A Fault Aware Broad Learning System for Concurrent Network Failure Situations

ADEGOKE, Muideen; WONG, Hiu Tung; LEUNG, Chi Sing

Published in:
IEEE Access

Published: 01/01/2021

Document Version:
Final Published version, also known as Publisher’s PDF, Publisher’s Final version or Version of Record

License:
CC BY

Publication record in CityU Scholars:
Go to record

Published version (DOI):
10.1109/ACCESS.2021.3066217

Publication details:
ADEGOKE, M., WONG, H. T., & LEUNG, C. S. (2021). A Fault Aware Broad Learning System for Concurrent Network Failure Situations. IEEE Access, 9, 46129-46142. Article 9380139. https://doi.org/10.1109/ACCESS.2021.3066217

Citing this paper
Please note that where the full-text provided on CityU Scholars is the Post-print version (also known as Accepted Author Manuscript, Peer-reviewed or Author Final version), it may differ from the Final Published version. When citing, ensure that you check and use the publisher's definitive version for pagination and other details.

General rights
Copyright for the publications made accessible via the CityU Scholars portal is retained by the author(s) and/or other copyright owners and it is a condition of accessing these publications that users recognise and abide by the legal requirements associated with these rights. Users may not further distribute the material or use it for any profit-making activity or commercial gain.

Publisher permission
Permission for previously published items are in accordance with publisher's copyright policies sourced from the SHERPA RoMEO database. Links to full text versions (either Published or Post-print) are only available if corresponding publishers allow open access.

Take down policy
Contact lbscholars@cityu.edu.hk if you believe that this document breaches copyright and provide us with details. We will remove access to the work immediately and investigate your claim.

Download date: 30/11/2024
A Fault Aware Broad Learning System for Concurrent Network Failure Situations

MUIDEEN ADEGOKE, HIU TUNG WONG, AND CHI SING LEUNG, (Senior Member, IEEE)

Department of Electrical Engineering, City University of Hong Kong, Hong Kong
Corresponding author: Chi Sing Leung (eeleungc@cityu.edu.hk)

This work was supported research grant from City University of Hong Kong under Grant 7005223 and Grant 7610431.

ABSTRACT The broad learning system (BLS) framework gives an efficient solution for training flat-structured feedforward networks and flat structured deep neural networks. However, the classical BLS model and other variants focus on the faultless situation only, where enhancement nodes, feature mapped nodes, and output weights of a BLS network are assumed to be realized in a perfect condition. When a trained BLS network suffers from coexistence of weight/node failures, the trained network has a greatly degradation in its performance if a countermeasure is not taken. In order to reduce the effect of weight/node failures on the BLS network’s performance, this paper proposes an objective function for enhancing the fault aware performance of BLS networks. The objective function contains a fault aware regularizer term which handles the weight/node failures. A learning algorithm is then derived based on the objective function. The simulation results show that the performance of the proposed fault aware BLS (FABLS) algorithm is superior to the classical BLS and two state-of-the-arts BLS algorithms, namely correntropy criterion BLS (CBLS) and weighted BLS (WBLS).

INDEX TERMS Fault tolerance, broad learning system, regression, multiplicative noise, open fault, incremental learning.

I. INTRODUCTION

Without doubt, human brain is capable to handle fault and noise situations [1]. For instance, human being is able to recognize a partially occluded object without much difficulty. Moreover, human brain can still operate with a certain ability when a meager amount of synapses or neurons are malfunctioned. As the idea of artificial neural networks (ANNs) is based on human brain, it was presumed that a trained ANN should have an inherent ability against weight/node failures. In the inception of ANN researches, a common misconception was that ANNs have intrinsic tolerance against any form of failures, such as weight noise, node noise, and open fault. Meanwhile, if ANNs suffer from weight failures or node failures, without any fault aware measures in place, then the performance of such ANNs is greatly degraded [2], [3].

In the realization of a trained ANN in electronic systems, weight/node failures are not avoidable [4], [5]. For instance, when digital technology is employed to represent the weights of a trained ANN, there are precision errors and the errors can be modelled as multiplicative noise. Also, when analog technology is used, hardware components do have precision errors and thermal noise do exist. Furthermore, damages such as open weight faults or open node faults may be introduced when the communication link between two nodes are broken [6]. Besides, when realizing a ANN using the very large scale integration (VLSI) at nano-scale level, transient noise or failures are unavoidable during the operation [7], [8].

One of basic requirements of many electronic systems is dependability. Many safety critical systems would be using complex ANNs, such as deep neural network (DNN). It is expected that such systems must have high dependability. Hence, it is vital or crucial to embed good fault aware techniques in such systems, such as build-in self test and error detection/correction coding [9].

To enhance the behaviour of a well trained ANN during the failure situation, it is paramount to figure out how fault affects the behavior or performance of a trained ANN in an analytical way. There are related studies on fault resistant ability of ANNs. Recently, in [10], empirical results of a fault tolerant ability of a small convolutional neural network (CNN) are presented. Furthermore, a survey paper reported the effects of those imperfect conditions for various neural network models, including feedforward networks and
radial basis function (RBF) networks [11], [12]. For RBF networks, fault aware ability and some fault aware training algorithms were extensively investigated in [3], [4].

The broad learning system (BLS) [13], [14] concept is very efficient for constructing flat structured ANN, and BLS networks are universal approximators [15]. In addition, due to its good properties, the BLS concept has caught many attentions [16]–[20]. Although many BLS results were reported, commonly available BLS algorithms focus on the faultless situation. To the best of our knowledge, there are not many literatures related to fault tolerant BLS networks.

This paper studies the effect of coexistence of weight/node failures on BLS [13]. It first examines the behaviour of the original BLS model under the coexistence of several fault/noise, including open weight fault on output weights, multiplicative noise on the nodes, and multiplicative noise on the output weights. To compensate the degradation effect from the aforementioned noise/fault, an objective function for enhancing the fault aware performance of BLS is developed. The objective function contains a fault aware regularizer term. The learning algorithm for minimizing the fault aware objective function is proposed. The proposed algorithm is called fault aware BLS (FABLS) algorithm. Several real life datasets from the UCI repository [21] are used to verify the effectiveness of the proposed FABLS algorithm. From the simulation results, our proposed FABLS is superior to the original BLS [13], and two state of arts algorithms, namely weighted BLS (WBLS) [19] and correntropy criterion based BLS (CBLS) [20].

In summary, our major contributions are as follows.

- This paper investigates the effect of the coexistence of several fault/noise on the performance of BLS network.
- A fault aware objective function is proposed. The objective function contains a fault aware regularizer term which mitigates the effects of weight/node failures.
- Based on the developed fault aware objective function, the fault aware BLS (FABLS) algorithm is developed.
- From the simulation results, our proposed FABLS is superior to the comparison BLS algorithms.

The paper is presented as follows. Section II discusses related works, Section III presents the background on the traditional broad learning system (BLS) network and model. Section IV provides the concept of coexistence of a weight failure situation and the weight failure models. The fault-aware objective function and the learning algorithm are developed in Section V. Section VI uses several real life datasets to demonstrate the superiority of the proposed FABLS algorithm. Finally, Section VII provides conclusion remarks. Table 1 lists some key notations used in this paper.

### II. RELATED WORKS

#### A. FAULT TOLERANT IN ANNS

One fascinating feature or characteristic of ANNs is fault tolerance. However, if proper learning steps are not adopted, a trained ANN usually has poor fault tolerant ability. In the last two decades, various results on fault-tolerant ability of ANNs were reported. For instance, the common node fault situations are stuck-at-zero and stuck-at-one [6], [22]–[24]. For stuck-at-zero, the output of a faulty node is stuck at zero. For stuck-at-one, the output of a faulty node is stuck at one. Another kind of “stuck-at” fault is “stuck-at-any-values”, in which the output of a faulty node is stuck at a value in between the output range of the activation function. This stuck-at-any-values situation was analyzed in [25]. Apart from node fault/failure, connection weights in a ANN may also have fault or failure [26]–[28]. In the open weight fault, the connection weight between two nodes is disconnected. In the weight noise case, due to the technology limitation, there are some precision errors/noise in the realization. The common used model to describe the behaviour of the weight noise is the multiplicative weight noise model [2], [3], [26].

There are many methods to handle node/weight failures. Some approaches inject random failure during training [22], [24], [25]. The demerit of this approach is that a large number of training cycles is required to achieve some certain performance. Otherwise, the statistical information of the fault cannot be expressed. Furthermore, another common approach for protecting fault/failure is replication approach...
In the replication approach, a number of networks from a trained network are replicated. Hence, the overall output is equal to the median of the outputs of the replicated networks. The disadvantage of this approach is that it requires extra resources. Besides, a median circuit is more complicated than the traditional linear neuron.

There are other approaches which focus on weight noise/fault or combination of two failures on a ANN. For instance, the works in [26]–[28] investigated the effect of failure situation on the performance of faulty radial basis function (RBF). Parara and Catala in [32] showed how a fault tolerant RBF network can be obtained based on weight decay regularizer [33]. Furthermore, from the perspective of model sensitivity, an approach called explicit regularizer to attain a multi-layer perceptron (MLP) [34]–[36] or RBF [37], [38] was proposed. This approach can handle or tolerate multiplicative weight noise. Recently, failure tolerance ability of the extreme learning machine (ELM) were studied [12], [39]–[42].

Additionally, in [10], fault tolerant ability of a convolutional neural network (CNN) was investigated. Besides, in [43] fault tolerant methods based on diversifying learning were proposed to improve deep neural networks (DNNs) dependability. Moreover, in [44] Xu et al. studied the feasibility of developing fault-tolerant deep learning systems based on model redundancy.

Similarly, recent literatures indicate that soft errors are unavoidable in modern electronic systems, from edge computing devices to supercomputers [45], [46] due to multiple factors [47] such as device aging, wear devices, and high-energy cosmic radiation. In [48], an algorithm for improving fault tolerance of CNNs was proposed.

Furthermore, Leung et al. [49] investigated the effect of open weight faults and multiplicative noise on the performance of bidirectional associative memories (BAM). Moreover, in [50] literature review on fault and noise tolerance in neural networks is discussed. For fault tolerant systems, principles and concepts, interested readers are referred to [51]–[54].

**B. BLS**

The BLS concept [13] is a recent technique for broad connection of hidden nodes of a neural network. As shown in Fig. 1, these nodes are arranged in a broad sense. In a BLS network, there are two types of hidden nodes, feature mapped nodes, and enhancement nodes.

The feature mapped nodes form a layer, namely feature mapped node layer. The input connection weights of these nodes are constructed based on auto-encoder learning. That means, the feature mapped node layer is designed for feature extraction. The outputs of the feature mapped layer are connected to the output layer and another layer, calling enhancement node layer. The enhancement nodes receive the inputs from the feature nodes. In the original BLS learning, the connection weights between the input to the enhancement node are generated in a random manner.

As shown in Fig. 1, the output layer collects the information from the feature node layer and enhancement node layer. In other words, the outputs of feature nodes and enhancement nodes are concatenated in a wide manner and are fed to the output layer. The output weights could be obtained by a least-mean square technique based on Moore-Penrose’s generalized inverse [55], [56]. The overall training computational cost in BLS is not excessive and is much lower than that of other classical training algorithms, such as meta-heuristic techniques (differential evolution [57], particle swarm optimization [58], and gradient descent methods [59]).

As the BLS framework provides efficient ways for constructing a flat structured ANN, and a BLS network owns the universal approximation ability [13], [14], the BLS framework has attracted increasing attentions. Many variants of BLS models and algorithm were proposed to enhance its performance [16], [60], [61]. Besides, many applications of BLS were reported in the last several years.

In [16], a hybrid model of BLS and fuzzy concept was proposed, in which the feature nodes are replaced with some fuzzy subsystems. In [18], the BLS system is applied to fault diagnosis in rotor systems. In [62], a BLS model was proposed for object detection using the event data obtained from event cameras. In this model, the incremental learning concept was also used for constructing the resultant BLS network. In [63], the BLS concept was used for processing hyperspectral images. Apartment from applications, there are modifications on its learning algorithm [19], [20]. In [19], the weighted BLS (WBLs) algorithm was proposed for compensate the effect of noise and outliers. In [20], the corentropy criterion based BLS (CBLS) was proposed for handling outlier data based on the maximum correntropy criterion (MCC).

Although the BLS concept is a quite promising framework, the original BLS and other variants focus on the faultless situation only, where a BLS network is assumed to be perfectly implemented. To the best of our knowledge, there are not many literatures related to fault tolerant BLS networks. This paper studies the behaviour of original BLS which are concurrently affected by node noise, weight noise, and open weight fault. Also, a fault aware objective function is derived. Furthermore, based on the proposed objective function, a fault...
tolerant BLS (FABLS) is proposed to mitigate the effect of network failures.

III. BLS NETWORKS AND NETWORK FAILURE
A. OPERATION OF BLS NETWORKS
This subsection gives an introduction to the operation of a BLS network. Consider that a BLS network is designed for solving nonlinear regression problem. Let \(x \in \mathbb{R}^p\) and \(o \in \mathbb{R}\), respectively, be the input data and output of a BLS network. First, for clear and easy presentation, this paper denotes the input \(x\) augmented with 1 as \(\mathbf{x} = [x^T, 1]^T\).

1) FEATURE MAPPED NODES
In the BLS network, the feature mapped nodes are grouped into \(n\) groups. Each group is used to extract different features. The \(i\)-th group has \(f_i\) feature mapped nodes. Hence, the total number of feature mapped nodes is

\[
f = \sum_{i=1}^{n} f_i.
\]

For the \(i\)-th group, there is a learned projection matrix, given by

\[
\Psi_i = \begin{pmatrix}
\psi_{i,1,1} & \cdots & \psi_{i,1,(D+1)} \\
\vdots & \ddots & \vdots \\
\psi_{i,f_i,1} & \cdots & \psi_{i,f_i,(D+1)}
\end{pmatrix},
\]

where \(\Psi_i \in \mathbb{R}^{f_i \times (D+1)}\). The feature mapped nodes are employed to explore the hidden features of the input data. The mapped features \(\mathbf{g}_i\) of the \(i\)-th group are obtained by projecting the input data with the matrix \(\Psi_i\). They are given by

\[
\mathbf{g}_i = [g_{i,1}, \cdots, g_{i,f_i}]^T = \Psi_i \mathbf{x}, \quad \forall \ i = 1, \cdots, n.
\]

where \(g_{i,u}\) is the \(u\)-th feature of the \(i\)-th group, \(i = 1, \cdots, n\), and \(u = 1, \cdots, f_i\).

The construction procedure of \(\Psi_i\)'s is established via sparse optimization step. This can be achieved in several ways, such as solving sparse optimization problem based on the alternating direction method of multipliers ADMM [64] algorithm. The construction procedure of \(\Psi_i\)'s is presented in the next subsection III-B1. It should be noticed that a nonlinear operation on \(\mathbf{g}_i\) can be used. In the classical BLS framework, no nonlinear operation is applied on \(\mathbf{g}_i\). This paper follows the classical BLS framework and does not apply nonlinear operation on \(\mathbf{g}_i\).

All the outputs from the feature mapped nodes are collected together as

\[
\mathbf{g} = [\mathbf{g}_1^T, \cdots, \mathbf{g}_n^T]^T \in \mathbb{R}^l.
\]

The augmented vector \(\mathbf{q}\) of \(\mathbf{g}\) is denoted as

\[
\mathbf{q} = [\mathbf{g}^T, 1]^T \in \mathbb{R}^{l+1}.
\]

2) ENHANCEMENT NODES
The BLS network has \(m\) groups of enhancement nodes. In the BLS framework, the \(j\)-th group of enhancement nodes has \(e_j\) nodes. The total number of enhancement nodes in the BLS network is given by

\[
e = \sum_{j=1}^{m} e_j.
\]

Besides, the output of \(j\)-th group of enhancement nodes is given by

\[
\mathbf{h}_j = [h_{j,1}, \cdots, h_{j,e_j}]^T = \xi(\mathbf{W}_j \mathbf{q}),
\]

where \(j = 1, \cdots, m\) and \(\mathbf{W}_j\) is a randomly generated. The elements of \(\mathbf{W}_j\) are denoted as

\[
\mathbf{W}_j = \begin{pmatrix}
w_{j,1,1} & \cdots & w_{j,1,f+1} \\
\vdots & \ddots & \vdots \\
w_{j,e_j,1} & \cdots & w_{j,e_j,f+1}
\end{pmatrix}.
\]

It should be noticed that \(\xi(\cdot)\) is the activation function for enhancement nodes. Generally, each group of enhancement nodes can have its own activation function. In the original BLS algorithm, hyperbolic tangent is employed as the activation function for all the enhancement nodes. This paper uses hyperbolic tangent as the activation function for all enhancement nodes. Let \(\eta\) be the collection of all the enhancement node outputs, given by

\[
\eta = [\mathbf{h}_1^T, \cdots, \mathbf{h}_m^T]^T \in \mathbb{R}^{e}.
\]

3) NETWORK OUTPUT
For a given input vector \(x\), the output of the network is

\[
o = [\mathbf{g}^T | \eta^T] \beta,
\]

where \(\beta\) is the output weight vector. The number of elements in \(\beta\) is equal to \(f + e\). Hence, its components are given by

\[
\beta = [\beta_1, \cdots, \beta_{f+e}]^T.
\]

B. CONSTRUCTION OF WEIGHT MATRICES AND VECTORS
Consider that there are \(N\) training pairs in the training set \(\mathcal{D}_{\text{train}} = \{(x_k, y_k) : k = 1, \cdots, N\}\), where \(x_k = [x_{k,1}, \cdots, x_{k,D}]^T\) is the \(k\)-th training input vector and \(y_k\) is the corresponding target output. First, the training data matrix is formed by packing all the input \(x_k\)'s together. The augmented data matrix denoted as \(\mathbf{X}\) is given by

\[
\mathbf{X} = \begin{pmatrix}
x_1^T & 1 \\
\vdots & \vdots \\
x_N^T & 1
\end{pmatrix}.
\]
1) CONSTRUCTION OF THE PROJECTION MATRIX: \( \Psi_i \)

For each group of feature mapped nodes, one vital thing is how to construct the projection matrix, \( \Psi_i \). The technique described here is based on the procedures in [13], [60], [65]. In the BLS, a random matrix \( P_i \in \mathbb{R}^{(D+1)\times f_i} \) is generated first for each group of feature mapped nodes. Afterwards, a random projection data matrix \( Q_i \) is obtained, given by

\[
Q_i = X P_i
\]

where in (13), the term \( \rho \| \Psi_i \|_1 \) is to enforce the solution of (13) to be a sparse matrix. Also, \( \rho \) is a regularization parameter for sparse regularization.

2) CONSTRUCTION OF THE WEIGHT MATRICES OF THE ENHANCEMENT NODES: \( W_j \)

The construction step of \( W_j \)'s is a bit simple. The traditional BLS algorithm and other variants randomly generate the weight matrices for each group of the enhancement nodes. The articles [13], [60], [65] give the details. This paper follows this random generation concept.

3) CONSTRUCTION OF OUTPUT WEIGHT VECTOR

This section gives the procedure to construct the output weight vector \( \beta \). Given the projection matrix \( \Psi_i \) for each group \( i \), the \( i \)-th training data feature matrix for all training samples is given by

\[
Z_i = \begin{bmatrix} z_{i,1}^T \\ \vdots \\ z_{i,N}^T \end{bmatrix} = X \Psi_i^T,
\]

where \( z_{i,k} = [z_{i,k,1}, \ldots, z_{i,k,f_i}]^T \), and

\[
z_{i,k,u} = \sum_{l=1}^{D} \psi_{i,u,l} x_{i,l} + \psi_{i,u,D+1}.
\]

Let \( Z \) be the collection of all training data feature matrices, given by

\[
Z = [Z_1, \ldots, Z_m]
\]

In this way, \( Z \) is an \( N \times f \) matrix, denoted as

\[
Z = \begin{bmatrix} Z_{1,1} & \ldots & Z_{1,f} \\ \vdots & \ddots & \vdots \\ Z_{N,1} & \ldots & Z_{N,f} \end{bmatrix} = \begin{bmatrix} Z_{1,1}^T \\ \vdots \\ Z_{N,1}^T \end{bmatrix},
\]

where the \( k \)-th row vector \( Z_{k,1}^T \) of \( Z \) contains the inputs of the enhancement nodes (the outputs of the feature mapped nodes) for the \( k \)-th training input vector \( x_k \). In order to handle input biases, an one vector is added into \( Z \), given by

\[
Z' = \begin{bmatrix} Z_{1,1}^T & 1 \\ \vdots & \vdots \\ Z_{N,1}^T & 1 \end{bmatrix} = \begin{bmatrix} Z_{1,1}^T \\ \vdots \\ Z_{N,1}^T \end{bmatrix}.
\]

Furthermore, given \( Z \), the enhancement node outputs of the \( j \)-th enhancement group for all training data are given by

\[
H_j = \xi (Z W_j^T) = \begin{bmatrix} h_{j,1}^T \\ \vdots \\ h_{j,N}^T \end{bmatrix},
\]

for \( j = 1, \ldots, m \), where

\[
h_{j,k} = [h_{j,k,1}, \ldots, h_{j,k,\tau}]^T,
\]

and

\[
h_{j,k,v} = \xi (\sum_{\tau=1}^{f+1} W_{j,\tau,v} Z_{k,\tau}').
\]

Packing all the enhancement node outputs together, a \( N \times (\sum_{j=1}^{m} \ell_j) = N \times e \) matrix \( \mathcal{H} \) is obtained:

\[
\mathcal{H} = [H_1, \ldots, H_m].
\]

Define \( \mathcal{A} = [Z, \mathcal{H}] \). The output weight vector \( \beta \) can be calculated based on least square techniques

\[
\arg \min_{\beta} : \| \mathcal{A} \beta - y \|_2^2 + q \| \beta \|_2^2,
\]

where \( y = [y_1, \ldots, y_N]^T \) is the collection of all training outputs, and \( q \) is a regularization parameter.

IV. MULTIPLICATIVE NOISE AND OPEN WEIGHT FAULT IN BLS

In the digital realization of a well trained neural network, the finite precision representation of the weights leads to multiplicative weight noise [3], [4]. Also, using analog implementation, the precision errors are usually specified based on the percentage of error. Hence, the weight deviation is proportional to the magnitude of the weight value.

The output weights \( \beta_i \)’s \( (i = 1, \ldots, f + e) \) can be decomposed into two sets. One set contains the output weights related to the feature mapped nodes. Another set contains the output weights related to the enhancement nodes.

For easy of reading, this paper introduces a duplication labelling for the output weights based on the feature mapped node set and the enhancement node set. The duplication labelling for \( \beta_i \)’s is as follows.

- The symbol \( \beta_{g,i,u} \) denotes the \( u \)-th output weight in the \( i \)-th group of feature mapped nodes, where \( i = 1, \ldots, n \) and \( u = l, \ldots, f_i \).
- The symbol \( \beta_{h,j,v} \) denotes the \( v \)-th output weight in the \( j \)-th group of feature mapped nodes, where \( j = 1, \ldots, m \) and \( v = l, \ldots, e_j \).
A. MULTIPlicative NOISE IN OUTPUT WEIGHTS \( \beta \)
This paper uses the following model to describe the behaviour of of multiplicative noise on output weights \( \beta_i 's \), given by

\[
\bar{\beta}_{g,i,u} = (1 + \gamma_{g,i,u}) \beta_{g,i,u},
\]
(24)

\[
\bar{\beta}_{h,j,v} = (1 + \gamma_{h,j,v}) \beta_{h,j,v},
\]
(25)

where \( \bar{\beta}_{g,i,u} \) and \( \bar{\beta}_{h,j,v} \) are implemented values of \( \beta_{g,i,u} \) and \( \beta_{h,j,v} \), respectively. The variables \( \gamma_{g,i,u} 's \) and \( \gamma_{h,j,v} 's \) are the noise factors that describe the percentage of deviation from the feature node output \( z_{i,k,u} \). This paper assumes that these noise factors are iid zero mean random variables with variance equal to \( \sigma^2 \).

B. OPEN WEIGHT FAULT IN OUTPUT WEIGHTS \( \beta \)
Due to hardware imperfection, open weight faults [2], [4], [6] may happen. For the BLS case, feature mapped nodes and enhancement nodes may be disconnected to the output layer. For open weight fault, an implemented output weight can be described as

\[
\bar{\beta}_{g,i,u} = \zeta_{g,i,u} \beta_{g,i,u}, \quad \text{and} \quad \bar{\beta}_{h,j,v} = \zeta_{h,j,v} \beta_{h,j,v},
\]
(26)

where \( \zeta_{g,i,u} 's \) and \( \zeta_{h,j,v} 's \) are open weight factors that describes whether the output weights are opened or not. If an output weight is opened, \( \zeta = 0 \). Otherwise, \( \zeta = 1 \). In this paper, the open weight factors are modelled as iid binary random variables. Hence, their probability mass functions are given by

\[
\text{Prob}(\zeta) = \begin{cases} 
 p_\beta, & \zeta = 0, \\
 1 - p_\beta, & \zeta = 1,
\end{cases}
\]
(27)

where \( p_\beta \) is the fault rate of the output weights.

C. CONCURRENT WEIGHTs IN OUTPUT WEIGHTs \( \beta \)
When concurrent weight failures (multiplicative weight noise and open weight fault) occur on the output weights, the output weights can be described as

\[
\bar{\beta}_{g,i,u} = (1 + \gamma_{g,i,u}) \zeta_{g,i,u} \beta_{g,i,u},
\]
(28)

\[
\bar{\beta}_{h,j,v} = (1 + \gamma_{h,j,v}) \zeta_{h,j,v} \beta_{h,j,v}.
\]
(29)

From (28) and (29), when an output weight is opened, its weight value is equal to 0. When the output weight is not opened, its weight value is affected by weight noise only.

D. NOISE IN FEATURE MAPPED NODEs
Feature nodes can be affected by multiplicative noise. For easy of reading, denote the current input vector is the \( k \)-th training vector \( x_k \). That means, the augmented input is \( x_k^T = [x_k^T, 1]^T \). Under the multiplicative noise, the feature node output is given by

\[
\bar{z}_{i,k,u} = (1 + \alpha_{i,u}) \zeta_{i,k,u} \\
= (1 + \alpha_{i,u}) \sum_{i=1}^{D} \psi_{i,u,i} x_{k,i} + \psi_{i,u,D+1}).
\]
(30)

where \( i = 1, \ldots, n, u = 1, \ldots, f_j \) and \( i = 1, \ldots, D + 1 \). In (30), \( \alpha_{i,u} 's \) are noise factors that describe the percentage of deviation from the feature node output \( z_{i,k,u} \). This paper assumes that these noise factors are iid zero mean random variables with variance equal to \( \sigma^2 \). From (30), the noisy version of the training data feature matrices are denoted as \( \mathbf{Z} = [\bar{Z}_1, \ldots, \bar{Z}_n] \).

E. NOISE IN ENHANCEMENT NODEs
Also, enhancement nodes can be affected by multiplicative noise. This paper considers that the input weights of the enhancement nodes have precision errors under implementation. Hence, under the multiplicative noise model, the affected input weight, \( \tilde{w}_{j,v} \), is given by

\[
\tilde{w}_{j,v} = (1 + \delta_{j,v}) w_{j,v},
\]
(31)

where \( j = 1, \ldots, m, v = 1, \ldots, e_j \), and \( f + 1 \). \( \delta_{j,v} 's \) are noise factors that describe the percentage of deviation in input weights. Given the current input is \( x_k \), with the aids of Taylor series, the output \( h_{j,k,v} \) of an enhancement node can be modelled as

\[
\tilde{h}_{j,k,v} = h_{j,k,v} + \sum_{\tau=1}^{f+1} \delta_{j,v,\tau} w_{j,v,\tau} \frac{\partial h_{j,k,v}}{\partial w_{j,v,\tau}}.
\]
(32)

This paper assumes that \( \delta_{j,v,\tau} 's \) are iid zero mean random variables with variance equal to \( \sigma^2 \).

It should be noticed from (7) and (16),

\[
h_{j,k,v} = \xi \left( \sum_{\tau=1}^{f+1} w_{j,v,\tau} z_{k,\tau}' \right).
\]
(33)

When the hyperbolic tangent is used as the activation,

\[
\frac{\partial h_{j,k,v}}{\partial w_{j,v,\tau}} = z_{k,\tau}' (1 - h^2_{j,k,v})
\]
(34)

Hence, from (19), (32), and (34), the noise version of enhancement node outputs for all training inputs are denoted as \( \mathbf{H} = [\tilde{H}_1, \ldots, \tilde{H}_m] \).

F. NOISE STATISTICS OF THE WEIGHTED INPUTS TO THE OUTPUT NODES
This section presents the statistical properties of the weighted inputs to the output node, given by \( \tilde{z}_{i,k,u} = \bar{\beta}_{g,i,u} \bar{z}_{i,k,u} \) and \( h_{j,k,v} = \bar{\beta}_{h,j,v} h_{j,k,v} \). From the statistical properties of \( \gamma_{g,i,u} 's, \zeta_{g,i,u} 's, \) and \( \alpha_{i,u} 's \), the statistics of \( \tilde{z}_{i,k,u} 's \) are given by

\[
\langle \tilde{z}_{i,k,u} \rangle = (1 - p_\beta) \langle z_{i,k,u} \rangle \beta_{g,i,u}
\]
(35)

\[
\langle \tilde{z}^2_{i,k,u} \rangle = (1 - p_\beta) (1 + \sigma^2) (1 + \sigma^2)
\]
(36)

where \( \langle \cdot \rangle \) is the expectation operation. It should be noticed that \( \tilde{z}_{i,k,u} = \bar{\beta}_{g,i,u} z_{i,k,u} \) and \( \tilde{z}_{j,k,v} = \bar{\beta}_{h,j,v} h_{j,k,v} \) are uncorrelated for \( i' \neq i \) or \( u' \neq u \).
In addition, the statistics of \( \tilde{h}_{j,k,v} \tilde{p}_{h_{j,v},v} \) are given by
\[
\left( \tilde{h}_{j,k,v} \tilde{p}_{h_{j,v},v} \right) = (1 - p_p) \tilde{h}_{j,k,v} \tilde{p}_{h_{j,v},v} \tag{37}
\]
\[
\left( \tilde{h}_{j,k,v}^2 \tilde{p}_{h_{j,v},v}^2 \right) = (1 - p_p)(1 + \sigma^2_p) \tilde{p}_{h_{j,v},v}^2 \left( \tilde{h}_{j,k,v}^2 + \sigma^2(1 - \tilde{h}_{j,k,v}^2) \sum_{r=1}^{f+1} w_{j,v,r}^2 Z_{k,r}^2 \right) \tag{38}
\]
It should be noticed that \( \tilde{h}_{j,k,v} \) and \( \tilde{h}_{j,k,v}^2 \) are uncorrelated for \( j \neq j' \) and \( v \neq v' \). Also, \( (\tilde{z}_{i,k,u} \tilde{p}_{g_{i},u}) \) and \( (\tilde{h}_{j,k,v} \tilde{h}_{j,v}) \) are uncorrelated.

\section{THE PROPOSED FAULT-AWARE OBJECTIVE FUNCTION AND LEARNING ALGORITHM}

In the noise/fault situation, this paper proposes to minimize the expected error over all possible noise/fault patterns. First, the training set error for a particular noise/fault pattern is considered. With the fault model (24)-(26), and (28)-(32), the training set error for a particular noise/fault pattern is given by
\[
\bar{e} = \frac{1}{N} \sum_{k=1}^{N} \left( \sum_{i=1}^{m} \sum_{u=1}^{n} \tilde{z}_{i,k,u} \tilde{p}_{g_{i},u} + \sum_{j=1}^{f+1} \tilde{h}_{j,k,v} \tilde{p}_{h_{j,v},v} - y_k \right)^2 \tag{39}
\]
where \( \tilde{A} = [\tilde{Z} \tilde{H}] \).

With (35)-(38) and some manipulations, the average training set error of noisy networks over all possible noise/fault is the given by
\[
J = \langle \bar{e} \rangle = \frac{1}{N} \left\| \tilde{A} \beta - y \right\|_2^2 \tag{40}
\]
where \( \text{diag}(\cdot) \) is to return a diagonal matrix from the main diagonal of a square matrix. Note that since the squared error stated in (39) is a convex function of the output weight vector \( \beta \), the average squared error stated in (40) is a convex function too.

The matrix \( S \) is a block diagonal matrix, given by
\[
S = \begin{bmatrix} S_1 & \emptyset \\ \emptyset & S_2 \end{bmatrix} \tag{41}
\]
\[
S_1 = \sigma^2_{\xi}(Z)Z \tag{42}
\]
Define \( R \) as
\[
R = (p_p + \sigma^2_p) \text{diag}(A^T A) + (1 + \sigma^2_p) \text{diag}(S) - p_p(A^T A). \tag{43}
\]
With (46), the average training set error, stated in (40), can be rewritten as
\[
J = \frac{1}{N} p_p \| A \beta - y \|^2 + \frac{1}{N} p_p \beta^T R \beta + \frac{p_p}{N} \| y \|^2. \tag{44}
\]
Since the last term in (47) is independent of the output weight vector, the objective function can be rewritten as
\[
J = \| A \beta - y \|^2 + \beta^T R \beta. \tag{45}
\]
In (48), it can be seen that the term \( \beta^T R \beta \) can be considered as regularization term for handling the noise/fault in the BLS network.

Since (40) is a convex function of \( \beta \), (48) is a convex function too. Hence, the optimal solution can be obtained by setting the gradient of (48) to zero. The proposed FABLS solution is
\[
\beta^* = (R + A^T A)^{-1} A^T y. \tag{49}
\]
The FABLS algorithm is summarized in Algorithm 1.

Remark: One advantage of the proposed FABLS algorithm is that there is no tuning parameter. In the original BLS (see (23)) and other variants, such as WBLS [19] and CBLS [20], there is a tuning parameter \( \varrho \). The good values of \( \varrho \) are chosen by trial and error.

\section{EXPERIMENTS}

\subsection{EXPERIMENTAL SETTINGS AND DATASETS}

In this section, the proposed algorithm FABLS is compared with the original BLS algorithm, WBLS [19] and CBLS [20]. The WBLS algorithm was designed to handle both noisy data and outliers. Similarly, CBLS is a robust approach based on correntropy criterion. The CBLS algorithm has an anti-noise agent in its objective function. Hence, these comparison algorithms, original BLS, WBLS, and CBLS are best suitable for comparison with the proposed FABLS under network failure situations.

In the comparison, several real life datasets from the UCI machine learning repository [21] are used. They are CTslice,
Algorithm 1: The Training Process of FABLS Algorithm

Require: Training data matrix $X$ according to (11) and the corresponding target $y$, number of groups of feature nodes $n$, number of groups of enhancement nodes $m$.

Construct the projection matrices $\Psi_j$'s based on (13).

Output: Output weight $\beta^*$

1: Compute $Z_i$ using (13)-(14), where $i = 1, \cdots, n$.
2: Set $Z = [Z_1, \cdots, Z_n]$.
3: Compute $H_j$ using (18) and (19), where $j = 1, \cdots, m$.
4: Set $\mathcal{H} = [H_1, \cdots, H_m]$. 
5: Compute $S$ and $R$ using (41) and (46) respectively.
6: Compute $\beta^*$ according to (49).

TABLE 2. Details of the datasets.

| Dataset                  | Number of samples | Number of features |
|--------------------------|-------------------|-------------------|
| CTslice                  | 53500             | 385               |
| Airfoil Self Noise(ASN)  | 1530              | 7                 |
| Building Energy          | 4208              | 14                |
| Abalone                  | 4177              | 8                 |
| CCPP                     | 9566              | 4                 |
| Coil2000                 | 5822              | 85                |

TABLE 3. The data division over the 5 fold.

| Datasets        | Subset 1 | Subset 2 | Subset 3 | Subset 4 | Subset 5 |
|-----------------|----------|----------|----------|----------|----------|
| CTslice         | 10700    | 10700    | 10700    | 10700    | 10700    |
| ASN             | 301      | 301      | 301      | 301      | 299      |
| Building Energy | 842      | 842      | 842      | 842      | 840      |
| Abalone         | 835      | 835      | 835      | 835      | 837      |
| CCPP            | 1914     | 1914     | 1914     | 1914     | 1912     |
| Coil2000        | 1164     | 1164     | 1164     | 1164     | 1166     |

TABLE 4. Various tested $\varrho$ values for BLS, CBLS and WBLS algorithms.

| Dataset          | Parameter               |
|------------------|-------------------------|
| CTslice          | $\varrho \in \{0.001, 0.005, 0.007, 0.035, 0.5, 0.6, 0.9\}$ |
| ASN              | $\varrho \in \{0.001, 0.02, 0.05, 0.07, 0.09, 0.1, 0.2\}$ |
| Building Energy  | $\varrho \in \{0.002, 0.03, 0.06, 0.08, 0.25, 0.4, 0.6\}$ |
| Abalone          | $\varrho \in \{0.000, 0.002, 0.004, 0.006, 0.008, 0.2, 0.4, 0.6\}$ |
| CCPP             | $\varrho \in \{0.002, 0.006, 0.009, 0.2, 0.4, 0.6\}$ |
| Coil2000         | $\varrho \in \{0.001, 0.005, 0.008, 0.1, 0.2, 0.3, 0.6\}$ |

Airfoil Self Noise(ASN), Building Energy, Abalone, CCPP, and Coil2000. Table 2 shows the details of the datasets. The 5-fold evaluation method is used in our experiments. Table 3 provides the details of the five folds.

In this paper, the input features of the datasets are normalized to $[-1, 1]$. Also, the target outputs are normalized to [0, 1]. In the original BLS frame-work, the input weights and the biases of the enhancement nodes are generated randomly and then fixed afterward. Based on the original BLS concept, the values of the generated weights and biases are between the range $[-1, 1]$.

The WBLS algorithm has a weighted penalty factor which enhances its performance. In this work, the weighted penalty factor in WBLS is constructed by the Huber weight function with positive adjustable Huber parameter given by $b = 0.6$.

In the BLS concept, the feature nodes of a BLS network uses linear activation function to process the input data. The problem with this approach is that the output values of feature nodes could be very large. This behaviour can affect the performance of the original BLS algorithm, if the output value is forwarded to enhance node directly, without any additional process. In the traditional BLS, the commonly used solution is that after getting the output values of feature nodes, the values are mapped or normalized to range [0, 1].

The original BLS algorithm, WBLS algorithm and CBLS algorithm have a regularizer term $\varrho$ that can mitigate the effect of fault in the network. In the original BLS algorithm and other variants, there is a tuning regularization parameter $\varrho$. The value of $\varrho$ affects the performance of resultant BLS networks. There is no analytical way of selecting the suitable value of $\varrho$. In this experiment, various $\varrho$ values are tested for each dataset, and then the best $\varrho$ value is found based on the test set. Table 4 summarizes the various tested $\varrho$ values.

The original BLS algorithm, WBLS algorithm and CBLS algorithm have a regularizer term $\varrho$ that can mitigate the effect of fault in the network. In the original BLS algorithm and other variants, there is a tuning regularization parameter $\varrho$. The value of $\varrho$ affects the performance of resultant BLS networks. There is no analytical way of selecting the suitable value of $\varrho$. In this experiment, various $\varrho$ values are tested for each dataset, and then the best $\varrho$ value is found based on the test set. Table 4 summarizes the various tested $\varrho$ values.

The original BLS algorithm, WBLS algorithm and CBLS algorithm have a regularizer term $\varrho$ that can mitigate the effect of fault in the network. In the original BLS algorithm and other variants, there is a tuning regularization parameter $\varrho$. The value of $\varrho$ affects the performance of resultant BLS networks. There is no analytical way of selecting the suitable value of $\varrho$. In this experiment, various $\varrho$ values are tested for each dataset, and then the best $\varrho$ value is found based on the test set. Table 4 summarizes the various tested $\varrho$ values.

In the BLS concept, the feature nodes of a BLS network uses linear activation function to process the input data. The problem with this approach is that the output values of feature nodes could be very large. This behaviour can affect the performance of the original BLS algorithm, if the output value is forwarded to enhance node directly, without any additional process. In the traditional BLS, the commonly used solution is that after getting the output values of feature nodes, the values are mapped or normalized to range [0, 1].

The normalized values are further processed by enhancement node [66]. This approach can improve the performance of the original BLS algorithm. In this paper, for fair comparison, this normalization concept is used for the original BLS algorithm, WBLS algorithm, CBLS algorithm and the proposed FABLS algorithm.
because when noise levels are small, the advantage of our algorithms are similar. It is 

algorithms are listed in tabular form in Table 5.

Besides, the numerical values of the average test set MSE 

ation. Each figure shows the results for an open fault rate.

the corresponding standard deviation over the 5 fold evalu-

summarize the average test set MSE of noisy networks and 
genenerated. For an easy and quick view of the result, Figs. 2–4 

For each dataset and each fold, 2000 faulty networks are 

algorithms in term of average test set MSE of noisy networks.

This subsection presents the performance of the comparison 

ture mapped nodes. For the enhancement nodes, 
m = 1 and d = 100. Hence, there are 100 enhancement nodes. With the setting, 

there are 200 hidden nodes in the BLS network.

B. PERFORMANCE COMPARISON

This subsection presents the performance of the comparison 

algorithms in term of average test set MSE of noisy networks.

The 5-fold evaluation strategy is used in the experiments. 

For each dataset and each fold, 2000 faulty networks are 
genenerated. For an easy and quick view of the result, Figs. 2–4 summarize the average test set MSE of noisy networks and the corresponding standard deviation over the 5 fold evaluation. Each figure shows the results for an open fault rate. Besides, the numerical values of the average test set MSE and the corresponding standard deviation of the comparison algorithms are listed in tabular form in Table 5.

From Figs. 2–4, when the noise levels are small, the average test MSE values of the four algorithms are similar. It is because when noise levels are small, the advantage of our proposed FABLS is not fully demonstrated. However, the average test set MSE values of the proposed FABLS algorithm are still small those of the original BLS algorithm, WBLS algorithm, and CBLS algorithm. The improvements of the proposed FABLS are more significant when the noise levels is large.

For example, for the ASN dataset, at low noise level of \(p_{\beta} = 0.01\), \(\sigma_{w}^{2} = \sigma_{z}^{2} = \sigma_{a}^{2} = 0.01\), the average test set MSE of the original BLS is 0.18626, the average test set MSE of the WBLS is 0.018159, and the average test set MSE of the CBLS is 0.018603. For the proposed FABLS algorithm, it has an average test set MSE of 0.016880 only. When the open weight fault level is fixed at \(p_{\beta} = 0.01\) and the multiplicative noise level is changed to \(\sigma_{w}^{2} = \sigma_{z}^{2} = \sigma_{a}^{2} = 0.25\), the average test set MSE of the original BLS jumps to 0.082964. Also, the average test set MSE of the WBLS jumps to 0.082206. However, the average test set MSE of the CBLS jumps to 0.067633 and the average test set MSE of the proposed FABLS is not fully demonstrated. However, the average test set MSE values of the proposed FABLS algorithm are still small those of the original BLS algorithm, WBLS algorithm, and CBLS algorithm. The improvements of the proposed FABLS are more significant when the noise levels is large.
TABLE 5. Average test set MSE and Standard Deviation of BLS, WBLS, CBLS and FABLS. There are 200 nodes in the network. The 5-fold evaluation strategy is used. For each dataset and each fold, 2,000 faulty networks are generated.

| Dataset      | Noise Level | \( p_\beta \) | \( \sigma_\beta^2 = \sigma_z^2 = \sigma_w^2 \) | BLS AVG MSE | SD | WBLS AVG MSE | SD | CBLS AVG MSE | SD | FABLS AVG MSE | SD |
|--------------|-------------|--------------|-----------------------------------------------|-------------|----|-------------|----|-------------|----|-------------|----|
| Building Energy | 0.01        | 0.01         | 0.028401                                      | 0.002164    |    | 0.027930    | 0.002157 | 0.028390 | 0.002172   | 0.027001 | 0.002025 |
|              | 0.05        | 0.01         | 0.048481                                      | 0.003670    |    | 0.048670    | 0.003680 | 0.048670 | 0.003697   | 0.029672 | 0.003665 |
|              | 0.25        | 0.01         | 0.091911                                      | 0.005856    |    | 0.077659    | 0.005659 | 0.091378 | 0.005825   | 0.039499 | 0.001451 |
|              | 0.05        | 0.01         | 0.038549                                      | 0.001798    |    | 0.027971    | 0.001739 | 0.028232 | 0.001802   | 0.027301 | 0.001836 |
|              | 0.05        | 0.09        | 0.048558                                      | 0.002841    |    | 0.043649    | 0.002742 | 0.043686 | 0.002831   | 0.030066 | 0.001645 |
|              | 0.05        | 0.25        | 0.091782                                      | 0.005863    |    | 0.077470    | 0.004946 | 0.091237 | 0.005649   | 0.034589 | 0.002847 |
|              | 0.10        | 0.01         | 0.059195                                      | 0.001562    |    | 0.039604    | 0.001449 | 0.059091 | 0.001534   | 0.051222 | 0.000737 |
|              | 0.10        | 0.09        | 0.059170                                      | 0.002266    |    | 0.035386    | 0.002049 | 0.059026 | 0.002248   | 0.034676 | 0.000594 |
|              | 0.10        | 0.25        | 0.098315                                      | 0.004648    |    | 0.084508    | 0.003404 | 0.097853 | 0.004610   | 0.039117 | 0.000532 |

average test set MSE of the WBLS algorithm is 0.147297 and the average test set for CBLS algorithm is 0.156059. On the other hand, for FABLS, the test set MSE is 0.023772 only. When \( p_\beta \) is fixed to 0.01 and \([\sigma_\beta^2 = \sigma_z^2 = \sigma_w^2]\) is increased to 0.25, the original BLS algorithm has a very large average test set MSE, which is equal to 4.246942. In addition, the average test set MSE of WBLS is 4.018248 and the average test set MSE of CBLS algorithms is 4.245743. However, the average test set MSE of our proposed FABLS algorithm is 0.069069 only. For other datasets, similar results are obtained. Hence, our proposed FABLS algorithm outperforms the original BLS algorithm, the WBLS algorithm and the CBLS algorithm under the concurrent noise situation.

C. PAIRED T-TEST

As explained earlier in Subsection VI-B, from Table 5, in terms of average test set MSE, the proposed FABLS algorithm are better than the comparison algorithms. However, there is still a key question: if the improvements of using our FABLS are statistically significant. Hence, the paired t-test analysis is performed. It a standard method to investigate whether the mean difference between two sets of samples
are statistically significant or not. For the two tailed test with 5 trials and 95% confident level, the critical t-value is 2.132.

1) PAIRED T-TEST BETWEEN FABLS AND BLS

Table 6 summarizes the paired t-test result between the proposed FABLS algorithm and the original BLS algorithm. From the table, all the t-values are greater than the critical t-value, i.e., 2.132. Hence there is a strong evidence to conclude that the improvements are statistically significant. For instance, in the Coil2000 dataset with noise/fault level equal to \( p_{\beta} = 0.1, \sigma_\beta^2 = \sigma_\epsilon^2 = \sigma_{\epsilon_{\beta}}^2 = 0.09 \), the t-value is 13.12, which is greater than 2.132. Furthermore, the average MSE improvement of using FABLS is 0.053712 and the corresponding confident interval of average improvement is \([0.042344 - 0.065081]\). In the table, similar situations are found in other noise levels and datasets. With these results, there is a strong evidence that the FABLS is better than BLS. In fact, from the t-values at the large noise levels, the confidence

| Dataset | Noise Level | AVG MSE Improvement | t-value | Confident interval of improvement |
|---------|-------------|---------------------|---------|----------------------------------|
| Noise | \(p_{\beta}\) | \(\sigma_\beta^2\) | \(\sigma_\epsilon^2\) | \(\sigma_{\epsilon_{\beta}}^2\) |
| Building Energy | 0.01 | 0.00140 | 7.32 | (0.00056 - 0.00193) |
| | 0.05 | 0.00579 | 24.77 | (0.01565 - 0.02129) |
| | 0.10 | 0.00971 | 16.97 | (0.00112 - 0.02110) |
| Energy CCP | 0.01 | 0.00093 | 16.49 | (0.00028 - 0.00040) |
| | 0.05 | 0.00474 | 44.38 | (0.00039 - 0.00349) |
| | 0.10 | 0.00956 | 46.42 | (0.00093 - 0.01556) |
| CTSlice | 0.01 | 0.13233 | 75.59 | (0.12479 - 0.13719) |
| | 0.05 | 0.17577 | 158.39 | (0.14622 - 2.4112) |
| | 0.10 | 0.18927 | 33.59 | (0.08447 - 0.09952) |
| Abalone | 0.01 | 0.00800 | 13.53 | (0.00757 - 0.01419) |
| | 0.05 | 0.00064 | 20.02 | (0.00563 - 0.03779) |
| | 0.10 | 0.00101 | 37.67 | (0.00133 - 0.01527) |
| Coil2000 | 0.01 | 0.00655 | 7.36 | (0.00420 - 0.00889) |
| | 0.05 | 0.03537 | 13.12 | (0.04234 - 0.06508) |
| | 0.10 | 0.01388 | 10.28 | (0.03013 - 0.07079) |
| ASN | 0.01 | 0.00174 | 7.39 | (0.00109 - 0.00214) |
| | 0.05 | 0.00204 | 12.19 | (0.01609 - 0.02558) |
| | 0.10 | 0.00209 | 12.41 | (0.01624 - 0.02561) |
| 0.01 | 0.00416 | 3.14 | (0.00044 - 0.00788) |
| 0.05 | 0.00289 | 14.31 | (0.00016 - 0.00721) |
| 0.10 | 0.0092 | 14.73 | (0.00516 - 0.07567) |

| Dataset | Noise Level | AVG MSE Improvement | t-value | Confident interval of improvement |
|---------|-------------|---------------------|---------|----------------------------------|
| Noise | \(p_{\beta}\) | \(\sigma_\beta^2\) | \(\sigma_\epsilon^2\) | \(\sigma_{\epsilon_{\beta}}^2\) |
| Building Energy | 0.01 | 0.00092 | 5.27 | (0.00044 - 0.00141) |
| | 0.05 | 0.01401 | 16.60 | (0.01166 - 0.01955) |
| | 0.10 | 0.04371 | 19.58 | (0.03751 - 0.04990) |
| Energy CCP | 0.01 | 0.00666 | 3.21 | (0.00009 - 0.00124) |
| | 0.05 | 0.01358 | 16.42 | (0.01024 - 0.01888) |
| | 0.10 | 0.04258 | 19.35 | (0.03672 - 0.04903) |
| CTSlice | 0.01 | 0.00838 | 17.37 | (0.00704 - 0.00792) |
| | 0.05 | 0.01919 | 21.19 | (0.01667 - 0.02170) |
| | 0.10 | 0.05439 | 22.20 | (0.03971 - 0.05106) |
| Abalone | 0.01 | 0.00025 | 14.00 | (0.00023 - 0.00032) |
| | 0.05 | 0.00245 | 41.18 | (0.00228 - 0.00261) |
| | 0.10 | 0.00996 | 40.92 | (0.00919 - 0.01055) |
| Coil2000 | 0.01 | 0.12352 | 80.75 | (0.11297 - 0.12771) |
| | 0.05 | 1.29125 | 173.18 | (1.27054 - 1.31195) |
| | 0.10 | 0.36318 | 181.25 | (0.34897 - 0.37741) |
| ASN | 0.01 | 0.00153 | 7.97 | (0.00310 - 0.00197) |
| | 0.05 | 0.00263 | 15.53 | (0.00216 - 0.00310) |
| | 0.10 | 0.00553 | 23.53 | (0.00505 - 0.00664) |
| 0.01 | 0.00993 | 36.35 | (0.00834 - 0.00972) |
| 0.05 | 0.00986 | 40.92 | (0.00919 - 0.01055) |
| 0.10 | 0.02159 | 44.63 | (0.01811 - 0.01337) |
TABLE 8. The paired t-test result of FABLS and CBLS. There are 200 nodes in the network. The 5-fold evaluation strategy is used. For each dataset and each fold, 2,000 faulty networks are generated.

| Dataset | Noise Level | AVG MSE Improvement | t-value | Confident interval of improvement |
|---------|-------------|---------------------|---------|----------------------------------|
| Building Energy | 0.01 | 0.01 | 0.00138 | 7.30 | [0.00086-0.0191] |
| | 0.05 | 0.01 | 0.01122 | 5.84 | [0.00040-0.0180] |
| | 0.10 | 0.01 | 0.01830 | 20.97 | [0.01588-0.02072] |
| | 0.25 | 20.97 | 24.69 | [0.05907-0.06388] |
| | 0.10 | 0.01 | 0.02435 | 24.14 | [0.02155-0.02715] |
| | 0.25 | 24.14 | 26.91 | [0.05267-0.06479] |
| CCPP | 0.01 | 0.01 | 0.00034 | 16.53 | [0.00288-0.00840] |
| | 0.05 | 0.01 | 0.00377 | 21.49 | [0.00328-0.00425] |
| | 0.10 | 0.01 | 0.00919 | 33.95 | [0.00844-0.00995] |
| | 0.25 | 33.95 | 37.90 | [0.00973-0.01127] |
| CTSlice | 0.01 | 0.01 | 0.13238 | 75.7 | [0.12743-0.13174] |
| | 0.05 | 0.01 | 0.13593 | 24.84 | [0.13795-0.13908] |
| | 0.10 | 0.01 | 0.13472 | 26.56 | [0.13781-0.13828] |
| | 0.25 | 26.56 | 27.79 | [0.07180-0.08789] |
| Abalone | 0.01 | 0.01 | 0.00056 | 3.12 | [0.00008-0.00152] |
| | 0.05 | 0.01 | 0.00098 | 3.00 | [0.00007-0.00189] |
| | 0.10 | 0.01 | 0.00281 | 12.53 | [0.00185-0.00317] |
| | 0.25 | 12.53 | 19.22 | [0.02294-0.03009] |
| Coil2000 | 0.01 | 0.01 | 0.00536 | 7.12 | [0.00343-0.00729] |
| | 0.05 | 0.01 | 0.00457 | 7.83 | [0.00353-0.00563] |
| | 0.10 | 0.01 | 0.01370 | 13.17 | [0.01412-0.15657] |
| | 0.25 | 13.17 | 13.80 | [0.01412-0.15657] |
| ASN | 0.01 | 0.01 | 0.00172 | 7.37 | [0.00107-0.00237] |
| | 0.05 | 0.01 | 0.00196 | 5.87 | [0.00100-0.00292] |
| | 0.10 | 0.01 | 0.00266 | 12.38 | [0.01602-0.02529] |
| | 0.25 | 12.38 | 12.57 | [0.04739-0.07458] |
| | 0.10 | 0.01 | 0.02261 | 14.93 | [0.01841-0.02682] |
| | 0.25 | 14.93 | 14.70 | [0.05101-0.07477] |

of the improvements is extremely strong for high noise/fault levels.

2) PAIRED T-TEST BETWEEN FABLS AND WBLS

Table 7 presents the paired t-test results between the proposed FABLS algorithm and the WBLS algorithm in a tabular form. Clearly, it is seen that all the t-values are greater than the critical t-value, which is 2.132. In addition, all confident intervals of the average improvements of using FABLS do not include the zero. For instance, in the ASN dataset with noise level equal to \( p_\beta = 0.01, \sigma_\beta^2 = \sigma_c^2 = \sigma_w^2 = 0.09 \), the t-value is 11.50, which is greater than 2.132. Furthermore, the average improvement of using FABLS is 0.015640 and the corresponding confident interval of average improvement does not include zero. In the table, similar situations are found in other noise/fault levels and datasets. With these results, there is a strong evidence that the FABLS is better than WBLS.

3) PAIRED-T TEST BETWEEN FABLS AND CBLS

Similar to the other comparison algorithms, Table 8 summarizes the paired t-test results between the proposed FABLS algorithm and the CBLS algorithm.

As shown in Table 8, all confident intervals of the average improvements of using FABLS do not include the zero, and all the t-values are greater than the critical t-value. For instance, in the CTslice dataset with noise level equal to \( \{ p_\beta = 0.01, \sigma_\beta^2 = \sigma_c^2 = \sigma_w^2 = 0.09 \} \), the t-value is 173, which is much greater than 2.132. Furthermore, the average MSE improvement of using FABLS is 1.368225 and the corresponding confident interval of average improvement \([1.34635 – 1.39009]\). In the table, similar situations are found in other noise/fault levels and datasets. With these results, there is a strong evidence that the FABLS is better than CBLS.

VII. CONCLUSION

This paper first analyzed the effects of open weight fault and multiplicative noise in the BLS network. Four kinds of noise/fault are considered. They are open weight fault in output weights, multiplicative noise in output weights, multiplicative noise in feature nodes, and multiplicative noise in input weights of enhancement nodes. From the analyzed result, a fault aware objective function for BLS networks is then developed. The objective function contains a fault aware regularizer term which is capable of mitigating the effect of aforementioned fault/noise. Besides, there is no the regularizer parameter in our approach. On the other hand, in the traditional BLS, WBLS and CBLS, there is tuning regularization parameter.

Based on the proposed objective function, a fault-aware learning algorithm was developed to train a BLS network. The simulation experiments show that the proposed FABLS algorithm outperforms the original BLS algorithm, the WBLS algorithm and the CBLS algorithm. Also, the paired t-test results show that the improvements of the proposed FABLS algorithm are statistically significant.

REFERENCES

[1] C. A. Bertulani, “Fault tolerance in the brain,” Nature, vol. 30, p. 100, Apr. 1966.

[2] C.-S. Leung and J. P.-F. Sum, “A fault-tolerant regularizer for RBF networks,” IEEE Trans. Neural Netw., vol. 19, no. 3, pp. 493–507, Mar. 2008.

[3] C. S. Leung, H.-J. Wang, and J. Sum, “On the selection of weight decay parameter for faulty networks,” IEEE Trans. Neural Netw., vol. 21, no. 8, pp. 1232–1244, Aug. 2010.

[4] R.-B. Feng, Z.-F. Han, W. Y. Wan, and C.-S. Leung, “Properties and learning algorithms for faulty RBF networks with coexistence of weight and node failures,” Neurocomputing, vol. 224, pp. 166–176, Feb. 2017.
H. Yu, P. D. Reiner, T. Xie, T. Bartczak, and B. M. Wilamowski, “An incremental design of radial basis function networks,” IEEE Trans. Neural Netw. Learn. Syst., vol. 25, no. 10, pp. 1793–1803, Oct. 2014.

E. B. Tchernev, R. G. Mulvaney, and D. S. Phatak, “Investigating the fault tolerance of neural networks,” Neural Comput., vol. 17, no. 7, pp. 1646–1664, Jul. 2005.

T. Haruhiko, M. Masahiko, K. Hidehiko, and H. Terumine, “Enhancing both generalization and fault tolerance of multilayer neural networks,” in Proc. Int. Joint Conf. Neural Netw., Aug. 2007, pp. 1429–1433.

A. Ahmad, M. H. Sargolzaie, S. M. Fakhraie, C. Lucas, and S. Vakili, “A low-cost fault-tolerant approach for hardware implementation of artificial neural networks,” in Proc. Int. Conf. Comput. Eng. Technol., vol. 2, Jan. 2009, pp. 93–97.

X. Parra and A. Catala, “Fault tolerance in the learning algorithm of radial basis function networks,” in Proc. IEEE-INNS-ENNS Int. Joint Conf. Neural Netw. Neural Comput. New Challenges Perspect. New Millennium, vol. 3, Jul. 2000, pp. 527–532.

J. E. Moody, “Note on generalization, regularization and architecture selection in nonlinear learning systems,” in Proc. Neural Netw. Signal Process., 1991, pp. 1–10.

J. L. Bernier, J. Ortega, I. Rojas, E. Ros, and A. Prieto, “Obtaining fault tolerant multilayer perceptrons using an explicit regularization,” Neural Process. Lett., vol. 12, no. 2, pp. 107–113, 2000.

J. L. Bernier, J. Ortega, E. Ros, I. Rojas, and A. Prieto, “A quantitative study of fault tolerance, noise immunity, and generalization ability of MLPs,” Neural Comput., vol. 12, no. 12, pp. 2941–2964, Dec. 2000.

J. L. Bernier, J. Ortega, I. Rojas, and A. Prieto, “Improving the tolerance of multilayer perceptrons by minimizing the statistical sensitivity to weight deviations,” Neurocomputing, vol. 31, nos. 1–4, pp. 87–103, Mar. 2000.

J. Sum, W.-H. Luo, Y.-F. Huang, and Y.-T. Jhung, “Equivalence between weight decay learning and explicit regularization to improve fault tolerance of RBF,” in Proc. 8th Int. Conf. Intell. Syst. Design Appl., vol. 1, Nov. 2008, pp. 152–157.

J. L. Bernier, A. F. Diaz, F. Fernandez, A. Cañas, J. Gonzalez, P. Martin-Smith, and J. Ortega, “Assessing the noise immunity and generalization of radial basis function networks,” Neural Process. Lett., vol. 18, no. 1, pp. 35–48, 2003.

H. C. Leung, C. S. Leung, and E. W. M. Wong, “Fault and noise tolerance in the incremental extreme learning machine,” IEEE Access, vol. 7, pp. 155171–155183, 2019.

H.-C. Leung, C.-S. Leung, and E. W. Wong, “Fault-tolerant incremental learning for extreme learning machines,” in Neural Information Processing, A. Hirose, S. Ozawa, K. Doya, K. Ikeda, M. Lee, and D. Liu, Eds. Cham, Switzerland: Springer, 2016, pp. 168–176.

K. Y. Yung, “Node selection algorithm for fault-tolerant extreme learning machines,” Dept. Elect. Eng., City Univ. Hong Kong, Hong Kong, Final Year Project Rep., 2017.

H. T. Wong, C.-S. Leung, and S. Kwong, “A generalized I-ELM algorithm for handling node noise in single-hidden layer feedforward networks,” in Neural Information Processing, D. Liu, S. Xie, Y. Li, D. Zhao, and E.-S. M. El-Alfy, Eds. Cham, Switzerland: Springer, 2017, pp. 424–433.

K. Rzhazali, B. Lussier, W. Schön, and S. Géronimi, “Fault tolerant deep neural networks for detection of unrecognizable situations,” IFAC-PapersOnLine, vol. 51, no. 24, pp. 31–37, 2018.

H. Xu, Z. Chen, W. Wu, Z. Jin, S.-Y. Kuo, and M. Lyu, “NV-DNN: Towards fault-tolerant DNN systems with N-version programming,” in Proc. 49th Ann. IEEE/IFIP Int. Conf. Dependable Syst. Network, Workshops (DSN-W), Jun. 2019, pp. 44–47.

M. Snir et al., “Addressing failures in exascale computing,” Int. J. High Perform. Comput. Appl., vol. 28, no. 2, pp. 129–173, 2014.

L. Bautista-Gomez, F. Zyulkayrov, O. Unsal, and S. McIntosh-Smith, “Unprotected computing: A large-scale study of DRAM raw error rate on a supercomputer,” in Proc. Int. Conf. High Perform. Comput. Netw., Storage Anal. (SC), Nov. 2016, pp. 645–655.

L. Tan, S. L. Song, P. Wu, Z. Chen, R. Ge, and D. J. Kerbyson, “Investigating the interplay between energy efficiency and resilience in high performance computing,” in Proc. IEEE Int. Parallel Distrib. Process. Symp., May 2015, pp. 786–796.

K. Zhao, S. Di, S. Li, X. Liang, Y. Zhai, J. Chen, K. Ouyang, F. Cappello, and Z. Chen, “FT-CNN: Algorithm-based fault tolerance for convolutional neural networks,” 2020, arXiv:2003.12203. [Online]. Available: http://arxiv.org/abs/2003.12203.

A. C.-S. Leung, P. F. Sum, and K. Ho, “The effect of weight fault on associative networks,” Neural Comput., vol. 20, no. 1, pp. 113–121, Feb. 2011.

C. Torres-Huitzil and B. Girau, “Fault and error tolerance in neural networks: A review,” IEEE Access, vol. 5, pp. 17322–17341, 2017.
A. Avižienis, “Framework for a taxonomy of fault-tolerance attributes in computer systems,” in Proc. 10th Annua. Int. Symp. Comput. Archit., 1983, pp. 16–21.

V. P. Nelson, “Fault-tolerant computing: Fundamental concepts,” Computer, vol. 23, no. 7, pp. 19–25, Jul. 1990.

A. K. Somani and N. H. Vaidya, “Understanding fault tolerance and reliability,” Computer, vol. 30, no. 4, pp. 45–50, Apr. 1997.

V. Duddu, D. V. Rao, and V. E. Balas, “Adversarial fault tolerant training for deep neural networks,” 2019, arXiv:1907.03103. [Online]. Available: http://arxiv.org/abs/1907.03103

K. M. Prasad and R. B. Bapat, “The generalized Moore–Penrose inverse,” Linear Algebra Appl., vol. 165, pp. 59–69, Mar. 1992.

D. Carlson, E. Haynsworth, and T. Markham, “A generalization of the Schur complement by means of the Moore–Penrose inverse,” SIAM J. Appl. Math., vol. 26, no. 1, pp. 169–175, Jan. 1974.

K. Price, R. M. Storn, and J. A. Lampinen, Differential Evolution: A Practical Approach to Global Optimization. Berlin, Germany: Springer-Verlag, 2005.

J. Kennedy, “Particle swarm optimization,” in Encyclopedia of Machine Learning, C. Sammut and G. I. Webb, Eds. Boston, MA, USA: Springer, 2011, doi: 10.1007/978-0-387-30164-8_630.

F. Guely and P. Siarry, “Gradient descent method for optimizing various fuzzy rule bases,” in Proc. 2nd IEEE Int. Conf. Fuzzy Syst., Mar. 1993, pp. 1241–1246.

J.-W. Jin and C. L. P. Chen, “Regularized robust broad learning system for uncertain data modeling,” Neurocomputing, vol. 322, pp. 58–69, Dec. 2018.

M. Xu, M. Han, C. P. Chen, and T. Qiu, “Recurrent broad learning systems for time series prediction,” IEEE Trans. Cybern., vol. 50, no. 4, pp. 1405–1417, Sep. 2018.

S. Gao, G. Guo, H. Huang, X. Cheng, and C. L. P. Chen, “An end-to-end broad learning system for event-based object classification,” IEEE Access, vol. 8, pp. 45974–45984, 2020.

Y. Chu, H. Lin, L. Yang, D. Zhang, Y. Diao, X. Fan, and C. Shen, “Hyperspectral image classification based on discriminative locality preserving broad learning system,” Knowl.-Based Syst., vol. 206, Oct. 2020, Art. no. 106319.

S. Boyd, “Distributed optimization and statistical learning via the alternating direction method of multipliers,” Found. Trends Mach. Learn., vol. 3, no. 1, pp. 1–122, 2010.

J. Jin, Z. Liu, and C. L. P. Chen, “Discriminative graph regularized broad learning system for image recognition,” Sci. China Inf. Sci., vol. 61, no. 11, pp. 1–14, Nov. 2018.

C. P. Chen and Z. Liu. (2019). Broad Learning System. Accessed: Oct. 4, 2019. [Online]. Available: http://broadlearning.ai/bls/