Downsizing and enhancing broad learning systems by feature augmentation and residuals boosting

Runshan Xie^{1,2} · Shitong Wang^{1,2}

Received: 20 December 2019 / Accepted: 23 March 2020 / Published online: 9 April 2020
© The Author(s) 2020

Abstract

Recently, a broad learning system (BLS) has been theoretically and experimentally confirmed to be an efficient incremental learning system. To get rid of deep architecture, BLS shares the same architecture and learning mechanism of the well-known functional link neural networks (FLNN), but works in broad learning way on both the randomly mapped features of original features of data and their randomly generated enhancement nodes. As such, BLS often requires a huge heap of hidden nodes to achieve the prescribed or satisfactory performance, which may inevitably cause both overwhelming storage requirement and overfitting phenomenon. In this study, a stacked architecture of broad learning systems called D&BLS is proposed to achieve enhanced performance and simultaneously downsize the system architecture. By boosting the residuals between previous and current layers and simultaneously augmenting the original input space with the outputs of the previous layer as the inputs of current layer, D&BLS stacks several lightweight BLS sub-systems to guarantee stronger feature representation capability and better classification/regression performance. Three fast incremental learning algorithms of D&BLS are also developed, without the need for the whole re-training. Experimental results on some popular datasets demonstrate the effectiveness of D&BLS in the sense of both enhanced performance and reduced system architecture.

Keywords Broad learning system (BLS) · Stacked structure · Boosting · Functional link neural network (FLNN) · Incremental learning

Introduction

As everyone may know well, over the past decade, deep learning systems have earned great success in many application fields [1, 2] and have been drawing more and more attentions in today's academic and industrial communities. Typical deep learning systems include deep restricted Boltzmann machines (RBM) [3], deep belief networks (DBN) [4, 5] and deep convolutional neural networks (CNN) [6]. Deep learning systems exhibit their overwhelming performance by taking their complicated deep structures and deep learning methods. However, they often suffer from very

time-consuming training because a huge number of hyper-parameters in the corresponding complicated deep structures are involved. Besides, updating their deep structure becomes an extraordinarily tedious task, due to the need for the whole re-training. To get rid of complicated deep architecture and effectively avoid very time-consuming training (and even re-training), as an alternative to deep learning systems, broad learning system (BLS in brevity) [7] has been recently invented to organize the well-known functional link neural networks (FLNN) [8, 9] in broad learning way. Up to date, several variants of BLS have been developed. A typical example is fuzzy BLS [10] which is based on Takagi–Sugeno–Kang fuzzy systems [11–13]. Theoretical and experimental evidences have revealed BLS's excellent classification/regression performance, with fast wide learning capability.

Since FLNN has both its simple structure of single-layer feedforward neural networks and its fast learning which can be realized quickly using its analytical solution, BLS takes FLNN as its basic structure and then expands it in broad learning way. In contrast to FLNN, BLS has two distinct-

✉ Runshan Xie
runshan_xie@foxmail.com

Shitong Wang
wxwangst@aliyun.com

¹ School of Digital Media, Jiangnan University, Wuxi, Jiangsu, China

² Jiangsu Key Laboratory of Digital Design and Software Technology, Wuxi 214122, China

tive discrepancies in the structure sense: (1) BLS transforms the set of the original input features of data into randomly mapped features and then take sparse autoencoders (SAE) to optimize them into sparse mapped features [14] so as to capture the intrinsic correlations of the input features. (2) BLS generates the enhancement nodes using nonlinear activation functions with randomly generated weights on all the sparse mapped features. All the sparse mapped features and the enhancement nodes are linked to the output layer, where the weights between the hidden nodes and the output layer are determined analytically by ridge regression [15], without the need for iterative updates of the weights. More importantly, without the need for re-training the whole network, BLS can train its structure in broad learning way for the incremental increase of randomly mapped features, enhancement nodes and input data, thereby avoiding very time-consuming training especially for big data. BLS has been theoretically proved to be a universal approximator [16]. A large amount of experimental evidences has indicated that BLS indeed has at least comparably and even much more satisfactory classification/regression performance and strong generalization capability, in contrast to the corresponding deep learning system. However, because the mapped features and enhancement nodes are randomly generated, BLS often needs a huge heap of enhancement nodes to achieve the prescribed performance, which may inevitably cause both overwhelming storage requirement and overfitting phenomenon, therefore, how to downsize a BLS system and simultaneously keep the strong capability of the whole system is becoming an urgent demand.

In this study, we attempt to tackle with this issue by stacking several lightweight BLS sub-systems with both feature augmentation and residuals boosting into its stacked structure called D&BLS. The basic idea can be clearly stated as follows.

1. To avoid overfitting phenomenon of the general BLS and simultaneously downsize its structure, D&BLS is proposed to stack several lightweight BLS sub-systems. As a deep classifier, the whole stacked structure and learning of D&BLS has its novelty in the sense of two joint stacking ways: feature augmentation and residuals boosting, which indeed guarantees enhanced performance of D&BLS.
2. Using the bootstrap strategy on feature nodes of each lightweight BLS sub-system, enhancement nodes are generated and hence each lightweight BLS sub-system is built to maintain powerful capability in both fast training and good generalization.
3. When D&BLS has its fixed depth, three incremental learning algorithms are designed for three incremental cases to endow D&BLS with the ability of fast learn-

ing in broad expansion of each layer without the whole re-training process.

4. Experimental results on six classification datasets demonstrate that D&BLS can reduce the number of enhancement nodes of BLS to about 76% or less with the same or better performance. On ten regression datasets, D&BLS can achieve smaller error with comparable number of enhancement nodes. On the MNIST dataset, three incremental algorithms of D&BLS can reach the performance comparable to that of D&BLS with one-shot construction and the training time of three incremental algorithms is much less than that of D&BLS with the one-shot construction. In addition, D&BLS only needs much less running time than BLS when they both share the same sizes. In general, D&BLS with some (i.e., from 2 to 4) lightweight sub-systems can obtain promising learning ability.

The rest of this paper is organized as follows. Section “On BLS” reviews BLS in brevity. Section “On D&BLS” proposes the structure of D&BLS, gives the learning algorithm of D&BLS, and discusses its computational complexity. Besides, three incremental algorithms of D&BLS are also designed in this section. Experimental results about image/non-image dataset classification, regression and incremental learning are presented in section “Experimental results”. And the last section concludes this paper.

On BLS

In this section, we give a brief review about BLS’s structure and three incremental learning algorithms.

Framework of BLS

In [7], BLS is conceptually composed of four basic parts: (1) original input; (2) feature nodes; (3) enhancement nodes; and (4) output layer. BLS begins with the original input as the input to certain feature mapping algorithms [7], which then generates the feature nodes by taking the sparse autoencoder to slightly fine-tune the random features. As such, all the mapped feature nodes actually form an efficient representation way of the original data. Accordingly, BLS generates a series of enhancement nodes by consecutively acting certain activation function with random parameters on all the mapped feature nodes, then all mapped features generated an enhancement node. Finally, BLS links both feature nodes and enhancement nodes to the output layer with each output weight. As one of the most famous merits, all the output weights can be trained analytically by the well-known ridge regression method [15].

Fig. 1 An illustrative structure of BLS

Figure 1 gives an illustrative structure of BLS. According to Fig. 1, we can concretely state how to construct a BLS as follows.

Without loss of generality, here we only consider the multi-input and single-output case. Suppose the training dataset X contains the total T original input samples with D dimensions and its output vector Y corresponds the total T target outputs, i.e., $X \in \mathbb{R}^{T \times D}$, $Y \in \mathbb{R}^{T \times 1}$. We assume BLS has the total n groups of feature nodes in which the i th group of feature nodes takes the mapping function ϕ_i , $i = 1, 2, \dots, n$ and contains the total d_i feature nodes and the total m groups of enhancement nodes in which the j th group of enhancement nodes takes the activation function ξ_j , $j = 1, 2, \dots, m$ and contains the total q_j enhancement nodes. As such, the i th group of feature nodes may be expressed as:

$$M_i = \phi_i(XW_{ei} + \beta_{ei}), \quad i = 1, 2, \dots, n \quad (1)$$

where the weight matrix $W_{ei} \in \mathbb{R}^{D \times d_i}$ and the bias matrix $\beta_{ei} \in \mathbb{R}^{T \times d_i}$ are randomly generated according to a certain distribution, such as normal distribution. According to [7], sparse autoencoder is taken to slightly fine-tune the random feature matrix M_i into sparse and compact feature matrix. Using alternating direction method of multiplier (ADMM), the weight matrix tuned by sparse autoencoder can be solved by the following iterative steps:

$$\begin{cases} \hat{W}_{k+1} = (\hat{M}_i^T \hat{M}_i + rI)^{-1} (\hat{M}_i^T X + r(o_k - u_k)) \\ o_{k+1} = S_{\lambda/r}(\hat{W}_{k+1} + u_k) \\ u_{k+1} = u_k + (\hat{W}_{k+1} - o_{k+1}) \end{cases} \quad (2)$$

where $\hat{M}_i$ is the desired sparse feature matrix, $\hat{W}$ is the sparse autoencoder solution, $\hat{W}_0$, o_0 and u_0 are zero matrices, λ is the given regularization parameter. In addition, $r > 0$ and S is the soft thresholding operator which is defined as:

$$S_t(a) = \begin{cases} a - t, & a > t \\ 0, & |a| \leq t \\ a + t, & a < -t \end{cases} \quad (3)$$

After a certain number of iterations, the sparse feature matrix can be obtained by:

$$M_i = X\hat{W} \quad (4)$$

We collect all the groups of feature nodes to form the group $M^n \triangleq [M_1, M_2, \dots, M_n]$, which will be used to generate the enhancement nodes. Similarly, the j th group of enhancement nodes may be expressed as:

$$E_j = \xi_j(M^n W_{hj} + \beta_{hj}), \quad j = 1, 2, \dots, m \quad (5)$$

where the weight matrix $W_{hj} \in \mathbb{R}^{(\sum_{i=1}^n d_i) \times q_j}$ and the bias matrix $\beta_{hj} \in \mathbb{R}^{T \times q_j}$ are also randomly generated according to a certain distribution. We collect all the groups of enhancement nodes E_j to form the group $E^m \triangleq [E_1, E_2, \dots, E_m]$. In this way, the output vector of such a BLS becomes:

$$\begin{aligned} Y &= [M_1, M_2, \dots, M_n | E_1, E_2, \dots, E_m] W_n^m \\ &= [M^n | E^m] W_n^m \end{aligned} \quad (6)$$

where $W_n^m \in \mathbb{R}^{(\sum_{i=1}^n d_i + \sum_{j=1}^m q_j) \times 1}$ are the output weight vector that can be solved analytically by ridge regression

method. That is to say, after denoting $U_n^m = [M^n | E^m]$, with the Moore–Penrose inverse of U_n^m , we have the analytical solution of W_n^m .

$$W_n^m = [U_n^m]^\dagger Y \quad (7)$$

Using ridge regression [15], the pseudoinverse of U_n^m can be solved as follows:

$$[U_n^m]^\dagger = \begin{cases} (\lambda I + (U_n^m)^T U_n^m)^{-1} (U_n^m)^T, & \text{if } T \geq h \\ (U_n^m)^T (\lambda I + U_n^m (U_n^m)^T)^{-1}, & \text{if } T < h \end{cases} \quad (8)$$

where λ is the given regularization parameter, $h = \sum_{i=1}^n d_i + \sum_{j=1}^m q_j$ is the total number of hidden nodes of BLS.

As a result, such a BLS is built quickly, without the need for iterative updates at a very slow learning speed, likewise in BP neural network learning [18].

Incremental learning of BLS

Except for fast training of BLS, another famous merit exists in its simple yet fast incremental learning methods, which indeed reveals its very applicability in online scenarios and/or practical situations where the more promising performance is expected. According to [7], without re-training the whole network structure, incremental learning of BLS can be realized from three cases.

Increment of enhancement nodes

In this case, BLS is expanded by adding the $(m+1)$ th group of enhancement nodes. That is to say, we have

$$E_{m+1} = \xi_{m+1} (M^n W_{h(m+1)} + \beta_{h(m+1)}) \quad (9)$$

$$U_n^{m+1} = [U_n^m | E_{m+1}] \quad (10)$$

After denoting $U^{ex} = E_{m+1}$, we can decompose the pseudoinverse of U_n^{m+1} as,

$$[U_n^{m+1}]^\dagger = \begin{bmatrix} [U_n^m]^\dagger - AB^T \\ B^T \end{bmatrix} \quad (11)$$

where

$$\begin{aligned} A &= [U_n^m]^\dagger U^{ex} \\ B^T &= \begin{cases} C^\dagger, & \text{if } C \neq 0, \\ (\mathbf{1} + A^T A)^{-1} A^T [U_n^m]^\dagger, & \text{if } C = 0. \end{cases} \\ C &= U^{ex} - U_n^m A \end{aligned} \quad (12)$$

Therefore, the new output weight vector W_n^{m+1} becomes

$$W_n^{m+1} = \begin{bmatrix} W_n^m - AB^T Y \\ B^T Y \end{bmatrix} \quad (13)$$

Since W_n^m , U_n^m and $[U_n^m]^\dagger$ have been obtained in advance, W_n^{m+1} can be quickly obtained according to Eqs. (12) and (13).

Increment of feature nodes

In this case, BLS is expanded by adding the $(n+1)$ th group of feature nodes. As such, we have,

$$M_{n+1} = \phi_{n+1} (X W_{e(n+1)} + \beta_{e(n+1)}) \quad (14)$$

$$U_{n+1}^m = [U_n^m | M_{n+1} | E_1^{ex} | E_2^{ex} | \dots | E_m^{ex}] \quad (15)$$

where $E_1^{ex} | E_2^{ex} | \dots | E_m^{ex}$ are the additional outputs of the enhancement nodes corresponding to the $(n+1)$ th group of feature nodes. After denoting $U^{ex} = [M_{n+1} | E_1^{ex} | E_2^{ex} | \dots | E_m^{ex}]$, we can decompose the pseudoinverse of $[U_{n+1}^m]$ as,

$$[U_{n+1}^m]^\dagger = \begin{bmatrix} [U_n^m]^\dagger - AB^T \\ B^T \end{bmatrix} \quad (16)$$

where A and B can be obtained according to Eq. (12). Therefore, the new output weight vector W_{n+1}^m is,

$$W_{n+1}^m = \begin{bmatrix} W_n^m - AB^T Y \\ B^T Y \end{bmatrix} \quad (17)$$

Obviously, since W_n^m , U_n^m and $[U_n^m]^\dagger$ have been obtained in advance, W_{n+1}^m can be quickly obtained according to Eqs. (12) and (17).

Increment of input data

In this case, BLS is fed with the additional training data $\{X^{ex}, Y^{ex}\}$. The additional feature node M^{ex} generated from X^{ex} is

$$M^{ex} = [\phi_1(X^{ex} W_{e1} + \beta_{e1}), \dots, \phi_n(X^{ex} W_{en} + \beta_{en})] \quad (18)$$

The output of enhancement nodes corresponding to M^{ex} is

$$E^{ex} = [\xi_1(M^{ex} W_{h1} + \beta_{h1}), \dots, \xi_m(M^{ex} W_{hm} + \beta_{hm})] \quad (19)$$

Please note, both W_{ei}, β_{ei} ($i = 1, \dots, n$) and W_{hj}, β_{hj} ($j = 1, \dots, m$) are initialized by taking their corresponding values in Eqs. (1) and (5). Let

$$(U_n^m)^{ex} = \begin{bmatrix} U_n^m \\ U^{ex} \end{bmatrix} \quad (20)$$

and $U^{ex} = [M^{ex}|E^{ex}]$, the pseudoinverse of $(U_n^m)^{ex}$ can be decomposed as,

$$((U_n^m)^{ex})^\dagger = \left[(U_n^m)^\dagger - B A^T | B \right] \quad (21)$$

where

$$\begin{aligned} A &= \left((U_n^m)^T \right)^\dagger (U^{ex})^T \\ B^T &= \begin{cases} C^\dagger, & \text{if } C \neq 0, \\ \left((1 + A^T A)^{-1} A^T \left((U_n^m)^T \right)^\dagger \right), & \text{if } C = 0. \end{cases} \\ C &= (U^{ex})^T - (U_n^m)^T A \end{aligned} \quad (22)$$

Therefore, the new output weight vector $(W_n^m)^{ex}$ is

$$(W_n^m)^{ex} = W_n^m + B(Y^{ex} - U^{ex} W_n^m) \quad (23)$$

Obviously, since W_n^m, U_n^m and $[U_n^m]^\dagger$ have been obtained in advance, $(W_n^m)^{ex}$ can be quickly obtained according to Eqs. (22) and (23).

While BLS shares the fast training originating from the flexible yet random construction of both feature nodes and enhancement nodes, it indeed requires a large amount of feature nodes and especially enhancement nodes to achieve satisfactory performance, which may cause an overwhelming storage requirement and overfitting phenomenon. For example, for MNIST dataset, according to [7], the built BLS needs 11,000 enhancement nodes. Therefore, in the next section, we will deepen BLS by stacking its structures into the proposed classifier so as to downsize the structure of BLS and simultaneously share the promising advantages of BLS.

On D&BLS

In this section, by means of feature augmentation, we first state the deep structure of D&BLS and its learning algorithm and then derive three incremental learning algorithms of D&BLS for three incremental cases. Their computational complexities are also discussed.

Structure and learning of D&BLS

Given D&BLS with L layers, the original training dataset $X \in \mathbb{R}^{T \times D}$ and its target output vector $Y \in \mathbb{R}^{T \times 1}$. The

parameters of the i th lightweight sub-system include the total n_i groups of feature nodes in which each group contains the total d_i feature nodes, the number of the selected feature nodes p_i that are chosen to generate enhancement nodes, and the total m_i groups of enhancement nodes in which each group contains the total q_i enhancement nodes. For convenience, we use $(d_1 \times n_1, p_1, q_1 \times m_1) - \dots - (d_L \times n_L, p_L, q_L \times m_L)$ to represent the parameters setting of this D&BLS and $\left(\sum_{i=1}^L \sum d_i \times n_i, \sum_{i=1}^L q_i \times m_i \right)$ to denote its whole structure.

Figure 2 illustrates its deep structure. According to Fig. 2, D&BLS essentially stacks several lightweight BLS sub-systems into its deep structure simultaneously by two ways: (1) augmenting the original input space with the outputs of previous layer as the inputs of current layer; (2) boosting the residual outputs between the original outputs and the outputs of all the previous layers as the target outputs of current layer. According to stacked generalization principle [17], the use of feature augmentation guarantees enhanced performance of D&BLS, whereas the use of residual outputs maintains good local approximation of current BLS sub-system for subtle output differences remaining in current layer. In general, a lightweight BLS sub-system takes not so many feature nodes and enhancement nodes, in contrast to the BLS system for the same training dataset. To stack lightweight BLS sub-systems well, the deep structure of D&BLS embodies our three reliable considerations as follows.

(1) According to the adopted feature augmentation, the i th lightweight sub-system has its augmented training dataset X_i , i.e.,

$$X_i = \begin{cases} X, & i = 1 \\ [X|Y'_{i-1}], & i = 2, 3, \dots, L \end{cases} \quad (24)$$

where Y'_{i-1} corresponds to the augmented feature and denotes the output vector of the $(i-1)$ th sub-system. In other words, the augmented training dataset X_i contains the output information of previous sub-system. Please note, each system always has only one augmented feature whose values are the outputs of previous layer in D&BLS.

(2) Inspired by the idea of residuals boosting [19–21], each target output of sub-system in current layer is taken to be the residual value between the corresponding target output and the corresponding outputs of the sub-systems of all the previous layers. As such, the i th sub-system has its target output vector Y_i :

$$Y_i = \begin{cases} Y, & i = 1 \\ Y_{i-1} - Y'_{i-1}, & i = 2, 3, \dots, L \end{cases} \quad (25)$$

Obviously, according to stacked generalization principle, the above stacking way of both feature augmentation and

Fig. 2 Structure of D&BLS

residuals boosting can surely enhance the performance of D&BLS with the increase of the number of sub-systems. In other words, with the prescribed performance, a downsized structure can be expected in the sense of the total number of hidden nodes, in contrast to the general BLS.

(3) When combining all the lightweight sub-systems together, diversity between them should be emphasized to achieve an effective ensemble. To do so, D&BLS only randomly bootstraps [22] some feature nodes to generate an enhancement node. Suppose the i th sub-system selects p_i feature nodes to generate an enhancement node. After denoting $M^{P_i} \triangleq [M_{c_1}, M_{c_2}, \dots, M_{c_{P_i}}]$ as the random selected feature nodes, where $c_1, c_2, \dots, c_{P_i} \in [1, n_i]$ denote random integers, then an enhancement node E_j can be generated as,

$$E_j = \xi_j(M^{P_i} W_{hj} + \beta_{hj}), \quad j = 1, 2, \dots, m \quad (26)$$

Once feature nodes and enhancement nodes of each lightweight BLS system are fixed according to the above strategy, this sub-system can be quickly constructed by the same learning algorithm of BLS as in the last section.

D&BLS begins with the construction of the first lightweight BLS sub-system on the original training dataset X and its target output vector Y . And then D&BLS obtains the output vector Y'_1 of the first lightweight BLS sub-system. After generating $X_2 = [X | Y'_1]$ and $Y_2 = Y - Y'_1$, respectively, as the inputs and target outputs of the second lightweight BLS sub-system, the second sub-system is built in the same way as in the above. This procedure is repeated

until the maximum number of sub-systems (i.e., layers) or the prescribed performance of D&BLS is achieved. As a result, the final output vector of D&BLS on the original training dataset can be expressed as,

$$Y' = \sum_{i=1}^L Y'_i \quad (27)$$

In summary, the learning algorithm is given in Algorithm 1. To visualize the effectiveness of Algorithm 1, a simple two-class classification dataset contained 200 training samples (100 positive samples) is taken for didactic experimentation. The dataset is generated from the model make_moons of Scikit-learn [29] package with 0 noise, as shown in Fig. 3a. Four D&BLSs with different number of sub-systems are taken and the deeper one is deepened from the shallower one. The deepest one is a four-layer D&BLS: $(6 \times 6, 9, 1 \times 3) - (5 \times 6, 9, 1 \times 2) - (5 \times 6, 9, 1 \times 2) - (5 \times 6, 9, 1 \times 2)$, its whole structure becomes $(126, 9)$. Here the average results of the ten runs are reported. Figure 3b–e shows the decision boundary of all D&BLSs.

From Fig. 3, the decision boundary of D&BLS becomes more complex to match the distribution of training samples as D&BLS becomes deeper. It indicates that Algorithm 1 is effective and the learning ability of D&BLS can be enhanced by adding more sub-systems with appropriate parameters.

The computational complexity of Algorithm 1 can be analyzed in terms of D&BLS with L sub-systems. Without loss of generality, according to the same strategy of BLS in [7],

Fig. 3 The moon dataset and corresponding decision boundary of four D&BLSs with different number of sub-systems

here we suppose ξ in step 12 is a sigmoid activation function [23] and the number of iterations of sparse autoencoder taken in step 6 is K .

Thus the computational complexity of step 2 and 3 can be calculated to be $O(2T) \approx O(T)$. The computational burden from step 4 to step 7 about the generation of feature nodes takes,

$$O(m_i(Tp_i + p_iq_i + Tq_i p_i + 3Tq_i)) \approx O(Tm_iq_i p_i)$$

Then steps 14 and 15 require $O(2Tm_iq_i) \approx O(Tm_iq_i)$. The calculation of the pseudoinverse of $(U_{n_i}^m)_i$ in step 16 requires

$$\begin{cases} O((n_i d_i + m_i q_i)^3 + 2T(n_i d_i + m_i q_i)^2 + 2(n_i d_i + m_i q_i)) \approx O(T(n_i d_i + m_i q_i)^2), & \text{if } (n_i d_i + m_i q_i) \leq T \\ O(T^3 + 2T^2(n_i d_i + m_i q_i) + 2T) \approx O(T^2(n_i d_i + m_i q_i)), & \text{if } (n_i d_i + m_i q_i) > T \end{cases}$$

$$\begin{aligned} & O\left(n_i \left(\frac{d_i(D+1) + 2Td_i(D+1) + 3Td_i + K(Td_i^2 + 2d_i + d_i^3 + Td_i(D+1) + 7d_i(D+1)d_i^2 + (D+1))}{K} \right)\right) \\ & \approx O(2TDn_i d_i + Kn_i(Td_i^2 + d_i^3 + TDd_i + Dd_i^2)) \end{aligned}$$

In general, $d_i < T$, $d_i < D$ and $2 < K$ can be assured, so the computational complexity mentioned above will be reduced to $O(KTDn_i d_i)$. Then step 8 requires $O(Tn_i d_i)$. The computational burden of the generation of enhancement nodes from step 9 to step 13 takes

while step 17 and step 18 require

$$O(2T(n_i d_i + m_i q_i)) \approx O(T(n_i d_i + m_i q_i))$$

Hence, with the total L sub-systems, the computational complexity of Algorithm 1 becomes

$$\begin{cases}
O\left((L-1)T + \sum_{i=1}^L \left(KT D n_i d_i + T n_i d_i + T m_i q_i p_i + T m_i q_i + \right. \right. \\
\quad \left. \left. T(n_i d_i + m_i q_i)^2 + T(n_i d_i + m_i q_i) \right) \right) \\
\approx O\left(\sum_{i=1}^L \left(KT D n_i d_i + T m_i q_i p_i + T(n_i d_i + m_i q_i)^2 \right) \right), \\
\text{if } (n_i d_i + m_i q_i) \leq T \\
O\left((L-1)T + \sum_{i=1}^L \left(KT D n_i d_i + T n_i d_i + T m_i q_i p_i + T m_i q_i + \right. \right. \\
\quad \left. \left. T^2(n_i d_i + m_i q_i) + T(n_i d_i + m_i q_i) \right) \right) \\
\approx O\left(\sum_{i=1}^L \left(KT D n_i d_i + T m_i q_i p_i + T^2(n_i d_i + m_i q_i) \right) \right), \\
\text{if } (n_i d_i + m_i q_i) > T
\end{cases}$$

As we can see, the most time-consuming parts for each sub-system in D&BLS are the generation of feature nodes, the generation of enhancement nodes and the calculation of pseudoinverse. Similarly, we can calculate the computational complexity of BLS for comparison. Suppose BLS contains $d \times n$ feature nodes and $q \times m$ enhancement nodes, its computational complexity is

$$\begin{cases}
O(KT D n d + T m q n d + T(n d + m q)^2), & \text{if } (n d + m q) \leq T \\
O(KT D n d + T m q n d + T^2(n d + m q)), & \text{if } (n d + m q) > T
\end{cases}$$

For easy comparison, here we suppose each sub-system of D&BLS has the total $\lceil n/L \rceil \times d$ feature nodes and $\lceil m/L \rceil \times q$ enhancement nodes. As such, the computational complexity of Algorithm 1 becomes

$$\begin{cases}
O\left(KT D n d + \frac{T m q}{L} \sum_{i=1}^L p_i + \frac{T}{L} (n d + m q)^2\right), & \text{if } \frac{(n d + m q)}{L} \leq T, \\
O\left(KT D n d + \frac{T m q}{L} \sum_{i=1}^L p_i + T^2(n d + m q)\right), & \text{if } \frac{(n d + m q)}{L} > T
\end{cases}$$

In general, $\frac{1}{L} \sum_{i=1}^L p_i < n d$ and $(n d + m q) \leq T$ can be assured. Thus the computational complexity is obviously less than that of BLS.

Algorithm 1: learning of D&BLS

Input: training dataset $\mathbf{X} \in \mathbb{R}^{T \times D}$ and its target output vector $\mathbf{Y} \in \mathbb{R}^{T \times 1}$

Output: $\mathbf{Y}'_i, (\mathbf{W}_{n_i}^{m_i})_i, i=1, 2, \dots, L$ and $\mathbf{Y}'$

Procedure:

```

1 for  $i = 1; i \leq L$  do;
2   Calculate  $\mathbf{X}_i = \begin{cases} \mathbf{X}, & \text{if } i = 1 \\ [\mathbf{X} | \mathbf{Y}_{i-1}'], & \text{else;} \end{cases}$ ;
3   Calculate  $\mathbf{Y}_i = \begin{cases} \mathbf{Y}, & \text{if } i = 1 \\ \mathbf{Y}_{i-1} - \mathbf{Y}_{i-1}', & \text{else;} \end{cases}$ ;
4   for  $k = 1, k \leq n_i$  do
5     Randomly generate  $\mathbf{W}_{ek}, \beta_{ek}$ ;
6     Calculate  $\mathbf{M}_k = \phi_k(\mathbf{X}_i \mathbf{W}_{ek} + \beta_{ek})$  with the same sparse autoencoder
       strategy in [7];
7   end
8   Set  $\mathbf{M}^{n_i} = [\mathbf{M}_1, \mathbf{M}_2, \dots, \mathbf{M}_{n_i}]$ ;
9   for  $k = 1, k \leq m_i$  do
10    Randomly select  $p_i$  feature nodes and set  $\mathbf{M}^p$ ;
11    Randomly generate  $\mathbf{W}_{hk}, \beta_{hk}$ ;
12    Calculate  $\mathbf{E}_k = \xi_k(\mathbf{M}^p \mathbf{W}_{hk} + \beta_{hk})$ ;
13  end
14  Set  $\mathbf{E}^{m_i} = [\mathbf{E}_1, \mathbf{E}_2, \dots, \mathbf{E}_{m_i}]$ ;
15  Set  $(\mathbf{U}_{n_i}^{m_i})_i = [\mathbf{M}^{n_i} | \mathbf{E}^{m_i}]$ ;
16  Calculate  $[(\mathbf{U}_{n_i}^{m_i})_i]^\dagger$  by ridge regression;
17  Calculate  $(\mathbf{W}_{n_i}^{m_i})_i = [(\mathbf{U}_{n_i}^{m_i})_i]^\dagger \mathbf{Y}_i$ ;
18  Calculate  $\mathbf{Y}'_i = (\mathbf{U}_{n_i}^{m_i})_i (\mathbf{W}_{n_i}^{m_i})_i$ ;
19 Calculate  $\mathbf{Y}' = \sum_{i=1}^L \mathbf{Y}'_i$ ;
20 Output  $\mathbf{Y}'$  by step 19;
21 end

```

Incremental learning for D&BLS

To achieve enhanced performance, it seems that we may expand D&BLS by increasing a lightweight BLS sub-system successively. However, too many lightweight BLS sub-systems often cause overfitting phenomenon. Our experimental evidences demonstrate that the total number of layers (i.e., sub-systems) should generally take a small integer (i.e., from 2 to 4). Therefore, a feasible strategy is to develop its incremental algorithms with the fixed number of layers. What is more, we should still consider how to expand D&BLS for the case of incremental input data. Here we develop three incremental algorithms of D&BLS for three incremental cases, without the need of re-training the whole classifier.

Increment of enhancement nodes

In this case, each sub-system in D&BLS is expanded by adding $\bar{m}$ (i.e., a small integer) groups of enhancement nodes in which each group contains $\bar{q}$ enhancement nodes.

The additional enhancement nodes $E_1^{ex}, E_2^{ex}, \dots, E_{\bar{m}}^{ex}$ are generated from a portion of the original feature nodes. After denoting $U^{ex} = [E_1^{ex}, E_2^{ex}, \dots, E_{\bar{m}}^{ex}]$, the pseudoinverse of $U_{n_i}^{m_i+\bar{m}} = [U_{n_i}^{m_i} | U^{ex}]$ can be calculated by Eqs. (11) and (12). Then the output weight vector can be determined by $(W_{n_i}^{m_i+\bar{m}})_i = (U_{n_i}^{m_i+\bar{m}})^{\dagger} Y_i$, and hence the output of this sub-system after expansion becomes $Y'_i = U_{n_i}^{m_i+\bar{m}} (W_{n_i}^{m_i+\bar{m}})_i$. The incremental algorithm for this case is summarized in Algorithm 2.

Algorithm 2: Increment of enhancement nodes

Input: input data $\{X, Y\}$ and original D&BLS with L layers

Output: $Y'_i, (W_{n_i}^{m_i+\bar{m}})_{i=1,2,\dots,L}$ and Y'

Procedure:

```

1 for  $i = 1; i \leq L$  do
2   Calculate  $Y_i = \begin{cases} Y, & \text{if } i = 1 \\ Y_{i-1} - Y'_{i-1}, & \text{else} \end{cases}$ ;
3   for  $k = 1; k \leq \bar{m}$  do
4     Randomly select  $p_i$  feature nodes and set  $M^{p_i}$ ;
5     Randomly generate  $W_{hk} \cdot \beta_{hk}$ ;
6     Calculate  $E_k^{ex} = \xi_k (M^{p_i} W_{hk} + \beta_{hk})$ ;
7   end
8   Set  $U^{ex} = [E_1^{ex}, E_2^{ex}, \dots, E_{\bar{m}}^{ex}]$ ;
9   Set  $U_{n_i}^{m_i+\bar{m}} = [U_{n_i}^{m_i} | U^{ex}]$ ;
10  Calculate  $(U_{n_i}^{m_i+\bar{m}})^{\dagger}$  by Eq.(11) and Eq.(12);
11  Calculate  $(W_{n_i}^{m_i+\bar{m}})_i = (U_{n_i}^{m_i+\bar{m}})^{\dagger} Y_i$ ;
12  Calculate  $Y'_i = U_{n_i}^{m_i+\bar{m}} (W_{n_i}^{m_i+\bar{m}})_i$ ;
13 Calculate  $Y' = \sum_{i=1}^L Y'_i$ ;
14 Output  $Y'$  by step 13;
15 end

```

Below let us analyze the computational complexity of Algorithm 2 in terms of L sub-systems. Because the training of each sub-system in this incremental case only deals with both the generation of the incremental enhancement nodes and the calculation of the corresponding pseudoinverse, we mainly observe the corresponding steps in Algorithm 2.

That is to say, the computational complexity from step 3 to step 7 takes $O(T\bar{m}\bar{q}p_i)$, while steps 8 and 9 require $O(2T\bar{m}\bar{q}) \approx O(T\bar{m}\bar{q})$. What is more, because the number of the additional enhancement nodes is generally a small inte-

ger, i.e., $\bar{m}\bar{q} < (n_i d_i + m_i q_i)$ and $\bar{m}\bar{q} < T$, the calculation of the pseudoinverse $(U_{n_i}^{m_i+\bar{m}})^{\dagger}$ in step 10 requires

$$O(3T\bar{m}\bar{q}(n_i d_i + m_i q_i) + T(n_i d_i + m_i q_i) + T(\bar{m}\bar{q})^2 + 2T\bar{m}\bar{q}) \\ \approx O(T\bar{m}\bar{q}(n_i d_i + m_i q_i))$$

Obviously, step 11 and step 12 require $O(2T(n_i d_i + m_i q_i + \bar{m}\bar{q})) \approx O(T(n_i d_i + m_i q_i))$. Hence, with the total L iterations, the computational complexity of Algorithm 2 becomes

$$O\left(LT\bar{m}\bar{q} + \sum_{i=1}^L (T\bar{m}\bar{q}p_i + T\bar{m}\bar{q}(n_i d_i + m_i q_i) + T(n_i d_i + m_i q_i))\right) \\ \approx O\left(\sum_{i=1}^L (T\bar{m}\bar{q}(n_i d_i + m_i q_i))\right)$$

If we re-train the whole model based on Algorithm 1 and assume $(n_i d_i + m_i q_i) < T$ is assured, its computation complexity will become

$$O\left(\sum_{i=1}^L (KT D n_i d_i + T(m_i q_i + \bar{m}\bar{q})p_i + T(n_i d_i + m_i q_i + \bar{m}\bar{q})^2)\right)$$

Obviously, increment of enhancement nodes can save much training time.

Increment of feature nodes

In this case, after $\bar{n}$ (i.e., a small integer) groups of feature nodes with $\bar{d}$ feature nodes are added, each sub-system in D&BLS is further expanded by the corresponding additional $\bar{m}$ (i.e., a small integer) groups of enhancement nodes with $\bar{q}$ enhancement nodes.

After the total $\bar{n}$ groups of the additional feature nodes M^{ex} are generated from X_i , the total $\bar{m}$ groups of the additional enhancement nodes E^{ex} are accordingly generated from a portion of the additional feature nodes M^{ex} . After denoting $U^{ex} = [M^{ex} | E^{ex}]$, the pseudoinverse of $U_{n_i+\bar{n}}^{m_i} = [U_{n_i}^{m_i} | U^{ex}]$ can be calculated by Eqs. (12) and (16). As such, the output weight vector can be obtained by $(W_{n_i+\bar{n}}^{m_i})_i = (U_{n_i+\bar{n}}^{m_i})^{\dagger} Y_i$ and the output of this sub-system accordingly

becomes $Y'_i = U_{n_i+\bar{n}}^{m_i} (W_{n_i+\bar{n}}^{m_i})_i$. The incremental algorithm for this case is summarized in following Algorithm 3.

The computational complexity of Algorithm 3 can be analyzed in terms of L sub-systems. Obviously, the training of each sub-system in this incremental case mainly has three stages, namely, the generation of the additional feature nodes and the corresponding additional enhancement nodes and the calculation of the pseudoinverse. According to Algorithm 3, the computational burden from step 4 to step 7 about the generation of feature nodes takes $O(KTD\bar{n}\bar{d})$. Step 8 requires

$O(T\bar{n}\bar{d})$. The computational complexity from step 9 to step 13 about the generation of enhancement nodes is $O(T\bar{m}\bar{q}p_i)$. Then steps 14 and 15 require $O(2T\bar{m}\bar{q}) \approx O(T\bar{m}\bar{q})$, step 16 requires $O(T(\bar{n}\bar{d} + \bar{m}\bar{q}))$. Because the number of both the additional feature nodes and the additional enhancement nodes is generally a small integer, i.e., $(\bar{n}\bar{d} + \bar{m}\bar{q}) < (n_i d_i + m_i q_i)$ and $(n_i d_i + m_i q_i) < T$, and the calculation of the pseudoinverse $(U_{n_i+\bar{n}}^{m_i})^+$ in step 17 requires

$$\begin{aligned} & O(3T(\bar{n}\bar{d} + \bar{m}\bar{q})(n_i d_i + m_i q_i) + T(n_i d_i + m_i q_i) \\ & \quad + T(\bar{n}\bar{d} + \bar{m}\bar{q})^2 + 2T(\bar{n}\bar{d} + \bar{m}\bar{q})) \\ & \approx O(T(\bar{n}\bar{d} + \bar{m}\bar{q})(n_i d_i + m_i q_i)). \end{aligned}$$

What is more, step 18 and step 19 require $O(2T(n_i d_i + m_i q_i + \bar{n}\bar{d} + \bar{m}\bar{q})) \approx O(T(n_i d_i + m_i q_i))$. In summary, with the total L iterations, the computational complexity of Algorithm 3 becomes

$$\begin{aligned} & O\left(LKT\bar{D}\bar{n}\bar{d} + 2LT\bar{m}\bar{q} + 2LT\bar{n}\bar{d} \right. \\ & \quad \left. + \sum_{i=1}^L (T\bar{m}\bar{q}p_i + T(\bar{n}\bar{d} + \bar{m}\bar{q})(n_i d_i + m_i q_i) \right. \\ & \quad \left. + T(n_i d_i + m_i q_i))\right) \\ & \approx O\left(\sum_{i=1}^L (T(\bar{n}\bar{d} + \bar{m}\bar{q})(n_i d_i + m_i q_i))\right) \end{aligned}$$

If we re-train the whole structure based on Algorithm 1 and suppose $(n_i d_i + m_i q_i) < T$ is assured, its computation complexity will become

$$\begin{aligned} & O\left(\sum_{i=1}^L (KT\bar{D}(n_i d_i + \bar{n}\bar{d}) + T(m_i q_i + \bar{m}\bar{q})p_i \right. \\ & \quad \left. + T(n_i d_i + m_i q_i + \bar{n}\bar{d} + \bar{m}\bar{q})^2)\right) \end{aligned}$$

Obviously, increment of feature nodes can save much training time.

Algorithm 3: Increment of feature nodes

Input: input data $\{X, Y\}$ and original D&BLS with L layers

Output: $Y'_i, (W_{n_i+\bar{n}}^{m_i})_i, i=1,2,\dots,L$ and Y'

Procedure:

```

1 for  $i=1; i \leq L$  do
2   Calculate  $X_i = \begin{cases} X, & \text{if } i=1 \\ [X|Y'_{i-1}], & \text{else} \end{cases}$ ;
3   Calculate  $Y_i = \begin{cases} Y, & \text{if } i=1 \\ Y_{i-1} - Y'_{i-1}, & \text{else} \end{cases}$ ;
4   for  $k=1; k \leq \bar{n}$  do
5     Random generate  $W_{ek}, \beta_{ek}$ ;
6     Calculate  $M_k^{ex} = \phi_{ek}(X_i W_{ek} + \beta_{ek})$  after using the same sparse
        autoencoder strategy in [7];
7   end
8   Set  $M^{ex} = [M_1^{ex}, M_2^{ex}, \dots, M_{\bar{n}}^{ex}]$ ;
9   for  $k=1; k \leq \bar{m}$  do
10    Randomly select  $p_i$  the additional feature nodes and set  $M^{\bar{n}}$ ;
11    Randomly generate  $W_{hk}, \beta_{hk}$ ;
12    Calculate  $E_k^{ex} = \xi_k(M^{\bar{n}} W_{hk} + \beta_{hk})$ ;
13  end
14  Set  $E^{ex} = [E_1^{ex}, E_2^{ex}, \dots, E_{\bar{m}}^{ex}]$ ;
15  Set  $U^{ex} = [M^{ex} | E^{ex}]$ ;
16  Set  $U_{n_i+\bar{n}}^{m_i} = [U_{n_i}^{m_i} | U^{ex}]$ ;
17  Calculate  $(U_{n_i+\bar{n}}^{m_i})^+$  by Eq.(16) and Eq.(12);
18  Calculate  $(W_{n_i+\bar{n}}^{m_i})_i = (U_{n_i+\bar{n}}^{m_i})^+ Y_i$ ;
19  Calculate  $Y'_i = U_{n_i+\bar{n}}^{m_i} (W_{n_i+\bar{n}}^{m_i})_i$ ;
20 Calculate  $Y' = \sum_{i=1}^L Y'_i$ ;
21 Output  $Y'$  by step 20;
22 end

```

Increment of input data

In this case, D&BLS is expanded by adding $\bar{T}$ input data $\{X_{\bar{T}}, Y_{\bar{T}}\}$, $X_{\bar{T}} \in \mathbb{R}^{\bar{T} \times D}$, $Y_{\bar{T}} \in \mathbb{R}^{\bar{T} \times 1}$. Suppose that $\{X_{T+\bar{T}}, Y_{T+\bar{T}}\}$ denotes the all input data after incremental operation. After the total n_i groups of the additional feature nodes M^{ex} with d_i feature nodes are generated from $X_{\bar{T}}$, the total m_i groups of the additional enhancement nodes E^{ex} with q_i enhancement nodes are generated from a portion of the additional feature nodes M^{ex} . After denoting $U^{ex} = [M^{ex} | E^{ex}]$, the pseudoinverse of $(U_{n_i}^{m_i})^{ex} = \begin{bmatrix} U_{n_i}^{m_i} \\ U^{ex} \end{bmatrix}$ can be calculated by Eqs. (21) and (22). As a result, the output weight vector can be obtained by $(W_{n_i}^{m_i})_i^{ex} = ((U_{n_i}^{m_i})^{ex})^+ Y_i$ and the output of this sub-system becomes $Y'_i = (U_{n_i}^{m_i})^{ex} (W_{n_i}^{m_i})_i^{ex}$.

The incremental algorithm for this case is summarized in Algorithm 4.

Algorithm 4: Increment of input data

Input: additional input data $\{X_T, Y_T\}$ and original D&BLS with L layers

Output: $Y'_i, (W_{n_i}^{m_i})_i^{\alpha}, i=1,2,\dots,L$ and Y'

Procedure:

```

1 for  $i = 1; i \leq L$  do
2   Calculate  $X_i = X_T$ ;
3   Calculate  $Y_i = \begin{cases} Y_T, & \text{if } i=1 \\ Y_{i-1} - Y'_{i-1}, & \text{else} \end{cases}$ ;
4   for  $k=1; k \leq n_i$  do
5     Randomly generate  $W_{ik}, \beta_{ik}$ ;
6     Calculate  $M_k^{\alpha} = \phi_k(X_i W_{ik} + \beta_{ik})$  after using the same sparse
       autoencoder strategy in [7];
7   end
8   Set  $M^{\alpha} = [M_1^{\alpha}, M_2^{\alpha}, \dots, M_{n_i}^{\alpha}]$ ;
9   for  $k=1; k \leq m_i$  do
10    Randomly select  $p_i$  the additional feature nodes and set  $M^{p_i}$ ;
11    Random generate  $W_{hk}, \beta_{hk}$ ;
12    Calculate  $E_k^{\alpha} = \zeta_k(M^{p_i} W_{hk} + \beta_{hk})$ ;
13  end
14  Set  $E^{\alpha} = [E_1^{\alpha}, E_2^{\alpha}, \dots, E_{m_i}^{\alpha}]$ ;
15  Set  $U^{\alpha} = [M^{\alpha} | E^{\alpha}]$ ;
16  Set  $(U_{n_i}^{m_i})^{\alpha} = \begin{bmatrix} U^{m_i} \\ U^{\alpha} \end{bmatrix}$ ;
17  Calculate  $((U_{n_i}^{m_i})^{\alpha})^{\dagger}$  by Eq.(21) and Eq.(22);
18  Calculate  $(W_{n_i}^{m_i})_i^{\alpha} = ((U_{n_i}^{m_i})^{\alpha})^{\dagger} Y_i$ ;
19  Calculate  $Y'_i = (U_{n_i}^{m_i})^{\alpha} (W_{n_i}^{m_i})_i^{\alpha}$ ;
20 Calculate  $Y' = \sum_{i=1}^L Y'_i$ ;
21 Output  $Y'$  by step 20;
22 end

```

The computational complexity of Algorithm 4 can also be analyzed in terms of L sub-systems. The computational complexity of each sub-system in this incremental case mainly contains three stages: the generation of the additional feature nodes and the additional enhancement nodes and the calculation of pseudoinverse. According to Algorithm 4, the computational complexity from step 4 to step 7 about the generation of feature nodes takes $O(K\bar{T}Dn_id_i)$. Step 8 requires $O(\bar{T}n_id_i)$. Then the computational burden from step 9 to step 13 about the generation of enhancement nodes is $O(\bar{T}m_iq_i p_i)$. Step 14 and 15 require $O(2\bar{T}m_iq_i) \approx O(\bar{T}m_iq_i)$, then step 16 requires $O(\bar{T}(n_id_i + m_iq_i))$. Suppose

$(n_id_i + m_iq_i) < \bar{T}$ and $(n_id_i + m_iq_i) < T$ are assured, the calculation of $((U_{n_i}^{m_i})^{\alpha})^{\dagger}$ in step 17 requires

$$\begin{aligned}
 &O(\bar{T}(n_id_i + m_iq_i)^2 + 2\bar{T}(n_id_i + m_iq_i) \\
 &\quad + T(n_id_i + m_iq_i) + 3T\bar{T}(n_id_i + m_iq_i)) \\
 &\approx O(T\bar{T}(n_id_i + m_iq_i))
 \end{aligned}$$

What is more, steps 18 and 19 require $O(2(T + \bar{T})(n_id_i + m_iq_i)) \approx O((T + \bar{T})(n_id_i + m_iq_i))$. Suppose $\bar{T} < T$ and $KD < T$ are assured, with the total L iterations, the total computational complexity of Algorithm 4 can be described as

$$\begin{aligned}
 &O\left(\sum_{i=1}^L (K\bar{T}Dn_id_i + 2\bar{T}n_id_i + \bar{T}m_iq_i p_i + 2\bar{T}m_iq_i \right. \\
 &\quad \left. + T\bar{T}(n_id_i + m_iq_i) + (T + \bar{T})(n_id_i + m_iq_i))\right) \\
 &\approx O\left(\sum_{i=1}^L (T\bar{T}(n_id_i + m_iq_i))\right)
 \end{aligned}$$

If we re-train the whole structure based on Algorithm 1 and suppose $(n_id_i + m_iq_i) < T + \bar{T}$ is assured, its computation complexity will become

$$O\left(\sum_{i=1}^L (K(T + \bar{T})Dn_id_i + (T + \bar{T})m_iq_i p_i + (T + \bar{T})(n_id_i + m_iq_i)^2)\right)$$

Obviously, increment of input data can save training time.

Finally, let us state how to set appropriate parameters in the above algorithms. As pointed out in the above, the total number of layers generally take a small integer (i.e., from 2 to 4). Besides, according to our extensive experiments, the weight matrix W and the bias matrix β involved in hidden nodes generally are drawn from standard normal distributions, respectively. Since the number p_i of selected feature nodes should generally be much less than the total of feature nodes, it is set to be at most 75% of the total number of feature nodes. As for the regularization parameters λ and r involved in sparse autoencoder, they are simply set to be 0.001 and 1, respectively. The regularization parameter λ involved in ridge regression is simply set to be 2^{-30} . In our experiments, the number of additional hidden nodes for each sub-system is simply taken to be 10 and the number of additional input data is simply set to be 3000.

Experimental results

In this section, to evaluate the performance of D&BLS, we organize four groups of experiments. The first group about classification is carried out on both five image datasets and 1 UCI [24] classification datasets, the second about regression on ten UCI regression datasets, the third about incremental learning on popular datasets MNIST [25], and the last

about running time is compared between BLS and D&BLS. All the experiments are carried out under a computer that equips Intel-i3 3.40 GHz CPU, 4 GB memory. The experiments of LSSVM are carried out in MATLAB environment and the other experiments are carried out in Python environment. Before the experiments, all the datasets are divided into training and testing set.

The comparative methods include BLS [7], support vector machine (SVM) [26], least squared SVM (LSSVM) [27]. Because the regularization parameter C and the kernel parameter γ of SVM play an essential role in performance, they need to be chosen appropriately for a fair comparison. In our experiments, we determine them by taking a grid search from $\{2^{-24}, 2^{-23}, \dots, 2^{24}, 2^{25}\}$ for the parameters (C, γ) in SVM, while the parameters (C, γ) of LSSVM are decided using *LS-SVMlab* Toolbox.

In our experiments, the original codes of BLS in [7] (classification version) and in [16] (regression version) are taken. As for SVM, its classification version SVC and its regression version SVR are taken from Scikit-learn [29] package.

As for BLS and D&BLS, to make a fair comparison between them, the weight matrix W and the bias matrix β are drawn from standard normal distributions and the *tansig* functions are taken as the activation functions of enhancement nodes. All other parameters involved in BLS and D&BLS keep the same and their settings have been stated in the last section.

Classification

This group of experiments consists of two parts. The first part is to observe the performance of D&BLS and the comparative methods on image data, while the second part deals with the 1 *UCI* non-image classification datasets. Five popular image datasets are *MNIST* [25] and *USPS* [29] about handwritten digits, *COIL20* [30] and *COIL100* [31] about 3-D objects, and *extended YaleB* [32] about human faces. A *UCI* classification dataset is *isolet* [33] about spoken letter recognition. To obtain good performance, we use “one vs. one” strategy for LSSVM when faced with multi-class problems and thus omit its parameters setting here.

According to the accuracies obtained by BLS on the adopted datasets, we determine D&BLS by the trial and error strategy. This involves the number of lightweight BLS subsystems in D&BLS. To make our comparison fair, here we report average results (i.e., mean and standard deviations) for ten trials on each dataset.

MNIST

MNIST contains 70,000 handwritten digits of ten classes. Every digit is represented by an image with the size of 28×28 gray-scaled pixels. 60,000 out of all the handwritten dig-

its are partitioned into the training dataset and the remaining 10,000 handwritten digits into the testing dataset. Here a subset of *MNIST* is chosen for our experiment with the first 10,000 training images and all the 10,000 testing images. In our experiments, we perform a grid search for the optimal parameters of BLS, including the size of feature node groups, the number of feature node groups and the number of enhancement nodes from $[10, 30] \times [10, 20] \times [100, 5000]$ with the steps being 1, 1 and 100 due to the large searching range of enhancement nodes. The parameters setting can be checked in Table 1 and the classification accuracies of all the adopted methods are given in Table 2.

Obviously, D&BLS reaches better accuracy (97.37%) than that (97.19%) of BLS, while D&BLS only needs 3490 enhancement nodes, which is about 72.71% number of enhancement nodes (i.e., 4800 nodes) in BLS. In addition, D&BLS exhibits its superiority of classification performance over other comparative methods: SVM (94.32%) and LSSVM (91.40%).

USPS

US Postal Service (*USPS*) handwritten digits recognition corpus is another well know dataset, which contains 9298 gray images of size 16×16 pixels, with ten class digit 0–9. We randomly select 7500 images for training and the rest 1798 images for testing. In our experiment, the search range for BLS is $[10, 30] \times [10, 20] \times [100, 5000]$ with the steps being 1, 1 and 100. The parameters setting is shown in Table 1 and the classification accuracies are listed in Table 2.

We can see that D&BLS exceeds the accuracy of BLS (i.e. 95.90% vs. 95.77%) with much less enhancement nodes (i.e., only 1877 nodes), which is about half the number of enhancement nodes (i.e., 3700 nodes) in BLS. Compared to SVM (93.44%) and LSSVM (95.27%), D&BLS also achieves the best performance.

COIL20

COIL20 contains 1440 gray images of 20 different 3-D objects with 128×128 pixels. The images of each object were taken 5 degrees apart as the object is rotated on a turntable and each object has 72 images. There are 36 images of each object for training and the remaining 720 images for testing. In our experiment, the search range for BLS is $[10, 30] \times [10, 20] \times [100, 3000]$ with the steps being 1, 1 and 100. The parameters setting are given in Table 1 and the classification accuracies can be checked in Table 2.

In contrast to BLS, D&BLS has higher accuracy (99.78%) with much less enhancement nodes (i.e., only 1357 nodes), which is about 61.68% of the number of enhancement nodes (i.e., 2200 nodes) in BLS. D&BLS also achieves best performance compared to SVM (83.19%) and LSSVM (89.86%).

Table 1 Parameters setting of the adopted methods on the classification datasets

Methods	<i>MNIST</i>	<i>USPS</i>	<i>COIL20</i>
(1)			
SVM			
$C \times \gamma$	$2^5 \times 2^{-23}$	$2^2 \times 2^{-6}$	$2^1 \times 2^{-24}$
BLS			
$(d \times n, q \times m)$	$(15 \times 13, 1 \times 4800)$	$(13 \times 10, 1 \times 3700)$	$(10 \times 17, 1 \times 2200)$
Total structure	(195, 4800)	(130, 3700)	(170, 2200)
D&BLS			
Parameters setting	$(11 \times 29, 40, 1 \times 1247)$ $(10 \times 29, 40, 1 \times 1184)$ $(9 \times 29, 40, 1 \times 1059)$	$(14 \times 21, 3, 1 \times 683)$ $(13 \times 21, 3, 1 \times 648)$ $(12 \times 21, 3, 1 \times 546)$	$(13 \times 20, 2, 1 \times 905)$ $(12 \times 20, 2, 1 \times 452)$
Total structure	(870, 3490)	(819, 1877)	(500, 1357)
Methods	<i>COIL100</i>	<i>YaleB</i>	<i>Isolet</i>
(2)			
SVM			
$C \times \gamma$	$2^6 \times 2^{-22}$	$2^8 \times 2^{-24}$	$2^5 \times 2^{-9}$
BLS			
$(d \times n, q \times m)$	$(14 \times 12, 1 \times 4400)$	$(60 \times 30, 1 \times 6000)$	$(15 \times 14, 1 \times 400)$
Total structure	(168, 4400)	(1800, 6000)	(210, 400)
D&BLS			
Parameters setting	$(18 \times 20, 41, 1 \times 986)$ $(17 \times 20, 41, 1 \times 936)$ $(16 \times 20, 41, 1 \times 788)$	$(17 \times 20, 19, 1 \times 119)$ $(16 \times 20, 19, 1 \times 113)$	$(11 \times 11, 8, 1 \times 156)$ $(10 \times 11, 8, 1 \times 148)$
Total structure	(1020, 2710)	(660, 232)	(231, 304)

COIL100

COIL100 contains 7200 color images of 100 different 3-D objects with 128×128 pixels. The images of each object were taken 5 degrees apart as the object is rotated on a turntable and each object has 72 images. In this experiment, each image was resized into 32×32 and converted into gray image. We randomly select 5000 images for training and the rest 2200 images for testing.

In our experiment, the search range for BLS is $[10, 30] \times [10, 20] \times [100, 5000]$ with the steps being 1, 1 and 100. The

Table 2 Accuracies (%) of the adopted methods on the classification datasets

Datasets	SVM	LSSVM		BLS		D&BLS	
	Training	Testing	Training	Testing	Training	Testing	Testing
<i>MNIST</i>	99.99 (± 0.0000)	94.32 (± 0.0000)	97.77 (± 0.0000)	91.40 (± 0.0000)	100.00 (± 0.0000)	99.64 (± 0.0277)	97.37 (± 0.0753)
<i>USPS</i>	99.79 (± 0.0000)	93.44 (± 0.0000)	99.79 (± 0.0000)	95.27 (± 0.0000)	99.71 (± 0.7089)	99.67 (± 0.0452)	95.90 (± 0.1609)
<i>COIL20</i>	100.00 (± 0.0000)	83.19 (± 0.0000)	100.00 (± 0.0000)	89.86 (± 0.0000)	100.00 (± 0.0000)	100.00 (± 0.0000)	99.78 (± 0.1274)
<i>COIL100</i>	100.00 (± 0.0000)	96.36 (± 0.0000)	99.96 (± 0.0000)	95.05 (± 0.0000)	100.00 (± 0.0000)	100.00 (± 0.0000)	99.02 (± 0.1305)
<i>YaleB</i>	98.60 (± 0.0000)	89.32 (± 0.0000)	98.51 (± 0.0000)	89.95 (± 0.0000)	100.00 (± 0.0000)	100.00 (± 0.0000)	97.92 (± 0.2466)
<i>Isolet</i>	100.00 (± 0.0000)	89.32 (± 0.0000)	99.91 (± 0.0000)	90.60 (± 0.0000)	100.00 (± 0.0000)	99.85 (± 0.0837)	95.49 (± 0.2938)

Bold values are the best one among all the results

parameters setting are listed in Table 1 and the classification accuracies are shown in Table 2.

As we can see, D&BLS exceeds the accuracies of all the adopted methods with much less enhancement nodes (i.e., only 2710 nodes), which is about 61.59% of the number of enhancement nodes in BLS.

Extended YaleB

Extended YaleB face consists of 2414 cropped images of 38 objects with the size of 32×32 pixels. The images contain large variations in terms of illumination conditions and expressions for each subject. There are 30 images of each person for training and the remaining 1274 images for testing. Following [16], BLS contains 60×30 feature nodes and 1×6000 enhancement nodes. The parameters setting of other models are shown in Table 1 and the classification accuracies of all the adopted methods are listed in Table 2.

From Tables 1 and 2, D&BLS matches the accuracy (97.92%) of BLS with only 232 enhancement nodes, which is about 3.87% of the number of enhancement nodes in BLS. D&BLS also exceeds the accuracy of SVM (89.32%) and LSSVM (89.95%).

UCI

To verify the effectiveness of D&BLS for non-image data, we choose the *isolet* dataset here, which contains 1560 spoken letter recognition samples of 26 classes with 617 features in each sample. We randomly choose 1092 samples for training and the rest 468 samples for testing. In our experiment, the search range for BLS is $[10, 30] \times [10, 20] \times [100, 3000]$ with the steps being 1, 1 and 100. The parameter details are shown in Table 1 and the classification performance are given in Table 2.

D&BLS indeed gets the best result (95.49%) among all the adopted methods and only needs about 76% of the number of enhancement nodes in BLS, which actually demonstrates the effectiveness of D&BLS for non-image samples.

Regression

In this subsection, we attempt to verify that D&BLS can achieve smaller regression errors than BLS when they both have comparable number of enhancement nodes. We take ten *UCI* regression datasets, whose details are listed in Table 3

Following [16], the same parameters setting of BLS are taken by a grid search from $[1, 10] \times [1, 30] \times [1, 200]$ with step being 1 and the best experimental results about the root-mean-square errors (RMSE) from ten trials are taken to make a fair comparison. The detailed parameter settings and the experimental results by SVM, LSSVM, BLS and D&BLS on these datasets are given in Tables 4 and 5, respectively.

Table 3 Details of the *UCI* regression datasets

Datasets	No. of samples		Features
	Training	Testing	
<i>Abalone</i>	2784	1393	8
<i>Basketball</i>	64	32	4
<i>Bodyfat</i>	168	84	14
<i>Cleveland</i>	202	101	13
<i>Housing</i>	337	169	13
<i>Mortgage</i>	699	350	15
<i>Pyrim</i>	49	25	27
<i>Quake</i>	1452	726	3
<i>Strike</i>	416	209	6
<i>Weather Izmir</i>	974	487	9

Clearly, D&BLS performs comparably to BLS on the datasets *Bodyfat* and *Weather Izmir* and better than BLS with comparable number of enhancement nodes in the remaining eight datasets. It can be concluded that with almost the same sizes of the architectures, D&BLS attractively outperforms BLS in the sense of testing accuracy on these datasets, which actually hints that D&BLS may take smaller sizes of the architectures to achieve comparable regression performance to BLS. Compared to SVM and LSSVM, the D&BLS also achieves the best performance.

Incremental algorithms

In this section, we divide it into three cases to observe the average performances (with their standard deviations) of incremental learning algorithms of D&BLS for ten runs on the dataset MNIST: (1) increment of enhancement nodes (corresponding to Algorithm 2); (2) feature nodes (corresponding to Algorithm 3); and (3) input data (corresponding to Algorithm 4).

Our experiments begin with a three-layer D&BLS, whose whole structure is (696,275) consisting of $(30 \times 8, 10, 1 \times 100)$ – $(29 \times 8, 10, 1 \times 95)$ – $(28 \times 8, 10, 1 \times 80)$ on the MNIST dataset where the first 10,000 samples from the total 60,000 training samples are taken as the initial training samples, and the same 10,000 testing samples are still kept. The parameter p_i of the additional enhancement nodes is also set to be 10. Besides, the testing accuracy, training time and testing time of D&BLS with one-shot construction are also provided to highlight the advantage of the proposed three incremental algorithms.

Increment of enhancement nodes

Two groups of experiments are arranged to observe Algorithm 2 for this case. The first group considers the total 4

Table 4 Parameters setting of the adopted methods (1) on the *UCI* regression datasets and (2) on the *UCI* datasets for regression

Methods	<i>Abalone</i>	<i>Basketball</i>	<i>Bodyfat</i>	<i>Cleveland</i>	<i>Housing</i>
(1)					
SVM					
$C \times \gamma$	$2^2 \times 2^{-1}$	$2^0 \times 2^0$	$2^2 \times 2^{-2}$	$2^2 \times 2^2$	$2^2 \times 2^1$
LSSVM					
C	2.8932	5.7360	3168.3251	0.6616	49.1373
γ	3.0774	26.4981	188.1635	32.3909	7.4926
BLS					
$(d \times n, q \times m)$	$(5 \times 6, 1 \times 41)$	$(6 \times 7, 1 \times 4)$	$(6 \times 5, 1 \times 13)$	$(1 \times 10, 1 \times 9)$	$(5 \times 29, 1 \times 50)$
Total structure	(30, 41)	(42, 4)	(30, 13)	(10, 9)	(145, 50)
D&BLS					
Parameters setting	$(9 \times 4, 3, 1 \times 21)$ $(8 \times 4, 3, 1 \times 19)$ $(7 \times 4, 3, 1 \times 17)$	$(9 \times 6, 6, 1 \times 2)$ $(8 \times 6, 6, 1 \times 1)$	$(6 \times 5, 7, 1 \times 17)$ $(5 \times 5, 7, 1 \times 16)$ $(4 \times 5, 7, 1 \times 14)$	$(8 \times 3, 1, 1 \times 20)$ $(7 \times 3, 1, 1 \times 19)$	$(10 \times 4, 2, 1 \times 39)$ $(9 \times 4, 2, 1 \times 37)$ $(8 \times 4, 2, 1 \times 33)$
Total structure	(96, 57)	(102, 3)	(75, 47)	(45, 39)	(108, 109)
Methods	<i>Mortgage</i>	<i>Pyrim</i>	<i>Quake</i>	<i>Strike</i>	<i>Weather Izmir</i>
(2)					
SVM					
$C \times \gamma$	$2^{10} \times 2^{-1}$	$2^{10} \times 2^8$	$2^5 \times 2^5$	$2^0 \times 2^{-4}$	$2^4 \times 2^{-2}$
LSSVM					
C	58,034.7023	59.6682	0.0990	0.4055	4928.7328
γ	3.1781	3.4092	0.0153	0.6905	68.4401
BLS					
$(d \times n, q \times m)$	$(9 \times 4, 1 \times 135)$	$(3 \times 7, 1 \times 2)$	$(10 \times 2, 1 \times 6)$	$(9 \times 11, 1 \times 30)$	$(4 \times 3, 1 \times 87)$
Total structure	(36, 135)	(21, 2)	(20, 6)	(99, 30)	(12, 87)
D&BLS					
Parameters setting	$(5 \times 10, 6, 1 \times 70)$ $(4 \times 10, 6, 1 \times 66)$ $(3 \times 10, 6, 1 \times 59)$	$(7 \times 9, 3, 1 \times 14)$ $(6 \times 9, 3, 1 \times 13)$	$(5 \times 4, 9, 1 \times 14)$ $(4 \times 4, 9, 1 \times 13)$ $(3 \times 4, 9, 1 \times 11)$	$(5 \times 10, 8, 1 \times 27)$ $(4 \times 10, 8, 1 \times 25)$	$(5 \times 6, 9, 1 \times 49)$ $(4 \times 6, 9, 1 \times 46)$ $(3 \times 6, 9, 1 \times 41)$
Total structure	(120, 195)	(117, 27)	(48, 38)	(90, 52)	(72, 136)

Table 5 RMSEs of the adopted methods on the *UCI* datasets for regression

Datasets	SVM		LSSVM		BLS		D&BLS	
	Training	Testing	Training	Testing	Training	Testing	Training	Testing
<i>Abalone</i>	0.0764	0.0786	0.0691	0.0756	0.0728	0.0757	0.0732	0.0755
<i>Basketball</i>	0.0805	0.0768	0.0880	0.0744	0.0786	0.0703	0.0805	0.0674
<i>Bodyfat</i>	0.0124	0.0105	0.0026	0.0038	0.0038	0.0031	0.0043	0.0031
<i>Cleveland</i>	0.0031	0.1517	0.1024	0.1257	0.1075	0.1234	0.0975	0.1228
<i>Housing</i>	0.0360	0.0826	0.0332	0.0783	0.0647	0.0816	0.0540	0.0699
<i>Mortgage</i>	0.0064	0.0078	0.0009	0.0054	0.0035	0.0050	0.0040	0.0049
<i>Pyrim</i>	0.0132	0.1549	0.0233	0.1129	0.0468	0.0703	0.0175	0.0576
<i>Quake</i>	0.1604	0.2030	0.1603	0.1737	0.1708	0.1725	0.1691	0.1716
<i>Strike</i>	0.0596	0.1059	0.0547	0.1007	0.0508	0.1007	0.0475	0.1006
<i>Weather Izmir</i>	0.0183	0.0209	0.0167	0.0191	0.0164	0.0189	0.0166	0.0189

Bold values are the best one among all the results

Table 6 Experimental results of the *MNIST* dataset using increment of enhancement nodes

Number of feature nodes	Number of enhancement nodes	Testing accuracy (%)	Each additional training time (s)	Accumulative training time (s)	Each additional testing time (s)	Accumulative testing time (s)
(1)						
696	275	94.12 (± 0.1088)	5.2998 (± 0.0971)	5.2998 (± 0.0971)	1.8600 (± 0.0285)	1.8600 (± 0.0285)
696	275 $\xrightarrow{30}$ 305	94.35 (± 0.1351)	0.4462 (± 0.0056)	0.4462 (± 0.0056)	0.1408 (± 0.0019)	0.1408 (± 0.0019)
696	305 $\xrightarrow{30}$ 335	94.48 (± 0.1520)	0.4499 (± 0.0030)	0.8961 (± 0.0060)	0.1443 (± 0.0025)	0.2851 (± 0.0039)
696	335 $\xrightarrow{30}$ 365	94.51 (± 0.0888)	0.4658 (± 0.0090)	1.3619 (± 0.0101)	0.1482 (± 0.0033)	0.4333 (± 0.0056)
696	365 $\xrightarrow{30}$ 395	94.60 (± 0.0869)	0.4780 (± 0.0066)	1.8399 (± 0.0141)	0.1491 (± 0.0017)	0.5824 (± 0.0057)
696	395	94.74 (± 0.1294)	5.5217 (± 0.1507)	5.5217 (± 0.1507)	1.9094 (± 0.0588)	1.9094 (± 0.0588)
(2)						
696	275	94.23 (± 0.1440)	5.3288 (± 0.0965)	5.3288 (± 0.0965)	1.8853 (± 0.0377)	1.8853 (± 0.0377)
696	275 $\xrightarrow{36}$ 311	94.37 (± 0.2104)	0.4554 (± 0.0074)	0.4554 (± 0.0074)	0.1439 (± 0.0024)	0.1439 (± 0.0024)
696	311 $\xrightarrow{36}$ 347	94.50 (± 0.1836)	0.4644 (± 0.0097)	0.9198 (± 0.0123)	0.1485 (± 0.0028)	0.2924 (± 0.0030)
696	347 $\xrightarrow{36}$ 383	94.64 (± 0.1607)	0.4767 (± 0.0077)	1.3965 (± 0.0164)	0.1502 (± 0.0026)	0.4426 (± 0.0052)
696	383 $\xrightarrow{36}$ 419	94.73 (± 0.2091)	0.4892 (± 0.0073)	1.8857 (± 0.0188)	0.1524 (± 0.0031)	0.5950 (± 0.0072)
696	419	94.97 (± 0.1299)	5.5546 (± 0.0914)	5.5546 (± 0.0914)	1.9281 (± 0.0485)	1.9281 (± 0.0485)

Bold values are the best one among all the results

times of incremental operations. That is, it takes the addition of ten enhancement nodes in an incremental way. The second group keeps the same except 12 rather 10 enhancement nodes. Thus, the whole network with three sub-systems has 30 or 36 more enhancement nodes after enhancement nodes are increased in a step-by-step way. The results are given in Table 6.

As we can see, the testing accuracy continues to be improved as the additional enhancement nodes are inserted into D&BLS in an incremental way and finally reaches the performance comparable to that of D&BLS with the one-shot construction (see the last row in Table 6). Obviously, the training time of Algorithm 2 is much less than that of D&BLS with the one-shot construction, e.g., 1.8399 s vs. 5.5217 s.

Increment of feature nodes

Two groups of experiments are arranged to observe Algorithm 3 for this case. The first group considers the total 4 times of incremental operations. That is, it takes the addition of ten feature nodes and ten corresponding enhancement nodes in an incremental way. The second group keeps the same except 12 rather 10 feature nodes and enhancement nodes. All the results can be checked in Table 7.

It can be seen that the testing accuracy continues to go up as the additional feature nodes are inserted into D&BLS

in an incremental way and finally reaches the performance comparable to that of D&BLS with the one-shot construction (see the last row in Table 7). Obviously, the training time of Algorithm 3 is much less than that of D&BLS with the one-shot construction, e.g., 3.3657 s vs. 6.4858 s.

Increment of input data

Two groups of experiments are arranged to observe Algorithm 4 for this case. The first group considers the total 4 times of incremental operations. That is, it takes the addition of 3000 training samples each time in an incremental way from all the remaining training samples. The second group keeps the same except 8000 rather than 3000 training samples. The incremental results are shown in Table 8.

It can be observed that the testing accuracy has a slight decline (e.g., from 94.18 to 93.54%) when the additional input data are fed in the first time into D&BLS, while the testing accuracy begins to go steadily up as more input data are fed in an incremental way and finally reaches the performance comparable to that of D&BLS with the one-shot construction (see the last row in Table 8). Obviously, the training time of Algorithm 4 is much less than that of D&BLS with the one-shot construction, e.g., 3.6203 s vs. 10.4513 s.

Table 7 Experimental results of the *MNIST* dataset using increment of feature nodes

Number of feature nodes	Number of enhancement nodes	Testing accuracy (%)	Each additional training time (s)	Accumulative training time (s)	Each additional testing time (s)	Accumulative testing time (s)
(1)						
696	275	94.10 (± 0.1031)	5.3374 (± 0.2309)	5.3374 (± 0.2309)	1.7812 (± 0.021)	1.7812 (± 0.021)
696 $\xrightarrow{30}$ 726	275 $\xrightarrow{30}$ 305	94.34 (± 0.1045)	0.7969 (± 0.0221)	0.7969 (± 0.0221)	0.3220 (± 0.0125)	0.3220 (± 0.0125)
726 $\xrightarrow{30}$ 756	305 $\xrightarrow{30}$ 335	94.54 (± 0.1120)	0.8312 (± 0.0117)	1.6281 (± 0.0278)	0.3173 (± 0.0122)	0.6393 (± 0.0215)
756 $\xrightarrow{30}$ 786	335 $\xrightarrow{30}$ 365	94.59 (± 0.1275)	0.8532 (± 0.0103)	2.4813 (± 0.0355)	0.3251 (± 0.0063)	0.9644 (± 0.0185)
786 $\xrightarrow{30}$ 816	365 $\xrightarrow{30}$ 395	94.63 (± 0.1249)	0.8844 (± 0.0254)	3.3657 (± 0.0514)	0.3344 (± 0.0103)	1.2988 (± 0.0236)
816	395	94.78 (± 0.0726)	6.4858 (± 0.1771)	6.4858 (± 0.1771)	2.2359 (± 0.0578)	2.2359 (± 0.0578)
(2)						
696	275	94.18 (± 0.1342)	5.2389 (± 0.0867)	5.2389 (± 0.0867)	1.7844 (± 0.0575)	1.7844 (± 0.0575)
696 $\xrightarrow{36}$ 732	275 $\xrightarrow{36}$ 311	94.38 (± 0.1463)	0.8266 (± 0.0148)	0.8266 (± 0.0148)	0.3204 (± 0.0105)	0.3204 (± 0.0105)
732 $\xrightarrow{36}$ 768	311 $\xrightarrow{36}$ 347	94.56 (± 0.1395)	0.8610 (± 0.0147)	1.6876 (± 0.0232)	0.3266 (± 0.0084)	0.6470 (± 0.0144)
768 $\xrightarrow{36}$ 804	347 $\xrightarrow{36}$ 383	94.65 (± 0.0905)	0.8891 (± 0.0148)	2.5767 (± 0.0283)	0.3391 (± 0.0122)	0.9861 (± 0.0148)
804 $\xrightarrow{36}$ 840	383 $\xrightarrow{36}$ 419	94.65 (± 0.1121)	0.9219 (± 0.0121)	3.4986 (± 0.0316)	0.3516 (± 0.0105)	1.3377 (± 0.0159)
840	419	94.87 (± 0.1374)	6.7951 (± 0.1470)	6.7951 (± 0.1470)	2.3531 (± 0.0678)	2.3531 (± 0.0678)

Bold values are the best one among all the results

Table 8 Experimental results of the *MNIST* dataset using increment of input data

Number of input data	Whole structure	Testing accuracy (%)	Each additional training time (s)	Accumulative training time (s)	Each additional testing time (s)	Accumulative testing time (s)
(1)						
10,000	(696, 275)	94.18 (± 0.1319)	5.2749 (± 0.0856)	5.2749 (± 0.0856)	1.8015 (± 0.0320)	1.8015 (± 0.0320)
10,000 $\xrightarrow{3000}$ 13,000	(696, 275)	93.54 (± 0.1247)	0.8344 (± 0.0254)	0.8344 (± 0.0254)	0.0438 (± 0.0062)	0.0438 (± 0.0062)
13,000 $\xrightarrow{3000}$ 16,000	(696, 275)	93.58 (± 0.1702)	0.8812 (± 0.0254)	1.7156 (± 0.0388)	0.0407 (± 0.0076)	0.0845 (± 0.0103)
16,000 $\xrightarrow{3000}$ 19,000	(696, 275)	93.70 (± 0.1340)	0.9297 (± 0.0290)	2.6453 (± 0.0437)	0.0344 (± 0.0117)	0.1189 (± 0.0200)
19,000 $\xrightarrow{3000}$ 22,000	(696, 275)	93.80 (± 0.1282)	0.9750 (± 0.0223)	3.6203 (± 0.0494)	0.0438 (± 0.0062)	0.1627 (± 0.0200)
22,000	(696, 275)	94.45 (± 0.0772)	10.4513 (± 0.4399)	10.4513 (± 0.4399)	1.8297 (± 0.0422)	1.8297 (± 0.0422)
(2)						
10,000	(696, 275)	94.23 (± 0.1352)	5.2811 (± 0.1100)	5.2811 (± 0.1100)	1.8047 (± 0.0234)	1.8047 (± 0.0234)
10,000 $\xrightarrow{8000}$ 18,000	(696, 275)	93.90 (± 0.1103)	2.1500 (± 0.3417)	2.1500 (± 0.3417)	0.0360 (± 0.0100)	0.0360 (± 0.0100)
18,000 $\xrightarrow{8000}$ 26,000	(696, 275)	93.91 (± 0.1181)	2.2375 (± 0.0503)	4.3875 (± 0.3849)	0.0391 (± 0.0078)	0.0751 (± 0.0136)
26,000 $\xrightarrow{8000}$ 34,000	(696, 275)	93.96 (± 0.1033)	2.3359 (± 0.0664)	6.7234 (± 0.4277)	0.0438 (± 0.0062)	0.1189 (± 0.0159)
34,000 $\xrightarrow{8000}$ 42,000	(696, 275)	93.99 (± 0.0980)	2.4609 (± 0.0515)	9.1843 (± 0.4776)	0.0391 (± 0.0105)	0.1580 (± 0.0203)
42,000	(696, 275)	94.53 (± 0.1463)	19.1291 (± 0.6380)	19.1291 (± 0.6380)	1.8875 (± 0.0853)	1.8875 (± 0.0853)

Bold values are the best one among all the results

Table 9 Running time of BLS and D&BLS (1) on the *COIL-100* dataset (s) and (2) on the *USPS* dataset (s)

Methods	Whole structure					
	(30 × 10, 1 × 2100)	(30 × 10, 1 × 2400)	(30 × 10, 1 × 2700)	(30 × 10, 1 × 3000)	(30 × 10, 1 × 3300)	(30 × 10, 1 × 3600)
(1)						
BLS	8.2685 (± 0.0781)	10.2137 (± 0.3123)	13.5371 (± 0.1296)	15.3323 (± 0.1815)	18.1041 (± 0.1018)	21.6025 (± 0.0645)
D&BLS	7.3779 (± 0.0319)	7.8779 (± 0.0435)	8.8247 (± 0.0219)	9.0825 (± 0.0243)	9.9622 (± 0.0671)	10.6684 (± 0.1216)
(2)						
BLS	9.6809 (± 0.1528)	12.0559 (± 0.0490)	15.2277 (± 0.0577)	21.8181 (± 0.2100)	23.4649 (± 0.1259)	27.2773 (± 0.0805)
D&BLS	5.3655 (± 0.1083)	6.1201 (± 0.0389)	7.2201 (± 0.0373)	8.3294 (± 0.0404)	9.3528 (± 0.0555)	9.8075 (± 0.0305)

Bold values are the best one among all the results

Running time comparison

In this subsection, we will show D&BLS's another attractive property—it only needs much less running time (i.e., training time and testing time) than BLS when they both share the same sizes (i.e., feature nodes and enhancement nodes). The *COIL-100* and the *USPS* datasets are taken for experiments. For a fair comparison, here we report average results and standard deviations for ten trials on each dataset.

D&BLS takes a three-layer structure and each sub-system keeps the same parameters setting, especially the number of selected feature nodes is 1. In other words, if a BLS has 30 × 10 feature nodes and 1 × 2100 enhancement nodes, each sub-system in the corresponding D&BLS will contain 10 × 10 feature nodes, 1 selected feature node, and 1 × 700 enhancement nodes. The obtained experimental results are given in Table 9.

According to Table 9, D&BLS indeed needs much less running time than BLS. What is more, D&BLS runs much more quickly than BLS with the increasing number of enhancement nodes. This tendency becomes considerably surprising in the case of 3600 enhancement nodes for the *USPS* dataset. That is, BLS occupies 27.2773 s while D&BLS occupies only 9.8075 s for the *USPS* dataset.

Conclusion

While the recently developed broad learning system BLS has been exhibiting its promising performance without the need of deep structure, it often requires a huge heap of hidden nodes, which may perhaps inevitably cause both overwhelming storage requirement and overfitting phenomenon. In this paper, D&BLS is proposed to synthesize both deep and broad learning so as to guarantee enhance performance, downsize the structure and reduce the running time of the general BLS. D&BLS consists of several lightweight BLS sub-systems. D&BLS has its structure novelty in its joint

stacking way of feature augmentation and residuals boosting. The whole learning algorithm and three incremental algorithms of D&BLS are proposed and then verified by our experiments about image datasets and *UCI* datasets.

Since the parameter settings of each sub-system in D&BLS are given only by the trial and error strategy, a deeper D&BLS sometimes cannot assure better performance than a shallower D&BLS. As a result, how to determine the structure of D&BLS in a moderate time and how to design a deeper D&BLS with desirable ability for practical application scenarios are our research works in near future.

Funding This work was supported in part by the National Natural Science Foundation of China under Grants 61572236, 6197071117.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit <http://creativecommons.org/licenses/by/4.0/>.

References

1. Gong M, Zhao J, Liu J, Miao Q, Jiao L (2016) Change detection in synthetic aperture radar images based on deep neural networks. *IEEE Trans Neural Netw Learn Syst* 27(1):125–138
2. Shao F, Lin W, Wang S, Jiang G, Yu M (2015) Blind image quality assessment for stereoscopic images using binocular guided quality lookup and visual codebook. *IEEE Trans Broadcast* 61(2):154–165
3. Salakhutdinov R, Hinton GE (2009) Deep boltzmann machine. In: *Proceedings of AISTATS*, vol 1, p 3
4. Hinton GE, Osindero S, The Y-W (2006) A fast learning algorithm for deep belief nets. *Neural Comput* 18:899–7667

5. Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. *Science* 313(5786):504–507
6. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition (online). <https://arxiv.org/abs/1409.1556>
7. Chen CLP, Liu ZL (2018) Broad learning system: an effective and efficient increment learning system without the need for deep architecture. *IEEE Trans Neural Netw Learn Syst* 29(1):10–24
8. Pao Y-H, Takefuji Y (1992) functional-link net computing: theory, system architecture and functionalities. *Computer* 25(5):76–79
9. Pao Y-H, Park G-H, Sobajic DJ (1994) Learning and generalization characteristics of the random vector functional-link net. *Neurocomputing* 6(2):163–180
10. Feng S, Chen CLP (2018) Fuzzy broad learning system: a novel neuro-fuzzy model for regression and classification. In: *IEEE transactions on cybernetics*
11. Jiang Y, Chung F, Ishibuchi H, Deng Z, Wang S (2015) Multitask TSK fuzzy system modeling by mining intertask common hidden structure. *IEEE Trans Cybern* 45(3):534–547
12. Gu X, Chung F, Wang S (2017) Bayesian Takagi–Sugeno–Kang fuzzy classifier. *IEEE Trans Fuzzy Syst* 25(6):1655–1671
13. Deng Z, Cao L, Jiang Y, Wang S (2015) Minimax probability TSK fuzzy system classifier: a more transparent and highly interpretable classification model. *IEEE Trans Fuzzy Syst* 23(4):813–826
14. Gong M, Liu J, Li H, Cai Q, Su L (2015) A multiobjective sparse feature learning model for deep neural networks. *IEEE Trans Neural Netw Learn Syst* 26(12):263–327
15. Hoerl ZAE, Kennard RW (2000) Ridge regression: biased estimation for nonorthogonal problems. *Technometrics* 42(1):80–86
16. Chen CLP, Liu ZL, Shuang F (2019) Universal approximation capability of broad learning system and its structural variations. *IEEE Trans Neural Netw Learn Syst* 30(4):1191–1204
17. Ting KM, Witten IH (1997) Stacked generalization: when does it work? In: *Fifteenth international joint conference on artificial intelligence*. Morgan Kaufmann
18. Rumelhart DE, McClelland JL (1986) *Parallel distributed processing: explorations in the microstructure of cognition*, vol 1. Foundations of Research. MIT Press, Cambridge
19. Freund Y, Schapire R, Abe N (1999) A short introduction to boosting. *Jpn Soc Artif Intell* 14(771–780):1612
20. Bühlmann P, Hothorn T (2007) Boosting algorithms: regularization, prediction and model fitting. *Statistical Science*, pp 477–505
21. Friedman J, Hastie T, Tibshirani R (2000) Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *Ann Stat* 28(2):337–407
22. Notley SV, Bell SL, Simpson D, Smith DC (2008) Applying bootstrap techniques to detect differences in auditory evoked potentials: possible use in anaesthesia monitoring. In: *4th IET international conference on advances in medical, signal and information processing—MEDSIP 2008*, Santa Margherita Ligure, pp 1–4
23. Gulcehre C, Moczulski M, Denil M et al (2016) Noisy activation functions. In: *International conference on machine learning*, pp 3059–3068
24. Blake CL, Merz CJ (1988) *UCI repository of machine learning databases*, vol. 55. Dept. Inf. Comput. Sci., Univ. California, Irvine (online). <http://archive.ics.uci.edu/ml/dataset.html>
25. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. *Proc IEEE* 86(11):2278–2324
26. Cortes C, Vapnik VN (1995) Support vector networks. *Mach Learn* 20(3):273–297
27. Suykens JAK, Vandewalle J (1999) Least squares support vector machine classifiers. *Neural Process Lett* 9(3):293–300
28. Pedregosa et al (2011) Scikit-learn: machine Learning in Python. *JMLR* 12:2825–2830
29. Hull JJ (1994) A database for handwritten text recognition research. *IEEE Trans Syst Man Cybern B Cybern* 16(5):550–554
30. S. A. Nene, S. K. Nayar and H. Murase, “Columbia Object Image Library (COIL-20),” *Technical Report CUCS-005-96*, February 1996
31. Nene SA, Nayar SK, Murase H (1969) Columbia Object Image Library (COIL-100), Technical Report CUCS-006-96
32. Lee K-C, Ho J, Kriegman D (2005) Acquiring linear subspaces for face recognition under variable lighting. *IEEE Trans Pattern Anal Mach Intell* 27(5):138–142
33. Isolet Dataset (1994) (online). <http://featureselection.asu.edu/datasets.php>

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.