Fast Sensitivity Analysis Based Online Self-Organizing Broad Learning System

***Abstract*— This paper proposes a fast sensitivity analysis based online self-Organizing broad learning system (SASO-BLS) to enhance the accuracy and simplicity of BLS during incremental learning. Specifically, the SASO-BLS inherits the incremental learning capability of the original BLS and incorporates a novel fast partial differential-based sensitivity analysis (FPD-SA) approach to make the model more precise and concise. Unlike conventional SA methods that require iteratively evaluate SA indices for all samples, the FPD-SA employs the deduced chain rule across categories, thus mitigating the effect of large amounts of data and significantly reducing computational time. Furthermore, FPD-SA is a general method that provides high-performance and miniature choices for any differentiable model. By combining FPD-SA with BLS, the offline SASO-BLS algorithm for discrete data is derived and can be extended to an online mode for processing streaming data. Unlike existing model self-organizing methods permanently add or delete neurons, SASO-BLS retains sensitivity information for all nodes and adjusts the global structure of the model during incremental learning, resulting in a proper balance between model size and accuracy that is optimal for the current data and requirements. Finally, experimental results on a benchmark dataset and a real-world one demonstrate the effectiveness of the proposed approach.**

***Index Terms*— Broad learning system, incremental learning, sensitivity analysis, self-organizing.**

I. Introduction

In modern industrial processes, the increasing complexity has highlighted the significance of data-driven models, which can deliver exceptional performance without prior knowledge [1]. Among the various data-driven models, convolutional neural networks (CNNs) [2] and long short-term memory (LSTM) networks [3] are widely used for processing discrete and streaming data, respectively. However, the high hardware cost and lack of theoretical foundations pose challenges for enterprises to fully realize their benefits [4]. As an alternative, single hidden layer feedforward neural networks (SLFNs) [5] have gained increasing attention in real-world applications. The first known SLFN, the random vector functional link (RVFL) network [6], is a lightweight and theoretically rigorous model. However, it requires manual specification of the model structure and the range of random input weights is not determined [7]. To address these limitations, Wang *et al.* propose the stochastic configuration network (SCN) with a novel supervisory mechanism to automatically generate meaningful nodes and determine the theoretical ranges of input weights [8]. The SCN has been applied to a number of real-world applications, but it is commonly considered as a static model, lacking the ability to quickly update over time.

Recently, a novel SLFN named broad learning system (BLS) demonstrates robust generalization capabilities in the context of big data analysis [9]. Furthermore, the BLS's ability to be incrementally updated without the need for retraining from scratch has garnered widespread interest in data-driven modeling [10], [11]. While numerous studies have leveraged the broad expansion capabilities of BLS, the established universal approximation theorem does not provide a criterion for constructing an appropriate network structure given a dataset, particularly for streaming datasets with dynamic properties [12]. This can result in an excessive storage burden and elevated risk of overfitting [13]. To mitigate these challenges, Chen *et al.* proposed the use of singular value decomposition to eliminate redundant structures, however, this approach suffers from high computational costs [9]. To address this issue, Zhang *et al.* utilized the dropout mechanism [13], but this requires tuning the dropout rate for different layers, as a constant dropout rate may result in either excessive perturbation or insufficient regularization [14]. Currently, there is a lack of theoretical guidance for determining the appropriate architecture for BLS.

Over the past few decades, model self-organizing algorithms have garnered significant attention in the academic community, with studies exploring their implementation based on both biological and mathematical principles [15], [16]. Among them, sensitivity analysis (SA) based methods have proven to be a highly effective and widely used approach. This is because SA-based methods are able to theoretically assess the impact of individual neurons on the overall model output [17]. For instance, Han *et al.* develop a self-organizing fuzzy neural network (FNN), with the structure learning approach being based on the variance-based SA method [18]. However, it should be noted that variance-based SA methods are only applicable for prediction tasks [19]. In contrast, Kowalski *et al.* employ the differential-based SA method to streamline the structure of probabilistic neural networks in classifier models [20]. Nevertheless, the differential-based SA method adopted in [20] is burdened by computational expenses when processing large data sets. As a result, SA-based self-organizing methods are mainly utilized in offline learning.

In the field of online self-organizing technologies, there is a significant emphasis on the utilization of FNN due to their ability to dynamically construct and eliminate fuzzy rules through a cost-efficient approach [21]. For instance, Wang *et al.* propose a fast and precise online self-organizing FNN that accelerates the learning process through the integration of pruning techniques into the growth criteria [22]. On the other hand, Han *et al.* develop a second-order algorithm that can simultaneously estimate both the network size and parameters [23]. Nevertheless, the computation of the fuzzy parameters is a computationally intensive task that remains challenging with current technological architectures [24]. Additionally, the conventional online self-organizing methods, which periodically add or remove neurons, may increase the risk of performance degradation over time. In view of these limitations, there is a pressing need for the development of a more general and robust online self-organizing model that can better serve various applications.

In order to address the aforementioned challenges, this paper proposes a fast SA-based self-organizing BLS (SASO-BLS) method. This approach enhances the generalization capacity of BLS by producing a more compact structure, making it suitable for both offline and online learning scenarios. The main contributions can be summarized as follows.

1) We propose a novel fast partial derivative-based SA (FPD-SA) method as a novel solution to the computational inefficiencies associated with traditional methods. Unlike conventional approaches that calculate the SA indices for all samples, the FPD-SA method employs a matrix multiplication form to determine these indices across categories, leading to a more efficient and rational approach. Furthermore, FPD-SA is a differentiable model-orientated general method that eliminates the need for parameter tuning, yet possesses robust performance capabilities and offers scalable options to address diverse requirements.
2) By embedding FPD-SA into BLS, we develop SASO-BLS and derive its rapid update method to enable offline and online learning without the need for retraining from scratch. Unlike existing model self-organizing methods that irreversibly add or delete neurons, SASO-BLS maintains the information of all nodes and adjusts the model structure from a holistic perspective, thereby minimizing the impact on task accuracy and making it more suited for processing streaming data.
3) Comprehensive experimental evaluations were conducted on both a benchmark dataset and a real-world one to demonstrate the superiority of the proposed FPD-SA and SASO-BLS technologies[1].

The layout of this paper is given as follows. Section II provides a brief overview of the BLS and partial differential-based SA. Section III details the proposed FPD-SA method, offline SASO-BLS, and online SASO-BLS. Section IV presents experimental results for a benchmark dataset and a real-world heavy plate one. Finally, Section V concludes this paper.

II. Preliminaries

This section provides a brief introduction of the BLS and partial differential-based SA.

[1]The source code will be made available upon acceptance of the article.

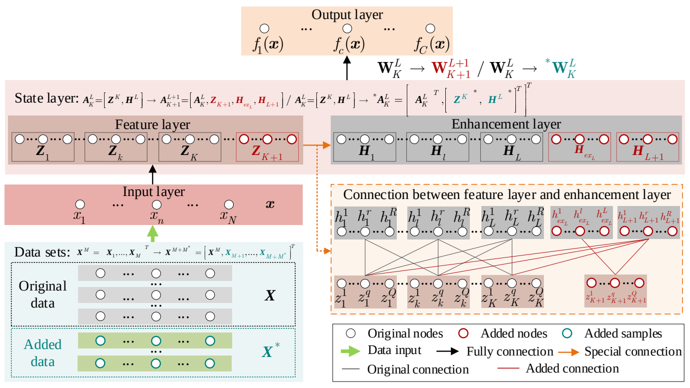

Fig. 1. Framework of the basic and incremental BLS.

A. Broad Learning System

The framework of BLS is shown in Fig. 1. Given a dataset $\{\boldsymbol{X},\boldsymbol{Y}\} = \{(\boldsymbol{x},\boldsymbol{y})_m \in \mathbb{R}^N \times \mathbb{R}^C\}_{m=1}^M$, where M is the sample size, N and C are the input and output dimensions. As shown in Fig. 1, the basic BLS model is formulated as

$$F(\boldsymbol{X}) = [f_1(\boldsymbol{X}), ..., f_c(\boldsymbol{X}), ..., f_C(\boldsymbol{X})] = \boldsymbol{A}_K^L \mathbf{W}_K^L \quad (1)$$

where $\boldsymbol{A}_K^L = [\boldsymbol{Z}^K, \boldsymbol{H}^L]$ is the state layer matrix and

$$\boldsymbol{Z}^K = [\boldsymbol{Z}_1, ..., \boldsymbol{Z}_k, ..., \boldsymbol{Z}_K], \boldsymbol{H}^L = [\boldsymbol{H}_1, ..., \boldsymbol{H}_l, ..., \boldsymbol{H}_L] \quad (2)$$

are the feature layer matrix with K sets of feature nodes and the enhancement layer matrix with L sets of enhancement nodes, respectively. Suppose each set of feature layer and enhancement layer have Q and R nodes, we have

$$\boldsymbol{Z}_k = \left[z_k^1, ..., z_k^q, ..., z_k^Q\right], \boldsymbol{H}_l = \left[h_l^1, ..., h_l^r, ..., h_l^R\right] \quad (3)$$

where z_k^q and h_l^r are obtained by

$$z_k^q = \phi\left(\sum_N w_{x_n}^{z_k^q} x_n + \beta_{\boldsymbol{Z}_k}\right), h_l^r = \phi\left(\sum_Q w_{z_k^q}^{h_l^r} z_k^q + \beta_{\boldsymbol{H}_l}\right), \quad (4)$$

respectively. $w_{x_n}^{z_k^q}$ and $w_{z_k^q}^{h_l^r}$ are the weight between the nodes represented by the upper and lower corners. $\beta_{\boldsymbol{Z}_k}$ and $\beta_{\boldsymbol{H}_l}$ are the corresponding bias. $\phi(.)$ is the activation function. All the weights are randomly generated except for the output weights $\mathbf{W}_K^L$, which can be obtained by

$$\mathbf{W}_K^L = \left(\boldsymbol{A}_K^L\right)^\dagger \boldsymbol{Y} \quad (5)$$

where $\left(\boldsymbol{A}_K^L\right)^\dagger$ is the inverse of the $\boldsymbol{A}_K^L$.

The model generalization ability can be improved by adding samples. Consider a new dataset $\{\boldsymbol{X}^*, \boldsymbol{Y}^*\} = \{(\boldsymbol{x}^*, \boldsymbol{y}^*)_{m^*} \in \mathbb{R}^N \times \mathbb{R}^C\}_{m^*=1}^{M^*}$ as shown in Fig. 1, where M^* is added sample size. The updated state layer matrix can be written as

$$^*\boldsymbol{A}_K^L = \left[\left(\boldsymbol{A}_K^L\right)^T, \left(\left(\boldsymbol{A}_K^L\right)^*\right)^T\right]^T \quad (6)$$

where $\left(\boldsymbol{A}_K^L\right)^* = \left[\left(\boldsymbol{Z}^K\right)^*, \left(\boldsymbol{H}^L\right)^*\right]$ is the state layer matrix obtained by the new samples and the inverse of $^*\boldsymbol{A}_K^L$ is

$$\left(^*\boldsymbol{A}_K^L\right)^\dagger = \left[\left(\boldsymbol{A}_K^L\right)^\dagger - \boldsymbol{b}\boldsymbol{d}^T, \boldsymbol{b}\right] \quad (7)$$

where

$$\begin{aligned} \boldsymbol{d}^T &= \left(\boldsymbol{A}_K^L\right)^* \left(\boldsymbol{A}_K^L\right)^\dagger \\ \boldsymbol{b} &= \begin{cases} (\boldsymbol{c})^\dagger, & \text{if} \boldsymbol{c} \neq 0 \\ \left(\boldsymbol{A}_K^L\right)^\dagger \boldsymbol{d}\left(1+\boldsymbol{d}^T\boldsymbol{d}\right)^{-1}, & \text{if} \boldsymbol{c} = 0 \end{cases} \\ \boldsymbol{c} &= \left(\boldsymbol{A}_K^L\right)^* - \boldsymbol{d}^T\boldsymbol{A}_K^L \end{aligned} \tag{8}$$

Therefore, the output weights can be updated by

$$ {}^*\mathbf{W}_K^L = \mathbf{W}_K^L + \boldsymbol{b}\left(\boldsymbol{Y}^* - \left(\boldsymbol{A}_K^L\right)^*\mathbf{W}_K^L\right) \tag{9}$$

Adding nodes can also effectively improve the model generalization ability. Consider adding red nodes as shown in Fig. 1. Then, the updated state layer matrix can be expressed as

$$\boldsymbol{A}_{K+1}^{L+1} = \left[\boldsymbol{A}_K^L, \boldsymbol{Z}_{K+1}, \boldsymbol{H}_{ex_L}, \boldsymbol{H}_{L+1}\right] \tag{10}$$

where $\boldsymbol{Z}_{K+1}$, $\boldsymbol{H}_{L+1}$ and $\boldsymbol{H}_{ex_L}$ are added feature, enhancement and corresponding enhancement matrix, respectively. The inverse of (10) is formulated as

$$\left(\boldsymbol{A}_{K+1}^{L+1}\right)^\dagger = \begin{bmatrix} \left(\boldsymbol{A}_K^L\right)^\dagger - \mathbf{d}\mathbf{b}^T \\ \mathbf{b}^T \end{bmatrix} \tag{11}$$

where

$$\begin{aligned} \mathbf{d} &= \left(\boldsymbol{A}_K^L\right)^\dagger \left[\boldsymbol{Z}_{K+1}, \boldsymbol{H}_{ex_L}, \boldsymbol{H}_{L+1}\right] \\ \mathbf{b}^T &= \begin{cases} (\mathbf{c})^\dagger, & \text{if} \mathbf{c} \neq 0 \\ \left(1+\mathbf{d}^T\mathbf{d}\right)^{-1}\mathbf{d}^T\left(\boldsymbol{A}_K^L\right)^\dagger, & \text{if} \mathbf{c} = 0 \end{cases} \\ \mathbf{c} &= \left[\boldsymbol{Z}_{K+1}, \boldsymbol{H}_{ex_L}, \boldsymbol{H}_{L+1}\right] - \boldsymbol{A}_K^L\mathbf{d} \end{aligned} \tag{12}$$

and the output weight can be updated by

$$\mathbf{W}_{K+1}^{L+1} = \begin{bmatrix} \mathbf{W}_K^L - \mathbf{d}\mathbf{b}^T\boldsymbol{Y} \\ \mathbf{b}^T\boldsymbol{Y} \end{bmatrix} \tag{13}$$

B. Partial Differential Based SA

This subsection briefly describes the partial differential-based SA for network structure reduction [20]. Given a model $\boldsymbol{Y} = f(\boldsymbol{X}; \mathbf{W})$, where $\boldsymbol{Y} \in \mathbb{R}^C$. Suppose $\mathbf{L} \in \mathbb{R}^N$ is a layer of the model to be reduced, and N is the nodes number of $\mathbf{L}$. Let the sensitivity matrix for a single sample be $\boldsymbol{S} = (S_{n,c}) \in \mathbb{R}^{N\times C}$, the sensitivity indices of the nth neuron l_n of $\mathbf{L}$ to the cth output Y_c can be obtained by

$$is \tag{14}$$

One can obtain the sensitivity matrix for the entire dataset by taking the maximum value (or mean-square average, absolute value average) along the sample dimension. Then, we have

$$\mathbf{S} = \max_M \{\boldsymbol{S}^m\} \in \mathbb{R}^{N\times C} \tag{15}$$

where M is the sample size. Sorting each row of $\mathbf{S}$ in descending order as

$$\vec{\mathbf{S}}_c = \left\{S_{(1),c}, ..., S_{(n),c}, ..., S_{(N),c}\right\}, c = 1, 2, ...C \tag{16}$$

and the difference between the adjacent elements of $\vec{\mathbf{S}}_c$ is

$$\begin{aligned} \Delta\vec{\mathbf{S}}_c &= \{S_{(n+1),c} - S_{(n),c}\}_{n=1}^{N-1} \\ &= \{\Delta S_{(n),c}\}_{n=1}^{N-1} \end{aligned} \tag{17}$$

The neuron selection for the cth class is defined as follow.

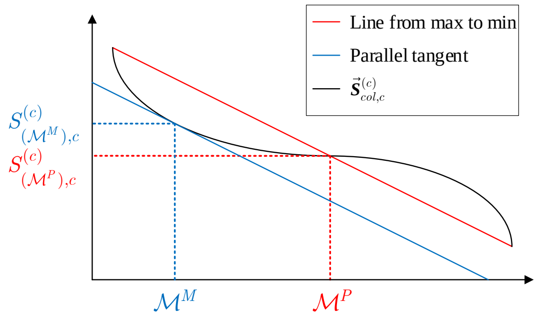
Line from max to min
Parallel tangent
$\vec{\mathbf{S}}_{col,c}^{(c)}$
$S_{(\mathcal{M}^M),c}^{(c)}$
$S_{(\mathcal{M}^P),c}^{(c)}$
$\mathcal{M}^M$
$\mathcal{M}^P$

Fig. 2. The method of choosing cut point

Definition 1: Let $\boldsymbol{R}_c$ be the subset of $\{1, ..., N\}$. For each $r \in \boldsymbol{R}_c$, $\mathbf{L}_c^{\text{de}} = \mathbf{L}_c\{\boldsymbol{R}_c\}$ is the deactivated set of the neurons if the following holds

$$\begin{cases} \frac{\Delta S_{(r),c}}{\vec{S}_{(r),c}} > \eta \\ \vec{S}_{(r),c} > \tau \cdot \overline{\Delta\vec{\mathbf{S}}_c} \end{cases} \tag{18}$$

where $\overline{\Delta\vec{\mathbf{S}}_c}$ is the mean value of elements in $\Delta\vec{\mathbf{S}}_c$, η and τ are assumed parameters. The set $\mathbf{L}_c^{\text{de}}$ exist if $\mathbf{L}_c\{\boldsymbol{R}\} \neq \emptyset$ and $|\mathbf{L}_c\{\boldsymbol{R}_c\}| < N$. The entire set of reduced nodes can be obtained by taking the union of all the sets $(\mathbf{L}_1\{\boldsymbol{R}_1\}, ..., \mathbf{L}_c\{\boldsymbol{R}_c\}, ..., \mathbf{L}_C\{\boldsymbol{R}_C\})$

III. Methodology

This section elaborates on the proposed FPD-SA approach, offline SASO-BLS, and online SASO-BLS.

A. Fast Partial Differential Based Sensitivity Analysis

From Section II-B, the current partial differential based SA methods are limited by the following drawbacks: 1) a high computational cost particularly when dealing with large data sets; 2) an unreasonable uniform treatment of all samples despite differences in features between samples of different classes; 3) the requirement for extensive comparison experiments to determine parameters such as η and τ. In response to these limitations, this research proposes a highly efficient alternative approach.

Given a model as section II-B, suppose there are V hidden layers after $\mathbf{L}$ and activation function $\phi(.)$ is deployed in each layer. To address the first two drawbacks mentioned above, We process samples of different categories unified. Let $\boldsymbol{X}^{(c)}$ be the samples classified into the cth class, whose sensitivity matrix is

$$\mathbf{S}^{(c)} = \max_{M^{(c)}} \{\boldsymbol{S}\} \in \mathbb{R}^{N\times C} \tag{19}$$

where $M^{(c)}$ is the number of samples of the cth class. Then, the $\mathbf{S}^{(c)}$ can be calculated by

$$\mathbf{S}^{(c)} = \mathbf{W}_{\mathbf{L}}^{\mathbf{L+1}}\mathbf{D}_{\mathbf{L}}^{(c)}\mathbf{W}_{\mathbf{L+1}}^{\mathbf{L+2}}\mathbf{D}_{\mathbf{L+1}}^{(c)}...\mathbf{W}_{\mathbf{L}}^{\mathbf{L}+V} \tag{20}$$

$$\mathbf{D}_{(\cdot)}^{(c)} = \begin{cases} E - D_{(\cdot)}^{(c)} \otimes D_{(\cdot)}^{(c)} & , \phi(.) = \tanh(.) \\ D_{(\cdot)}^{(c)} - D_{(\cdot)}^{(c)} \otimes D_{(\cdot)}^{(c)} & , \phi(.) = \text{sigmoid}(.) \end{cases} \tag{21}$$

where $\mathbf{W}_{\mathbf{L}}^{\mathbf{L+1}}$ is the weight matrix from layer $\mathbf{L}$ to $\mathbf{L+1}$ and so on. $\otimes$ represents dot product, $D_{(\cdot)}^{(c)}$ is the matrix of layer

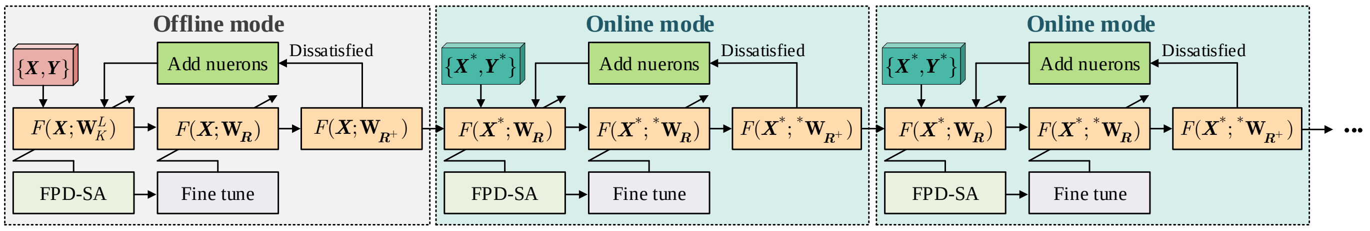

Fig. 3. Offline and online mode of the SASO-BLS.

$(\cdot)$ obtained by $\boldsymbol{X}^{(c)}$ during the forward propagation of the network. Sorting each column of $\mathbf{S}^{(c)}$ in descending order, we have

$$\vec{\mathbf{S}}^{(c)}_{col} = \left\{ \vec{\boldsymbol{S}}^{(c)}_{col,1}, ..., \vec{\boldsymbol{S}}^{(c)}_{col,c}, ..., \vec{\boldsymbol{S}}^{(c)}_{col,C} \right\} \tag{22}$$

$$\vec{\boldsymbol{S}}^{(c)}_{col,c} = \left\{ S^{(c)}_{(1),c}, ..., \ S^{(c)}_{(n),c}, ..., S^{(c)}_{(N),c} \right\} \tag{23}$$

Let the nth row vector of $\mathbf{S}^{(c)}$ be $\boldsymbol{S}^{(c)}_{n,row}$, we can develop the following definition to select important nodes for cth output.

Definition 2: Let $\boldsymbol{R}_c$ be the subset of $\{1, ..., N\}$. For each $r \in \boldsymbol{R}_c$, $\mathbf{L}^{\mathrm{se}} = \mathbf{L}\{\boldsymbol{R}_c\}$ is the selected set of the neurons if the following holds

$$\begin{cases} S^{(c)}_{r,c} & = & \max\left\{ \boldsymbol{S}^{(c)}_{n,row} \right\} \\ S^{(c)}_{r,c} & \geq & S^{(c)}_{(\mathcal{M}),c}, \mathcal{M} \in \{\mathcal{M}^M, \mathcal{M}^P\} \end{cases} \tag{24}$$

where $\mathcal{M}^M$ and $\mathcal{M}^P$ are two optional cut points shown in Fig. 2 for lightweight and high-performance, respectively. The set $\mathbf{L}^{\mathrm{se}}$ exist if $\boldsymbol{R}_c \neq \emptyset$ and $|\boldsymbol{R}_c| < N$.

Thereafter, by repeating above process over all the classes, the selected neurons can be obtained as

$$\boldsymbol{R} = \bigcup_{c=1}^{C} \boldsymbol{R}_c \tag{25}$$

B. SASO-BLS under Offline Mode

As shown in Fig. 3, the proposed SASO-BLS of offline mode is first elaborated in this subsection. Given a BLS $F\left(\boldsymbol{X}; \mathbf{W}^L_K\right)$ as (1), the sensitivity matrix of the state layer obtained by $\boldsymbol{X}^{(c)}$ can be denoted as

$$\mathbf{S}^{(c)}_{\boldsymbol{A}^L_K} = \left[\mathbf{S}^{(c)}_{\boldsymbol{Z}^K}, \mathbf{S}^{(c)}_{\boldsymbol{H}^L} \right] = \left[\frac{\partial F\left(\boldsymbol{X}^{(c)}\right)}{\partial \boldsymbol{Z}^K}, \frac{\partial F\left(\boldsymbol{X}^{(c)}\right)}{\partial \boldsymbol{H}^L} \right] \tag{26}$$

where

$$\begin{aligned} \mathbf{S}^{(c)}_{\boldsymbol{Z}^K} &= \mathbf{W}^{\boldsymbol{H}^L}_{\boldsymbol{Z}^K} \mathbf{D}^{(c)}_{\boldsymbol{H}^L} \mathbf{W}_{\boldsymbol{H}^L} + \mathbf{W}_{\boldsymbol{Z}^K} \\ &= \left(S^{(c)}_{\boldsymbol{Z}_k} \right) \in \mathbb{R}^{N_{\boldsymbol{Z}} \times C} \end{aligned} \tag{27}$$

$$\mathbf{D}^{(c)}_{\boldsymbol{H}^L} = \begin{cases} \boldsymbol{E} - \boldsymbol{H}^L_{(c)} \otimes \boldsymbol{H}^L_{(c)} & , \phi(.) = \tanh(.) \\ \boldsymbol{H}^L_u - \boldsymbol{H}^L_{(c)} \otimes \boldsymbol{H}^L_{(c)} & , \phi(.) = \mathrm{sigmoid}(.) \end{cases} \tag{28}$$

$$\mathbf{S}^{(c)}_{\boldsymbol{H}^L} = \mathbf{W}_{\boldsymbol{H}^L} = \left(S^{(c)}_{\boldsymbol{H}_l} \right) \in \mathbb{R}^{N_{\boldsymbol{H}} \times C}. \tag{29}$$

$\mathbf{W}_{(\cdot)}$ is output weight matrix of $(\cdot)$, $\mathbf{W}^{\boldsymbol{H}^L}_{\boldsymbol{Z}^K}$ is weight matrix between the feature layer and enhancement layer, $N_{\boldsymbol{Z}}$ and $N_{\boldsymbol{H}}$ denote the nodes number of corresponding layer. $\boldsymbol{E}$ is unit matrix, $\boldsymbol{H}^L_{(c)}$ is the enhancement matrix obtained by $\boldsymbol{X}^{(c)}$. $\phi(.)$ stands different activation functions. Note that $\mathbf{S}^{(c)}_{\boldsymbol{H}^L}$ is constant over all classes. Sorting each column of $\mathbf{S}^{(c)}_{\boldsymbol{A}^L_K}$ in descending order, we have

$$\vec{\mathbf{S}}^{(c)}_{col,\boldsymbol{A}^L_K} = \left\{ \vec{\boldsymbol{S}}^{(c)}_{col,1}, ..., \vec{\boldsymbol{S}}^{(c)}_{col,c}, ..., \vec{\boldsymbol{S}}^{(c)}_{col,C} \right\} \tag{30}$$

$$\vec{\boldsymbol{S}}^{(c)}_{col,c} = \left\{ S^{(c)}_{(1),c}, ..., \ S^{(c)}_{(n),c}, ..., S^{(c)}_{(N_{\boldsymbol{A}}),c} \right\} \tag{31}$$

where $N_{\boldsymbol{A}}$ is nodes number of the state layer.

Let the nth row vector of $\mathbf{S}^{(c)}_{\boldsymbol{A}^L_K}$ be $\boldsymbol{S}^{(c)}_{n,row}$, we can obtain the selected neurons of the state layer $\boldsymbol{A}\{\boldsymbol{R}_c\}$ by Definition 2. Therefore, the selected neuron set is

$$\boldsymbol{A}\{\boldsymbol{R}\} = \bigcup_{c=1}^{C} \boldsymbol{A}\{\boldsymbol{R}_c\} \tag{32}$$

Let $\boldsymbol{A}_{\boldsymbol{R}}$ denote the updated state layer matrix obtained by $\boldsymbol{X}$. From (11), we have

$$\left(\boldsymbol{A}^L_K\right)^{\dagger} = \begin{bmatrix} \boldsymbol{A}^{\dagger}_{\boldsymbol{R}} - \mathbf{d}\mathbf{b}^T \\ \mathbf{b}^T \end{bmatrix} \tag{33}$$

where

$$\begin{aligned} &\mathbf{d} = \left(\boldsymbol{A}^L_K\right)^{\dagger} \boldsymbol{A}_D \\ &\mathbf{b}^T = \begin{cases} (\mathbf{c})^{\dagger}, & \mathrm{if}\,\mathbf{c} \neq 0 \\ \left(1 + \mathbf{d}^T\mathbf{d}\right)^{-1} \mathbf{d}^T \left(\boldsymbol{A}^L_K\right)^{\dagger}, & \mathrm{if}\,\mathbf{c} = 0 \end{cases} \\ &\mathbf{c} = \boldsymbol{A}_D - \boldsymbol{A}^L_K \mathbf{d} \end{aligned} \tag{34}$$

and $\boldsymbol{A}_D$ is the matrix of inactivated state layer nodes. $\boldsymbol{A}^{\dagger}_R$ is inverse of $\boldsymbol{A}_R$. Let

$$\boldsymbol{A}^{\dagger}_{\boldsymbol{R}-} = \boldsymbol{A}^{\dagger}_{\boldsymbol{R}} - \mathbf{d}\mathbf{b}^T, \tag{35}$$

and

$$\mathbf{W}_{\boldsymbol{D}} = \boldsymbol{A}^{\dagger}_{R-} \boldsymbol{Y} \tag{36}$$

then we have

$$\boldsymbol{A}^{\dagger}_{\boldsymbol{R}} = \boldsymbol{A}^{\dagger}_{\boldsymbol{R}-} \left(1 - \boldsymbol{A}_{\boldsymbol{D}} \mathbf{b}^T\right)^{-1} \tag{37}$$

$$\mathbf{d} = \boldsymbol{A}^{\dagger}_{R-} \left(\left(1 - \boldsymbol{A}_{\boldsymbol{D}} \mathbf{b}^T\right)^{-1} - 1 \right) \left(\mathbf{b}^T\right)^{-1} \tag{38}$$

and we can multiply both sides of (35) by $\boldsymbol{Y}$ to obtain the output weights as

$$\mathbf{W}_{\boldsymbol{R}} = \mathbf{W}_{\boldsymbol{D}} + \boldsymbol{A}^{\dagger}_{R-} \left(\left(1 - \boldsymbol{A}_{\boldsymbol{D}} \mathbf{b}^T\right)^{-1} - 1 \right) \boldsymbol{Y} \tag{39}$$

Subsequently, we add neurons as (10) to enhance the generalization ability and the state layer matrix is denoted as

$$\boldsymbol{A}_{\boldsymbol{R}^+} = \left[\boldsymbol{A}_{\boldsymbol{R}}, \boldsymbol{Z}_{K+1}, \boldsymbol{H}_{ex_L}, \boldsymbol{H}_{L+1} \right] \tag{40}$$

whose sensitivity matrix obtained by $\boldsymbol{X}^{(c)}$ is similarly expressed as

$$\mathbf{S}^{(c)}_{\boldsymbol{A_{R^+}}} = \left[\mathbf{S}^{(c)}_{\boldsymbol{A_R}}, \mathbf{S}^{(c)}_{\boldsymbol{Z}_{K+1}}, \mathbf{S}^{(c)}_{\boldsymbol{H}_{L+1,c}}, \mathbf{S}^{(c)}_{\boldsymbol{H}_{ex_L}}\right] \tag{41}$$

where the $\mathbf{S}^{(c)}_{\boldsymbol{H}_{L+1}}$ and $\mathbf{S}^{(c)}_{\boldsymbol{H}_{ex_L}}$ are equal to the corresponding output weights $\mathbf{W}_{\boldsymbol{H}_{L+1}}$ and $\mathbf{W}_{\boldsymbol{H}_{ex_L}}$, respectively. The sensitivity matrix of the added feature layer is

$$\begin{aligned} \boldsymbol{S}^{(c)}_{\boldsymbol{Z}_{K+1}} =& \mathbf{W}^{\boldsymbol{H}_{L+1}}_{\boldsymbol{Z}_{K+1}} \mathbf{D}^{(c)}_{\boldsymbol{H}_{L+1}} \mathbf{W}_{\boldsymbol{H}_{L+1}} + ... \\ & ...\mathbf{W}^{\boldsymbol{H}_{ex_L}}_{\boldsymbol{Z}_{K+1}} \mathbf{D}^{(c)}_{\boldsymbol{H}_{ex_L}} \mathbf{W}_{\boldsymbol{H}_{ex_L}} + \mathbf{W}_{\boldsymbol{Z}_{K+1}} \end{aligned} \tag{42}$$

where $\mathbf{D}^{(c)}_{\boldsymbol{H}_{ex_L}}$ and $\mathbf{D}^{(c)}_{\boldsymbol{H}_{L+1}}$ can be calculated in the same way as (28). Note that the added enhancement nodes are also connected to the original feature nodes, causing a change in the sensitivity matrix of the original feature nodes as

$$\hat{\boldsymbol{S}}^{(c)}_{\boldsymbol{Z}^K} = \boldsymbol{S}^{(c)}_{\boldsymbol{Z}^K} + \mathbf{W}^{\boldsymbol{H}_{L+1}}_{\boldsymbol{Z}_K} \mathbf{D}^{(c)}_{\boldsymbol{H}_{L+1}} \mathbf{W}_{\boldsymbol{H}_{L+1}} \tag{43}$$

and the sensitivity matrix of the state layer is finally formulated as

$$\boldsymbol{S}^{(c)}_{\boldsymbol{A_{R^+}}} = \left[\hat{\boldsymbol{S}}^{(c)}_{\boldsymbol{Z}^K}, \boldsymbol{S}^{(c)}_{\boldsymbol{H}^L}, \mathbf{S}^{(c)}_{\boldsymbol{H}_{L+1}}, \mathbf{S}^{(c)}_{\boldsymbol{H}_{ex_L}}\right] \tag{44}$$

Algorithm 1: SASO-BLS under Offline Mode

```
Input: A trained BLS with $\boldsymbol{A}^L_K, \mathbf{W}^L_K, \boldsymbol{X}$
Output: A SASO-BLS with $\boldsymbol{A}\{\boldsymbol{R}^+\}, \mathbf{W}_{\hat{\boldsymbol{R}}^+}$
1  Calculate $\boldsymbol{S}_{\boldsymbol{H}^L}$ by (29);
2  for $c = 1$ to $C$ do
3      Select samples $\boldsymbol{X}^{(c)}$;
4      Calculate $\boldsymbol{S}^{(c)}_{\boldsymbol{Z}^K}$ by (27)-(28);
5      Obtain $\vec{\mathbf{S}}^{(c)}_{col,\boldsymbol{A}^L_K}$ by (30);
6      Obtain $\boldsymbol{A}\{\boldsymbol{R}_c\}$ according to Definition 2;
7  Obtain $\boldsymbol{A}\{\boldsymbol{R}\}$ by (32);
8  $\boldsymbol{A}_R \leftarrow \boldsymbol{A}\{\boldsymbol{R}\}, \boldsymbol{X}$;
9  Calculate $\mathbf{W}_{\mathbf{R}}$ by (39);
10 while one of (24) is not satisfied do
11     Add neurons $[\boldsymbol{Z}_{K+1}, \boldsymbol{H}_{ex_L}, \boldsymbol{H}_{L+1}]$;
12     $\mathbf{S}^{(c)}_{\boldsymbol{H}_{L+1}}, \mathbf{S}^{(c)}_{\boldsymbol{H}_{ex_L}} \leftarrow \mathbf{W}_{\boldsymbol{H}_{L+1}}, \mathbf{W}_{\boldsymbol{H}_{ex_L}}$;
13     Calculate $\mathbf{S}^{(c)}_{\boldsymbol{Z}_{K+1}}$ by (42);
14     $\hat{\boldsymbol{S}}^{(c)}_{\boldsymbol{Z}^K} \leftarrow \boldsymbol{S}^{(c)}_{\boldsymbol{Z}^K}$ by (43);
15     Obtain $\boldsymbol{S}_{\boldsymbol{A}_{R^+}}$ by (44);
16     Obtain $\boldsymbol{A}\{\boldsymbol{R}^+\}$ as (32);
17     $\boldsymbol{A}_{\boldsymbol{R}^+} \leftarrow \boldsymbol{A}\{\boldsymbol{R}^+\}, \boldsymbol{X}$;
18     Update $\mathbf{W}_{\boldsymbol{R}^+}$ as (39);
```

Same as the processes from (30)-(32), we can obtain the selected neurons $\boldsymbol{A}\{\boldsymbol{R}^+\}$. With the inverse of the state layer matrix $\boldsymbol{A}^\dagger_{\boldsymbol{R}^+}$, the output weights $\mathbf{W}_{\boldsymbol{R}^+}$ can be updated by (39).

Subsequently, the incremental learning process is repeated until one of the following conditions are satisfied

$$\left\{ \begin{array}{lcl} \Delta\Phi_t & < & \psi \\ t & = & T_{max} \end{array} \right. \tag{45}$$

where $\Delta\Phi_t$ is training accuracy increment at t incremental step, ψ is a preset threshold, and T_{max} is the maximum incremental step. Finally, the algorithm of SASO-BLS under offline mode is shown as Algorithm 1.

C. SASO-BLS under Online Mode

The process of SASO-BLS under Online Mode is shown in the right part of Fig. 3. Given a trained SASO-BLS with the state layer neutrons $\boldsymbol{A}\{\boldsymbol{R}^+\}$ and output weights $\mathbf{W}_{\boldsymbol{R}^+}$. Considering add new samples $\{\boldsymbol{X}^*, \boldsymbol{Y}^*\}$, the output weights can be updated by (9) as

$${}^*\mathbf{W}_{\boldsymbol{R}^+} = \mathbf{W}_{\boldsymbol{R}^+} + \boldsymbol{b}\left(\boldsymbol{Y}^* - (\boldsymbol{A}_{\boldsymbol{R}^+})^* \mathbf{W}_{\boldsymbol{R}^+}\right) \tag{46}$$

where $(\boldsymbol{A}_{\boldsymbol{R}^+})^*$ denotes the state layer matrix obtained by $\boldsymbol{X}^*$, $\boldsymbol{b}$ is the same as (8). Let $\boldsymbol{X}^{*,(c)}$ be the samples of cth class, we can recalculate the sensitivity matrix of all the state layer as

$$\boldsymbol{S}^{(c)}_{{}^*\boldsymbol{A}^L_K} = \left[\boldsymbol{S}^{(c)}_{{}^*\boldsymbol{Z}^K}, \boldsymbol{S}_{{}^*\boldsymbol{H}^L}\right] \tag{47}$$

where the sensitivity matrix of enhancement layer is

$$\boldsymbol{S}_{{}^*\boldsymbol{H}^L} = \boldsymbol{S}_{\boldsymbol{H}^L} + \mathbf{W}_{(\boldsymbol{H}^L)^*} \tag{48}$$

and $\mathbf{W}_{(\boldsymbol{H}^L)^*}$ is the output weight of enhancement layer obtained by the added samples. From (41), the sensitivity matrix of the feature layer is

Algorithm 2: SASO-BLS under Online Mode

```
Input: A trained SASO-BLS with $\boldsymbol{A}\{\boldsymbol{R}^+\}, \mathbf{W}_{\boldsymbol{R}^+}$,
       new samples $\{\boldsymbol{X}^*, \boldsymbol{Y}^*\}$
Output: A SASO-BLS with $\boldsymbol{A}\{\boldsymbol{R}^+\}, {}^*\mathbf{W}_{\boldsymbol{R}^+}$
1  while new samples $\boldsymbol{X}^*$ is added do
2      Update ${}^*\mathbf{W}_{\boldsymbol{R}^+}$ by (46);
3      Calculate $\boldsymbol{S}_{{}^*\boldsymbol{H}^L}$ by (48);
4      for $c = 1$ to $C$ do
5          Select samples $\boldsymbol{X}^{*,(c)}$ of the $c$th class;
6          Calculate $\boldsymbol{S}^{(c)}_{{}^*\boldsymbol{Z}^K}$ by (49)-(49);
7          Obtain $\vec{\mathbf{S}}^{(c)}_{col,{}^*\boldsymbol{A}^L_K}$ by (29);
8          Obtain $\boldsymbol{A}\{\boldsymbol{R}_c\}$ according to Definition 2;
9      Obtain $\boldsymbol{A}\{\boldsymbol{R}\}$ as (32);
10     $\boldsymbol{A}_R \leftarrow \boldsymbol{A}\{\boldsymbol{R}\}$;
11     Calculate ${}^*\mathbf{W}_{\boldsymbol{R}}$ by (39);
12     Obtain $\boldsymbol{A}\{\boldsymbol{R}^+\}, {}^*\mathbf{W}_{\boldsymbol{R}^+}$ by repeating 10-18 step of
         Algorithm 1;
```

$$\boldsymbol{S}^{(c)}_{{}^*\boldsymbol{Z}^K} = \boldsymbol{S}^{(c)}_{\boldsymbol{Z}^K} + \mathbf{W}^{(\boldsymbol{H}^L)^*}_{(\boldsymbol{Z}^K)^*} \mathbf{D}^* \mathbf{W}_{(\boldsymbol{H}^L)^*} + \mathbf{W}_{(\boldsymbol{Z}^K)^*} \tag{49}$$

$$\mathbf{D}^* = \left\{ \begin{array}{ll} E - \boldsymbol{H}^L_{(c)^*} \otimes \boldsymbol{H}^L_{(c)^*} & ,\tanh \\ \boldsymbol{H}^L_{(c)^*} - \boldsymbol{H}^L_{(c)^*} \otimes \boldsymbol{H}^L_{(c)^*} & ,\text{sigmoid} \end{array} \right. \tag{50}$$

where $\boldsymbol{H}^L_{(c)^*}$ is the enhancement layer calculated by added samples. Thereafter, we can obtain $\vec{\mathbf{S}}^{(c)}_{col,{}^*\boldsymbol{A}^L_K}$ as (30) and reselect neuron set $\boldsymbol{A}\{\boldsymbol{R}\}$ as the offline mode. Finally, the output weight ${}^*\mathbf{W}_{\boldsymbol{R}}$ can be updated by (39).

Note that each online cell evaluates whether equation (45) is satisfied to determine if additional nodes are required. The algorithm for online SASO-BLS is presented as Algorithm 2.

IV. Experiments

In this section, two datasets are utilized to assess the effectiveness of the proposed approach: the class-balanced Tennessee Eastman (TE) process benchmark dataset and a class-imbalanced real-world dataset from a steel plate. The performance of each method for each class is evaluated through the following metrics

$$\begin{cases} Recall & = TP/(TP+FN) \\ Macro-F1 & = 2TP/(2TP+FP+FN) \end{cases} \tag{51}$$

where TP, FP, and FN denote true positive, false positive, and false negative, respectively. The overall performance on both the class-balanced and class-imbalanced datasets is evaluated through the following metrics

$$\begin{cases} Average & = \left(\sum_{C}\mathbf{M}_c\right)/C \\ W-Ave & = \sum_{C}(W_c \cdot \mathbf{M}_c) \end{cases} \tag{52}$$

where $\mathbf{M}_c$ is any metric in (51) for the cth class, W_c stands for the proportion of class c samples to the total samples.

The proposed FPD-SA method is compared to other four SA methods, including elementary effects test-based SA (EET-SA) method [28], variance-based local/global SA method (SV-SA/GV-SA) [29] and partial differential-based SA method (PD-SA) proposed in [20]. Note that EET-SA, SV-SA and GV-SA are implemented by a public toolbox [25]. Note that EET-SA, SV-SA and GV-SA are originally tailored for prediction tasks, we adapt the strategy detailed in [20] to render them appropriate for classification tasks. The proposed SASO-BLS is compared with four random weight feed-forward neural networks (RVFL [6], CRVFL [26], SCN [8], and BLS) and two accepted online learning methods (LSTM [3] and GRU [27]). The MATLAB implementations of RVFL, CRVFL, SCN, BLS, and SASO-BLS were run on a computer with an Intel 3.4GHz CPU, while the PyTorch implementations of LSTMs and GRUs were run on a computer with an NVIDIA GTX 1080 GPU.

A. Case Studies on Benchmark Datasets

In this paper, the normal and first 9 fault conditions of the Tennessee Eastman (TE) process are utilized. The training data consists of 480 samples for each fault and 520 samples for normal conditions, while the test data comprises 800 samples for each fault and 800 samples for normal conditions, yielding a total of 4820 training samples and 8000 testing samples. The data dimension is 52.

1) Parameter Setting: In the comparison experiments of the SA methods, we set BLS with varying structures as the baseline and fix the number of nodes in the feature layer to 10×10, with the number of nodes in the enhancement layer ranging from 100 to 1000. The metric $\mathcal{M}$ in (24) is set to $\mathcal{M}^P$. All the competitors and FPD-SA approach conduct one-shot processing for the various structures of BLS.

In the offline experiments, the number of nodes in the feature layer and enhancement layer were incrementally increased, with the feature layer increasing by 10 nodes and the enhancement layer increasing by 10 nodes per step. The

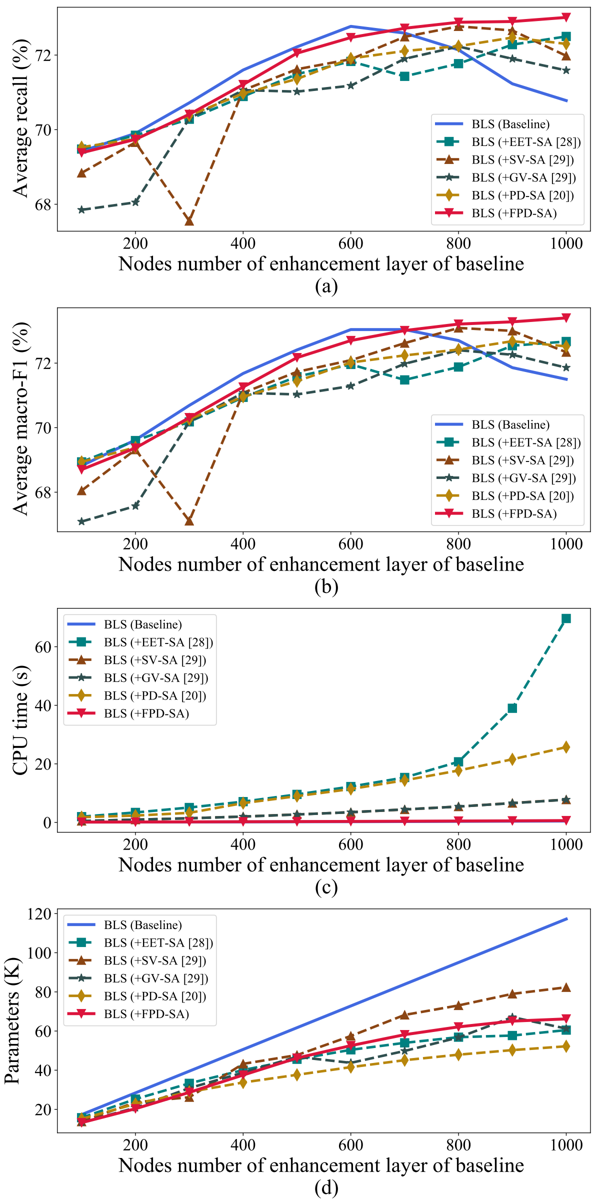

Fig. 4. Comparison with the existing SA methods on the TE process datasets

parameters T_{max} and ψ in equation (45) were set to 20 and 0.005, respectively. The metrics $\mathcal{M}^M$ and $\mathcal{M}^P$ in equation (24) correspond to SASO-BLS-M and SASO-BLS-P, respectively. The RVFL method has 1000 hidden nodes, while the maximum number of hidden nodes and candidate nodes for the SCN method were 500 and 100, respectively. The architecture of CRVFL is Conv$(1\times3)-$Pool$(1\times2)-$Conv$(1\times3)-$Pool$(1\times2)-$FC$(256)-$FC(128). The number of nodes in the feature layer and enhancement layer of the BLS method were 10×10 and 3000, respectively. The number of hidden nodes in both the LSTM and GRU methods is 256, with 1 block in both cases.

In the online experiments, the size of the initial training set is 1/8 of the total and increased incrementally by 1/8 per step. The efficiency of online models is a key consideration, thus SASO-BLS-M is selected for this experiment, with $\mathcal{M}$ in (24)

TABLE I
SENSITIVITY ANALYSIS METHOD PERFORMANCE COMPARISONS

Structure (Feature+Enhance)		100+100	100+200	100+300	100+400	100+500	100+600	100+700	100+800	100+900	100+1000
Method	Metric (Average)										
BLS (Baseline)	Recall	69.42	69.90	70.72	71.60	72.22	72.77	72.59	72.14	71.23	70.78
	Macro-F1	68.82	69.62	70.69	71.68	72.41	73.04	73.04	72.70	71.86	71.50
	CPU Time (s)	0.10	0.12	0.15	0.18	0.22	0.25	0.30	0.35	0.41	0.47
	Parameters	17.31K	28.41K	39.51K	50.61K	61.71K	72.81K	83.91K	95.01K	106.11K	117.21K
BLS (+EET-SA [28])	Recall	69.48	69.85	70.28	70.89	71.49	71.83	71.43	71.77	72.28	72.50
	Macro-F1	68.94	69.60	70.18	70.94	71.58	71.96	71.48	71.88	72.54	72.67
	CPU Time (s)	(+1.86)	(+3.28)	(+4.93)	(+6.91)	(+9.34)	(+11.99)	(+15.04)	(+20.35)	(+38.57)	(+69.18)
	Parameters	15.73K	25.24K	33.28K	40.07K	45.58K	50.41K	53.98K	56.87K	57.70K	60.48K
BLS (+SV-SA [29])	Recall	68.84	69.66	67.55	71.06	71.62	71.89	72.49	72.77	72.66	71.98
	Macro-F1	68.05	69.32	67.11	71.08	71.73	72.09	72.62	73.09	73.00	72.34
	CPU Time (s)	(+0.39)	(+0.82)	(+1.26)	(+1.81)	(+2.50)	(+3.22)	(+4.12)	(+5.06)	(+6.20)	(+7.32)
	Parameters	13.72K	23.63K	26.19K	43.25K	47.71K	57.47K	68.24K	73.08K	78.99K	82.34K
BLS (+GV-SA [29])	Recall	67.85	68.05	70.30	71.06	71.02	71.18	71.90	72.23	71.90	71.59
	Macro-F1	67.09	67.57	70.18	71.08	71.03	71.29	71.98	72.40	72.26	71.86
	CPU Time (s)	(+0.40)	(+0.81)	(+1.26)	(+1.81)	(+2.47)	(+3.23)	(+4.13)	(+5.06)	(+6.16)	(+7.30)
	Parameters	13.43K	20.79K	30.85K	38.75K	46.83K	43.75K	49.83K	56.71K	67.09K	61.25K
BLS (+PD-SA [20])	Recall	69.53	69.75	70.38	70.97	71.36	71.91	72.11	72.24	72.47	72.30
	Macro-F1	68.94	69.39	70.26	70.96	71.44	72.02	72.24	72.42	72.68	72.52
	CPU Time (s)	1.68	2.21	3.11	6.40	8.75	11.16	14.08	17.38	21.14	25.24
	Parameters	15.16K	22.79K	28.91K	33.73K	37.64K	41.57K	45.12K	47.93K	50.27K	52.21K
BLS (+FPD-SA)	Recall	69.38	69.74	70.41	71.21	72.05	72.47	72.72	72.88	72.90	73.01
	Macro-F1	68.70	69.37	70.31	71.26	72.17	72.70	73.01	73.21	73.28	73.40
	CPU Time (s)	(+0.01)	(+0.02)	(+0.03)	(+0.05)	(+0.06)	(+0.08)	(+0.10)	(+0.12)	(+0.13)	(+0.15)
	Parameters	13.21K	20.32K	28.68K	37.53K	46.21K	52.55K	58.15K	62.13K	65.07K	66.19K

^{a} In the CPU time comparison, the values in brackets indicate the time consumed by the corresponding SA methods.

TABLE II
OFFLINE MODEL PERFORMANCE COMPARISONS

Method	Metric	Normal	Fault 1	Fault 2	Fault 3	Fault 4	Fault 5	Fault 6	Fault 7	Fault 8	Fault 9	Average	Para
RVFL [6]	Recall	32.50	98.75	98.25	35.75	98.83	73.50	**100**	**100**	54.00	18.25	70.94	61.05K
	Macro-F1	32.24	94.39	94.30	34.65	98.60	60.29	**100**	99.56	63.95	21.31	70.63	
CRVFL [26]	Recall	36.13	98.00	97.75	38.38	**100**	**99.88**	56.50	99.63	**63.25**	29.63	71.91	30.75K
	Macro-F1	34.86	95.84	**96.55**	**36.57**	97.15	82.24	72.20	99.69	**73.60**	29.79	72.93	
SCN [8]	Recall	36.75	98.13	97.75	35.25	**100**	**99.88**	67.63	98.50	52.13	30.25	71.63	30.27K
	Macro-F1	35.38	97.22	95.48	34.26	84.61	94.73	80.69	98.99	67.15	28.70	72.87	
LSTM [3]	Recall	29.25	**98.50**	98.25	32.38	99.62	99.75	99.50	**100**	54.62	**37.75**	74.96	0.31M
	Macro-F1	31.81	95.75	94.53	31.59	**99.74**	**99.81**	99.75	**100**	69.69	**31.24**	76.09	
GRU [27]	Recall	33.75	**98.50**	98.25	**40.50**	99.38	99.62	99.38	**100**	45.75	24.50	73.96	0.23M
	Macro-F1	34.77	96.69	95.21	33.85	99.62	99.74	99.69	**100**	61.77	23.36	**76.80**	
BLS [9]	Recall	35.63	98.13	98.38	36.88	**100**	**99.88**	92.00	98.50	41.13	29.75	73.13	0.33M
	Macro-F1	32.91	**97.88**	92.32	34.24	98.70	98.16	95.83	99.00	57.98	27.74	74.77	
SASO-BLS-M	Recall	35.00	98.13	98.13	38.38	**100**	**99.88**	91.00	**100**	46.38	30.38	73.73	**28.62K**
	Macro-F1	33.23	96.68	95.85	36.03	98.46	96.68	95.29	**100**	62.73	27.61	75.20	
SASO-BLS-P	Recall	**37.38**	98.38	**98.85**	38.13	**100**	**99.88**	99.88	**100**	46.88	31.88	**75.06**	0.24M
	Macro-F1	**35.98**	97.52	96.14	35.40	98.59	99.07	99.94	99.87	63.56	29.24	76.45	
$+/=/-$ (M)	Recall	3/0/4	1/2/4	2/0/5	5/1/1	3/4/0	3/4/0	2/0/5	3/4/0	2/0/5	5/0/2	4/0/3	7/0/0
	Macro-F1	3/0/4	3/0/4	5/0/2	6/0/1	2/0/5	3/0/4	2/0/5	5/2/0	2/0/5	2/0/5	4/0/3	
$+/=/-$ (P)	Recall	7/0/0	4/0/3	7/0/0	4/0/3	3/4/0	3/4/0	6/0/1	3/4/0	3/0/4	6/0/1	7/0/0	2/0/5
	Macro-F1	7/0/0	6/0/1	6/0/1	5/0/2	3/0/4	5/0/2	6/0/1	4/0/3	3/0/4	5/0/2	6/0/1	

^{a} The last two lines lay out the rank sum test results of SASO-BLS-M and SASO-BLS-P, where each version is better than (+), worse than (-) and equal to (=) others. The best results for 10 classes and their average values across all methods are marked in **boldface**.

set to $\mathcal{M}^M$. The parameters of all methods remain unchanged. As RVFL, CRVFL, and SCN cannot be executed in an online manner, the current and historical training samples are used as the comparison training set.

2) Comparison and Discussion: we first present a comprehensive comparison of the results obtained from EET-SA, SV-SA, GV-SA, PD-SA and the proposed FPD-SA. Four metrics, including recall, macro-F1, operate time and parameter acquisition, are used to assess the performance of the methods, with the results being the average values across 10 categories. The performance evaluation, as depicted in Fig. 4 and Table I, demonstrates the superiority of FPD-SA over the aforementioned techniques with respect to model accuracy and computational efficiency. The results depicted in Fig. 4(a) and (b) reveal that FPD-SA outperforms other SA techniques in terms of recall and macro-F1 across most scenarios. The additional computational cost of FPD-SA, as shown in Fig. 4(c), is negligible compared to the rapidly increasing cost of the compared SA methods. Furthermore, FPD-SA, as shown in Fig. 4(a) and (d), is capable of overcoming overfitting by

TABLE III
ONLINE MODEL PERFORMANCE COMPARISONS

Method	Metirc(Average)	Stage 0	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7
RVFL [6]	Recall	28.45	29.88	52.19	65.64	65.14	67.07	66.71	66.64
	Macro-F1	28.14	29.12	53.11	65.36	64.40	66.76	67.70	67.66
	CPU Time(s)	1.61E-01	4.45E-01	4.89E-01	5.73E-01	6.19E-01	6.82E-01	7.60E-01	8.62E-01
	Parameters	80.33×	14.78×	9.77×	7.64×	6.13×	5.15×	4.40×	2.40×
CRVFL [26]	Recall	34.99	58.44	64.07	65.59	67.44	68.93	69.66	70.66
	Macro-F1	35.81	57.58	62.55	64.48	66.70	68.22	69.46	70.47
	CPU Time(s)	8.88E-02	7.54E-02	8.91E-02	1.04E-01	1.39E-01	1.54E-01	1.59E-01	1.88E-01
	Parameters	40.46×	7.45×	4.92×	3.85×	3.09×	2.60×	2.21×	1.21×
SCN [8]	Recall	17.09	38.44	58.57	64.41	67.11	68.41	69.70	71.00
	Macro-F1	16.32	39.18	58.33	63.07	66.26	68.06	69.19	70.89
	CPU Time(s)	2.14E+01	3.19E+01	4.52E+01	5.46E+01	6.96E+01	9.02E+01	1.24E+02	1.33E+02
	Parameters	39.83×	7.33×	4.84×	3.79×	3.04×	2.55×	2.18×	1.19×
LSTM [3]*	Recall	47.68	53.23	56.00	60.19	63.44	66.97	70.24	72.04
	Macro-F1	45.328	52.7	55.96	58.98	62.25	66.78	70.18	71.74
	GPU Time(s)	4.45E-01	2.33E-01	2.38E-01	2.77E-01	2.42E-01	2.39E-01	2.38E-01	2.50E-01
	Parameters	4.18E+04×	7.69E+03×	5.08E+03×	3.97E+03×	3.19E+03×	2.68E+03×	2.29E+03×	1.25E+03×
GRU [27]*	Recall	47.08	52.64	57.64	60.54	64.70	69.34	70.74	72.81
	Macro-F1	47.13	50.37	58.01	60.17	64.54	68.09	70.33	72.15
	GPU Time(s)	3.48E-01	3.05E-01	3.05E-01	3.48E-01	2.99E-01	3.01E-01	3.01E-01	3.08E-01
	Parameters	3.10E+04×	5.70E+03×	3.77E+03×	2.95E+03×	2.36E+03×	1.99E+03×	1.70E+03×	9.24E+02×
BLS [9]*	Recall	29.10	41.81	44.83	58.60	59.72	61.38	63.23	64.36
	Macro-F1	29.37	41.96	44.07	58.72	60.67	62.47	64.29	65.22
	CPU Time(s)	3.31E-03	1.27E-02	1.42E-02	6.73E-03	2.45E-02	3.70E-02	3.08E-02	6.28E-02
	Parameters	0.80K(1.05×)	4.31K(1.04×)	6.60K(1.06×)	8.52K(1.07×)	10.64K(1.07×)	12.96K(1.09×)	15.48K(1.11×)	27.56K(1.08×)
SASO-BLS-M *	Recall	29.53	52.61	59.55	65.61	67.43	70.20	70.53	73.05
	Macro-F1	30.48	51.74	59.95	64.88	66.46	69.48	69.54	71.70
	CPU Time(s)	8.01E-03	2.08E-02	2.12E-02	9.10E-03	3.46E-02	5.29E-02	4.27E-02	8.31E-02
	Parameters	**0.67K(0.88×)**	**4.00K(0.97×)**	**5.39K(0.86×)**	**6.66K(0.83×)**	**7.64K(0.77×)**	**7.66K(0.65×)**	**8.66K(0.62×)**	**9.88K(0.39×)**
SASO-BLS-P *	Recall	31.41	48.06	60.60	66.20	67.90	71.12	72.35	73.58
	Macro-F1	33.01	47.39	60.32	65.12	66.84	70.58	71.46	72.94
	CPU Time(s)	9.84E-03	2.55E-02	2.78E-02	2.56E-02	4.56E-02	4.74E-02	5.32E-02	9.78E-02
	Parameters	0.76K(1×)	4.13K(1×)	6.25K(1×)	7.99K(1×)	9.96K(1×)	11.86K(1×)	13.89K(1×)	25.48K(1×)

[a] The superscript * represents online model.

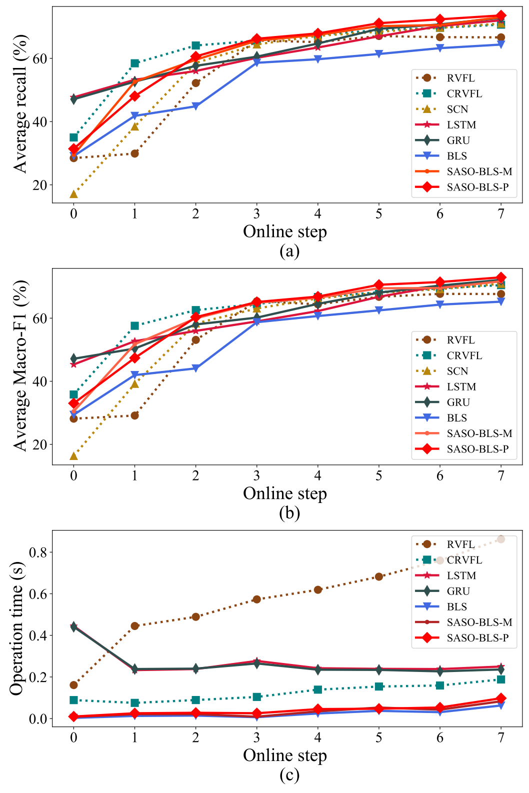

Fig. 5. Performance of each method in online mode

adaptively adjusting the model size. The results of Fig. 4 are detailed in Table I for review. In conclusion, it is evident that the FPD-SA approach demonstrates superior performance in balancing model accuracy, simplicity, and computational cost over existing SA methods.

The comparison of the proposed SASO-BLS with six other methods in offline classification is presented in Table II. The results in the penultimate column of Table II show that the SASO-BLS-P approach surpasses its counterparts, outstripping them with regards to the average recall. Additionally, in terms of the average macro-f1, the SASO-BLS-P method exhibits superior performance, securing the sixth position among seven competitors in this regard. These observations thus lend support to the efficacy of the SASO-BLS-P approach in achieving optimal performance in the offline task at hand. Concomitantly, SASO-BLS-M boasts of a notably lightweight architecture without compromising on its competitive performance. In light of this feature, various incarnations of SASO-BLS may be aptly chosen in accordance with specific requirements, rendering them exceedingly conducive for pragmatic industrial deployment.

The online experiment results of the compared methods and the proposed SASO-BLS are presented in Table III and Fig. 5. The first three methods in Table III display performance with $i/8$ of the training dataset in the ith step, while the remaining three methods and SASO-BLS can be performed online. Fig. 5(a) shows that SASO-BLS-P steadily improves its

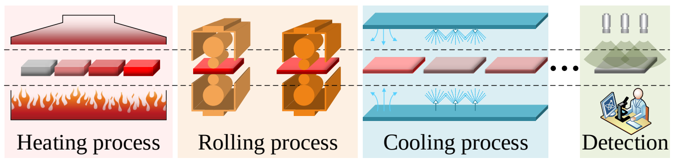

Fig. 6. Process of heavy plate production.

TABLE IV
DATA SET DESCRIPTION OF FIVE KINDS HEAVY PLATE

No.	Fault Description	Training Samples	Test Samples
1	Normal	2386	608
2	Bend	520	65
3	Horizon wave	404	54
4	Left wave	1834	378
5	Right wave	269	41

performance with increasing online steps, eventually achieving the best result among all methods. Note that the SASO-BLS series models, in its early stages of online processing, may demonstrate a less optimal level of performance in contrast to alternative methods. This phenomenon arises from the algorithm's inclination towards reducing model complexity in order to bolster its generalization capabilities, as demonstrated by the parameter statistics presented in Table III. For instance, RVFL and CRVFL have nearly 40 and 80 times more parameters than SASO-BLS-P in the early step, and LSTM and GRU have 40K and 30K times more parameters, respectively. Furthermore, SASO-BLS-M displays a more radical disposition towards compressing its structure in response to newly acquired data, prioritizing structural simplicity whilst endeavoring to sustain a competitive level of performance. On the contrary, Fig. 5(a) and Fig. 5(b) show that BLS method evinces a meager improvement as the number of online steps increases, thus substantiating the efficacy of the proposed self-organizing mechanism in ameliorating the BLS's online learning generalization capability. Finally, Fig. 5(c) demonstrates SASO-BLS's superiority in efficiency over other methods.

B. Case Studies on Real-world Datasets

Heavy plate products are widespread in construction, transportation, and manufacturing industries, where plate shape is a critical quality indicator. Early detection of plate shape faults is crucial for optimizing operations and improving production pass rates. To assess the proposed method, a real-world dataset of heavy plate production from a steel plate is adopted. The dataset, which is comprised of 116 process variables and 5 types of heavy plates across three processes (heating, rolling, and cooling), is depicted in Fig. 6 and described in Table IV. As Table IV reveals, the dataset presents a class-imbalance challenge that tests the generalization ability of models. Thus, weighted average accuracy and weighted average maroc-F1 score are utilized to evaluate the overall model performance objectively.

1) Parameter Setting: The parameters of the SA method comparison experiment are identical to the previous case study. In offline experiments, the parameters of RVFL, CRVFL, SCN,

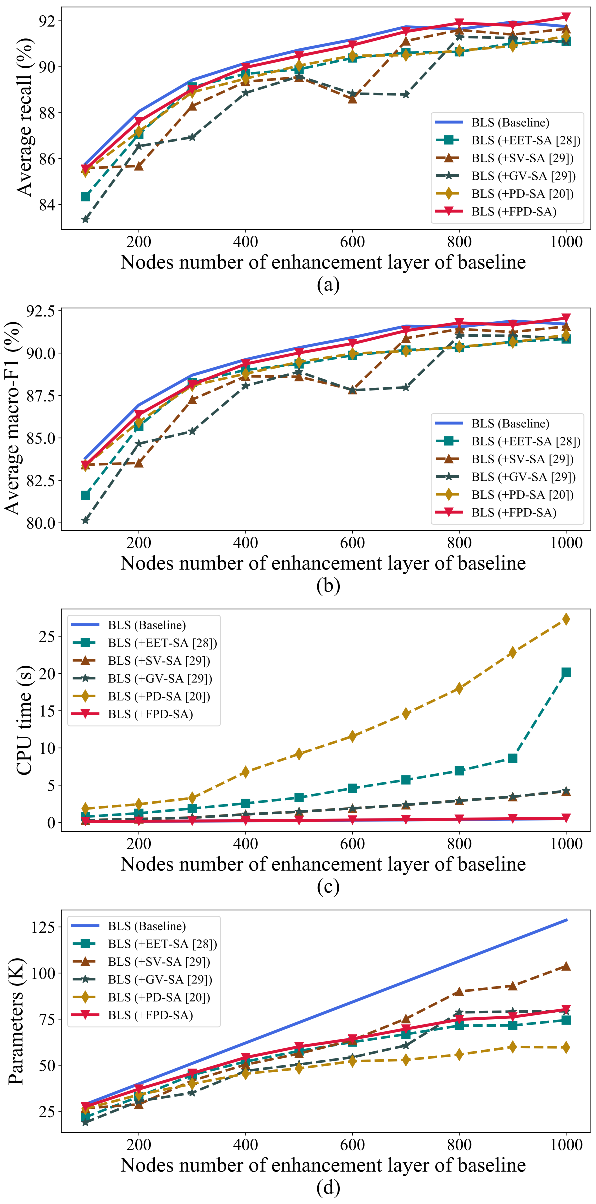

Fig. 7. Comparison with the existing SA methods on the heavy plate production datasets

BLS, SASO-BLS-M, and SASO-BLS-P remain unchanged. The maximum step, T_{max}, is set to 30, and the preset threshold, ψ, is set to 0.001. The number of nodes in the hidden layer of LSTM and GRU is 1000, with a block number of 1. The online experiment setup is also consistent with the previous case.

2) Comparison and Discussion: As with the prior case, we evaluate the effectiveness of the FPD-SA method utilizing the same competitors and metrics. Detailed graphical representation of these results is presented in Fig. 7, while a tabular summary of the findings is presented in Table V. As can be observed from Fig. 7(a) and (b), FPD-SA delivers a remarkable performance when juxtaposed against other SA methods, while the performance of the variance-based techniques is characterized by a degree of instability. Moreover, while accuracy improvement of BLS method slows down with

TABLE V
SENSITIVITY ANALYSIS METHOD PERFORMANCE COMPARISONS

Structure (Feature+Enhance)											
Method	Metric (Average)	100+100	100+200	100+300	100+400	100+500	100+600	100+700	100+800	100+900	100+1000
BLS (Baseline)	Recall	85.77	88.05	89.42	90.16	90.73	91.18	91.74	91.63	91.95	91.75
	Macro-F1	83.80	86.94	88.70	89.62	90.32	90.92	91.59	91.54	91.89	91.71
	CPU Time (s)	0.13	0.15	0.18	0.21	0.24	0.30	0.33	0.39	0.44	0.50
	Parameters	28.81K	39.91K	51.01K	62.11K	73.21K	84.31K	95.41K	106.51K	117.61K	128.71K
BLS (+EET-SA [20])	Recall	84.34	87.06	89.12	89.69	89.89	90.38	90.61	90.65	91.02	91.11
	Macro-F1	81.62	85.69	88.27	89.01	89.36	89.87	90.18	90.32	90.67	90.83
	CPU Time (s)	(+0.64)	(+1.08)	(+1.69)	(+2.35)	(+3.09)	(+4.29)	(+5.38)	(+6.54)	(+8.18)	(+19.68)
	Parameters	21.82K	33.00K	44.73K	52.00K	57.74K	62.57K	66.84K	71.52K	71.60K	74.49K
BLS (+SV-SA [20])	Recall	85.58	85.68	88.29	89.35	89.54	88.60	91.12	91.60	91.40	91.65
	Macro-F1	83.42	83.53	87.26	88.63	88.61	87.84	90.87	91.42	91.24	91.57
	CPU Time (s)	(+0.17)	(+0.31)	(+0.46)	(+0.88)	(+1.19)	(+1.59)	(+2.03)	(+2.54)	(+3.01)	(+3.69)
	Parameters	26.68K	28.79K	41.66K	50.33K	56.31K	63.65K	75.25K	89.99K	93.10K	103.80K
BLS (+GV-SA [20])	Recall	83.35	86.54	86.93	88.86	89.58	88.83	88.79	91.30	91.25	91.08
	Macro-F1	80.14	84.66	85.39	88.07	88.90	87.81	87.98	91.04	91.02	90.87
	CPU Time (s)	(+0.17)	(+0.30)	(+0.46)	(+0.87)	(+1.20)	(+1.58)	(+2.04)	(+2.52)	(+3.01)	(+3.73)
	Parameters	18.98K	30.51K	35.09K	46.93K	50.42K	54.34K	60.73K	78.69K	79.12K	79.24K
BLS (+PD-SA [20])	Recall	85.45	87.18	88.89	89.48	90.05	90.48	90.50	90.70	90.90	91.34
	Macro-F1	83.38	85.94	88.09	88.78	89.48	89.97	90.12	90.37	90.64	91.07
	CPU Time (s)	(+1.72)	(+2.29)	(+3.10)	(+6.56)	(+8.96)	(+11.27)	(+14.27)	(+17.61)	(+22.38)	(+26.80)
	Parameters	26.44K	34.01K	39.94K	45.43K	48.33K	52.20K	52.91K	55.81K	59.95K	59.68K
BLS (+FPD-SA)	Recall	85.54	87.62	89.01	89.97	90.47	90.94	91.53	91.90	91.80	92.16
	Macro-F1	83.39	86.38	88.14	89.36	90.01	90.56	91.32	91.78	91.66	92.06
	CPU Time (s)	(+0.01)	(+0.01)	(+0.01)	(+0.02)	(+0.03)	(+0.03)	(+0.04)	(+0.05)	(+0.06)	(+0.07)
	Parameters	27.54K	37.08K	45.66K	54.22K	60.07K	64.27K	69.66K	74.83K	76.20K	80.27K

[a] In the CPU time comparison, the values in brackets indicate the time consumed by the corresponding SA methods.

TABLE VI
OFFLINE MODEL PERFORMANCE COMPARISONS

Method	Metric	Norm	Fault 1	Fault 2	Fault 3	Fault 4	WA	Para
RVFL [6]	Rec	96.38	13.85	35.18	92.59	0	84.12	0.12M
	F1	92.07	24.33	49.99	84.64	0	83.15	
CRVFL [26]	Rec	96.71	36.92	53.70	93.39	17.07	87.35	58.56K
	F1	95.45	46.15	65.17	86.52	27.45	86.91	
SCN [8]	Rec	97.37	49.23	66.67	91.53	58.53	89.88	59.08K
	F1	96.26	57.14	69.91	89.17	67.60	89.62	
LSTM [3]	Rec	95.72	63.08	74.07	92.33	68.29	90.75	0.42M
	F1	96.36	65.60	74.07	90.30	71.80	90.71	
GRU [27]	Rec	94.08	58.46	72.22	91.01	80.49	89.53	0.31M
	F1	96.22	60.32	67.24	89.24	73.34	89.77	
BLS [9]	Rec	96.38	64.62	**75.93**	91.01	**87.80**	91.54	0.33M
	F1	97.50	62.69	72.57	91.37	**80.00**	91.74	
SASO-BLS(MV)	Rec	97.20	**69.23**	70.37	91.53	80.49	91.88	**55.20K**
	F1	97.04	69.23	73.08	91.65	77.65	91.87	
SASO-BLS(PV)	Rec	**98.19**	64.62	74.07	**93.65**	78.05	**92.93**	0.19M
	F1	**97.63**	**71.80**	**77.67**	**92.31**	78.05	**92.83**	
+/=/−(MV)	Rec	5/0/2	7/0/0	3/0/4	2/1/4	5/1/1	6/0/1	7/0/0
	F1	5/0/2	6/0/1	5/0/2	6/0/1	5/0/2	6/0/1	
+/=/−(PV)	Rec	7/0/0	7/0/0	5/1/1	7/0/0	4/0/3	7/0/0	3/0/4
	F1	7/0/0	7/0/0	7/0/0	7/0/0	6/0/1	7/0/0	

[a] The last two lines lay out the rank sum test results of SASO-BLS-M and SASO-BLS-P, where each version is better than (+), worse than (-) and equal to (=) others. The best results for 5 classes and their weighted average (WA) values across all methods are marked in **boldface.**

increasing model size, FPD-SA can still enhance performance by adjusting the model scale. Fig. 7(c) demonstrates superiority of FPD-SA over its counterparts in terms of computational efficiency, particularly in comparison with EET-SA and PD-SA. From Fig. 7(d) one may find that, although the FPD-SA does not boast the most diminutive dimensions at model size, it possesses a remarkable capacity to balance optimal structural attributes with promising performance measures.

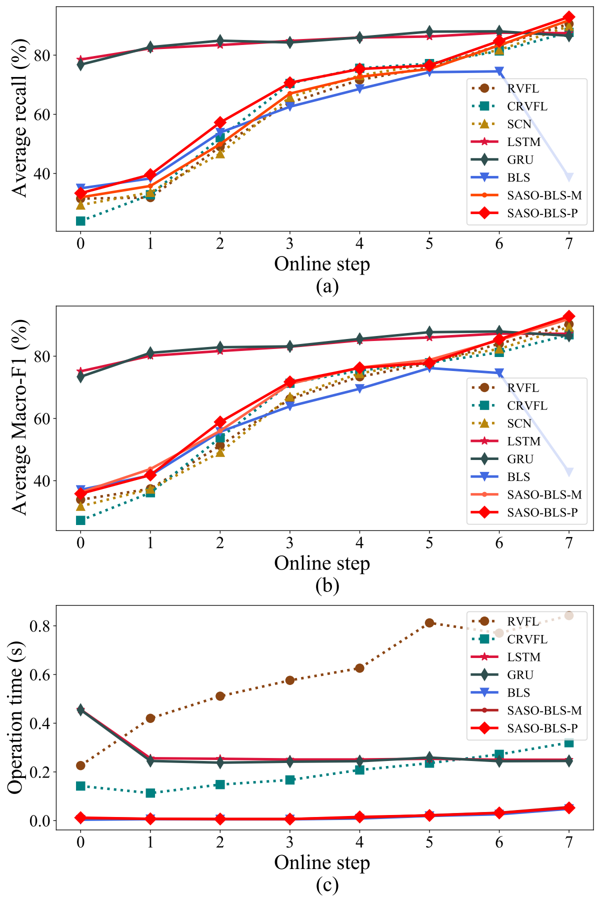

Fig. 8. Performance of each method in online mode

TABLE VII
ONLINE MODEL PERFORMANCE COMPARISONS

Method	Metirc(Average)	Stage 0	Stage 1	Stage 2	Stage 3	Stage 4	Stage 5	Stage 6	Stage 7
RVFL [6]	Weighted Recall	31.50	31.85	49.04	63.96	71.64	75.92	83.33	90.58
	Weighted F1-score	33.92	37.40	51.31	66.28	73.39	77.74	83.98	90.28
	CPU Time(s)	2.26E-01	4.20E-01	5.11E-01	5.76E-01	6.26E-01	8.12E-01	7.70E-01	8.42E-01
	Parameters	162.63×	8.47×	4.76×	4.47×	4.52×	3.61×	3.25×	2.00×
CRFVL [26]	Weighted Recall	23.91	32.81	52.09	70.42	75.57	77.14	81.41	87.70
	Weighted F1-score	27.26	36.15	53.80	71.31	75.39	77.89	81.22	86.88
	CPU Time(s)	1.42E-01	1.13E-01	1.48E-01	1.67E-01	2.08E-01	2.36E-01	2.72E-01	3.20E-01
	Parameters	77.51×	4.04×	2.27×	2.13×	2.15×	1.72×	1.57×	0.95×
SCN [8]	Weighted Recall	29.32	33.60	46.68	65.79	73.12	77.05	81.76	89.79
	Weighted F1-score	31.80	37.31	49.13	67.01	74.45	78.58	82.33	89.38
	CPU Time(s)	1.50E+01	2.37E+01	3.37E+01	4.09E+01	5.31E+01	6.24E+01	7.83E+01	8.82E+01
	Parameters	78.12×	4.08×	2.29×	2.15×	2.17×	1.73×	1.56×	0.96×
LSTM [3]*	Weighted Recall	78.53	82.29	83.42	84.82	85.95	86.30	87.61	87.43
	Weighted F1-score	75.15	80.12	81.66	83.04	85.10	86.00	87.26	87.19
	GPU Time(s)	4.56E-01	2.56E-01	2.54E-01	2.51E-01	2.51E-01	2.54E-01	2.50E-01	2.50E-01
	Parameters	5.55E+02×	29.76×	16.66×	15.66×	15.78×	12.63×	11.36×	7.00×
GRU [27]*	Weighted Recall	76.79	82.72	84.90	84.29	85.95	87.96	88.05	86.56
	Weighted F1-score	73.40	81.08	82.87	83.15	85.55	87.72	87.95	86.56
	GPU Time(s)	4.55E-01	2.45E-01	2.38E-01	2.42E-01	2.44E-01	2.59E-01	2.44E-01	2.45E-01
	Parameters	4.20E+02×	21.89×	12.30×	11.55×	11.67×	9.32×	8.40×	5.18×
BLS [9]*	Weighted Recall	34.99	38.31	53.84	62.57	68.59	74.25	74.52	38.66
	Weighted F1-score	37.08	41.71	55.72	63.90	69.54	76.18	74.60	42.70
	CPU Time(s)	3.63E-03	5.98E-03	5.29E-03	5.36E-03	8.62E-03	1.92E-02	2.59E-02	4.83E-02
	Parameters	1.88K(2.47×)	16.22K(1.11×)	30.92K(1.23×)	34.16K(1.25×)	34.16K(1.25×)	44.48K(1.32×)	51.86K(1.36×)	99.60K(1.62×)
SASO-BLS-M*	Weighted Recall	31.93	35.78	49.91	67.02	72.69	75.31	83.33	91.80
	Weighted F1-score	36.28	43.76	56.02	71.10	76.33	78.82	85.17	91.95
	CPU Time(s)	1.01E-02	7.21E-03	6.46E-03	6.70E-03	1.21E-02	2.21E-02	3.22E-02	5.55E-02
	Parameters	**0.34K(0.45×)**	**3.48K(0.24×)**	**5.16K(0.20×)**	**6.32K(0.23×)**	**8.17K(0.30×)**	**9.54K(0.28×)**	**12.87K(0.34×)**	**21.48K(0.35×)**
SASO-BLS-P*	Weighted Recall	33.33	39.62	57.24	70.68	75.31	76.53	84.82	92.93
	Weighted F1-score	35.83	41.82	58.87	71.75	76.27	77.80	85.43	92.83
	CPU Time(s)	1.21E-02	7.32E-03	6.52E-03	6.23E-03	1.52E-02	2.11E-02	3.15E-02	5.23E-02
	Parameters	0.76K(1×)	14.62K(1×)	25.23K(1×)	27.28K(1×)	27.30K(1×)	33.79K(1×)	38.02K(1×)	61.62K(1×)

[a] The superscript * represents online model.

The results of offline experiment are presented in Table VI, which shows the remarkable competency of SASO-BLS-P in identifying the majority of fault categories, alongside obtaining the best results on all weighted average metrics. SASO-BLS-P achieved a noteworthy 1.47% improvement in weighted average recall while reducing parameters by 57% in comparison to BLS, demonstrating its efficacy in optimizing performance while minimizing resource utilization. In comparison, SASO-BLS-M emerges as a highly competitive approach, exhibiting the second-best performance while boasting the lightest structure. Meanwhile, SASO-BLS-M presents an 83.66% reduction in structure while still managing to secure a 0.33% increase in weighted average recall when compared to BLS, a testament to its ability to achieve comparable performance with a more parsimonious architecture. Taken together, these results suggest that SASO-BLS-P and SASO-BLS-M are highly promising approaches with the capability to outperform BLS on class-imbalanced problems.

Table VII and Fig. 8 present the results of online learning for all methods. In this case, the data is also divided into eight parts and fed into the model sequentially. As shown in Table VII, LSTM and GRU manifest superior performance in the early steps, but their model size is substantially larger than SASO-BLS series models, and the subsequent improvement in accuracy is restricted over time. In contrast, SASO-BLS initially performs comparably to offline models with a small structure, resulting in lower accuracy but higher generalization potential. Furthermore, one can observe from 8(a) and (b) that BLS encounters significant performance degradation as the online step increase. SASO-BLS-P address the issue and ultimately achieving the best results. Note that although SASO-BLS-M may not possess the same degree of superiority in accuracy as SASO-BLS-P, it achieve a competitive performance while possessing a definitive advantage in terms of model size. In terms of computational efficiency, SASO-BLS-M and SASO-BLS-P have similar level of computation to BLS, making them suitable for online learning, as depicted in Fig. 8(c).

V. CONCLUSION

This paper propose an efficient online self-organizing framework, SASO-BLS, capable of dynamically adjusting its structure during incremental learning. The SASO-BLS is born with the incremental ability of BLS and incorporated with a novel general SA-based model compression method, named FPD-SA. The computation cost of FPD-SA is insensitive to data size due to a given fast evaluation criterion. By introducing FPD-SA into BLS, we have derived the quick updating algorithms of SASO-BLS under offline and extend to online mode, addressing BLS's performance degradation during incremental learning. Furthermore, we offer high-performance (SASO-BLS-P) and miniature (SASO-BLS-M) options of SASO-BLS to satisfy different requirements. The obtained results demonstrates SASO-BLS's remarkable performance in both offline and online tasks, as well as superior handling of class-imbalance tasks with low computational overhead, which shows considerable potential for practical application.

In future research, we aim to extend the proposed self-organizing method to more powerful deep learning models, such as various transformer models. It is widely recognized that adapting the structures of these models to new data is a computationally intensive task. Therefore, it is of important to develop a self-organizing algorithm that can effectively and efficiently determine the optimal configuration of large-scale deep models.

References

[1] X. Dai and Z. Gao, "From model, signal to knowledge: A data-driven perspective of fault detection and diagnosis," *IEEE Trans. Ind. Informat.*, vol. 9, no. 4, pp. 2226–2238, 2013.

[2] W. Sun, R. Zhao, R. Yan, S. Shao, and X. Chen, "Convolutional discriminative feature learning for induction motor fault diagnosis," *IEEE Trans. Ind. Informat.*, vol. 13, no. 3, pp. 1350–1359, 2017.

[3] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[4] X. Gong, T. Zhang, C. L. P. Chen, and Z. Liu, "Research review for broad learning system: Algorithms, theory, and applications," *IEEE Trans. Cybern.*, pp. 1–29, 2021.

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

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

[7] M. Li and D. Wang, "Insights into randomized algorithms for neural networks: Practical issues and common pitfalls," *Inform. Sci.*, vol. 382-383, pp. 170–178, 2017.

[8] D. Wang and M. Li, "Stochastic configuration networks: Fundamentals and algorithms," *IEEE Trans. Cybern.*, vol. 47, no. 10, pp. 3466–3479, 2017.

[9] C. L. 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, 2018.

[10] X. Pu and C. Li, "Online Semisupervised Broad Learning System for Industrial Fault Diagnosis," *IEEE Trans. Ind. Informat.*, vol. 17, no. 10, pp. 6644–6653, 2021.

[11] 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, 2020.

[12] Y. Fu, H. Cao, and X. Chen, "Adaptive Broad Learning System for High-Efficiency Fault Diagnosis of Rotating Machinery," *IEEE Trans. Instrum. Meas.*, vol. 70, 2021.

[13] L. Zhang, J. Li, G. Lu, P. Shen, M. Bennamoun, S. A. A. Shah, Q. Miao, G. Zhu, P. Li, and X. Lu, "Analysis and variants of broad learning system," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 52, no. 1, pp. 334–344, 2022.

[14] S. Wang, T. Zhou, and J. Bilmes, "Jumpout : Improved dropout for deep neural networks with ReLUs," in *Proc. Conf. on International Conference on Machine Learning*, 2019, pp. 6668–6676.

[15] H.-G. Han, W. Lu, Y. Hou, and J.-F. Qiao, "An adaptive-pso-based self-organizing rbf neural network," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 1, pp. 104–117, 2018.

[16] J. Qiao, G. Wang, W. Li, and X. Li, "A deep belief network with PLSR for nonlinear system modeling," *Appl. Soft Comput.*, vol. 65, pp. 170–183, 2018.

[17] F. E. Fernandes and G. G. Yen, "Automatic searching and pruning of deep neural networks for medical imaging diagnostic," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 32, no. 12, pp. 5664–5674, 2021.

[18] H. Han and J. Qiao, "A self-organizing fuzzy neural network based on a growing-and-pruning algorithm," *IEEE Trans. Fuzzy Syst.*, vol. 18, no. 6, pp. 1129–1143, 2010.

[19] F. Fernández-Navarro, M. Carbonero-Ruz, D. Becerra Alonso, and M. Torres-Jiménez, "Global sensitivity estimates for neural network classifiers," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 11, pp. 2592–2604, 2017.

[20] P. A. Kowalski and M. Kusy, "Sensitivity analysis for probabilistic neural network structure reduction," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 5, pp. 1919–1932, 2018.

[21] J. de Jesus Rubio, "Sofmls: Online self-organizing fuzzy modified least-squares network," *IEEE Trans. Fuzzy Syst.*, vol. 17, no. 6, pp. 1296–1309, 2009.

[22] N. Wang, M. J. Er, and X. Meng, "A fast and accurate online self-organizing scheme for parsimonious fuzzy neural networks," *Neurocomputing*, vol. 72, no. 16, pp. 3818–3829, 2009.

[23] H. Han, L. Zhang, X. Wu, and J. Qiao, "An efficient second-order algorithm for self-organizing fuzzy neural networks," *IEEE Trans. Cybern.*, vol. 49, no. 1, pp. 14–26, 2019.

[24] R. Das, S. Sen, and U. Maulik, "A survey on fuzzy deep neural networks," *ACM Comput. Surv.*, vol. 53, no. 3, pp. 1–25, 2020.

[25] F. Pianosi, F. Sarrazin, and T. Wagener, "A matlab toolbox for global sensitivity analysis," *Environ Modell Softw*, vol. 70, pp. 80–85, 2015.

[26] L. Zhang and P. N. Suganthan, "Visual tracking with convolutional random vector functional link network," *IEEE Trans. Cybern.*, vol. 47, no. 10, pp. 3243–3253, 2017.

[27] K. Cho *et al.*, "Learning phrase representations using RNN encoder–decoder for statistical machine translation," in *Proc. Conf. on Empirical Methods in Natural Lang. Process (EMNLP)*, 2014, pp. 1724–1734.

[28] R. A. Cropp and R. D. Braddock, "The new morris method: an efficient second-order screening method," *Reliab Eng Syst Safe*, vol. 78, no. 1, pp. 77–83, 2002.

[29] A. Saltelli, P. Annoni, I. Azzini, F. Campolongo, M. Ratto, and S. Tarantola, "Variance based sensitivity analysis of model output. design and estimator for the total sensitivity index," *Comput. Phys. Commun.*, vol. 181, no. 2, pp. 259–270, 2010.