Received December 14, 2019, accepted December 25, 2019, date of publication January 23, 2020, date of current version January 31, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2968983

Local Sigmoid Method: Non-Iterative Deterministic Learning Algorithm for Automatic Model Construction of Neural Network

SYUKRON ABU ISHAQ ALFAROZI[1], (Member, IEEE), KITSUCHART PASUPA[1], (Senior Member, IEEE), MASANORI SUGIMOTO[2], (Member, IEEE), AND KUNTPONG WORARATPANYA[1], (Member, IEEE)

[1]Faculty of Information Technology, King Mongkut's Institute of Technology Ladkrabang, Bangkok 10520, Thailand
[2]Graduate School of Information Science and Technology, Hokkaido University, Sapporo 060-0814, Japan

Corresponding author: Kuntpong Woraratpanya (kuntpong@it.kmitl.ac.th)

This work was supported in part by the King Mongkut's Institute of Technology Ladkrabang (KMITL), Thailand, and in part by the ASEAN University Network/Southeast Asia Engineering Education Development Network (AUN/SEED-Net) under the Japan International Cooperation Agency (JICA), Japan.

ABSTRACT A non-iterative learning algorithm for artificial neural networks is an alternative to optimize the neural network parameters with extremely fast convergence time. Extreme learning machine (ELM) is one of the fastest learning algorithms based on a non-iterative method for a single hidden layer feedforward neural network (SLFN) model. ELM uses a randomization technique that requires a large number of hidden nodes to achieve the high accuracy. This leads to a large and complex model, which is slow at the inference time. Previously, we reported analytical incremental learning (AIL) algorithm, which is a compact model and a non-iterative deterministic learning algorithm, to be used as an alternative. However, AIL cannot grow its set of hidden nodes, due to the node saturation problem. Here, we describe a local sigmoid method (LSM) that is also a sufficiently compact model and a non-iterative deterministic learning algorithm to overcome both the ELM randomization and AIL node saturation problems. The LSM algorithm is based on "divide and conquer" method that divides the dataset into several subsets which are easier to optimize separately. Each subset can be associated with a local segment represented as a hidden node that preserves local information of the subset. This technique helps us to understand the function of each hidden node of the network built. Moreover, we can use such a technique to explain the function of hidden nodes learned by backpropagation, the iterative algorithm. Based on our experimental results, LSM is more accurate than other non-iterative learning algorithms and one of the most compact models.

INDEX TERMS Neural network, compact model, hidden node interpretation, sigmoid function, function approximation, slope information.

I. INTRODUCTION

Machine learning has become an active research area in recent years, especially for neural network (NN) models: a widely used algorithm to train the neural network models has been backpropagation (BP) [1]. The BP algorithm uses local gradient parameters to iteratively minimize the loss function, until those parameters converge to the optimal point solution. These iterative algorithms can be very slow to converge and sometime difficult to choose an appropriate learning rate. Thus, several works have tried to improve the convergence time using conjugate gradient and second order methods [2], [3], such as the Levenberg-Marquardt (LM) algorithm [4] that usually requires a large memory. Recently, several implementations of gradient descent algorithms have improved the convergence time, by using an

The associate editor coordinating the review of this manuscript and approving it for publication was Derek Abbott.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see http://creativecommons.org/licenses/by/4.0/

adaptive learning rate method, such as AdaDelta [5], AdaGrad [6] and Adam [7]. However, those algorithms follow the iterative style of BP and are inherently slow. Another problem with BP arises in weight initiation: in improper weight initiation of a non-convex model, such as a single hidden layer architecture, BP does not always converge to the optimum point [8].

A common neural network model is a feedforward neural network (FNN). Specifically, one type of FNN is a single-hidden layer feedforward neural network (SLFN), which can be used as a universal approximator for an arbitrary function at any desired accuracy level: this has been proved theoretically [9]–[11]. Although an SLFN has only a single layer, it is still hard to optimize the model parameters because the search space has a complex shape. Adding a constraint can make an SLFN easier to optimize; that is, the input weight parameters are assigned randomly in some ranges, as in the extreme learning machine (ELM) [12]. Consequently, the ELM objective function depends on the output weight parameters only and optimization becomes the least square (LS) problem, and so a single global optimal solution will be found, subject to the current set of random constrained input weights. With the constraint, ELM can be extremely faster than BP, in a single non-iterative stage. In other words, ELM randomly maps the input features into a set of hidden features with random input weights. Thus, ELM is still one kind of neural network with random weights as comprehensively explained in [13].

Nevertheless, ELM can be unstable [14], [15], especially if the number of hidden nodes is small. Conversely, if the number of hidden nodes is large, we may find a subset of the optimal features in a large number of random features. Hence, the number of hidden nodes for ELM is commonly larger than that for BP. In the training phase, ELM is much faster than BP because the optimization can be computed in a single stage, without iterations, but the large number of hidden nodes leads to the more complex model and heavy computation load in the testing phase.

Inspired by the randomization of ELM, there were many improved ELMs that try to optimize the original ELM. Huang *et al.* further improved their ELM, with an incremental ELM (I-ELM) [16] and enhanced incremental ELM (EI-ELM) [17]. I-ELM added new random hidden nodes, one-by-one, by projecting the error vector into the new additional hidden node vector to reduce the error norm. However, I-ELM reduced the error norm only based on the new additional hidden node, regardless of existing hidden nodes, thus the final model leads to a non-LS solution. In other words, with the same random set of hidden nodes in ELM and I-ELM, ELM will be better than I-ELM, because ELM has an LS solution. I-ELM is generally faster for a model with a small number of hidden nodes, because the projection method is simpler than the inversion one. However, if the number of hidden nodes is very large, I-ELM has an overhead of residual error calculation that needs to be calculated on each iteration, while ELM does not have. EI-ELM improved on I-ELM, by using a random search that chooses the best nodes from a set of random nodes on each incremental process. All these methods (ELM, I-ELM and EI-ELM) require a large number of hidden nodes because they are based on a random mapping. In addition, Yang *et al.* [18] developed semi deterministic ELM algorithm called bidirectional extreme learning machine (B-ELM) that generates the odd hidden nodes using randomization and the even hidden nodes using a deterministic formula, while the output weights is calculated using the projection method, similar to I-ELM. Further, Cao *et al.* [19] improved the B-ELM on the odd hidden nodes using the random search, similar to EI-ELM, called enhanced bidirectional ELM (EB-ELM). However, I-ELM, EI-ELM, B-ELM and EB-ELM are based on projection method that is not optimal (not a least squares solution).

Another way for improving ELM is constructing the optimal structure from a set of random hidden node features, such as pruned ELM (P-ELM) [20] and optimally pruned ELM (OP-ELM) [21]. P-ELM and OP-ELM remove redundant nodes from hidden node features. However, both P-ELM and OP-ELM need to update their output weights, thus they require more computation time, especially with large structures of hidden node features. Similarly, a pruned sparse extreme learning machine (PS-ELM) algorithm [22] tried to remove the insignificant hidden node based on sparse coding. PS-ELM used gradient projection (GP) algorithm which is basically an iterative algorithm. Wang *et al.* [14] described two methods: constructive parsimonious ELM (CP-ELM) and destructive parsimonious ELM (DP-ELM), that use a recursive orthogonal least square (ROLS) technique [23] to prune faster. The constructive algorithm (CP-ELM) incrementally adds the hidden nodes from a set of random hidden node features, that have significant error reduction. On the other hand, the DP-ELM algorithm removes the hidden nodes, from a set of random hidden node features, that contribute less to error reduction. Moreover, the optimal input weight of the model might be or not in the set of random hidden features, due to the weight randomization. Thus, we cannot guarantee that the optimal input weights are in the random hidden features. Therefore, a key problem of ELM and its variants is the weight randomization that leads to a non-optimal solution.

To overcome the ELM randomization problem, Castaño *et al.* [24] described PCA-ELM, based on principal component analysis. Input weight parameters are generated from the subset of eigenvectors of an input feature covariance matrix, that explains the data with a chosen confidence level. However, the PCA-ELM does not reveal the relation between the input features and the prediction variables. In our previous work [25], we described analytical incremental learning (AIL), that adds a hidden node based on the current error vector. However, AIL cannot further grow the hidden nodes, because it is struggling to solve non-linearity of the error vector, thus leading to node saturation; that is, the newly generated hidden node is similar to the previous one.

Several authors have proved that it is possible to approximate a function, given some error criterion, using superposition of several sigmoids [26]–[29]. However, it is still not clear how to construct the appropriate model that automatically knows the suitable number of hidden nodes required for a particular dataset to provide an optimal solution. Therefore, this paper introduces a local sigmoid method (LSM) to overcome the non-linear structure of the error vector and automatically build its model. Here, the LSM generates the hidden nodes based on the characteristic of error vectors of a particular dataset, instead of choosing from a set of random hidden nodes as the ELM and its variants do. Furthermore, an LSM algorithm was derived from the characteristic of a sigmoid function used in the neural network. This characteristic makes it easy for us to understand the role of each node in constructing the prediction variable, using an interval segmentation method, which is especially effective for a non-linear function.

Mostly, a neural network model is able to solve many tasks with good performance. However, it is still a challenging task to understand how the model really works. Recently, many researchers have looked inside neural networks, aiming to explain what a neural network is really doing, with a variety of approaches, many focusing on deep convolutional neural network (CNN) [30]. Feature visualization [31]–[34] shows us how learned hidden neurons react to a particular input, e.g., show which of the hidden neurons for an image classification task are fired, when the input is a dog or a cat. Another approach is the attribution method, which searches for the relationships between neurons [35]–[38]. These knowledge extraction techniques try to find a high level representation, obtained from the learned model, that tells us about the semantic knowledge contributions from neurons, hidden layers and channel representations to an object or image. In this respect, we can conclude that some group of neurons are responsible for a particular input image pattern after the training, visualization and attribution processes.

The construction of our LSM is based on the compact representation approximation, described in Section II-D, thus, it enables us to understand how an LSM model actually works and builds its structure, instead of treating it as a black box. However, our work differs from the existing neural network interpretations, in that LSM gives us a low level (functional) representation; therefore, one can understand how a neural network builds the model instead of extracting the post knowledge after the training procedure. Furthermore, LSM avoids the randomization problem of ELM and the node saturation problem of AIL. As a non-iterative algorithm, then it would be faster than BP, an iterative approach.

The main contributions of our work are listed as follows.

- A novel non-iterative deterministic learning algorithm based on local sigmoid properties for SLFN is described. This algorithm called LSM is more accurate than other non-iterative learning baselines.
- An LSM algorithm can overcome the random feature mapping problem of ELM and the node saturation problem of AIL.
- An automated hidden layer node construction of LSM is based on interpretable hidden nodes using LSM node representation.
- An LSM node interpretation technique is applied on models learned by BP in order to know how each hidden node works. As shown in Section IV-A, both algorithms approximately form similar hidden nodes.
- A comprehensive benchmark comparison is done for both non-iterative algorithms, i.e., LSM, AIL, ELM, I-ELM, EI-ELM, EB-ELM, CP-ELM and DP-ELM, and an iterative algorithm, i.e., BP.

The rest of this paper is organized as follows. Section II analyzes the sigmoid function and the theoretical background for our approach—function approximation by using several representations of a sigmoid. Section III describes how the LSM algorithm approximates a function as a SLFN model. The experimental results and discussion are given in Section IV. Finally, we conclude our work in Section V.

II. THEORETICAL ANALYSIS OF SIGMOID FUNCTION

The purpose of this section is to describe a theoretical analysis of a sigmoid function and its properties. The sigmoid function is essential for developing the proposed LSM method; i.e., it is used for dividing a whole dataset into several subsets, and then solving each subset separately. The ability of several functions in the sigmoid family is explained for approximating any functions as well. One can skip this section, regardless of the detail explanations of these complementary theories and methods, and go directly to Section III to get the general idea of the LSM method.

A. THE SIGMOID FUNCTION

The sigmoid function has an 'S' shape and is a monotonic function, with the property that, if $\lim_{x\to-\infty}\sigma(x) = 0$ and $\lim_{x\to+\infty}\sigma(x) = 1$.

Two simple sigmoid functions: the cut and logistic functions were used for validating our method.

1) CUT FUNCTION

The cut function is defined by

$$\mathrm{cut}_{[a,b]}(x) = \begin{cases} 0, & \text{if} \quad x < a, \\ \dfrac{x-a}{b-a}, & \text{if} \quad a \le x \le b, \\ 1, & \text{if} \quad x > b. \end{cases} \tag{1}$$

It has a positive linear slope in $[a, b]$, otherwise, it has 0 gradient. Thus, we can save the slope of an input within the interval $[a, b]$ while ignoring it elsewhere.

2) LOGISTIC FUNCTION

The logistic function is defined by

$$\sigma_{[\gamma,c]}(x) = \frac{1}{1 + e^{-\gamma(x-c)}}, \tag{2}$$

where γ is the gradient multiplier and c is the center point of the slope of the function.

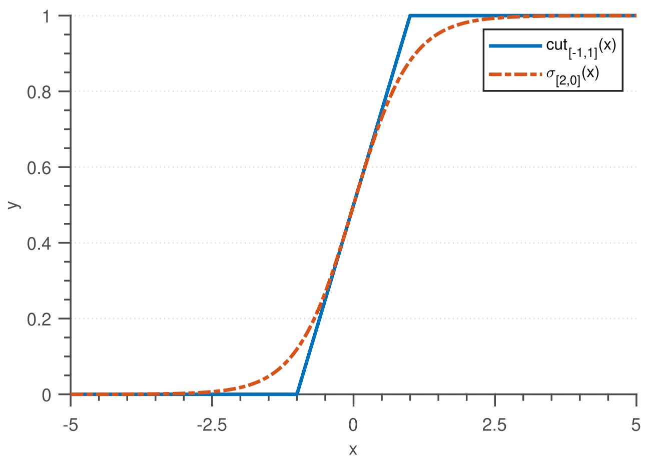


FIGURE 1. Cut and logistic functions.

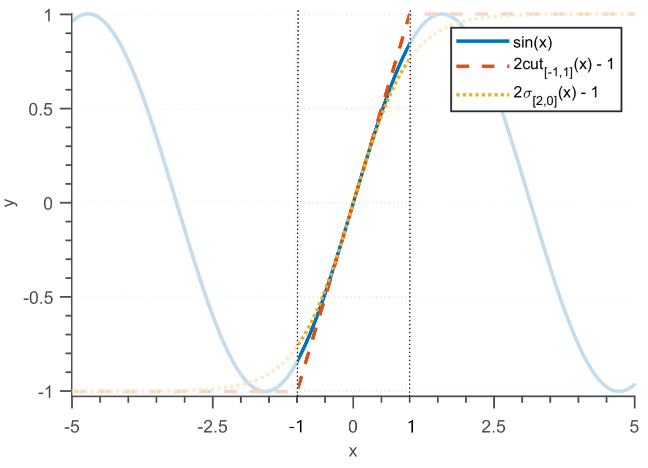


FIGURE 2. Cut and logistic function approximations to sin(x) in $[-1, 1]$, with $\alpha = 2$ and $\beta = -1$.

The hyperbolic tangent, $\tanh(x)$, may be defined in terms of $\sigma_{[\gamma,c]}(x)$:

$$\tanh(x) = 2\sigma_{[2,0]}(x) - 1, \tag{3}$$

$\tanh(x)$ is usually preferred to $\sigma(x)$ for the activation function in neural network hidden nodes, due to its output range of $(-1, 1)$, center at 0 and its odd property, i.e., $\tanh(-x) = -\tanh(x)$. As with cut, $\sigma(.)$ and $\tanh(.)$ can store a slope or gradient in its non-saturated interval. $\tanh(.)$ function is commonly used as activation function in neural network implementations because they are smooth and centered at 0 and have a finite differential everywhere. However, finding an appropriate non-saturated interval for the $\tanh(.)$ function is more difficult, but, as shown in the next section, the interval for the cut function is trivially set from the desired interval for approximating the target function and the parameters for a logistic function, which has a uniform error approximation to the cut function, has been proved [39], [40]. Fig. 1 shows $\text{cut}_{[-1,1]}(x)$ and logistic $\sigma_{[2,0]}(x)$ functions, centered at 0.

B. LOCALITY PROPERTY OF SIGMOID FUNCTIONS

In general, sigmoid functions are able to preserve slope information of a particular input, in a local interval, called the locality property. For instance, using the cut function, $\text{cut}_{[a,b]}(x)$, to preserve slope information of an arbitrary function in $[a, b]$, we can use $g(x)$:

$$g(x) = \beta + \alpha \, \text{cut}_{[a,b]}(x), \tag{4}$$

where α is the slope multiplier and β is the bias. We can find values of α and β to approximate a linear segment of a function in the interval of $[a, b]$. For example, Fig. 2 shows an approximation of sin(x) in $[-1, 1]$ using a cut function with $\alpha = 2$ and $\beta = -1$. Now, the logistic function, $\sigma(x)$, can be used to approximate the cut function, cut(x), in the same input interval, by choosing the appropriate values of γ and c: Iliev *et* al. have proved that $\sigma_{[\gamma,c]}(x)$, with $\gamma = 4/(b - a)$ and $c = (a + b)/2$, is the logistic function approximating $\text{cut}_{[a,b]}(x)$, with a uniform error, $\rho(\text{cut}, \sigma) = \frac{1}{1+e^2}$ [39], [40], where ρ is the Hausdorff distance (H-distance) [41], [42].

Therefore, to approximate the cut function, $\text{cut}_{[-1,1]}(x)$, we can use a logistic function $\sigma_{[\gamma,c]}(x)$ with $\gamma = 4/(b - a) = 2$ and $c = (a+b)/2 = 0$, i.e., $\sigma_{[2,0]}(x)$. Then, a relation between $\sigma_{[2,0]}(x)$, $\text{cut}_{[-1,1]}(x)$, and sin(x) can be written as $\sigma_{[2,0]}(x) \to \text{cut}_{[-1,1]}(x) \to \sin(x)$ in the interval of $[-1, 1]$ with $\alpha = 2$ and $\beta = -1$ as depicted in Fig. 2. Moreover, we can determine the best values of α and β for the cut function in (4), using a least square (LS) method. Further, to approximate another interval, e.g., $[a, b]$, we can use $\text{cut}_{[a,b]}(x)$. Note that a single cut function can be used to approximate a single linear segment. Multiple functions can be used to approximate a non-linear function as described in Section II-C.

However, in practice, neural networks use a single activation function, e.g., $\tanh(x)$ and $\sigma_{[2,0]}$. As previously described, the $\sigma_{[2,0]}(x)$ function approximates to $\text{cut}_{[-1,1]}(x)$. Hence, if we can transform any cut function from $\text{cut}_{[a,b]}(x)$ to $\text{cut}_{[-1,1]}(x)$, then we can approximate the cut function in any interval using a single log sigmoid function, i.e., $\sigma_{[2,0]}$ or $\tanh(x)$. For this reason, we propose a **Proposition 1** to transform any cut function in a particular interval to another interval.

Proposition 1: (An Interval Transformation for the Cut Function): A cut function $\text{cut}_{[a,b]}(x)$ *can be represented by another cut function in a different interval by applying a linear transformation to its input*:

$$\text{cut}_{[a,b]}(x) = \text{cut}_{[c,d]}(T_{[c,d]}(x)),$$

where

$$T_{[c,d]}(x) = \frac{(x - a)}{(b - a)}(d - c) + c$$

transforms the interval $[a, b]$ *to* $[c, d]$.

Proof: $T_{[c,d]}(x)$ is a linear function that transforms an interval of $[a, b]$ to $[c, d]$. It suffices to prove that

$$\text{cut}_{[a,b]}(a) = \text{cut}_{[c,d]}(T_{[c,d]}(a)) = \text{cut}_{[c,d]}(c) = 0,$$

and

$$\text{cut}_{[a,b]}(b) = \text{cut}_{[c,d]}(T_{[c,d]}(b)) = \text{cut}_{[c,d]}(d) = 1,$$

by substituting a and b to $T(x)$. This is follows from the definition of the cut function (1). □

As $\sigma_{[2,0]}(x)$ approximates $\text{cut}_{[-1,1]}(x)$, then, we define:

Definition 1: The non-saturated input interval of $\sigma_{[2,0]}(x)$ *or* $\tanh(x)$ *is* $[-1, 1]$.

Thus, if we want to use a single activation function, e.g., $\sigma_{[2,0]}(x)$ or $\tanh(x)$, then we need to transform the input interval to $[-1, 1]$ of the cut function that approximates a target function, using the **Proposition 1**.

C. FUNCTION APPROXIMATION USING LOCALITY PROPERTY

Many authors [9]–[11] have proved the ability of SLFN, as a universal approximation, that can approximate any arbitrary function. In addition, approximations using superposition of several functions, including the sigmoid function, have been proved [26]–[29]. Here, we prove that a superposition of cut functions can approximate an arbitrary function, within some error ϵ and some $n \in \mathbb{N}$, with a polyline approximation and its locality property in **Theorem 1**:

Theorem 1: There exists an $\epsilon > 0$ *and* $n \in \mathbb{N}$ *such that,*

$$\|f(x) - \text{G}_n(x)\| < \epsilon,$$

where $f(x)$ *is the target function and*

$$\text{G}_n(x) = \beta + \sum_{i=1}^{n} \alpha_i \, \text{cut}_{[a_i,a_{i+1}]}(x) \tag{5}$$

is the approximation of the target function $f(x)$ *in* $[a_1, a_{n+1}]$ *in which* $a_1 < a_2 < \cdots < a_{n+1}$.

Proof: Let us define a length of step, $\Delta x = a_{i+1} - a_i = (a_{n+1} - a_1)/n$, such that we have a uniform segment length for all $\text{cut}_{[a_i,a_{i+1}]}(x)$. From (5), we form a line approximation of $f(x)$, by setting up $\beta = f(a_1)$ and $\alpha_i = \Delta y_i = f(a_{i+1}) - f(a_i)$. Let us define the approximation in each local segment such that

$$g_i(x) = f(a_i) + \frac{\Delta y_i}{\Delta x_i}(x - a_i) = f(a_i) + \alpha_i \, \text{cut}_{[a_i,a_{i+1}]},$$

which is the Taylor series approximation of $f_i(x)$ in an interval of $[a_i, a_{i+1}]$ with an error

$$R_i(x) = \frac{f^{(2)}(z)}{2}(x - a_i)^2,$$

for $z \in [a_i, a_{i+1}]$ and $f(a_i) = g_{i-1}(a_i)$. If n is a large number, then $\Delta x \to 0$ that quadratically makes $(x - a_i)^2 \to 0$. Then, there exists

$$\|f_i(x) - g_i(x)\| = R_i(x) < \epsilon/n,$$

for all $i = 1, \ldots, n$, which implies that

$$\|f(x) - \text{G}_n(x)\| = \sum_{i=1}^{n} \|f_i(x) - g_i(x)\| < \epsilon.$$

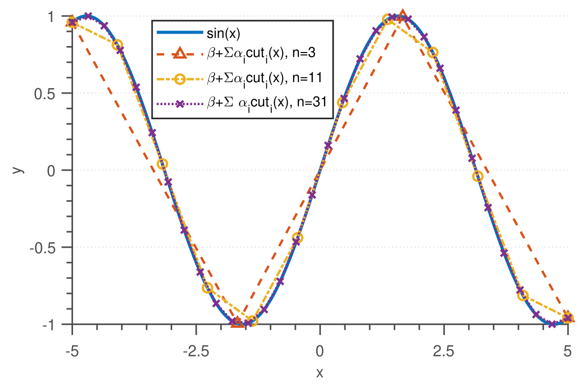


FIGURE 3. Approximations to $\sin(x)$, using a superposition of several sigmoid (cut) functions in $[-5, 5]$, with a differing number of segments: $n = \{3, 11, 31\}$.

This theorem can also be proved using the line integral theory of calculus. □

As shown earlier, we can generalize the proof by using the logistic function. Fig. 3 shows several approximations to the target function, $\sin(x)$, using polyline (5) with values of n in $[-5, 5]$. With $n = 31$, we have an excellent model for $\sin(x)$ in that interval. One line segment, $[a_i, a_{i+1}]$, is represented by $\text{cut}_{[a_i,a_{i+1}]}(x)$, i.e., the local representation of this segment. Clearly, larger n leads to a more accurate representation of $G_n(x)$, according to our **Theorem 1**. In addition, (5) is a linear system, so, we can further optimize $G_n(x)$ by changing β and α_i, calculated for segment, $[a_i, a_{i+1}]$, using a simple LS method. This will further reduce the error.

From the neural network point of view, (5) is an SLFN model with $\text{cut}_{[a,b]}(x)$ as the activation function and n nodes in its hidden layer. Hence, if n is large, the model will be large and space consuming. Here, the set of segment points, $\{a_i\}$, becomes an important parameter for model optimization. From the proof of **Theorem 1**, the fixed length of each segment is defined by $\Delta x = (a_{n+1} - a_1)/n$, which is inefficient. Therefore, an efficient method that keeps n small and provides an acceptable error will be introduced in the next subsection, which shows how to achieve a compact and efficient model.

D. COMPACT REPRESENTATION APPROXIMATION

In this section, we describe an efficient method for segmenting an input interval, into several line segments, that keeps n as small as possible and within acceptable error.

1) STATIONARY SEGMENT POINTS ($\nabla f(x) = 0$)

We show a simple way to segment the input interval into n segments, considering the error of the approximation function, $G_n(x)$. We choose the stationary points that satisfy:

$$\nabla f(x) = \lim_{\Delta x \to 0} \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} = 0. \tag{6}$$

Unfortunately, for a discrete set of points, such as random samples of $f(x)$ in some real dataset, we cannot use (6) due to the changing of Δx values. Hence, a turning point, i.e., a

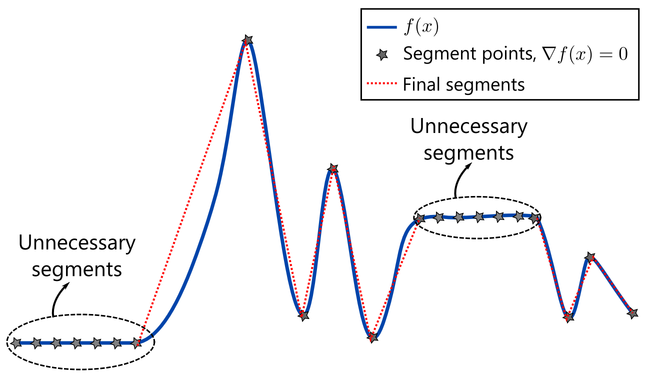


FIGURE 4. Interval segmentation with stationary segment points leading to many unnecessary segments.

point, where the gradient changes sign, is used as a segment point. A practical criterion for detecting the turning point, x_i, of a function is:

$$(f(x_i) - f(x_{i-1}))(f(x_{i+1}) - f(x_i)) \leq 0. \tag{7}$$

As a result, when $\sin(x)$ is segmented by (6) or (7) in $[-5, 5]$, we will have a set of segment points, $P = \{l, -3\pi/2, -\pi/2, \pi/2, 3\pi/2, h\}$, where $l = -5$ and $h = 5$ are the lowest and highest values of the interval. The number of segments with the set $P = \{a_1, \ldots, a_n\}$ is $|P| - 1$ and the set of segments is $S = \{[a_1, a_2], \ldots, [a_{n-1}, a_n]\}$.

2) UNNECESSARY SEGMENTS

In any set of segments, S, from points in P, not all segments will add significant information to our predictive model, G_n. In this case, the significant information is the gradient, Δy_i, for a particular segment. If $\Delta y_i \to 0$, then $\alpha_i \to 0$ and $f_i(x)$ has a small effect on G_n, according to (4). Then, we should remove this insignificant segment from S. For instance, for the function, $f(x)$, Fig. 4 shows the set of points, P, and segments, S, derived from (6). Two groups of segments have slope close to zero and are thus unnecessary segments, which can be removed from S so that the model is more compact and less complex. Therefore, we need to define a criterion for detecting unnecessary segments in S, based on the value of Δy_i:

$$|\Delta y_i| \geq \phi |\max_k f(a_k) - \min_k f(a_k)|, \tag{8}$$

where $0 \leq \phi \leq 1$ is the threshold that controls the contribution to G_n. For instance, $\phi = 0.05$ implies that we will ignore segments, with Δ_y less than 5% of the length of the range of $f(x)$.

We used **Algorithm 1** to obtain the local segment representation based on our analyses. The following algorithm, local segment representation (LSR), finds all potential segment points to build a segment set, S, and then removes the unnecessary segments based on the value of ϕ, using criterion (8). In the next section, we collect everything together to build a training algorithm, that works with the SLFN model.

Algorithm 1 Local Segment Representation (LSR)

```
Data: A function $f(x), f : \mathbb{R} \to \mathbb{R}$, for $x \in [l, h]$.
Result: A set of segments, $S$.
1 begin
2     set threshold $\phi$;
3     detect segment points, $P$, using (6) or (7);
4     create a segment set, $S$, based on $P$;
5     remove unnecessary segments in $S$, using (8);
6     return final segment set, $S$;
7 end
```

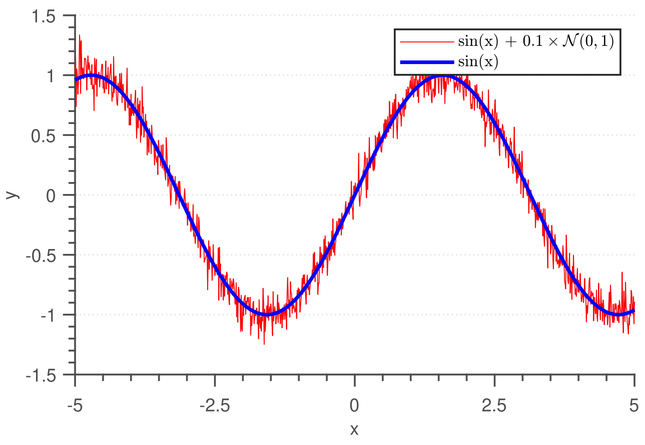


FIGURE 5. A noisy target function, which generates many local gradient sign changes.

III. LOCAL SIGMOID METHOD

In this section, we describe local sigmoid method (LSM) and its pseudocode, based on our previous analyses for an SLFN model.

A. PROBLEM OF LOCAL SEGMENT REPRESENTATION ALGORITHM

LSR, **Algorithm 1**, requires a sorted single-feature input, x, and a smooth target function, y, to divide the data into several segments based on the stationary segment points. However, if these requirements are not satisfied, then we face problems in running LSR. Here, we show how to overcome such problems, so that the algorithm can accept any type of data.

1) NOISY TARGET FUNCTION

In this subsection, we show how to handle noisy target functions. Let us be given N training examples, $D = \{(x_i, y_i)\}_{i=1}^{N}$, where $x_i \in \mathbb{R}$ and $y_i = f(x_i) \in \mathbb{R}$. If the target function, y, is noisy and not smooth, many points, which become segment points, are detected, and leads to many unneeded segments. For instance, in Fig. 5, we plot two $\sin(x)$ functions with and without the Gaussian noise. For the smooth target function, $\sin(x)$, we can easily apply **Algorithm 1**, because the segment points are nicely located at the maxima and minima of $\sin(x)$ function, using (6) or (7). However, for $\sin(x)$ with added

Gaussian noise, $\sin(x) + 0.1 \times \mathcal{N}(0,1)$, many inefficient segment points are detected.

To smooth the noisy target function, we can use a moving average (MA) low pass filter and recover the smoothed signal, $f_s(x)$, as follows:

$$f_s(x) = \text{smooth}_p(f_n(x)) = f_n(x) * h_p(x), \qquad (9)$$

where $h_p(x)$ is the moving average filter with a length of p, i.e., $h_p(x) = [1\ldots1]_{1\times p}$. Such a filter can be applied $q \in \mathbb{N}$ times to obtain a smoother curve. Then, we can find the segment set, S, using **Algorithm 1**.

2) MULTIPLE INPUT FEATURES

Mostly, we deal with multiple input feature datasets. Thus, to use LSR algorithm we need to transform the multiple features to a single feature so that we can sort the data before smoothing and segmenting the data. Let us be given N training examples $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, where $\mathbf{x}_i \in \mathbb{R}^m$ and $\hat{y}_i = f(\mathbf{x}_i) \in \mathbb{R}$. As **Algorithm 1**, LSR, works on a univariate function, thus we need to map $\mathbf{x}_i$ to a single feature component so that we can segment the interval based on the single feature $\hat{\mathbf{y}}$. There are many methods to solve this mapping function, such as averaging value, principal component analysis (PCA), least square (LS) method, etc. In our work, the LS method was selected to solve this problem, because it allows the mapping function not only to use the information of input feature but also the information of prediction variable, instead of using the input feature information only such as PCA or averaging value.

Let $\hat{y} = f_{\text{map}}(x) : \mathbb{R}^m \to \mathbb{R}$ be a linear mapping function defined by

$$f_{\text{map}}(\mathbf{x}_i) = \mathbf{x}_i^T \mathbf{w}_{\text{map}}, \qquad (10)$$

where $\mathbf{w}_{\text{map}}$ are the mapping weights. Thus, to approximate the target function, we need the weights, $\mathbf{w}_{\text{map}}$, which can be trivially found by solving:

$$\underset{\mathbf{w}_{\text{map}}}{\arg\min} \sum_i \|y_i - \mathbf{x}_i^T \mathbf{w}_{\text{map}}\|, \qquad (11)$$

using LS. Now, we have a single input feature after applying the mapping function (10), then sorting the dataset $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, based on values in $\mathbb{R}$ generated by $f_{\text{map}}(x)$. Finally, we can use **Algorithm 1**, LSR, to divide the dataset D into several subsets based on the $f_{\text{map}}(x)$ feature.

Fig. 6 shows how to segment the multiple input feature dataset, D, into several small subsets based on the interval segmentation. A segment set, S, divides the dataset D into subsets $D_1, \ldots, D_n$. As each segment s_i represents an almost linear segment, thus each D_i can have a small error by solving through LS method solely. This segmentation follows the "divide and conquer" strategy.

B. LSM ALGORITHM

After smoothing of input function and generating of single feature sorting index using mapping function $f_{\text{map}}(.)$, the LSR algorithm can be used. Now, we describe the deterministic

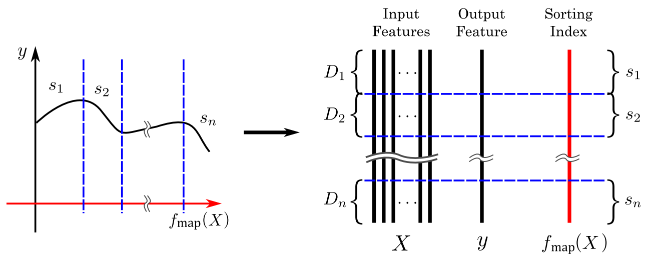


FIGURE 6. Segment s_i represents each subset D_i that can be linearly approximated based on the mapping feature $f_{\text{map}}(X)$.

LSM algorithm: being deterministic means that every time we run LSM, we will get the same result.

Suppose that we run **Algorithm 1** to obtain a segment set $S = \{s_1, \ldots, s_n\}$, where each s_i divides the dataset D into subsets $D_1, \ldots, D_n$, representing those segments. Then, the approximating function, $G_n(x)$, in (5) can be rewritten for a SLFN model:

$$G_n(\mathbf{x}) = \beta + \sum_{i=1}^{n} \alpha_i \tanh(L_i(\mathbf{x})), \qquad (12)$$

where

$$L_i(\mathbf{x}_j) = \mathbf{x}_j^T \mathbf{w}_i^{in} \qquad (13)$$

is the approximation of a local segment, s_i, and $\mathbf{w}_i^{in}$ is the i-th input weight of SLFN, $G_n(x)$, obtained from the LS optimization of a subset D_i:

$$\underset{\mathbf{w}_i^{in}}{\arg\min} \sum_j \|T_{[-1,1]}(y_j) - \mathbf{x}_j^T \mathbf{w}_i^{in}\|, \qquad (14)$$

where $T_{[-1,1]}(.)$ transforms $y_j \in D_i$ into $[-1, 1]$, such that $T_{[-1,1]}(\min(y_j)) = -1$ and $T_{[-1,1]}(\max(y_j)) = 1$. The y_j needs to be transformed to $[-1, 1]$, because the input of $\tanh(.)$, $\mathbf{x}_j^T \mathbf{w}_i^{in}$, lies in the non-saturated interval — see **Definition 1**. Subsequently, all 'ineffective' nodes with $\|\mathbf{w}_i^{in}\| < 10^{-3}$ are removed because they have a very small contribution to the approximating function, $G_n(x)$.

Lastly, β and α_i are determined by (12) — a linear problem that can be solved by LS as well. Here, to use multiple axes error reduction, the AIL recursive block inversion [25] based on Banachiewicz [43] and Petkovic and Stanimirovic [44] is adopted. Then, we can reform the SLFN in (12) by incrementally adding a hidden k-th node, so that

$$[\mathbf{1}_{N\times1} \quad \mathbf{H}_{N\times k}] \begin{bmatrix} \beta \\ \boldsymbol{\alpha} \end{bmatrix} = \mathbf{H}^{(k)} \boldsymbol{\beta}^{(k)} = \mathbf{y}, \qquad (15)$$

where $\mathbf{H}^{(k)} = [\mathbf{1}_{N\times1}, \tanh(L_1(\mathbf{X})), \ldots, \tanh(L_k(\mathbf{X}))]$ and $\boldsymbol{\alpha} = [\alpha_1, \ldots, \alpha_k]^T$. Using least squares, the optimum weight at the current iteration is

$$\begin{aligned} \boldsymbol{\beta}^{*(k)} &= (\mathbf{H}^{(k)^{\mathrm{T}}} \mathbf{H}^{(k)})^{-1} \mathbf{H}^{(k)^{\mathrm{T}}} \mathbf{y} \\ &= \mathbf{S}^{(k)^{-1}} \mathbf{u}^{(k)}, \end{aligned} \qquad (16)$$

where $\mathbf{S}^{(k)} = \mathbf{H}^{(k)^{\mathrm{T}}} \mathbf{H}^{(k)}$ and $\mathbf{u}^{(k)} = \mathbf{H}^{(k)^{\mathrm{T}}} \mathbf{y}$. Note that both $\mathbf{S}^{(k)}$ and its inverse $\mathbf{S}^{(k)^{-1}}$ are symmetric. Hence, the next optimum weight parameter $\boldsymbol{\beta}^{*(k+1)}$, with an additional hidden node vector $\mathbf{h}_{k+1}$, can be obtained recursively:

$$\begin{aligned}\boldsymbol{\beta}^{*(k+1)} &= \left(\begin{bmatrix}\mathbf{H}^{(k)^{\mathrm{T}}}\\ \mathbf{h}_{k+1}^{\mathrm{T}}\end{bmatrix}\begin{bmatrix}\mathbf{H}^{(k)} & \mathbf{h}_{k+1}\end{bmatrix}\right)^{-1}\begin{bmatrix}\mathbf{H}^{(k)^{\mathrm{T}}}\\ \mathbf{h}_{k+1}^{\mathrm{T}}\end{bmatrix}\mathbf{y}\\ &= \begin{bmatrix}\mathbf{H}^{(k)^{\mathrm{T}}}\mathbf{H}^{(k)} & \mathbf{H}^{(k)^{\mathrm{T}}}\mathbf{h}_{k+1}\\ \mathbf{h}_{k+1}^{\mathrm{T}}\mathbf{H}^{(k)} & \mathbf{h}_{k+1}^{\mathrm{T}}\mathbf{h}_{k+1}\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{H}^{(k)^{\mathrm{T}}}\mathbf{y}\\ \mathbf{h}_{k+1}^{\mathrm{T}}\mathbf{y}\end{bmatrix}\\ &= \begin{bmatrix}\mathbf{S}^{(k)} & \mathbf{H}^{(k)^{\mathrm{T}}}\mathbf{h}_{k+1}\\ \mathbf{h}_{k+1}^{\mathrm{T}}\mathbf{H}^{(k)} & \mathbf{h}_{k+1}^{\mathrm{T}}\mathbf{h}_{k+1}\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{u}^{(k)}\\ \mathbf{h}_{k+1}^{\mathrm{T}}\mathbf{y}\end{bmatrix}\\ &= \mathbf{S}^{(k+1)^{-1}}\mathbf{u}^{(k+1)}. \end{aligned} \tag{17}$$

Applying Banachiewicz block inversion technique [43] to the symetric matrix $\mathbf{S}^{(k+1)^{-1}}$ in (17), setting $\mathbf{v} = \mathbf{H}^{(k)^{\mathrm{T}}}\mathbf{h}_{k+1}$, $d = \mathbf{h}_{k+1}^{\mathrm{T}}\mathbf{h}_{k+1}$, $\theta = \mathbf{S}^{(k)^{-1}}\mathbf{v}$ and $\psi = d - \mathbf{v}^{\mathrm{T}}\theta$, then we have

$$\mathbf{S}^{(k+1)^{-1}} = \begin{bmatrix}\mathbf{S}^{(k)^{-1}} + \psi^{-1}\theta\theta^{\mathrm{T}} & -\psi^{-1}\theta\\ -\psi^{-1}\theta^{\mathrm{T}} & \psi^{-1}\end{bmatrix} \tag{18}$$

Here, ψ measures the linear dependence of a new node on existing nodes, if $\psi = 0$, then the new node $\mathbf{h}_{k+1}$ is linearly dependent on $\mathbf{H}^{(k)}$.

Note that, for each new node, we need to check the independence of the new node to the existing nodes, using ψ, and also the relative error of its residual. Therefore, the additional node can be omitted by setting a criterion for its independence, idp, from existing nodes, $0 < \eta \ll 1$,

$$\mathrm{idp} = \mathrm{abs}\left(\frac{\psi}{d}\right) \leq \eta \tag{19}$$

and also when the relative error of its residual is very small at iteration k

$$\mathrm{err} = \mathrm{abs}\left(\frac{\|\mathbf{e}^{(k+1)}\| - \|\mathbf{e}^{(k)}\|}{\|\mathbf{e}^{(k)}\|}\right) \leq \eta, \tag{20}$$

where $\mathbf{e}^{(k)} = \mathbf{y} - G_k(\mathbf{x})$. We can terminate the computation early, if there are no new nodes that satisfy (19) and (20). An efficient and fast way to calculate (20) is provided in our code [45], which uses a lower bound for how much of a new hidden node $\mathbf{h}_{k+1}$ can reduce the current error $\mathbf{e}^{(k)}$:

$$\mathrm{reduce}(\mathbf{h}_{k+1}) \geq \|\mathbf{e}^{(k)} - \frac{(\mathbf{e}^{(k)^T}\mathbf{h}_{k+1})}{\mathbf{h}_{k+1}^T\mathbf{h}_{k+1}}\mathbf{h}_{k+1}\| \tag{21}$$

to approximate $\mathbf{e}^{(k+1)}$ because LSM adds multiple hidden nodes in a single step.

Algorithm 2 shows the LSM algorithm in detail. This algorithm uses multiple iterations for multiple $f_{\mathrm{map}}^{(l)}(x)$ based on the residual error, $\mathbf{e}^{(l)} = \mathbf{y} - G_n^{(l)}(\mathbf{x})$ at iteration l. Thus, to obtain the $\mathbf{w}_{\mathrm{map}}^{(l)}$ of $f_{\mathrm{map}}^{(l)}(x)$, we just need to replace y_i with $e_i^{(l)}$ in (11) - see **Algorithm 2**, line 8.

Basically, LSM and AIL use the same incremental output weight calculation method. The main difference between LSM and AIL algorithms is of generating their hidden nodes as depicted in Fig. 7. LSM generates multiple hidden nodes based on interval segmentation on each iteration of the sorting index $f_{\mathrm{map}}^{(l)}(X)$ so that it can approximate function nonlinearity, while AIL generates a single hidden node using a pointwise inverse on the error vector of the activation function, $\tanh^{-1}(e_i^{(l)})$, so that it only approximates an almost linear segment for all data points, on each iteration. For the input weight calculation, AIL calculates it using an LS method based on $\tanh^{-1}(e_i^{(l)})$ without segmentation.

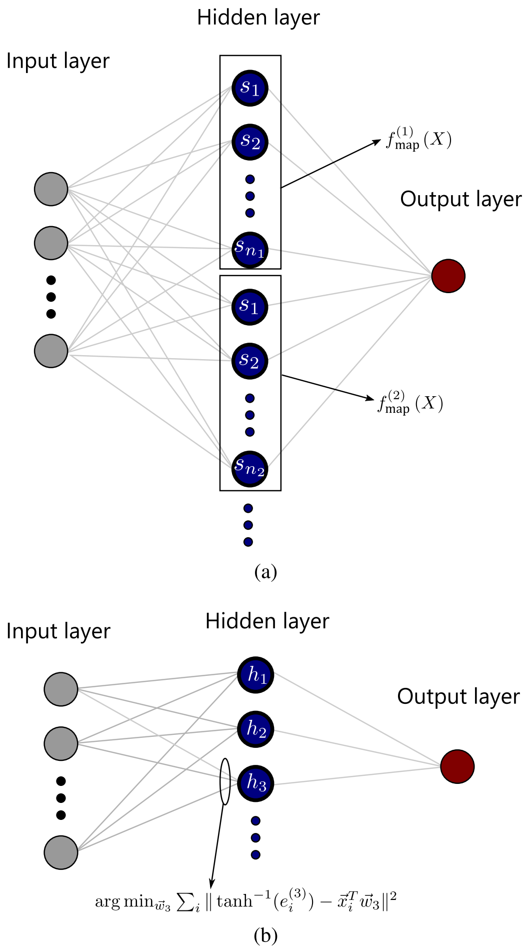

Hidden layer
Input layer
s_1
s_2
s_{n_1}
$f_{\mathrm{map}}^{(1)}(X)$
Output layer
s_1
s_2
s_{n_2}
$f_{\mathrm{map}}^{(2)}(X)$
(a)
Input layer
Hidden layer
h_1
h_2
h_3
Output layer
$\arg\min_{\vec{w}_3}\sum_i\|\tanh^{-1}(e_i^{(3)}) - \vec{x}_i^T\vec{w}_3\|^2$
(b)

FIGURE 7. Hidden layer construction of (a) LSM and (b) AIL. Note that the input weight of each segment s_i in LSM is calculated using (14).

IV. EXPERIMENTAL RESULTS AND DISCUSSION

Here, we compare our LSM algorithm with state of the art algorithms, both iterative and non-iterative variants. First, we analyzed approximating the sigmoid hidden node features of SLFN so that we can understand how the neural network trained by the BP algorithm works using LSM segmentation, instead of treating it as a black box. Second, we compare the performance of all test algorithms on real datasets either for regression or classification problems. Finally, we also explain the advantages and disadvantages of our LSM algorithm when compared to other test algorithms. We used MATLAB R2018b on a laptop equipped with an i7-7700HQ 2.80 GHz CPU. All code implementations of this paper is available in our repository [45].

Algorithm 2 Local Sigmoid Method (LSM)

```
Data: $N$ training examples $D = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, $\mathbf{x}_i \in \mathbb{R}^m$,
      $y_i = f(\mathbf{x}_i) \in \mathbb{R}$.
Result: SLFN model $G_n(x)$: $\mathbf{W} = \{\mathbf{w}_1^{in}, \ldots, \mathbf{w}_n^{in}\}$ and
        $\boldsymbol{\beta}^*$.
begin
    set minimum error $\eta$;
    set smoothing parameters, $p$ and $q$;
    set maximum number of iterations, $l$;
    initialize $\mathbf{e}^{(1)} = \mathbf{y}$;
    for $i \leftarrow 1$ to $l$ do
        map features to 1 dimensions (section III-A.2);
        map $\mathbf{x}$ using $f_{\text{map}}^{(i)}(\mathbf{x})$ based on (10) and $\mathbf{e}^{(i)}$;
        sort $D$ based on this new feature;
        smooth the target function $\mathbf{y}$ (section III-A.1);
        $\mathbf{f}_s \leftarrow \mathbf{y}$;
        for $j \leftarrow 1$ to $q$ do
            $\mathbf{f}_s = \text{smooth}_p(\mathbf{f}_s)$;
        end
        run LSR (Algorithm 1) using $\mathbf{f}_s$ and obtain
        segment set $S$ that divides $D$ into subsets,
        $\{D_1, \ldots, D_z\}$;
        calculate input weights $\mathbf{w}_k^{(in)}$ using (14) for each
        $D_k$ that $\|\mathbf{w}_k^{in}\| \geq 10^{-3}$ ;
        let $\mathbf{h}_k = \tanh(L_k(\mathbf{x})) \in \mathbf{H}$ from $\mathbf{w}_k^{in}$;
        sort (descending) $\mathbf{H} = \{\mathbf{h}_1, \ldots, \mathbf{h}_z\}$ based on
        (21) to current error $\mathbf{e}^{(i)}$;
        if $\forall \mathbf{h} \in \mathbf{H} |\, \text{reduce}(\mathbf{h}) < \eta$ then
            return; end the algorithm
        end
        foreach $\mathbf{h}_k \in \mathbf{H}$ do
            if reduce$(\mathbf{h}_k) \leq \eta$ then
                break;
            end
            if idp $\geq \eta$ then
                calculate $\boldsymbol{\beta}^*$ with $\mathbf{h}_k$ node using block
                inversion (17);
                add to set $\mathbf{W} \leftarrow \{\mathbf{W}\} + \{\mathbf{w}_k^{(in)}\}$;
            end
        end
        calculate $\mathbf{e}^{(i+1)}$;
    end
end
```

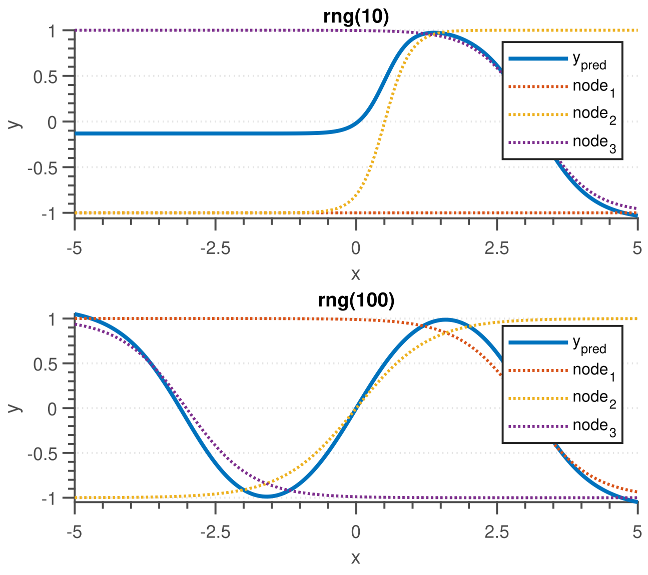


FIGURE 8. Segmentation by BP using LSM segmentation to predict sin(*x*) for *x* ∈ [−5, 5]. A poor weight initiation, based on the random number choice, led to bad segmentation (above) but a better choice led to good approximation (lower).

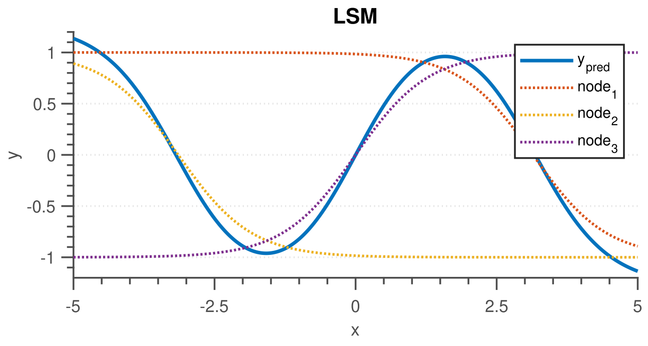


FIGURE 9. Segmentation of LSM and its prediction for sin(*x*) for *x* ∈ [−5, 5].

A. BP AND LSM SEGMENTS

Here, we compare how BP and LSM segment a function: we opened the neural network 'black box', using LSM segmentation based on a sigmoid hidden node tanh(.) activation function. For instance, we want to approximate $\sin(x)$ for $x \in [-5, 5]$, using LSM, this requires five segments based on points, $P = \{-5, -3\pi/2, -\pi/2, \pi/2, 3\pi/2, 5\}$. However, segments $[-5, -3\pi/2]$ and $[3\pi/2, 5]$ have very low slopes that will be removed automatically by (8); hence, three segments are sufficient. For simulation, 1,001 data points are generated using linespace function, i.e., $[-5 : 0.01 : 5]$.

According to the above data points and three suitable segments, SLFN is built with three hidden nodes and trained by a fast implementation of BP, i.e., Levenberg-Marquardt (LM) algorithm [4], [46], [47]. 100 epochs with 85% of training data and 15% of validation data are set and also the early stopping criteria are implemented. For the LSM algorithm, the threshold and smoothing parameter are set to $\phi = 0.05$ and $(p, q) = (0.02, 10)$, respectively, and then let the algorithm itself decide the required number of hidden nodes, i.e., the nodes after the training has completed. Finally, the segmentation of each hidden node is plotted by ignoring the output weights, so that it is pleasurable to compare.

Fig. 8 shows two BP segmentations resulting from different choices of the input random number: the plots show results from two seeds for the random number generator (`rng(seed)`). These results show that BP relies on the weight initiation to achieve the optimal solution.

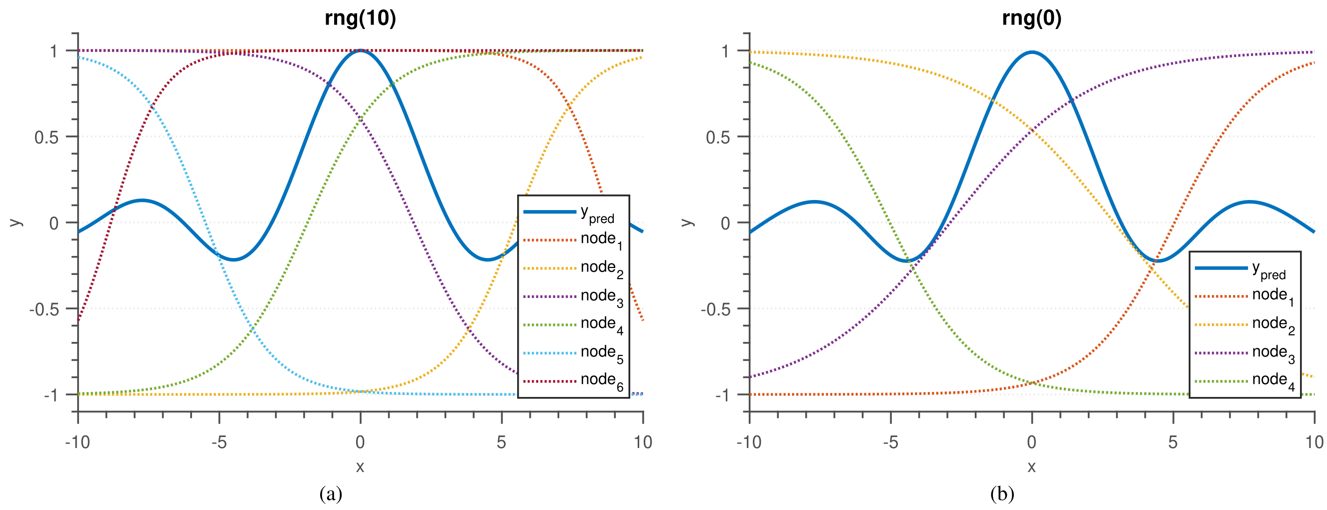


FIGURE 10. Segmentation for BP using LSM and its prediction for sinc(x) for $x \in [-10, 10]$ with proper weight initiations. (a) six hidden nodes approximation from LSM segmentation theory; (b) four hidden nodes approximation, a special case for overlapping segmentation.

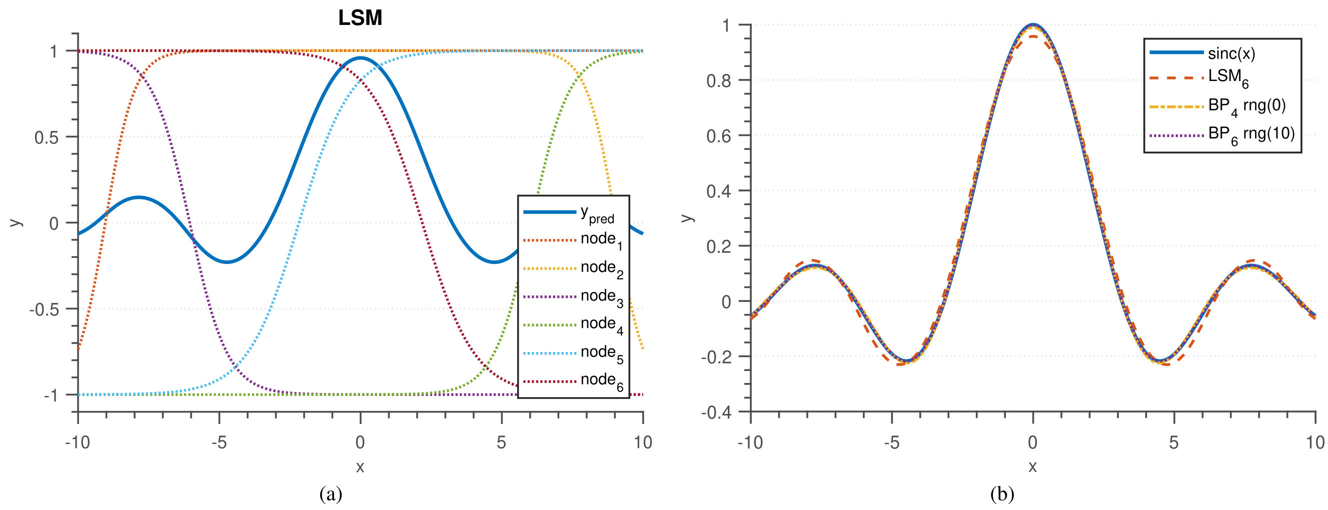


FIGURE 11. Segmentation for (a) LSM for sinc(x) for $x \in [-10, 10]$ and (b) comparison of prediction functions from BP and LSM.

The segmentation computed by each node, follows the LSR segmentation method (**Algorithm 1**), each node saves the information for a particular interval in its input weight, in this case, the input interval is divided into three segments. Fig. 9 shows the LSM algorithm on the same function. The LSM segmentation was similar to BP, but around the edges of interval, LSM was less accurate, i.e., $|y_{pred}| > 1$.

We also investigated the segmentation of BP and LSM for another function, i.e., sinc(x) = $\sin(x)/x$ for $x \in [-10, 10]$, with the same algorithm settings. For this simulation, 2,001 data points are generated using linespace function, i.e., $[-10 : 0.01 : 10]$. In this case, LSM predicted six segments; therefore, SLFN requires six hidden nodes. With an appropriate seed, the segmentation of BP algorithm also follows the LSM segmentation, i.e., one hidden node is responsible for one segment, as depicted in Fig. 10(a). One more thing that we did, we set the number of hidden nodes to four nodes instead of six with an appropriate seed. Surprisingly, after training, the approximation function still approximates well the sinc(x) function. It means that BP is able to choose overlapping segments needed to represent sinc(x). However, the segments obtained from BP are different from LSM. One hidden node can represent multiple LSM segments, together with other segments, as depicted in Fig. 10(b). Moreover, the LSM algorithm segments similar to BP, with six hidden nodes, as shown in Fig. 11(a) and the functions are compared in Fig. 11(b). Here, LSM is less accurate than BP, but it approximates the function adequately.

Furthermore, we ran the experiment on noisy data points of sin(x) and sinc(x). 5,000 random uniform data points are generated and added the Gaussian noise with $0.1 \times \mathcal{N}(0, 1)$ to each dataset. Fig. 12 shows the estimation of BP on noisy sin(x) for $x \in [-5, 5]$ with different `rng(.)`s that affect the quality of segmentation due to the different weight

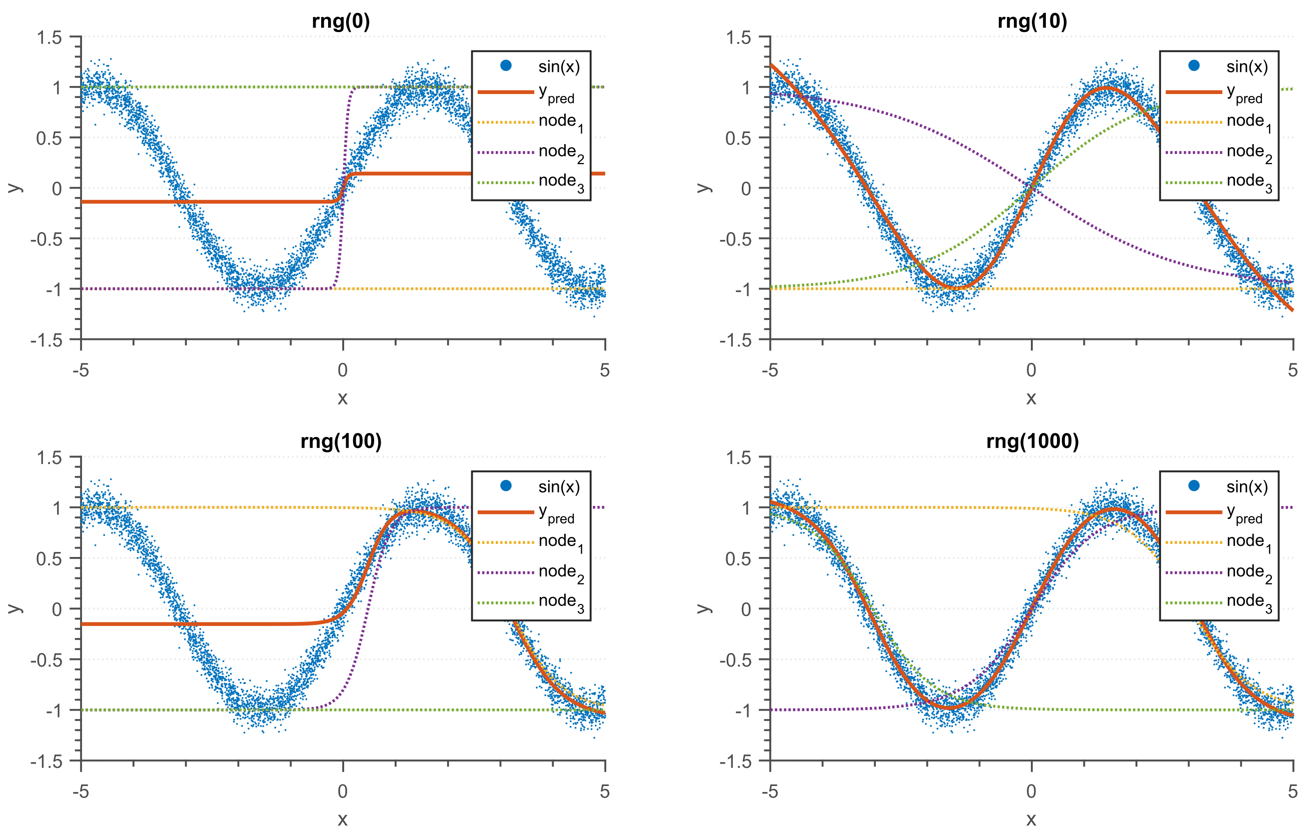


FIGURE 12. Segmentation by BP to predict noisy sin(x) for $x \in [-5, 5]$ with different `rng` (weight initiations).

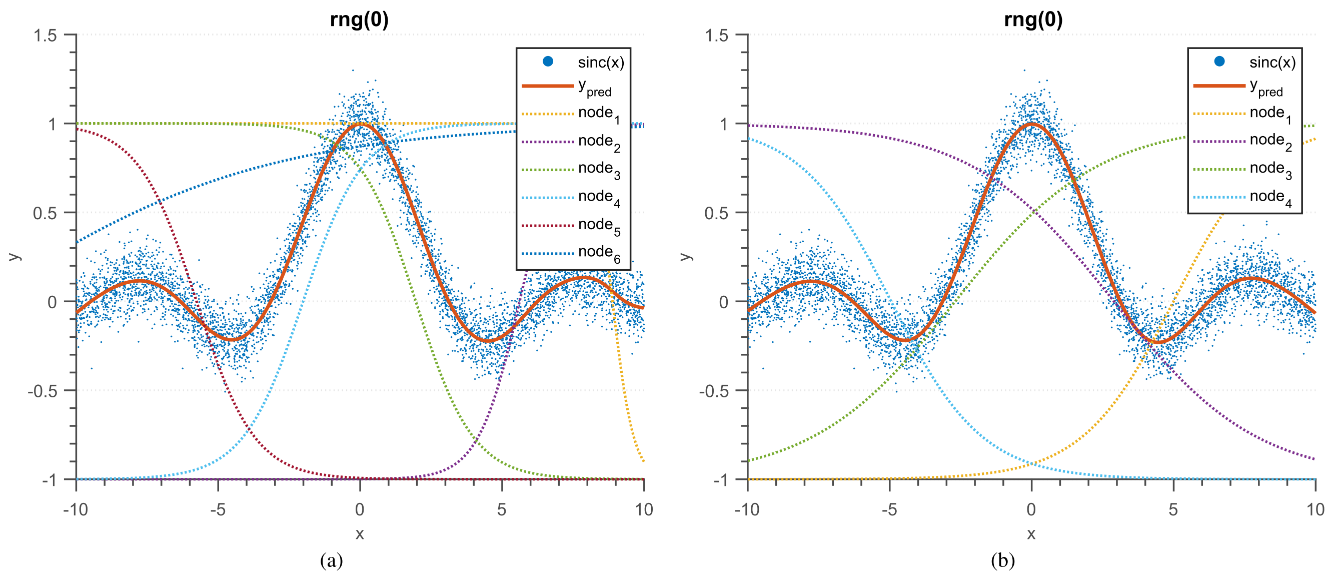


FIGURE 13. Segmentation by BP for noisy sinc(x) for $x \in [-10, 10]$ with proper weight initiations. (a) using six hidden nodes approximation; (b) using four hidden nodes approximation, a special case for overlapping segmentation.

initiations. For example, using `rng(0)` and `rng(100)`, BP was trapped in local minima. For `rng(10)`, BP basically required two overlapping segments to approximate the sin(x) in that interval because node$_1$ did not contribute to the approximation. However, in terms of fitting performance, `rng(1000)`, using three segments, was more accurate than `rng(10)`, using two segments. This showed that how BP performance relid on the weight initiation. In the same way, for sinc(x), BP showed the similar result of segmentation as in the case of non-noisy data as depicted in Fig. 13, as long

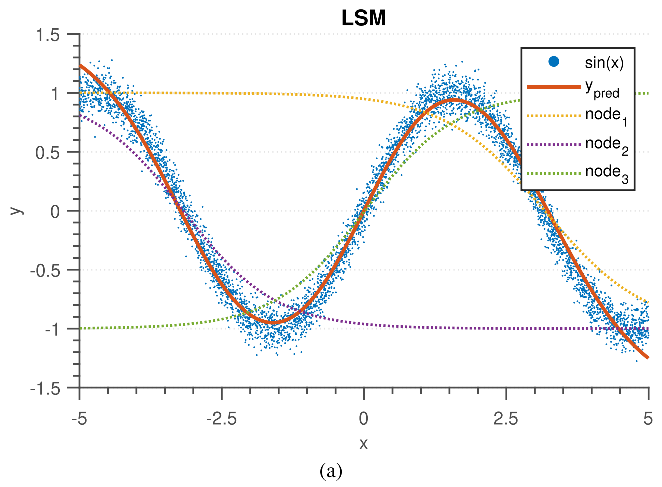


FIGURE 14. Segmentation of LSM and its prediction for both noisy sin(x) for $x \in [-5, 5]$ and sinc(x) for $x \in [-10, 10]$.

as we properly initialized the weights. Fig. 14 shows LSM approximation on these noisy data resulting similar segmentation as in the case of non-noisy data for both sin(x) and sinc(x), regardless of `rng(.)` (deterministic).

Therefore, it can be concluded that both algorithms have advantages and disadvantages. BP was generally more accurate than LSM, and also able to use overlapping segmentation. However, LSM algorithm was generally faster than BP as described in the next subsection, because the LSM algorithm was based on the LS method (non-iterative), and also was not needed to manually specify the number of hidden nodes. Moreover, the LSM is deterministic — it has a single solution, whereas BP may be trapped in local minima, for poor choices of the initial weights.

Furthermore, by using LSM segmentation for node representation, we can extract information of each hidden node dealing with the data for the SLFN structure. In a word, each hidden node represents each local segment by ignoring other information outside its non-saturated interval.

B. BENCHMARK DATASETS

In this subsection, we compared several algorithms, both iterative and non-iterative methods. Here, only the fast second-order BP algorithm, the LM algorithm, was selected as the iterative method. The non-iterative algorithms were LSM, AIL [25], ELM [12], I-ELM [16], EI-ELM [17], EB-ELM [19], PCA-ELM [24], CP- and DP-ELM [14]. Note that LSM, AIL and PCA-ELM are non-iterative and deterministic.

We compared these algorithms on both regression or classification problems. Single output benchmarks were selected, because the LSM method was designed for SLFN with a single output model. Note that one could treat the multiple output models as multiple single output models if desired.

We also used rank score as a metric for the performance comparison for those algorithms using linear scaling function on each dataset:

$$\text{rankscore}(x) = \left\lfloor 100 \times \frac{x - \min_d}{\max_d - \min_d} \right\rfloor, \qquad (22)$$

where $\min_d$ is the best performed algorithm that will score zero. On the other hand, $\max_d$ is the worst performed algorithm that will score 100. In a word, the lower score, the better performance.

For general parameter settings, the tanh(x) function is used as the activation function in all test algorithms and other parameters are described in Table 1. I-ELM, EI-ELM and EB-ELM used the stopping criteria $\eta = 10^{-4}$ which equals to RMSE = 0.01 which is basically larger than the stopping criteria defined in [16], [17], i.e., $\|E\| < 0.01$. The number of hidden nodes is set out in each section. For LSM, we do not need to specify the number of nodes, because it is specified

TABLE 1. General parameter settings.

Algorithms	Parameters	Types
LSM	Stopping criterion, $\eta = 10^{-4}$ LS regularization, $\lambda = 10^{-6}$ Smoothing parameters, $p = 0.02$ and $q = 10$ Thresholding parameter, $\phi = 0.05$ for regression and $\phi = 0.25$ for classification	Non-iterative
AIL	Stopping criterion, $\eta = 10^{-4}$ LS regularization, $\lambda = 10^{-6}$	Non-iterative
CP-ELM, DP-ELM	Stopping criterion, $\eta = 10^{-3}$ LS regularization, $\lambda = 10^{-6}$	Non-iterative
I-ELM, EI-ELM, EB-ELM	Stopping criterion, $\eta = 10^{-4}$ LS regularization, $\lambda = 10^{-6}$ Random search iteration, $k = 10$ for EI-ELM and EB-ELM	Non-iterative
PCA-ELM	Confidence level, $Cl = 0.95$	Non-iterative
BP	maximum epochs = 100 Train:val ratio, 85:15 Early stopping applied	Iterative

TABLE 2. Parameter settings - regression datasets.

Dataset	Split		Parameters								
			LSM	AIL	ELM	I-ELM	EI-ELM	EB-ELM	DP-ELM	CP-ELM	BP
	#training	#testing	l	maxNodes	#Nodes	maxNodes	maxNodes	maxNodes	maxNodes	maxNodes	#Nodes
Abalone	2002	2177	3	20	45	45	45	45	45	45	10
Delta ailerons	3000	4129	3	40	65	65	65	65	65	65	20
Auto MPG	198	200	1	20	30	30	30	30	30	30	10
Bank	4501	3692	10	40	200	200	200	200	200	200	20
Boston	250	256	3	10	35	35	35	35	35	35	5
California	8000	12640	10	20	105	105	105	105	105	105	10
Delta Elevators	4000	5517	2	10	80	80	80	80	80	80	5
Servo	80	87	2	20	20	20	20	20	20	20	10
Computer activity	4000	4192	11	90	125	125	125	125	125	125	45
Machine CPU	100	109	2	20	10	10	10	10	10	10	10
Triazines	100	86	1	10	25	25	25	25	25	25	5
Breast cancer	100	94	1	10	10	10	10	10	10	10	5

automatically, but, we need to specify the maximum number of iterations, l, — see Tables 2 and 6.

1) REGRESSION DATASETS

We used 12 datasets, collected by Torgo [48], from several repositories, which have been used for ELM [12] and CP- and DP-ELM [14] - see Table 2.

Here, one of the important parameter settings is the optimum number of hidden nodes, especially for ELM and its variants. So, we searched for the optimum number of hidden nodes by searching in steps of 5 up to 200, i.e., $N = \{5, 10, \ldots, 200\}$, and then chose the nearly optimal one, the same method as Huang *et* al. [12]. The maximum number of hidden nodes of I-ELM, EI-ELM, EB-ELM, CP- and DP-ELM were set to the number of ELM hidden nodes. However, PCA-ELM does not need any additional parameters, because the number of hidden nodes depends on the confidence levels in its eigenvectors. For AIL, the maximum number of hidden nodes was set to be twice the number of BP hidden nodes obtained from cross-validation method. Lastly, the maximum number of iterations, l, for LSM was obtained by cross-validation. Table 2 shows these parameter settings and the split between training and testing for each dataset. In addition, the input features and output features were normalized into $[-1, 1]$ and $[0, 1]$, respectively, the same setting as in [12].

We ran each algorithm 50 times, using a random split between the training and testing data — see Table 2. Tables 3, 4 and 5 show averages and standard deviations (σ) over 50 runs for RMSE, final number of hidden nodes and training time. The rank scores for each algorithm on each metric are shown in each table: boldface, bold-underline and bold-italic fonts denote the first, second and third best algorithms. BP showed the best RMSE on all datasets. LSM was generally second, followed by AIL, which performed fairly in the most cases — see Table 3. However, in terms of mean score, clearly, BP was the best algorithm that scored zero because it had the best performance on all datasets, followed by the LSM that scored 15.5, then DP-ELM that scored 28.4. Here, although AIL had better performance in the most cases than DP-ELM and ELM, it did not ensure that AIL had a lower mean scores than those algorithms. This is because if an algorithm had a large relative difference in a particular dataset, it would greatly affect the rank score. PCA-ELM had poor performance, because the number of hidden nodes depends on the number of its eigenvectors, which is commonly very small number on most datasets with a small number of input features. Note that PCA-ELM only generates its hidden nodes using eigenvectors, that cover 95% of input data, regardless of target feature information, but it still performed well, leading to the second best on the Triazines and Breast Cancer datasets. I-ELM also performed poorly due to its non-LS solution optimization. For instance, with the same number of hidden nodes and the same random input weights, ELM was better than I-ELM in training data, because ELM used an LS solution. Clearly, EI-ELM improved the performance of I-ELM, while EB-ELM, semi deterministic, further imrproved the performance of I-ELM and EI-ELM. When considering RMSE mean score, BP still preformed the best, followed by LSM, DP-ELM, ELM, CP-ELM, AIL and the other ELM variants.

Table 4 shows the final average number of hidden nodes. Based on the ranking, PCA-ELM was the most compact algorithm, with a mean score at 8.3, followed by AIL (11.2) and BP (19.1). However, PCA-ELM performed poorly as previously explained in Table 3. AIL cannot grow the number of hidden nodes, due to the node saturation. This motivated us to develop the LSM algorithm, that not only can add more hidden nodes, but also can perform more accurate prediction. As shown in Tables 3 and 4, LSM used a little bit more hidden node than AIL, but it performed more accurate. ELM and its variants need more hidden nodes when compared to LSM, AIL and BP for most datasets. Although I-ELM, EI-ELM and EB-ELM added the hidden node one-by-one incrementally until they reached the stopping criteria, the final number of hidden nodes were the same as those of ELM. This means that they did not achieve the stopping criteria. In balance, BP remains the best algorithm so far in terms of RMSE

TABLE 3. Testing RMSE and RMSE ranks for regression problems.

Dataset	Algorithms : $\mathrm{RMSE}^{(\mathrm{rankscore})}(\sigma)$									
	LSM	AIL	ELM	I-ELM	EI-ELM	EB-ELM	PCA-ELM	DP-ELM	CP-ELM	BP
Abalone	**0.0766**$^{(12)}$ (0.0013)	0.0784$^{(18)}$ (0.0012)	0.0775$^{(15)}$ (0.0022)	0.1022$^{(99)}$ (0.0097)	0.0846$^{(39)}$ (0.0031)	0.0806$^{(26)}$ (0.0026)	0.1025$^{(100)}$ (0.0019)	0.0778$^{(16)}$ (0.0018)	***0.0773***$^{(14)}$ (0.0016)	**0.0731**$^{(0)}$ (0.0014)
Delta ailerons	**0.0392**$^{(1)}$ (0.0005)	**0.0400**$^{(1)}$ (0.0005)	**0.0394**$^{(1)}$ (0.0006)	0.0608$^{(11)}$ (0.0088)	0.0448$^{(3)}$ (0.0038)	0.0444$^{(3)}$ (0.0032)	0.2542$^{(100)}$ (0.0027)	0.0415$^{(2)}$ (0.0011)	0.0406$^{(2)}$ (0.0008)	**0.0373**$^{(0)}$ (0.0007)
Auto MPG	**0.0818**$^{(20)}$ (0.0054)	0.0893$^{(28)}$ (0.0046)	***0.0838***$^{(22)}$ (0.0057)	0.1318$^{(72)}$ (0.0207)	0.1066$^{(46)}$ (0.0129)	0.1064$^{(46)}$ (0.0137)	0.1591$^{(100)}$ (0.0089)	0.0842$^{(23)}$ (0.0057)	***0.0836***$^{(22)}$ (0.0058)	**0.0623**$^{(0)}$ (0.0064)
Bank	**0.0414**$^{(3)}$ (0.0029)	***0.0424***$^{(4)}$ (0.0004)	0.0475$^{(7)}$ (0.0012)	0.0903$^{(32)}$ (0.0078)	0.0578$^{(13)}$ (0.0024)	0.0543$^{(11)}$ (0.0020)	0.2098$^{(100)}$ (0.0019)	0.0480$^{(7)}$ (0.0013)	0.0480$^{(7)}$ (0.0013)	**0.0353**$^{(0)}$ (0.0004)
Boston	**0.0965**$^{(27)}$ (0.0080)	***0.1124***$^{(43)}$ (0.0096)	0.1145$^{(45)}$ (0.0115)	0.1692$^{(100)}$ (0.0247)	0.1259$^{(56)}$ (0.0137)	0.1227$^{(53)}$ (0.0133)	0.1267$^{(57)}$ (0.0069)	0.1152$^{(46)}$ (0.0113)	0.1145$^{(45)}$ (0.0115)	**0.0699**$^{(0)}$ (0.0146)
California	**0.1305**$^{(12)}$ (0.0019)	0.1418$^{(21)}$ (0.0008)	***0.1311***$^{(13)}$ (0.0038)	0.1574$^{(34)}$ (0.0064)	0.1462$^{(25)}$ (0.0022)	0.1436$^{(23)}$ (0.0026)	0.2402$^{(100)}$ (0.0009)	***0.1308***$^{(13)}$ (0.0038)	***0.1314***$^{(13)}$ (0.0042)	**0.1151**$^{(0)}$ (0.0014)
Delta Elevators	**0.0535**$^{(0)}$ (0.0005)	**0.0536**$^{(0)}$ (0.0005)	**0.0535**$^{(0)}$ (0.0005)	0.0759$^{(9)}$ (0.0100)	0.0656$^{(5)}$ (0.0074)	0.0641$^{(4)}$ (0.0062)	0.3212$^{(100)}$ (0.0021)	0.0557$^{(1)}$ (0.0015)	0.0549$^{(1)}$ (0.0007)	**0.0525**$^{(0)}$ (0.0005)
Servo	**0.1104**$^{(29)}$ (0.0160)	0.1671$^{(68)}$ (0.0175)	0.2055$^{(94)}$ (0.0200)	0.2144$^{(100)}$ (0.0193)	0.1923$^{(85)}$ (0.0167)	***0.1645***$^{(66)}$ (0.0170)	0.2075$^{(95)}$ (0.0152)	0.2050$^{(94)}$ (0.0198)	0.2054$^{(94)}$ (0.0200)	**0.0688**$^{(0)}$ (0.0329)
Computer Activity	**0.0358**$^{(6)}$ (0.0022)	0.0712$^{(34)}$ (0.0079)	0.0615$^{(26)}$ (0.0350)	0.1291$^{(78)}$ (0.0241)	0.0857$^{(45)}$ (0.0073)	0.0865$^{(45)}$ (0.0093)	0.1573$^{(100)}$ (0.0024)	***0.0510***$^{(18)}$ (0.0195)	0.0621$^{(27)}$ (0.0398)	**0.0276**$^{(0)}$ (0.0007)
Machine CPU	**0.0621**$^{(19)}$ (0.0235)	***0.0662***$^{(25)}$ (0.0247)	0.0813$^{(45)}$ (0.0311)	0.1198$^{(97)}$ (0.0253)	0.0949$^{(64)}$ (0.0211)	0.0757$^{(38)}$ (0.0263)	0.1219$^{(100)}$ (0.0209)	0.0798$^{(43)}$ (0.0298)	0.0813$^{(45)}$ (0.0310)	**0.0478**$^{(0)}$ (0.0157)
Triazines	***0.1964***$^{(29)}$ (0.0156)	0.3587$^{(100)}$ (0.1756)	0.2320$^{(45)}$ (0.0449)	0.2961$^{(73)}$ (0.0732)	0.1975$^{(30)}$ (0.0269)	0.2348$^{(46)}$ (0.0427)	**0.1877**$^{(25)}$ (0.0164)	0.2332$^{(45)}$ (0.0478)	0.2319$^{(45)}$ (0.0450)	**0.1294**$^{(0)}$ (0.0360)
Breast cancer	***0.2718***$^{(28)}$ (0.0150)	0.3651$^{(100)}$ (0.1377)	0.2783$^{(33)}$ (0.0158)	0.2998$^{(49)}$ (0.0256)	0.2843$^{(37)}$ (0.0198)	0.2943$^{(45)}$ (0.0205)	**0.2661**$^{(23)}$ (0.0128)	0.2785$^{(33)}$ (0.0147)	0.2783$^{(33)}$ (0.0159)	**0.2362**$^{(0)}$ (0.0344)
Mean scores	**15.5**	36.8	28.8	62.8	37.3	33.8	83.3	***28.4***	29	**0**

Rank scores for each algorithm are indicated by superscript numbers in parentheses. Deviations, σ, are shown in parentheses under each RMSE.

TABLE 4. Final number of hidden nodes and node ranking for the regression problems.

Dataset	Algorithms : $\mathrm{nodes}^{(\mathrm{rankscore})}(\sigma)$									
	LSM	AIL	ELM	I-ELM	EI-ELM	EB-ELM	PCA-ELM	DP-ELM	CP-ELM	BP
Abalone	19.3$^{(37)}$ (4.9)	**4.3**$^{(1)}$ (1)	45$^{(100)}$ (0)	45$^{(100)}$ (0)	45$^{(100)}$ (0)	45$^{(100)}$ (0)	**4**$^{(0)}$ (0)	19.8$^{(39)}$ (2.1)	37$^{(80)}$ (3.8)	***10***$^{(15)}$ (0)
Delta ailerons	12.6$^{(14)}$ (6)	**4.8**$^{(1)}$ (0.5)	65$^{(100)}$ (0)	65$^{(100)}$ (0)	65$^{(100)}$ (0)	65$^{(100)}$ (0)	**4**$^{(0)}$ (0)	***7.5***$^{(6)}$ (1.9)	17.2$^{(22)}$ (4.2)	20$^{(26)}$ (0)
Auto MPG	12.1$^{(31)}$ (2.2)	**4.8**$^{(3)}$ (1.5)	30$^{(100)}$ (0)	30$^{(100)}$ (0)	30$^{(100)}$ (0)	30$^{(100)}$ (0)	**4**$^{(0)}$ (0)	17$^{(50)}$ (1.8)	27.7$^{(91)}$ (1.8)	***10***$^{(23)}$ (0)
Bank	**7.5**$^{(2)}$ (5.5)	**4**$^{(0)}$ (0.7)	200$^{(100)}$ (0)	200$^{(100)}$ (0)	200$^{(100)}$ (0)	200$^{(100)}$ (0)	**7**$^{(2)}$ (0)	88.5$^{(43)}$ (6)	181$^{(90)}$ (6.9)	20$^{(8)}$ (0)
Boston	35.4$^{(100)}$ (4.1)	***7.2***$^{(7)}$ (2.1)	35$^{(99)}$ (0)	35$^{(99)}$ (0)	35$^{(99)}$ (0)	35$^{(99)}$ (0)	**6.9**$^{(6)}$ (0.3)	22.5$^{(58)}$ (2.1)	33.9$^{(95)}$ (1.1)	**5**$^{(0)}$ (0)
California	16.3$^{(13)}$ (5.4)	**5**$^{(2)}$ (0.6)	105$^{(100)}$ (0)	105$^{(100)}$ (0)	105$^{(100)}$ (0)	105$^{(100)}$ (0)	**3**$^{(0)}$ (0)	55$^{(51)}$ (3.7)	95.9$^{(91)}$ (3.8)	***10***$^{(7)}$ (0)
Delta Elevators	6.4$^{(5)}$ (3.3)	**2.7**$^{(0)}$ (1)	80$^{(100)}$ (0)	80$^{(100)}$ (0)	80$^{(100)}$ (0)	80$^{(100)}$ (0)	***6***$^{(4)}$ (0)	8.4$^{(7)}$ (2.6)	27.4$^{(32)}$ (6.7)	**5**$^{(3)}$ (0)
Servo	***9.5***$^{(17)}$ (3.2)	**7.4**$^{(0)}$ (2.8)	20$^{(100)}$ (0)	20$^{(100)}$ (0)	20$^{(100)}$ (0)	20$^{(100)}$ (0)	**9.2**$^{(14)}$ (0.4)	15.8$^{(67)}$ (1)	19.7$^{(98)}$ (0.6)	10$^{(21)}$ (0)
Computer Activity	21.2$^{(16)}$ (5.3)	**8.5**$^{(5)}$ (2.4)	125$^{(100)}$ (0)	125$^{(100)}$ (0)	125$^{(100)}$ (0)	125$^{(100)}$ (0)	**2**$^{(0)}$ (0)	***19.8***$^{(14)}$ (2.9)	57.8$^{(45)}$ (8)	45$^{(35)}$ (0)
Machine CPU	10.8$^{(100)}$ (2.8)	**6.9**$^{(56)}$ (2.1)	10$^{(91)}$ (0)	10$^{(91)}$ (0)	10$^{(91)}$ (0)	10$^{(91)}$ (0)	**2**$^{(0)}$ (0.2)	***7***$^{(57)}$ (1.3)	9.6$^{(86)}$ (0.7)	10$^{(91)}$ (0)
Triazines	**7.6**$^{(13)}$ (1.7)	8.7$^{(19)}$ (1.8)	25$^{(100)}$ (0)	25$^{(100)}$ (0)	25$^{(100)}$ (0)	25$^{(100)}$ (0)	***8.4***$^{(17)}$ (0.6)	15.3$^{(52)}$ (1.9)	23.4$^{(92)}$ (1.4)	**5**$^{(0)}$ (0)
Breast cancer	9.6$^{(92)}$ (1.8)	**7**$^{(40)}$ (2.8)	10$^{(100)}$ (0)	10$^{(100)}$ (0)	10$^{(100)}$ (0)	10$^{(100)}$ (0)	7.8$^{(56)}$ (0.4)	***7.3***$^{(46)}$ (0.9)	9.9$^{(98)}$ (0.4)	**5**$^{(0)}$ (0)
Mean scores	36.7	**11.2**	99.2	99.2	99.2	99.2	**8.3**	40.8	76.7	***19.1***

and node numbers when compared to LSM algorithm and others.

Table 5 compares training times. ELM and its variants were faster than LSM and BP, except EB-ELM, which is almost the same as BP, because the half of hidden nodes are calculated using a formula, instead of randomization. ELM needed only one matrix inversion (recursive inversion and node selection methods for CP-ELM or DP-ELM). However, in terms of error, LSM and BP were much better than ELM and its variants. Now, let us focus on LSM, ELM and BP algorithms: for time ranking, ELM was the best: ELM (7.2) $<$ LSM (36.4) $<$ BP (64.1), but, for RMSE, BP was the best: BP (0)

TABLE 5. Training time and time ranks for the regression problems.

Dataset	Algorithms : $\mathrm{trtime}^{(\mathrm{rankscore})}(\sigma)$									
	LSM	AIL	ELM	I-ELM	EI-ELM	EB-ELM	PCA-ELM	DP-ELM	CP-ELM	BP
Abalone	0.1922$^{(39)}$ (0.0960)	***0.0375***$^{(4)}$ (0.0354)	0.0444$^{(5)}$ (0.0342)	**0.0269**$^{(1)}$ (0.0485)	0.1691$^{(34)}$ (0.0545)	0.4534$^{(100)}$ (0.1413)	**0.0219**$^{(0)}$ (0.0360)	0.0672$^{(10)}$ (0.0454)	0.1406$^{(28)}$ (0.0497)	0.2041$^{(42)}$ (0.2209)
Delta ailerons	0.1513$^{(24)}$ (0.0747)	***0.0406***$^{(6)}$ (0.0327)	***0.0431***$^{(6)}$ (0.0430)	**0.0328**$^{(4)}$ (0.0500)	0.3078$^{(51)}$ (0.0426)	0.5984$^{(100)}$ (0.0999)	**0.0081**$^{(0)}$ (0.0228)	0.1563$^{(25)}$ (0.0457)	0.2319$^{(38)}$ (0.0447)	0.2575$^{(42)}$ (0.0917)
Auto MPG	0.0322$^{(27)}$ (0.0265)	0.015$^{(12)}$ (0.0181)	***0.0119***$^{(10)}$ (0.0250)	**0.0006**$^{(0)}$ (0.0031)	**0.0041**$^{(3)}$ (0.0069)	0.0838$^{(71)}$ (0.0280)	***0.0119***$^{(10)}$ (0.0242)	0.0228$^{(19)}$ (0.0276)	0.0641$^{(54)}$ (0.0243)	0.1175$^{(100)}$ (0.0545)
Bank	***0.1544***$^{(2)}$ (0.1039)	**0.0350**$^{(0)}$ (0.0366)	***0.1303***$^{(2)}$ (0.0351)	***0.1509***$^{(2)}$ (0.0536)	1.5663$^{(26)}$ (0.0834)	2.3269$^{(38)}$ (0.2634)	**0.0134**$^{(0)}$ (0.0320)	6.0384$^{(100)}$ (1.5312)	3.0672$^{(51)}$ (0.1673)	0.4075$^{(7)}$ (0.1683)
Boston	0.1291$^{(100)}$ (0.0543)	0.0238$^{(18)}$ (0.0212)	***0.0075***$^{(6)}$ (0.0212)	**0.0003**$^{(0)}$ (0.0022)	**0.0059**$^{(4)}$ (0.0077)	0.0909$^{(70)}$ (0.0216)	0.0147$^{(11)}$ (0.0281)	0.0328$^{(25)}$ (0.0222)	0.0819$^{(63)}$ (0.0327)	0.0863$^{(67)}$ (0.0462)
California	0.4097$^{(30)}$ (0.1547)	**0.0459**$^{(3)}$ (0.0308)	***0.0981***$^{(7)}$ (0.0359)	0.1238$^{(8)}$ (0.0390)	1.2113$^{(91)}$ (0.0659)	1.3313$^{(100)}$ (0.1766)	**0.0119**$^{(0)}$ (0.0271)	0.9522$^{(71)}$ (0.1351)	0.9538$^{(71)}$ (0.4018)	0.3781$^{(28)}$ (0.2766)
Delta Elevators	0.1234$^{(17)}$ (0.0613)	**0.0241**$^{(2)}$ (0.0296)	***0.0384***$^{(5)}$ (0.0331)	0.0625$^{(8)}$ (0.0541)	0.4691$^{(69)}$ (0.0562)	0.6725$^{(100)}$ (0.2920)	**0.0081**$^{(0)}$ (0.0171)	0.3272$^{(48)}$ (0.0948)	0.0913$^{(13)}$ (0.0293)	0.1581$^{(23)}$ (0.0802)
Servo	0.0344$^{(20)}$ (0.0268)	0.0194$^{(11)}$ (0.0166)	0.0103$^{(6)}$ (0.0225)	**0.0003**$^{(0)}$ (0.0022)	**0.0019**$^{(1)}$ (0.0051)	0.0534$^{(32)}$ (0.0192)	0.0138$^{(8)}$ (0.025)	0.0069$^{(4)}$ (0.0123)	***0.0056***$^{(3)}$ (0.0076)	0.1672$^{(100)}$ (0.0473)
Computer Activity	0.3281$^{(12)}$ (0.1203)	**0.0728**$^{(2)}$ (0.0454)	**0.0738**$^{(2)}$ (0.0300)	0.0816$^{(3)}$ (0.0430)	0.7903$^{(30)}$ (0.0718)	1.1063$^{(42)}$ (0.2172)	**0.0094**$^{(0)}$ (0.0284)	2.6453$^{(100)}$ (1.0697)	0.2297$^{(8)}$ (0.0472)	1.5809$^{(60)}$ (1.3328)
Machine CPU	0.0469$^{(62)}$ (0.0352)	0.0138$^{(18)}$ (0.0153)	0.0122$^{(16)}$ (0.0208)	**<10**$^{-4\,(0)}$	**0.0003**$^{(0)}$ (0.0022)	0.0297$^{(39)}$ (0.0165)	0.0047$^{(6)}$ (0.0152)	0.0053$^{(7)}$ (0.0121)	***0.0016***$^{(2)}$ (0.0047)	0.0753$^{(100)}$ (0.0398)
Triazines	0.0444$^{(26)}$ (0.0416)	0.0225$^{(13)}$ (0.0217)	***0.0147***$^{(9)}$ (0.0269)	**<10**$^{-4\,(0)}$	**0.0028**$^{(2)}$ (0.0061)	0.0703$^{(42)}$ (0.0222)	0.0184$^{(11)}$ (0.0296)	0.0381$^{(23)}$ (0.0337)	***0.0159***$^{(9)}$ (0.0224)	0.1681$^{(100)}$ (0.0816)
Breast cancer	0.0613$^{(78)}$ (0.0364)	0.0188$^{(24)}$ (0.0214)	0.0094$^{(12)}$ (0.0238)	**0.0003**$^{(0)}$ (0.0022)	**0.0006**$^{(0)}$ (0.0031)	0.0269$^{(34)}$ (0.0134)	0.0125$^{(16)}$ (0.0277)	0.0138$^{(17)}$ (0.0301)	***0.0078***$^{(10)}$ (0.0226)	0.0784$^{(100)}$ (0.0290)
Mean scores	36.4	9.4	*7.2*	**2.2**	25.9	64	**5.2**	37.4	29.2	64.1

<10^{-4} : both mean and deviation are less than 10^{-4}.

TABLE 6. Parameter settings - classification datasets.

Dataset	Parameters								
	LSM	AIL	ELM	I-ELM	EI-ELM	EB-ELM	DP-ELM	CP-ELM	BP
	l	maxNodes	#Nodes	maxNodes	maxNodes	maxNodes	maxNodes	maxNodes	#Nodes
Banana	11	30	50	50	50	50	50	50	15
Breast cancer	1	90	20	20	20	20	20	20	45
Diabetis	1	90	30	30	30	30	30	30	45
Flare solar	3	20	10	10	10	10	10	10	10
German	4	150	40	40	40	40	40	40	75
Heart	2	20	30	30	30	30	30	30	10
Image	15	30	200	200	200	200	200	200	15
Ringnorm	15	100	40	40	40	40	40	40	50
Splice	7	160	165	165	165	165	165	165	80
Thyroid	6	50	35	35	35	35	35	35	25
Titanic	4	50	20	20	20	20	20	20	25
Twonorm	1	10	65	65	65	65	65	65	5
Waveform	5	20	85	85	85	85	85	85	10

$<$ LSM (15.5) $<$ ELM (28.8). As expected, there is a trade-off between the training time and the error — see Table 3. For the non-iterative algorithms, i.e., LSM, AIL and ELM and its variants, EB-ELM and LSM are the slowest. However, LSM performed better than others in terms of testing RMSE.

2) CLASSIFICATION DATASETS

For classification problems, the commonly used datasets in common areas, such as the kernel and boosting methods, were selected for performance evaluation. In this experiment, we used IDA benchmark [49], which contains 13 artificial and real-world datasets, used in several research papers [49]–[53], available in [54] and for MATLAB version in [55]. The IDA benchmark covers a variety of different datasets: from small to high expected error rates, from low- to high-dimensional data and from small to large sample sizes. There are 13 binary classification datasets, — see Table 6, which consists of 100 realizations of predefined splits into training and testing samples (20 in the case of the image and splice datasets). The input features were standardized from the source, the setting was used the same as in [49]–[53], and the class values were set to 1 (positive) and -1 (negative). Thus, if the model output value is larger than 0, then, this example is assigned as the positive class, otherwise, it is a negative class. For each dataset, we recored the average and the standard deviation for error rate, number of hidden nodes

TABLE 7. Testing error rate and error rate ranks for classification problems.

Dataset	Algorithms : error$^{(\text{rankscore})}$ (σ)									
	LSM	AIL	ELM	I-ELM	EI-ELM	EB-ELM	PCA-ELM	DP-ELM	CP-ELM	BP
Banana	0.1299$^{(6)}$ (0.0224)	0.3983$^{(79)}$ (0.0558)	**0.1080**$^{(1)}$ (0.0053)	0.3432$^{(64)}$ (0.0459)	0.2319$^{(34)}$ (0.0341)	0.253$^{(40)}$ (0.0341)	0.4738$^{(100)}$ (0.0353)	**0.1103**$^{(1)}$ (0.0052)	**0.1107**$^{(1)}$ (0.0055)	**0.1060**$^{(0)}$ (0.0050)
Breast cancer	**0.2765**$^{(77)}$ (0.0476)	**0.2761**$^{(77)}$ (0.0482)	0.2918$^{(86)}$ (0.0500)	0.2879$^{(84)}$ (0.0463)	0.2861$^{(83)}$ (0.0527)	0.2809$^{(80)}$ (0.0475)	0.3166$^{(100)}$ (0.0481)	0.2944$^{(88)}$ (0.0451)	0.2956$^{(88)}$ (0.0519)	**0.1386**$^{(0)}$ (0.0401)
Diabetis	***0.2378***$^{(64)}$ (0.0185)	**0.2363**$^{(62)}$ (0.0187)	0.2544$^{(88)}$ (0.0209)	0.2537$^{(87)}$ (0.0221)	0.2489$^{(80)}$ (0.0197)	0.2461$^{(76)}$ (0.0197)	0.2626$^{(100)}$ (0.0194)	0.2491$^{(81)}$ (0.0187)	0.2528$^{(86)}$ (0.0214)	**0.1933**$^{(0)}$ (0.0175)
Flare solar	0.3402$^{(17)}$ (0.0256)	**0.3379**$^{(15)}$ (0.0153)	0.3508$^{(29)}$ (0.0226)	0.3959$^{(77)}$ (0.0494)	**0.3382**$^{(15)}$ (0.0191)	0.3439$^{(21)}$ (0.0295)	0.4167$^{(100)}$ (0.0354)	0.3558$^{(34)}$ (0.0313)	0.3544$^{(33)}$ (0.0303)	**0.3244**$^{(0)}$ (0.0220)
German	**0.2432**$^{(64)}$ (0.0222)	**0.2432**$^{(64)}$ (0.0212)	0.2666$^{(75)}$ (0.0280)	0.2771$^{(80)}$ (0.0274)	0.2582$^{(71)}$ (0.0260)	0.2492$^{(67)}$ (0.0225)	0.3172$^{(100)}$ (0.0239)	0.2659$^{(75)}$ (0.0262)	0.2632$^{(74)}$ (0.0245)	**0.1126**$^{(0)}$ (0.0280)
Heart	0.1865$^{(84)}$ (0.0318)	***0.1781***$^{(76)}$ (0.0343)	0.1939$^{(91)}$ (0.0352)	0.2036$^{(100)}$ (0.0390)	0.1952$^{(92)}$ (0.0349)	0.1836$^{(81)}$ (0.0389)	**0.1594**$^{(58)}$ (0.0343)	0.1982$^{(95)}$ (0.0330)	0.2006$^{(97)}$ (0.0390)	**0.0979**$^{(0)}$ (0.0377)
Image	***0.062***$^{(17)}$ (0.0080)	0.1702$^{(56)}$ (0.0106)	***0.0622***$^{(17)}$ (0.0084)	0.1308$^{(41)}$ (0.0133)	0.0897$^{(27)}$ (0.0103)	0.1054$^{(32)}$ (0.0086)	0.2926$^{(100)}$ (0.0110)	**0.0588**$^{(15)}$ (0.0111)	***0.0627***$^{(17)}$ (0.0084)	**0.0164**$^{(0)}$ (0.0055)
Ringnorm	**0.0947**$^{(22)}$ (0.0100)	0.2487$^{(89)}$ (0.0133)	0.2693$^{(98)}$ (0.0086)	0.2745$^{(100)}$ (0.0107)	0.2629$^{(95)}$ (0.0057)	***0.1997***$^{(68)}$ (0.0119)	0.2467$^{(88)}$ (0.0053)	0.2695$^{(98)}$ (0.0084)	0.2691$^{(98)}$ (0.0078)	**0.0429**$^{(0)}$ (0.0105)
Splice	**0.1659**$^{(37)}$ (0.0087)	***0.1701***$^{(39)}$ (0.0068)	0.2487$^{(72)}$ (0.0166)	0.2607$^{(77)}$ (0.0142)	0.2034$^{(53)}$ (0.0083)	0.1729$^{(40)}$ (0.0114)	0.1931$^{(49)}$ (0.0079)	0.3153$^{(100)}$ (0.0687)	0.2513$^{(73)}$ (0.0135)	**0.0776**$^{(0)}$ (0.0487)
Thyroid	**0.0639**$^{(47)}$ (0.032)	0.1151$^{(92)}$ (0.0317)	0.0692$^{(52)}$ (0.0332)	0.1167$^{(94)}$ (0.0357)	0.0901$^{(70)}$ (0.0348)	0.0955$^{(75)}$ (0.0331)	0.1237$^{(100)}$ (0.0424)	0.0728$^{(55)}$ (0.0307)	***0.0672***$^{(50)}$ (0.0329)	**0.0101**$^{(0)}$ (0.0123)
Titanic	0.2328$^{(63)}$ (0.0174)	0.2287$^{(44)}$ (0.0126)	**0.2238**$^{(22)}$ (0.0100)	0.2288$^{(45)}$ (0.0073)	0.2292$^{(47)}$ (0.0113)	0.2285$^{(43)}$ (0.0066)	0.2411$^{(100)}$ (0.0193)	***0.2254***$^{(30)}$ (0.0133)	***0.2255***$^{(30)}$ (0.0132)	**0.2188**$^{(0)}$ (0.0089)
Twonorm	***0.0296***$^{(12)}$ (0.0031)	0.0381$^{(28)}$ (0.0077)	0.0427$^{(37)}$ (0.0044)	0.0753$^{(100)}$ (0.008)	0.0655$^{(81)}$ (0.0066)	0.0559$^{(63)}$ (0.0100)	**0.0233**$^{(0)}$ (0.0010)	0.0422$^{(36)}$ (0.0046)	0.0428$^{(38)}$ (0.0042)	**0.0292**$^{(11)}$ (0.0055)
Waveform	**0.1422**$^{(40)}$ (0.0095)	***0.1483***$^{(46)}$ (0.0131)	0.1628$^{(59)}$ (0.0097)	0.1806$^{(76)}$ (0.0118)	0.1581$^{(55)}$ (0.0091)	0.1583$^{(55)}$ (0.0105)	0.2057$^{(100)}$ (0.0039)	0.1619$^{(59)}$ (0.0083)	0.1618$^{(59)}$ (0.0089)	**0.0999**$^{(0)}$ (0.0117)
Mean scores	**42.3**	59	***55.9***	78.8	61.8	57	84.2	59	57.2	**0.8**

TABLE 8. Final number of hidden nodes and node ranking for the classification problems.

Dataset	Algorithms : nodes$^{(\text{rankscore})}$ (σ)									
	LSM	AIL	ELM	I-ELM	EI-ELM	EB-ELM	PCA-ELM	DP-ELM	CP-ELM	BP
Banana	19.2$^{(36)}$ (4.4)	**3**$^{(2)}$ (0.4)	50$^{(100)}$ (0)	50$^{(100)}$ (0)	50$^{(100)}$ (0)	50$^{(100)}$ (0)	**2**$^{(0)}$ (0)	39.9$^{(79)}$ (1.9)	49$^{(98)}$ (1.1)	***15***$^{(27)}$ (0)
Breast cancer	**3.8**$^{(0)}$ (1.4)	**6.1**$^{(6)}$ (1.8)	20$^{(39)}$ (0)	20$^{(39)}$ (0)	20$^{(39)}$ (0)	20$^{(39)}$ (0)	***8***$^{(10)}$ (0)	16.1$^{(30)}$ (1.3)	19.7$^{(39)}$ (0.5)	45$^{(100)}$ (0)
Diabetis	**2.3**$^{(0)}$ (1.1)	**5.6**$^{(8)}$ (1.4)	30$^{(65)}$ (0)	30$^{(65)}$ (0)	30$^{(65)}$ (0)	30$^{(65)}$ (0)	***7.4***$^{(12)}$ (0.5)	23$^{(48)}$ (1.8)	29.6$^{(64)}$ (0.6)	45$^{(100)}$ (0)
Flare solar	9.6$^{(95)}$ (2.4)	**3.5**$^{(16)}$ (1.6)	10$^{(100)}$ (0)	10$^{(100)}$ (0)	10$^{(100)}$ (0)	10$^{(100)}$ (0)	**2.3**$^{(0)}$ (0.5)	***8***$^{(74)}$ (0.7)	9.9$^{(99)}$ (0.3)	10$^{(100)}$ (0)
German	**12.5**$^{(10)}$ (5)	**5.9**$^{(0)}$ (1)	40$^{(49)}$ (0)	40$^{(49)}$ (0)	40$^{(49)}$ (0)	40$^{(49)}$ (0)	***18***$^{(18)}$ (0)	32.2$^{(38)}$ (1.7)	39.7$^{(49)}$ (0.5)	75$^{(100)}$ (0)
Heart	13.4$^{(29)}$ (3.1)	**6.5**$^{(0)}$ (2.9)	30$^{(100)}$ (0)	30$^{(100)}$ (0)	30$^{(100)}$ (0)	30$^{(100)}$ (0)	***11.9***$^{(23)}$ (0.3)	23.6$^{(73)}$ (1.4)	29.7$^{(99)}$ (0.6)	**10**$^{(15)}$ (0)
Image	42.9$^{(19)}$ (7)	**6.8**$^{(0)}$ (3.4)	200$^{(100)}$ (0)	200$^{(100)}$ (0)	200$^{(100)}$ (0)	200$^{(100)}$ (0)	**9.5**$^{(1)}$ (0.5)	154$^{(76)}$ (4.1)	198.3$^{(99)}$ (1.5)	***15***$^{(4)}$ (0)
Ringnorm	67.7$^{(100)}$ (6.3)	**5.5**$^{(0)}$ (3)	40$^{(55)}$ (0)	40$^{(55)}$ (0)	40$^{(55)}$ (0)	40$^{(55)}$ (0)	**19**$^{(22)}$ (0)	***32.2***$^{(43)}$ (2.1)	39.6$^{(55)}$ (0.7)	50$^{(72)}$ (0)
Splice	**15.2**$^{(7)}$ (5.1)	**4.7**$^{(0)}$ (2.3)	165$^{(100)}$ (0)	165$^{(100)}$ (0)	165$^{(100)}$ (0)	165$^{(100)}$ (0)	***33***$^{(18)}$ (0)	85.9$^{(51)}$ (51.3)	164.4$^{(100)}$ (0.8)	80$^{(47)}$ (0)
Thyroid	29.1$^{(81)}$ (5)	**6**$^{(5)}$ (1.6)	35$^{(100)}$ (0)	35$^{(100)}$ (0)	35$^{(100)}$ (0)	35$^{(100)}$ (0)	**4.4**$^{(0)}$ (0.5)	26.1$^{(71)}$ (2.2)	34.4$^{(98)}$ (0.9)	***25***$^{(67)}$ (0)
Titanic	***5.4***$^{(11)}$ (1.2)	**3.4**$^{(2)}$ (1)	20$^{(77)}$ (0)	20$^{(77)}$ (0)	20$^{(77)}$ (0)	20$^{(77)}$ (0)	**3**$^{(0)}$ (0.1)	11.1$^{(37)}$ (1.1)	12.1$^{(41)}$ (1.1)	25$^{(100)}$ (0)
Twonorm	**1**$^{(0)}$ (0)	***5.5***$^{(7)}$ (1.8)	65$^{(100)}$ (0)	65$^{(100)}$ (0)	65$^{(100)}$ (0)	65$^{(100)}$ (0)	19$^{(28)}$ (0)	43.5$^{(66)}$ (3)	63.3$^{(97)}$ (1.5)	**5**$^{(6)}$ (0)
Waveform	18.4$^{(14)}$ (5.6)	**7.7**$^{(0)}$ (2.5)	85$^{(100)}$ (0)	85$^{(100)}$ (0)	85$^{(100)}$ (0)	85$^{(100)}$ (0)	***17.1***$^{(12)}$ (0.2)	66.1$^{(76)}$ (2.7)	84.2$^{(99)}$ (0.9)	**10**$^{(3)}$ (0)
Mean scores	***30.9***	**3.5**	83.5	83.5	83.5	83.5	**11.1**	58.6	79.8	57

and training time — see Tables 7, 8 and 9. We searched for the best parameter, i.e., the number of hidden nodes or the maximum of hidden nodes, using the first 10 realizations of each dataset, with the cross-validation method — see Table 6. For BP and ELM algorithms, we searched for the optimum number of hidden nodes by searching in steps of 5 up to 200,

TABLE 9. Training time and time ranks for the classification problems.

Dataset	Algorithms : $\mathtt{trtime}^{(\mathrm{rankscore})}(\sigma)$									
	LSM	AIL	ELM	I-ELM	EI-ELM	EB-ELM	PCA-ELM	DP-ELM	CP-ELM	BP
Banana	$0.1216^{(49)}$ (0.0609)	$0.0217^{(7)}$ (0.0307)	$\boldsymbol{\mathit{0.0116}}^{(3)}$ (0.0270)	$\mathbf{0.0055}^{(0)}$ (0.0240)	$0.0242^{(8)}$ (0.0382)	$0.2402^{(100)}$ (0.0828)	$\mathbf{0.0064}^{(0)}$ (0.0186)	$0.0272^{(9)}$ (0.0229)	$0.0417^{(15)}$ (0.0268)	$0.0795^{(32)}$ (0.0385)
Breast cancer	$0.0153^{(5)}$ (0.0282)	$0.0145^{(5)}$ (0.0167)	$\boldsymbol{\mathit{0.0066}}^{(2)}$ (0.0189)	$\mathbf{0.0023}^{(0)}$ (0.0139)	$\underline{\mathbf{0.0058}}^{(1)}$ (0.0185)	$0.0542^{(20)}$ (0.0158)	$\boldsymbol{\mathit{0.0070}}^{(2)}$ (0.0202)	$0.0089^{(3)}$ (0.0213)	$0.0128^{(4)}$ (0.0239)	$0.2595^{(100)}$ (0.0539)
Diabetis	$\boldsymbol{\mathit{0.0103}}^{(4)}$ (0.0239)	$0.0372^{(14)}$ (0.0322)	$\boldsymbol{\mathit{0.0119}}^{(4)}$ (0.0280)	$\mathbf{0.0011}^{(0)}$ (0.0109)	$0.0183^{(7)}$ (0.0343)	$0.1588^{(62)}$ (0.0736)	$\underline{\mathbf{0.0075}}^{(3)}$ (0.0178)	$0.0144^{(5)}$ (0.0236)	$0.0177^{(7)}$ (0.0209)	$0.2538^{(100)}$ (0.0553)
Flare solar	$0.0597^{(66)}$ (0.0338)	$0.0250^{(27)}$ (0.0328)	$\underline{\mathbf{0.0047}}^{(4)}$ (0.0158)	$\mathbf{0.0011}^{(0)}$ (0.0109)	$0.0105^{(11)}$ (0.0307)	$0.0573^{(63)}$ (0.0376)	$0.0075^{(7)}$ (0.0205)	$\boldsymbol{\mathit{0.0056}}^{(5)}$ (0.0197)	$0.0072^{(7)}$ (0.0215)	$0.0903^{(100)}$ (0.0356)
German	$0.0591^{(2)}$ (0.0369)	$0.0419^{(1)}$ (0.0299)	$\mathbf{0.0147}^{(0)}$ (0.0322)	$\mathbf{0.0042}^{(0)}$ (0.0208)	$0.0630^{(2)}$ (0.0534)	$0.3403^{(11)}$ (0.0721)	$\mathbf{0.0075}^{(0)}$ (0.0211)	$\mathbf{0.0188}^{(0)}$ (0.0236)	$0.0275^{(1)}$ (0.0234)	$3.162^{(100)}$ (0.2997)
Heart	$0.0230^{(25)}$ (0.0234)	$0.0183^{(19)}$ (0.0211)	$\underline{\mathbf{0.0047}}^{(3)}$ (0.0151)	$\mathbf{0.0023}^{(0)}$ (0.0139)	$\boldsymbol{\mathit{0.0066}}^{(5)}$ (0.0193)	$0.0833^{(98)}$ (0.0206)	$0.0075^{(6)}$ (0.0203)	$0.0117^{(11)}$ (0.0218)	$0.0172^{(18)}$ (0.0180)	$0.0847^{(100)}$ (0.0353)
Image	$0.3117^{(12)}$ (0.0765)	$\boldsymbol{\mathit{0.0531}}^{(2)}$ (0.0400)	$\boldsymbol{\mathit{0.0500}}^{(2)}$ (0.0241)	$\underline{\mathbf{0.043}}^{(1)}$ (0.0541)	$0.4492^{(18)}$ (0.0343)	$1.8258^{(75)}$ (0.3619)	$\mathbf{0.0117}^{(0)}$ (0.0299)	$2.4328^{(100)}$ (0.1481)	$0.5836^{(24)}$ (0.0406)	$0.2695^{(11)}$ (0.1983)
Ringnorm	$0.2745^{(23)}$ (0.0456)	$0.0433^{(3)}$ (0.0395)	$\boldsymbol{\mathit{0.0116}}^{(1)}$ (0.0279)	$\mathbf{0.0028}^{(0)}$ (0.0165)	$0.0247^{(2)}$ (0.0375)	$0.2155^{(18)}$ (0.0593)	$\mathbf{0.0086}^{(0)}$ (0.0208)	$\boldsymbol{\mathit{0.0142}}^{(1)}$ (0.0137)	$0.0233^{(2)}$ (0.0222)	$1.1675^{(100)}$ (0.2039)
Splice	$\mathbf{0.0969}^{(0)}$ (0.0417)	$\mathbf{0.0367}^{(0)}$ (0.0337)	$\mathbf{0.0453}^{(0)}$ (0.0358)	$\mathbf{0.0328}^{(0)}$ (0.0514)	$0.4172^{(1)}$ (0.0476)	$1.5047^{(5)}$ (0.1826)	$\mathbf{0.0188}^{(0)}$ (0.0364)	$1.3914^{(4)}$ (0.4409)	$0.3820^{(1)}$ (0.0330)	$32.7688^{(100)}$ (2.2782)
Thyroid	$0.0605^{(67)}$ (0.0400)	$0.0159^{(17)}$ (0.0184)	$0.0089^{(9)}$ (0.0200)	$\mathbf{0.0008}^{(0)}$ (0.0064)	$\underline{\mathbf{0.0017}}^{(1)}$ (0.0049)	$0.0898^{(100)}$ (0.0211)	$\boldsymbol{\mathit{0.0066}}^{(7)}$ (0.0189)	$0.0113^{(12)}$ (0.0151)	$0.0177^{(19)}$ (0.0147)	$0.0881^{(98)}$ (0.0377)
Titanic	$0.0791^{(100)}$ (0.0387)	$0.0089^{(11)}$ (0.0136)	$0.0103^{(13)}$ (0.0246)	$\mathbf{<10}^{-4(0)}$	$\underline{\mathbf{0.0047}}^{(6)}$ (0.0161)	$0.0575^{(73)}$ (0.0207)	$\boldsymbol{\mathit{0.0066}}^{(8)}$ (0.0204)	$0.0077^{(10)}$ (0.0168)	$0.0072^{(9)}$ (0.0177)	$0.0448^{(57)}$ (0.0303)
Twonorm	$\mathbf{0.0069}^{(0)}$ (0.0198)	$0.0366^{(9)}$ (0.0391)	$0.0141^{(2)}$ (0.0287)	$\mathbf{0.0083}^{(0)}$ (0.0283)	$0.0273^{(6)}$ (0.0371)	$0.3347^{(100)}$ (0.1031)	$\mathbf{0.0081}^{(0)}$ (0.0212)	$0.0736^{(20)}$ (0.0272)	$0.0594^{(16)}$ (0.0276)	$0.1230^{(35)}$ (0.0670)
Waveform	$0.0780^{(19)}$ (0.0374)	$0.0614^{(15)}$ (0.0387)	$\boldsymbol{\mathit{0.0164}}^{(4)}$ (0.0310)	$\mathbf{0.0017}^{(0)}$ (0.0125)	$0.0502^{(12)}$ (0.0323)	$0.4133^{(100)}$ (0.0909)	$\underline{\mathbf{0.0066}}^{(1)}$ (0.0200)	$0.1239^{(30)}$ (0.0325)	$0.0922^{(22)}$ (0.0302)	$0.1570^{(38)}$ (0.0520)
Mean scores	28.6	10	***3.6***	**0.1**	6.2	63.5	$\underline{\mathbf{2.6}}$	16.2	11.2	74.7

i.e., $N = \{5, 10, \ldots, 200\}$. For I-ELM, EI-ELM, EB-ELM, CP-ELM and DP-ELM, the maximum number of hidden nodes was set the same as ELM. The maximum number of hidden nodes in AIL was set to twice the number of nodes of BP. For LSM, we searched for the maximum iteration, l, from 1 to 15.

Table 7 shows the average and deviation of the error rate on each dataset. Results showed almost similar trend of mean scores to those in the regression problems, i.e., BP (0.8) < LSM (42.3) < ELM (55.9) and the poor mean score algorithms remained PCA-ELM (84.2) and I-ELM (78.8), while other ELMs scored from 55 to 60. Mostly, BP was the best, when compared to other algorithms on all datasets, except for Twonorm, in which BP scored 11 which is the second best after the PCA-ELM (0). This happened because Twonorm dataset might have a nice data distribution. It was confirmed by the LSM performance (the third best) that was only one hidden node (see Table 8), which means that Twonorm dataset was a linear problem because it only had one segment. In other words, Twonorm dataset can be solved using a linear combination of its eigenvectors. Generally, LSM was the second or third best in the most cases, followed by AIL.

The number of hidden nodes required for each algorithm is shown in Table 8. Here, AIL was the most compact algorithm, followed by PCA-ELM and LSM, which were very small, when compared to the ELM and its variants. Again, we are not only finding the most compact and fastest algorithm but also requiring the most accurate one. Note that an accurate algorithm is the first criteria in our work; a compact and fast algorithm is useless if it is not accurate. Surprisingly, the mean score of LSM (30.9) is much more compact than BP (57) for this classification problem. A compact model implies that the inference time will be much faster than a non-compact model. Hence, the model only needs a small operation count, because it has a small parameter set. Moreover, a compact model will lead to faster training time as well, when we compare the same algorithm with a different number of hidden nodes.

Training time for each algorithm on each dataset are listed in Table 9. As with the regression case, the fastest algorithms were I-ELM, PCA-ELM and ELM. However, I-ELM and PCA-ELM were generally the most inaccurate models. Interestingly, on the Twonorm dataset, LSM was the fastest algorithm. This happened because LSM on Twonorm only used one hidden node - see the second last row of Table 8. Thus, the number of hidden nodes required by the algorithm not only affected the inference speed but also the training speed. The more compact the model, the faster the model is. For Splice dataset, BP was around 300 times slower than others. BP might use all 100 epochs for updating the weights since the gradient based method is generally hard to find the global solution on non-convex problems. In summary, overall algorithm performance for classification problems was generally similar to that for the regression problems: BP and LSM were the most accurate algorithms; AIL, PCA-ELM, LSM and BP were the most compact algorithms; then, I-ELM, PCA-ELM and ELM were the fastest algorithms.

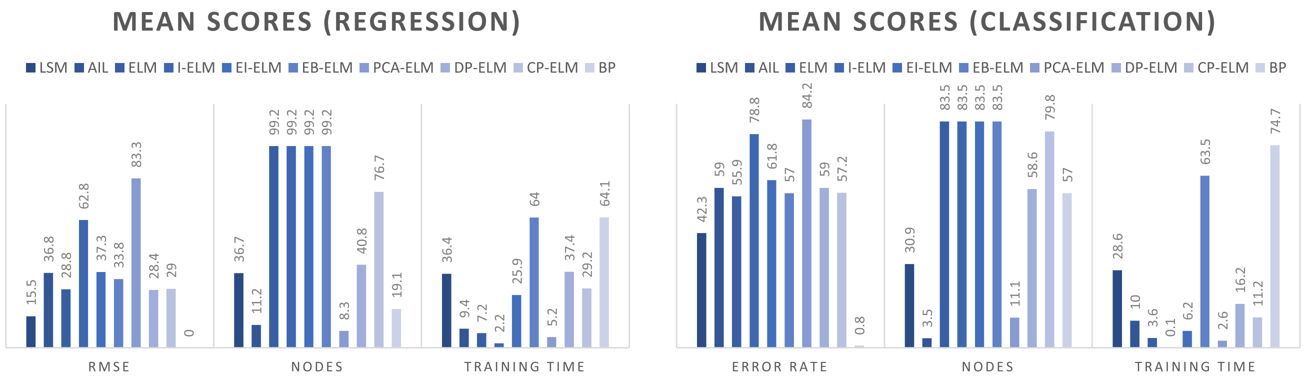


FIGURE 15. **Average ranks for accuracy, final number of hidden nodes and training time for regression problems (left) and classification problem (right).**

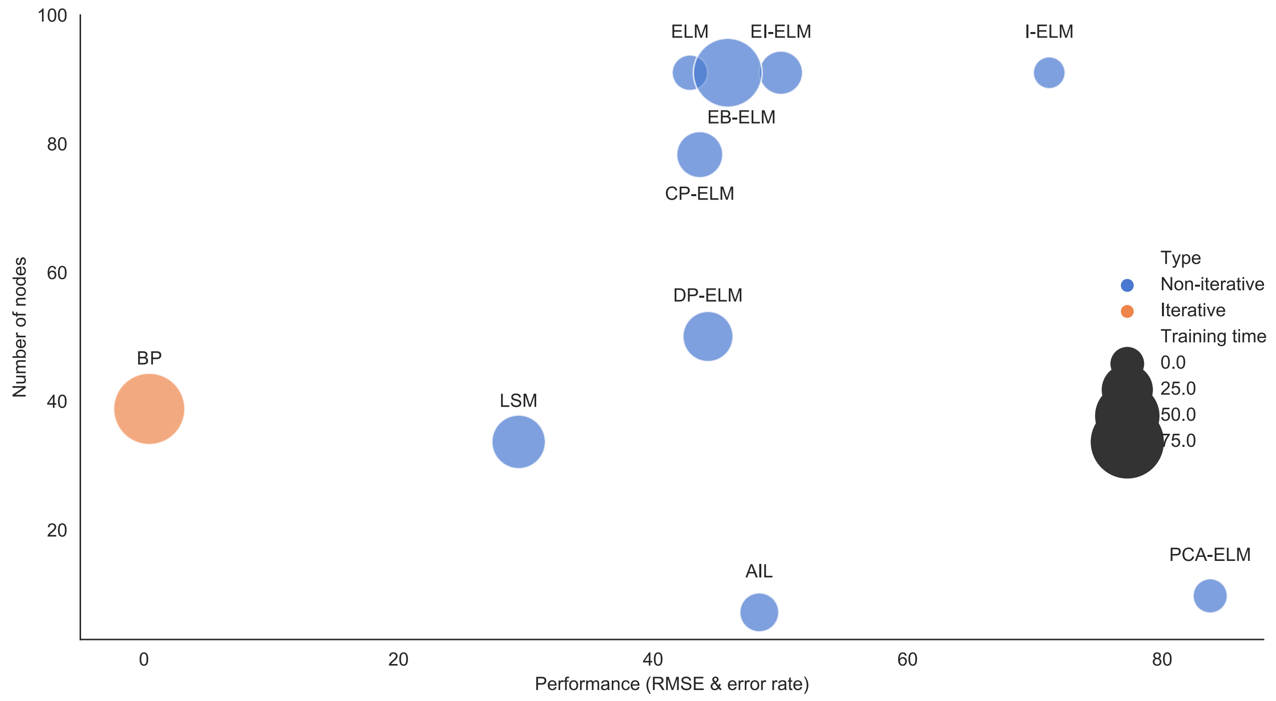


FIGURE 16. **Average rank scores for performance, final number of hidden nodes and training time for all cases. Closer to the origin (0,0) and a smaller bubble size implies a better algorithm.**

C. DISCUSSION

We have shown that LSM outperformed in terms of accuracy and compactness. Relative performance is visualized in Fig. 15 for both regression and classification problems. We can see that LSM is more accurate than other non-iterative methods on both problems. ELM and its variants use randomization without any optimization of the input weight assignment that leads to faster training time. However, they have a large number of hidden nodes that lead to the slowest testing time. According to LSM segmentation method, ELM will add random segments, which may cover some key segments in a particular interval, while LSM uses a deterministic method without randomization to define the optimized segment set. Therefore, LSM is more stable than ELM and its variants. AIL and PCA-ELM are also deterministic, but LSM still performs better. However, different from LSM and AIL, PCA-ELM has poor performance, because it does not use any information of the prediction variable and only relies on the input feature distribution. However, BP remains the best algorithm in terms of the RMSE and error rate. In terms of hidden node numbers, AIL is the most compact algorithm, but it cannot generate more hidden nodes, due to the node saturation and thus terminates early. With the AIL's limitation, LSM can be used as an alternative that generates more hidden nodes, but is still sufficiently compact and more accurate. Most importantly, LSM is based on the sigmoid node that represents a subset of data; therefore, we are able to know how the algorithm builds its model structure and how each node segments the data.

We took the average rank scores for all cases (both regression and classification) and plotted the mean scores of three metrics (error, number of nodes and training time) using closeness to the origin and small size of the 'bubble' as the optimum point in Fig. 16. Clearly, this shows that LSM is the best algorithm among the non-iterative algorithms and LSM is among the best three algorithms, that have a small hidden node count. However, we can exclude the PCA-ELM, because it performed poorly, so LSM became the most compact algorithm after AIL. In addition, in terms of speed, ELM and its variants are still the fastest algorithms. Moreover, LSM was faster than BP, due to its non-iterative nature.

Finally, we summarize the advantages and disadvantages of our LSM as follows:

(+) It does not have to defined the exact number of hidden nodes (uses the maximum iteration, l), i.e., automatic hidden layer construction.
(+) It does not depend on any randomization (deterministic).
(+) It is generally more accurate than other non-iterative learning algorithms, in our evaluation.
(+) It produces one of the most compact models, that leads to faster inference time.
(−) It is slower than ELM, in terms of training time.
(−) It is not more accurate than BP, in terms of performances.
(−) It only uses a sigmoid function as the activation function.

D. ABILITY TO UNDERSTAND THE NETWORK STRUCTURE

Mostly, we treat a neural network model as a black box: we can train using a labeled dataset and obtain the result with high performance. As we explained in Section II, LSM uses a sigmoid as a segmenting function, that allows us to keep input information in a particular input interval (segment) and ignore input that is outside this segment. Using this capability, we can examine the optimal model trained by BP and interpret each hidden node using interval segmentation, i.e., each node is responsible for a particular interval in the input. As a result, we showed that BP, using the same number of hidden nodes as LSM, acts similarly, i.e., it segments the input interval into several segments to approximate a function, such as $\sin(x)$ and $\text{sinc}(x)$ functions. Using the LSM theory, for the example, $\text{sinc}(x)$ in $[-10, 10]$, we showed that BP is able to form an overlapped segmentation, that can combine with other segments.

V. CONCLUSION

We have described an LSM algorithm that addresses both problems of ELM randomization and AIL node saturation. The LSM algorithm builds its structure based on the interval segmentation method that allows us to divide a dataset into several subsets (segments), represented by subsets of the data points in the input, so that each segment is represented by a single node. Thus the LSM building blocks are nodes, which can be associated with specific segments of the input data, i.e., the network is no longer a black box, one can understand the role of each node and its contribution to the final output.

Furthermore, we showed that LSM is generally faster than BP, an iterative method, and is more accurate than those in a set of non-iterative algorithms visualized in Fig. 16. Moreover, LSM model is one of the most compact models that leads to the faster inference time on testing data. An associated benefit is that, in the training phase, LSM can automatically construct a model without requiring a predefined number of hidden nodes, because it builds nodes that can explain individual segments of the input data.

A. FUTURE WORK

LSM is still in early phase development. Thus, we will further improve to make LSM efficient. In this work, LSM is only for a single output dataset. Moreover, as discussed in Section IV-D, BP is able to form an overlapped segmentation. Hence, we will also investigate further improvement for LSM to handle these overlaps, thus leading to a more compact structure. Also, there are many activation functions used in neural network models, e.g., Rectified Linear Unit (ReLU), Leaky-ReLU [56], parameterized ReLU (P-ReLU) [57] and Exponential Linear Unit (ELU) [58]. There is a possibility that we can build learning methods, based on the properties of those functions. Furthermore, we suggest that one can use multiple activation functions in a single model based on their properties and exploit them to represent different, perhaps larger, segments of the input, so that the number of nodes can be reduced without sacrificing accuracy.

ACKNOWLEDGMENT

We thank John Morris, KRIS KMITL, for correcting and reducing the volume of this paper.

REFERENCES

[1] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," in *Readings in Cognitive Science: A Perspective from Psychology and Artificial Intelligence*, A. Collins and E. E. Smith, Eds. San Mateo, CA, USA: Kaufmann, 1988, pp. 399–421.
[2] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, U.K.: Oxford Univ. Press, 1995.
[3] E. Alpaydin, *Introduction to Machine Learning*. Cambridge, MA, USA: MIT Press, 2009.
[4] D. W. Marquardt, "An algorithm for least-squares estimation of nonlinear parameters," *J. Soc. Ind. Appl. Math.*, vol. 11, no. 2, pp. 431–441, 1963.
[5] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," 2012, *arXiv:1212.5701*. [Online]. Available: http://arxiv.org/abs/1212.5701
[6] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, Feb. 2011.
[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014, *arXiv:1412.6980*. [Online]. Available: https://arxiv.org/abs/1412.6980
[8] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Upper Saddle River, NJ, USA: Prentice-Hall, 1994.
[9] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Netw.*, vol. 4, no. 2, pp. 251–257, 1991.
[10] M. Leshno, V. Y. Lin, A. Pinkus, and S. Schocken, "Multilayer feedforward networks with a nonpolynomial activation function can approximate any function," *Neural Netw.*, vol. 6, no. 6, pp. 861–867, 1993.
[11] V. E. Ismailov, "Approximation by neural networks with weights varying on a finite set of directions," *J. Math. Anal. Appl.*, vol. 389, no. 1, pp. 72–83, 2012.
[12] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications," *Neurocomputing*, vol. 70, nos. 1–3, pp. 489–501, 2006.

[13] W. Cao, X. Wang, Z. Ming, and J. Gao, "A review on neural networks with random weights," *Neurocomputing*, vol. 275, pp. 278–287, Jan. 2018.

[14] 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, pp. 1828–1841, Oct. 2014.

[15] C. Zhang, X. Bian, P. Liu, X. Tan, Q. Fan, W. Liu, and L. Lin, "Subagging for the improvement of predictive stability of extreme learning machine for spectral quantitative analysis of complex samples," *Chemometrics Intell. Lab. Syst.*, vol. 161, pp. 43–48, Feb. 2017.

[16] 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, pp. 879–892, Jul. 2006.

[17] G.-B. Huang and L. Chen, "Enhanced random search based incremental extreme learning machine," *Neurocomputing*, vol. 71, nos. 16–18, pp. 3460–3468, Oct. 2008.

[18] Y. Yang, Y. Wang, and X. Yuan, "Bidirectional extreme learning machine for regression problem and its learning effectiveness," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 9, pp. 1498–1505, Sep. 2012.

[19] W. Cao, Z. Ming, X. Wang, and S. Cai, "Improved bidirectional extreme learning machine based on enhanced random search," *Memetic Comput.*, vol. 11, no. 1, pp. 19–26, Mar. 2019, doi: 10.1007/s12293-017-0238-1.

[20] H.-J. Rong, Y.-S. Ong, A.-H. Tan, and Z. Zhu, "A fast pruned-extreme learning machine for classification problem," *Neurocomputing*, vol. 72, no. 1, pp. 359–366, 2008.

[21] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten, and A. Lendasse, "OP-ELM: Optimally pruned extreme learning machine," *IEEE Trans. Neural Netw.*, vol. 21, no. 1, pp. 158–162, Jan. 2010.

[22] Y. Yu and Z. Sun, "A pruning algorithm for extreme learning machine based on sparse coding," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2016, pp. 2596–2602.

[23] J. E. Bobrow and W. Murray, "An algorithm for RLS identification parameters that vary quickly with time," *IEEE Trans. Autom. Control*, vol. 38, no. 2, pp. 351–354, Feb. 1993.

[24] A. Castaño and 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, pp. 377–392, 2013.

[25] 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 *Proc. Int. Conf. Neural Inf. Process.* Cham, Switzerland: Springer, 2016, pp. 259–268.

[26] A. N. Kolmogorov, "On the representation of continuous functions of many variables by superposition of continuous functions of one variable and addition," *Doklady Akademii Nauk*, vol. 114, no. 5, pp. 953–956, 1957.

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

[28] Y. Xu, W. Light, and E. Cheney, "Constructive methods of approximation by ridge functions and radial functions," *Numer. Algorithms*, vol. 4, no. 2, pp. 205–223, 1993.

[29] D. Costarelli and R. Spigler, "Constructive approximation by superposition of sigmoidal functions," *Anal. Theory Appl.*, vol. 29, no. 2, pp. 169–196, 2013.

[30] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel "Backpropagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.

[31] D. Erhan, Y. Bengio, A. Courville, and P. Vincent, "Visualizing higher-layer features of a deep network," *Univ. Montreal*, vol. 1341, no. 3, p. 1, 2009.

[32] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," 2013, *arXiv:1312.6034*. [Online]. Available: https://arxiv.org/abs/1312.6034

[33] C. Olah, A. Mordvintsev, and L. Schubert, "Feature visualization," *Distill*, vol. 2, no. 11, 2017, Art. no. e7. [Online]. Available: https://distill.pub/2017/feature-visualization

[34] A. Nguyen, J. Clune, Y. Bengio, A. Dosovitskiy, and J. Yosinski, "Plug & play generative networks: Conditional iterative generation of images in latent space," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Jul. 2017, pp. 4467–4477.

[35] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proc. Eur. Conf. Comput. Vis.* Cham, Switzerland: Springer, 2014, pp. 818–833.

[36] P.-J. Kindermans and K. T. Schütt, M. Alber, K.-R. Müller, D. Erhan, B. Kim, and S. Dähne, "Learning how to explain neural networks: Patternnet and patternattribution," 2017, *arXiv:1705.05598*. [Online]. Available:

[37] R. C. Fong and A. Vedaldi, "Interpretable explanations of black boxes by meaningful perturbation," 2017, *arXiv:1704.03296*. [Online]. Available: https://arxiv.org/abs/1704.03296

[38] C. Olah, A. Satyanarayan, I. Johnson, S. Carter, L. Schubert, K. Ye, and A. Mordvintsev, "The building blocks of interpretability," *Distill*, vol. 3, no. 3, 2018, Art. no. e10. [Online]. Available: https://distill.pub/2018/building-blocks

[39] N. Kyurkchiev and S. Markov, "Sigmoid functions: Some approximation and modelling aspects," in *Some Moduli in Programming Environment Mathematica*, Lambert Academic, Saarbrucken, Germany, 2015, pp. 3–978.

[40] A. I. Iliev, N. Kyurkchiev, and S. Markov, "On the approximation of the cut and step functions by logistic and Gompertz functions," *Biomath*, vol. 4, no. 2, 2015, Art. no. 1510101.

[41] F. Hausdorff, *Set theory*. North Providence, RI, USA: American Mathematical Society, 2005, pp. 119.

[42] B. Sendov, *Hausdorff Approximations*, vol. 50. Amsterdam, The Netherlands: Springer, 1990.

[43] T. Banachiewicz, "Zur berechnung der determinanten, wie auch der inversen und zur darauf basierten auflosung der systeme linearer gleichungen," *Acta Astronom. C*, vol. 3, pp. 41–67, 1937.

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

[45] S. A. I. Alfarozi, "Least square neural network," 2019. [Online]. Available: https://github.com/sykrn/lsnn

[46] K. Levenberg, "A method for the solution of certain non-linear problems in least squares," *Quart. J. Appl. Math.*, vol. 2, no. 2, pp. 164–168, Jul. 1944.

[47] J. J. Moré, "The levenberg-marquardt algorithm: Implementation and theory," in *Numer. Anal.* Berlin, Germany: Springer, 1978, pp. 105–116.

[48] L. Torgo, "Regression datasets," 2019. [Online]. Available: http://www.dcc.fc.up.pt/~ltorgo/Regression/DataSets.html

[49] S. Mika and G. Ratsch, J. Weston, B. Scholkopf, and K.-R. Müller, "Fisher discriminant analysis with kernels," in *Proc. Neural Netw. Signal Process. IX, IEEE Signal Process. Soc. Workshop*, Aug. 1999, pp. 41–48.

[50] G. Rätsch, T. Onoda, and K.-R. Müller, "Soft margins for adaboost," *Mach. Learn.*, vol. 42, no. 3, pp. 287–320, 2001.

[51] K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf, "An introduction to kernel-based learning algorithms," *IEEE Trans. Neural Netw.*, vol. 12, no. 2, pp. 181–201, Mar. 2001.

[52] M. Viola, M. Sangiovanni, G. Toraldo, and M. R. Guarracino, "Semi-supervised generalized eigenvalues classification," *Ann. Oper. Res.*, vol. 276, nos. 1–2, pp. 249–266, 2019.

[53] S. S. Mullick, S. Datta, and S. Das, "Adaptive learning-based k-nearest neighbor classifiers with resilience to class imbalance," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 11, pp. 5713–5725, Nov. 2018.

[54] G. Rätsch, "Ida benchmark repository," 2018. [Online]. Available: http://www.raetschlab.org/Members/raetsch/benchmark

[55] T. Diethe, "13 benchmark datasets derived from the UCI, DELVE and STATLOG repositories," 2015. [Online]. Available: https://github.com/tdiethe/gunnar_raetsch_benchmark_datasets/

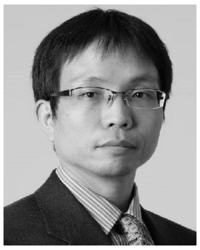
[56] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, vol. 30, no. 1, Jun. 2013, p. 3.

[57] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proc. IEEE Int. Conf. Comput. Vis.*, Dec. 2015, pp. 1026–1034.

[58] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (Elus)," 2015, *arXiv:1511.07289*. [Online]. Available: https://arxiv.org/abs/1511.07289

SYUKRON ABU ISHAQ ALFAROZI (Member, IEEE) received the B.Eng. degree in electrical engineering and information technology from Universitas Gadjah Mada, Yogyakarta, Indonesia, in 2014. He is currently pursuing the Ph.D. degree in information technology with the King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand, under the sandwich program with Hokkaido University, Sapporo, Japan. His research interests include machine learning and its application, computer vision, and signal processing.

KITSUCHART PASUPA (Senior Member, IEEE) received the B.Eng. degree in electrical engineering from the Sirindhorn International Institute of Technology, Thammasat University, Thailand, in 2003, and the M.Sc. (Eng.) and Ph.D. degrees in automatic control and systems engineering from the Department of Automatic Control and Systems Engineering, The University of Sheffield, in 2004 and 2008, respectively. He was a Research Fellow with the University of Southampton and The University of Sheffield. He is currently an Associate Professor with the Faculty of Information Technology, King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand. His main research interests include the application of machine learning techniques in the real world application.



MASANORI SUGIMOTO (Member, IEEE) received the B.E., M.E., and D.E. degrees in aeronautics and astronautics from The University of Tokyo, Tokyo, Japan, in 1990, 1992 and 1995, respectively. He is currently a Professor with the Graduate School of Information Science and Technology, Hokkaido University, Sapporo, Japan. His research interests include acoustic engineering, signal processing, artificial intelligence, and human–computer interaction technologies for designing smart systems and environments.

KUNTPONG WORARATPANYA (Member, IEEE) received the B.Ind.Tech. degree in computer technology, the M.Eng. degree in computer engineering, and the D.Eng. degree in electrical engineering from the King Mongkut's Institute of Technology Ladkrabang, Bangkok, Thailand, in 1992, 1996, and 2005, respectively. He is currently an Assistant Professor with the Faculty of Information Technology, King Mongkut's Institute of Technology Ladkrabang. His research interests include stereoscopic acquisition and compression, multimedia coding and processing, signal processing, speech recognition and processing, pattern recognition and image processing, computer vision, and machine learning/deep learning.