A Hybrid Recursive Implementation of Broad Learning With Incremental Features

Liu, Di; Baldi, Simone; Yu, Wenwu; Chen, C. L.P.

DOI
10.1109/TNNLS.2020.3043110

Publication date
2022

Document Version
Final published version

Published in
IEEE Transactions on Neural Networks and Learning Systems

A Hybrid Recursive Implementation of Broad Learning With Incremental Features

Di Liu, Simone Baldi, *Senior Member, IEEE*, Wenwu Yu, *Senior Member, IEEE*, and C. L. Philip Chen, *Fellow, IEEE*

***Abstract*— The broad learning system (BLS) paradigm has recently emerged as a computationally efficient approach to supervised learning. Its efficiency arises from a learning mechanism based on the method of least-squares. However, the need for storing and inverting large matrices can put the efficiency of such mechanism at risk in big-data scenarios. In this work, we propose a new implementation of BLS in which the need for storing and inverting large matrices is avoided. The distinguishing features of the designed learning mechanism are as follows: 1) the training process can balance between efficient usage of memory and required iterations (hybrid recursive learning) and 2) retraining is avoided when the network is expanded (incremental learning). It is shown that, while the proposed framework is equivalent to the standard BLS in terms of trained network weights,much larger networks than the standard BLS can be smoothly trained by the proposed solution, projecting BLS toward the big-data frontier.**

***Index Terms*— Big data, broad learning system (BLS), recursive learning, training time.**

I. Introduction

SUPERVISED learning is a branch of machine learning whose goal is to learn a predictor function (sometimes called "hypothesis") from input data and labeled data. The learning mechanism consists of optimization algorithms that shape the predictor function, so as to be as accurate and general as possible [1], [2]. For example, in deep supervised learning, the predictor function takes the form of a neural network with many hierarchical layers [3]–[6]. Such deep structures have been successfully adopted in pattern recognition [7]–[10], biological data analysis [11], [12], adversarial networks [13], [14], brain activity [15], [16], tomography [17], finance and trading [18], and many other domains.

Manuscript received November 4, 2019; revised March 8, 2020, June 24, 2020, and October 7, 2020; accepted November 25, 2020. Date of publication December 22, 2020; date of current version April 5, 2022. This work was supported in part by the Special Guiding Funds Double First-Class under Grant 3307012001A; in part by the Natural Science Foundation of China under Grant 62073074, Grant 61673107, and Grant 62073076; and in part by the Jiangsu Provincial Key Lab of Networked Collective Intelligence under Grant BM2017002. *(Di Liu and Simone Baldi contributed equally to this work.) (Corresponding author: Wenwu Yu.)*

Di Liu is with the School of Cyber Science and Engineering, Southeast University, Nanjing 210096, China (e-mail: liud923@126.com).

Simone Baldi is with the School of Mathematics, Southeast University, Nanjing 210096, China, and also with the Delft Center for Systems and Control, Delft University of Technology (TU Delft), 2628 CD Delft, The Netherlands (e-mail: s.baldi@tudelft.nl).

Wenwu Yu is with the School of Cyber Science and Engineering, Southeast University, Nanjing 210096, China, and also with the School of Mathematics, Southeast University, Nanjing 210096, China (e-mail: wwyu@seu.edu.cn).

C. L. Philip Chen is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510641, China (e-mail: philip.chen@ieee.org).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TNNLS.2020.3043110.

Digital Object Identifier 10.1109/TNNLS.2020.3043110

A. Background on Broad Learning System

The broad learning system (BLS) paradigm has recently emerged as a supervised learning method in which the hypothesis is improved by expanding the network in width rather than depth. Reasons behind the study of BLS come from some recognized issues of deep structures, such as overcoming the need for gradient-descent training that might make learning slow [19]–[21], preventing complete retraining when new hypotheses must be formulated [22]–[24], and avoiding huge processing power, such as graphics processing units (GPUs) that are typically needed in deep structures [25].

Seminal ideas of BLS are in [26] and [27]. In these works, "one-shot" nonrecursive least-squares training was studied for flat neural networks (functional-link networks [28]). The so-called "incremental learning," i.e., training an expanded network without retraining the whole network, was proposed in [29]. The competitive performance of BLS with respect to many deep learning algorithms was demonstrated in the same work, via data set benchmarks from MNIST and NYU-NORB. Furthermore, the nonrecursive least-squares training mechanism of BLS was shown to be extremely faster than gradient-descent training. Other applications of BLS include leakage detection [30], traffic flow prediction [31], process monitoring [32], and fault diagnosis [33]. Variants to the standard BLS have been also proposed in terms of broad-deep (recurrent, neurofuzzy, and convolutional) combined structures [34]–[37]. Comparative experiments in these works confirm the effectiveness of BLS and its variants in terms of training speed and prediction capability: however, BLS does not come without challenges, as clarified hereafter.

B. Open Challenges and Motivation for This Work

Let us use a standard notation in which $\mathbb{R}^{N\times M}$ represents the space of real matrices of dimension $N \times M$. Vectors and matrices are boldface italic: the letters $\boldsymbol{X}$, $\boldsymbol{A}$, and $\boldsymbol{Y}$ are used to represent input data, transformed BLS input data, and labeled data; the transformed BLS data $\boldsymbol{A}$ contain feature nodes and enhancement nodes, indicated with the letters $\boldsymbol{Z}$ and $\boldsymbol{H}$; and the letter $\boldsymbol{W}$ is used for network weights. The identity matrix is

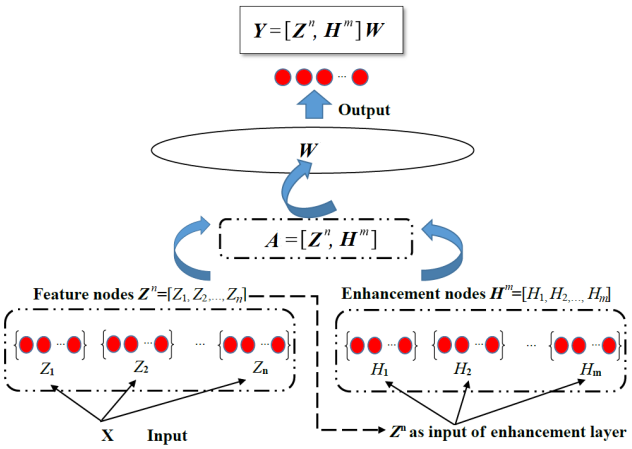


Fig. 1. Illustration of a BLS.

indicated with $\boldsymbol{I}$. Depending on their dimensions and on the required operations, vectors will be stacked along columns, as in $[\boldsymbol{Z}, \boldsymbol{H}]$, or along rows, as in $\begin{bmatrix} \boldsymbol{W}_f \\ \boldsymbol{W}_c \end{bmatrix}$.

Let us denote a set of N labeled data samples and a set of N BLS input data samples with

$$\boldsymbol{Y} = \begin{bmatrix} \boldsymbol{Y}(1) \\ \boldsymbol{Y}(2) \\ \vdots \\ \boldsymbol{Y}(N) \end{bmatrix} \in \mathbb{R}^{N\times Q} \quad \boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}(1) \\ \boldsymbol{A}(2) \\ \vdots \\ \boldsymbol{A}(N) \end{bmatrix} \in \mathbb{R}^{N\times M} \tag{1}$$

where N is the number of samples, Q is the size of the output and M is the size of the network. In BLS, the input data $\boldsymbol{A}$ result from a transformation of the original input $\boldsymbol{X}$ via feature and enhancement nodes: this is sketched in Fig. 1 and will be further clarified later. The key learning mechanism of BLS[1] requires to invert an $M \times M$ matrix, i.e.,

$$\hat{\boldsymbol{Y}} = \boldsymbol{A}\boldsymbol{W} \overset{\text{training}}{\rightarrow} \boldsymbol{W} = \left(\boldsymbol{A}^T\boldsymbol{A} + \lambda \boldsymbol{I}\right)^{-1}\boldsymbol{A}^T\boldsymbol{Y} \tag{2}$$

with $\boldsymbol{W} \in \mathbb{R}^{M\times Q}$ being the set of weights to be learned, $\hat{\boldsymbol{Y}}$ the predicted labels, and λ a positive regularization constant. Least-squares training in (2) processes all data "one-shot" at the same time via the matrix inversion. However, for large/complex data sets requiring large networks, it is practically impossible to perform the $M \times M$ matrix inversion in (2) to obtain the weights $\boldsymbol{W}$. In fact, matrix inversion has computational cost $\mathcal{O}(M^3)$, and in the authors' experience, out-of-memory problems are easily faced in standard desktop PCs (Intel Core i5 Quad-Core—8-GB DDR4—BLS implemented in MATLAB) when $N > 100\,000$, $M > 10\,000$, and $Q > 100$.

The main drive behind this work is to explore such training tradeoffs in order to push the capabilities of BLS toward the big-data frontier. The objectives of this work are finding an alternative BLS method with similar training performance as "one-shot" least-squares, but without requiring the inversion of large matrices, and retaining the same incremental properties of BLS, i.e., train only the network portions added to the original structure.

[1]Let us neglect, for simplicity, other mechanisms of BLS, such as sparse autoencoder and SVD simplification.

To accomplish these objectives, a novel recursive least-squares method is tailored to the needs of BLS. Different from the standard recursive least-squares algorithms proposed in the signal processing literature, the distinguishing features of the proposed one are embodying a hybrid nature that can trade between efficient usage of memory and number of recursions, which is achieved by exploiting a hybrid version of the matrix inversion lemma, and enjoying incremental learning capabilities that avoid a complete retraining process when the BLS network is expanded, which is achieved by appropriately exploiting the linear-in-the-parameter structure of BLS. We call the proposed framework hybrid recursive BLS (HR-BLS). We show that the learning accuracy of HR-BLS is equivalent to the standard BLS, implying that all comparisons between BLS and deep algorithms shown in [29], [34], [37], and [38] are inherited by HR-BLS. However, regression experiments on a data set with more than 100 000 samples show that HR-BLS smoothly accomplishes training on huge networks where the standard BLS exhibits out-of-memory problems. We further show that the incremental learning speed of the proposed formulation is faster than incremental learning of the original BLS due to more efficient usage of memory. This work is focused on the BLS framework, but we expect the proposed ideas to find application in other learning frameworks where the inversion of large matrices places a crucial rule. An example in this sense is the weights and structure determination (WASD) framework [39], [40], where matrix inversion is achieved via pseudoinverse and used to determine the network weights and the network structure (e.g., number of hidden-layer neurons) [41].

The rest of this article is organized as follows. Section II presents the BLS with recursive and hybrid recursive training. Section III extends such tools for incrementally training BLS when the network architecture is expanded. Section IV presents the experiments, with concluding remarks in Section V.

II. Hybrid Recursions in Broad Learning

Given input data $\boldsymbol{X}$, a broad learning network takes the form

$$\hat{\boldsymbol{Y}} = \left[\boldsymbol{Z}_1, \boldsymbol{Z}_2, \ldots, \boldsymbol{Z}_n\right]\begin{bmatrix} \boldsymbol{W}_{f_1} \\ \vdots \\ \boldsymbol{W}_{f_n} \end{bmatrix} + \left[\boldsymbol{H}_1, \boldsymbol{H}_2, \ldots, \boldsymbol{H}_m\right]\begin{bmatrix} \boldsymbol{W}_{c_1} \\ \vdots \\ \boldsymbol{W}_{c_m} \end{bmatrix} \tag{3}$$

where $\boldsymbol{Z}^n = [\boldsymbol{Z}_1, \boldsymbol{Z}_2, \ldots, \boldsymbol{Z}_n]$ are the mapped features

$$\boldsymbol{Z}_i = \varphi_i\left(\boldsymbol{X}\boldsymbol{W}_{e_i} + \boldsymbol{\beta}_{e_i}\right), \quad i = 1, \ldots, n \tag{4}$$

and $\boldsymbol{H}^m = [\boldsymbol{H}_1, \boldsymbol{H}_2, \ldots, \boldsymbol{H}_m]$ are the enhancement nodes

$$\boldsymbol{H}_j = \zeta_j\left(\boldsymbol{Z}^n\boldsymbol{W}_{h_j} + \boldsymbol{\beta}_{h_j}\right), \quad j = 1, \ldots, m. \tag{5}$$

The weights $\boldsymbol{W}_{e_i}$, $\boldsymbol{\beta}_{e_i}$, $\boldsymbol{W}_{h_j}$, and $\boldsymbol{\beta}_{h_j}$, are the weights of the activation functions φ_i and ζ_j that are selected randomly according to the functional network architecture [28]. The activation functions φ_i and ζ_j are sigmoid functions as in

standard neural networks. After defining

$$\boldsymbol{W}_f^n = \begin{bmatrix} \boldsymbol{W}_{f_1} \\ \vdots \\ \boldsymbol{W}_{f_n} \end{bmatrix} \quad \boldsymbol{W}_c^m = \begin{bmatrix} \boldsymbol{W}_{c_1} \\ \vdots \\ \boldsymbol{W}_{c_m} \end{bmatrix} \tag{6}$$

the model (3) can be written in a compact way as

$$\hat{\boldsymbol{Y}} = \underbrace{[\boldsymbol{Z}^n, \boldsymbol{H}^m]}_{\boldsymbol{A}} \underbrace{\begin{bmatrix} \boldsymbol{W}_f^n \\ \boldsymbol{W}_c^m \end{bmatrix}}_{\boldsymbol{W}} \tag{7}$$

which has the structure in (2) and can be trained using the one-shot training in (2). Different from one-shot training, in the following, we will explain how recursive training can be formulated, giving the same results as (2), but avoiding the inversion of large matrices.

A. Recursive Learning

The training in (2) is referred to as "one-shot" since the data $\boldsymbol{X}$ are collected *a priori* over large intervals of time, transformed into $\boldsymbol{A}$ and processed together offline to generate the network weights $\boldsymbol{W}$. This strategy is also referred to as nonrecursive and can be written as

$$\boldsymbol{W} = f(\boldsymbol{A}, \boldsymbol{Y})$$

where $f(\cdot)$ is the transformations performed by the BLS network in (2) using all data. Different from this strategy, recursive algorithms aim to generate new network weights $\boldsymbol{W}(k)$ at discrete steps k, using data that can be collected/transformed at step k and also using the previous network weights $\boldsymbol{W}(k-1)$. Updating past $\boldsymbol{W}(k)$ from $\boldsymbol{W}(k-1)$ requires a set of recursive equations (i.e., difference equations), which can be written as

$$\begin{aligned} \boldsymbol{W}(k) &= f_{\boldsymbol{W}}(\boldsymbol{W}(k-1), \boldsymbol{P}(k-1), \boldsymbol{A}(k), \boldsymbol{Y}(k)) \\ \boldsymbol{P}(k) &= f_{\boldsymbol{P}}(\boldsymbol{P}(k-1), \boldsymbol{A}(k), \boldsymbol{Y}(k)) \\ \boldsymbol{W}(0) &= \boldsymbol{W}_0, \quad \boldsymbol{P}(0) = \boldsymbol{P}_0 \end{aligned}$$

where $\boldsymbol{P}$ is an auxiliary variable used for the recursive update (i.e., the covariance matrix). The recursions must initialized with $\boldsymbol{W}(0)$ and $\boldsymbol{P}(0)$, representing the initial values of the network weights and covariance matrix. Because $f_{\boldsymbol{W}}$ and $f_{\boldsymbol{P}}$ only use current data $\boldsymbol{A}(k)$ and $\boldsymbol{Y}(k)$, recursions are more suited for real-time online processing. In particular, they well suit the purpose of incremental learning where the structure of the network can grow online as new data arrive.

To more clearly present the recursive equations, consider an input data set $\boldsymbol{X}(1), \ldots, \boldsymbol{X}(k)$, and create the BLS transformed input $\boldsymbol{A}(1), \ldots, \boldsymbol{A}(k)$ as in (7). Also, consider the labeled data $\boldsymbol{Y}(1), \ldots, \boldsymbol{Y}(k)$, and define, for compactness

$$\boldsymbol{Y}_k = \begin{bmatrix} \boldsymbol{Y}(1) \\ \boldsymbol{Y}(2) \\ \vdots \\ \boldsymbol{Y}(k) \end{bmatrix} \in \mathbb{R}^{k \times Q} \quad \boldsymbol{A}_k = \begin{bmatrix} \boldsymbol{A}(1) \\ \boldsymbol{A}(2) \\ \vdots \\ \boldsymbol{A}(k) \end{bmatrix} \in \mathbb{R}^{k \times M}. \tag{8}$$

Let us formulate the training task via the minimization of the following cost at step k:

$$\begin{aligned} J_k(\boldsymbol{W}) = &\frac{1}{2}(\boldsymbol{Y}_k - \boldsymbol{A}_k \boldsymbol{W})^T (\boldsymbol{Y}_k - \boldsymbol{A}_k \boldsymbol{W}) \\ &+ \frac{1}{2}(\boldsymbol{W} - \boldsymbol{W}_0)^T \boldsymbol{P}_0^{-1} (\boldsymbol{W} - \boldsymbol{W}_0) + \frac{1}{2}\lambda \boldsymbol{W}^T \boldsymbol{W} \end{aligned} \tag{9}$$

where the first term refers to the approximation of $\boldsymbol{Y}_k$ via the estimate $\hat{\boldsymbol{Y}}_k = \boldsymbol{A}_k \boldsymbol{W}$ (regression task); the last term with λ is used for regularization to avoid overfitting; and the second term with $\boldsymbol{W}_0$ and $\boldsymbol{P}_0$ also plays the role of a regularization term and can be further used for initialization purposes (in fact, it allows to initialize the network weights $\boldsymbol{W}(0) = \boldsymbol{W}_0$ and the covariance matrix $\boldsymbol{P}(0) = \boldsymbol{P}_0$). The network weights $\boldsymbol{W}$ are to be determined as a result of training. Due to the linear-in-the-parameter structure, the cost (9) is convex in $\boldsymbol{W}$ and can be minimized by setting $\nabla J_k(\boldsymbol{W}) = 0$, giving the optimal $\boldsymbol{W}^*(k)$

$$\boldsymbol{W}^*(k) = \left(\boldsymbol{A}_k^T \boldsymbol{A}_k + \boldsymbol{P}_0^{-1} + \lambda \boldsymbol{I}\right)^{-1} \left(\boldsymbol{P}_0^{-1} \boldsymbol{W}_0 + \boldsymbol{A}_k^T \boldsymbol{Y}_k\right). \tag{10}$$

In the following, we consider for simplicity $\boldsymbol{P}_0^{-1} = q_0^{-1} \boldsymbol{I}$, for a design scalar $q_0 > 0$. Let us also consider (10) in place of (2) because, for large q_0, $\boldsymbol{W}^*(N)$ from (10) tends to the solution in (2) [42, Sec. 4.6]. The following result can be derived.

Proposition 1: The recursive algorithm

$$\begin{aligned} \boldsymbol{\epsilon}(k) &= \boldsymbol{Y}(k) - \boldsymbol{A}(k) \boldsymbol{W}(k-1) \\ \boldsymbol{P}(k) &= \boldsymbol{P}(k-1) - \frac{\boldsymbol{P}(k-1) \boldsymbol{A}^T(k) \boldsymbol{A}(k) \boldsymbol{P}(k-1)}{1 + \boldsymbol{A}(k) \boldsymbol{P}(k-1) \boldsymbol{A}^T(k)} \\ \boldsymbol{W}(k) &= \boldsymbol{W}(k-1) + \boldsymbol{P}(k) \boldsymbol{A}^T(k) \boldsymbol{\epsilon}(k) \end{aligned} \tag{11}$$

with $\boldsymbol{W}(0) = \boldsymbol{W}_0$ and $\boldsymbol{P}(0) = (q_0 + \lambda^{-1}) \boldsymbol{I}$, makes $\boldsymbol{W}(k) = \boldsymbol{W}^*(k)$ at each step k. That is, $\boldsymbol{W}(k)$ resulting from the recursions (11) coincides with the "one-shot" solution $\boldsymbol{W}^*(k)$ in (10) that processes k data at the same time.

Proof: Let us define

$$\begin{aligned} \boldsymbol{P}^{-1}(k) &= \boldsymbol{A}_k^T \boldsymbol{A}_k + \boldsymbol{P}_0^{-1} + \lambda \boldsymbol{I} \\ &\Rightarrow \boldsymbol{P}^{-1}(k) = \boldsymbol{P}^{-1}(k-1) + \boldsymbol{A}^T(k) \boldsymbol{A}(k). \end{aligned} \tag{12}$$

Let us apply the matrix inversion lemma (or the Woodbury matrix identity [43, Sec. 2.7.3])

$$(\boldsymbol{D} + \boldsymbol{B}\boldsymbol{C})^{-1} = \boldsymbol{D}^{-1} - \boldsymbol{D}^{-1} \boldsymbol{B} \left(\boldsymbol{I} + \boldsymbol{C} \boldsymbol{D}^{-1} \boldsymbol{B}\right)^{-1} \boldsymbol{C} \boldsymbol{D}^{-1} \tag{13}$$

to (12), by taking $\boldsymbol{D} = \boldsymbol{P}^{-1}(k-1)$, $\boldsymbol{B} = \boldsymbol{A}^T(k)$, and $\boldsymbol{C} = \boldsymbol{A}(k)$. This results in

$$\begin{aligned} \boldsymbol{P}(k) = &\boldsymbol{P}(k-1) - \boldsymbol{P}(k-1) \boldsymbol{A}^T(k) \\ &\cdot \left(1 + \boldsymbol{A}(k) \boldsymbol{P}(k-1) \boldsymbol{A}^T(k)\right)^{-1} \boldsymbol{A}(k) \boldsymbol{P}(k-1). \end{aligned} \tag{14}$$

Note that the inversion involved in (14) is the inversion of a scalar. Substitute (14) into (10), so as to obtain

$$\begin{aligned} \boldsymbol{W}^*(k) &= \boldsymbol{P}(k) \left(\boldsymbol{P}_0^{-1} \boldsymbol{W}_0 + \boldsymbol{A}_k^T \boldsymbol{Y}_k\right) \\ &= \boldsymbol{P}(k) \left(\boldsymbol{P}_0^{-1} \boldsymbol{W}_0 + \boldsymbol{A}_{k-1}^T \boldsymbol{Y}_{k-1} + \boldsymbol{A}^T(k) \boldsymbol{Y}(k)\right). \end{aligned} \tag{15}$$

From using (10), at step $k-1$, we obtain

$$\begin{aligned} &\boldsymbol{P}_0^{-1} \boldsymbol{W}_0 + \boldsymbol{A}_{k-1}^T \boldsymbol{Y}_{k-1} \\ &\quad = \left(\boldsymbol{A}_{k-1}^T \boldsymbol{A}_{k-1} + \boldsymbol{P}_0^{-1} + \lambda \boldsymbol{I}\right) \boldsymbol{W}^*(k-1) \\ &\quad = \boldsymbol{P}^{-1}(k-1) \boldsymbol{W}^*(k-1) \end{aligned} \tag{16}$$

which, substituted in (15), leads to

$$\begin{aligned} \boldsymbol{W}^*(k) &= \boldsymbol{P}(k) \left[\left(\boldsymbol{P}^{-1}(k) - \boldsymbol{A}^T(k) \boldsymbol{A}(k)\right) \boldsymbol{W}^*(k-1) \right. \\ &\qquad \left. + \boldsymbol{A}^T(k) \boldsymbol{Y}(k)\right] \\ &= \boldsymbol{W}^*(k-1) \\ &\quad + \boldsymbol{P}(k) \boldsymbol{A}^T(k) \left[-\boldsymbol{A}(k) \boldsymbol{W}^*(k-1) + \boldsymbol{Y}(k)\right] \end{aligned} \tag{17}$$

which is a recursive relation between $\boldsymbol{W}^*(k)$ and $\boldsymbol{W}^*(k-1)$. Therefore, it can be seen that (11) makes $\boldsymbol{W}(k) = \boldsymbol{W}^*(k)$, provided that the initial conditions are selected as $\boldsymbol{P}(0) = (q_0 + \lambda^{-1})\boldsymbol{I}$ and $\boldsymbol{W}(0) = \boldsymbol{W}_0$. This concludes the proof. □

Remark 1: Even though $\boldsymbol{W}^*(k)$ and $\boldsymbol{W}(k)$ both arise from minimization of the same cost function (9), it is important to remark the difference between calculating $\boldsymbol{W}^*(k)$ from (10) and calculating $\boldsymbol{W}(k)$ from (11). When new data are considered at step $k+1$, (10) requires to process all data "one-shot" again by inverting an $M \times M$ matrix. The algorithm (11) only needs to process the new samples.

Algorithm (11) is known in the signal processing literature as the recursive least-squares algorithm [42, Sec. 4.6]. In what follows, we want to tailor its mechanism to BLS.

B. Hybrid Recursions

Recursive least-squares avoid matrix inversion but need to span all data one sample at a time. With huge amounts of data, processing might be slow. In other words, learning tradeoffs in least-square optimization problems arise from matrix manipulations, i.e., the time required to store/invert large matrices needs time (this is the case of the nonrecursive least-squares (10), where the matrix manipulation time increases with the network size) and the number of iterations, i.e., the time required to span the data one sample at a time (this is the case of the recursive least-squares (11), where the number of iterations grows with the number of data). In the following, we would like to find an efficient tradeoff between matrix manipulations and the number of iterations. That is, we want to derive a recursive algorithm with efficient usage of memory but also fast learning capabilities.

The idea is to adopt a hybrid recursive strategy, in which data are processed in small batches, so that the number of iterations is smaller. After every batch, the least-square gains are updated recursively, so no large matrix manipulation is required. Let us denote with b the size of the batch, and define the indices spanning from one batch to another: $b_1 = 1, \ldots, b$, $b_2 = b+1, \ldots, 2b, \ldots, b_{\bar{k}} = (\bar{k}-1)b+1, \ldots, \bar{k}b$. Define also

$$\boldsymbol{A}_{b_{\bar{k}}} = \begin{bmatrix} \boldsymbol{A}(b_1) \\ \boldsymbol{A}(b_2) \\ \vdots \\ \boldsymbol{A}(b_{\bar{k}}) \end{bmatrix} \in \mathbb{R}^{\bar{k}b \times M}, \quad \boldsymbol{A}(b_i) = \begin{bmatrix} \boldsymbol{A}((i-1)b+1) \\ \boldsymbol{A}((i-1)b+2) \\ \vdots \\ \boldsymbol{A}(ib) \end{bmatrix} \tag{18}$$

with $i = 1, 2, \ldots$ [note that $\bar{k}b = k$, with k as in (8)].

Proposition 2: The recursive algorithm

$$\begin{aligned} \boldsymbol{\epsilon}(b_{\bar{k}}) &= \boldsymbol{Y}(b_{\bar{k}}) - \boldsymbol{A}(b_{\bar{k}})\boldsymbol{W}(b_{\bar{k}-1}) \\ \boldsymbol{P}(b_{\bar{k}}) &= \boldsymbol{P}(b_{\bar{k}-1}) - \boldsymbol{P}(b_{\bar{k}-1})\boldsymbol{A}^T(b_{\bar{k}}) \\ &\quad \cdot \left(\boldsymbol{I}_b + \boldsymbol{A}(b_{\bar{k}})\boldsymbol{P}(b_{\bar{k}-1})\boldsymbol{A}^T(b_{\bar{k}})\right)^{-1} \boldsymbol{A}(b_{\bar{k}})\boldsymbol{P}(b_{\bar{k}-1}) \\ \boldsymbol{W}(b_{\bar{k}}) &= \boldsymbol{W}(b_{\bar{k}-1}) + \boldsymbol{P}(b_{\bar{k}})\boldsymbol{A}^T(b_{\bar{k}})\boldsymbol{\epsilon}(b_{\bar{k}}) \end{aligned} \tag{19}$$

with $\boldsymbol{W}(0) = \boldsymbol{W}_0$ and $\boldsymbol{P}(0) = (q_0 + \lambda^{-1})\boldsymbol{I}$, makes $\boldsymbol{W}(b_{\bar{k}}) = \boldsymbol{W}^*(b_{\bar{k}})$ at each batch index $b_{\bar{k}}$. That is, $\boldsymbol{W}(b_{\bar{k}})$ resulting from the recursions (19) coincides with the "one-shot" solution $\boldsymbol{W}^*(b_{\bar{k}})$ in (10) that processes k data at the same time.

Proof: In place of (14), let us consider

$$\begin{aligned} \boldsymbol{P}^{-1}(b_{\bar{k}}) - \boldsymbol{P}^{-1}(b_{\bar{k}-1}) &= \boldsymbol{A}_{b_{\bar{k}}}^T \boldsymbol{A}_{b_{\bar{k}}} - \boldsymbol{A}_{b_{\bar{k}-1}}^T \boldsymbol{A}_{b_{\bar{k}-1}} \\ &= \boldsymbol{A}^T(b_{\bar{k}})\boldsymbol{A}(b_{\bar{k}}). \end{aligned} \tag{20}$$

The proof follows similar steps as the proof of Proposition 1, provided that the matrix inversion lemma (13) for (20) is used with $\boldsymbol{D} = \boldsymbol{P}^{-1}(b_{\bar{k}-1})$, $\boldsymbol{B} = \boldsymbol{A}^T(b_{\bar{k}})$, and $\boldsymbol{C} = \boldsymbol{A}(b_{\bar{k}})$. This results in

$$\begin{aligned} \boldsymbol{P}(b_{\bar{k}}) = \boldsymbol{P}(b_{\bar{k}-1}) &- \boldsymbol{P}(b_{\bar{k}-1})\boldsymbol{A}^T(b_{\bar{k}}) \\ &\cdot \left(\boldsymbol{I}_b + \boldsymbol{A}(b_{\bar{k}})\boldsymbol{P}(b_{\bar{k}-1})\boldsymbol{A}^T(b_{\bar{k}})\right)^{-1} \boldsymbol{A}(b_{\bar{k}})\boldsymbol{P}(b_{\bar{k}-1}). \end{aligned} \tag{21}$$

Then, an analogous recursive relation as (17) between $\boldsymbol{W}^*(b_{\bar{k}})$ and $\boldsymbol{W}^*(b_{\bar{k}-1})$ can be obtained, leading to $\boldsymbol{W}(b_{\bar{k}}) = \boldsymbol{W}^*(b_{\bar{k}})$, provided that the algorithm (19) is initialized with $\boldsymbol{P}(0) = (q_0 + \lambda^{-1})\boldsymbol{I}$ and $\boldsymbol{W}(0) = \boldsymbol{W}_0$. This concludes the proof. □

We refer to algorithm (19) as hybrid recursive least-square algorithm. The algorithm suggests a different implementation to the standard BLS framework stemming from (14). This implementation will be explored in Section III. Compared with (14), the matrix inversion in (21) has now dimension $b \times b$, which leads to some remarks.

Remark 2: The relation (21) reveals a tradeoff between matrix manipulations and the number of iterations: for $b = 1$, one has the standard recursive least-squares (11) (low cost for matrix inversion but needs to span one sample at a time). For $b > 1$, one can span faster all data, at the price of increasing the cost for matrix inversion. We will see, in the simulation experiments and from computational considerations, that there is an optimum batch size b for which training time is minimized (see Figs. 2 and 3).

Remark 3: The benefit of the inversion lemma (13), resulting in (21), is to transform the $M \times M$ matrix inversion in (10) into a $b \times b$ matrix inversion, where b can be chosen independently on the network size.

III. Recursions for Incremental Training

Let us assume that a linear-in-the-parameter model has been trained in the form

$$\boldsymbol{Y} \approx \boldsymbol{A}\boldsymbol{W} \tag{22}$$

where the notation $\approx$ is used to indicate that $\boldsymbol{A}\boldsymbol{W}$ is an approximation of $\boldsymbol{Y}$ after training. Let us now assume that the incremental learning phase involves augmenting the model (22) with some extra regressor elements, thus resulting in the incremental learning model

$$\boldsymbol{Y} \approx \boldsymbol{A}\boldsymbol{W} + \boldsymbol{A}_{\text{incr}}\boldsymbol{W}_{\text{incr}} \tag{23}$$

where $\boldsymbol{A}_{\text{incr}}$ refers to the incremental regressor, and $\boldsymbol{W}_{\text{incr}}$ refers to the new weights to be trained. That is, we want to keep the same $\boldsymbol{W}$ and find only $\boldsymbol{W}_{\text{incr}}$. One idea for performing such an incremental training is to rearrange (23) as

$$\underbrace{\boldsymbol{Y} - \boldsymbol{A}\boldsymbol{W}}_{\tilde{\boldsymbol{Y}}} \approx \boldsymbol{A}_{\text{incr}}\boldsymbol{W}_{\text{incr}}. \tag{24}$$

At the beginning of the incremental learning phase, $\bar{\boldsymbol{Y}}$ is perfectly known since the labels data $\boldsymbol{Y}$ are known and $\boldsymbol{AW}$ was obtained in the previous learning phase. The incremental learning can, thus, be performed by applying the recursive algorithm (11) to the linear-in-the-parameter model

$$\hat{\bar{\boldsymbol{Y}}} = \boldsymbol{A}_{\text{incr}} \boldsymbol{W}_{\text{incr}} \tag{25}$$

where $\hat{\bar{\boldsymbol{Y}}}$ approximates $\bar{\boldsymbol{Y}}$ in (24). Let us now discuss how to transfer such concepts into the training process of BLS.

A. Case 1: Additional Enhancement Nodes

Consider the following incremental model to be trained:

$$\boldsymbol{Y} \approx \left[\boldsymbol{Z}^n,\ \boldsymbol{H}^m\right] \begin{bmatrix} \boldsymbol{W}_f^n \\ \boldsymbol{W}_c^m \end{bmatrix} + \boldsymbol{H}^{m,\bar{m}} \boldsymbol{W}_c^{m,\bar{m}} \tag{26}$$

where $\boldsymbol{H}^{m,\bar{m}} = [\boldsymbol{H}_{m+1}, \boldsymbol{H}_{m+2}, \ldots, \boldsymbol{H}_{m+\bar{m}}]$ represents the additional (incremental) enhancement nodes

$$\boldsymbol{H}_j = \zeta_j\left(\boldsymbol{Z}^n \boldsymbol{W}_{h_j} + \boldsymbol{\beta}_{h_j}\right), \quad j = m+1, \ldots, m+\bar{m} \tag{27}$$

and $\boldsymbol{W}_c^{m,\bar{m}}$ are the new weights to be trained (without retraining $\boldsymbol{W}_f^n$ and $\boldsymbol{W}_c^m$). Then, (26) can be rearranged as

$$\underbrace{\boldsymbol{Y} - \left[\boldsymbol{Z}^n,\ \boldsymbol{H}^m\right] \begin{bmatrix} \boldsymbol{W}_f^n \\ \boldsymbol{W}_c^m \end{bmatrix}}_{\tilde{\boldsymbol{Y}}} \approx \boldsymbol{H}^{m,\bar{m}} \boldsymbol{W}_c^{m,\bar{m}} \tag{28}$$

and attain incremental training after applying the recursive least-squares to (28). The details of the incremental algorithm can be found in Algorithm 1.

Remark 4: In the proposed formulation, $M \times M$ inversion is avoided both in the first and the incremental learning phases. In the original BLS formulation, both the first and the incremental phases require inversion of large matrices. The first phase of the standard BLS requires to invert the $M \times M$ matrix in

$$\boldsymbol{A} = \left[\boldsymbol{Z}^n,\ \boldsymbol{H}^m\right] \quad \boldsymbol{W} = \begin{bmatrix} \boldsymbol{W}_f^n \\ \boldsymbol{W}_c^m \end{bmatrix}$$

$$\boldsymbol{W} = \left(\boldsymbol{A}^T \boldsymbol{A} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{A}^T \boldsymbol{Y} \tag{29}$$

while the incremental phase of the standard BLS requires to solve the following problem:

$$([\boldsymbol{A},\ \boldsymbol{A}_{\text{incr}}])^+ = \begin{bmatrix} \boldsymbol{A}^+ - \boldsymbol{\Xi}\boldsymbol{\Psi}^T \\ \boldsymbol{\Psi}^T \end{bmatrix} \tag{30}$$

with $\boldsymbol{\Xi} = \boldsymbol{A}^+ \boldsymbol{H}^{m,\bar{m}}$ and

$$\boldsymbol{\Psi}^T = \begin{cases} \boldsymbol{\Gamma}^+, & \text{if } \boldsymbol{\Gamma} \neq 0 \\ \left(\boldsymbol{I} + \boldsymbol{\Xi}^T \boldsymbol{\Xi}\right)^{-1} \boldsymbol{\Xi}^T \boldsymbol{A}^+, & \text{if } \boldsymbol{\Gamma} = 0 \end{cases} \tag{31}$$

$$\boldsymbol{\Gamma} = \boldsymbol{H}^{m,\bar{m}} - \boldsymbol{A}\boldsymbol{\Xi}. \tag{32}$$

The standard BLS does not solve (30) explicitly, but it shows that the incremental weights $\boldsymbol{W}_{\text{incr}}$ can be calculated as

$$\begin{bmatrix} \boldsymbol{W} \\ \boldsymbol{W}_{\text{incr}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{W} - \boldsymbol{\Xi}\boldsymbol{\Psi}^T \boldsymbol{Y} \\ \boldsymbol{\Psi}^T \boldsymbol{Y} \end{bmatrix}. \tag{33}$$

Even though (30) is not solved explicitly, it can be seen that the incremental phase of the standard BLS still involves a new matrix inverse ($(\boldsymbol{I} + \boldsymbol{\Xi}^T \boldsymbol{\Xi})^{-1}$ or $(\boldsymbol{I} + \boldsymbol{\Gamma}^T \boldsymbol{\Gamma})^{-1}$), while

Algorithm 1 HR-BLS With Additional Enhancement Nodes

INPUT
Training data $\boldsymbol{X}(k)$ and labels $\boldsymbol{Y}(k)$, $k = 1, \ldots, N$
Testing data $\boldsymbol{X}_t(k)$ and labels $\boldsymbol{Y}_t(k)$, $k = 1, \ldots, N_t$
INITIALIZATION
$\boldsymbol{P}_0 = q_0 \boldsymbol{I}$, $\boldsymbol{W}_0 = 0$
design parameters λ, n, m, $\bar{m}$, b
FIRST TRAINING PHASE
for $i = 1, \ldots, n$
Randomly initialize $\boldsymbol{W}_{e_i}$, $\boldsymbol{\beta}_{e_i}$
Compute $\boldsymbol{Z}_i = \varphi_i(\boldsymbol{X}\boldsymbol{W}_{e_i} + \boldsymbol{\beta}_{e_i})$
Collect the feature nodes $\boldsymbol{Z}^n = [\boldsymbol{Z}_1, \boldsymbol{Z}_2, \ldots, \boldsymbol{Z}_n]$
for $j = 1, \ldots, m$
Randomly initialize $\boldsymbol{W}_{h_j}$, $\boldsymbol{\beta}_{h_j}$
Compute $\boldsymbol{H}_j = \zeta_j(\boldsymbol{Z}^n \boldsymbol{W}_{h_j} + \boldsymbol{\beta}_{h_j})$
Collect the enhanc. nodes $\boldsymbol{H}^m = [\boldsymbol{H}_1, \boldsymbol{H}_2, \ldots, \boldsymbol{H}_m]$
for $k = 1, \ldots, N$
Perform the hybrid recursive least-squares

$$\boldsymbol{Y} \approx \left[\boldsymbol{Z}^n,\ \boldsymbol{H}^m\right] \begin{bmatrix} \boldsymbol{W}_f^n \\ \boldsymbol{W}_c^m \end{bmatrix}$$

OUTPUT
Obtain the weights $\boldsymbol{W}_f^n$, $\boldsymbol{W}_c^m$
TESTING
for $i = 1, \ldots, n$
Compute $\boldsymbol{Z}_{t,i} = \varphi_i(\boldsymbol{X}_t \boldsymbol{W}_{e_i} + \boldsymbol{\beta}_{e_i})$
Collect the feature nodes $\boldsymbol{Z}_t^n = [\boldsymbol{Z}_{t,1}, \boldsymbol{Z}_{t,2}, \ldots, \boldsymbol{Z}_{t,n}]$
for $j = 1, \ldots, m$
Compute $\boldsymbol{H}_{t,j} = \zeta_j(\boldsymbol{Z}_t^n \boldsymbol{W}_{h_j} + \boldsymbol{\beta}_{h_j})$
Collect the enhancement nodes
$\boldsymbol{H}_t^m = [\boldsymbol{H}_{t,1}, \boldsymbol{H}_{t,2}, \ldots, \boldsymbol{H}_{t,m}]$
Calculate the testing error from

$$\hat{\boldsymbol{Y}}_t = \left[\boldsymbol{Z}_t^n,\ \boldsymbol{H}_t^m\right] \begin{bmatrix} \boldsymbol{W}_f^n \\ \boldsymbol{W}_c^m \end{bmatrix}$$

INCREMENTAL TRAINING PHASE
for $j = m+1, \ldots, m+\bar{m}$
Randomly initialize $\boldsymbol{W}_{h_j}$, β_{h_j}
Compute $\boldsymbol{H}_j = \zeta_j(\boldsymbol{Z}^n \boldsymbol{W}_{h_j} + \boldsymbol{\beta}_{h_j})$
Collect the enhancement nodes
$\boldsymbol{H}^{m,\bar{m}} = [\boldsymbol{H}_{m+1}, \boldsymbol{H}_{m+2}, \ldots, \boldsymbol{H}_{m+\bar{m}}]$
for $k = 1, \ldots, N$
Perform the hybrid recursive least-squares

$$\boldsymbol{Y} - \left[\boldsymbol{Z}^n,\ \boldsymbol{H}^m\right] \begin{bmatrix} \boldsymbol{W}_f^n \\ \boldsymbol{W}_c^m \end{bmatrix} \approx \boldsymbol{H}^{m,\bar{m}} \boldsymbol{W}_c^{m,\bar{m}}$$

INCREMENTAL OUTPUT
Obtain the weights $\boldsymbol{W}_c^{m,\bar{m}}$
INCREMENTAL TESTING
for $j = m+1, \ldots, m+\bar{m}$
Compute $\boldsymbol{H}_{t,j} = \zeta_j(\boldsymbol{Z}_t^n \boldsymbol{W}_{h_j} + \bar{\boldsymbol{\beta}}_{h_j})$
Collect enhancement nodes
$\boldsymbol{H}_t^{m,\bar{m}} = [\boldsymbol{H}_{t,m+1}, \boldsymbol{H}_{t,m+2}, \ldots, \boldsymbol{H}_{t,m+\bar{m}}]$
Calculate the incremental testing error from

$$\hat{\boldsymbol{Y}}_t = \left[\boldsymbol{Z}_t^n,\ \boldsymbol{H}_t^m,\ \boldsymbol{H}_t^{m,\bar{m}}\right] \begin{bmatrix} \boldsymbol{W}_f^n \\ \boldsymbol{W}_c^m \\ \boldsymbol{W}_c^{m,\bar{m}} \end{bmatrix}$$

$\boldsymbol{A}^+$ should be kept in the memory (or recalculated). In the proposed recursive Algorithm 1, no large matrices must be inverted, and there is no need for storing past matrices other than the current weights $\boldsymbol{W}_f^n$ and $\boldsymbol{W}_c^m$. We will see that this more efficient usage of memory significantly speeds up the incremental phase compared with the standard BLS.

B. Case 2: Additional Feature Mappings

Consider the following incremental model to be trained:

$$\boldsymbol{Y} \approx \left[\boldsymbol{Z}^n,\, \boldsymbol{H}^m\right]\begin{bmatrix}\boldsymbol{W}_f^n\\ \boldsymbol{W}_c^m\end{bmatrix} + \left[\boldsymbol{Z}^{n,\bar{n}},\, \boldsymbol{H}^{m,\bar{m}}\right]\begin{bmatrix}\boldsymbol{W}_f^{n,\bar{n}}\\ \boldsymbol{W}_c^{m,\bar{m}}\end{bmatrix} \tag{34}$$

where $\boldsymbol{Z}^{n,\bar{n}} = [\boldsymbol{Z}_{n+1}, \boldsymbol{Z}_{n+2}, \ldots, \boldsymbol{Z}_{n+\bar{n}}]$ are the additional (incremental) feature mapping

$$\boldsymbol{Z}_i = \varphi_i\left(\boldsymbol{X}\boldsymbol{W}_{e_i} + \boldsymbol{\beta}_{e_i}\right), \quad i = n+1, \ldots, n+\bar{n} \tag{35}$$

and $\boldsymbol{H}^{m,\bar{m}} = [\boldsymbol{H}_{m+1}, \boldsymbol{H}_{m+2}, \ldots, \boldsymbol{H}_{m+\bar{m}}]$ are the additional (incremental) enhancement nodes

$$\boldsymbol{H}_j = \zeta_j\left(\boldsymbol{Z}^{n,\bar{n}}\boldsymbol{W}_{h_j} + \boldsymbol{\beta}_{h_j}\right), \quad j = m+1, \ldots, m+\bar{m}. \tag{36}$$

Correspondingly, $\boldsymbol{W}_f^{n,\bar{n}}$ and $\boldsymbol{W}_c^{m,\bar{m}}$ are the new feature and enhancement weights to be trained (without retraining $\boldsymbol{W}_f^n$ and $\boldsymbol{W}_c^m$). Then, one can rearrange the model (34) as

$$\underbrace{\boldsymbol{Y} - \left[\boldsymbol{Z}^n,\, \boldsymbol{H}^m\right]\begin{bmatrix}\boldsymbol{W}_f^n\\ \boldsymbol{W}_c^m\end{bmatrix}}_{\bar{\boldsymbol{Y}}} \approx \left[\boldsymbol{Z}^{n,\bar{n}},\, \boldsymbol{H}^{m,\bar{m}}\right]\begin{bmatrix}\boldsymbol{W}_f^{n,\bar{n}}\\ \boldsymbol{W}_c^{m,\bar{m}}\end{bmatrix} \tag{37}$$

and attain incremental training after applying the recursive least-squares to (37). The details of the incremental algorithm can be found in Algorithm 2.

C. Case 3: Additional Labeled Data

New labeled data might arise whenever a new sensor is placed in the system. This case can be divided into two subcases. In the first subcase, one requires the network structure to "evolve" with the data, requiring new feature/enhancement nodes to be trained when more and more data arrive. In the second subcase, one may not desire new feature/enhancement nodes, but still the new labels require to train the extra weights of the output layer. For the first subcase, consider the following incremental model:

$$[\boldsymbol{Y},\, \boldsymbol{Y}_x] \approx \left[\left[\boldsymbol{Z}^n,\, \boldsymbol{H}^m\right]\begin{bmatrix}\boldsymbol{W}_f^n\\ \boldsymbol{W}_c^m\end{bmatrix} + \left[\boldsymbol{Z}_x^{n,\bar{n}},\, \boldsymbol{H}_x^{m,\bar{m}}\right]\begin{bmatrix}\boldsymbol{W}_f^{n,\bar{n}}\\ \boldsymbol{W}_c^{m,\bar{m}}\end{bmatrix},\right.$$
$$\left.\left[\boldsymbol{Z}_x^{n,\bar{n}},\, \boldsymbol{H}_x^{m,\bar{m}}\right]\begin{bmatrix}\boldsymbol{W}_{f_x}^{n,\bar{n}}\\ \boldsymbol{W}_{c_x}^{m,\bar{m}}\end{bmatrix}\right] \tag{38}$$

where $\boldsymbol{Z}_x^{n,\bar{n}} = [\boldsymbol{Z}_{n+1}, \boldsymbol{Z}_{n+2}, \ldots, \boldsymbol{Z}_{n+\bar{n}}]$ are the additional (incremental) feature mapping

$$\boldsymbol{Z}_i = \varphi_i\left(\boldsymbol{X}_x\boldsymbol{W}_{e_i} + \boldsymbol{\beta}_{e_i}\right), \quad i = n+1, \ldots, n+\bar{n}. \tag{39}$$

$\boldsymbol{Y}_x$ are new labels, $\boldsymbol{X}_x$ are new data, and $\boldsymbol{H}_x^{m,\bar{m}} = [\boldsymbol{H}_{m+1}, \boldsymbol{H}_{m+2}, \ldots, \boldsymbol{H}_{m+\bar{m}}]$ are the additional (incremental) enhancement nodes

$$\boldsymbol{H}_j = \zeta_j\left(\boldsymbol{Z}_x^{n,\bar{n}}\boldsymbol{W}_{h_j} + \boldsymbol{\beta}_{h_j}\right), \quad j = m+1, \ldots, m+\bar{m}. \tag{40}$$

Algorithm 2 HR-BLS With Additional Feature Nodes

INPUT
Training data $\boldsymbol{X}(k)$ and labels $\boldsymbol{Y}(k)$, $k = 1, \ldots, N$
Testing data $\boldsymbol{X}_t(k)$ and labels $\boldsymbol{Y}_t(k)$, $k = 1, \ldots, N_t$
INITIALIZATION
$\boldsymbol{P}_0 = q_0\boldsymbol{I}$, $\boldsymbol{W}_0 = 0$
design parameters λ, n, m, $\bar{n}$, $\bar{m}$, b
FIRST TRAINING PHASE
Same as Algorithm 1
OUTPUT
Obtain the weights $\boldsymbol{W}_f^n$, $\boldsymbol{W}_c^m$
TESTING
Same as Algorithm 1
Calculate the testing error from

$$\hat{\boldsymbol{Y}}_t = \left[\boldsymbol{Z}_t^n,\, \boldsymbol{H}_t^m\right]\begin{bmatrix}\boldsymbol{W}_f^n\\ \boldsymbol{W}_c^m\end{bmatrix}$$

INCREMENTAL TRAINING PHASE
for $i = n+1, \ldots, n+\bar{n}$
Randomly initialize $\boldsymbol{W}_{e_i}$, $\boldsymbol{\beta}_{e_i}$
Compute $\boldsymbol{Z}_i = \varphi_i(\boldsymbol{X}\boldsymbol{W}_{e_i} + \boldsymbol{\beta}_{e_i})$
Collect the features $\boldsymbol{Z}^{n,\bar{n}} = [\boldsymbol{Z}_{n+1}, \boldsymbol{Z}_{n+2}, \ldots, \boldsymbol{Z}_{n+\bar{n}}]$
for $j = m+1, \ldots, m+\bar{m}$
Randomly initialize $\boldsymbol{W}_{h_j}$, $\boldsymbol{\beta}_{h_j}$
Compute $\boldsymbol{H}_j = \zeta_j(\boldsymbol{Z}^{n,\bar{n}}\boldsymbol{W}_{h_j} + \boldsymbol{\beta}_{h_j})$
Collect the enhancement nodes
$\boldsymbol{H}^{m,\bar{m}} = [\boldsymbol{H}_{m+1}, \boldsymbol{H}_{m+2}, \ldots, \boldsymbol{H}_{m+\bar{m}}]$
for $k = 1, \ldots, N$
Perform the hybrid recursive least-squares

$$\boldsymbol{Y} - \left[\boldsymbol{Z}^n,\, \boldsymbol{H}^m\right]\begin{bmatrix}\boldsymbol{W}_f^n\\ \boldsymbol{W}_c^m\end{bmatrix} \approx \left[\boldsymbol{Z}^{n,\bar{n}},\, \boldsymbol{H}^{m,\bar{m}}\right]\begin{bmatrix}\boldsymbol{W}_f^{n,\bar{n}}\\ \boldsymbol{W}_c^{m,\bar{m}}\end{bmatrix}$$

INCREMENTAL OUTPUT
Obtain the weights $\boldsymbol{W}_f^{n,\bar{n}}$, $\boldsymbol{W}_c^{m,\bar{m}}$
INCREMENTAL TESTING
for $i = n+1, \ldots, n+\bar{n}$
Compute $\boldsymbol{Z}_i = \varphi_i(\boldsymbol{X}_t\boldsymbol{W}_{e_i} + \boldsymbol{\beta}_{e_i})$
Collect the feature nodes
$\boldsymbol{Z}_t^{n,\bar{n}} = [\boldsymbol{Z}_{t,n+1}, \boldsymbol{Z}_{t,n+2}, \ldots, \boldsymbol{Z}_{t,n+\bar{n}}]$
for $j = m+1, \ldots, m+\bar{m}$
Compute $\boldsymbol{H}_{t,j} = \zeta_j(\boldsymbol{Z}_t^{n,\bar{n}}\boldsymbol{W}_{h_j} + \boldsymbol{\beta}_{h_j})$
Collect the enhancement nodes
$\boldsymbol{H}_t^{m,\bar{m}} = [\boldsymbol{H}_{t,m+1}, \boldsymbol{H}_{t,m+2}, \ldots, \boldsymbol{H}_{t,m+\bar{m}}]$
Calculate the incremental testing error from

$$\hat{\boldsymbol{Y}}_t = \left[\boldsymbol{Z}_t^n,\, \boldsymbol{H}_t^m,\, \boldsymbol{Z}_t^{n,\bar{n}},\, \boldsymbol{H}_t^{m,\bar{m}}\right]\begin{bmatrix}\boldsymbol{W}_f^n\\ \boldsymbol{W}_c^m\\ \boldsymbol{W}_f^{n,\bar{n}}\\ \boldsymbol{W}_c^{m,\bar{m}}\end{bmatrix}$$

Correspondingly, $\boldsymbol{W}_f^{n,\bar{n}}$, $\boldsymbol{W}_c^{m,\bar{m}}$, $\boldsymbol{W}_{f_x}^{n,\bar{n}}$, and $\boldsymbol{W}_{c_x}^{m,\bar{m}}$ are the new feature and enhancement weights to be trained (without retraining $\boldsymbol{W}_f^n$ and $\boldsymbol{W}_c^m$). Then, (38) can be

rearranged as

$$\underbrace{\left[\boldsymbol{Y}-\left[\boldsymbol{Z}^n,\,\boldsymbol{H}^m\right]\begin{bmatrix}\boldsymbol{W}_f^n\\ \boldsymbol{W}_c^m\end{bmatrix},\,\boldsymbol{Y}_x\right]}_{\check{\boldsymbol{Y}}} \approx \left[\boldsymbol{Z}_x^{n,\bar{n}},\,\boldsymbol{H}_x^{m,\bar{m}}\right]\begin{bmatrix}\boldsymbol{W}_f^{n,\bar{n}} & \boldsymbol{W}_{f_x}^{n,\bar{n}}\\ \boldsymbol{W}_c^{m,\bar{m}} & \boldsymbol{W}_{c_x}^{m,\bar{m}}\end{bmatrix} \quad (41)$$

and attain incremental training by applying the recursive least-squares to (41). In the second subcase when no new nodes are desired, an incremental model can be written as

$$[\boldsymbol{Y},\,\boldsymbol{Y}_x] \approx \left[\left[\boldsymbol{Z}^n,\,\boldsymbol{H}^m\right]\begin{bmatrix}\boldsymbol{W}_f^n\\ \boldsymbol{W}_c^m\end{bmatrix}\left[\boldsymbol{Z}^n,\,\boldsymbol{H}^m\right]\begin{bmatrix}\boldsymbol{W}_{f_x}^{n,\bar{n}}\\ \boldsymbol{W}_{c_x}^{m,\bar{m}}\end{bmatrix}\right] \quad (42)$$

in which the new labeled data $\boldsymbol{Y}_x$ that were not available before still require to retrain part of the output layer (without modifying the network structure). The details of the incremental algorithm can be found in Algorithms 3 (if new feature/enhancement nodes are desired) and 4 (if no new feature/enhancement nodes are desired).

Some remarks are given in the following.

Remark 5: Algorithms 3 and 4 deals with new labeled data being available: if new data are made available (both input labels $\boldsymbol{X}_x$ and labels $\boldsymbol{Y}_x$), one can treat them straightforwardly due to the recursive nature of the algorithm. Different from the standard BLS algorithm (including the incremental versions), as in [29], the proposed algorithm avoids the inversion of large matrices at all stages. For example, the incremental learning for the increment of input data in [29] requires the following calculations:

$$(\boldsymbol{A}_{\text{incr}})^+ = \left[\boldsymbol{A}^+ - \boldsymbol{\Psi}\boldsymbol{\Xi}^T,\ \boldsymbol{\Psi}\right] \quad (43)$$

with $\boldsymbol{\Xi}^T = (\boldsymbol{A_x}^T)\boldsymbol{A}^+$, $\boldsymbol{A_x}$ being the increment of mapped feature nodes and the enhancement nodes, and

$$\boldsymbol{\Psi} = \begin{cases}\left(\boldsymbol{\Gamma}^T\right)^+, & \text{if } \boldsymbol{\Gamma}\neq 0\\ \left(\boldsymbol{I}+\boldsymbol{\Xi}^T\boldsymbol{\Xi}\right)^{-1}\boldsymbol{A}^+\boldsymbol{\Xi}, & \text{if } \boldsymbol{\Gamma}=0\end{cases} \quad (44)$$

$$\boldsymbol{\Gamma}^T = \boldsymbol{A_x}^T - \boldsymbol{\Xi}^T\boldsymbol{A}. \quad (45)$$

Therefore, similar to what discussed in Remark 4, the incremental phase of the standard BLS still involves a new matrix inverse, while $\boldsymbol{A}^+$ should be kept in the memory (or recalculated). In the proposed recursive algorithms, no large matrices must be inverted, and there is no need for storing past matrices other than the current weights. This marks the major difference between the state-of-the-art "incremental learning" and the proposed "recursive learning." In this sense, all state-of-the-art BLS algorithms in [29], [34], [37], and [38] are incremental but not recursive.

Remark 6: In view of Propositions 1 and 2, all the proposed HR-BLS Algorithms 1–4 have the same learning accuracy of the standard BLS, being their network weights $\boldsymbol{W}$ identical and solving the same optimization problem (9). Crucially, Algorithms 1–4 differ with the standard BLS with respect to the tradeoffs associated with the calculation of the weights. It is also important to remark that any add-on proposed in

Algorithm 3 HR-BLS With Additional Labels and New Nodes

INPUT
Training data $\boldsymbol{X}(k)$ and labels $\boldsymbol{Y}(k)$, $k=1,\ldots,N$
Testing data $\boldsymbol{X}_t(k)$ and labels $\boldsymbol{Y}_t(k)$, $k=1,\ldots,N_t$
New training data $\boldsymbol{X}_x(k)$ and $\boldsymbol{Y}_x(k)$, $k=1,\ldots,\bar{N}$
New testing data $\boldsymbol{X}_{x,t}(k)$ and $\boldsymbol{Y}_{x,t}(k)$, $k=1,\ldots,\bar{N}_t$
INITIALIZATION
$\boldsymbol{P}_0 = q_0\boldsymbol{I}$, $\boldsymbol{W}_0 = 0$
design parameters λ, n, m, $\bar{n}$, $\bar{m}$, b
FIRST TRAINING PHASE
Same as Algorithm 1
OUTPUT
Obtain the weights $\boldsymbol{W}_f^n$, $\boldsymbol{W}_c^m$
TESTING
Same as Algorithm 1
Calculate the testing error from

$$\hat{\boldsymbol{Y}}_t = \left[\boldsymbol{Z}_t^n,\,\boldsymbol{H}_t^m\right]\begin{bmatrix}\boldsymbol{W}_f^n\\ \boldsymbol{W}_c^m\end{bmatrix}$$

INCREMENTAL TRAINING PHASE
for $i=n+1,\ldots,n+\bar{n}$
Randomly initialize $\boldsymbol{W}_{ex_i}$, $\boldsymbol{\beta}_{ex_i}$
Compute $\boldsymbol{Z}_i = \varphi_i(\boldsymbol{X}_x\boldsymbol{W}_{ex_i} + \boldsymbol{\beta}_{ex_i})$
Collect the feature nodes
$\boldsymbol{Z}_x^{n,\bar{n}} = [\boldsymbol{Z}_{n+1}, \boldsymbol{Z}_{n+2}, \ldots, \boldsymbol{Z}_{n+\bar{n}}]$
for $j=m+1,\ldots,m+\bar{m}$
Randomly initialize $\boldsymbol{W}_{hx_j}$, $\boldsymbol{\beta}_{hx_j}$
Compute $\boldsymbol{H}_j = \zeta_j(\boldsymbol{Z}_x^{n,\bar{n}}\boldsymbol{W}_{h_j} + \boldsymbol{\beta}_{h_j})$
Collect the enhancement nodes
$\boldsymbol{H}_x^{m,\bar{m}} = [\boldsymbol{H}_{m+1}, \boldsymbol{H}_{m+2}, \ldots, \boldsymbol{H}_{m+\bar{m}}]$
for $k=1,\ldots,N$
Perform the hybrid recursive least-squares on

$$\left[\boldsymbol{Y}-\left[\boldsymbol{Z}^n,\,\boldsymbol{H}^m\right]\begin{bmatrix}\boldsymbol{W}_f^n\\ \boldsymbol{W}_c^m\end{bmatrix},\,\boldsymbol{Y}_x\right] \approx \left[\boldsymbol{Z}_x^{n,\bar{n}},\,\boldsymbol{H}_x^{m,\bar{m}}\right]\begin{bmatrix}\boldsymbol{W}_f^{n,\bar{n}} & \boldsymbol{W}_{f_x}^{n,\bar{n}}\\ \boldsymbol{W}_c^{m,\bar{m}} & \boldsymbol{W}_{c_x}^{m,\bar{m}}\end{bmatrix}$$

INCREMENTAL OUTPUT
Obtain the weights $\boldsymbol{W}_f^{n,\bar{n}}$, $\boldsymbol{W}_c^{m,\bar{m}}$, $\boldsymbol{W}_{f_x}^{n,\bar{n}}$, $\boldsymbol{W}_{c_x}^{m,\bar{m}}$
INCREMENTAL TESTING
for $i=n+1,\ldots,n+\bar{n}$
Compute $\boldsymbol{Z}_i = \varphi_i(\boldsymbol{X}_{x_t}\boldsymbol{W}_{e_i} + \boldsymbol{\beta}_{e_i})$
Collect the feature nodes
$\boldsymbol{Z}_{x,t}^{n,\bar{n}} = [\boldsymbol{Z}_{t,n+1}, \boldsymbol{Z}_{t,n+2}, \ldots, \boldsymbol{Z}_{t,n+\bar{n}}]$
for $j=m+1,\ldots,m+\bar{m}$
Compute $\boldsymbol{H}_{t,j} = \zeta_j(\boldsymbol{Z}_{x,t}^n\boldsymbol{W}_{h_j} + \boldsymbol{\beta}_{h_j})$
Collect the enhancement nodes
$\boldsymbol{H}_{x,t}^{m,\bar{m}} = [\boldsymbol{H}_{t,m+1}, \boldsymbol{H}_{t,m+2}, \ldots, \boldsymbol{H}_{t,m+\bar{m}}]$
Calculate the incremental testing error from

$$\left[\hat{\boldsymbol{Y}}_t,\,\hat{\boldsymbol{Y}}_{x,t}\right] = \left[\left[\boldsymbol{Z}_t^n,\,\boldsymbol{H}_t^m\right]\begin{bmatrix}\boldsymbol{W}_f^n\\ \boldsymbol{W}_c^m\end{bmatrix} + \left[\boldsymbol{Z}_{x,t}^{n,\bar{n}},\,\boldsymbol{H}_{x,t}^{m,\bar{m}}\right] \times \begin{bmatrix}\boldsymbol{W}_f^{n,\bar{n}}\\ \boldsymbol{W}_c^{m,\bar{m}}\end{bmatrix}, \left[\boldsymbol{Z}_{x,t}^{n,\bar{n}},\,\boldsymbol{H}_{x,t}^{m,\bar{m}}\right]\begin{bmatrix}\boldsymbol{W}_{f_x}^{n,\bar{n}}\\ \boldsymbol{W}_{c_x}^{m,\bar{m}}\end{bmatrix}\right]$$

Algorithm 4 HR-BLS With Additional Labels, No New Nodes

1: **INPUT**
Training data $\boldsymbol{X}(k)$ and labels $\boldsymbol{Y}(k)$, $k = 1, \ldots, N$
Testing data $\boldsymbol{X}_t(k)$ and labels $\boldsymbol{Y}_t(k)$, $k = 1, \ldots, N_t$
New training data $\boldsymbol{X}_x(k)$ and $\boldsymbol{Y}_x(k)$, $k = 1, \ldots, \bar{N}$
New testing data $\boldsymbol{X}_{x,t}(k)$ and $\boldsymbol{Y}_{x,t}(k)$, $k = 1, \ldots, \bar{N}_t$
2: **INITIALIZATION**
$\boldsymbol{P}_0 = q_0\boldsymbol{I}$, $\boldsymbol{W}_0 = 0$
design parameters λ, n, m, $\bar{n}$, $\bar{m}$, b
3: **FIRST TRAINING PHASE**
Same as Algorithm 1
4: **OUTPUT**
Obtain the weights $\boldsymbol{W}_f^n$, $\boldsymbol{W}_c^m$
5: **TESTING**
Same as Algorithm 1
6: **Calculate** the testing error from

$$\hat{\boldsymbol{Y}}_t = \left[\boldsymbol{Z}_t^n,\ \boldsymbol{H}_t^m\right]\begin{bmatrix}\boldsymbol{W}_f^n\\ \boldsymbol{W}_c^m\end{bmatrix}$$

7: **INCREMENTAL TRAINING PHASE**
8: **for** $k = 1, \ldots, N$
Perform the hybrid recursive least-squares

$$\boldsymbol{Y}_x \approx \left[\boldsymbol{Z}_x^n,\ \boldsymbol{H}_x^m\right]\begin{bmatrix}\boldsymbol{W}_{f_x}^{n,\bar{n}}\\ \boldsymbol{W}_{c_x}^{m,\bar{m}}\end{bmatrix}$$

9: **INCREMENTAL OUTPUT**
Obtain the weights $\boldsymbol{W}_f^{n,\bar{n}}$, $\boldsymbol{W}_c^{m,\bar{m}}$, $\boldsymbol{W}_{f_x}^{n,\bar{n}}$, $\boldsymbol{W}_{c_x}^{m,\bar{m}}$
10: **INCREMENTAL TESTING**
11: **Calculate** the incremental testing error from

$$\left[\hat{\boldsymbol{Y}}_t,\ \hat{\boldsymbol{Y}}_{x,t}\right] = \left[\boldsymbol{Z}_t^n,\ \boldsymbol{H}_t^m\right]\begin{bmatrix}\boldsymbol{W}_f^n & \boldsymbol{W}_{f_x}^{n,\bar{n}}\\ \boldsymbol{W}_c^m & \boldsymbol{W}_{c_x}^{m,\bar{m}}\end{bmatrix}$$

the BLS literature, such as sparse encoder regularization or SVD simplification [29], can be transferred to the proposed formulation in a straightforward manner. The same applies to the BLS variants proposed in the literature, in terms of broad-deep (recurrent, neurofuzzy, and convolutional) combined structures [34]–[37].

IV. Comparative Experiments

The performance measurement system (PeMS) is used as a benchmark data set for the proposed framework. PeMS is one of the most popular data sets in the traffic field, managed and updated by the California Department of Transportation (Caltrans), with data collected in real-time from nearly 40 000 individual detectors spanning the freeway system across all major metropolitan areas of California. The data set under consideration in this work is consistent with the one in [44]. It contains the flow, speed, and occupancy data at 33 different locations in *I*405 freeway (North-bound Interstate 405), resulting in $R = 99$. The data are aggregated every 5 min, resulting in 120 000 data samples. The data set is more than ten times larger than the MNIST and NYU-NORB data sets used for benchmarking BLS in [29], and this is the main reason why it is used here to test the efficiency of the proposed framework in handling huge numbers of data.

The algorithms will be abbreviated as BLS (standard BLS), HR-BLS, BLS enh. (standard BLS with added enhancement nodes), HR-BLS enh. (hybrid-recursive BLS with added enhancement nodes), BLS feat. (standard BLS with added feature mappings), and HR-BLS feat. (hybrid-recursive BLS with added feature mappings). The following parameters are used in the experiments: for BLS and HR-BLS (without incremental steps), $n = 1\,600$ and $m = 3\,250$; for BLS and HR-BLS with incremental enhancement nodes, $n = 1\,600$ and $m = 500$ (first phase), with $\bar{m} = 500$ new enhancement nodes in four incremental steps; and for BLS and HR-BLS with incremental feature nodes, $n = 650$ and $m = 1000$ (first phase), with $\bar{n} = 50$ and $\bar{m} = 500$ new feature/enhancement nodes in four incremental steps. For all algorithms, we set the regularization factor $\lambda = 1/64$, and for HR-BLS, we set $q_0^{-1} = \lambda$. Unless specified otherwise, the size of the batch for HR-BLS is $b = 500$. The regression task is to predict the traffic (flow, speed, and occupancy) 15, 25, and 40 min ahead in time. The training samples are 50 000, and to test the generalization capabilities of the prediction, 70 000 data samples are used for testing. All tests are run on a desktop PC with Intel Core i5 (Quad-Core), 8-GB DDR4, and MATLAB R2017b. The reported data of computational time and memory usage are valid for a MATLAB implementation, and it is likely that different platforms (C++, Python, and so on) will give slightly different results: however, let us mention that, in order to make the comparison as fair as possible, MATLAB is restarted before every simulation, so as to start from clean memory: furthermore, the code is written in such a way to clear memory whenever a variable is not being used anymore, which prevents the garbage collection mechanism from growing too much.

A. Learning Performance

The results of the comparisons are provided in Table I (for 15-, 25-, and 40-min-ahead predictions, respectively). Some observations follow about accuracy and training time.

1) Testing Accuracy: The testing accuracies of BLS and HR-BLS are equivalent. In fact, it can be shown that they converge to almost identical network weights: Propositions 1 and 2 highlight that BLS and HR-BLS solve the same least-squares problem with different methods. Thus, let us further refer to the PeMS comparison study in [31] between BLS and other supervised learning algorithms (shallow and deep neural networks, stacked autoencoders, and so on). Such a comparison study automatically extends to HR-BLS.

2) Incremental Versions: As shown in the literature [29], the nonincremental versions of BLS exhibit better testing accuracy than its incremental version when the network structure of the former is richer than the structure of the latter. The incremental BLS and HR-BLS improve their accuracy after some incremental steps.

3) Training Time (First Phase): The first training phase of BLS is from two to four times faster than the first training phase of HR-BLS. This means that, for $N = 50\,000$ and $M = 5000$, solving the least-squares in one-shot is more efficient than solving the least-squares in a recursive way.

TABLE I

COMPARISON RESULTS FOR BLS AND HR-BLS. FOR INCREMENTAL LEARNING, BOTH THE FIRST (1ST) AND THE INCREMENTAL (INC.) PHASES ARE REPORTED. FOR THE INCREMENTAL PHASES, THE TRAINING TIME IS SUMMED OVER ALL FOUR PHASES, WHILE THE TESTING ACCURACY IS AT THE END OF THE FOURTH PHASE

15 min.	Training time (s)	Testing accuracy (%)
BLS	36.95	87.44
HR-BLS	151.89	87.44
BLS enh.	19.54 (1st) 260.55 (inc.)	87.14 (1st) 87.34 (inc.)
HR-BLS enh.	36.33 (1st) 70.16 (inc.)	87.14 (1st) 87.33 (inc.)
BLS feat.	13.38 (1st) 146.43 (inc.)	87.06 (1st) 87.22 (inc.)
HR-BLS feat.	27.73 (1st) 58.50 (inc.)	87.06 (1st) 87.19 (inc.)
25 min.	Training time (s)	Testing accuracy (%)
BLS	34.39	83.59
HR-BLS	137.93	83.59
BLS enh.	18.88 (1st) 333.93 (inc.)	83.28 (1st) 83.43 (inc.)
HR-BLS enh.	37.04 (1st) 148.40 (inc.)	83.28 (1st) 83.39 (inc.)
BLS feat.	13.50 (1st) 134.69 (inc.)	83.31 (1st) 83.38 (inc.)
HR-BLS feat.	27.32 (1st) 41.44 (inc.)	83.31 (1st) 83.33 (inc.)
40 min.	Training time (s)	Testing accuracy (%)
BLS	41.90	80.78
HR-BLS	174.78	80.78
BLS enh.	21.97 (1st) 283.31 (inc.)	80.32 (1st) 80.58 (inc.)
HR-BLS enh.	40.93 (1st) 59.07 (inc.)	80.32 (1st) 80.62 (inc.)
BLS feat.	14.38 (1st) 145.38 (inc.)	80.20 (1st) 80.55 (inc.)
HR-BLS feat.	27.69 (1st) 39.63 (inc.)	80.20 (1st) 80.44 (inc.)

4) Training Time (Incremental Phases): Here, HR-BLS becomes around three times faster than BLS. In fact, the incremental steps of BLS require to calculate/store two pseudoinverse matrices (see Remark 4). In HR-BLS, only the new matrix $\boldsymbol{P}$ associated with the new feature/enhancement node has to be memorized: the previous $\boldsymbol{P}$ can be deleted from the memory. This efficient usage of memory leads to faster incremental learning.

Despite that the training time results suggest using the standard BLS for the first learning phase and the HR-BLS for the incremental learning phase, we will now show that the standard BLS can exhibit out-of-memory problem when the data/network size is too large.

B. Tradeoff Between Recursions and Memory Usage

Fig. 2 shows the training time of HR-BLS as a function of the size of the batch: the size of the batch goes from $b = 1$ to $b = 5000$. The figure highlights two points: for small sizes of the batch, the recursive least-squares algorithm has to run over many iterations, which contributes to high training time; on the other side, when the size of the batch is very large, the training time tends to increase due to the computational cost of inverting large matrices. For intermediate sizes of the batch, a good tradeoff can be found in terms of training time. For example, the training time for $b \approx 500$ is 30 times faster than for $b = 1$ (6000 s versus 200 s).

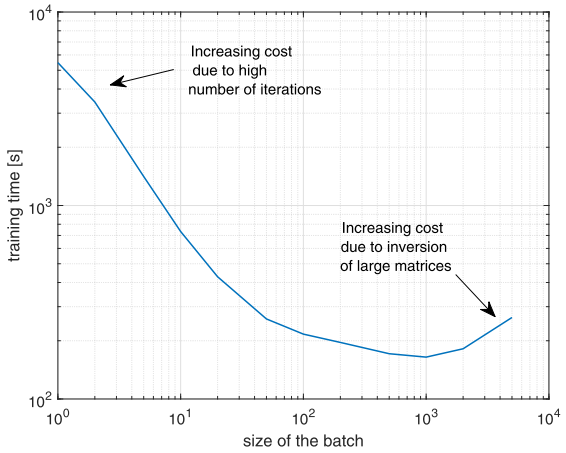


Fig. 2. Training time as a function of the size of the batch.

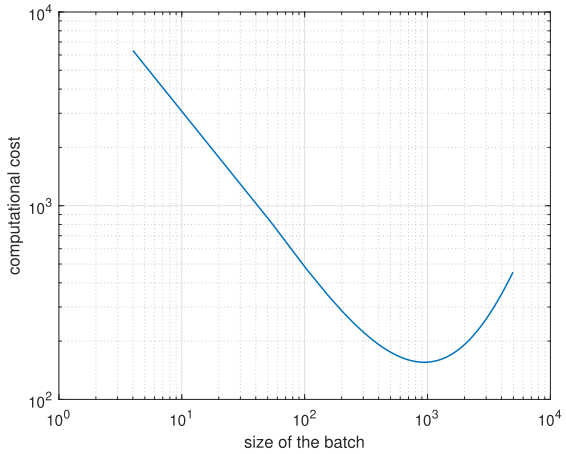


Fig. 3. Computational cost resulting from the model (46) as a function of the size of the batch. The model fits the actual training time in Fig. 2, and its minimum is located approximately in the same region.

The result in Fig. 2 can be explained by developing the following computational model for the hybrid recursions in (21): the multiplication of two matrices of dimension $M_1 \times M_2$ and $M_2 \times M_3$ has computational cost $\mathcal{O}(M_1\ M_2\ M_3)$; matrix inversion of dimension $M \times M$ has computational cost $\mathcal{O}(M^3)$; and we consider that matrix summation and transposition of dimension $M_1 \times M_2$ both have computational cost $\mathcal{O}(M_1\ M_2)$. Using this computational model, it is possible to derive that the hybrid recursions in (21) have computational cost

$$\big(\mathcal{O}(M^3) + \mathcal{O}(M^2) + 3\mathcal{O}(M^2 b) + 2\mathcal{O}(Mb^2) + 2\mathcal{O}(Mb) + \mathcal{O}(b^2) + \mathcal{O}(b^3)\big)N/b. \quad (46)$$

The result of the computational model (46) is reported in Fig. 3 for $N = 50\,000$ and $M = 5000$ and various sizes of the batch b: the computational model (46) fits quite well the computational time observed in Fig. 2, and thus, it can be used to calculate a tradeoff depending on b in terms of training time (the tradeoff also depends on M and N). Both the experiments in Fig. 2 and the computational model in Fig. 3 suggest that a good tradeoff lies around $b \approx 1000$. In the rest of this article, we will use $b = 500$ although the values up to $b = 1000$ give similar results as the ones that we report.

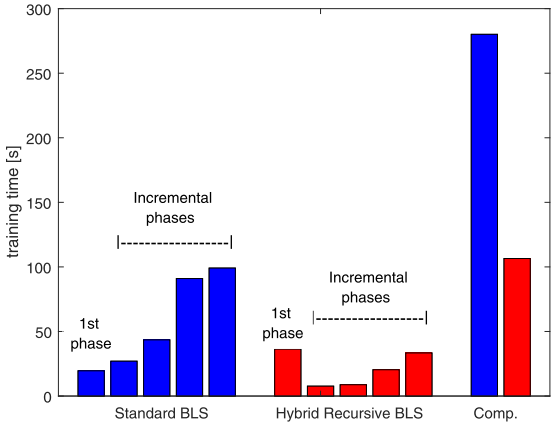


Fig. 4. Training time during the first learning phase and during the incremental phases, for BLS and HR-BLS with incremental enhancement nodes. The last bar represents, for each method, the summation of the training times.

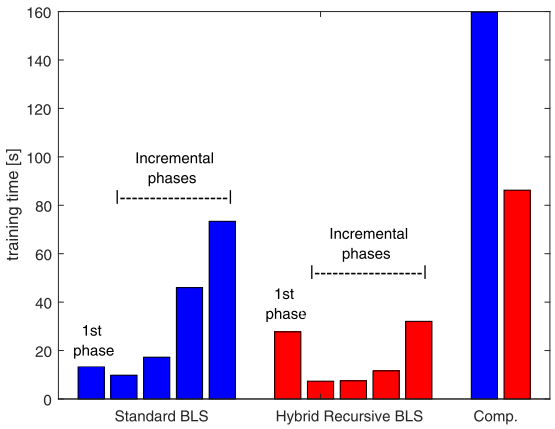


Fig. 5. Training time during the first learning phase and during the incremental phases, for BLS and HR-BLS with incremental feature mappings. The last bar represents, for each method, the summation of the training times.

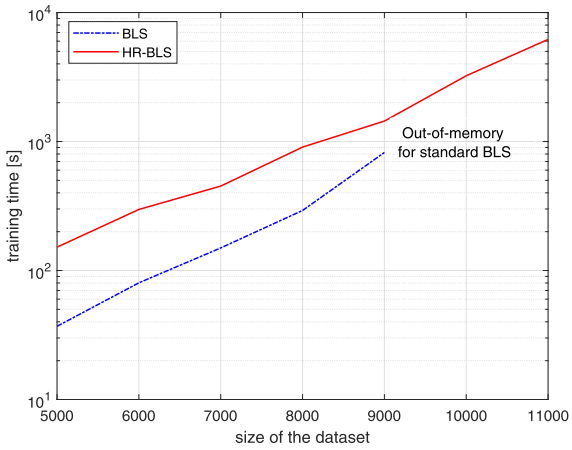


Fig. 6. Training time as the size of the data set and the size of the network increase, for BLS and HR-BLS (first learning phase).

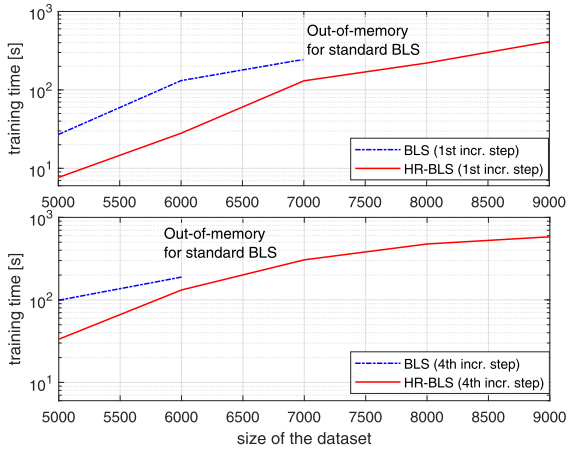


Fig. 7. Training time as the size of the data set and the size of the network increase, for BLS and HR-BLS (first incremental learning step and fourth incremental learning step).

C. Handling Larger and Larger Data Sets

For 15 min-ahead prediction, Fig. 4 shows two groups of five bars: the first bar is the training time during the first learning phase, while the other four bars are the training times during four incremental phases, for BLS and HR-BLS with incremental enhancement nodes. It can be seen that BLS is two times faster in the first phase, while HR-BLS becomes four times more effective during the incremental steps. The last two bars on the right of Fig. 4 (summation of all training times) show that, overall, HR-BLS might be more effective if many incremental steps are performed (HR-BLS is 2.5 times faster than BLS in total). Similar comments apply to Fig. 5, where the incremental steps involve incremental feature mappings: BLS is two times faster than HR-BLS in the first phase, HR-BLS becomes 2.5 times faster than BLS during incremental phases, and overall HR-BLS is around two times faster.

Finally, we would like to study the capabilities of BLS and HR-BLS when the size of the data set and the size of the network increase. In particular, starting from a data set with $N = 50\,000$ and $M = 5000$, we repeatedly increase N by 10 000 and M by 1000 and run both BLS and HR-BLS. It can be noted from Fig. 6 that BLS is put at stake for $N = 100\,000$ and $M = 10\,000$, values for which out-of-memory problems occur (due to inverting a matrix whose size is too large). On the other hand, HR-BLS is never put at stake due to its recursive nature when processing data (it is clear that, as the size of the data set/network increases, a larger training time is to be expected). It is also interesting to note that BLS is initially four times faster in terms of training time for $N = 50\,000$ and, eventually, only two times faster for $N = 90\,000$ before running into out-of-memory issues. Fig. 6 refers to the first learning phase, while Fig. 7 shows that the incremental versions of BLS would present out-of-memory issues already for $N = 70\,000$ samples. The figure shows the training time required for the first incremental step and the fourth incremental step. Previously, it was already discussed that the incremental steps of HR-BLS are faster than the incremental steps of BLS: this is confirmed here as well. Remarkably, the incremental versions of HR-BLS run smoothly for all experiments. This implies that all the comparisons in [31] showing that BLS outperforms many other learning algorithms are valid for HR-BLS as well.

D. Comparisons on MNIST Data Set

In this section, a series of experiments are performed on the classical MNIST data set, consisting of 70 000 handwritten

TABLE II
COMPARISON RESULTS FOR BLS AND HR-BLS ON THE MNIST DATA SET. FOR INCREMENTAL LEARNING, THE FIRST (1ST) AND ALL THE INCREMENTAL (INC.) PHASES ARE REPORTED

MNIST	Training time (s)	Testing accuracy (%)
BLS	515.79	97.06
HR-BLS	594.67	97.08
BLS enh.	425.44 (1st) Out-of-mem. (inc.) Out-of-mem. (inc.)	96.66 (1st) – –
HR-BLS enh.	261.34 (1st) 101.46 (inc.) 185.78 (inc.)	96.66 (1st) 96.72 (inc.) 96.72 (inc.)
BLS feat.	16.75 (1st) 140.35 (inc.) 306.43 (inc.) Out-of-mem. (inc.) Out-of-mem. (inc.)	96.58 (1st) 96.95 (inc.) 97.16 (inc.) – –
HR-BLS feat.	32.26 (1st) 11.45 (inc.) 32.08 (inc.) 133.41 (inc.) 413.76 (inc.)	96.58 (1st) 96.93 (inc.) 97.13 (inc.) 97.19 (inc.) 97.29 (inc.)

digital images partitioned into a training set of 60 000 samples and a test set of 10 000 samples. Different from the PeMS data set, the MNIST data set is a classification problem. This allows us to test the proposed HR-BLS not only as a regression but also as a classification algorithm. Extensive experiments on the MNIST data set have been performed in [29], and the MATLAB source codes of the BLS used in that article can be downloaded from the website of Prof. Chen: http://www.fst.umac.mo/en/staff/pchen.html. Two implementations of the BLS code are provided on the website: BLS for PCs with low memory and BLS for PCs with high memory. Because we want to test the capability of HR-BLS to efficiently use memory, we have used the high memory implementation with the following settings: for BLS (no incremental steps), $n = 100$ and $m = 11\,000$; for BLS with incremental enhancement nodes, $n = 100$ and $m = 9000$ (first phase), with $\bar{m} = 500$ new enhancement nodes in two incremental steps; for BLS with incremental feature nodes, $n = 60$ and $m = 3000$ (first phase), with $\bar{n} = 10$, $\bar{m} = 1250$ new feature/enhancement nodes in four incremental steps.

The same settings are applied to HR-BLS. The experiments run on the same desktop PC as before (Intel Core i5 Quad-Core and 8-GB DDR4). The results of the experiments are given in Table II. It is evident that BLS and HR-BLS are basically equivalent in terms of testing accuracy (this was shown via Propositions 1 and 2, and furthermore, it can be verified that the two BLS implementations converge to almost the same network weights). This makes any comparisons with other algorithms, such as stacked autoencoders (SAEs), multilayer perceptron (MLP), multilayer extremely learning machine structures (MLELM and HELM), deep belief nets (DBNs), deep Boltzmann machines (DBMs), fuzzy restricted Boltzmann machine (FRBM), redundant since all such comparisons have been recently performed in [29]. Where BLS and HR-BLS differ is in their usage of memory, resulting in a very different training time. From Table II, it is crucial to note that the nonincremental version of BLS and HR-BLS has almost the same training time, which indicates that inverting a matrix of large dimension might sometimes be as costly as solving the least-squares recursively via the inversion lemma. Note that, in the BLS and HR-BLS with incremental enhancement nodes, the first phase of the HR-BLS is even faster than the first phase of BLS.

As in the PeMS experiments, HR-BLS shows its full potentialities in the incremental phases: in the BLS with incremental enhancement nodes, an out-of-memory issue arises already in the first incremental phase, whereas the HR-BLS is able to complete all incremental phases smoothly. In the BLS with incremental feature nodes, out-of-memory arises in the third incremental phase: notably, the first two incremental phases of HR-BLS are almost ten times faster than the first two incremental phases of BLS.

TABLE III
COMPARISON RESULTS FOR BLS AND HR-BLS ON THE NYU-NORB DATA SET. FOR INCREMENTAL LEARNING, THE FIRST (1ST) AND ALL THE INCREMENTAL (INC.) PHASES ARE REPORTED

NYU-NORB	Training time (s)	Testing accuracy (%)
BLS	331.59	97.98
HR-BLS	754.42	97.98
BLS enh.	132.26 (1st) 51.63 (inc.) 69.93 (inc.) 98.21 (inc.) Out-of-mem. (inc.)	97.82 (1st) 97.94 (inc.) 97.94 (inc.) 98.00 (inc.) –
HR-BLS enh.	272.64 (1st) 17.38 (inc.) 22.14 (inc.) 22.55 (inc.) 28.03 (inc.)	97.82 (1st) 97.91 (inc.) 97.91 (inc.) 97.96 (inc.) 98.04 (inc.)
BLS feat.	128.18 (1st) 111.83 (inc.) 173.22 (inc.) Out-of-mem. (inc.) Out-of-mem. (inc.)	97.47 (1st) 97.58 (inc.) 97.95 (inc.) – –
HR-BLS feat.	195.11 (1st) 35.53 (inc.) 44.27 (inc.) 50.32 (inc.) 59.03 (inc.)	97.47 (1st) 97.59 (inc.) 97.94 (inc.) 98.00 (inc.) 98.07 (inc.)

E. Comparisons on NYU-NORB Data Set

In this section, a series of experiments are performed on the classical NYU-NORB data set, which, in another data set for classification, consists of stereo image pairs of 50 uniform-colored toys under 36 azimuths, nine elevations, and six lighting conditions. We have used the following settings for BLS: for BLS (no incremental steps), $n = 600$ and $m = 11\,000$; for BLS with incremental enhancement nodes, $n = 600$ and $m = 7000$ (first phase), with $\bar{m} = 1250$ new enhancement nodes in four incremental steps; and for BLS with incremental feature nodes, $n = 350$ and $m = 6000$ (first phase), with $\bar{n} = 50$ and $\bar{m} = 1500$ new feature/enhancement nodes in four incremental steps. The same settings are applied to HR-BLS. The results of the experiments are in Table III. Once more,

it is evident that BLS and HR-BLS are basically equivalent in terms of testing accuracy, whereas the different usage of memory prevents HR-BLS from getting out-of-memory problems.

As in the PeMS and MNIST experiments, HR-BLS shows its full potentialities in the incremental phases: in all incremental phases, HR-BLS is extremely faster than BLS. Overall, the simulation experiments confirm the capability of HR-BLS of handling huge data sets in a more efficient way compared with the standard formulation.

V. Conclusion

BLS is a computationally efficient supervised learning method in which learning is improved by expanding the network architecture in width. In this work, we have illustrated how, due to the need of storing and inverting large matrices, the computational efficiency of BLS might be at risk when the data or the network structure increase to very large values. To overcome this issue, we have proposed a recursive implementation of BLS in which the need of storing and inverting large matrices was avoided. The proposed framework has two distinguishing features: a hybrid nature that can trade off between efficient usage of memory and number of recursion and incremental learning capabilities that avoid a complete retraining process (the already trained portion of the network does not have to be retrained when the width of the network is expanded). A computational model for the proposed method has been developed, which explains that the tradeoff between efficient usage of memory and number of recursion depends on the size of the batch. Regression experiments on a database with more than 100 000 training samples, 100 labels, and up to 10 000 network nodes have shown that the computational efficiency of the proposed framework overcomes one of the standard BLS methodologies. Further experiments on the benchmark data sets have further confirmed the performance of the method.

In the future, we aim at dedicated comparisons between HR-BLS and other shallow and deep supervised learning algorithms. Extensions of the BLS framework in semisupervised or unsupervised settings can be also of interest: examples include feature extraction, clustering tasks [45], or unsupervised encoding as a way of learning and reusing features [37], [46].

References

[1] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, NY, USA: Springer-Verlag, 2010.

[2] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, 2017.

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

[4] P. P. Brahma, D. Wu, and Y. She, "Why deep learning works: A manifold disentanglement perspective," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 27, no. 10, pp. 1997–2008, Oct. 2016.

[5] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[6] D. Wu *et al.*, "Deep learning-based methods for person re-identification: A comprehensive review," *Neurocomputing*, vol. 337, pp. 354–371, Apr. 2019.

[7] D. Liu and S. Yue, "Event-driven continuous STDP learning with deep structure for visual pattern recognition," *IEEE Trans. Cybern.*, vol. 49, no. 4, pp. 1377–1390, Apr. 2019.

[8] J. Xie, G. Dai, F. Zhu, L. Shao, and Y. Fang, "Deep nonlinear metric learning for 3-D shape retrieval," *IEEE Trans. Cybern.*, vol. 48, no. 1, pp. 412–422, Jan. 2018.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2016, pp. 770–778.

[10] Y. Guo and L. Zhang, "One-shot face recognition by promoting underrepresented classes," 2017, *arXiv:1707.05574*. [Online]. Available: http://arxiv.org/abs/1707.05574

[11] F. Xing, Y. Xie, H. Su, F. Liu, and L. Yang, "Deep learning in microscopy image analysis: A survey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 10, pp. 4550–4568, Oct. 2018.

[12] M. Mahmud, M. S. Kaiser, A. Hussain, and S. Vassanelli, "Applications of deep learning and reinforcement learning to biological data," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 6, pp. 2063–2079, Jun. 2018.

[13] H. Tembine, "Deep learning meets game theory: Bregman-based algorithms for interactive deep generative adversarial networks," *IEEE Trans. Cybern.*, vol. 50, no. 3, pp. 1132–1145, Mar. 2020.

[14] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 9, pp. 2805–2824, Sep. 2019.

[15] A. Taherkhani, A. Belatreche, Y. Li, and L. P. Maguire, "A supervised learning algorithm for learning precise timing of multiple spikes in multilayer spiking neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 11, pp. 5394–5407, Nov. 2018.

[16] T. Zhang, W. Zheng, Z. Cui, Y. Zong, and Y. Li, "Spatial–temporal recurrent neural network for emotion recognition," *IEEE Trans. Cybern.*, vol. 49, no. 3, pp. 839–847, Mar. 2019.

[17] T. Wurfl *et al.*, "Deep learning computed tomography: Learning projection-domain weights from image domain in limited angle problems," *IEEE Trans. Med. Imag.*, vol. 37, no. 6, pp. 1454–1463, Jun. 2018.

[18] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, "Deep direct reinforcement learning for financial signal representation and trading," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 3, pp. 653–664, Mar. 2017.

[19] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994.

[20] G. Pandey and A. Dukkipati, "To go deep or wide in learning?" 2014, *arXiv:1402.5634*. [Online]. Available: http://arxiv.org/abs/1402.5634

[21] B. Chandra and R. K. Sharma, "Deep learning with adaptive learning rate using Laplacian score," *Expert Syst. Appl.*, vol. 63, pp. 1–7, Nov. 2016.

[22] A. A. Rusu *et al.*, "Progressive neural networks," 2016, *arXiv:1606.04671*. [Online]. Available: https://arxiv.org/abs/1606.04671

[23] H.-T. Cheng *et al.*, "Wide & deep learning for recommender systems," 2016, *arXiv:1606.07792*. [Online]. Available: http://arxiv.org/abs/1606.07792

[24] C. Yan, L. Li, C. Zhang, B. Liu, Y. Zhang, and Q. Dai, "Cross-modality bridging and knowledge transferring for image understanding," *IEEE Trans. Multimedia*, vol. 21, no. 10, pp. 2675–2685, Oct. 2019.

[25] G. Marcus, "Deep learning: A critical appraisal," 2018, *arXiv:1801.00631*. [Online]. Available: https://arxiv.org/abs/1801.00631

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

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

[28] S. Dehuri and S.-B. Cho, "A comprehensive survey on functional link neural networks and an adaptive PSO–BP learning for CFLNN," *Neural Comput. Appl.*, vol. 19, no. 2, pp. 187–205, Mar. 2010.

[29] C. L. P. Chen and Z. Liu, "Broad learning system: An effective and efficient incremental learning system without the need for deep architecture," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 1, pp. 10–24, Jan. 2018.

[30] D. Ma, J. Wang, Q. Sun, and X. Hu, "A novel broad learning system based leakage detection and universal localization method for pipeline networks," *IEEE Access*, vol. 7, pp. 42343–42353, 2019.

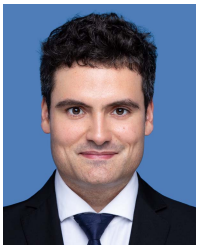
[31] D. Liu, W.-W. Yu, and S. Baldi, "Broad learning for optimal short-term traffic flow prediction," in *Proc. 16th Int. Symp. Neural Netw. (ISNN)*, Moscow, Russia, Jun. 2019, pp. 1–6.

[32] W. Yu and C. Zhao, "Recursive exponential slow feature analysis for fine-scale adaptive processes monitoring with comprehensive operation status identification," *IEEE Trans. Ind. Informat.*, vol. 15, no. 6, pp. 3311–3323, Jun. 2019.

[33] S. B. Jiang, P. K. Wong, R. Guan, Y. Liang, and J. Li, "An efficient fault diagnostic method for three-phase induction motors based on incremental broad learning and non-negative matrix factorization," *IEEE Access*, vol. 7, pp. 17780–17790, 2019.
[34] C. L. P. Chen, Z. Liu, and S. Feng, "Universal approximation capability of broad learning system and its structural variations," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 30, no. 4, pp. 1191–1204, Apr. 2019.
[35] W. Yu and C. Zhao, "Broad convolutional neural network based industrial process fault diagnosis with incremental learning capability," *IEEE Trans. Ind. Electron.*, vol. 67, no. 6, pp. 5081–5091, Jun. 2020.
[36] S. Feng and C. L. P. Chen, "Fuzzy broad learning system: A novel neuro-fuzzy model for regression and classification," *IEEE Trans. Cybern.*, vol. 50, no. 2, pp. 414–424, Feb. 2018.
[37] M. Xu, M. Han, C. L. P. Chen, and T. Qiu, "Recurrent broad learning systems for time series prediction," *IEEE Trans. Cybern.*, vol. 50, no. 4, pp. 1405–1417, Apr. 2020.
[38] T.-L. Zhang, R. Chen, X. Yang, and S. Guo, "Rich feature combination for cost-based broad learning system," *IEEE Access*, vol. 7, pp. 160–172, 2019.
[39] Y. Zhang, Y. Yin, D. Guo, X. Yu, and L. Xiao, "Cross-validation based weights and structure determination of chebyshev-polynomial neural networks for pattern classification," *Pattern Recognit.*, vol. 47, no. 10, pp. 3414–3428, Oct. 2014.
[40] Y. Zhang, D. Guo, Z. Luo, K. Zhai, and H. Tan, "CP-activated WASD neuronet approach to asian population prediction with abundant experimental verification," *Neurocomputing*, vol. 198, pp. 48–57, Jul. 2016.
[41] Y. Zhang, Y. Wang, W. Li, Y. Chou, and Z. Zhang, "WASD algorithm with pruning-while-growing and twice-pruning techniques for multi-input euler polynomial neural network," *Int. J. Artif. Intell. Tools*, vol. 25, no. 02, Apr. 2016, Art. no. 1650007, doi: 10.1142/S021821301650007X.
[42] P. Ioannou and B. Fidan, *Adaptive Control Tutorial*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2006.
[43] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*. New York, NY, USA: Cambridge Univ. Press, 2007.
[44] Y. Wu, H. Tan, L. Qin, B. Ran, and Z. Jiang, "A hybrid deep learning based traffic flow prediction method and its understanding," *Transp. Res. C, Emerg. Technol.*, vol. 90, pp. 166–180, May 2018.
[45] C. Yan *et al.*, "STAT: Spatial-temporal attention mechanism for video captioning," *IEEE Trans. Multimedia*, vol. 22, no. 1, pp. 229–241, Jan. 2020.
[46] Q. Zhou and X. He, "Broad learning model based on enhanced features learning," *IEEE Access*, vol. 7, pp. 42536–42550, 2019.

Di Liu received the B.Sc. degree in electronic information science and technology from the Hubei University of Science and Technology, Xianning, China, and the M.Sc. degree in control science and engineering from the Chongqing University of Posts and Telecommunications, Chongqing, China, in 2014 and 2017, respectively. She is currently pursuing double Ph.D. degree with the School of Cyber Science and Engineering, Southeast University, Nanjing, China, and with Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, Groningen, The Netherlands.

Her research interests are in adaptive control/robust control and adaptive learning systems, with application in intelligent traffic systems and automated vehicles.



Simone Baldi (Senior Member, IEEE) received the B.Sc. degree in electrical engineering and the M.Sc. and Ph.D. degrees in automatic control systems engineering from the University of Florence, Florence, Italy, in 2005, 2007, and 2011, respectively.

He is currently a Professor with the School of Mathematics, Southeast University, Nanjing, China, with a guest position at the Delft Center for Systems and Control, Delft University of Technology, Delft, The Netherlands, where he was an Assistant Professor. His research interests include adaptive and learning systems with applications in networked control systems, smart energy, and intelligent vehicle systems.

Dr. Baldi was the Awarded Outstanding Reviewer of *Applied Energy* in 2016, *Automatica* in 2017, and the *IET Control Theory and Applications* in 2018. Since March 2019, he has been a Subject Editor of the *International Journal of Adaptive Control and Signal Processing*.

Wenwu Yu (Senior Member, IEEE) received the B.Sc. degree in information and computing science and the M.Sc. degree in applied mathematics from the Department of Mathematics, Southeast University, Nanjing, China, in 2004 and 2007, respectively, and the Ph.D. degree from the Department of Electronic Engineering, City University of Hong Kong, Hong Kong, in 2010.

He held several visiting positions in Australia, China, Germany, Italy, The Netherlands, and the USA. He is currently the Founding Director of the Laboratory of Cooperative Control of Complex Systems, the Deputy Associate Director of the Jiangsu Provincial Key Laboratory of Networked Collective Intelligence, an Associate Dean of the School of Mathematics, and a Full Professor with the Young Endowed Chair Honor in Southeast University. He has published about 100 SCI journal articles with more than 10000 citations. His research interests include multiagent systems, complex networks and systems, disturbance control, distributed optimization, neural networks, game theory, cyberspace security, smart grids, intelligent transportation systems, and big-data analysis.

Dr. Yu serves as an Editorial Board Member of several flag journals, including the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS II, the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS, *Science China Information Sciences*, and *Science China Technological Sciences*. He was a recipient of the Second Prize of State Natural Science Award of China in 2016. He was listed by Clarivate Analytics/Thomson Reuters Highly Cited Researchers in Engineering from 2014 to 2019

C. L. Philip Chen (Fellow, IEEE) received the M.Sc. degree in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 1985, and the Ph.D. degree in electrical engineering from Purdue University, West Lafayette, IN, USA, in 1988.

He was a Chair Professor and the Dean of the Department of Computer and Information Science, Faculty of Science and Technology, University of Macau, Macau. He is currently a Chair Professor and the Dean of the College of Computer Science and Engineering, South China University of Technology, Guangzhou, China. His current research interests include systems, cybernetics, and computational intelligence.

Dr. Chen is also a fellow of the American Association for the Advancement of Science, the International Association of Pattern Recognition (IAPR), the Chinese Association of Automation (CAA), and the Hong Kong Institute of Engineers (HKIE). He is also a member of Academia Europaea (AE), the European Academy of Sciences and Arts (EASA), and the International Academy of Systems and Cybernetics Science (IASCYS). He received the IEEE Norbert Wiener Award in 2018 for his contribution to systems and cybernetics, and machine learnings. He is a Highly Cited Researcher by Clarivate Analytics from 2018 to 2020. He was the Chair of the TC 9.1 Economic and Business Systems of International Federation of Automatic Control from 2015 to 2017 and a Program Evaluator of the Accreditation Board of Engineering and Technology Education of the U.S. He was the Editor-in-Chief of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS: SYSTEMS from 2014 to 2019. He is also an associate editor of several IEEE TRANSACTIONS. He has been the Editor-in-Chief of the IEEE TRANSACTIONS ON CYBERNETICS since 2019. He was the IEEE SMC Society President from 2012 to 2013 and the Vice President of the Chinese Association of Automation (CAA).