A COMPACT OPTIMAL LEARNING MACHINE

Kanathip Sae-pae[1], Kuntpong Woraratpanya[2]

[1,2]Faculty of Information Technology
King Mongkut's Institute of Technology Ladkrabang
Chalongkrung Rd. Ladkrabang, Bangkok, 10520, Thailand

E-mail: kanathip.saepae@gmail.com[1], kuntpong@it.kmitl.ac.th[2]

DOI: https://doi.org/10.22452/mjcs.sp2019no2.8

ABSTRACT

Artificial neural networks (ANNs) have been developed and applied to a variety of problems, such as pattern recognition, clustering, function approximation, forecasting, optimization, etc. However, existing ANNs have a high computational cost, since their learning methods are mostly based on a parameter tuning approach. Extreme learning machine (ELM) is a state-of-the-art method that generally dramatically reduces the computational cost. An analysis of the ELM method reveals that there are unsolved key factors, including inefficient hidden node construction, redundant hidden nodes, and unstable results. Therefore, we describe a new learning machine based on analytical incremental learning (AIL) in conjunction with principal component analysis (PCA). This learning machine, PCA-AIL, inherited the advantages from the original one and solved the unsolved key factors of ELM, and also extended AIL capability to serve a multiple-output structure. PCA-AIL was implemented with a single-layer feed-forward neural network architecture, used an adaptive weight determination technique to achieve a compact optimal structure and also used objective relations to support multiple output regression tasks. PCA-AIL has two steps: objective relation estimation and multiple optimal hidden node constructions. In the first step, PCA estimated the objective relations from multiple-output residual errors. In the second step, the multiple optimal nodes were obtained from objective relations and added to the model. PCA-AIL was tested with 16 multiple-objective regression datasets. PCA-AIL mostly outperformed other methods (ELM, EM-ELM, CP-ELM, DP-ELM, PCA-ELM, EI-ELM) in terms of fast testing speed – 0.0017 second, a compact model – 19.9 nodes, an accurate performance – RMSE 0.11261, and a stable result – S.D. of RMSE 0.00911: reported in averaged.

Keywords: ***Extreme learning machine (ELM), Analytical incremental learning (AIL), Principal component analysis (PCA), Objective relation learning (ORL), Optimal hidden node determination, Multiple-output architecture, Multiple-output regression***

1.0 INTRODUCTION

Artificial neural networks (ANNs) are widely used algorithms for solving a variety of problems in pattern recognition, clustering, function approximation, forecasting, optimization, etc. [1]. Furthermore, their activation functions can efficiently map from an input space to a feature space. This capability can be applied to either simple or complex data patterns. The simple data patterns can be estimated with an essentially linear trend, but complex data patterns require more nonlinear functions to estimate. Therefore, datasets can be characterized as either simple or complex data patterns independently from its number of features.

However, conventional ANNs have a high computational cost, since their learning methods are mostly based on a parameter tuning, such as back-propagation (BP). In 2004, Huang et al. presented a breakthrough in learning methods - the "extreme learning machine" (ELM) [2], which is implemented in a single layer feed-forward neural network (SLFN) and does not tune parameters as it learns, thus dramatically reducing the computational cost. An ELM generates a fixed number of hidden nodes with random input layer weights and hidden layer biases and then computes all hidden layer weights at once by using an ordinary least square method. As a result, it is typically a thousand times faster than BP, but also computes more accurate prediction results. Nevertheless, ELM still suffers from a manual determination of optimal structure construction, redundant hidden nodes [3], and unstable results, since it requires a large number of randomly created hidden nodes to achieve a good generalization performance.

Since then, many improved ELM approaches have been designed. To the best of our knowledge, their major improvements can roughly be divided into three strategies for obtaining ELM parameters: (i) input layer weight randomization, (ii) input layer weight determination and (iii) adaptive weight determination. The first strategy attempts to determine the optimal structure of a hidden layer to provide the best performance with low resource consumption. Feng et al. introduced an Error Minimized ELM (EM-ELM) [4] as a learning method. Feng's learning method added hidden nodes to the model singly or in groups, where the group size can be varied. The hidden nodes were added continuously until stopping criteria, including the number of nodes or expected prediction performance, are satisfied. Thus, EM-ELM can obtain an optimal structure for an accurate prediction. Later, Wang et al. demonstrated that EM-ELM did not guarantee that the newly added hidden nodes will increase the performance, since their learning method did not have a generalization measure [5]. Furthermore, the EM-ELM structure was similar to the original one. This meant that EM-ELM still inherited problems from the original ELM, when high accuracy was required. Therefore, Wang et al. [5] described constructive and destructive parsimonious (CP and DP) ELMs, which incrementally determined the appropriate nodes in a given set of hidden nodes. CP-ELM selected hidden nodes, which led to significant error reduction, while DP-ELM removed hidden nodes, which did not significantly reduce errors. Both learning methods used QR decomposition of a recursive orthogonal least square algorithm for greatly reducing the size of operation matrices. Consequently, their memory consumption is very low, but their time complexity is very high. Although many enhanced ELMs that were improved by input layer weight randomization can lead to accurate predictions, their results were still unstable, because the hidden nodes were randomly generated. Furthermore, this concept requires many more hidden nodes. In the second strategy, ELMs were improved with an input layer weight determination technique. These ELMs did not randomize input layer weights; therefore, they provided stable results. A good example of an input layer weight determination learning method is Principal Component Analysis ELM (PCA-ELM) - Castaño et al. [6]. The PCA-ELM used 90% of the sum of PCA coefficients as the input layer weights. PCA coefficients were calculated from input samples. This method determined an optimal structure with stable results and obtained a specific number of hidden nodes, determined by the number of PCA coefficients. Even though PCA-ELM can ignore randomization and determine the appropriate number of hidden nodes automatically, the performance tends to be low on complex data patterns, for small input dimensions, since the number of hidden nodes is less than or equal to the number of input dimensions. Therefore, Castaño et al. [7] improved PCA-ELM, in conjunction with a Linear Discriminant Analysis method - LDA-PCA-ELM, which performed better than PCA-ELM on complex data patterns for classification tasks, but not for regression problems. LDA-PCA-ELM determined the optimal structure by using a combination of hidden nodes created from PCA and LDA coefficients. In the last strategy, adaptive weight determination was used as a key factor for calculating the optimal parameters. All optimal parameters were retrieved from the learning situation. A good example of this concept can be found in the incremental ELM (I-ELM) of Huang et al [8]. I-ELM determined the optimal structure, by singly adding a randomly hidden node, and then computing the hidden layer weights. The hidden layer weights, for the added hidden nodes, were optimized based on residual error until the target norm of residual error was reached. However, this approach still had limitations in redundant hidden nodes and un-retrained hidden layer weights as reported by Huang and Chan [9], [10]. These limitations tended to increase unnecessary complexity and slow convergence: they further improved I-ELMs [9]–[11] to improve performance. However, the hidden layer weights for both I-ELM and the improved I-ELMs were calculated with a non-least-square method; therefore, they do not reach optimal solutions [12].

To overcome these limitations, Alfarozi et al. designed an Analytical Incremental Learning (AIL), with adaptive weight determination to overcome the problems of either conventional or improved ELM approaches, by gradually increasing the number of hidden nodes and calculating parameters deterministically [13]. AIL avoided redundant hidden nodes, fully automatically determined an optimal structure, guaranteed stable performance in each iteration, and achieved good generalization performance with a small number of hidden nodes. Moreover, only one regularized hyperparameter needed to be tuned. The advantages of AIL mainly came from the construction of hidden nodes, such that the input layer weights and hidden layer biases were calculated by means of ridge regression [14] on the residual error of the previous iteration. However, AIL still needs an important improvement for multiple-output regression tasks. AIL weaknesses mainly came from the lack of objective relation determination, that can represent the model information, such as relationships among the outputs and residual error to create the optimal hidden nodes.

To find an optimal hidden node solution and to solve a multiple-output architecture problem of AIL, we propose a Principal Component Analysis AIL (PCA-AIL) with an objective relation estimation, based on PCA. The objective relation is the key to the increased prediction performance of AIL for the multiple-output regression. In contrast to

PCA-ELM [6], which applied PCA to the input data, our algorithm uses PCA to estimate objective relations from the residual error matrix, representing relative objectives and then uses them to obtain the optimal hidden nodes in each iteration. Furthermore, PCA provides significant objective relations to obtain multiple hidden nodes, based on added hidden nodes. This also leads to an increment of the AIL convergence rate. As a result, PCA-AIL has better prediction compared to most of the baseline methods in even low or high input dimensions. Furthermore, our method has also a compact structure.

This paper is organized as follows. Section 2 provides backgrounds of ELM and AIL and analyzes their limitations. Our algorithm is described in Section 3. Section 4 discusses the experimental results. Finally, we conclude in Section 5.

Table 1: Summary of reviewed learning methods

Strategy	Method	Objective relation learning	Automatic structure determination	Multiple output architecture	Redundant node avoidance	Compact structure	Stable result
Input weight randomization	ELM	✗	✗	✓	✗	✗	✗
	EM-ELM	✗	✓	✓	✗	✗	✗
	CP-ELM	✗	✓	✓	✓	✓	✗
	DP-ELM	✗	✓	✓	✓	✓	✗
Input weight determination	PCA-ELM	✗	✓	✓	✓	✓	✓
	LDA-PCA-ELM*	✓	✓	✓	✓	✓	✓
Adaptive weight determination	I-ELM	✗	✓	✓	✗	✗	✗
	CI-ELM	✗	✓	✓	✗	✗	✗
	EI-ELM	✗	✓	✓	✓	✓	✗
	AIL	✗	✓	✗	✓	✓	✓

Note: * LDA-PCA-ELM supports only classification tasks.

2.0 BACKGROUND AND PROBLEM FORMULATION

2.1 Extreme Learning Machine and Its Extensions

Extreme learning machine [2], [15] is a well-known and efficient learning method for a single layer feed-forward neural network (SLFN) architecture. It outperforms traditional learning methods, such as gradient descent [16], with extremely fast learning speed and good generalization. ELM efficiency comes from the construction of hidden nodes that are randomly generated from input layer weights and hidden layer bias parameters. A model constructed from the random generation based on a universal approximation theorem [17] can have as many hidden nodes as possible. This succeeds in reducing residual error to zero on any continuous function. Moreover, this method analytically calculates hidden layer weights from many hidden nodes at once, with an ordinary least square method, instead of tuning each parameter; therefore, the local minimum problem [18] is eliminated. Mathematically, the ELM for the SLFN architecture with L additive nodes can be written:

$$\mathbf{T} = \sum_{j=1}^{L} \beta_j \, \mathrm{g}(\mathbf{X} \cdot \mathbf{w}_j + b_j), \tag{1}$$

where $\mathbf{T} \in \mathbb{R}^{N \times M}$ is the desired output matrix with N samples and M objectives, $\mathrm{g}(\cdot)$: $\mathbb{R} \rightarrow \mathbb{R}$ is an activation function, $\mathbf{X} \in \mathbb{R}^{N \times n}$ is the input sample matrix with n features, $\mathbf{w}_j \in \mathbb{R}^{n \times 1}$ is the input layer weight vector that connects each input node to the j-th hidden node, $b_j \in \mathbb{R}$ is the j-th hidden node bias and $\beta_j \in \mathbb{R}^{1 \times M}$ is the hidden layer weight vector that connects the j-th hidden node to every output node. From Eq.(1), we can say that the desired output matrix is generated from a linear combination of the product of the hidden layer weight vector and the activation function; that is, the learning efficiency mainly depends on the hidden layer weight vector. This will be discussed in Subsection 2.2.

Although ELM is more efficient than traditional learning methods, its limitations are unavoidable and are as follows. A suitable number of hidden nodes for an optimal structure are manually set; redundant hidden nodes still remain [3]; and testing speed is always slow due to a large number of generated and redundant hidden nodes. In 2006,

Huang et al. [8] introduced an incremental ELM (I-ELM) to overcome the ELM limitations. They focused on the construction of the optimal structure based on an incremental technique, which adds a single hidden node to the model. This method differs from the traditional ones, in that it does not require any control criteria, such as learning rate and learning epoch counts, to obtain the proper weights. Moreover, I-ELM shows that its hidden layer weights of added hidden nodes are optimal. This optimization attempts to maximize the difference between the previous and the current iteration. However, the hidden nodes of I-ELM are frozen, after they have been added into the model. In other words, the hidden layer weights of I-ELM are not updated, thus causing an inefficient structure. Huang et al. [9] attempted to improve the performance of I-ELM by introducing a convex incremental ELM (CI-ELM) based on Barron's convex optimization. The CI-ELM method updates the existing hidden layer weights each time a hidden node is added. In this concept, CI-ELM demonstrates that the hidden layer weight update can improve the convergence rate and provide a more compact structure. Even though both I-ELM and CI-ELM can construct optimal structures that provide efficient prediction performance and improve the convergence rate, they still suffer from redundant nodes in the hidden layer [12], as in the original ELM. These redundant nodes occur from a very

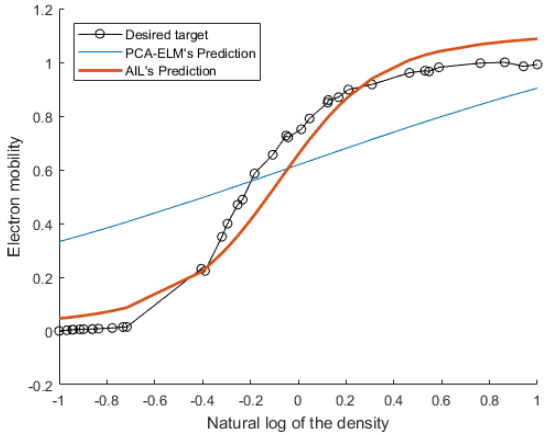


Fig. 1: Prediction for PCA-ELM and AIL using the Thurber dataset [23]

small residual error difference between the previous and current iterations: the newly added hidden node leads to a very small improvement in prediction, increases complexity unnecessarily and reduces the convergence rate. Later, an enhanced incremental ELM (EI-ELM), also introduced by Huang et al. [10], showed that the optimal node among the group of nodes can be found to construct a more compact structure and to improve performance. The optimal node is selected from hidden nodes that provide the smallest residual error of the group of randomly created nodes at each iteration, where the size of the group can be varied. If there is only one hidden node in the given group, then EI-ELM is similar to I-ELM. However, the hidden layer weights of I-ELM are obtained by a non-least-square method. Hence, the estimated output does not reach the minimum error.

Feng et al. introduced an error minimized ELM (EM-ELM) [4] to improve ELM performance by adding one or more hidden nodes to the model continuously. The optimal structure determined by adding a group of hidden nodes can converge faster than traditional incremental learning methods, such as I-ELM. Furthermore, EM-ELM determines proper hidden layer weights and updates existing hidden layer weights with an ordinary least square method, by inverting a block matrix [19]. The block matrix inversion greatly reduces the complexity of the least square method. Nevertheless, EM-ELM still suffers from an over-fitting: it does not guarantee that added hidden nodes will increase the generalization performance [5]. Thus, it forces one to manually set a criterion, the expected learning accuracy, to provide the suitable number of hidden nodes for an optimal structure and avoid over-fitting in a training set.

Castaño et al. introduced principal component analysis ELM (PCA-ELM) to overcome the unstable result [6]. The PCA-ELM method obtains the optimal structure with 90% of the sum of PCA coefficients as sample hidden node parameters. This method can efficiently determine the specific number of hidden nodes and is also fast. For the hidden layer, the hidden layer weights are determined in the same way as an improved ELM [20], using ridge regression. Even though its accuracy tends to be low on complex data patterns, when the input dimension is small, the number of hidden nodes is less than or equal to the number of input dimensions. For example, when the PCA-ELM is applied to a sigmoid-like data pattern with only one independent variable, the accuracy of PCA-ELM

depends on a least square solution of a training set with a zero-mean as can be seen in Fig. 1. Furthermore, the model generated contains only one hidden node. This phenomenon will be analyzed in Subsection 2.3.

Wang et al. introduced constructive and destructive parsimonious ELMs (CP- and DP-ELMs) to eliminate the redundant nodes of the original ELM by using incremental and decremental techniques. These methods gradually search for hidden node regressors that cause the most significant error reduction from a group of randomly created hidden nodes, until the error reduction criterion is satisfied. They can obtain a compact structure, achieve better accuracy and can be applied on an online-learning task, when the training set is fed chunk by chunk. Furthermore, the hidden layer weights are not calculated in the searching procedure but derived from backward substitution. Nevertheless, searching for a group of optimal hidden node regressors is slow. As reported in [21], the QR decomposition with a Givens rotation is commonly used in a recursive orthogonal least square method, which is the main cause of excessive time. Moreover, the unstable result still remains due to the use of randomly created hidden nodes. Abilities of the learning methods are summarized in Table 1.

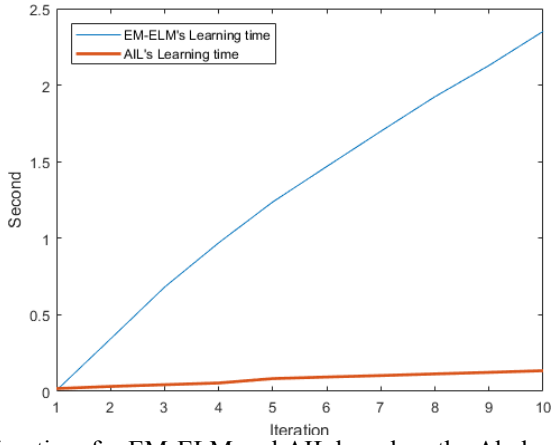


Fig. 2: Learning time for EM-ELM and AIL based on the Abalone dataset [26]

2.2 Analytical Incremental Learning

In 2016, analytical incremental learning (AIL) [13] was described by Alfarozi et al. to obtain an optimal structure under SLFN architecture, by adding an optimal hidden node to the model. Hidden node parameters, input layer weights and hidden layer biases, were approximated from the residual error of a previous learning iteration by using ridge regression, an ordinary least square method with regularization. As a result, the errors of all added hidden nodes were reduced to minima and the over-fitting problem was avoided by regularization. Furthermore, AIL always provided stable results. Another advantage of AIL was its hidden layer parameter determination. AIL was developed on the same basis as ELM. Therefore, the optimal structure was generated based on the hidden layer parameter optimization. Thus, AIL incrementally updated the hidden layer parameters by using an ordinary least square method. Thus a key factor in AIL was the use of recursive block matrix inversion [19], instead of a Moore-Penrose generalized inverse, commonly used in improved ELMs [4], [6], [7], [15], [20]. This sped up the hidden layer parameter determination. In addition, unlike the EM-ELM, the recursive block matrix inversion of AIL was reusable. Hence, the hidden layer parameter determination of AIL had a lower computational cost than the EM-ELM on a large dataset (See Fig. 2). Moreover, AIL was accurate, even on small-input-dimension datasets (See Fig. 1). The SLFN structure for AIL can be written as

$$\mathbf{t} = \sum_{j=1}^{k} \beta_j \, \mathrm{z}(\mathbf{X} \cdot \mathbf{w}_j + b_j) + \beta_0 \,, \tag{2}$$

where $\mathbf{t} \in \mathbb{R}^{N \times 1}$ is the desired output vector, $\mathrm{z}(\cdot)$: $\mathbb{R} \rightarrow \mathbb{R}$ is the invertible bipolar activation function, $\mathbf{X} \in \mathbb{R}^{N \times n}$ is the input matrix with N samples and n features, $\mathbf{w}_j \in \mathbb{R}^{n \times 1}$ is the input layer weight vector that connects each input node, which corresponds to each input feature, to the j-th hidden node, $b_j \in \mathbb{R}$ is the j-th hidden node bias, $\beta_j \in \mathbb{R}$ is the hidden layer weight that connects the j-th hidden node to the output node, $\beta_0 \in \mathbb{R}$ is the output layer bias, and k is the number of hidden nodes. The significant term of AIL, different from that of ELM, expressed in Eq. (1), was the hidden layer parameters. The hidden layer parameters of AIL consist of hidden layer weights β_j and output layer bias β_0, while those of ELM only consist of hidden layer weights, β_j. From Eq. (2), the prediction no longer depends

on the hidden layer weight vector, but includes an output layer bias. Therefore, the output layer bias, β_0, provides a more flexible prediction for the compact model of AIL. As shown in Fig. 3, the prediction of ELM that is modified with an output layer bias outperforms the original ELM, when its structure contains only two hidden nodes. Hence, the output layer bias is important for a learning method that provides a small structure. An experimental justification is shown in Subsection 4.3.

In order to reduce the computational complexity of the structure of AIL, let $\mathbf{H}^{(k)}$ be the hidden layer output matrix of the k-th iteration, let $\beta^{(k)}$ be the compact form of the hidden layer parameters of the k-th iteration, and let L_k be the number of hidden nodes of the k-th iteration. At first, the first column of the hidden layer output matrix, $\mathbf{H}^{(k)}$, is set to a vector of ones. Equation (2) can then be simplified as

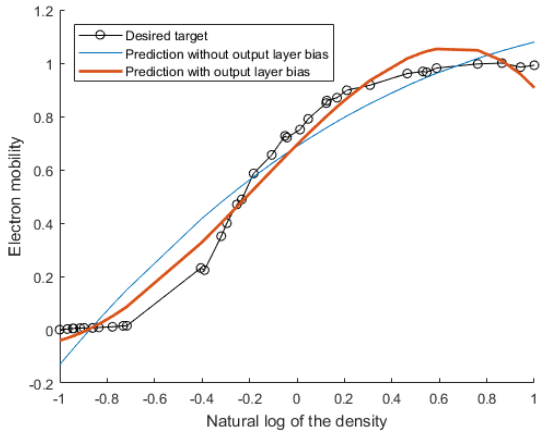


Fig. 3: Prediction of ELM without (blue) and with (red) and output layer bias on the Thurber dataset [23]

$$\mathbf{t} = \mathbf{H}^{(k)}\beta^{(k)}. \tag{3}$$

where

$$\mathbf{H}^{(k)} = \begin{bmatrix} 1 & g(\mathbf{x}_1^{\mathrm{T}}\cdot\mathbf{w}_1 + b_1) & \cdots & g(\mathbf{x}_1^{\mathrm{T}}\cdot\mathbf{w}_k + b_k) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & g(\mathbf{x}_N^{\mathrm{T}}\cdot\mathbf{w}_1 + b_1) & \cdots & g(\mathbf{x}_N^{\mathrm{T}}\cdot\mathbf{w}_k + b_k) \end{bmatrix}_{N\times L_k} \quad \text{and} \quad \beta^{(k)} = \begin{bmatrix} \beta_0 \\ \vdots \\ \beta_k \end{bmatrix}_{L_k}$$

Theoretically, SLFN achieves the desired output with zero error, by providing a very large number of hidden nodes and hidden layer parameters, $\beta^{(k)}$, to solve Eq. (3). However, in practice, it is difficult to achieve the zero-error output. One way to achieve a very small error is by providing a finite number of hidden nodes and optimal hidden layer parameters. The optimal hidden layer parameters are those that provide minimal error between desired and predicted outputs. For AIL, the number of hidden nodes equals the number of learning iterations. Therefore, to determine the optimal hidden layer parameters of AIL with k hidden nodes at the k-th iteration, $\beta^{*(k)}$, it can be incrementally estimated by the least square optimization:

$$\begin{aligned} \beta^{*(k)} &= \left(\mathbf{H}^{(k)^{\mathrm{T}}}\mathbf{H}^{(k)}\right)^{-1}\mathbf{H}^{(k)^{\mathrm{T}}}\mathbf{t} \\ &= \mathbf{S}^{(k)^{-1}}\mathbf{u}(k) \end{aligned} \tag{4}$$

where $\mathbf{S}^{(k)^{-1}} = \left(\mathbf{H}^{(k)^{\mathrm{T}}}\mathbf{H}^{(k)}\right)^{-1}$ and $\mathbf{u}^{(k)} = \mathbf{H}^{(k)^{\mathrm{T}}}\mathbf{t}$. The optimal hidden layer parameters of the current hidden node, $\beta^{*(k)}$, and existing hidden nodes are optimized by a least square method, based on the block recursive generalized inversion, instead of Moore-Penrose inversion. In this way, AIL learns faster on a large hidden layer output matrix, $\mathbf{H}^{(k)}$, because the block recursive inversion complexity is $O(L_k^{\varepsilon})$ [22], where $2 < \varepsilon \ll 3$, whereas $\varepsilon = 3$ for Moore-Penrose Let $\mathbf{h}_k \in \mathbb{R}^{N\times 1}$ be an output vector of a newly created hidden node at the k-th iteration. Then optimal hidden layer parameters are determined by

$$\begin{aligned}\beta^{*(k)} &= \begin{bmatrix} \mathbf{S}^{(k-1)^{-1}} + \alpha^{-1}\boldsymbol{\theta}\boldsymbol{\theta}^{\mathrm{T}} & -\alpha^{-1}\boldsymbol{\theta} \\ -\alpha^{-1}\boldsymbol{\theta}^{\mathrm{T}} & \alpha^{-1} \end{bmatrix} \begin{bmatrix} \mathbf{u}^{(k-1)} \\ h_k^{\mathrm{T}}\mathbf{t} \end{bmatrix} \\ &= \mathbf{S}^{(k)^{-1}}\mathbf{u}^{(k)} \end{aligned} \tag{5}$$

where $\mathbf{v} = \mathbf{H}^{(k-1)^{\mathrm{T}}}\mathbf{h}_k$, $\boldsymbol{\theta} = \mathbf{S}^{(k)^{-1}}\mathbf{v}$, $d = \mathbf{h}_k^{\mathrm{T}}\mathbf{h}_k$, and $\alpha = d - \mathbf{v}^{\mathrm{T}}\boldsymbol{\theta}$. Here, α is a Schur complement [23] that is used to detect the redundancy of the newly created hidden node to existing hidden nodes. Furthermore, AIL's construction of hidden nodes guarantees that every added node provides the optimal parameter and always decreases the norm of residual error, the difference between the desired output and the predicted output, as much as possible. In AIL, if a newly created hidden node is equal or orthogonal to the previous residual error, $\mathbf{e}_r^{(k-1)} \in \mathbb{R}^{N\times 1}$, then zero error is obtained. The construction of the current node can be expressed:

$$\mathbf{h}_k = c\mathbf{e}_r^{(k-1)}, \tag{6}$$

where $c \in \mathbb{R}$ is the scaling parameter of the residual error from the previous iteration, $\mathbf{e}_r^{(k-1)}$. This scaling parameter can be determined by a scaling function, $q(\cdot)$: $\mathbb{R}^{N\times 1} \rightarrow \mathbb{R}$:

$$q\left(\mathbf{e}_r^{(k-1)}\right) = \frac{1-\varepsilon}{\max\left\{\left|\max\left\{\mathbf{e}_r^{(k-1)}\right\}\right|, \left|\min\left\{\mathbf{e}_r^{(k-1)}\right\}\right|\right\}} \tag{7}$$

Then, replacing $\mathbf{h}_k$ of Eq. (6) with $Z\left(\hat{\mathbf{X}}\hat{\mathbf{w}}_k\right)$, we have

$$\mathrm{Z}\left(\hat{\mathbf{X}}\hat{\mathbf{w}}_k\right) = c\mathbf{e}_r^{(k-1)}, \tag{8}$$

where $\mathrm{Z}(\cdot)$ is the element-wise function of $\mathrm{z}(\cdot)$, $\hat{\mathbf{X}} = [\mathbf{1}_{N\times 1}\ \mathbf{X}_{N\times n}]$ is the training sample matrix and $\hat{\mathbf{w}}_k = \left[b_k\ \mathbf{w}_k^{\mathrm{T}}\right]^{\mathrm{T}} \in \mathbb{R}^{n+1\times 1}$ is the compact form of the hidden layer bias and input layer weights of the k-th node. From this, the parameter vector of the k-th hidden node can be computed by taking the inverse of $\mathrm{Z}(\cdot)$ to Eq. (8), leading to:

$$\hat{\mathbf{X}}\hat{\mathbf{w}}_k = \mathrm{Z}^{-1}\left(c\mathbf{e}_r^{(k-1)}\right). \tag{9}$$

Then, approximate Eq. (9), by least squares, to obtain the optimal weight vector:

$$\hat{\mathbf{w}}_k^* = \left(\hat{\mathbf{X}}^{\mathrm{T}}\hat{\mathbf{X}}\right)^{-1}\hat{\mathbf{X}}^{\mathrm{T}}\,\mathrm{Z}^{-1}\left(c\mathbf{e}_r^{(k-1)}\right) \tag{10}$$

Moreover, AIL applies least square optimization with regularization; thus, it is able to avoid ill-conditioned solutions and achieve better generalization.

$$\hat{\mathbf{w}}_k^* = \left(\hat{\mathbf{X}}^{\mathrm{T}}\hat{\mathbf{X}} + \lambda\mathbf{I}\right)^{-1}\hat{\mathbf{X}}^{\mathrm{T}}\,\mathrm{Z}^{-1}\left(c\mathbf{e}_r^{(k-1)}\right), \tag{11}$$

where $\lambda \in \mathbb{R}$ is the regularization hyper-parameter. Hence, the current hidden node is constructed:

$$\mathbf{h}_k = \mathrm{Z}\left(\hat{\mathbf{X}}\hat{\mathbf{w}}_k^*\right) \tag{12}$$

The residual error of the k-th iteration is:

$$\mathbf{e}_r^{(k)} = \mathbf{t} - \mathbf{H}^{(k)}\beta^{*(k)} \tag{13}$$

The steps of the AIL procedure can be summarized:

1. Set $k \leftarrow 0$ and initialize a starting residual error, $\mathbf{e}_r^{(0)}$, from a hidden layer bias, $\mathbf{H}^{(0)} = \mathbf{h}_0 = \mathbf{1}_{N\times 1}$, by using the hidden layer parameters calculated by Eq. (4).
2. $k \leftarrow k + 1$.
3. Calculate the input layer weights and hidden layer bias of the *k*-th iteration, $\hat{\mathbf{w}}_k^*$ using Eq. (11) with respect to the residual error of the previous iteration, $\mathbf{e}_r^{(k-1)}$.
4. Create the hidden node, $\mathbf{h}_k$.
5. If the current node is linearly dependent on an existing node, then stop adding nodes and terminate.
6. Determine the hidden layer parameters of the current iteration, $\beta^{*(k)}$, and update all existing hidden nodes by using Eq. (5).
7. Estimate an output and calculate the residual error of the current iteration, $\mathbf{e}_r^{(k)}$.
8. If the relative residual error of the current iteration is less than the expected value or the number of hidden nodes reaches the maximum value, stop adding nodes and terminate.
9. Repeat Step 2.

2.3 Analysis of Extreme Learning Machine and Analytical Incremental Learning

The following features of all reviewed learning machines, such as objective relation learning, automatic structure determination, multiple-output architecture, redundant node avoidance, compact structure, and stable result, are analyzed and their limitations highlighted. To achieve a compact optimal learning machine, objective relation learning is an important feature. As summarized in Table. 1, all reviewed learning methods, except LDA-PCA-ELM, do not achieve the objective relation learning feature, since they generate a model based on the input-output mapping. For the second feature, all reviewed methods can provide automatic structure determination, except for the original ELM, which was designed for a manually predefined number of hidden nodes. For a multiple-output architecture, all reviewed methods can support this feature, except for the original AIL, which was designed only for the optimal structure generation, by using the residual error vector, but not for a multiple-output architecture. For the fourth and fifth features, ELM, EM-ELM, and I-ELM were analyzed to show that redundant node avoidance and a compact structure was not achieved. The main reasons follow.

Define $\beta^{(k-1)} \in \mathbb{R}^{L_{k-1}\times M}$, a matrix of hidden layer weights from the previous iteration, $\mathbf{H}^{(k-1)} \in \mathbb{R}^{N\times L_{k-1}}$ as a hidden layer output matrix from the previous iteration, $\beta^{(k)} \in \mathbb{R}^{L_k\times M}$ be hidden layer weights of the *k*-th iteration, $\mathbf{H}^{(k)} \in \mathbb{R}^{N\times L_k}$ be a hidden layer output matrix at the *k*-th iteration, $\delta\mathbf{H}^{(k)} \in \mathbb{R}^{N\times \delta L_k}$ be an output matrix of δL_k newly added hidden nodes at the *k*-th iteration, and $\mathbf{T} \in \mathbb{R}^{N\times M}$ is the desired output matrix. To prove that the selected node is redundant, we start with an objective function of the ELM implemented with the incremental technique. The objective function is:

$$\lim_{k\to\infty}\left\|\mathbf{T}-\mathbf{H}^{(k)}\beta^{(k)}\right\|_2^2 = 0 \tag{14}$$

The hidden layer output matrix at iteration, $\mathbf{H}^{(k)}$, can be replaced with a decomposed matrix, $\left[\mathbf{H}^{(k-1)}\ \ \delta\mathbf{H}^{(k)}\right]$, we have

$$\lim_{k\to\infty}\left\|\mathbf{T}-\left[\mathbf{H}^{(k-1)}\quad \delta\mathbf{H}^{(k)}\right]\beta^{(k)}\right\|_2^2 = 0 \tag{15}$$

If $\delta\mathbf{H}^{(k)}$ is randomly created with almost linear dependence on $\mathbf{H}^{(k-1)}$:

$$\left\|\delta\mathbf{H}^{(k)}-\left(\mathbf{H}^{(k-1)^{\mathrm{T}}}\mathbf{H}^{(k-1)}\right)^{-1}\mathbf{H}^{(k-1)^{\mathrm{T}}}\mathbf{H}^{(k)}\right\|_2^2 \approx 0\,, \tag{16}$$

then $\delta\mathbf{H}^{(k)}$ is in the column space of $\mathbf{H}^{(k-1)}$. At this point, we can conclude that the newly added hidden nodes do not contribute to reducing the norm of the residual error. Therefore, the norm of the residual error at the *k*-th iteration is almost equal to the norm of error at the previous iteration:

$$\left\|\mathbf{T}-\mathbf{H}^{(k)}\beta^{(k)}\right\|_2^2 \approx \left\|\mathbf{T}-\mathbf{H}^{(k-1)}\beta^{(k-1)}\right\|_2^2 \tag{17}$$

□

This proves that the redundant hidden node is useless for error reduction. Therefore, redundant node avoidance becomes an essential feature for the compact optimal model. For the last feature, the PCA-ELM, improved by the input layer weight determination, solution can provide stable results, since they do not use randomization. As shown in Table 1, the input layer weight determination strategy succeeded in all features considered here. When we look inside the input layer weight determination strategy, used for PCA-ELM, its accuracy was still low as illustrated in Subsection 4.4. This limitation always arose when PCA-ELM was applied to complex data patterns. Furthermore, as mentioned in Subsection 2.1, when PCA-ELM was applied on a dataset, with a single independent variable, it obtained only one hidden node and performed poorly. The disadvantage of PCA-ELM comes from two causes, which are fundamental PCA constraints: (i) the number of principal components (PC) is always less than or equal to the number of input dimensions: the number of hidden nodes is too small, and (ii) PCA does not provide the bias parameters. To determine hidden nodes, PCA-ELM uses eigenvectors that are decomposed from the covariance matrix, as input layer weights. PCA-ELM computation can be summarized formally as:

$$\mathbf{T} = \sum_{j=1}^{L} \mathrm{g}\left(\bar{\mathbf{X}} \cdot \upsilon_j\right)\beta_j, \tag{18}$$

where $\bar{\mathbf{X}} \in \mathbb{R}^{N\times n}$ is the input with zero mean and $\upsilon_j \in \mathbb{R}^{n\times 1}$ is the eigenvector that is used as input layer weights to connect the j-th hidden node. For a single independent variable dataset, written as $\bar{\mathbf{x}} \in \mathbb{R}^{N\times 1}$, the eigenvector does not have any impact on the input data, since it is always equal to 1 as shown below Eqs. (19) to (21):

$$\sigma^2 \upsilon = \Lambda \upsilon, \tag{19}$$

where $\sigma^2 \in \mathbb{R}$ is the covariance of $\bar{\mathbf{x}}$, $\upsilon \in \mathbb{R}$ is eigenvector, and $\Lambda \in \mathbb{R}$ is eigenvalue. One way to solve Eq. (19) is to substitute 1 for υ.

$$\sigma^2 = \Lambda \tag{20}$$

Then, the hidden node $\mathbf{h}$ of $\bar{\mathbf{x}}$ is created:

$$\begin{aligned} \mathbf{h} &= \mathrm{g}\left(\bar{\mathbf{x}} \cdot \upsilon\right) \\ &= \mathrm{g}\left(\bar{\mathbf{x}}\right) \end{aligned} \tag{21}$$

Therefore, the prediction from PCA-ELM on $\bar{\mathbf{x}}$ is similar to the linear least squares solution of the transformed zero-mean input samples:

$$\Omega\left(\mathbf{h}\right) = \min_{\beta} \left\|\mathbf{T} - \mathbf{h}\beta\right\|_2^2, \tag{22}$$

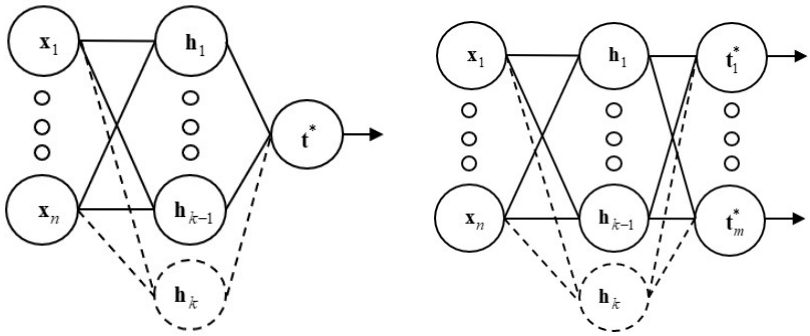
where $\Omega(\cdot)$ is the objective function of PCA-ELM. From this argument, we see that the eigenvector, extracted from a single independent variable, does not affect the hidden node construction. Therefore, the prediction performance of PCA-ELM tends to be low on complex data patterns, with only one independent variable. In Fig. 1, PCA-ELM was tested on the Thurber dataset [23]. This dataset is a complex data pattern, since its characteristic is nonlinear, even though it has only one independent variable. In this case, PCA-ELM did not achieve accurate prediction.

From analysis of EM-ELM, we found that it suffers from slow training speed. This can be observed with the EM-ELM method on large datasets, because the hidden layer weight determination is $O(N^3)$, where N is the number of samples, for every iteration. This complexity can be derived as follows. Let $\mathbf{H}_k \in \mathbb{R}^{N\times L_k}$ be the existing hidden nodes created by EM-ELM, $\delta\mathbf{H}_k \in \mathbb{R}^{N\times \delta L_k}$ be the newly created hidden nodes by EM-ELM, and let † be Moore-Penrose matrix inverse operation. We start with the hidden layer parameter determination expressed by Eq. (23). The hidden layer parameters are determined by matrix multiplication, subtraction and inversion. Time-complexity of those operations depend on the dimension of their operands, $\mathbf{H}_k$ and $\delta\mathbf{H}_k$. In this case, the most complex operation is inversion of an $N \times N$ matrix in the first row of the projection term. Here, the inversion complexity is $O(N^3)$. Therefore, in large datasets, EM-ELM is slower than other ELMs and AIL. To avoid this, EM-ELM adds a large number of hidden nodes at each iteration.

$$\beta_{k+1} = \begin{bmatrix} \left(\left(\mathbf{I} - \mathbf{H}_k \mathbf{H}_k^{\dagger}\right)\delta \mathbf{H}_k\right)^{\dagger} \\ \mathbf{H}_k^{\dagger}\left(\mathbf{I} - \delta \mathbf{H}_k^{\mathrm{T}}\left(\left(\mathbf{I} - \mathbf{H}_k \mathbf{H}_k^{\dagger}\right)\delta \mathbf{H}_k\right)^{\dagger}\right) \end{bmatrix} \mathbf{T}, \tag{23}$$

where β_{k+1} is the hidden layer weights of the (k + 1)-th iteration and $\mathbf{H}_k^{\dagger} = \left(\mathbf{H}_k^{\mathrm{T}}\mathbf{H}_k\right)^{-1}\mathbf{H}_k^{\mathrm{T}}\mathbf{H}$.

As a compact optimal structure is our goal, an adaptive weight determination strategy, implemented with AIL, is the best choice. The main reasons are: AIL (i) can determine the optimal structure automatically, (ii) avoids redundant



(a) Single-output structure (b) Multiple-output structure

Fig. 4: Single and multiple-output node architectures

hidden nodes, (iii) provides a compact structure, (iv) achieves accuracy when applied to small size or small dimension datasets, (v) runs faster, (vi) optimizes hidden layer parameters with a least square method, (vii) provides stable results, and (viii) has bias parameters on both hidden and output layers. However, although the original AIL, which is a single-objective learning method, can solve multiple-objective tasks, it was not as effective and efficient as the multiple-output learning method [26]. In other words, it required a large number of models for a single-objective learning method. Moreover, AIL cannot generate the optimal structure by using the relation between the objectives. This implies that, even though AIL has an advantage in creating hidden nodes, that fit the single-output learning environment, it is still challenging to create an optimal node, that is suitable to a multiple-output node learning environment. In other words, when AIL attempts to create a new hidden node, it used a residual error vector to represent the model performance and to approximate parameters by using an ordinary least square method. In the case of multiple output nodes as shown in Fig. 4, the residual error was changed from vector to matrix form. The number of columns corresponds to the number of output nodes, where the residual error matrix is

$$\mathbf{E}_r = \mathbf{T} - \mathbf{T}^* = \begin{bmatrix} t_{11} - t_{11}^* & \cdots & t_{1M} - t_{1M}^* \\ \vdots & \ddots & \vdots \\ t_{N1} - t_{N1}^* & \cdots & t_{NM} - t_{NM}^* \end{bmatrix}, \tag{24}$$

where $\mathbf{T} \in \mathbb{R}^{N \times M}$ is the predicted output matrix with respect to N samples and M objectives, and $\mathbf{E}_r = \left[\mathbf{e}_{r_1}, \ldots, \mathbf{e}_{r_M}\right] \in \mathbb{R}^{N \times M}$ is the residual error matrix with respect to M objectives. Furthermore, the ridge regression method in Eq. (11) does not provide an objectives relation, but it provides the input layer weight by shrinkage projection of each decoded residual error, $\mathrm{Z}^{-1}(\mathbf{C}\mathbf{E}_r)$, onto training data column space, as seen in Fig. 5, $\widehat{\mathbf{W}} \in \mathbb{R}^{n+1 \times M}$ is the input layer parameter matrix with respect to the residual error matrix,

$$\hat{\mathbf{W}}_k = \left(\hat{\mathbf{X}}^{\mathrm{T}}\hat{\mathbf{X}} + \lambda \mathbf{I}\right)^{-1} \hat{\mathbf{X}}^{\mathrm{T}}\, \mathrm{Z}^{-1}\left(\mathbf{C}\mathbf{E}_r\right), \tag{25}$$

where $\mathbf{C} = [c_1, \ldots, c_M] \in \mathbb{R}^M$ is the scaling parameter vector with respect to each column of residual error matrix (See Eq. (24)). Each scaling parameter c_m is obtained by applying the *corresponding* residual error vector $\mathbf{e}_{r_m}$ to Eq. (7). $\mathrm{Z}^{-1}(\mathbf{C}\mathbf{E}_r)$ in Eq. (25) can be decomposed as

$$\begin{aligned}\hat{\mathbf{W}}_k &= \left(\hat{\mathbf{X}}^{\mathrm{T}}\hat{\mathbf{X}}+\lambda\mathbf{I}\right)^{-1}\hat{\mathbf{X}}^{\mathrm{T}}\left[\mathrm{Z}^{-1}\left(c_1\mathbf{e}_{r_1}\right)\quad\cdots\quad\mathrm{Z}^{-1}\left(c_M\mathbf{e}_{r_M}\right)\right]\\ &=\left[\left(\hat{\mathbf{X}}^{\mathrm{T}}\hat{\mathbf{X}}+\lambda\mathbf{I}\right)^{-1}\hat{\mathbf{X}}^{\mathrm{T}}\,\mathrm{Z}^{-1}\left(c_1\mathbf{e}_{r_1}\right)\quad\cdots\quad\left(\hat{\mathbf{X}}^{\mathrm{T}}\hat{\mathbf{X}}+\lambda\mathbf{I}\right)^{-1}\hat{\mathbf{X}}^{\mathrm{T}}\,\mathrm{Z}^{-1}\left(c_M\mathbf{e}_{r_M}\right)\right],\\ &=\left[\hat{\mathbf{w}}_1\quad\cdots\quad\hat{\mathbf{w}}_M\right],\end{aligned} \tag{26}$$

where $\hat{\mathbf{w}}_m$ is the input layer weight that corresponds to the m-th residual error vector. From Eq. (26), we found that each column vector is independent from other column vectors. Thus, we conclude that the AIL learning procedure cannot preserve the relation among objectives.

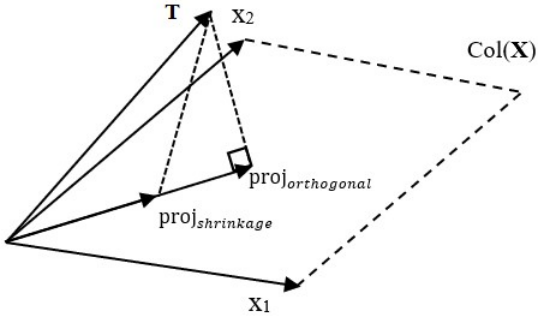


Fig. 5: Geometric illustration of the shrinkage projection

3.0 PCA-AIL METHOD

We aimed to achieve the optimal hidden node solution, compact structure, and multiple-output regression for AIL. Key concepts, implementation processes, and algorithms of PCA-AIL are covered in the following subsections: definition of optimal hidden node and model determination, objective relation estimation and multiple optimal hidden node construction.

3.1 Definition of Optimal Hidden Node and Model Determination

A node that is added to a hidden layer of a single layer feed-forward neural network — a model — is called an optimal hidden node if the model provides a minimal error. Formally, an optimal hidden node can be defined

Definition 1: Let $u(\cdot)$ be an optimal node construction function, $f(\cdot)$ be an objective relation learning (ORL) function, and $\mathbf{E}_r$ be relative objectives of a dataset. The newly created hidden nodes, $\delta\mathbf{H}$, of a single layer feed-forward neural network are determined by

$$\delta\mathbf{H} = u\left(f\left(\mathbf{E}_r\right)\right) \tag{27}$$

$\delta\mathbf{H}$ are called the optimal hidden nodes if they are added to a model, then the model provides the least square norm of residual error based on an objective function of the model with newly created hidden nodes, $\psi(\cdot)$:

$$\psi\left(\delta\mathbf{H}\right) = \min_{\beta}\left\|\mathbf{T}-\left[\mathbf{H}\quad\delta\mathbf{H}\right]\beta\right\|_2^2 \tag{28}$$

where $\mathbf{T}$ is the desired output matrix, β is the set of hidden layer parameters, and $\mathbf{H}$ is the existing hidden nodes.

From *Definition 1*, the optimal hidden nodes always provide a compact optimal model that can be defined in *Definitions 2* and *3*.

Definitions 2: Let β_i be optimal hidden layer weights of the *i*-th hidden node, $\mathbf{h}_i$ be the *i*-th optimal hidden node, which is derived from $\delta\mathbf{H}$ of *Definition 1*, β_0 be optimal output layer biases, and L be the number of optimal hidden nodes. An optimal model ϕ of a single layer feed-forward neural network for multiple outputs is determined by

$$\phi = \sum_{i=1}^{L} \mathbf{h}_i \beta_i + \beta_0 \ . \tag{29}$$

Definitions 3: A model of a single layer feed-forward neural network is called a compact structure, if $\delta\mathbf{H}$ of *Definition 1* is the least square norm of the residual error and the number of optimal hidden nodes, L, of *Definition 2* is minimal.

3.2 Objective Relation Estimation

The objective relation is the key to learning the model in each iteration. It plays an important role in multiple-output neural network architectures and obtaining an optimal node and model. In practice, the objective relation can be estimated by an ORL function and used for obtaining the optimal node and model as defined in *Definitions 1* and *2*. Here, principal component analysis (PCA) was used as the ORL function. The PCA generated the first principal component (PC) and successive PCs for representing the highest and successive variances of relative objectives, respectively. Moreover, all PCs must be orthogonal to each other. According to PCA's properties, the objective relation, $\tilde{\mathbf{E}}_r \in \mathbb{R}^{N\times P}$, can be estimated by applying PCA on a residual error matrix:

$$f\left(\mathbf{E}_r\right) = \tilde{\mathbf{E}}_r = \bar{\mathbf{E}}_r \mathbf{V}_P \,, \tag{30}$$

where $f(\cdot): \mathbb{R}^M \to \mathbb{R}^P$ is PCA method that used as objective relation learning function, $\mathbf{E}_r \in \mathbb{R}^{N\times M}$ is a relative objective matrix with respect to M objectives, which is a residual error matrix, $\bar{\mathbf{E}}_r \in \mathbb{R}^{N\times M}$ is the residual error matrix with respect to M objectives, where the mean is subtracted from each column, and $\mathbf{V}_P \in \mathbb{R}^{N\times P}$ is the matrix of the eigenvectors, selected from the first P elements that satisfy 90% of eigenvalues. This eigenvalue threshold comes from our preliminary experiments, the threshold was set to be 90 %, as a trade-off between obtaining a compact and nearly optimal model. Furthermore, our eigenvalue threshold was similar to that of PCA-ELM and the commonly accepted rule of thumb [27]. The residual error matrix and covariance matrix are computed from:

$$\bar{\mathbf{E}}_r = \begin{bmatrix} e_{r_{11}} - \bar{e}_{r_{11}} & \cdots & e_{r_{1M}} - \bar{e}_{r_{1M}} \\ \vdots & \ddots & \vdots \\ e_{r_{N1}} - \bar{e}_{r_{N1}} & \cdots & e_{r_{NM}} - \bar{e}_{r_{NM}} \end{bmatrix} \tag{31}$$

$$Cov = \frac{\bar{\mathbf{E}}_r^{\mathrm{T}} \bar{\mathbf{E}}_r}{N-1} \tag{32}$$

The PCA coefficient calculated from the covariance matrix of the residual error matrix is computed as

$$Cov\mathbf{V}_P = \Lambda_P \mathbf{V}_P \,, \tag{33}$$

where $\Lambda_P \in \mathbb{R}^{P\times P}$ is the eigenvalue matrix that collects the variance of corresponding PCA coefficients along the diagonal.

3.3 Multiple Optimal Hidden Node Construction

Although the original analytical incremental learning (AIL) was an efficient and effective learning method, it added a single optimal hidden node at each learning iteration. Thus, AIL required many learning iterations. Therefore, we designed a principal component analysis AIL (PCA-AIL) method, that can obtain multiple optimal hidden nodes in each learning iteration. Thus, PCA-AIL required fewer learning iterations than the original AIL. In other words, PCA-AIL converged faster than the original AIL. The optimal hidden nodes were created by applying the objective

relation matrix with P columns. The PCs at each iteration of the learning process obtained P hidden nodes. The hidden nodes of PCA-AIL were determined in the same way as the original AIL. Thus, Eq. (6) can be rewritten as

$$\delta\mathbf{H}^{(k)} = \mathbf{C}\tilde{\mathbf{E}}_r^{(k-1)}, \tag{34}$$

where $\delta\mathbf{H}^{(k)} \in \mathbb{R}^{N\times P}$ are the P newly created hidden nodes at the k-th iteration and $\mathbf{C} \in \mathbb{R}^{P}$ is the scaling parameter vector with respect to each P columns of the objective relation matrix. Scaling parameters are obtained in the same way as Eq. (25) Then, Eq. (34) can be simplified to

$$\mathrm{Z}\left(\hat{\mathbf{X}}\hat{\mathbf{W}}_k\right) = \mathbf{C}\tilde{\mathbf{E}}_r^{(k-1)}, \tag{35}$$

where $\widehat{\mathbf{W}} \in \mathbb{R}^{n+1\times P}$ is the input layer weight matrix and $\widehat{\mathbf{X}} \in \mathbb{R}^{N\times n+1}$ is the input matrix containing a vector of ones. The input layer weight of the k-th hidden nodes from Eq. (35) can be solved by linear algebra as follows.

$$\hat{\mathbf{X}}\hat{\mathbf{W}}_k = \mathrm{Z}^{-1}\left(\mathbf{C}\tilde{\mathbf{E}}_r^{(k-1)}\right) \tag{36}$$

The optimal input layer weight of the ridge regression as defined by Eq. (36) is computed by

$$\hat{\mathbf{W}}_k^* = \left(\hat{\mathbf{X}}^{\mathrm{T}}\hat{\mathbf{X}} + \lambda\mathbf{I}\right)^{-1}\hat{\mathbf{X}}^{\mathrm{T}}\,\mathrm{Z}^{-1}\left(\mathbf{C}\mathbf{E}_r^{(k-1)}\right). \tag{37}$$

Hence, at k-th iteration, we obtained P optimal hidden nodes approximated with the objective relation.

$$\delta\mathbf{H}^{(k)} = \mathrm{Z}\left(\hat{\mathbf{X}}\hat{\mathbf{W}}_k^*\right) \tag{38}$$

After the optimal hidden nodes were completely created, then the hidden layer parameters were determined from the newly created hidden node connecting to each output node and the existing hidden layer parameters were also updated. The hidden layer parameter determination of the k-th iteration is

$$\beta^{*(k)} = \left(\mathbf{H}^{(k)^{\mathrm{T}}}\mathbf{H}^{(k)}\right)^{-1}\mathbf{H}^{(k)^{\mathrm{T}}}\mathbf{T}, \tag{39}$$

and the optimal hidden layer parameter determination of the next iteration can be computed recursively as

$$\begin{aligned}
\beta^{*(k+1)} &= \left(\mathbf{H}^{(k+1)^{\mathrm{T}}}\mathbf{H}^{(k+1)}\right)^{-1}\mathbf{H}^{(k+1)^{\mathrm{T}}}\mathbf{T} \\
&= \left(\begin{bmatrix}\mathbf{H}^{(k)^{\mathrm{T}}}\\ \delta\mathbf{H}^{(k+1)^{\mathrm{T}}}\end{bmatrix}\begin{bmatrix}\mathbf{H}^{(k)} & \delta\mathbf{H}^{(k+1)}\end{bmatrix}\right)^{-1}\begin{bmatrix}\mathbf{H}^{(k)^{\mathrm{T}}}\\ \delta\mathbf{H}^{(k+1)^{\mathrm{T}}}\end{bmatrix}\mathbf{T} \\
&= \begin{bmatrix}\mathbf{H}^{(k)^{\mathrm{T}}}\mathbf{H}^{(k)} & \mathbf{H}^{(k)^{\mathrm{T}}}\delta\mathbf{H}^{(k+1)}\\ \delta\mathbf{H}^{(k+1)^{\mathrm{T}}}\mathbf{H}^{(k)} & \delta\mathbf{H}^{(k+1)^{\mathrm{T}}}\delta\mathbf{H}^{(k+1)}\end{bmatrix}\begin{bmatrix}\mathbf{H}^{(k)^{\mathrm{T}}}\mathbf{T}\\ \delta\mathbf{H}^{(k+1)^{\mathrm{T}}}\mathbf{T}\end{bmatrix} \\
&= \begin{bmatrix}\mathbf{S}^{(k)} & \mathbf{H}^{(k)^{\mathrm{T}}}\delta\mathbf{H}^{(k+1)}\\ \delta\mathbf{H}^{(k+1)^{\mathrm{T}}}\mathbf{H}^{(k)} & \delta\mathbf{H}^{(k+1)^{\mathrm{T}}}\delta\mathbf{H}^{(k+1)}\end{bmatrix}\begin{bmatrix}\mathbf{u}^{(k)}\\ \delta\mathbf{H}^{(k+1)^{\mathrm{T}}}\mathbf{T}\end{bmatrix} = \mathbf{S}^{(k+1)}\mathbf{u}^{(k+1)},
\end{aligned} \tag{40}$$

where $\mathbf{S}^{(k+1)} \in \mathbb{R}^{L_{k+1}\times L_{k+1}}$ is written in a block matrix form as

$$\mathbf{S}^{(k+1)} = \begin{bmatrix} \mathbf{S}_{11} & \mathbf{S}_{12} \\ \mathbf{S}_{21} & \mathbf{S}_{22} \end{bmatrix}; \mathbf{S}_{11} \in \square^{L_k \times L_k}, \tag{41}$$

where $\mathbf{S}_{11} = \mathbf{S}^{(k)}$, $\mathbf{S}_{12} = \mathbf{H}^{(k)^{\mathrm{T}}} \delta \mathbf{H}^{(k)}$, $\mathbf{S}_{21} = \delta \mathbf{H}^{(k)^{\mathrm{T}}} \mathbf{H}^{(k)}$, $\mathbf{S}_{21} = \delta \mathbf{H}^{(k)^{\mathrm{T}}} \delta \mathbf{H}^{(k)}$, L_{k+1} is the total number of hidden nodes at the (k + 1)-th iteration, with L_k existing hidden nodes and P newly created hidden nodes. The inversion of $\mathbf{S}^{(k+1)}$,

$$\mathbf{S}^{(k+1)^{-1}} = \begin{bmatrix} \mathbf{S}^{(k)^{-1}} + \Theta \mathbf{A}^{-1} \Theta^{\mathrm{T}} & -\Theta \mathbf{A}^{-1} \\ -\mathbf{A}^{-1} \Theta^{\mathrm{T}} & \mathbf{A}^{-1} \end{bmatrix}, \tag{42}$$

where $\mathbf{A} = \mathbf{S}_{22} - \mathbf{S}_{12}^{\mathrm{T}} \mathbf{\Theta}$ is the Schur complement and $\mathbf{\Theta} = \mathbf{S}_{11}^{-1} \mathbf{S}_{12}$. The stopping criteria for PCA-AIL are the same as for AIL. However, the linear dependence criterion differs from the original one, since the Schur complement of PCA-AIL is not a scalar. In addition, the determinant of the Schur complement can be used to confirm the linear dependence of the newly created and existing hidden nodes. In other words, when the relation between the determinant $\mathbf{A}$ and $\mathbf{S_{22}}$ is less than the expected value, $0 < \eta \ll 1$, the newly created hidden nodes are dependent on the existing hidden nodes.

$$idp = \left| \frac{\det(\mathbf{A})}{\det(\mathbf{S}_{22})} \right| \leq \eta \tag{43}$$

As can be seen in Eqs. (37) and (42), the time complexities are for the input layer weight determination $O(n^3)$ and hidden layer parameter determination is $O(L_k^{2+\varepsilon})$, where n is the number of input features and L_k is the number of hidden nodes at the k-th iteration such that $0 < \varepsilon < 1$. Therefore, the time complexity of PCA-AIL depends on the maximum of the number of input features and the number of hidden nodes. Usually, the overall complexity is $O(L_k^{2+\varepsilon})$ since there are k learning iterations. However, in the worst case when n is much larger than L_k, the complexity is dominated by $O(n^3)$. The accuracy and compactness of the PCA-AIL was demonstrated in subsection 4.2.

The learning procedure of the PCA-AIL mainly differed from the original AIL in the objective relation used in optimal hidden node construction. It can be briefly explained as follows.

1. Set $k \leftarrow 0$ and initialize a starting residual error, $\mathbf{E}_r^{(0)}$, from the hidden layer bias, $\mathbf{H}^{(0)} = \mathbf{h}_0 = \mathbf{1}_{N \times 1}$, by using the hidden layer parameters calculated by Eq. (39).
2. $k \leftarrow k + 1$,
3. Calculate the objective relation $\tilde{\mathbf{E}}_r^{(k)}$ from the residual error matrix of the previous iteration, $\mathbf{E}_r^{(k-1)}$, by using Eq. (30).
4. Determine the input layer weights and hidden layer biases of the k-th iteration, $\widehat{\mathbf{W}}_k^*$, by using Eq. (37) with respect to the objective relation,
5. Calculate P hidden nodes output $\delta \mathbf{H}^{(k)}$ by using Eq. (38).
6. Calculate $\mathbf{S}^{(k)}$, $\mathbf{S}_{22}$, and $\mathbf{A}$ by using Eqs. (41) and (42).
7. After that, if the linearly dependent criterion defined in Eq. (43) is satisfied, then stop adding nodes.
8. Calculate the hidden layer parameters of the current iteration, $\beta^{*(k)}$, and update existing hidden node by using Eq. (40).
9. Estimate the output and calculate the residual error of the current iteration, $\mathbf{E}_r^{(k)}$.
10. If the relative residual error of the current iteration is less than an expected criterion or the maximum number of hidden nodes is satisfied with a condition, then stop adding nodes and terminate.
11. Repeat Step 2.

4.0 EXPERIMENTS AND DISCUSSION

In this section, PCA-AIL was evaluated in three ways: (i) the improvement of convergence rate and multiple-output architecture, (ii) the performance improvement from the output layer bias, and (iii) the performance evaluation on real-world datasets. For these assessments, three experiments were set up to test each of these aspects.

4.1 Evaluation Metric and Data Preparation

The performance metrics for all test methods were the root mean square error (*RMSE*), the number of hidden nodes, and the training and testing time. *RMSE* is a prediction performance metric, measuring the difference between the predicted and desired outputs:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}\sum_{j=1}^{M}\left(t_{ij} - t_{ij}^{*}\right)^{2}}{N}}, \tag{44}$$

where $t_{ij} \in \mathbb{R}$ is the desired output of the i-th sample on the m-th objective, $t_{ij}^{*} \in \mathbb{R}$ is the predicted output of the i-th sample on the m-th objective, N is the number of samples, and M is the number of the objectives. A low *RMSE* value means that a learning method can correctly predict, whereas a high *RMSE* value means that a learning method cannot correctly predict. The number of hidden nodes represents the complexity of the models; that is, a small number of hidden nodes mean that the model is more compact whereas a large number of hidden nodes mean that the model is less compact. Lastly, the training and testing time evaluates the learning speed.

Table 2: Specifications of benchmark datasets

Group	Name	Train	Test	Feature	Objectives	Criterion
1	andro	33	16	30	6	0.20
	edm	103	51	16	2	0.20
	slump	69	34	7	3	0.20
2	Enb	512	256	8	2	0.08
	jura	239	120	15	3	0.10
	sf1	215	108	10	3	0.15
	sf2	711	355	10	3	0.07
	wq	707	353	16	14	0.25
3	atp1d	225	112	411	6	0.10
	atp7d	197	99	411	6	0.10
	oes10	269	134	298	16	0.07
	oes97	223	111	263	16	0.08
4	rf1	6003	3002	64	8	0.08
	scm20d	5977	2989	61	16	0.07
5	rf2	5567	2784	576	8	0.10
	scm1d	6535	3268	280	16	0.10

Real-world datasets were used in experiments. These datasets were retrieved from Spyromitros-Xioufis et al. [25] and UCI repository [26], which contain various sample and feature sizes and consist of five groups: (i) Datasets with

small size and few features - Andromeda (andro), Electronic discharge management (edm), and Slump test (slump); (ii) Datasets with medium size and few features - Energy building (enb), Jura, Solar Flare (sf1, sf2) and Water quality (wq); (iii) Datasets with medium size and many features - Airline ticket price (atp1d, atp7d) and Occupational employment survey (oes10, oes97); (iv) Datasets with large size and few features - River flow 1 (rf1) and Supply chain management 20 days (scm20d); and (v) Datasets with large size and many features - River flow 2 (rf2) and Supply chain management 1 day (scm1d). Table 2 summarizes characteristics of the benchmark datasets.

The experiment used the holdout cross-validation method, which randomly selects 75% of samples as the training set and 25% of samples as the testing set. Furthermore, the samples were pre-processed by removing samples with missing values, then the input data was normalized into [-1, 1], whereas the output data is normalized into [0, 1]. To evaluate the stability of the proposed and the baseline methods, each method was executed 30 times for each dataset. All the test algorithms, implemented with MATLAB, on an Intel Xenon E3-1226 v3 (3.30 GHz) with 8 GB RAM PC. The hyperparameters of both methods were obtained by cross-validation. The hyperparameters that provide the smallest *RMSE* were selected. The maximum number of hidden nodes was selected from a set of multiples of 10 - {10, 20, 30, …, 100}. The maximum number of hidden nodes for PCA-ELM was equal to the feature size of the datasets. The regularization parameter, which is used on ELM [20], PCA-ELM [6], and PCA-AIL, was selected as powers of 2 - $\{2^{-25}, 2^{-24}, 2^{-23}, \ldots, 2^{10}\}$. The size of the adding and searching group that was used in EM-ELM [4] and EI-ELM [10], were both selected from the set - {5, 10, 15, 20}. The expected learning accuracy criterion that is used in EM-ELM [4] and EI-ELM [10] was selected from {0.01, 0.02, 0.04, 0.06, …, 0.30} as summarized in Table. 2. The error reduction in CP- and DP-ELM was 0.001 (to 3-digit precision) [5]. The learning criteria for PCA-AIL, η is 0.00001 (5-digit precision) was taken from Alfarozi et al. [13]. The activation function of the baseline methods was a sigmoid. On the other hand, our method used a hyperbolic tangent activation function.

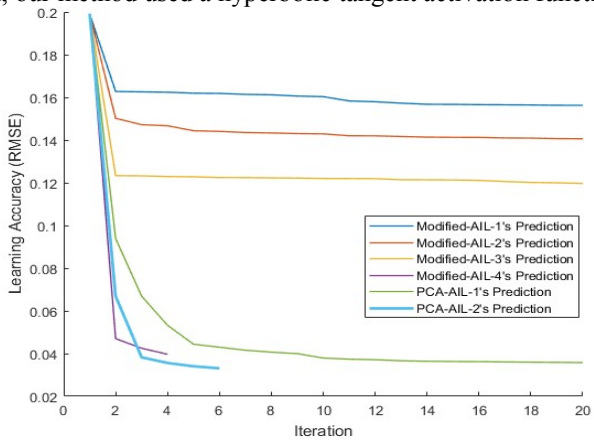


Fig. 6: Learning accuracy of modified AIL$_s$ and PCA-AIL$_s$ vs learning iteration

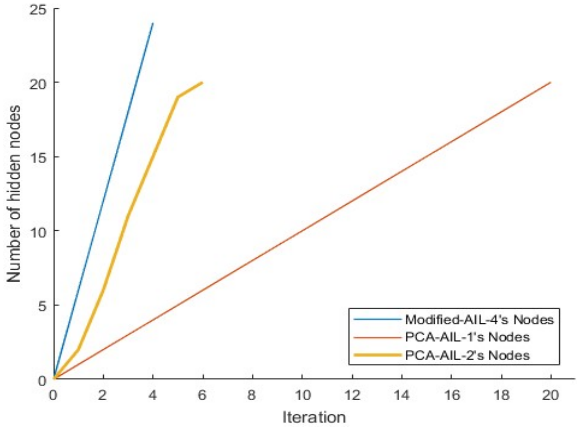


Fig. 7: Number of hidden nodes of modified-AIL-4 and PCA-AIL$_s$ on each learning iteration

4.2 Learning Accuracy Comparison between Modified AILs and PCA-AIL

The first experiment illustrated the improvement of convergence rate and multiple-output architecture according to *Definitions 1* and *2*. Following these definitions, AIL that was modified by selecting residual error suffered from a lack of objective relations. This problem caused the modified AIL to not provide an optimal node and structure. Furthermore, it caused a slow convergence rate. The baseline methods were modified AILs, that were fell into two groups. The first group was modified by selecting the residual error that corresponds to the first, second, and third objectives, called Modified-AIL-1, Modified-AIL-2, and Modified-AIL-3. The second group was modified by adding all hidden nodes that were obtained from the residual error matrices, Modified-AIL-4. Our method was used with two variants, singly added hidden nodes (PCA-AIL-1) and multiple added hidden nodes (PCA-AIL-2). The first experiment used the Airline ticket prices 7-day dataset (atp7d), which has a small and complex data pattern. Moreover, there is a relation between the objectives.

Fig. 6 shows that the first three baseline methods did not predict correctly, since they only preserved the corresponding objective, with *RMSEs* of 0.156, 0.141, and 0.120. On the other hand, our methods, PCA-AIL-1 and PCA-AIL-2, outperformed those baseline methods by reducing the *RMSE* factors of 4.20 (PCA-AIL-1) and 3.88 (PCA-AIL-2), because they were able to preserve relations among the objectives. Moreover, although the last method, Modified-AIL-4, can avoid the lost detail problem, its hidden nodes were not optimal. This problem came from Modified-AIL-4 not approximating the hidden node with the relation between the objectives. Hence, our methods achieved better prediction than Modified-AIL-4 by 1.20 (PCA-AIL-1) and 1.11 times (PCA-AIL-2). This showed that selecting the residual error to approximate hidden nodes caused inefficient hidden nodes. Moreover, using the relation between the objectives improved the hidden node effectiveness and convergence rate. Then it can be concluded that this experiment demonstrates *Definitions 1* and *2*.

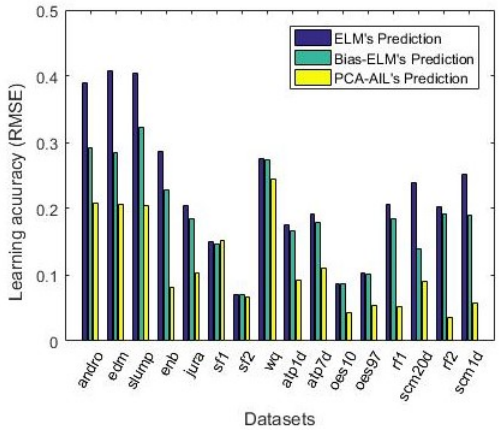


Fig. 8: Learning accuracy for three learning methods: ELM, Bias-ELM, and PCA-AIL

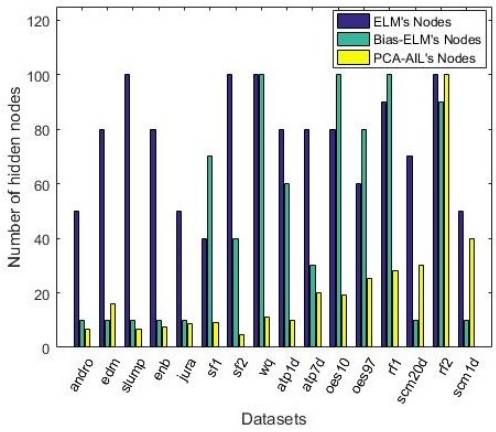


Fig. 9: Number of hidden nodes for three learning methods: ELM, Bias-ELM, and PCA-AIL

Furthermore, we assumed that adding multiple hidden nodes at each iteration would improve the convergence rate. Therefore, to show this, the number of hidden nodes was also considered. Fig. 6 shows that PCA-AIL-2 achieved the best prediction performance at fewer learning iterations, with *RMSE* of 0.038 after 3 iterations. On the other hand, the first three of the modified AIL group and PCA-AIL-1 required more than 20 iterations to achieve *RMSE* 0.038. Moreover, PCA-AIL-2 generated a large number of hidden nodes in a few iterations, as shown in Fig. 7. Although PCA-AIL-1 is not as effective as PCA-AIL-2, PCA-AIL-1 converged faster than the Modified-AILs with single added hidden nodes. This showed that adding multiple hidden nodes at each iteration increased the convergence rate.

4.3 Performance Improvement from Output Layer Bias

The second experiment demonstrated that the output layer bias would improve the performance, especially for solving regression tasks with a small-sized SLFN. In this experiment, Bias-ELM implemented with Eq. (45) derived from Eq. (1) by adding an output bias term, β_0.

$$\mathbf{T}_0 = \sum_{j=1}^{L} \mathrm{g}\left(\mathbf{X} \cdot \mathbf{w}_j + b_j\right)\beta_j + \beta_0 \tag{45}$$

where $\mathbf{T}_0 \in \mathbb{R}^{N \times M}$ is the desired output matrix of the Bias-ELM with *N* samples and *M* objectives, $\mathrm{g}(\cdot) : \mathbb{R} \to \mathbb{R}$ is an activation function, $\mathbf{X} \in \mathbb{R}^{N \times n}$ is the input matrix with *N* samples and *n* features, $\mathbf{w}_j \in \mathbb{R}^{n \times 1}$ is the input layer weight vector that connects each input node to the *j*-th hidden node, $b_j \in \mathbb{R}$ is the *j*-th hidden node bias, $\beta_j \in \mathbb{R}^{1 \times M}$ is the hidden layer weight vector that connects the *j*-th hidden node to every output node, $\beta_0 \in R^M$ is the vector of output layer biases, and *L* is the number of hidden nodes. Weight and bias parameters of Bias-ELM were determined in the same way as of the original ELM. The learning accuracy, *RMSE*, and the number of hidden nodes were used as a performance metric. As shown in Fig. 8, Bias-ELM outperformed the original ELM for learning accuracy, with better *RMSEs* on 14 of 16 datasets: andro, edm, slump, enb, jura, sf1, wq, atp1d, atp7d, oes97, rf1, scm20d, rf2, and scm1d. For Bias-ELM, the *RMSE*s were lower by factors ranging from 1.01 to 1.72 less the original ELM. The other two datasets: sf2 and oes10 were comparable. At the same time, the number of hidden nodes for Bias-ELM was less than for the original ELM in most cases, except for sf1, oes10, oes97, and rf1 datasets - see Fig. 9. This improvement came from the output layer bias that can be considered as a synthesized hidden node. In contrast, activation of output nodes of ELM-based methods was a linear combination; hence, the output layer bias is considered as a y-intercept parameter, which gives the linear equation an additional degree of freedom. In the same way, the learning accuracy of PCA-AIL was better than that of ELM and Bias-ELM for all except one (sf1) of the 16 datasets, and the number of hidden nodes of PCA-AIL was less than for ELM and Bias-ELM in almost every case.

4.4 Performance Evaluation

The third experiment evaluated and compared the performance between the PCA-AIL and the baseline methods, based on 16 real-world benchmark datasets. Results are shown in Tables 3 – 6. The experimental results are discussed below:

Table 3 compares the predictions. It can be seen that the PCA-AIL outperformed the others in terms of *RMSE* on dataset groups 1, 3, and 5. Moreover, PCA-AIL was better in 3 out of 5 datasets in group 2. This achievement came from the objectives of each dataset were related and PCA-AIL extracted the relationships, as additional features: an example from the Airline Ticket Price datasets follows. The objectives of Airline Ticket Price dataset were to predict next-day ticket (atp1d) prices or minimum price observed over the next 7 days (atp7d) for 6 target flight preferences: any airline and any number of stops, any airline non-stop only, Delta Airlines, Continental Airlines, AirTran Airlines and United Airlines. In the prediction, not only the number of days between the observation date and the departure date or day-of-the-week could be used to predict the ticket price, but the relation between ticket prices, among the companies, could be used to predict future prices. For example, if the prices of United Airlines were changed, then it may affect prices of other companies. PCA-AIL achieved better predictions than ELM, EM-ELM, CP-ELM, DP-ELM, PCA-ELM, and EI-ELM by multiples of from 1.22 to 2.03, in the atp1d dataset. For the

atp7d dataset, PCA-AIL achieved better predictions by factors from 1.31 to 1.67. Moreover, PCA-AIL provided the lowest standard deviation of prediction performance on most datasets and thus showed stable performance. Thus, using the objective relation, to preserve relationships among the objectives efficiently, extended the ability of AIL not only on objective relation learning and multiple-output architecture, in Table 1, but also stable prediction.

Table 4 shows the number of hidden nodes for each method. In overall, the number of hidden nodes for PCA-AIL is less than that of all other test methods, except PCA-ELM, at 6.33 nodes. Based on *Definition 3*, PCA-AIL did not obtain the most compact structure, but it achieved a compact optimal structure - see Tables 3 and 4. Thus PCA-AIL method not only provides the optimal structure but also a compact structure. For example, PCA-AIL achieved the best prediction performance, 0.07, and the number of hidden nodes, 4.4, in the sf2 dataset. For the sf1 dataset, PCA-AIL *RMSE* essentially overlapped ranges for the other methods and showed more than 2.9 nodes less for all the other methods. This showed that the relative objective was important for optimal hidden node construction, which strongly supports PCA-AIL to achieve redundant node avoidance and a compact structure.

From Tables 3 and 4, even though in most cases PCA-ELM achieved the minimum number of hidden nodes, it did not achieve the minimum *RMSE*s. In this situation, PCA-ELM did not satisfy our *Definition 3*. Therefore, PCA-ELM did not generate a compact structure.

Table 5 compares training times. PCA-AIL was in the top three of the fastest learning methods. Moreover, for large size datasets, groups 4 and 5 in Table 2, PCA-AIL was faster than EM-ELM, CP-ELM, DP-ELM, and EI-ELM by factors from 2.3 to 22.4 times. In addition, PCA-AIL outperformed PCA-ELM, which is the top one, on large feature datasets - atp1d (1.2 times) and oes97 (1.1 times). When the number of hidden nodes was considered along with training time, PCA-AIL also built a large structure faster than most other methods; for example, it built the largest structure on the rf2 dataset, but generated the structure faster than other methods, such as CP-ELM, DP-ELM and EI-ELM, 3.1 to 12.0 times faster. This achievement came from the multiple hidden nodes adding and the basis of the original AIL. When prediction was also considered, PCA-AIL achieved accurate predictions with acceptable training time. This implied that the PCA-AIL was also a fast learning machine.

Table 6 compares testing times. PCA-AIL predicted the result faster than all the other methods (from 1.1 to 3.9 times), except for PCA-ELM. PCA-ELM achieved the fast testing speed because it provided a small structure, as shown in Table 4, and it did not have bias parameters. From Tables 3 and 4, it is clear that PCA-AIL provides the most compact optimal structure. Therefore, the slower testing speed of PCA-AIL is acceptable, since it determined bias parameters that significantly improved prediction performance.

5.0 CONCLUSION

A new Analytical Incremental Learning based on Principal Component Analysis, PCA-AIL, achieved a compact optimal structure and supported multiple-output regression tasks. The contributions of PCA-AIL were the use of an adaptive weight determination strategy and an objective relation learning function (ORL), that obtained multiple optimal nodes for constructing the hidden layer of a single layer feed-forward neural network. The algorithm implemented on our *Definitions* as three main steps. First, the relative objectives of a dataset were calculated, and then the ORL function was applied to them. Second, the optimal node construction function generated the optimal nodes, and finally, the optimal model was constructed. The generated model was tested with 16 multiple-objective regression datasets. Compared to six other methods (ELM, EM-ELM, CP-ELM, DP-ELM, PCA-ELM, EI-ELM), PCA-AIL performed better than most of the other methods in terms of *RMSE* - 0.11261 ± 0.00911, number of hidden nodes - 19.9 nodes, training time - 0.0466 second, and testing time - 0.0017 second. Thus PCA-AIL achieved a fast testing speed, a stable result, a compact model and more accurate performance. Using the objective relation to preserve relationships was the key to achieving these objectives efficiently.

REFERENCES

[1] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: a tutorial," *Computer*, vol. 29, no. 3, Mar. 1996, pp. 31–44.

[2] G. B. Huang, Q. Y. Zhu, and C. K. Siew, "Extreme learning machine: a new learning scheme of feedforward neural networks," in *2004 IEEE International Joint Conference on Neural Networks*, 2004, vol. 2, pp. 985–990.

[3] H. Shao and N. Japkowicz, "Applying Least Angle Regression to ELM," in *Advances in Artificial Intelligence*, 2012, pp. 170–180.

[4] G. Feng, G. B. Huang, Q. Lin, and R. Gay, "Error Minimized Extreme Learning Machine With Growth of Hidden Nodes and Incremental Learning," *IEEE Trans. Neural Net.*, vol. 20, no. 8, Aug. 2009, pp. 1352–1357.

[5] N. Wang, M. J. Er, and M. Han, "Parsimonious Extreme Learning Machine Using Recursive Orthogonal Least Squares," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 10, Oct. 2014, pp. 1828–1841.

[6] A. Castaño, F. Fernández-Navarro, and C. Hervás-Martínez, "PCA-ELM: A Robust and Pruned Extreme Learning Machine Approach Based on Principal Component Analysis," *Neural Process. Lett.*, vol. 37, no. 3, Jun. 2013, pp. 377–392.

[7] A. Castaño, F. Fernández-Navarro, A. Riccardi, and C. Hervás-Martínez, "Enforcement of the principal component analysis - extreme learning machine algorithm by linear discriminant analysis," *Neural Comput. Appl.*, vol. 27, no. 6, 2016, pp. 1749–1760.

[8] G. B. Huang, L. Chen, and C. K. Siew, "Universal approximation using incremental constructive feedforward networks with random hidden nodes," *IEEE Trans. Neural Netw.*, vol. 17, no. 4, Jul. 2006, pp. 879–892.

[9] G. B. Huang and L. Chen, "Convex incremental extreme learning machine," *Neurocomputing*, vol. 70, no. 16, Oct. 2007, pp. 3056–3062.

[10] G. B. Huang and L. Chen, "Enhanced random search based incremental extreme learning machine," *Neurocomputing*, vol. 71, no. 16, Oct. 2008, pp. 3460–3468.

[11] G.B. Huang, M.B. Li, L. Chen, and C.K. Siew, "Incremental extreme learning machine with fully complex hidden nodes," *Neurocomputing*, vol. 71, no. 4, Jan. 2008, pp. 576–583.

[12] L. Ying, "Orthogonal incremental extreme learning machine for regression and multiclass classification," *Neural Comput. Appl.*, vol. 27, no. 1, Jan. 2016, pp. 111–120.

[13] S. A. I. Alfarozi, N. A. Setiawan, T. B. Adji, K. Woraratpanya, K. Pasupa, and M. Sugimoto, "Analytical Incremental Learning: Fast Constructive Learning Method for Neural Network," in *Neural Information Processing*, 2016, pp. 259–268.

[14] A. E. Hoerl and R. W. Kennard, "Ridge Regression: Biased Estimation for Nonorthogonal Problems," *Technometrics*, vol. 12, 1970, pp. 55–67.

[15] G. B. Huang, Q. Y. Zhu, and C. K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, no. 1, 2006, pp. 489–501.

[16] P. Baldi, "Gradient descent learning algorithm overview: a general dynamical systems perspective," *IEEE Trans. Neural Netw.*, vol. 6, no. 1, Jan. 1995, pp. 182–195.

[17] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Math. Control Signals Syst.*, vol. 2, no. 4, Dec. 1989, pp. 303–314.

[18] B. C. Cetin, J. W. Burdick, and J. Barhen, "Global descent replaces gradient descent to avoid local minima problem in learning with artificial neural networks," in *IEEE International Conference on Neural Networks*, 1993, vol.2, pp. 836–842.

[19] T. Banachiewicz, "Zur Berechnung der Determinanten, wie auch der Inversen und zur darauf basierten Auflosung der Systeme linearer Gleichungen," *Acta Astron. Ser C*, vol. 3, 1937, pp. 41–67.

[20] G. B. Huang, H. Zhou, X. Ding, and R. Zhang, "Extreme Learning Machine for Regression and Multiclass Classification," *IEEE Trans. Syst. Man Cybern. Part B Cybern.*, vol. 42, no. 2, pp. 513–529, Apr. 2012.

[21] Y.-P. Zhao and R. Huerta, "Improvements on parsimonious extreme learning machine using recursive orthogonal least squares," *Neurocomputing*, vol. 191, May 2016, pp. 82–94.

[22] M. D. Petković and P. S. Stanimirović, "Generalized matrix inversion is not harder than matrix multiplication," *J. Comput. Appl. Math.*, vol. 230, no. 1, Aug. 2009, pp. 270–282.

[23] A. M. Ostrowski, "On Schur's complement," *J. Comb. Theory Ser. A*, vol. 14, no. 3, May 1973, pp. 319–323.

[24] H. Borchani, G. Varando, C. Bielza, and P. Larrañaga, "A survey on multi-output regression," *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, vol. 5, no. 5, Sep. 2015, pp. 216–233.

[25] E. Spyromitros-Xioufis, G. Tsoumakas, W. Groves, and I. Vlahavas, "Multi-target regression via input space expansion: treating targets as inputs," *Mach. Learn.*, vol. 104, no. 1, 2016, pp. 55–98.

[26] D. Dheeru and E. Karra Taniskidou, *UCI Machine Learning Repository*. University of California, Irvine, School of Information and Computer Sciences, 2017.

[27] A. Rea and W. Rea, "How Many Components should be Retained from a Multivariate Time Series PCA?," arXiv:1610.03588 [stat], Oct. 2016.

Table 3: Relative prediction performance (*RMSE*) on benchmark datasets

Dataset	ELM		EM-ELM		CP-ELM		DP-ELM		PCA-ELM		EI-ELM		PCA-AIL	
	Avg.	SD	Avg.	SD	Avg.	SD	Avg.	SD	Avg.	SD	Avg.	SD	Avg.	SD
andro	0.3938	0.0714	0.2323	0.0291	0.2465	0.0442	0.2466	0.0217	0.2627	0.0713	0.2566	0.0450	**0.2064**	0.0341
edm	0.3920	0.0759	0.2158	0.0119	0.2161	0.0177	0.2251	0.0138	0.2163	0.0137	0.2218	0.0175	**0.2069**	0.0132
slump	0.4070	0.0704	0.2086	0.0240	0.2045	0.0227	0.2197	0.0162	0.2222	0.0227	0.2116	0.0202	**0.2037**	0.0148
enb	0.3002	0.1184	0.0854	0.0050	0.0801	0.0047	0.1861	0.0305	0.0599	0.0052	0.0612	0.0050	0.0810	0.0031
jura	0.2150	0.0446	0.1177	0.0111	0.1147	0.0080	0.1524	0.0132	0.1119	0.0083	0.1116	0.0083	**0.1024**	0.0069
sf1	0.1505	0.0227	0.1547	0.0238	0.1572	0.0169	0.1490	0.0170	0.1604	0.0183	0.1558	0.0200	0.1535	0.0212
sf2	0.0705	0.0098	0.0693	0.0088	0.0710	0.0086	0.0677	0.0073	0.0692	0.0069	0.0708	0.0074	**0.0662**	0.0063
wq	0.2760	0.0097	0.2521	0.0057	0.2490	0.0045	0.2485	0.0026	0.2503	0.0096	0.2489	0.0049	**0.2448**	0.0033
atp1d	0.1868	0.0277	0.1448	0.0120	0.1267	0.0114	0.1122	0.0082	0.1190	0.0135	0.1152	0.0105	**0.0915**	0.0047
atp7d	0.1850	0.0274	0.1578	0.0173	0.1492	0.0139	0.1370	0.0151	0.1509	0.0211	0.1447	0.0160	**0.1112**	0.0087
oes10	0.0873	0.0235	0.0691	0.0105	0.0730	0.0173	0.0605	0.0112	0.1210	0.1247	0.1332	0.2090	**0.0467**	0.0140
oes97	0.0993	0.0251	0.0795	0.0176	0.0826	0.0239	0.0645	0.0114	0.1055	0.0862	0.1259	0.1875	**0.0538**	0.0120
rf1	0.1964	0.0423	0.0703	0.0014	0.0666	0.0026	0.1937	0.0020	0.0436	0.0014	0.0438	0.0012	0.0509	0.0009
scm20d	0.2372	0.1151	0.1013	0.0021	0.0989	0.0015	0.1133	0.0034	0.0894	0.0014	0.0896	0.0012	0.0895	0.0007
rf2	0.2019	0.0441	0.1065	0.0077	0.1006	0.0021	0.1081	0.0026	0.0944	0.0050	0.0950	0.0041	**0.0361**	0.0011
scm1d	0.2402	0.0790	0.0819	0.0031	0.0825	0.0051	0.1053	0.0019	0.0740	0.0071	0.0725	0.0023	**0.0571**	0.0008

Table 4: Number of hidden nodes on benchmark datasets

Dataset	ELM			EM-ELM			CP-ELM			DP-ELM			PCA-ELM			EI-ELM			PCA-AIL		
	Max	Avg.	SD	Max	Avg.	SD	Max	Avg.	SD	Max	Avg.	SD	Max	Avg.	SD	Max	Avg.	SD	Max	Avg.	SD
andro	90	90.00	-	10	9.00	2.03	10	9.97	0.18	10	8.67	0.48	30	3.63	0.49	30	13.87	5.15	10	7.33	1.30
edm	50	50.00	-	20	13.67	3.20	10	9.70	0.47	10	15.50	1.14	16	5.20	0.41	50	32.10	13.67	20	16.40	1.10
slump	70	70.00	-	20	10.33	1.83	10	9.83	0.38	10	7.63	0.67	7	5.00	-	40	20.67	10.19	10	6.90	0.66
enb	70	70.00	-	40	21.17	5.03	100	93.33	3.72	100	62.70	3.36	8	4.00	-	90	86.13	14.75	10	6.60	0.56
jura	60	60.00	-	30	19.33	2.86	30	29.63	0.49	30	25.47	0.90	15	6.33	0.48	70	53.20	11.77	10	9.70	2.58
sf1	40	40.00	-	30	18.50	10.84	10	10.00	0.00	10	8.93	0.25	10	10.97	0.18	20	11.73	9.14	10	**6.00**	-
sf2	90	90.00	-	10	8.33	2.40	10	10.00	0.00	10	8.83	0.38	10	10.00	-	80	29.50	30.63	10	**4.40**	0.81
wq	100	100.00	-	20	9.50	2.01	20	20.00	0.00	20	18.90	0.31	16	7.00	-	50	20.87	9.36	20	11.00	-
atp1d	80	80.00	-	20	8.33	2.73	20	19.80	0.41	40	34.40	1.04	411	16.40	0.67	20	7.70	4.49	10	10.00	-
atp7d	60	60.00	-	20	11.33	3.70	20	19.83	0.38	40	26.27	0.87	411	22.13	0.68	20	5.40	2.62	30	30.20	3.87
oes10	70	70.00	-	10	5.67	1.73	10	10.00	-	10	8.90	0.31	298	11.73	2.65	10	1.77	0.68	20	13.43	2.74
oes97	60	60.00	-	10	5.67	1.73	10	10.00	-	10	9.00	-	263	11.53	3.25	10	1.93	0.52	20	14.53	1.96
rf1	100	100.00	-	30	19.67	3.20	100	90.20	2.01	100	70.73	2.00	64	3.00	-	80	64.30	15.94	30	30.00	-
scm20d	50	50.00	-	40	35.00	3.94	100	86.87	3.03	100	64.30	1.84	61	11.00	-	100	79.37	18.19	20	12.00	-
rf2	100	100.00	-	90	78.33	9.13	100	97.73	1.41	100	81.97	1.73	576	64.40	1.07	100	89.60	12.32	100	100.00	-
scm1d	80	80.00	-	70	49.17	8.10	100	88.30	2.63	100	51.53	1.78	280	20.00	-	100	59.30	15.74	50	39.97	0.18

Table 5: Training times on benchmark datasets

Dataset	ELM		EM-ELM		CP-ELM		DP-ELM		PCA-ELM		EI-ELM		PCA-AIL	
	Avg.	SD	Avg.	SD	Avg.	SD	Avg.	SD	Avg.	SD	Avg.	SD	Avg.	SD
andro	0.0027	0.0024	0.0003	0.0001	0.0194	0.0417	0.0017	0.0024	0.0012	0.0005	0.0020	0.0007	0.0013	0.0002
edm	0.0009	0.0001	0.0009	0.0002	0.0116	0.0074	0.0142	0.0035	0.0007	0.0005	0.0160	0.0082	0.0021	0.0007
slump	0.0051	0.0017	0.0006	0.0002	0.0136	0.0183	0.0042	0.0022	0.0005	0.0003	0.0127	0.0070	0.0013	0.0006
enb	0.0019	0.0009	0.0042	0.0015	0.6962	0.0962	1.9796	0.2928	0.0005	0.0003	0.0365	0.0100	0.0021	0.0003
jura	0.0016	0.0002	0.0031	0.0010	0.0675	0.0120	0.0315	0.0075	0.0007	0.0004	0.0137	0.0031	0.0016	0.0004
sf1	0.0009	0.0001	0.0021	0.0014	0.0107	0.0015	0.0028	0.0005	0.0010	0.0004	0.0065	0.0055	0.0012	0.0003
sf2	0.0034	0.0003	0.0031	0.0021	0.0122	0.0017	0.0044	0.0005	0.0011	0.0005	0.0223	0.0230	0.0020	0.0005
wq	0.0050	0.0006	0.0066	0.0036	0.0501	0.0070	0.0134	0.0013	0.0010	0.0004	0.0701	0.0344	0.0039	0.0008
atp1d	0.0042	0.0014	0.0010	0.0007	0.0390	0.0028	0.0640	0.0114	0.0330	0.0168	0.0064	0.0031	0.0282	0.0046
atp7d	0.0055	0.0028	0.0008	0.0002	0.0418	0.0074	0.0358	0.0065	0.0300	0.0142	0.0050	0.0019	0.0330	0.0073
oes10	0.0025	0.0005	0.0007	0.0010	0.0149	0.0040	0.0056	0.0015	0.0188	0.0101	0.0036	0.0011	0.0232	0.0023
oes97	0.0023	0.0013	0.0004	0.0005	0.0187	0.0150	0.0044	0.0008	0.0181	0.0257	0.0028	0.0007	0.0162	0.0008
rf1	0.0157	0.0040	0.3553	0.0216	1.4046	0.1105	3.0543	0.2777	0.0055	0.0010	1.5641	0.4036	0.0624	0.0098
scm20d	0.0124	0.0052	0.3384	0.0082	2.0416	0.0169	5.6079	0.1621	0.0058	0.0019	2.9929	0.7132	0.0194	0.0020
rf2	0.0920	0.0374	0.3400	0.0184	1.3022	0.0557	1.9622	0.1258	0.1006	0.0072	5.1056	0.7497	0.4245	0.0300
scm1d	0.0226	0.0026	0.4100	0.0126	1.8946	0.0868	2.7918	0.0565	0.0324	0.0023	4.4223	1.2027	0.1232	0.0277

Table 6. Testing times on benchmark datasets

Dataset	ELM		EM-ELM		CP-ELM		DP-ELM		PCA-ELM		EI-ELM		PCA-AIL	
	Avg. (10^{-4})	SD (10^{-4})	Avg. (10^{-4})	SD (10^{-4})	Avg. (10^{-4})	SD (10^{-4})	Avg. (10^{-4})	SD (10^{-4})	Avg. (10^{-4})	SD (10^{-4})	Avg. (10^{-4})	SD (10^{-4})	Avg. (10^{-4})	SD (10^{-4})
andro	0.67	0.12	0.54	0.16	1.62	3.66	0.40	0.15	0.17	0.06	0.48	0.14	0.25	0.03
edm	0.73	0.11	0.97	0.31	1.85	0.48	2.14	0.64	0.34	0.24	2.70	0.85	0.58	0.22
slump	1.51	0.64	1.07	0.33	1.30	0.24	1.53	0.87	0.25	0.10	2.30	0.88	0.28	0.12
enb	1.88	0.39	3.35	1.66	11.30	4.50	9.39	7.63	0.53	0.24	11.00	2.01	0.99	0.26
jura	1.61	0.18	1.59	0.62	5.82	3.01	4.80	2.78	0.57	0.49	3.19	0.46	0.65	0.16
sf1	0.89	0.09	2.59	2.10	2.89	0.57	2.45	0.26	0.67	0.33	2.98	0.99	0.79	0.23
sf2	4.10	0.64	2.53	0.38	4.60	0.93	4.07	0.40	1.11	0.48	11.60	6.12	2.41	0.57
wq	8.74	1.45	2.77	1.07	12.90	2.32	9.75	1.01	1.44	0.67	4.99	2.05	2.32	0.40
atp1d	7.23	3.14	8.75	1.41	13.60	2.57	15.90	2.40	2.28	1.51	11.70	2.90	5.90	3.85
atp7d	10.70	6.62	7.15	1.23	15.50	2.98	17.20	4.51	1.40	0.59	10.20	3.65	6.71	0.72
oes10	3.82	0.86	12.10	3.16	6.32	2.06	7.02	4.83	1.02	0.29	9.70	1.48	3.87	1.63
oes97	3.47	2.00	3.94	0.41	5.97	3.47	5.54	2.93	0.90	0.34	6.14	1.36	2.07	0.23
rf1	46.10	14.40	31.30	7.08	160.00	33.20	138.00	30.90	7.71	1.04	36.10	5.50	32.10	4.35
scm20d	39.20	15.80	40.40	3.28	142.00	9.96	111.00	7.33	8.88	3.64	41.60	5.84	13.60	2.25
rf2	375.00	156.00	120.00	12.60	412.00	21.80	380.00	16.40	40.50	7.32	123.00	12.90	131.00	10.90
scm1d	87.70	10.80	73.90	5.84	293.00	20.60	229.00	10.20	24.80	3.18	94.70	14.80	73.80	9.59