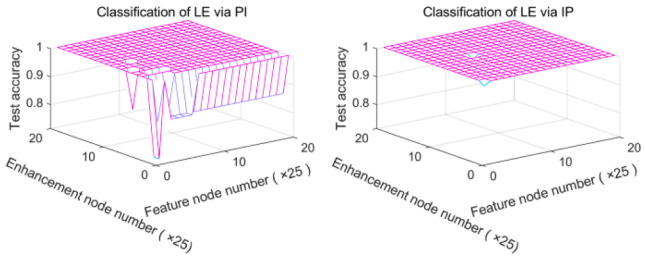

Fig. 6. Mesh graphs of test accuracy and node number.

Table 7
Statistics of the Test Errors and Test Accuracy of Robustness Experiments for Classification.

Data set	Mean (STD) of testing error		Mean (STD) of testing accuracy		Mean (STD) of magnitude of $\boldsymbol{W}$	
	PI	IP	PI	IP	PI	IP
LE	0.367 (0.130)	**0.314 (0.051)**	98.79% (4.23%)	**99.99% (0.10%)**	2.983E + 01 (4.155E + 01)	**1.148E + 01 (1.803E + 00)**
FE	3.549 (12.897)	**0.860 (0.059)**	72.90% (10.00%)	**79.38% (1.93%)**	1.503E + 03 (4.306E + 03)	**8.159E + 00 (7.609E-01)**
IR	1.646 (2.116)	**0.322 (0.047)**	94.43% (6.74%)	**99.24% (1.73%)**	1.369E + 04 (2.353E + 04)	**1.038E + 01 (1.779E + 00)**
SO	1.948 (1.870)	**0.802 (0.053)**	71.73% (10.08%)	**86.03% (3.03%)**	2.210E + 04 (3.692E + 04)	**1.368E + 01 (1.120E + 00)**
ST	8.132 (10.119)	**0.887 (0.009)**	60.82% (12.78%)	**65.55% (1.64%)**	7.467E + 04 (4.794E + 04)	**6.029E + 00 (6.060E-01)**
MU	2.700 (2.703)	**0.703 (0.033)**	74.69% (9.28%)	**81.47% (3.38%)**	1.092E + 05 (9.087E + 04)	**1.307E + 01 (1.032E + 00)**
BR	102.380 (97.930)	**0.558 (0.014)**	**90.80%** (2.94%)	88.24% **(0.96%)**	1.436E + 05 (5.041E + 04)	**6.131E + 00 (1.084E + 00)**
BA	4.330 (5.394)	**0.817 (0.011)**	88.67% (3.63%)	**89.79% (0.94%)**	1.591E + 05 (1.030E + 05)	**1.254E + 01 (2.633E + 00)**
PI	7.385 (12.869)	**0.813 (0.009)**	71.48% (5.27%)	**76.86% (1.07%)**	7.685E + 04 (5.153E + 04)	**6.642E + 00 (1.095E + 00)**
WA	0.985 (0.282)	**0.954 (0.034)**	**84.85% (1.04%)**	83.87% (1.05%)	5.720E + 04 (3.999E + 04)	**1.299E + 01 (1.437E + 00)**

Table 8
Comparison of Online Mode Performance Between PI and IP for Classification.

Data set	Testing accuracy		Training time		Magnitude of $\boldsymbol{W}$	
	PI	IP	PI	IP	PI	IP
LE	57.14%	**100.00%**	3.943	**0.066**	4.082E-01	**6.192E-05**
FE	80.00%	**80.00%**	2.550	**0.106**	4.451E + 01	**2.940E + 00**
IR	20.00%	**100.00%**	3.940	**0.276**	9.116E + 01	**3.869E + 00**
SO	38.10%	**85.71%**	5.336	**0.449**	3.035E + 01	**3.599E + 00**
ST	48.15%	**66.67%**	7.244	**0.704**	8.896E + 01	**5.878E + 00**
MU	41.67%	**87.50%**	26.095	**2.333**	7.648E-01	**4.069E-02**
BR	54.39%	**87.72%**	26.816	**3.649**	1.074E + 00	**6.260E-02**
BA	53.23%	**95.16%**	16.378	**5.519**	2.532E + 00	**3.216E-01**
PI	75.32%	**81.82%**	15.017	**5.151**	3.288E + 00	**2.751E-01**
WA	81.00%	**86.80%**	**41.910**	83.093	1.876E + 00	**3.022E-02**

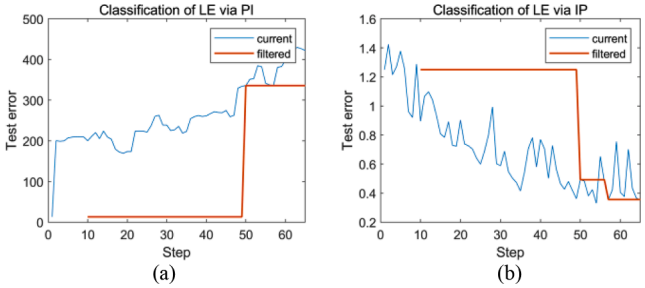

Fig. 7. Plots of the test error and online learning step.

process of error reduction and the effect of using adjacent error filtering in Fig. 7. Here also PI suffers from the problem of accuracy attenuation, but the proposed method can achieve the desired result. Online learning of the proposed method consumes less time on small data sets, but its training cost increases sharply as the data set size increases. This phenomenon is more visible on the classification problems than on the regression problems, because the classification problems need to calculate multiple outputs, which

Table 9
Comparison of Incremental Mode Performance Between PI and IP for Classification.

Data set		Initial	Step 1	Step 2	Step 3	Step 4	Step 5	Time	Magnitude increment
LE	PI	97.14%	95.71%	94.29%	97.14%	97.14%	95.71%	0.034	−2.457E-01
	IP	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**0.025**	4.084E-01
FE	PI	77.00%	77.00%	77.00%	78.00%	78.00%	77.00%	0.034	−7.303E-01
	IP	**80.00%**	**80.00%**	**80.00%**	**80.00%**	**80.00%**	**80.00%**	**0.023**	6.658E-01
IR	PI	99.33%	98.00%	94.67%	70.00%	33.33%	22.00%	0.048	7.885E + 06
	IP	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**100.00%**	**0.026**	4.898E-01
SO	PI	81.90%	77.14%	73.33%	69.05%	61.43%	55.71%	0.052	6.191E + 06
	IP	**87.62%**	**87.14%**	**86.67%**	**87.14%**	**87.14%**	**87.14%**	**0.024**	1.952E + 00
ST	PI	53.70%	52.59%	50.37%	52.22%	47.78%	47.04%	0.067	4.248E + 07
	IP	**64.81%**	**64.07%**	**64.81%**	**64.44%**	**64.81%**	**65.19%**	**0.028**	1.361E + 00
MU	PI	66.67%	49.17%	48.96%	47.92%	47.92%	50.63%	0.086	1.991E + 09
	IP	**85.83%**	**84.79%**	**85.63%**	**86.46%**	**85.63%**	**86.46%**	**0.040**	4.003E + 00
WD	PI	87.02%	85.96%	85.09%	82.28%	80.70%	81.05%	0.105	2.117E + 06
	IP	**87.54%**	**87.72%**	**87.72%**	**87.72%**	**87.72%**	**87.72%**	**0.044**	1.129E + 00
BA	PI	80.48%	78.87%	74.19%	71.45%	69.35%	65.00%	0.105	2.537E + 06
	IP	**89.68%**	**89.84%**	**90.16%**	**90.48%**	**90.48%**	**90.00%**	**0.050**	1.577E + 00
PI	PI	67.79%	65.84%	63.51%	62.08%	60.26%	59.48%	0.126	2.621E + 06
	IP	**76.49%**	**76.23%**	**77.14%**	**76.88%**	**76.88%**	**76.62%**	**0.051**	2.193E + 00
WA	PI	84.14%	84.06%	84.12%	83.94%	84.14%	83.68%	0.660	1.813E + 07
	IP	**85.60%**	**85.46%**	**85.48%**	**85.54%**	**85.70%**	**85.80%**	**0.370**	5.619E + 00

magnifies the difference several times. Thus, when the size of the data set increases significantly, the proposed method fails to maintain the computational efficiency.

Similar to the regression problems, the intermediate and final results of incremental learning mode for classification problems are listed in Table 9. The proposed method exhibits similar advantages as in the case of regression problems, but the PI results in different levels of performance for different data sets. It can be observed from the accuracy and weight increment of PI that incremental learning based on pseudo-inverse of partitioned matrix performs better here than the previous part. Especially in the data sets FE and LE, there is no drastic increase in the size of the weights and there is not much attenuation in accuracy. It is probably because the setting of regularization parameters is suitable for these problems. However, it does not perform well in other problems. Compared with the initial value in Table 9, for the PI method the weight of the model increases dramatically in the process of incremental learning and the accuracy is also attenuated. On the other hand, the proposed method can avoid this problem using a shorter training time, and the accuracy as well as the $l1$ norm of the weights remain stable in the incremental learning process.

Table 10
Performance Comparison Between Grid Search and Evolutionary Bilevel Programming.

Dataset	Grid search			Evolutionary bilevel programming				
	Final result	Iterations	Time	Number optimization	Group optimization	Final result	Iterations	Time
HE	4.835E-01	250,000	1.327E + 04	4.954E-01	4.953E-01	4.953E-01	**200 + 62**	**1.912E + 01**
RE	1.409E + 03	250,000	1.886E + 04	1.360E + 03	1.388E + 03	1.360E + 03	**200 + 96**	**3.406E + 01**
SO	4.342E-01	250,000	2.035E + 04	4.391E-01	4.387E-01	4.387E-01	**200 + 102**	**5.063E + 01**
SU	4.417E-01	250,000	2.188E + 04	4.455E-01	4.447E-01	4.447E-01	**200 + 77**	**3.406E + 01**
GA	1.757E-01	250,000	2.450E + 04	1.761E-01	1.757E-01	1.757E-01	**200 + 84**	**4.145E + 01**
WR	7.456E-01	250,000	1.304E + 04	7.000E-01	7.406E-01	7.000E-01	**200 + 62**	**2.148E + 01**
WW	7.318E-01	250,000	2.443E + 04	7.316E-01	7.396E-01	7.316E-01	**200 + 48**	**4.756E + 01**
EN	1.154E + 01	250,000	8.161E + 04	1.164E + 01	1.157E + 01	1.157E + 01	**200 + 66**	**1.564E + 02**
BL	2.754E + 01	250,000	2.319E + 05	2.827E + 01	2.799E + 01	2.799E + 01	**200 + 75**	**4.507E + 02**
TR	1.594E + 01	250,000	1.651E + 05	1.594E + 01	1.594E + 01	1.594E + 01	**200 + 51**	**3.585E + 02**

Dataset	Grid search			Evolutionary bilevel programming				
	Final result	Iterations	Time	Number optimization	Group optimization	Final result	Iterations	Time
LE	100.00%	250,000	1.248E+05	100.00%	100.00%	100.00%	**200+41**	**2.623E+02**
FE	80.00%	250,000	1.143E+04	80.00%	80.00%	80.00%	**200+62**	**2.071E+01**
IR	100.00%	250,000	1.311E+04	100.00%	100.00%	100.00%	**200+32**	**6.868E+00**
SO	80.95%	250,000	1.463E+04	80.95%	80.95%	80.95%	**200+56**	**2.657E+01**
ST	62.96%	250,000	1.565E+04	62.96%	66.67%	62.96%	**200+73**	**1.847E+01**
MU	91.67%	250,000	1.889E+04	91.67%	91.67%	91.67%	**200+54**	**3.631E+01**
BR	87.72%	250,000	1.614E+04	89.47%	87.72%	87.72%	**200+54**	**2.283E+01**
BA	95.16%	250,000	1.591E+04	91.94%	91.94%	91.94%	**200+60**	**3.137E+01**
PI	79.22%	250,000	1.707E+04	77.92%	77.92%	77.92%	**200+73**	**3.823E+01**
WA	87.60%	250,000	5.194E+04	84.60%	87.20%	84.60%	**200+46**	**7.157E+01**

5.3. Evolutionary bilevel programming

EBP aims at efficiently obtaining optimal settings of network structure hyperparameters instead of time-consuming grid search method. Regardless of the computational cost, grid search can find the exact global optimum of discrete problems, and theoretically other methods cannot find a better one. However, under the influence of random weights, the global optimum hovers in a certain interval rather than keeping a stable value. Hence, our expectation is to find an optimal value similar to grid search with less computational cost. Here we make a direct comparison with the grid search method on the hyperparameter optimization of node number and grouping. For the first part of EBP, the scaling factor F and crossover rate r_c are both set as 0.5, population size NP is 10, and the maximum number of iterations is set as 200. For the second part of EBP, integer factorization is performed on the best 10 results. The training error is used as the objective function. Note that due to the generalization error, this experiment is different from the second set of experiments in the first two parts which are obtained directly via the testing error.

Table 10 shows the comparison of optimization efficiency between EBP and grid search for regression and classification. Iterations in Table 10 represents the total number of model training times during the optimization process. The two methods obtain similar final results, and the proposed method is significantly better than the grid search in terms of iteration number and optimization time, which confirms the analysis of computation complexity mentioned in Section 4.

6. Conclusion

The focus of this work is to explore a novel approach of training neural network and learning system based on least squares estimation and combine it with BLS, an emerging learning model. We first analyze the feasibility and potential advantages of using the iterative least squares solver to train the output weights of BLS. The main contribution of this work is to improve the iteration efficiency of the iterative method and to propose the iterated projection learning of BLS based on the modified algorithm. The proposed approach is suitable for both offline and online learning modes and it can dynamically add feature mapping nodes and enhancement nodes as required through incremental learning. To address the limitation of grid search, we present the evolutionary bilevel programming to further optimize the hyperparameters of the system, which can find the desirable number and grouping of nodes with a few iterations.

According to the results of the numerical experiments, it can be concluded that compared with the training method based on the pseudo-inverse, the iterated projection learning is computationally more efficient and memory-saving. It is also found to exhibit better generalization ability – based on our limited experiments, the proposed method is found to be less sensitive to the choice of hyperparameters. Our method has exhibited better performance in regression compared to classification, but we believe there is still room for improvement in classification. Although the proposed method has a smaller error in classification problems, this advantage cannot be fully reflected in the accuracy, because the classification is also affected by the relationship between the outputs. Therefore, one direction of our future work is to introduce the relationships between classes into the target linear system by modifying the loss function to improve the classification performance. In addition, the computation of multiple outputs of the classification problem impairs the efficiency of the proposed method, so a more efficient iterative manner can be considered to solve this problem in the future.

In this study we have proposed an innovative incremental learning mechanism both for adding (incremental) data and nodes. We have demonstrated the effectiveness of the proposed learning approach using extensive numerical simulations. In the near future we plan to do rigorous mathematical analysis of the proposed method. In addition, we like to expand the scope of the iterated projection methods with a view to establishing a universal and systematic machine learning paradigm. Since the proposed iterative method can be trained by a single sample instead of loading all the data at once, it has a broad application prospect in dealing with big data problems, such as large databases with distributed storage. Limited by the property of the classical structure, our experiments only involve some typical machine learning data sets. Actually, the iterated projection learning proposed in this paper can be extended to the training of BLS variants and other kinds of neural networks based on least squares estimation, such as TS type [42] and AnYa [43] type fuzzy neural networks [44–47], and RVFLNN. Typical BLS structure is not suitable for complex large-scale problems such as computer vision or natural language processing. In the future, we will attempt to the combine the proposed iterated projection learning with the state-of-the-art neural networks and learning systems. Moreover, in Remark 2, using Lemma 1 and Lemma 2, we have provided an intuitive justification that the proposed learning scheme is expected to have a better generalization ability. As explained in Remark 2, the theoretical basis for improving the generalization ability in this work lies in the convergence property to the least norm solution of the iterated projection method. However, it would be interesting and useful to have a rigorous mathematical proof for the same. We would like to take this up as one of our future works. We hope that this work can provide a new research direction and application scenario for the future work of machine learning based on least squares estimation.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgements

This work is supported by the National Key Research and Development Program of China under Grant 2019YFA0708700; the National Natural Science Foundation of China under Grant 62173345, 52274057, 52074340 and 51874335; the Fundamental Research Funds for the Central Universities under Grant 22CX03002A; the Major Scientific and Technological Projects of CNOOC under Grant CCL2022RCPS0397RSN; the Science and Technology Support Plan for Youth Innovation of University in Shandong Province under Grant 2019KJH002; 111 Project under Grant B08028; the China-CEEC Higher Education Institutions Consortium Program under Grant 2022151; the Introduction Plan for High Talent Foreign Experts under Grant DL2023152001L; and the "The Belt and Road" Innovative Talents Exchange Foreign Experts Project under Grant G2023152012L.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.ins.2023.119648.

References

[1] C.L.P. Chen, Z. Liu, Broad Learning System: An Effective and Efficient Incremental Learning System Without the Need for Deep Architecture, IEEE Transactions on Neural Networks and Learning Systems 29 (1) (2018) 10–24, https://doi.org/10.1109/TNNLS.2017.2716952.
[2] C.L.P. Chen, Z. Liu, S. Feng, Universal Approximation Capability of Broad Learning System and Its Structural Variations, IEEE Transactions on Neural Networks and Learning Systems 30 (4) (2019) 1191–1204, https://doi.org/10.1109/TNNLS.2018.2866622.
[3] D. Liu, S. Baldi, W. Yu, J. Cao, W. Huang, On Training Traffic Predictors via Broad Learning Structures: A Benchmark Study, IEEE Transactions on Systems, Man, and Cybernetics: Systems 52 (2) (2022) 749–758, https://doi.org/10.1109/TSMC.2020.3006124.
[4] Y. Pao, Y. Takefuji, Functional-link net computing: theory, system architecture, and functionalities, Computer 25 (5) (1992) 76–79, https://doi.org/10.1109/2.144401.
[5] Y.-H. Pao, G.-H. Park, D.J. Sobajic, Learning and generalization characteristics of the random vector functional-link net, Neurocomputing 6 (2) (1994) 163–180, https://doi.org/10.1016/0925-2312(94)90053-1.
[6] B. Igelnik, P. Yoh-Han, Stochastic choice of basis functions in adaptive function approximation and the functional-link net, IEEE Transactions on Neural Networks 6 (6) (1995) 1320–1329, https://doi.org/10.1109/72.471375.
[7] X. Gong, T. Zhang, C.L.P. Chen, Z. Liu, Research Review for Broad Learning System: Algorithms, Theory, and Applications, IEEE Transactions on Cybernetics 99 (2021) (2021) 1–29, https://doi.org/10.1109/TCYB.2021.3061094.
[8] S. Wu, J. Wang, H. Sun, K. Zhang, N.R. Pal, Fractional Approximation of Broad Learning System, IEEE Transactions on Cybernetics (2022) 1–14, https://doi.org/10.1109/TCYB.2021.3127152.
[9] L. Zhang, et al., Analysis and Variants of Broad Learning System, IEEE Transactions on Systems, Man, and Cybernetics: Systems 99 (2020) (2020) 1–11, https://doi.org/10.1109/TSMC.2020.2995205.
[10] D. Liu, S. Baldi, W. Yu, C.L.P. Chen, A Hybrid Recursive Implementation of Broad Learning With Incremental Features, IEEE Transactions on Neural Networks and Learning Systems 33 (4) (2022) 1650–1662, https://doi.org/10.1109/TNNLS.2020.3043110.
[11] F. Yang, "A CNN-Based Broad Learning System," in *2018 IEEE 4th International Conference on Computer and Communications (ICCC)*, 7-10 Dec. 2018 2018, pp. 2105-2109, 10.1109/CompComm.2018.8780984.
[12] Q. Feng, Z. Liu, C.L.P. Chen, Broad and deep neural network for high-dimensional data representation learning, Information Sciences 599 (2022) 127–146, https://doi.org/10.1016/j.ins.2022.03.058.
[13] M. Xu, M. Han, C.L.P. Chen, T. Qiu, Recurrent Broad Learning Systems for Time Series Prediction, IEEE Transactions on Cybernetics 50 (4) (2020) 1405–1417, https://doi.org/10.1109/TCYB.2018.2863020.
[14] S. Feng, C.L.P. Chen, Fuzzy Broad Learning System: A Novel Neuro-Fuzzy Model for Regression and Classification, IEEE Transactions on Cybernetics 50 (2) (2020) 414–424, https://doi.org/10.1109/TCYB.2018.2857815.
[15] S. Feng, C.L.P. Chen, L. Xu, Z. Liu, On the Accuracy-complexity Trade-off of Fuzzy Broad Learning System, IEEE Transactions on Fuzzy Systems 99 (2020) (2020) 1, https://doi.org/10.1109/TFUZZ.2020.3009757.
[16] X. Hu, X. Wei, Y. Gao, H. Liu, L. Zhu, Variational expectation maximization attention broad learning systems, Information Sciences 608 (2022) 597–612, https://doi.org/10.1016/j.ins.2022.06.074.
[17] W. Fan, Y. Si, W. Yang, M. Sun, Class-specific weighted broad learning system for imbalanced heartbeat classification, Information Sciences 610 (2022) 525–548, https://doi.org/10.1016/j.ins.2022.07.074.
[18] L. Liu, L. Cai, T. Liu, C.L. Philip Chen, X. Tang, Cauchy regularized broad learning system for noisy data regression, Information Sciences 603 (2022) 210–221, https://doi.org/10.1016/j.ins.2022.04.051.
[19] J. Jin, Y. Li, T. Yang, L. Zhao, J. Duan, C.L. Philip Chen, Discriminative group-sparsity constrained broad learning system for visual recognition, Information Sciences 576 (2021) 800–818, https://doi.org/10.1016/j.ins.2021.06.008.
[20] S. Kaczmarz, Angenäherte Auflsung von Systemen Linearer Gleichungen, Bulletin International de l'Acadmie Polonaise des Sciences et des Lettres 35 (1937) 355–357.
[21] R. Gordon, R. Bender, G.T. Herman, Algebraic Reconstruction Techniques (ART) for three-dimensional electron microscopy and X-ray photography, Journal of Theoretical Biology 29 (3) (1970) 471–481, https://doi.org/10.1016/0022-5193(70)90109-8.
[22] G.T. Herman, Fundamentals of Computerized Tomography: Image Reconstruction from Projections, Springer-Verlag, Dordrecht, The Netherlands, 2009.
[23] G.T. Herman, L.B. Meyer, Algebraic reconstruction techniques can be made computationally efficient (positron emission tomography application), IEEE Transactions on Medical Imaging 12 (3) (1993) 600–609, https://doi.org/10.1109/42.241889.
[24] H. Feichtinger, C. Cenker, M. Mayer, H. Steier, T. Strohmer, New variants of the POCS method using affine subspaces of finite codimension with applications to irregular sampling, Proc. SPIE: Visual Communications and Image Processing 11/01 1992 (1992) (1818) 299–310, https://doi.org/10.1117/12.131447.
[25] T. Strohmer, R. Vershynin, A Randomized Kaczmarz Algorithm with Exponential Convergence, Journal of Fourier Analysis and Applications 15 (2) (2008) 262, https://doi.org/10.1007/s00041-008-9030-4.
[26] A. Zouzias, N. Freris, Randomized Extended Kaczmarz For Solving Least Squares, SIAM Journal on Matrix Analysis and Applications 34 (2) (2012) 773–793, https://doi.org/10.1137/120889897.
[27] C.T. Leonides, Control and Dynamic Systems V18: Advances in Theory and Applications (Control and dynamic systems), Elsevier, Amsterdam, The Netherlands, 2012.
[28] A. Ben-Israel, T. Greville, Generalized Inverses: Theory and Applications, Wiley, New York, NY, USA, 1974.
[29] M. Gan, H.T. Zhu, G.Y. Chen, C.L.P. Chen, Weighted Generalized Cross-Validation-Based Regularization for Broad Learning System, IEEE Transactions on Cybernetics 99 (2020) (2020) 1–9, https://doi.org/10.1109/TCYB.2020.3015749.

[30] J. Jin, Z. Liu, C.L.P. Chen, "Discriminative graph regularized broad learning system for image recognition," *Science China*, Information Sciences 61 (11) (2018), 112209, https://doi.org/10.1007/s11432-017-9421-3.
[31] J. Jin, C.L. Philip Chen, Regularized robust Broad Learning System for uncertain data modeling, Neurocomputing 322 (2018) 58–69, https://doi.org/10.1016/j.neucom.2018.09.028.
[32] A. Ma, D. Needell, A. Ramdas, Convergence Properties of the Randomized Extended Gauss-Seidel and Kaczmarz Methods, SIAM Journal on Matrix Analysis and Applications 36 (4) (2015) 1590–1604, https://doi.org/10.1137/15M1014425.
[33] D. Needell, Randomized Kaczmarz solver for noisy linear systems, BIT Numerical Mathematics 50 (2) (2010) 395–403, https://doi.org/10.1007/s10543-010-0265-5.
[34] B. Dumitrescu, On the relation between the randomized extended Kaczmarz algorithm and coordinate descent, BIT Numerical Mathematics 55 (4) (2015) 1005–1015, https://doi.org/10.1007/s10543-014-0526-9.
[35] X. Wang, B. Zhang, J. Wang, K. Zhang, Y. Jin, "A Cluster-Based Competitive Particle Swarm Optimizer with a Sparse Truncation Operator for Multi-Objective Optimization," *Swarm*, Evolutionary Computation 71 (2022), 101083, https://doi.org/10.1016/j.swevo.2022.101083.
[36] X. Wang, K. Zhang, J. Wang, Y. Jin, An Enhanced Competitive Swarm Optimizer With Strongly Convex Sparse Operator for Large-Scale Multiobjective Optimization, IEEE Transactions on Evolutionary Computation 26 (5) (2022) 859–871, https://doi.org/10.1109/TEVC.2021.3111209.
[37] P. K. V. Storn R M. "Differential evolution-a simple and efficient adaptive scheme for global optimaization over continuous spaces," International Computer Science Institute, Tecnical Report,Berkley,USA, TR95-012. 1995.
[38] S.R.M.K.V. Price, J.A. Lampinen, Differerntial evolution: a practical approach to global optimazation (Natural Computing Series), (Springer-Verlag), New York, USA, 2005.
[39] A. Asuncion and D. Newman. "UC Irvine Machine Learning Repository." University of California, Irvine, School of Information and Computer Sciences. http://archive.ics.uci.edu/ml/index.php (accessed 2021).
[40] J. Alcala-Fdez, A. Fernández, J. Luengo, J. Derrac, and S. García, "KEEL Data-Mining Software Tool: Data Set Repository, Integration of Algorithms and Experimental Analysis Framework," *Multiple-Valued Logic and Soft Computing*, vol. 17, pp. 255-287, 01/01 2011.
[41] C. L. P. CHEN, Z. Liu, S. Feng, and J. Jin. "Broad Learning System." https://broadlearning.ai/download_code/(accessed 2021).
[42] T. Takagi, M. Sugeno, Fuzzy identification of systems and its applications to modeling and control, IEEE Transactions on Systems, Man, and Cybernetics vol. SMC-15, no. 1 (1985) 116–132, https://doi.org/10.1109/TSMC.1985.6313399.
[43] P. Angelov and R. Yager, "Simplified fuzzy rule-based systems using non-parametric antecedents and relative data density," in *2011 IEEE Workshop on Evolving and Adaptive Intelligent Systems (EAIS)*, 11-15 April 2011 2011, pp. 62-69, 10.1109/EAIS.2011.5945926.
[44] J. Wang, Q. Chang, T. Gao, K. Zhang, N.R. Pal, Sensitivity analysis of Takagi-Sugeno fuzzy neural network, Information Sciences 582 (2022) 725–749, https://doi.org/10.1016/j.ins.2021.10.037.
[45] G. Xue, Q. Chang, J. Wang, K. Zhang, N.R. Pal, An Adaptive Neuro-Fuzzy System With Integrated Feature Selection and Rule Extraction for High-Dimensional Classification Problems, IEEE Transactions on Fuzzy Systems 31 (7) (2023) 2167–2181, https://doi.org/10.1109/TFUZZ.2022.3220950.
[46] G. Xue, J. Wang, B. Yuan, C. Dai, "DG-ALETSK: A High-Dimensional Fuzzy Approach With Simultaneous Feature Selection and Rule Extraction", IEEE Transactions on Fuzzy Systems (2023) 1–15, https://doi.org/10.1109/TFUZZ.2023.3270445.
[47] G. Xue, J. Wang, B. Zhang, B. Yuan, C. Dai, Double groups of gates based Takagi-Sugeno-Kang (DG-TSK) fuzzy system for simultaneous feature selection and rule extraction, Fuzzy Sets and Systems 469 (2023), 108627, https://doi.org/10.1016/j.fss.2023.108627.