SCIENCE CHINA
Information Sciences

• Supplementary File •

An accurate and efficient online broad learning system for data stream classification

Chunyu LEI[1,3,4], Guang-Ze CHEN[2], C. L. Philip CHEN[1,3,4] & Tong ZHANG[1,3,4*]

[1]*School of Computer Science & Engineering, South China University of Technology, Guangzhou 510006, China;*
[2]*Faculty of Science and Technology, University of Macau, Macau 999078, China;*
[3]*Research Center for AI Large Models and Intelligent Cognition, Pazhou Lab, Guangzhou 510335, China;*
[4]*Engineering Research Center of the Ministry of Education on Health Intelligent Perception and Paralleled Digital-Human, Guangzhou 510006, China*

Appendix A Related work

Appendix A.1 Online learning

Online learning (OL) is a family of stream machine learning algorithms designed to incrementally build models from sequential data [1]. Existing OL approaches fall into two main categories. The first category refers to conventional OL algorithms modified from traditional machine learning models. For example, the Hoeffding tree (HT) has been proposed to construct decision trees online from data streams [2]. Subsequently, an adaptive random forest (ARF) has been developed to handle concept drift scenarios [3]. Specifically, once concept drift is detected, ARF resets its base trees to adapt to the new concept. Traditional online machine learning methods also include passive-aggressive (PA) [4], perceptron [5], leveraging bagging (LB) [6], etc. Despite simplicity, all of them fail to extract effective features and learn complex nonlinear functions from data streams and thus degrade their accuracy. Moreover, several OL methods based on singular value decomposition (SVD) have been proposed and gained significant attention in recent years, such as adaptive subgradient method with diagonal matrix proximal functions (ADA-DIAG) [7], RADAGRAD [8], frequent directions-based sketched online Newton (FD-SON) [9], adaptive online learning via fast frequent directions (ADA-FFD) [10], Sketchy-ADAGRAD (S-ADA) [11], and follow the fast sketchy leader (FTFSL) [12]. Regrettably, these methods cannot address the issue of concept drift.

Inspired by deep learning [13, 14], the second one is online deep learning, which aims to learn a deep neural network on the fly. To be specific, online gradient descent (OGD) has been proposed to update network weights using gradients when a sample arrives [15]. Since network depth is difficult to determine in advance, [16] proposed the hedge backpropagation (HBP) to dynamically integrate multiple layers of neural networks, which is of great significance for subsequent research. Subsequently, [17] proposed an autonomous deep learning (ADL) model that can construct its network structure from scratch by adjusting its depth and width without an initial one. To solve the online hyperparameter optimization challenge of neural networks, a continuously adaptive neural network for data streams (CAND) [18] has been proposed, which chooses the best model from a candidate pool of neural networks trained with different hyperparameter combinations. An adaptive tree-like neural network (ATNN) has been proposed to mitigate the catastrophic forgetting problem by choosing suitable positions on the backbone to grow branches for the new concept [19]. To further handle concept drift and sub-network optimization conflict issues, elastic online deep learning (EODL) has been proposed, including depth adaptation and parameter adaptation strategies [20]. Despite significant progress, none of them has gotten rid of gradient descent algorithms. In other words, all of them execute only a single error backpropagation and weight update when a new sample arrives. These algorithms are far from enough to ensure the optimality of their model weights for future predictions. Hence, developing an OL framework based on closed-form solutions becomes crucial.

Appendix A.2 Incremental broad learning system

Due to its efficiency and versatility, broad learning system (BLS) has been widely used for a variety of tasks [21–26]. In particular, substantial works have been presented for incremental machine learning tasks. The first data incremental algorithm for BLS is deduced from Greville's theory [27], which we refer to incremental BLS (I-BLS) in this study [28]. However, the performance of I-BLS dramatically degrades when learning a new class or task. To this end, the task-incremental broad learning system (TiBLS) and broad learning system-based class incremental learning (BLS-CIL) have been proposed based on residual learning and graph regularization, respectively [29, 30]. Furthermore, motivated by the fact that I-BLS still fails to improve accuracy when the size of the training data is not roughly equal for each addition, Zhong et al. have proposed a robust incremental broad learning system (RI-BLS) [31]. Among them, TiBLS stacks a new BLS module on the current model whenever a new task arrives. It views each new training sample as a new task in an online machine learning scenario. Therefore, a new BLS model needs to be learned. Inevitably, it imposes an expensive

* Corresponding author (email: tony@scut.edu.cn)

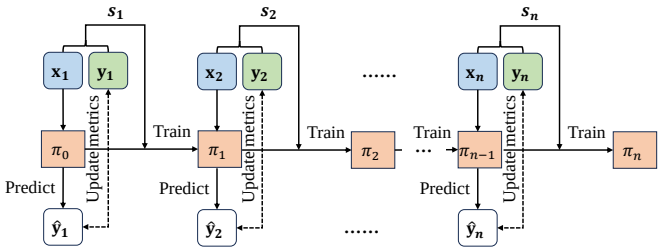

Figure B1 The problem setting of online learning from data stream.

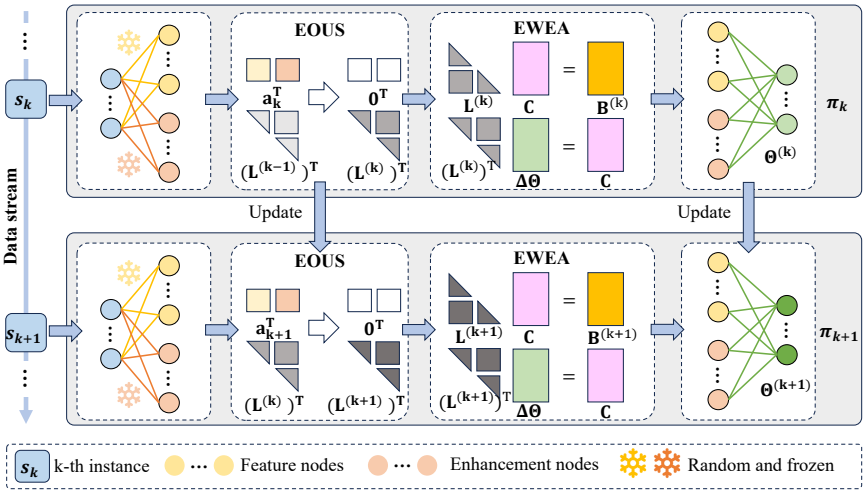

Figure B2 The online learning process of our proposed Online-BLS.

computational overhead and does not meet the requirements for efficient OL. In contrast, the remaining three algorithms only need to fine-tune their weights based on new arrivals, which is relatively efficient. Despite their feasibility, none of them can avoid matrix inversion operations. Thus, the defects of matrix inversion still come along with them.

Appendix B Method

Appendix B.1 Problem setting

In this study, we focus on online data stream classification tasks. The input data sequence can be represented as $\{\mathbf{s}_k := (\mathbf{x}_k, \mathbf{y}_k)|k = 1, 2, \cdots, n\}$, where $\mathbf{x}_k \in \mathbb{R}^d$ denotes the k-th sample with d feature values and $\mathbf{y}_k \in \mathbb{R}^t$ is the one-hot target. t is the total number of categories. A sequence of models $\pi_1, \pi_2, \cdots, \pi_n$ are generated on the given data stream $\mathbf{s}_1, \mathbf{s}_2, \cdots, \mathbf{s}_n$ as shown in Figure B1. The model $\pi_k : \mathbb{R}^d \to \mathbb{R}^t$ depends solely on its previous status π_{k-1} and the most recent data point $\mathbf{s}_k$. To be exact, when a sample $\mathbf{x}_k$ arrives, the prediction $\hat{\mathbf{y}}_k$ is obtained using π_{k-1} first. Subsequently, metrics are updated based on $\mathbf{y}_k$ and $\hat{\mathbf{y}}_k$. Lastly, the updated online learning model π_k is obtained through a single online update. To accomplish the above tasks, our Online-BLS method proposed in this study contains two main modules: an effective weight estimation algorithm (EWEA) and an efficient online update strategy (EOUS). The OL process of Online-BLS is shown in Figure B2. Since EWEA is used to estimate model weights and is an indispensable part of its online weight update step, while EOUS only aims to improve its online learning efficiency. Following the introduction of our Online-BLS framework, we detail EWEA before EOUS.

Appendix B.2 Online-BLS framework

In this section, we derive an Online-BLS framework. It provides a closed-form solution for each online update step and therefore resolves the issue of suboptimal online model weights.

Given the first sample $\mathbf{x}_1 \in \mathbb{R}^d$, assume there are n_2 feature layers and n_4 enhancement layers. The outputs of the i-th feature layer (i.e., the i-th group of feature nodes) $\mathbf{z}_i \in \mathbb{R}^{n_1}$ and the j-th enhancement layer (i.e., the j-th group of enhancement nodes) $\mathbf{h}_j \in \mathbb{R}^{n_3}$ are defined as

$$\mathbf{z}_i^\top = \phi\left(\mathbf{x}_1^\top \mathbf{W}_{f_i} + \beta_{f_i}^\top\right), \quad i = 1, 2, \cdots, n_2, \tag{B1}$$

$$\mathbf{h}_j^\top = \sigma\left(\mathbf{x}_1^\top \mathbf{W}_{e_j} + \beta_{e_j}^\top\right), \quad j = 1, 2, \cdots, n_4. \tag{B2}$$

Here, $\mathbf{W}_{f_i} \in \mathbb{R}^{d\times n_1}$, $\mathbf{W}_{e_j} \in \mathbb{R}^{d\times n_3}$, $\beta_{f_i} \in \mathbb{R}^{n_1}$, and $\beta_{e_j} \in \mathbb{R}^{n_3}$ are randomly generated weights and biases. $\phi(\cdot)$ and $\sigma(\cdot)$ are generally a linear transformation and a nonlinear activation, respectively. Through incorporating linear and non-linear features, the broad feature vector $\mathbf{a}_1$ is defined as

$$\mathbf{a}_1^\top \triangleq \left[\mathbf{z}_1^\top, \mathbf{z}_2^\top, \cdots, \mathbf{z}_{n_2}^\top, \mathbf{h}_1^\top, \mathbf{h}_2^\top, \cdots, \mathbf{h}_{n_4}^\top\right]. \tag{B3}$$

Then, the prediction of $\mathbf{x}_1$ can be calculated as $\hat{\mathbf{y}}_1^\top = \mathbf{a}_1^\top \boldsymbol{\Theta}^{(0)}$, where $\boldsymbol{\Theta}^{(0)}$ is a zero matrix. After $\mathbf{y}_1$ is revealed, the corresponding optimization problem is formulated as $\boldsymbol{\Theta}^{(1)} = \arg\min_{\boldsymbol{\Theta}} \left|\mathbf{a}_1^\top \boldsymbol{\Theta} - \mathbf{y}_1^\top\right|^2 + \lambda |\boldsymbol{\Theta}|^2$, where $|\cdot|$ denotes the L_2 norm. λ is a positive constant. Since it is a convex optimization problem, we derive its gradient with respect to $\boldsymbol{\Theta}$ and set it to $\mathbf{0}$. Then, we have

$$\left(\mathbf{a}_1\mathbf{a}_1^\top + \lambda\mathbf{I}\right)\boldsymbol{\Theta} = \mathbf{a}_1\mathbf{y}_1^\top. \tag{B4}$$

When the k-th sample $\mathbf{x}_k (k = 2, 3, \cdots, n)$ arrives, $\mathbf{a}_k$ is obtained by replacing $\mathbf{x}_1$ in Eq. (B1) and Eq. (B2) with $\mathbf{x}_k$. Then, its prediction is $\hat{\mathbf{y}}_k^\top = \mathbf{a}_k^\top \boldsymbol{\Theta}^{(k-1)}$. Suppose we retrain $\boldsymbol{\Theta}^{(k)}$ with the first k samples, Eq. (B4) becomes

$$\left([\mathbf{a}_1, \cdots, \mathbf{a}_k][\mathbf{a}_1, \cdots, \mathbf{a}_k]^\top + \lambda\mathbf{I}\right)\boldsymbol{\Theta} = [\mathbf{a}_1, \cdots, \mathbf{a}_k][\mathbf{y}_1, \cdots, \mathbf{y}_k]^\top. \tag{B5}$$

Let $\mathbf{K}^{(1)} = \mathbf{a}_1\mathbf{a}_1^\top + \lambda\mathbf{I}$, we have $\mathbf{K}^{(k)} = [\mathbf{a}_1, \cdots, \mathbf{a}_k][\mathbf{a}_1, \cdots, \mathbf{a}_k]^\top + \lambda\mathbf{I} = \mathbf{K}^{(k-1)} + \mathbf{a}_k\mathbf{a}_k^\top$ and

$$[\mathbf{a}_1, \cdots, \mathbf{a}_k][\mathbf{y}_1, \cdots, \mathbf{y}_k]^\top = \sum_{i=1}^{k-1} \mathbf{a}_i\mathbf{y}_i^\top + \mathbf{a}_k\mathbf{y}_k^\top = \mathbf{K}^{(k-1)}\boldsymbol{\Theta}^{(k-1)} + \mathbf{a}_k\mathbf{y}_k^\top = \mathbf{K}^{(k)}\boldsymbol{\Theta}^{(k-1)} - \mathbf{a}_k\left(\mathbf{a}_k^\top\boldsymbol{\Theta}^{(k-1)} - \mathbf{y}_k^\top\right). \tag{B6}$$

Substituting Eq. (B6) into Eq. (B5), we have

$$\mathbf{K}^{(k)}\boldsymbol{\Delta\Theta} = \mathbf{B}^{(k)}, \tag{B7}$$

where $\boldsymbol{\Delta\Theta} = \boldsymbol{\Theta} - \boldsymbol{\Theta}^{(k-1)}$ and $\mathbf{B}^{(k)} = \mathbf{a}_k\left(\mathbf{y}_k^\top - \mathbf{a}_k^\top\boldsymbol{\Theta}^{(k-1)}\right)$.

Importantly, the right-hand terms of Eq. (B4) and Eq. (B7) are equal when k is equal to 1 (i.e., $\mathbf{B}^{(1)} = \mathbf{a}_1\mathbf{y}_1^\top$). Thus, we conclude that the matrix $\boldsymbol{\Theta}^{(1)}$ solved via Eq. (B4) and Eq. (B7) is equivalent if and only if $\boldsymbol{\Theta}^{(0)} = \mathbf{0}$ and $\mathbf{K}^{(0)} = \lambda\mathbf{I}$. In other words, we only need to solve Eq. (B7) to get $\boldsymbol{\Delta\Theta}^{(k)}$. Then, we have

$$\boldsymbol{\Theta}^{(k)} = \boldsymbol{\Theta}^{(k-1)} + \boldsymbol{\Delta\Theta}^{(k)}, \quad k = 1, 2, \cdots, n. \tag{B8}$$

Since Eq. (B7) has a closed-form solution, our Online-BLS framework is completed without a gradient descent step. The algorithms for accurately and efficiently deriving closed-form solutions to Eq. (B7) will be described in later sections.

Appendix B.3 Effective weight estimation algorithm

To solve $\boldsymbol{\Delta\Theta}$ accurately, we propose an EWEA based on Cholesky factorization and forward-backward substitution. Since $\mathbf{K}^{(k)}$ is always a symmetric positive definite matrix, we perform a Cholesky decomposition on $\mathbf{K}^{(k)}$ to obtain $\mathbf{K}^{(k)} = \mathbf{L}^{(k)}\left(\mathbf{L}^{(k)}\right)^\top$, where $\mathbf{L}^{(k)}$ is a lower triangular matrix with positive diagonal entries.

Proposition B1. Let $\lambda > 0$. Then, $\mathbf{K}^{(k)}$ is a symmetric positive definite matrix.

Proof.

The proof is completed from symmetry and positive definiteness.

1) Symmetry: Clearly, $\mathbf{K}^{(0)} = \lambda\mathbf{I}$ is symmetric. Assuming $\mathbf{K}^{(k-1)}$ is symmetric, we have $\left(\mathbf{K}^{(k)}\right)^\top = \left(\mathbf{K}^{(k-1)} + \mathbf{a}_k\mathbf{a}_k^\top\right)^\top$ $= \mathbf{K}^{(k-1)} + \mathbf{a}_k\mathbf{a}_k^\top = \mathbf{K}^{(k)}$. According to mathematical induction, $\mathbf{K}^{(k)}$ is symmetric when $k \geqslant 0$ and k is an integer.

2) Positive definiteness: For any nonzero vector $\xi \in \mathbb{R}^m$, where m represents the broad feature dimension (i.e., $m = n_1\times n_2 + n_3\times n_4$), we have $\xi^\top\mathbf{K}^{(0)}\xi = \lambda\xi^\top\mathbf{I}\xi = \lambda|\xi|^2 > 0$, since $\lambda > 0$. Thus, $\mathbf{K}^{(0)}$ is positive definite. Suppose $\mathbf{K}^{(k-1)}$ is positive definite, that is $\xi^\top\mathbf{K}^{(k-1)}\xi > 0$. Then, we have $\xi^\top\mathbf{K}^{(k)}\xi = \xi^\top\left(\mathbf{K}^{(k-1)} + \mathbf{a}_k\mathbf{a}_k^\top\right)\xi = \xi^\top\mathbf{K}^{(k-1)}\xi + \left(\mathbf{a}_k^\top\xi\right)^2$. The first term is strictly positive by the induction hypothesis, and the second term is nonnegative. Hence, $\xi^\top\mathbf{K}^{(k)}\xi > 0$, and $\mathbf{K}^{(k)}$ is positive definite.

In conclusion, $\mathbf{K}^{(k)}$ is symmetric positive definite, which completes this proof.

Thus, Eq. (B7) can be solved via the following two equations:

$$\mathbf{L}^{(k)}\mathbf{C} = \mathbf{B}^{(k)} \quad \text{(Forward substitution)}, \tag{B9}$$

$$\left(\mathbf{L}^{(k)}\right)^\top\boldsymbol{\Delta\Theta} = \mathbf{C} \qquad \text{(Backward substitution)}. \tag{B10}$$

Here, $\mathbf{C}$ is an intermediate variable. Denotes $\mathbf{B}^{(k)} = [\mathbf{b}_1, \cdots, \mathbf{b}_m]^\top$, where m represents the broad feature dimension (i.e., $m = n_1 \times n_2 + n_3 \times n_4$), using forward and backward substitution, the solutions of Eq. (B9) and Eq. (B10) are given by Algorithm B1. Then, our weight matrix $\boldsymbol{\Theta}^{(k)}$ can be obtained online by Eq. (B8) without resorting to the numerically unstable matrix inverse operation. The corresponding error bounds are discussed in Appendix B.6.1.

Algorithm B1 Forward-backward substitution algorithm

Input: Lower triangular matrix $\mathbf{L}^{(k)} \in \mathbb{R}^{m\times m}$, right-hand term $\mathbf{B}^{(k)} \in \mathbb{R}^{m\times t}$.
Output: Incremental weight $\boldsymbol{\Delta\Theta}^{(k)}$.
1: Let $l_{ij}^{(k)}$ denote the (i,j)-th entry of $\mathbf{L}^{(k)}$.
2: **// Step 1. Forward substitution**
3: The first row of $\mathbf{C}$: $\mathbf{c}_1^\top = \left(\mathbf{b}_1^{(k)}\right)^\top \big/ l_{11}^{(k)}$, where $\left(\mathbf{b}_1^{(k)}\right)^\top$ denotes the first row of $\mathbf{B}^{(k)}$.
4: **for** $i = 2, 3, \cdots, m$ **do**
5: The i-th row of $\mathbf{C}$: $\mathbf{c}_i^\top = \left(\left(\mathbf{b}_i^{(k)}\right)^\top - \sum_{d=1}^{i-1} l_{di}^{(k)} \mathbf{c}_d^\top\right) \big/ l_{ii}^{(k)}$, where $\left(\mathbf{b}_i^{(k)}\right)^\top$ denotes the i-th row of $\mathbf{B}^{(k)}$.
6: **end for**
7: **// Step 2. Backward substitution**
8: The m-th row of $\boldsymbol{\Delta\Theta}^{(k)}$: $\left(\boldsymbol{\Delta}\theta_m^{(k)}\right)^\top = \mathbf{c}_m^\top \big/ l_{mm}^{(k)}$, where $\mathbf{c}_m^\top$ denotes the m-th row of $\mathbf{C}$.
9: **for** $j = m-1, m-2, \cdots, 1$ **do**
10: The j-th row of $\boldsymbol{\Delta\Theta}^{(k)}$: $\left(\boldsymbol{\Delta}\theta_j^{(k)}\right)^\top = \left(\mathbf{c}_j^\top - \sum_{d=j+1}^{m} l_{jd}^{(k)} \left(\boldsymbol{\Delta}\theta_d^{(k)}\right)^\top\right) \big/ l_{jj}^{(k)}$, where $\mathbf{c}_j^\top$ denotes the j-th row of $\mathbf{C}$.
11: **end for**
12: Assemble $\boldsymbol{\Delta\Theta}^{(k)} = \left[\left(\boldsymbol{\Delta}\theta_1^{(k)}\right)^\top, \left(\boldsymbol{\Delta}\theta_2^{(k)}\right)^\top, \cdots, \left(\boldsymbol{\Delta}\theta_m^{(k)}\right)^\top\right]^\top$.

Appendix B.4 Efficient online update strategy

To avoid expensive Cholesky decompositions for each online update, we discuss below how to efficiently derive $\mathbf{L}^{(k)}$ from $\mathbf{L}^{(k-1)}$ and $\mathbf{a}_k$, where $\mathbf{K}^{(k)} = \mathbf{L}^{(k)}\left(\mathbf{L}^{(k)}\right)^\top$. Motivated by the rank-one update strategy [32], we first cascade the k-th broad feature $\mathbf{a}_k$ and the previous Cholesky factor $\mathbf{L}^{(k-1)}$, forming $[\mathbf{a}_k, \mathbf{L}^{(k-1)}]$. The core idea for answering the above question is to construct an orthogonal matrix $\mathbf{G}$ that can transform the augmented matrix as follows:

$$\mathbf{G}\left[\mathbf{a}_k, \mathbf{L}^{(k-1)}\right]^\top = [\mathbf{0}, \bar{\mathbf{L}}]^\top, \tag{B11}$$

where $\mathbf{G}$ is a sequence of orthogonal Givens matrices of the form $\mathbf{G} = \mathbf{G}_m\mathbf{G}_{m-1}\cdots\mathbf{G}_1$. Here, $\mathbf{G}_i$ is responsible for converting the i-th element of $\mathbf{a}_k$ to 0. Next, we first assume that the orthogonal matrix $\mathbf{G}$ satisfying the above requirements exists to derive our key conclusion. Then, we discuss how to construct the matrix $\mathbf{G}$.

From Eq. (B11), we have

$$\left[\mathbf{0}, \bar{\mathbf{L}}\right]\left[\mathbf{0}, \bar{\mathbf{L}}\right]^\top = \bar{\mathbf{L}}\bar{\mathbf{L}}^\top = \left[\mathbf{a}_k, \mathbf{L}^{(\mathbf{k-1})}\right]\underbrace{\mathbf{G}^\top\mathbf{G}}_{=\mathbf{I}}\left[\mathbf{a}_k, \mathbf{L}^{(k-1)}\right]^\top = \mathbf{L}^{(k-1)}\left(\mathbf{L}^{(k-1)}\right)^\top + \mathbf{a}_k\mathbf{a}_k^\top = \mathbf{K}^{(k-1)} + \mathbf{a}_k\mathbf{a}_k^\top, \tag{B12}$$

since $\mathbf{G}$ is an orthogonal matrix. Thus, $\bar{\mathbf{L}} = \mathbf{L}^{(k)}$ is the updated Cholesky factor we need. In other words, as long as we find the orthogonal matrix $\mathbf{G}$ and perform the orthogonal transformation in Eq. (B11), we can quickly obtain the new Cholesky factor $\mathbf{L}^{(k)}$ whilst avoiding re-performing the expensive Cholesky decomposition. The improvement in time complexity will be discussed in Section Appendix B.6.2.

Next, we consider how to construct the above Givens matrix $\mathbf{G}$. Let our initial augmented matrix $\mathbf{M}_0$ be

$$\mathbf{M}_0 = \left[\mathbf{a}_k, \mathbf{L}^{(k-1)}\right]^\top = \begin{bmatrix}\mathbf{a}_k^\top \\ \left(\mathbf{L}^{(k-1)}\right)^\top\end{bmatrix} = \begin{bmatrix} a_{k1} & a_{k2} & \cdots & a_{km} \\ l_{11}^{(k-1)} & l_{21}^{(k-1)} & \cdots & l_{m1}^{(k-1)} \\ 0 & l_{22}^{(k-1)} & \cdots & l_{m2}^{(k-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & l_{mm}^{(k-1)} \end{bmatrix}, \tag{B13}$$

our aim is to create a series of Givens matrices to eliminate each element in the vector $\mathbf{a}_k$. To eliminate a_{k1}, we construct a Givens matrix $\mathbf{G}_1$ and apply it to $\mathbf{M}_0$ to produce a new matrix $\mathbf{M}_1$:

$$\mathbf{M}_1 = \mathbf{G}_1\mathbf{M}_0 = \begin{bmatrix} c_1 & -s_1 & 0 & \cdots & 0 \\ s_1 & c_1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{bmatrix}\begin{bmatrix} a_{k1} & a_{k2} & \cdots & a_{km} \\ l_{11}^{(k-1)} & l_{21}^{(k-1)} & \cdots & l_{m1}^{(k-1)} \\ 0 & l_{22}^{(k-1)} & \cdots & l_{m2}^{(k-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & l_{mm}^{(k-1)} \end{bmatrix} = \begin{bmatrix} 0 & a'_{k2} & \cdots & a'_{km} \\ l_{11}^{(k-1)'} & l_{21}^{(k-1)'} & \cdots & l_{m1}^{(k-1)'} \\ 0 & l_{22}^{(k-1)} & \cdots & l_{m2}^{(k-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & l_{mm}^{(k-1)} \end{bmatrix}, \tag{B14}$$

where $c_1 = l_{11}^{(k-1)} \big/ \sqrt{\left(l_{11}^{(k-1)}\right)^2 + a_{k1}^2}$, $s_1 = a_{k1} \big/ \sqrt{\left(l_{11}^{(k-1)}\right)^2 + a_{k1}^2}$. Analogously, to eliminate a_{k2}, we construct a Givens

Algorithm B2 Online-BLS algorithm

Input: Stream data $\{\mathbf{s}_k := (\mathbf{x}_k, \mathbf{y}_k) \mid k = 1, 2, \cdots, n\}$, parameters n_1, n_2, n_3, n_4, and λ.
Output: Prediction results $\{\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \cdots, \hat{\mathbf{y}}_\mathbf{n}\}$.
1: Initialize $\mathbf{\Theta}^{(0)} = \mathbf{0}$, $\mathbf{L}^{(0)} = \text{Chol}(\lambda \mathbf{I})$
2: **for** $k = 1, 2, \cdots, n$ **do**
3: Receive instance: $\mathbf{x}_k$
4: Obtain broad feature $\mathbf{a}_k$ via Eq. (B1), Eq. (B2), and Eq. (B3)
5: Predict $\hat{\mathbf{y}}_k^\top = \mathbf{a}_k^\top \mathbf{\Theta}^{(k-1)}$ and then reveal $\mathbf{y}_k$
6: Obtain the updated factor $\mathbf{L}^{(k)}$ by Eq. (B11)
7: Obtain $\mathbf{\Theta}^{(k)}$ by Algorithm B1 and Eq. (B8).
8: **end for**

matrix $\mathbf{G}_2$ and apply it to $\mathbf{M}_1$ to produce a new matrix $\mathbf{M}_2$:

$$\mathbf{M}_2 = \mathbf{G}_2\mathbf{M}_1 = \begin{bmatrix} c_2 & 0 & -s_2 & 0 & \cdots & 0 \\ 0 & 1 & 0 & 0 & \cdots & 0 \\ s_2 & 0 & c_2 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} 0 & a'_{k2} & \cdots & a'_{km} \\ l_{11}^{(k-1)\prime} & l_{21}^{(k-1)\prime} & \cdots & l_{m1}^{(k-1)\prime} \\ 0 & l_{22}^{(k-1)} & \cdots & l_{m2}^{(k-1)} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & l_{mm}^{(k-1)} \end{bmatrix} = \begin{bmatrix} 0 & 0 & a''_{k3} & \cdots & a''_{km} \\ l_{11}^{(k-1)\prime} & l_{21}^{(k-1)\prime} & l_{31}^{(k-1)\prime} & \cdots & l_{m1}^{(k-1)\prime} \\ 0 & l_{22}^{(k-1)\prime\prime} & l_{32}^{(k-1)\prime\prime} & \cdots & l_{m2}^{(k-1)\prime\prime} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & l_{mm}^{(k-1)} \end{bmatrix}, \tag{B15}$$

where $c_2 = l_{22}^{(k-1)} \Big/ \sqrt{\left(l_{22}^{(k-1)}\right)^2 + \left(a'_{k2}\right)^2}$, $s_2 = a'_{k2} \Big/ \sqrt{\left(l_{22}^{(k-1)}\right)^2 + \left(a'_{k2}\right)^2}$.

The above process is repeated m times, yielding $\mathbf{G}_1, \mathbf{M}_1, \mathbf{G}_2, \mathbf{M}_2, \cdots, \mathbf{G}_m, \mathbf{M}_m$ sequentially, where $\mathbf{M}_m$ is our final matrix as shown in Eq. (B11). Thus, our final Givens matrix is $\mathbf{G} = \mathbf{G}_m\mathbf{G}_{m-1}\cdots\mathbf{G}_1$.

Remark 1. The Givens matrices $(\mathbf{G}_1, \mathbf{G}_2, \cdots, \mathbf{G}_m)$ only alter the elements of specific two rows within the augmented matrix. Thus, in implementation, it is unnecessary to perform the full matrix multiplication as shown in Eq. (B14) and Eq. (B15). Instead, it is more efficient to multiply the reduced 2×2 Givens matrix by the specific two rows within the augmented matrix. Thus, the Givens transformations in Eq. (B14) and Eq. (B15) can be rewritten as

$$\mathbf{M}_1[\{0,1\},:] = \mathbf{G}'_1\mathbf{M}_0[\{0,1\},:] = \begin{bmatrix} c_1 & -s_1 \\ s_1 & c_1 \end{bmatrix} \begin{bmatrix} a_{k1} & a_{k2} & \cdots & a_{km} \\ l_{11}^{(k-1)} & l_{21}^{(k-1)} & \cdots & l_{m1}^{(k-1)} \end{bmatrix} = \begin{bmatrix} 0 & a'_{k2} & \cdots & a'_{km} \\ l_{11}^{(k-1)\prime} & l_{21}^{(k-1)\prime} & \cdots & l_{m1}^{(k-1)\prime} \end{bmatrix}, \tag{B16}$$

and

$$\mathbf{M}_2[\{0,2\},:] = \mathbf{G}'_2\mathbf{M}_1[\{0,2\},:] = \begin{bmatrix} c_2 & -s_2 \\ s_2 & c_2 \end{bmatrix} \begin{bmatrix} 0 & a_{k2} & \cdots & a_{km} \\ 0 & l_{22}^{(k-1)} & \cdots & l_{m2}^{(k-1)} \end{bmatrix} = \begin{bmatrix} 0 & 0 & a''_{k3} & \cdots & a''_{km} \\ 0 & l_{22}^{(k-1)\prime\prime} & l_{32}^{(k-1)\prime\prime} & \cdots & l_{m2}^{(k-1)\prime\prime} \end{bmatrix}, \tag{B17}$$

respectively, where $\mathbf{M}[\{i,j\},:]$ denotes extracting rows i and j of matrix $\mathbf{M}$. $\mathbf{G}'_j$ indicates the j-th reduced Givens matrix.

Remark 2. The Cholesky factor requires that the elements on the main diagonal are greater than 0. Thus, if $\bar{\mathbf{L}}_{jj} < 0$, we simply filp $c_j \leftarrow -c_j$ and $s_j \leftarrow -s_j$. Then, $\bar{\mathbf{L}}_{jj} > 0$ becomes satisfied.

Remark 3. The function $\text{Chol}(\mathbf{M})$ refers to the Cholesky factorization of $\mathbf{M}$ and returns its lower triangular factor $\mathbf{L}$.

The Online-BLS algorithm is summarized in Algorithm B2.

Appendix B.5 Handing concept drift

Concept drift is a common challenge in data streams, where the joint distribution of features and labels change over time. Existing incremental BLS algorithms cannot handle concept drift scenarios in stream learning. Thanks to the flexibility of our Online-BLS framework, we are able to adopt a simple solution to the concept drift problem, by modifying two lines of code upon Algorithm B2, which is listed in Algorithm B3. The core idea is that learning the time-varying concept with new data is better than with old one. Technically, more attention to new data is encouraged by multiplying history by a decay factor μ. Its effectiveness will be verified across four non-stationary datasets.

Appendix B.6 Theoretical analysis

Appendix B.6.1 *Error bound analysis*

First, we analyze the performance advantages via a discussion of their error bounds. The following lemmas are first introduced before deriving theoretical errors.

Lemma B1. The solution $\hat{\mathbf{\Theta}}$ derived via forward substitution satisfies $(\mathbf{L} + \mathbf{\Gamma}_1)\,\hat{\mathbf{\Theta}} = \mathbf{Y}, |\mathbf{\Gamma}_1| \leqslant C_m u|\mathbf{L}| + \mathcal{O}(u^2)$, where $\mathbf{L} \in \mathbb{R}^{m\times m}$ is a lower triangular matrix, C_m denotes a constant positively correlated with m, and $\mathcal{O}(u^2)$ represents an error tail term.

Algorithm B3 Handling concept drift

Input: Stream data $\{\mathbf{s}_k := (\mathbf{x}_k, \mathbf{y}_k) \mid k = 1, 2, \cdots, n\}$, parameters n_1, n_2, n_3, n_4, λ, and decay factor μ.
Output: Prediction results $\{\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, \cdots, \hat{\mathbf{y}}_\mathbf{n}\}$.
1: Initialize $\boldsymbol{\Theta}^{(0)} = \mathbf{0}$, $\mathbf{P}^{(0)} = \mathbf{0}$
2: **for** $k = 1, 2, \cdots, n$ **do**
3: Receive instance: $\mathbf{x}_k$
4: Obtain broad feature $\mathbf{a}_k$ via Eq. (B1), Eq. (B2), and Eq. (B3)
5: Predict $\hat{\mathbf{y}}_k^\top = \mathbf{a}_k^\top \boldsymbol{\Theta}^{(k-1)}$ and then reveal $\mathbf{y}_k$
6: $\mathbf{P}^{(k)} = \mu\mathbf{P}^{(k-1)} + \mathbf{a}_k\mathbf{a}_k^\top$, $\mathbf{L}^{(k)} = \text{Chol}(\mathbf{P}^{(k)} + \lambda\mathbf{I})$
7: Obtain $\boldsymbol{\Theta}^{(k)}$ by Algorithm B1 and Eq. (B8).
8: **end for**

The detailed proof can be referred to [33]. Lemma B1 states that its solution satisfies a slightly perturbed system. Also, each entry in the perturbation matrix $\boldsymbol{\Gamma}_1$ is much smaller than the corresponding element in $\mathbf{L}$.

Remark 4. $\forall x \in \mathbb{R}$, its floating-point representation is given by $fl(x) = x(1+\gamma)$, where γ is the floating-point error and its upper bound, unit roundoff u, (i.e., $|\gamma| \leqslant u$) can be defined as $u = \frac{1}{2}\times$(gap between 1 and next largest floating point number). To be specific, the unit roundoff for IEEE single format is about 10^{-7} and for double format is about 10^{-16}.

Lemma B2. The solution $\hat{\boldsymbol{\Theta}}$ derived via backward substitution satisfies $\left(\mathbf{L}^\top + \boldsymbol{\Gamma}_2\right)\hat{\boldsymbol{\Theta}} = \mathbf{Y}, |\boldsymbol{\Gamma}_2| \leqslant C_m u|\mathbf{L}^\top| + \mathcal{O}(u^2)$, where $\mathbf{L}^\top$ is an upper triangular matrix.

The proof also refers to [33].

Recall that the incremental weight $\boldsymbol{\Delta\Theta}^{(k)}$ in Online-BLS is solved via Eq. (B9) and Eq. (B10). Therefore, its error bound is given in Theorem B1. For brevity, we omit the superscript (k) in this section.

Theorem B1. Let $\hat{\mathbf{L}}$ be the estimated Cholesky factor $\mathbf{L}$, we have $\hat{\mathbf{L}}\hat{\mathbf{L}}^\top\boldsymbol{\Delta}\hat{\boldsymbol{\Theta}} = \left(\mathbf{L}\mathbf{L}^\top + \boldsymbol{\Gamma}\right)\boldsymbol{\Delta}\hat{\boldsymbol{\Theta}} = \mathbf{B}$ with $|\mathbf{B} - \mathbf{L}\mathbf{L}^\top\boldsymbol{\Delta}\hat{\boldsymbol{\Theta}}| = |\boldsymbol{\Gamma}||\boldsymbol{\Delta}\hat{\boldsymbol{\Theta}}| \leqslant C_m u|\mathbf{L}||\mathbf{L}^\top||\boldsymbol{\Delta}\hat{\boldsymbol{\Theta}}| + \mathcal{O}(u^2)$.

Proof. From Lemmas B1 and B2, we have $(\mathbf{L}+\boldsymbol{\Gamma}_1)\hat{\mathbf{C}} = \mathbf{B}, |\boldsymbol{\Gamma}_1| \leqslant C_m u|\mathbf{L}| + \mathcal{O}(u^2), (\mathbf{L}^\top + \boldsymbol{\Gamma}_2)\boldsymbol{\Delta}\hat{\boldsymbol{\Theta}} = \hat{\mathbf{C}}, |\boldsymbol{\Gamma}_2| \leqslant C_m u|\mathbf{L}^\top| + \mathcal{O}(u^2)$, and thus $\mathbf{B} = (\mathbf{L} + \boldsymbol{\Gamma}_1)\left(\mathbf{L}^\top + \boldsymbol{\Gamma}_2\right)\boldsymbol{\Delta}\hat{\boldsymbol{\Theta}} = \left(\mathbf{L}\mathbf{L}^\top + \mathbf{L}\boldsymbol{\Gamma}_2 + \boldsymbol{\Gamma}_1\mathbf{L}^\top + \boldsymbol{\Gamma}_1\boldsymbol{\Gamma}_2\right)\boldsymbol{\Delta}\hat{\boldsymbol{\Theta}}$. Thus, we find $(\mathbf{L}\mathbf{L}^\top + \boldsymbol{\Gamma})\boldsymbol{\Delta}\hat{\boldsymbol{\Theta}} = \mathbf{B}$ with $|\boldsymbol{\Gamma}| \leqslant |\mathbf{L}||\boldsymbol{\Gamma}_2| + |\boldsymbol{\Gamma}_1||\mathbf{L}^\top| + \mathcal{O}(u^2) \leqslant C_m u|\mathbf{L}||\mathbf{L}^\top| + \mathcal{O}(u^2)$ such that $|\mathbf{B} - \mathbf{L}\mathbf{L}^\top\boldsymbol{\Delta}\hat{\boldsymbol{\Theta}}| = |\boldsymbol{\Gamma}||\boldsymbol{\Delta}\hat{\boldsymbol{\Theta}}| \leqslant C_m u|\mathbf{L}||\mathbf{L}^\top||\boldsymbol{\Delta}\hat{\boldsymbol{\Theta}}| + \mathcal{O}(u^2)$. Thus, we can conclude Theorem B1.

As for the previous I-BLS, BLS-CIL, and RI-BLS, none of them avoids computing the inverse of a large matrix, which means to obtain $\boldsymbol{\Delta\Theta} = fl\left((\mathbf{A}^\top\mathbf{A} + \lambda\mathbf{I})^{-1}\mathbf{B}\right)$ with rounding errors. Then, we have $\boldsymbol{\Delta}\hat{\boldsymbol{\Theta}} = \left((\mathbf{A}^\top\mathbf{A} + \lambda\mathbf{I})^{-1} + \boldsymbol{\Gamma}\right)\mathbf{B}, |\boldsymbol{\Gamma}| \leqslant C_m u|(\mathbf{A}^\top\mathbf{A}+\lambda\mathbf{I})^{-1}| + \mathcal{O}(u^2)$, and $(\mathbf{A}^\top\mathbf{A}+\lambda\mathbf{I})\boldsymbol{\Delta}\hat{\boldsymbol{\Theta}} = \mathbf{B} + (\mathbf{A}^\top\mathbf{A}+\lambda\mathbf{I})\boldsymbol{\Gamma}\mathbf{B}$. Thus, their error bound is $|\mathbf{B} - (\mathbf{A}^\top\mathbf{A}+\lambda\mathbf{I})\boldsymbol{\Delta}\hat{\boldsymbol{\Theta}}| \leqslant C_m u|(\mathbf{A}^\top\mathbf{A} + \lambda\mathbf{I})||(\mathbf{A}^\top\mathbf{A} + \lambda\mathbf{I})^{-1}||\mathbf{B}| + \mathcal{O}(u^2) \leqslant C_m u|\mathbf{L}||\mathbf{L}^\top||(\mathbf{A}^\top\mathbf{A} + \lambda\mathbf{I})^{-1}||\mathbf{B}| + \mathcal{O}(u^2)$.

Since $\mathbf{A}$ is generally an ill-conditioned matrix for BLS, there exists $|\boldsymbol{\Delta}\hat{\boldsymbol{\Theta}}| \ll |(\mathbf{A}^\top\mathbf{A} + \lambda\mathbf{I})^{-1}||\mathbf{B}|$. In other words, for a single model update during online learning tasks, the error bound of our proposed Online-BLS is significantly lower than that of existing incremental BLS methods.

Appendix B.6.2 *Time complexity analysis*

In this section, we consider only the multiplication and addition operations in matrix multiplication, while other operations (e.g., vector subtraction and matrix-scalar multiplication) are omitted. From Algorithm B2, it can be observed that the primary computational overhead arises in Steps 6 and 7. Specifically, a single Givens transformation in Step 6 requires approximately $8m$ floating-point operations (see Eq. (B16) and (B17)). Since a total of m Givens transformations are required, the total floating-point operations and asymptotic time complexity of Step 6 are $8m^2$ and $\mathcal{O}(m^2)$, respectively. Before Step 7, we first compute $\mathbf{B}^{(k)} = \mathbf{a}_k\left(\mathbf{y}_k^\top - \mathbf{a}_k^\top\boldsymbol{\Theta}^{(k-1)}\right)$, which requires $4mt$ floating-point operations with a time complexity $\mathcal{O}(mt)$. Step 7 executes Algorithm B1 to attain $\boldsymbol{\Delta\Theta}^{(k)}$ and derives $\boldsymbol{\Theta}^{(k)}$ using Eq. (B8). The computational cost of Algorithm B1 lies in Steps 5 and 10, while that of Eq. (B8) is negligible. The total number of floating-point operations and time complexity for Steps 5 and 10 in Algorithm C1 are $2m^2t$ and $\mathcal{O}(m^2t)$, respectively. Thus, our Online-BLS has a total approximate floating-point operation count and time complexity of $2m^2t + 8m^2 + 4mt$ and $\mathcal{O}(m^2t + m^2 + mt)$, respectively. The time complexities of I-BLS, BLS-CIL, and RI-BLS are $\mathcal{O}(km + m^2 + m^3 + mt)/\mathcal{O}(km + m + mt)$, $\mathcal{O}(m^3 + m^2t + m^2 + mt + m)$, and $\mathcal{O}(m^3 + m^2t + m^2 + mt)$, respectively. From these results, several noteworthy observations can be drawn. First, the time complexity of I-BLS contains the primary term of k. That is, as the number of samples received by I-BLS increases, its online update time overhead increases linearly. This is quite terrible for online learning tasks, which tend to have a lot or even an infinite number of samples. Second, both BLS-CIL and RI-BLS eliminate the dependence on k in their time complexities. However, their complexities are dominated by the $\mathcal{O}(m^3)$ term. This cubic complexity arises from the necessity of performing a full matrix inversion or decomposition on an $m \times m$ matrix at each update step, incurring a severe computational bottleneck for large m. Finally, the most remarkable advantage of Online-BLS is the elimination of the $\mathcal{O}(m^3)$ term. This is attributed to our EOUS and EWEA methods, which update the Cholesky factor online and utilize forward-backward substitution to replace matrix inversion, respectively. Thus, our Online-BLS algorithm is remarkably efficient and well-suited for online machine learning tasks.

Appendix C Experiments

The most common prequential test-then-train experimental paradigm is adopted unless otherwise specified. All experiments are conducted on a machine running Ubuntu 20.04 with an AMD EPYC 7302 CPU.

Table C1 Descriptions of datasets. IS: image segment; IA: input attributes; C: classes; DP: data points (train/test).

Dataset	IA	C	DP	Type	Dataset	IA	C	DP	Type
IS	19	7	2,310	Stationary	Gisette	5,000	2	6,000/1,000	Stationary
USPS	256	10	9,298	Stationary	Epsilon	2,000	2	400,000/100,000	Stationary
Letter	16	26	20,000	Stationary	Hyperplane	20	2	100,000	Concept drift
Adult	14	2	45,222	Stationary	SEA	3	2	100,000	Concept drift
Shuttle	8	7	58,000	Stationary	Electricity	6	2	45,312	Concept drift
MNIST	784	10	70,000	Stationary	CoverType	54	7	581,012	Concept drift

Appendix C.1 Datasets and metrics

A total of 12 streaming datasets are used in this study, as summarized in Table C1. Among them, there are 8 stationary datasets and 4 non-stationary datasets with concept drift. Next, we introduce them in detail below. **Image Segment**: The Image Segment dataset is an image classification benchmark described by 19 high-level numerical attributes. Its samples are randomly sampled from a dataset containing 7 outdoor images. The images are manually segmented, with each pixel assigned a class label. Each instance contains a 3×3 region, resulting in a total of $2,310$ samples distributed across seven classes *brickface*, *sky*, *foliage*, *cement*, *window*, *path*, and *grass*. **USPS**: The USPS dataset is a benchmark for handwritten digit classification, consisting of $9,298$ samples from 10 classes ranging from 0 to 9. Each sample is a 16×16 grayscale image. To facilitate BLS modeling, we flatten each sample into a one-dimensional vector. **Letter**: The Letter dataset is a benchmark for capital letter classification in the English alphabet. It contains $20,000$ samples. Each sample contains 16 primary numeric attributes (statistical moments and edge counts), which are then scaled to an integer value range of 0 to 15. **Adult**: The Adult dataset is used for a binary classification task, aiming to predict whether a person earns more than $50k$ a year based on 14 attributes. It has both numerical and categorical attributes. We first convert categorical attributes to integers that the model can handle easily. Then, samples with missing values are removed, and the training and test sets are merged. Eventually, the Adult dataset has a total of $45,222$ samples. **Shuttle**: The Shuttle dataset is a seven-class classification dataset, including *Rad Flow*, *Fpv Close*, *Fpv Open*, *High*, *Bypass*, *Bpv Close*, and *Bpv Open*. There are 9 numerical attributes, the first of which represents time. Thus, we remove the first column of attributes and use the remaining 8 attributes to predict one of the seven categories. **MNIST**: The MNIST dataset is a handwritten digit recognition benchmark. Each sample is a grayscale image with 28×28 pixels. The training and test sets are merged to yield a dataset containing $70,000$ samples uniformly distributed from digits 0 to 9. To adapt this dataset for our model, each sample is flattened into a vector, and each attribute is normalized to fall within the range $[0, 1]$. **Gisette**: The Gisette dataset is a handwritten digit recognition problem that involves distinguishing between the highly confusable digits 4 and 9. By combining raw pixel values with additional noise features, it reaches a feature dimensionality of $5,000$. There are a total of $6,000$ training samples and $1,000$ test samples. **Epsilon**: The Epsilon dataset is a popular binary classification task, whose features have been suitably preprocessed by its publisher. It contains $400,000$ training instances and $100,000$ test instances, each comprising $2,000$ numerical features.

For non-stationary datasets, we choose two synthetic datasets and two real-world datasets. The synthetic datasets, Hyperplane and SEA, are generated using the River package. The two real-world datasets originate from actual applications and often exhibit complex and unpredictable concept drift. **Hyperplane**: The Hyperplane dataset is an artificial binary classification task. Its task is to divide points in a d-dimensional space into two classes by a $(d-1)$-dimensional hyperplane. A sample is labelled positive if it satisfies $\sum_{i=1}^{d} w_i x_i > w_0$ and negative if the opposite is true. By smoothly varying the parameters of the classification hyperplane, concept drift can be added to the generated data stream. Specifically, following [20], we generate a data stream containing 20 features. Its noise level and drift magnitude are set to 1% and 0.5%, respectively. **SEA**: The SEA dataset is also a synthetic binary classification benchmark. Each sample contains three features, of which only the first two are relevant to its label. If the sum of the first two features of a sample exceeds a certain threshold, then it is a positive example; otherwise, it is a negative example. There are four candidate thresholds. Concept drift is introduced by switching thresholds at the $25,000$-th, $50,000$-th, and $75,000$-th data points. Moreover, following [20], the noise level we introduce in the process of data stream generation is 10%. **Electricity**: The Electricity dataset is a binary classification benchmark, whose goal is to predict whether the electricity price of Australian New South Wales will increase or decrease. The prices in this market are dynamic and influenced by supply and demand, with updates occurring every five minutes. Each sample originally contains 8 attributes. Following [20], we delete the attributes of date and time located in the first two columns and keep the remaining 6 attributes. This dataset contains $45,312$ instances, and its concepts fluctuate over time. **CoverType**: The CoverType dataset is used to predict 7 forest cover types from 54 features of each instance. The feature types include integer and category types. First, we convert the categorical type to an integer for ease of handling. Then, the CoverType dataset has $581,012$ instances with unknown concept drift.

Next, we introduce the seven metrics used in this study. **Online cumulative accuracy (OCA)**: The OCA is used to evaluate the overall performance of online learning models and is defined as $\text{OCA} = \frac{1}{n}\sum_{k=1}^{n} \mathbb{I}_{(\hat{\mathbf{y}}_k = \mathbf{y}_k)}$, where $\mathbb{I}$ is the indicator function. $\hat{\mathbf{y}}_k$ and $\mathbf{y}_k$ represent the prediction and ground truth of the k-th instance, respectively. **Online cumulative error (OCE)**: The OCE is used to evaluate the overall error rate during the online learning process and is defined as follows: $\text{OCE} = \frac{1}{n}\sum_{k=1}^{n} \mathbb{I}_{(\hat{\mathbf{y}}_k \neq \mathbf{y}_k)}$. Note that the sum of the OCA and the OCE at the same time step is always 1. **Balanced accuracy (BACC)**: The BACC treats all categories equally, making it more reliable for data streams with imbalanced classes. It is defined as follows: $\text{BACC}_k = \left(\sum_{i=1}^{c} n_{c_i}^T / n_{c_i}\right)/c$, where $n_{c_i}^T$ and n_{c_i} denote the number of correctly predicted instances and instances belonging to the i-th class, respectively. c represents the total number of classes, and k is the current time step. **Average balanced accuracy (AVRBACC)**: The AVRBACC is the average of the

Table C2 Experimental results (mean±standard deviation %) under three metrics of OCA, Macro F1 and MCC on the first six stationary datasets. Note: Bold indicates the best result, while underlining denotes the second-best result.

Method	Metric	IS	USPS	Letter	Adult	Shuttle	MNIST
I-BLS	OCA (↑)	73.6±19.80	61.1±40.10	59.4±27.30	71.3±13.60	86.6±20.10	83.5±24.40
	Macro F1 (↑)	71.6±22.20	58.9±42.00	58.0±28.40	46.7±05.84	37.1±09.73	82.7±26.20
	MCC (↑)	69.5±23.10	57.4±43.70	57.9±28.40	12.7±04.21	70.2±30.50	81.6±27.00
BLS-CIL	OCA (↑)	83.6±0.423	**93.7±0.239**	79.7±0.711	75.4±0.020	95.9±0.299	89.9±0.180
	Macro F1 (↑)	83.0±0.487	**93.1±0.279**	79.3±0.753	44.7±0.144	51.4±02.78	89.7±0.189
	MCC (↑)	81.1±0.472	**93.0±0.268**	79.0±0.737	07.6±0.292	88.2±0.872	88.8±0.200
RI-BLS	OCA (↑)	89.9±0.668	92.8±0.228	87.6±0.541	72.0±0.856	90.2±1.100	89.9±0.246
	Macro F1 (↑)	90.0±0.662	92.1±0.240	87.6±0.548	**60.7±0.441**	46.3±1.390	89.8±0.253
	MCC (↑)	88.2±0.778	91.9±0.256	87.1±0.562	**21.6±1.050**	75.3±2.530	88.8±0.273
Ours	OCA (↑)	**90.8±0.229**	93.6±0.386	**88.9±0.137**	**76.4±0.051**	**98.2±0.026**	**92.5±0.105**
	Macro F1 (↑)	**90.8±0.246**	93.0±0.380	**88.9±0.132**	51.5±0.246	**77.7±2.650**	**92.4±0.107**
	MCC (↑)	**89.2±0.268**	92.9±0.429	**88.5±0.142**	18.6±0.375	**94.9±0.076**	**91.7±0.117**

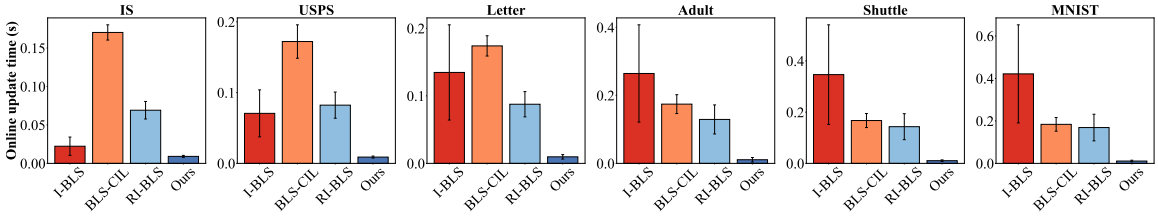

Figure C1 The mean and standard deviation of online update time on the first six stationary datasets.

balanced accuracy over all time steps, reflecting the overall performance of the online learning model throughout the learning process. It has the following definition: $\text{AVRBACC} = \sum_{k=1}^{n} \text{BACC}_k/n$, where n is the number of instances received by the online learning model. **Macro F1 score (Macro F1)**: Since most of the datasets we use are class-balanced, we use Macro F1, and its definition is $\text{Macro F1} = \frac{1}{c}\sum_{i=1}^{c}\left(2\frac{R_i \times P_i}{R_i + P_i}\right)$, where R_i and P_i denote the recall and precision of class i, respectively. **Matthews correlation coefficient (MCC)**: The MCC is generally considered a better metric than F1 score and accuracy. Its essence is a correlation coefficient value between -1 and 1. The coefficients $+1$, 0, and -1 represent a perfect prediction, an average random prediction, and an inverse prediction, respectively. Taking binary classification as an example, its formula is $\text{MCC} = (\text{TP} \times \text{TN} - \text{FP} \times \text{FN})/\sqrt{(\text{TP}+\text{FP})(\text{TP}+\text{FN})(\text{TN}+\text{FP})(\text{TN}+\text{FN})}$, where TP, TN, FP, and FN denote the number of true positives, ture negatives, false positives, and false negatives, respectively. We call sklearn's function to acquire MCC in our experiments. **Test accuracy (TA)**: To compute TA, it is necessary to partition our dataset into training and test sets. The model is trained on the training set in an online manner, and then its accuracy on the test set is calculated.

Appendix C.2 Experiments on stationary datasets

Appendix C.2.1 *Comparison with incremental BLS methods*

In this section, we compare Online-BLS with existing incremental BLS algorithms on the first six stationary datasets. To ensure fairness, we set the parameters common to all comparison methods and Online-BLS to be the same. Specifically, n_1, n_2, n_3, and n_4 were set to 10, 10, 1,000, and 1, respectively. The regularization parameter λ was set to $1e-8$ for I-BLS, RI-BLS, and Online-BLS. For BLS-CIL, λ_1 and λ_2 were set to 1 in its original paper, which is unstable in our online learning task. Therefore, we tuned both to 0.1 to achieve better performance. To draw credible conclusions, we repeated each experiment 10 times by changing the order of streaming data and the initialization model parameters.

The mean and standard deviation of the final OCA, Macro F1, and MCC over 10 independent runs are shown in Table C2. First, our proposed Online-BLS exceeds all baselines on OCA for 5 out of 6 datasets. For the USPS dataset, the OCA of Online-BLS is also very close to the highest one. Also, our Online-BLS exhibits low standard deviations, indicating excellent stability across different experimental trials. As for Macro F1 and MCC, we can observe similar experimental phenomena. To our surprise, our Online-BLS outperforms the second place by 26.3% and 6.7% on the Macro F1 and MCC metrics of the Shuttle dataset, respectively. To prove the efficiency of Online-BLS from a numerical perspective, we define the time required for a model to update its parameters after receiving an instance as the online update time. The mean and standard deviation of the online update time for each method throughout its entire online learning period are shown in Figure C1. First, our Online-BLS has the shortest online update time, indicating that our method is very efficient. Second, we find that the standard deviation of the online update time of I-BLS increases as the number of instances grows. These results are also consistent with the time complexity analysis in Appendix B.6.2. Figure C2 shows the convergence curves of all the methods. The sum of OCE and OCA is one. Online-BLS consistently converges rapidly and outperforms all baselines at almost all time steps on these six datasets, further reaffirming the effectiveness of our method. Furthermore,

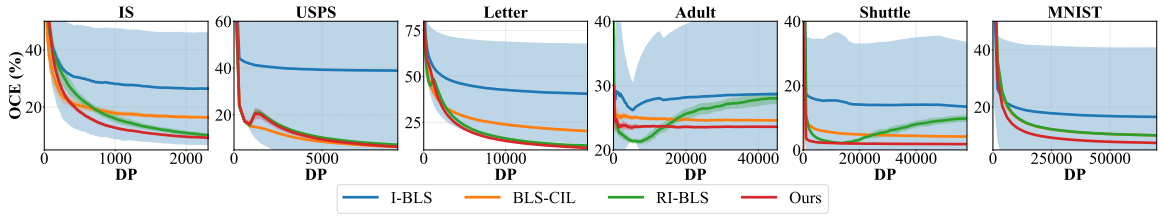

Figure C2 The convergence curves of mean and standard deviation of OCE on the first six stationary datasets.

Table C3 The average and standard deviation (%) of TA on the Gisette and Epsilon datasets. Note: The standard deviation is not available from reference [12].

Dataset	ADA-DIAG	RADAGRAD	FD-SON	ADA-FFD (M)	ADA-FFD (P)	S-ADA	FTFSL	**Ours**
Gisette	96.71	96.90	97.14	97.28	97.18	97.40	97.63	**98.00±0.32**
Epsilon	88.89	87.98	88.96	89.03	88.84	89.04	89.23	**89.82±0.02**

Table C2 and Figure C2 show that I-BLS exhibits a large standard deviation in performance. This instability is because the online learning algorithm of I-BLS updates its inverse matrix using Greville's theory, which incurs considerable numerical error. The multi-step update process of online learning further amplifies this error accumulation, leading to substantial performance fluctuations across different trials. On the contrary, our Online-BLS avoids matrix inversion via employing Cholesky factorization and forward-backward substitution, thereby achieving higher numerical stability.

Appendix C.2.2 *Comparison with matrix decomposition-based OL methods*

In this part, we focused on comparing Online-BLS with SOTA matrix decomposition-based OL methods, including: ADA-DIAG [7], RADAGRAD [8], FD-SON [9], ADA-FFD (M) [10], ADA-FFD (P) [10], S-ADA [11], and FTFSL [12]. The grid search strategy was employed to determine the optimal network parameters of Online-BLS (n_1, n_2, n_3, n_4) on $\{10, 20, \cdots, 50\} \times \{10, 20, \cdots, 50\} \times \{1000, 2000, \cdots, 5000\} \times \{1\}$. The optimal regularization coefficient was searched from $\{1e-2, 1e-4, \cdots, 1e-10\}$. Following FTFSL [12], experiments in this section are conducted on Gisette and Epsilon datasets. Each dataset was divided into training and test sets. The model was learned from the training set samples in an online manner, and then its test accuracy was calculated. To eliminate random interference, we independently ran 10 experiments by altering the order of training set samples and model initialization weights. The mean and standard deviation of TA on the Gisette and Epsilon datasets are reported in Table C3, with results for all comparison methods cited from [12]. From Table C3, we can observe a distinct advantage of Online-BLS over other matrix decomposition-based OL methods, which again proves the superiority of our Online-BLS approach.

Appendix C.2.3 *Universality for other random neural networks*

To validate the versatility of our proposed method, we discussed its applicability to other random neural networks. Specifically, we considered two kinds of random networks: random vector functional link neural network (RVFL) and extreme learning machine (ELM). Four methods, i.e., RVFL (offline) [34], OLRVFL [35], ELM (offline) [36], and OS-RELM [37], were chosen as comparison methods. The RVFL and ELM incorporating our proposed algorithm were termed Online-RVFL and Online-ELM, respectively. For fairness, the hyperparameters of all methods were kept consistent. To be specific, the hidden layer dimension and regularization coefficient for all six methods are set to $1,000$ and $1e-8$, respectively. Note that RVFL (offline) and ELM (offline) are essentially batch learning algorithms. To conduct the test-then-train experimental paradigm and compute their OCA, both models first made predictions using their previous models upon the arrival of a sample, then retrained themselves using all samples that had arrived. Each experimental case was run 10 times with different sample sequences and initialization parameters to eliminate randomness. The experimental results for RVFL and ELM are shown in Tables C4 and C5, respectively.

From Tables C4 and C5, several observations can be drawn. First, RVFL (offline) and ELM (offline) are expected to approximate the performance upper bounds of their online counterparts, as both methods retrain from scratch whenever

Table C4 Experimental results (mean±standard deviation %) of our proposed Online-RVFL on the first six stationary datasets. OUT is short for online update time ($\times 10^{-2}s$). Note: Bold indicates the best result.

Method	RVFL (offline)		OLRVFL		Online-RVFL	
Dataset	OCA (↑)	OUT (↓)	OCA (↑)	OUT (↓)	OCA (↑)	OUT (↓)
IS	90.7±0.395	6.43±1.380	90.7±0.389	1.65±0.65	**90.9±0.329**	**1.03±0.39**
USPS	92.7±0.313	18.9±8.240	90.3±0.341	2.87±0.51	**93.3±0.428**	**1.36±0.37**
Letter	88.5±0.197	23.3±10.99	88.0±0.131	1.90±0.51	**89.0±0.128**	**0.97±0.38**
Adult	**77.1±0.176**	45.1±22.68	75.5±0.865	1.81±0.45	76.4±0.051	**0.87±0.33**
Shuttle	**98.7±0.031**	51.3±26.94	96.4±4.180	1.86±0.42	98.2±0.018	**0.96±0.30**
MNIST	92.4±0.065	148.4±74.04	92.1±0.126	5.29±0.56	**93.0±0.096**	**1.59±0.36**

Table C5 Experimental results (mean±standard deviation %) of our proposed Online-ELM on the first six stationary datasets. OUT is short for online update time ($\times 10^{-2}s$). Note: Bold indicates the best result.

Method	ELM (offline)		OS-RELM		Online-ELM	
Dataset	OCA (↑)	OUT (↓)	OCA (↑)	OUT (↓)	OCA (↑)	OUT (↓)
IS	90.8±0.308	6.11±2.350	90.9±0.339	1.72±0.44	**90.9±0.334**	**0.75±0.24**
USPS	93.1±0.208	14.0±6.030	91.0±0.263	1.71±0.45	**93.7±0.274**	**0.79±0.29**
Letter	88.4±0.115	20.4±9.230	87.9±0.152	1.64±0.47	**88.9±0.178**	**0.95±0.35**
Adult	**77.1±0.046**	40.1±19.86	76.4±0.076	1.38±0.42	76.4±0.051	**0.55±0.27**
Shuttle	**98.7±0.029**	47.8±25.51	98.2±0.031	1.43±0.46	98.2±0.019	**0.94±0.36**
MNIST	90.6±0.109	71.6±38.53	90.6±0.128	1.67±0.45	**92.4±0.127**	**0.75±0.28**

Table C6 Experimental results (mean±standard deviation %) on the Hyperplane dataset. OUT is short for online update time ($\times 10^{-2}s$). The n_1, n_2, n_3 and λ of our method are 10, 10, 100 and 0.01, respectively. Note: Bold indicates the best result, while underlining denotes the second-best result. The standard deviation is not available from reference [20].

Method	OCA (↑)	BACC (↑)	AVRBACC (↑)	OUT (↓)
HT	85.1	86.7	85.4	3.59±2.54
ARF	76.8	76.2	76.2	4.32±3.09
HBP	87.0	<u>91.0</u>	86.6	4.96±0.56
ADL	77.3	80.7	77.2	5.98±3.54
CAND	88.1	89.8	87.9	-
EODL	<u>89.5</u>	90.5	<u>89.2</u>	18.58±17.55
Ours (Adaptive)	**92.6±0.062**	**92.6±0.062**	**91.7±0.074**	**0.36±0.73**

a new sample arrives. However, to our surprise, our proposed Online-RVFL and Online-ELM outperform their offline counterparts on most datasets, which is attributed to our method providing more favorable error bounds. For the Adult and Shuttle datasets, Online-RVFL and Online-ELM also surpass or match OLRVFL and OS-RELM, respectively. Second, Online-RVFL and Online-ELM are the most efficient against all comparison methods, since both methods achieve the lowest online update time. This efficiency is primarily attributed to our EOUS.

Appendix C.3 Experiments on concept drift datasets

In this section, the effectiveness and efficiency of our Online-BLS were validated in the concept drift scenario. To learn non-stationary data streams, we adopted a forgetting factor to force our Online-BLS to assign smaller weights to historical memory. Specifically, we simply set it to 0.99 for all datasets. We compared our method with two classes of baselines used for data stream classification with concept drift. The first category was online machine learning algorithms, including HT [2] and ARF [3]. The second one was SOTA online deep learning models such as HBP [16], ADL [17], CAND [18], ATNN [19], and EODL [20]. Beyond OCA, four other metrics were considered: BACC, AVRBACC, Macro F1, and MCC. It is worth noting that all of the above metrics, except OCE, are such that larger values represent better model performance. Furthermore, we also report the average online update time to validate the efficiency of our approach. Tables C6, C7, C8, and C9 demonstrate the final results on the Hyperplane, SEA, Electricity, and CoverType datasets, respectively.

To make the experimental conclusions more reliable, experimental results of all baseline methods were directly cited from the corresponding papers, while those of Online-BLS with forgetting factor (denoted with Adaptive) were averaged over 10 independent experiments. For a fair comparison of online update time, we executed the comparison methods on our computing platform using an open-source code repository, with parameter settings identical to those reported in the original papers. First, ours (Adaptive) surpasses typical online machine learning algorithms with a large gap, which benefits from the random feature mapping of BLS that can extract effective broad features. Additionally, ours (Adaptive) outperforms existing online deep learning methods across almost all metrics. We conjecture that this is because the derivation of our

Table C7 Experimental results (mean±standard deviation %) on the SEA dataset. OUT is short for online update time ($\times 10^{-2}s$). The n_1, n_2, n_3 and λ of our method are 5, 5, 50 and 0.01, respectively. Note: Bold indicates the best result, while underlining denotes the second-best result. The standard deviation is not available from reference [20].

Method	OCA (↑)	BACC (↑)	AVRBACC (↑)	OUT(↓)
HT	81.1	82.0	79.4	2.92±1.84
ARF	<u>83.8</u>	**86.5**	81.5	3.03±1.82
HBP	82.1	82.5	78.7	6.23±1.44
ADL	60.1	49.8	49.9	4.21±29.31
CAND	79.7	82.8	76.7	-
EODL	82.5	83.2	79.4	23.47±17.61
Ours (Adaptive)	**85.2±0.156**	<u>83.3±0.180</u>	**82.6±0.183**	**0.17±0.16**

Table C8 Experimental results (mean±standard deviation %) on the Electricity dataset. OUT is short for online update time ($\times 10^{-2}s$). The n_1, n_2, n_3 and λ of our method are 10, 10, 100 and $1e-8$, respectively. Note: Bold indicates the best result, while underlining denotes the second-best result. The standard deviation is not available from references [19] and [20].

Method	OCA (↑)/MCC (↑)	BACC (↑)/Macro F1 (↑)	AVRBACC (↑)	OUT (↓)
HT	77.0/-	74.3/-	74.7	1.14±0.68
ARF	72.8/-	76.1/-	69.3	1.30±0.71
HBP	74.8/-	79.6/-	70.3	4.51±0.49
ADL	74.9/-	81.6/-	70.7	2.48±1.80
CAND	78.5/-	<u>83.7</u>/-	<u>76.4</u>	-
ATNN	<u>83.6</u>/<u>66.0</u>	-/<u>83.0</u>	-	1.68±1.56
EODL	78.0/-	83.5/-	76.2	8.99±4.80
Ours (Adaptive)	**86.8±0.038/72.9±0.079**	**86.2±0.047/86.4±0.041**	**86.9±0.059**	**0.25±0.28**

Table C9 Experimental results (mean±standard deviation %) on the CoverType dataset. OUT is short for online update time ($\times 10^{-2}s$). The n_1, n_2, n_3 and λ of our method are 10, 10, 400 and $1e-8$, respectively. Note: Bold indicates the best result, while underlining denotes the second-best result. The standard deviation is not available from references [19] and [20].

Method	OCA (↑)/MCC (↑)	BACC (↑)/Macro F1 (↑)	AVRBACC (↑)	OUT (↓)
HT	80.1/-	73.0/-	70.4	33.12±22.05
ARF	83.7/-	80.9/-	64.2	36.06±26.81
HBP	91.2/-	82.5/-	74.7	7.25±1.54
ADL	90.5/-	86.7/-	76.8	60.44±37.00
CAND	92.8/-	85.4/-	79.2	-
ATNN	92.7/<u>89.0</u>	-/<u>87.0</u>	-	2.91±2.72
EODL	<u>93.5</u>/-	**90.0**/-	<u>81.0</u>	125.54±71.75
Ours (Adaptive)	**94.3±0.011/90.9±0.019**	<u>89.7±0.040</u>/**90.0±0.030**	**88.2±0.045**	**0.56±0.47**

method is based on a closed-form solution, which provides a strong guarantee of the optimality of the solution updated online. Finally, the OUT of our method is much smaller than that of the comparison method, which indicates that our method requires lower computational overhead and is time-efficient.

Appendix C.4 Ablation study

Our framework has two main parts, EWEA and EOUS. Amongst them, EWEA is an inseparable part of Online-BLS. Thus, we first examined the validity of EOUS in this section. The results of ablation experiments are shown in Table C10. From Table C10, we can find that EOUS can significantly reduce online update time while maintaining model performance. Hence, we can conclude that EOUS is an efficient algorithm that can significantly reduce the online update overhead of Online-BLS. To verify the effectiveness of our proposed adaptive mechanism, we eliminated the forgetting factor from our approach and conducted experiments on four non-stationary datasets. The hyperparameters and experimental settings were consistent with Section Appendix C.3. Figure C3 shows the convergence curves of Online-BLS with and without forgetting factors. From Figure C3, we find that the average OCE of Online-BLS increases significantly as concept drift occurs, while ours (Adaptive) can maintain a low error rate. Thus, we conclude that by adding a forgetting factor to the historical data, our framework can handle the non-stationary data stream classification task well.

Appendix C.5 Parameter sensitivity analysis

Online-BLS involves several tuning parameters, including n_1, n_2, n_3, n_4, λ, and μ. Typically, setting $n_4 = 1$ is sufficient. The remaining 5 parameters were divided into four groups to investigate their impact on model performance. For n_1, n_2,

Table C10 Experimental results (mean±standard deviation %) of ablation study. OUT: online update time ($\times 10^{-2}s$). Note: Bold indicates the best result

Method	Metric	IS	USPS	Letter	Adult	Shuttle	MNIST
Ours w/o EOUS	OCA (↑)	90.8±0.229	93.6±0.386	88.9±0.137	76.4±0.051	98.2±0.026	92.5±0.105
	Macro F1 (↑)	90.8±0.246	93.0±0.380	88.9±0.132	51.5±0.246	77.7±2.650	92.4±0.107
	MCC (↑)	89.2±0.268	92.9±0.429	88.5±0.142	18.6±0.375	94.9±0.076	91.7±0.117
	OUT (↓)	2.84±2.410	2.70±2.610	2.46±2.460	2.67±2.540	2.67±2.640	2.80±2.700
Ours	OCA (↑)	90.8±0.229	93.6±0.386	88.9±0.137	76.4±0.051	98.2±0.026	92.5±0.105
	Macro F1 (↑)	90.8±0.246	93.0±0.380	88.9±0.132	51.5±0.246	77.7±2.650	92.4±0.107
	MCC (↑)	89.2±0.268	92.9±0.429	88.5±0.142	18.6±0.375	94.9±0.076	91.7±0.117
	OUT (↓)	**0.91±0.130**	**0.89±0.140**	**0.97±0.330**	**1.07±0.660**	**1.05±0.320**	**1.10±0.390**

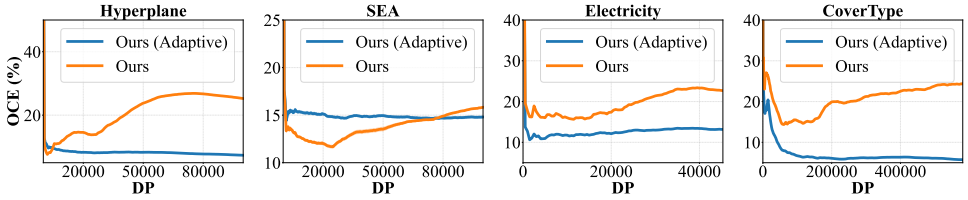

Figure C3 The mean and standard deviation of convergence curves of Online-BLS with and without forgetting factor on non-stationary datasets.

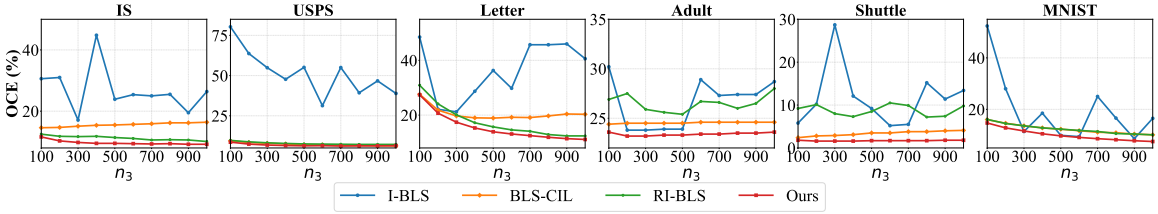

Figure C4 The mean of OCE under different n_3 values.

n_3, and λ, when examining the sensitivity of a specific parameter, the remaining 3 parameters were kept consistent with those in Appendix C.2. When investigating μ, the remaining 4 parameters (i.e., n_1, n_2, n_3, and λ) were kept the same as those in Appendix C.3.

The impact of different n_3 on the average OCE is first shown in Figure C4. Our method consistently achieves the lowest average OCE compared to all baselines on all n_3 cases across the six datasets. Then, we find that the average OCE of I-BLS changes significantly with different n_3. This is because in our experiments, we found that I-BLS is very unstable. That is, the accuracy of I-BLS in some trials is close to that of random guessing cases. Based on the above results, we can safely conclude that Online-BLS is robust to changes of n_3. Hence, we set n_3 to $1,000$ on all datasets and did not adjust it for different datasets. Subsequently, we investigated the impact of different n_1 and n_2 values on model performance, with the experimental results shown in Figure C5. From Figure C5, we can observe that the performance of Online-BLS remains quite stable across a wide range of n_1 and n_2 values. In other words, as long as n_1 and n_2 are not extremely large or small, Online-BLS can achieve good performance. Thus, both n_1 and n_2 in the three incremental BLS algorithms and Online-BLS were set to 10 for simplicity. Third, the impact of different λ values on model performance is provided in Table C11. When λ is adjusted within a given interval, the range of variation for model performance is from 0.1% to 4.4%. Although the optimal λ varies across different datasets, we chose to fix λ to $1e-8$ because it delivers relatively good performance across all datasets. Finally, we discussed the sensitivity of parameter μ across four non-stationary datasets. Table C12 shows the mean and standard deviation of OCA for our method when μ varies across $\{0.80, 0.85, 0.90, 0.95, 0.99\}$. From Table C12, we observe that when the forgetting factor μ is insufficiently large, the model rapidly forgets past valuable knowledge, leading to a degradation in its performance. On the contrary, as shown in Figure C3, we conclude that the model's performance remains terrible with a forgetting factor of 1. Thus, we set $\mu = 0.99$ here, which is suitable for most datasets.

References

1 Hoi S C H, Sahoo D, Lu J, et al. Online learning: A comprehensive survey. Neurocomputing, 2021, 459: 249–289
2 Hulten G, Spencer L, Domingos P. Mining time-changing data streams. In: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, San Francisco, 2001. 97–106
3 Gomes H M, Bifet A, Read J, et al. Adaptive random forests for evolving data stream classification. Mach. Learn., 2017, 106: 1469–1495
4 Crammer K, Dekel O, Keshet J, et al. Online passive-aggressive algorithms. J. Mach. Learn. Res., 2006, 7: 551–585
5 Rosenblatt F. The perceptron: a probabilistic model for information storage and organization in the brain. Psychol. Rev., 1958, 65: 386
6 Bifet A, Holmes G, Pfahringer B. Leveraging bagging for evolving data streams. In: Proceedings of the 2010th European Conference on Machine Learning and Knowledge Discovery in Databases-Volume Part I, Barcelona, 2010. 135–150
7 Duchi J, Hazan E, Singer Y. Adaptive subgradient methods for online learning and stochastic optimization. J. Mach. Learn. Res., 2011, 12: 2121–2159
8 Krummenacher G, McWilliams B, Kilcher Y, et al. Scalable adaptive stochastic optimization using random projections. In: Proceedings of Advances in Neural Information Processing Systems 29, Barcelona, 2016. 1758–1766
9 Luo L, Chen C, Zhang Z, et al. Robust frequent directions with application in online learning. J. Mach. Learn. Res., 2019,

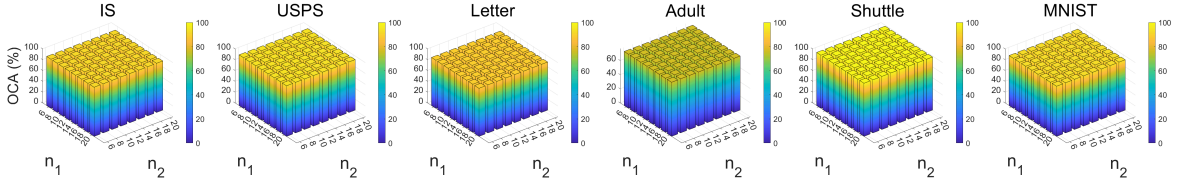

Figure C5 The mean of OCA under different n_1 and n_2 values.

Table C11 The mean and standard deviation (%) of OCA under different λ values.

λ	$1e-3$	$1e-4$	$1e-5$	$1e-6$	$1e-7$	$1e-8$	$1e-9$	$1e-10$
IS	86.5±0.36	87.6±0.29	88.9±0.28	90.0±0.31	90.7±0.18	**90.8±0.23**	90.6±0.43	90.1±0.53
USPS	**94.6±0.20**	94.1±0.29	93.9±0.37	93.7±0.38	93.7±0.38	93.6±0.39	93.6±0.39	93.6±0.38
Letter	84.6±0.16	87.3±0.09	88.2±0.08	88.7±0.10	**89.0±0.11**	88.9±0.14	88.6±0.11	88.2±0.12
Adult	75.5±0.03	75.6±0.03	75.7±0.03	75.9±0.04	76.1±0.06	76.4±0.05	76.8±0.07	**77.2±0.06**
Shuttle	95.2±0.03	95.8±0.02	96.3±0.03	97.3±0.05	97.9±0.04	98.2±0.03	98.4±0.04	**98.5±0.03**
MNIST	**92.6±0.10**	92.5±0.10	92.5±0.10	92.5±0.10	92.5±0.10	92.5±0.11	92.5±0.11	92.5±0.11

Table C12 The mean and standard deviation (%) of OCA under different μ values.

μ	0.80	0.85	0.90	0.95	0.99
Hyperplane	66.8±0.07	69.2±0.09	74.5±0.16	83.5±0.16	**92.6±0.06**
Sea	71.8±0.29	73.2±0.32	75.4±0.33	79.0±0.29	**85.2±0.16**
Electricity	88.0±0.11	88.6±0.09	89.3±0.06	**90.0±0.04**	86.8±0.04
CoverType	91.6±0.02	91.7±0.03	91.7±0.03	91.6±0.03	**94.3±0.01**

20: 1–41
10 Wan Y, Zhang L. Efficient adaptive online learning via frequent directions. IEEE Trans. Pattern Anal. Mach. Intell., 2022, 44: 6910–6923
11 Feinberg V, Chen X, Sun Y J, et al. Sketchy: Memory-efficient adaptive regularization with frequent directions. In: Proceedings of Advances in Neural Information Processing Systems 36, New Orleans, 2023. 75911–75924
12 Yang S, Wan Y, Li P, et al. Dimension-free adaptive subgradient methods with frequent directions. In: Proceedings of the 42th International Conference on Machine Learning, Vancouver, 2025. 1–11
13 Larochelle H, Bengio Y, Louradour J, et al. Exploring strategies for training deep neural networks. J. Mach. Learn. Res., 2009, 10: 1–40
14 Zhang T, Lei C Y, Zhang Z Y, et al. AS-NAS: Adaptive scalable neural architecture search with reinforced evolutionary algorithm for deep learning. IEEE Trans. Evol. Comput., 2021, 25: 830–841
15 Zinkevich M. Online convex programming and generalized infinitesimal gradient ascent. In: Proceedings of the 20th international conference on machine learning, Washington, 2003. 928–936
16 Sahoo D, Pham Q, Lu J, et al. Online deep learning: Learning deep neural networks on the fly. In: Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, Stockholm, 2018. 2660–2666
17 Ashfahani A, Pratama M. Autonomous deep learning: Continual learning approach for dynamic environments. In: Proceedings of the 2019 SIAM international conference on data mining, Calgary, 2019. 666–674
18 Gunasekara N, Gomes H M, Pfahringer B, et al. Online hyperparameter optimization for streaming neural networks. In: Proceedings of the 2022 international joint conference on neural networks, Padua, 2022. 1–9
19 Wen Y M, Liu X, Yu H. Adaptive tree-like neural network: Overcoming catastrophic forgetting to classify streaming data with concept drifts. Knowledge-Based Syst., 2024, 293: 111636
20 Su R, Guo H S, Wang W J. Elastic online deep learning for dynamic streaming data. Inf. Sci., 2024, 676: 120799
21 Lei C Y, Guo J F, Chen C L P. Convbls: An effective and efficient incremental convolutional broad learning system combining deep and broad representations. IEEE Trans. Artif. Intell., 2024, 5: 5075–5089
22 Gong X R, Chen C L P, Hu B, et al. Ciabl: Completeness-induced adaptative broad learning for cross-subject emotion recognition with eeg and eye movement signals. IEEE Trans. Affect. Comput., 2024, 15:1970–1984
23 Lei C Y, Chen C L P, Zhang T. Am-convbls: Adaptive manifold convolutional broad learning system for cross-session and cross-subject emotion recognition. IEEE Trans. Affect. Comput., 2025, 1-15
24 Guo J F, Liu Z L, Chen C L P. An incremental-self-training-guided semi-supervised broad learning system. IEEE Trans. Neural Netw. Learn. Syst., 2025, 36: 7196–7210
25 Yun F, Yu Z W, Yang K X, et al. Adaboost-stacking based on incremental broad learning system. IEEE Trans. Knowl. Data Eng., 2024, 36: 7585–7599
26 Zhang Z Y, Weng H H, Zhang T, et al. A broad generative network for two-stage image outpainting. IEEE Trans. Neural Netw. Learn. Syst., 2024, 35: 12731–12745
27 Greville T N E. Some applications of the pseudoinverse of a matrix. SIAM Rev., 1960, 2: 15–22
28 Chen C L P, Liu Z L. Broad learning system: an effective and efficient incremental learning system without the need for deep architecture. IEEE Trans. Neural Netw. Learn. Syst., 2017, 29: 10–24
29 Fu Y, Cao H R, Chen X F, et al. Task-incremental broad learning system for multi-component intelligent fault diagnosis of machinery. Knowledge-Based Syst., 2022, 246: 108730
30 Du J, Liu P, Vong C M, et al. Class-incremental learning method with fast update and high retainability based on broad learning system. IEEE Trans. Neural Netw. Learn. Syst., 2023, 35: 11332–11345
31 Zhong L J, Chen C L P, Guo J F, et al. Robust incremental broad learning system for data streams of uncertain scale. IEEE Trans. Neural Netw. Learn. Syst., 2025, 36: 7580–7593
32 Seeger M. Low rank updates for the Cholesky decomposition. Technical Report, 2007
33 Higham N J. Accuracy and stability of numerical algorithms. SIAM, 2002
34 Igelnik B, Pao Y H. Stochastic choice of basis functions in adaptive function approximation and the functional-link net. IEEE Trans. Neural Netw., 1995, 6: 1320–1329
35 Zhang L, Suganthan P N. Visual tracking with convolutional random vector functional link network. IEEE T. Cybern., 2016, 47: 3243–3253
36 Huang G B, Zhou H, Ding X, et al. Extreme learning machine for regression and multiclass classification. IEEE Trans. Syst. Man Cybern. B, Cybern., 2011, 42: 513–529
37 Shao Z, Er M J. An online sequential learning algorithm for regularized extreme learning machine. Neurocomputing, 2016, 173: 778–788