A Hybrid Online Sequential Extreme Learning Machine with Simplified Hidden Network

M. J. Er, L. Y. Zhai, X. Li and L. San

***Abstract*—In this paper, a novel learning algorithm termed Hybrid Online Sequential Extreme Learning Machine (HOS-ELM) is proposed. The proposed HOS-ELM algorithm is a fusion of the Online Sequential Extreme Learning Machine (OS-ELM) and the Minimal Resource Allocation Network (MRAN). It is capable of reducing the number of hidden nodes in Single-hidden Layer Feed-forward Neural Networks (SLFNs) with Radial Basis Function (RBF) by virtue of adjustment in node allocation and pruning capability. Simulation results show that the generalization performance of the proposed HOS-ELM is comparable to the original OS-ELM with significant reduction in the number of hidden nodes.**

***Index Terms*—single-hidden layer feed-forward neural networks (SLFN), minimal resource allocation network (MRAN), extreme learning machine (ELM), neural networks, online sequential learning**

I. INTRODUCTION

IT is well known that Artificial Neural Networks (ANN) can provide an input-output mapping of an unknown dynamic system whereby the system's output can be written as a function of inputs and parameters or weights of the network [1]. The learning process of neural networks amounts to approximation of an underlying function, which subsequently can be translated to an estimation of the parameters (or weights) that are optimal in some sense. In this regard, the feed forward neural networks have received extensive attention of researchers due to their capability of mapping any nonlinear and non-stationary functions to an arbitrary degree of accuracy. One of such popular feed-forward networks is the Radial Basis Function (RBF) network which offers an efficient mechanism for approximating complex nonlinear mappings between the input and output data. RBF networks have been popularly used in many applications in recent years due to their ability to approximate complex nonlinear mappings directly from the input-output data with a simple topological structure and ease of implementation of dynamic and adaptive network architecture [2-4]. However, as highlighted in many studies, conventional feed-forward neural networks are usually slow in the training process and in many cases, cannot meet the speed requirement for online learning. As a significant contribution in improving the training speed of neural networks, Platt [5] proposed a sequential learning algorithm through the development of a Resource Allocation Network (RAN), in which hidden neurons were added sequentially based on the novelty of the new data. In practical on-line applications, sequential learning algorithms are generally preferred over batch learning algorithms as they do not require retraining whenever a new data is received. Further enhancements of the RAN, known as RANEKF [6] and MRAN [7], were also proposed to improve the parameter update strategy and the pruning method incorporated. More recently, a further modification to the RBF, known as GAP-RBF [8], was proposed to simplify the sequential learning procedure by linking the required accuracy directly to the learning algorithm and increase the learning speed by adjusting the nearest neuron only.

As a novel approach to improve the speed of neural network training process, Huang et al. [9] proposed a new learning algorithm called Extreme Learning Machine (ELM) for Single-hidden Layer Feed-forward Neural Networks (SLFNs). The ELM can randomly choose the initial hidden nodes and determine the output weights of SLFNs using matrix calculations. Since its inception, the ELM has been shown to be extremely fast in training with better generalization performance than other batch training algorithms. An extended application of ELM to the sequential learning process, which is known as Online Sequential Extreme Learning Machine (OS-ELM) [10], provides another promising solution to online learning problems. However, compared to other learning algorithms such as MRAN and GAP-RAN, the ELM and OS-ELM algorithms usually require much more hidden nodes for the network, which would increase the demand for testing time (the operation time) in real-world applications due to more complex network structure.

With a view of simplifying the hidden node structure of the OS-ELM algorithm, a hybrid learning algorithm integrating the MRAN algorithm with the OS-ELM is proposed in this paper. The node generation and pruning strategies of the MRAN are extended to the OS-ELM so that the parameters of the hidden nodes could be automatically adjusted, and subsequently the number of nodes used in the neural network reduced. The rest of the paper is organized as follows. Section II presents a brief review of various learning methods including heuristic learning of SLFNs, MRAN and the OS-ELM algorithm. The performance of the OS-ELM in different function approximation problems are summarized in Section III. Section IV elaborates on the proposed hybrid OS-ELM algorithm (HOS-ELM). Section V discusses some issues encountered in the performance evaluation process of the HOS-ELM and suggestions for further improvement of the proposed algorithm are

Manuscript received May 25, 2011; revised August 15, 2011. This work was supported in part by A*STAR Science and Engineering Research Council (SERC) – Singapore-Poland Fund under Grant No. 072 134 0057.

M. J. Er is with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798 (phone: 0065-67904529; fax: 0065- 68968757; e-mail: emjer@ntu.edu.sg).

L. Y. Zhai is with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798. (e-mail: lyzhai@ntu.edu.sg).

X. Li is with Singapore Institute of Manufacturing Technology, 71 Nanyang Drive, Singapore 638075 (e-mail: xli@SIMTech.a-star.edu.sg).

L. San is with the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798 (e-mail: sanlinn@ntu.edu.sg).

highlighted. Major contributions of this work are summarized in Section VI.

II. RELATED WORKS

The problem of designing fast on-line learning algorithms for practical implementation of neural networks remains an active research topic due to the large computation cost required for the learning process. In this section, a brief review of related works concerning various learning methods for SLFNs is presented.

A. Heuristic Learning of SLFNs

For optimum statistical classification and generalization with SLFN models, two tasks must be performed, namely learning the best set of weights for a network of *k* hidden units and determining *k*, the best complexity fit [11]. In this regard, Indurkhya and Weiss [12] compared two approaches to construction of neural network classifiers, one of which is the standard back-propagation approach applied to a series of SLFNs with differing number of hidden units, and the other one is the heuristic cascade-correlation approach that can quickly and dynamically configure the hidden units in a network and learn the best set of weights for it. Real-world applications conducted in their research show that the back-propagation approach yields somewhat better results, but with far greater computation times. The best complexity fit, *k*, for both approaches were quite similar. Based on their research, they proposed a hybrid approach to construct SLFN classifiers in which the number of hidden units was determined by cascade-correlation and the weights were learned by back-propagation. It should be noted that the selection of weights of the new hidden units for sequential feed-forward neural networks usually involves a non-linear optimization problem that cannot be solved analytically in the general case. In most cases, a suboptimal solution is searched heuristically. Most models found in the literature choose the weights in the first layer that correspond to each hidden unit so that its associated output vector matches the previous residue as much as possible. In this regard, Romero and Alquézar [13] recently conducted an experimental study to select the weights for sequential feed-forward neural networks. Their results indicated that the orthogonalization of the output vectors of the hidden units outperformed the strategy of matching the residue, both for approximation and generalization purposes.

On the other hand, variational Bayesian (VB) methods (also known as Bayesian ensemble learning methods) provide another alternative approach to various learning problems. VB learning techniques are based on approximating the true posterior probability density of the unknown variables of the model by a function with a restricted form and simpler distribution. In VB methods, the most common technique is ensemble learning where the approximation is fitted to the exact posterior distribution and Kullback–Leibler divergence is used to measure the misfit (or difference) between the approximation and the true posterior distribution [14, 15]. In other words, the basic idea of VB learning methods is to simultaneously approximate the intractable joint distribution over both hidden states and parameters with a simpler distribution, usually by assuming the hidden states and parameters are independent. However, as pointed out by some researchers [16], the choice of the posterior approximation form for the sources and the mixing matrix has a significant effect on the final solution. In fact, probabilistic methods such as Gaussian processes and support vector machines usually suffer from the problem of model mismatch. In such a context, this paper attempts to explore some other alternatives to improve the online sequential leaning process based on established sequential learning algorithms such as MRAN and ELM.

B. Minimal Resource Allocation Network (MRAN)

Since the late eighties, there has been considerable interest in RBF neural networks due to their good global generalization ability and a simple network structure that avoids lengthy calculations [17]. A number of algorithms have been proposed for training the RBF network [18, 19]. The classical approach to RBF implementation is to fix the number of hidden neurons a priori along with its centres and widths based on some properties of the input data and then estimate the weights connecting the hidden and output neurons [20]. Two methods have been proposed to find the proper number of hidden neurons for a given problem. Lee and Kil [21] introduced the concept of building up the hidden neurons from zero to the required number with the update of the RBF parameters being done by a gradient descent algorithm. An alternative approach is to start with as many as hidden units as the number of inputs and then reduce them using a clustering algorithm which essentially puts patterns that are close in the input space into a cluster so as to remove unnecessary hidden neurons [22]. However, in all these studies, the main learning scheme is of batch type, which is not suitable for on-line learning. In 1991, Platt [5] proposed a sequential learning algorithm to remedy the above drawbacks. In Platt's RAN algorithm, hidden neurons are added based on the novelty of the new data and the weights connecting the hidden neurons to the output neurons are estimated using the least mean square method. Platt showed that the resulting network topology is more parsimonious than the classical RBF networks. Kadirkamanathan [6] proposed modifications to improve RAN by using an EKF instead of the least mean squares method to estimate the network parameters. The resulting network called RANEKF is more compact and has better accuracy than RAN. A further improvement to RAN and RANEKF was proposed by [6] in which a pruning strategy was introduced to remove those neurons that consistently made little contributions to the network output. The resulting network, MRAN, was shown to be more compact than RAN and RANEKF for several applications in the areas of function approximation and pattern classification [23].

MRAN is a sequential learning algorithm that realises a minimal RBF neural network structure by combining the growth criteria of RAN with a pruning strategy [6, 23]. This algorithm is an improvement to the RAN of [7] and the RANEKF algorithm of [6]. Fig. 1 is a schematic illustration of the basic principles of the MRAN algorithm [24].

Briefly speaking, the learning process of MRAN involves the allocation of new hidden nodes as well as the adjustment of network parameters. In MRAN algorithms, Gaussian functions are usually selected in majority of cases as radial basis functions even though other functions like thin plate functions can also be used [25]. The objective of learning is to determine the two parameters of the Gaussian function used, namely the centre and width of the function. There are three parameters associated with each hidden node in the network, namely the centre ($\boldsymbol{\mu}$), the width (σ) and

the connecting weights (α) to the output units. The centres, widths, and weights of the hidden neurons are adjusted using an extended Kalman filter (EKF). Mathematically, the output of an MRAN equalizer has the following form:

$$f(\mathbf{x}) = \sum_{k=1}^{K} \alpha_k \phi_k(\mathbf{x}) \quad (1)$$

where $\mathbf{x}$ is the input vector of the network, *K* indicates the total number of hidden neurons, α_k is the connection weight of the *k*th hidden neuron to the output unit and $\phi(\mathbf{x})$ is a Gaussian function given by

$$\phi_k(\mathbf{x}) = \exp\left(-\frac{1}{\sigma_k^2}\|\mathbf{x}-\boldsymbol{\mu}_\mathbf{k}\|^2\right) \quad (2)$$

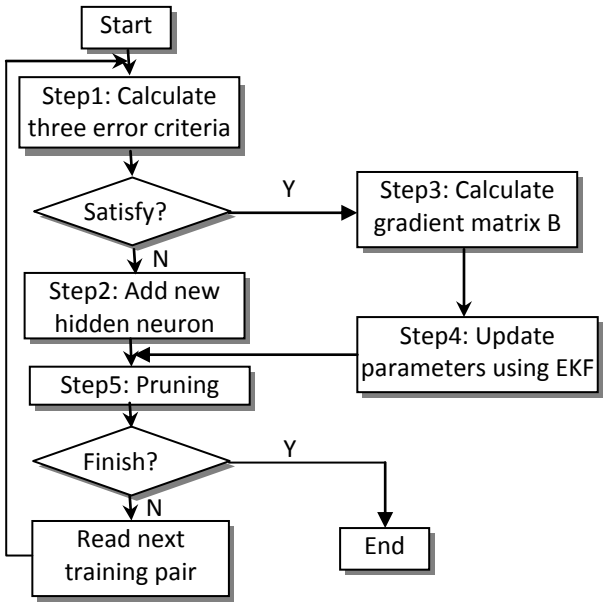


Fig. 1 Principles of MRAN

Here, $\boldsymbol{\mu}_k$ and σ_k refer to the center and width of the *k*th hidden neuron and $\|\ \|$ denotes the Euclidean norm. Notice that in Eqn. (1), the bias value of the system is assumed to be zero.

In the MRAN algorithm, the network starts with no hidden units. As the training process progresses, the algorithm adds hidden units and adjusts existing network parameters. More specifically, as each input-output training data (i.e. observation) $(\mathbf{x}_n, y_n)$ is received, the network is built up based on certain growth criteria. In the MRAN algorithm, the following three criteria must be met for an observation $(\mathbf{x}_n, y_n)$ to be used to generate a new hidden node in the network:

$$e_n = \| y_n - f(\mathbf{x}_n) \| > e_{\min} \quad (3)$$

$$\|\mathbf{x}_n - \boldsymbol{\mu}_{nr}\| > \varepsilon_n \quad (4)$$

$$e_{rmsn} = \sqrt{\frac{\sum_{i=n-(M-1)}^{n} e_i^2}{M}} > e'_{\min} \quad (5)$$

where e_n is the error between the network output ($f(\mathbf{x}_n)$) and the target output (y_n), μ_{nr} is the centre of the hidden node that is nearest to the current input $\mathbf{x}_n$, *M* is the total number of the past outputs of the network, and $e_{\min}$, ε_n , and $e'_{\min}$ are thresholds to be selected based on specific applications.

To be more precise, the first criterion (Eqn. 3) guarantees that the error between the network output and the target output must be significant. Here, $e_{\min}$ is an instantaneous error that is used to determine if existing nodes are insufficient to obtain a network output. The second criterion (Eqn. 4) guarantees that the input must be far from all the centres of current nodes. The term ε_n here ensures that the new node added is sufficiently far from all existing nodes in the network. The third criterion (Eqn. 5) guarantees that the network meets the required sum squared error specification for the past network outputs. The term $e'_{\min}$ ensures that the RMS value of the output error (e_{rms}) over a sliding data window (M) is significant. When a new observation arrives, the data in this window are updated by replacing the oldest one with the latest entry. This additional condition is introduced here to overcome the problem of over-fitting due to the effect of noise and ensure that the growth of hidden nodes in the network remains smooth and reasonable. When a new hidden node is added to the network, the parameters associated with this new node can be calculated as follows:

$$\alpha_{K+1} = e_n \quad (6)$$

$$\boldsymbol{\mu}_{K+1} = \mathbf{x}_n \quad (7)$$

$$\sigma_{K+1} = \kappa \|\mathbf{x}_n - \boldsymbol{\mu}_{nr}\| \quad (8)$$

where κ is an overlap factor that determines the overlap of hidden unit responses in the input space.

If the observation does not meet the criteria for adding a new hidden node, the network parameter vector $w = [\alpha_0, \alpha_1, \boldsymbol{\mu}_1^T, \sigma_1, \ldots, \alpha_K, \boldsymbol{\mu}_K^T, \sigma_K]^T$ is updated using the Extended Kalman Filter (EKF) as follows:

$$\mathbf{w}^{(n)} = \mathbf{w}^{(n-1)} + e_n \mathbf{k}_n \quad (9)$$

where $\mathbf{k}_n$ is the Kalman gain vector given by

$$\mathbf{k_n} = \left[\mathbf{R_n} + \mathbf{a_n^T P_{n-1} a_n}\right]^{-1} \mathbf{P_{n-1} a_n} \quad (10)$$

and R_n is the variance of the measurement noise and $\mathbf{a}_n$ is the gradient vector which has the following form:

$$\mathbf{a}_n = \left[\phi_1(\mathbf{x}_n)\frac{2\alpha_1}{\sigma_1^2}(\mathbf{x}_n - \boldsymbol{\mu}_1)^T, \phi_1(\mathbf{x}_n)\frac{2\alpha_1}{\sigma_1^3}\|\mathbf{x}_n - \boldsymbol{\mu}_1\|^2, \ldots, \phi_K(\mathbf{x}_n)\frac{2\alpha_K}{\sigma_K^2}(\mathbf{x}_n - \boldsymbol{\mu}_K)^T, \phi_K(\mathbf{x}_n)\frac{2\alpha_K}{\sigma_K^3}\|\mathbf{x}_n - \boldsymbol{\mu}_K\| \right]^T \quad (11)$$

The error covariance matrix $\mathbf{P}_n$ is updated by

$$\mathbf{P_n} = \left[\mathbf{I} - \mathbf{k_n a_n^T}\right]\mathbf{P_{n-1}} + \mathbf{Q_0 I} \quad (12)$$

where Q_0 is a scalar that determines the allowed random step in the direction of the gradient vector. The error

covariance matrix $\mathbf{P}_n$ is a $P \times P$ positive definite symmetric matrix where P is the number of parameters to be adapted. Whenever a new hidden node is allocated, the dimension of $\mathbf{P}_n$ increases and the new rows and columns must be initialized as follows:

$$\mathbf{P}_n = \begin{pmatrix} \mathbf{P}_{n-1} & 0 \\ 0 & P_0\mathbf{I} \end{pmatrix} \tag{13}$$

The new rows and columns are initialized by P_0, which is an estimate of uncertainties in the initial values assigned to the parameters.

During the learning process, although active initially, some hidden nodes may subsequently end up contributing little to the network output. In order to determine whether a hidden neuron should be removed or not, the output values for the hidden nodes given by

$$o_k = \alpha_k \exp\left\{-\frac{1}{\sigma_k^2}\left\|\mathbf{x}_n - \boldsymbol{\mu}_{nr}\right\|^2\right\} \tag{14}$$

are examined continuously. If the output of a hidden node is less than a threshold over a number of *M* consecutive inputs, that hidden node is removed from the network. This is called the pruning process of the MRAN algorithm. In order to eliminate the problem of inconsistency, the normalized output values of hidden nodes are used in the pruning process. This pruning strategy can be illustrated as follows:

Step 1 For every observation $(\mathbf{x_n}, y_n)$, compute the outputs of all hidden nodes o_k^n $(k = 1,\ldots,K)$ using Eqn. (14).

Step 2 Find the largest absolute hidden node output value $\left\|o_{\max}^n\right\|$ and compute the normalized output values $r_k^n = \left\|\frac{o_k^n}{o_{\max}^n}\right\|$.

Step 3 Remove the hidden nodes for which the normalized output is less than a threshold δ for *M* consecutive observations.

Step 4 Adjust the dimension of the EKF with respect to the reduced network.

C. Online Sequential Extreme Learning Machine (OS-ELM)

Since its inception, Online Sequential Extreme Learning Machine (OS-ELM) has shown to be successful in many cases of online learning with varying training data sets. The OS-ELM is a sequential learning algorithm based on SLFN with RBF neurons. The output of a standard SLFN with $\tilde{N}$ hidden nodes and activation function $G(\mathbf{x})$ is governed by

$$f_{\tilde{N}}(\mathrm{x}) = \sum_{\mathrm{i}=1}^{\tilde{\mathrm{N}}} \beta_i \, G(\mathbf{a}_i, b_i, \mathbf{x}) \qquad \mathbf{x} \in \mathrm{R}^{\mathrm{n}}, \quad \mathbf{a}_{\mathrm{i}} \in \mathrm{R}^{\mathrm{n}} \tag{15}$$

where $\mathbf{a}_i$ is the vector of input-to-hidden weights, β_i is the vector of hidden-to-output weights, b_i is the bias value of the *i*th hidden node and $G(\mathbf{a}_i, b_i, \mathbf{x})$ is the output of the *i*th hidden node with respect to the input **x**.

Given a set of initial training data, $\breve{\mathrm{N}}_0 = \{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=1}^{N_0}$ ($N_0 \geq \tilde{N}$), where $\mathbf{x}_i$ is the input vector and $\mathbf{t}_i$ is the corresponding desired output vector, the batch ELM algorithm is transformed into the problem of minimizing $\|\mathbf{H}_0\boldsymbol{\beta} - \mathbf{T}_0\|$ where

$$\mathbf{H_0} = \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_1) & \cdots & G(\mathbf{a}_{\tilde{N}}, b_{\tilde{N}}, \mathbf{x}_1) \\ \vdots & \ddots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_{N_0}) & \cdots & G(\mathbf{a}_{\tilde{N}}, b_{\tilde{N}}, \mathbf{x}_{N_0}) \end{bmatrix}_{N_0 \times \tilde{N}}$$

The solution to minimize $\|\mathbf{H}_0\boldsymbol{\beta} - \mathbf{T}_0\|$ is given by

$$\boldsymbol{\beta}^{(0)} = \mathbf{K}_0^{-1}\mathbf{H}_0^T\mathbf{T}_0 \tag{16}$$

where $\mathbf{K}_0 = \mathbf{H}_0^T\mathbf{H}_0$

The batch ELM algorithm described above assumes that all the training data are available for training. However, in real-world applications, the training data may arrive chunk-by-chunk or one-by-one. Therefore, the batch ELM algorithm has to be modified and extended to accommodate online sequential learning. In online sequential learning process, when another block of data $\breve{\mathrm{N}}_1 = \{(\mathbf{x}_i, \mathbf{t}_i)\}_{i=N_0+1}^{N_0+N_1}$ is received, where N_1 denotes the number of observations in this block, the problem becomes minimizing

$$\left\|\begin{bmatrix}\mathbf{H}_0 \\ \mathbf{H}_1\end{bmatrix}\boldsymbol{\beta} - \begin{bmatrix}\mathbf{T}_0 \\ \mathbf{T}_1\end{bmatrix}\right\| \tag{17}$$

where

$$\mathbf{H}_1 = \begin{bmatrix} G(\mathbf{a}_1, b_1, \mathbf{x}_{N_0+1}) & \cdots & G(\mathbf{a}_{\tilde{N}}, b_{\tilde{N}}, \mathbf{x}_{N_0+1}) \\ \vdots & \ddots & \vdots \\ G(\mathbf{a}_1, b_1, \mathbf{x}_{N_0+N_1}) & \cdots & G(\mathbf{a}_{\tilde{N}}, b_{\tilde{N}}, \mathbf{x}_{N_0+N_1}) \end{bmatrix}_{N_1 \times \tilde{N}}$$

Considering both blocks of training data sets $\breve{\mathrm{N}}_0$ and $\breve{\mathrm{N}}_1$, the output weight $\boldsymbol{\beta}$ becomes

$$\boldsymbol{\beta}^{(1)} = \mathbf{K}_1^{-1}\begin{bmatrix}\mathbf{H}_0 \\ \mathbf{H}_1\end{bmatrix}^T\begin{bmatrix}\mathbf{T}_0 \\ \mathbf{T}_1\end{bmatrix} \tag{18}$$

Where

$$\mathbf{K_1} = \left[\mathbf{H_0^T}\mathbf{H_1^T}\right]\begin{bmatrix}\mathbf{H_0} \\ \mathbf{H_1}\end{bmatrix} = \mathbf{K_0} + \mathbf{H_1^T}\mathbf{H_1} \tag{19}$$

and

$$\begin{bmatrix}\mathbf{H}_0 \\ \mathbf{H}_1\end{bmatrix}^T\begin{bmatrix}\mathbf{T}_0 \\ \mathbf{T}_1\end{bmatrix} = \mathbf{K}_1\boldsymbol{\beta}^{(0)} - \mathbf{H}_1^T\mathbf{H}_1\boldsymbol{\beta}^{(0)} + \mathbf{H}_1^T\mathbf{T}_1 \tag{20}$$

Note that in Eqn. (20), $\mathbf{H}_0$ and $\mathbf{K}_0$ are replaced by the function of $\mathbf{H}_1$ and $\mathbf{K}_1$ so that they will not appear in the expression for $\boldsymbol{\beta}^{(1)}$ and could be removed from the memory during computation. By combining Eqn. (18) and Eqn. (20), $\boldsymbol{\beta}^{(1)}$ is obtained by

$$\begin{aligned}\boldsymbol{\beta}^{(1)} &= \mathbf{K}_1^{-1}\begin{bmatrix}\mathbf{H}_0 \\ \mathbf{H}_1\end{bmatrix}^T\begin{bmatrix}\mathbf{T}_0 \\ \mathbf{T}_1\end{bmatrix} \\ &= \boldsymbol{\beta}^{(0)} + \mathbf{K}_1^{-1}\mathbf{H}_1^T(\mathbf{T}_1 - \mathbf{H}_1\boldsymbol{\beta}^{(0)})\end{aligned} \tag{21}$$

III. LIMITATIONS OF OS-ELM

In order to evaluate the performance of the OS-ELM and the MRAN algorithm, some simulation studies have been conducted using the regression benchmark problems described in [10]. The results are shown in Table I.

Table I Evaluation of OS-ELM and MRAN

Dataset	Algorithms	Learning Mode	Number of nodes	Average Training Time	Average Testing Time	Average Training RMSE	Average Testing RMSE
Auto-MPG	OS-ELM	1 by 1	25	0.0531	0.0075	0.0659	0.0772
	OS-ELM	20 by 20	25	0.0091	0.0081	0.0657	0.0778
	MRAN	1 by 1	4.7(1.9)	1.3359	0.0484	0.1234	0.1339
Abalone	OS-ELM	1 by 1	25	0.6853	0.0228	0.0753	0.0776
	OS-ELM	20 by 20	25	0.0709	0.0209	0.0753	0.0775
	MRAN	1 by 1	13.6(1.9)	17.4	0.7688	0.0784	0.0805
California Housing	OSELM	1 by 1	50	6.6456	0.4766	0.1297	0.1308
	OSELM	20 by 20	50	0.6672	0.4763	0.1293	0.1311
	MRAN	1 by 1	7.7(2.2)	63.025	2.1516	0.1461	0.1452

Simulation results are comparable to the results shown in [10]. Note that column four in Table I shows the average number of nodes generated by each algorithm. For MRAN, the standard deviation of the node number generated in 50 times of the simulation studies is also calculated and indicated in the parentheses. It is clear that the performance of the OS-ELM is much faster than that of the MRAN algorithm. However, the OS-ELM requires large number of hidden nodes compared to the MRAN. This will result in excessive testing time and sometimes it will also lead to a complex and intricate neural network. Table I also shows another advantage of the OS-ELM, i.e. the OS-ELM can achieve a lower testing Root Mean Square Error (RMSE), especially for the first two problems listed in Table I.

To further evaluate the performance of the OS-ELM, a simple Gaussian function approximation problem is introduced here to gain some insights into its mechanism of node generation. The simple nonlinear Gaussian function is governed by

$$y(\mathbf{x}) = \exp[-\frac{(x_1-0.3)^2+(x_2-0.2)^2}{0.01}] \quad (x_1, x_2 \in [-1, 1]) \tag{22}$$

In order to investigate the effect of node selection on the performance of the OS-ELM, a series of simulation studies have been conducted using the OS-ELM algorithm with 4 hidden nodes and RBF activation function. Table II presents two simulation results with different hidden node selections and their regression results for Eqn. (22) are plotted in Fig. 2.

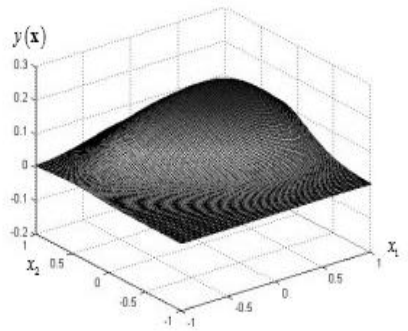
As shown in Table II, although Simulations (a) and (b) produce similar performance in terms of training and testing accuracy, the resulting output weight for each node varies. This is because, in the OS-ELM algorithm, the initial nodes are randomly selected with different centres and bias values, and their contributions to the regression process are different, which is reflected by the values of their weights. For example, the output weight of the first node in Simulation (a) is very small, which means that the node contributes very little to the entire network output. In fact, this can be understood from the mathematical point of view as the centre selected for this node is far from the actual centre of the approximation function.

Another interesting observation from Table II is that the last two nodes in Simulation (b) are quite close to each other, but their weights are very different – node 4 has a weight of 0.1199 while the output weight of node 3 is -0.2033. This is because when node 4 attempts to approach the actual peak value at (0.3, 0.2), node 3 has to balance the error caused by node 4 in the local region and eventually the weight of node 3 evolves into a negative value.

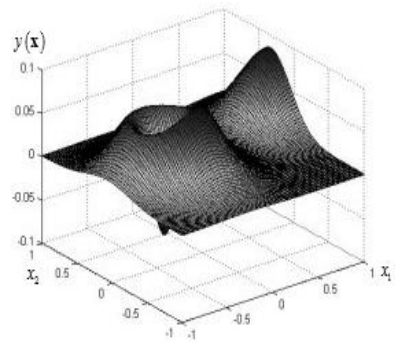
Table II OS-ELM Performances for the Gaussian Function Approximation Problem

S/N	Parameter	Node 1	Node 2	Node 3	Node 4
(a)	Centre	(0.962, 0.352)	(0.442,0.399)	(0.621,0.840)	(-0.389,0.480)
	Width	0.3276	0.4227	0.1531	0.1672
	Weight	0.0042	0.2536	-0.2915	0.0724
	Training Accuracy: 0.0726; Testing Accuracy: 0.0607				
(b)	Centre	(0.997, 0.340)	(-0.259, -0.319)	(-0.383,0.051)	(-0.387,0.079)
	Width	0.1566	0.1495	0.038	0.1798
	Weight	0.0842	0.0403	-0.2033	0.1199
	Training Accuracy: 0.0614; Testing Accuracy: 0.0547				

Fig. 2 shows the final results (regression functions) obtained from the two simulation studies. Obviously, Simulations (a) and (b) produce two different regression functions although the same objective function is given by Eqn. (22). In fact, the OS-ELM will generate a different form of regression function in each round of the simulation studies, depending on the initial node centres and bias values selected. In Fig. 2, neither (a) nor (b) has the same form of regression function as the given formula in Eqn. (22) although they can achieve very high and similar accuracy.



(a)



(b)

Fig.2 Regression performance of OS-ELM with 4 nodes

The above analysis of the simulation results reveals some limitations of the original OS-ELM algorithm. As the centres and width of the nodes cannot be adjusted in the

process, in order to get a more accurate resulting regression function, the OS-ELM usually needs to assign far more number of nodes than necessary to guarantee that at least one node is located near the actual peak location. This is also in line with the observation in Table I where the OS-ELM always needs many more nodes compared to the MRAN algorithms. Recently, another sequential learning algorithm has been proposed in [26, 27]. Their algorithm incorporates the idea of on-line structural adaptation to add new hidden neurons and the method uses an error sensitive clustering algorithm to adapt the centre and width of the hidden neurons. The algorithm known as On-line Structural Adaptive Hybrid Learning (ONSAHL) is shown in [27] to produce compact networks for nonlinear dynamic system identification problems. Inspired by their work, this paper attempts to explore another alternative to improve the learning process by combining the advantages of MRAN and ELM, which will be elaborated in the next section.

IV. HYBRID OS-ELM

In order to circumvent the limitations of the aforementioned OS-ELM, a hybrid OS-ELM (HOS-ELM) algorithm is proposed by integrating the advantage of the OS-ELM in speed and the merit of MRAN in node selection strategy. In MRAN algorithm, the parameters of the network including the centres, widths and weights of the hidden neurons have to be updated in every step. This results in significant increase in the size of the matrices to be updated as the number of hidden neurons increases and the RBF network structure becomes more complex computationally, which directly results in a large computation load and limits the use of MRAN for real-time implementation. Here, it should be noted that although some research works have been done to enhance capabilities of MRAN in recent years [28], the critical characteristics of MRAN, namely less number of hidden neurons and lower approximation errors, are still retained. For example, in the work done in [28], the extended MRAN (EMRAN) algorithm still retains the same form as MRAN and all the equations are the same except the adoption of the 'winner neuron' strategy to reduce the on-line computation load and avoid memory overflow. For simplicity, but without loss of generality, this research will use the basic form of MRAN to reduce the number of hidden nodes in the proposed algorithm. In the proposed HOS-ELM algorithm, the centres and widths of the nodes can be adjusted based on the EKF theory and new nodes will be added when the MRAN criteria are met. In the sequel, theoretical basis pertaining to the addition of new nodes in the SLFN network will be introduced before the HOS-ELM algorithm is described.

A. Strategy for Adding New Nodes in SLFN

As mentioned earlier, the *i*th column of $\mathbf{H}$ is the *i*th hidden node's output vector with respect to inputs $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N$. Therefore, when new hidden nodes are added, the number of columns in $\mathbf{H}$ needs to be expanded too. Assume that a SLFN starts with a few hidden nodes, which approximates an initial set of data sample $\breve{N}_0$ with small errors. After another block of data $\breve{N}_1$ arrives, $\boldsymbol{\beta}^{(1)}$ and $\mathbf{P}_1$ will be calculated respectively. However, the error $\left\| \begin{bmatrix} \mathbf{H} \\ \mathbf{H} \end{bmatrix} \boldsymbol{\beta}^{(1)} - \begin{bmatrix} \mathbf{T} \\ \mathbf{T}_1 \end{bmatrix} \right\|$ might be larger than the desired value. As one of the possible solutions, some new hidden nodes need to be added to the SLFN. To simplify the above process, it is assumed that each time there is only one node added and the hidden node parameters are assigned through the method of MRAN. Hence, the problem is equivalent to minimizing

$$\left\| \begin{bmatrix} \mathbf{H}_0 & \mathbf{Z}_0 \\ \mathbf{H}_1 & \mathbf{Z}_1 \end{bmatrix} \hat{\boldsymbol{\beta}}^{(1)} - \begin{bmatrix} \mathbf{T}_0 \\ \mathbf{T}_1 \end{bmatrix} \right\| \tag{23}$$

where

$$\mathbf{Z}_0 = \begin{bmatrix} G(\mathbf{a}_{\tilde{N}+1}, b_{\tilde{N}+1}, \mathbf{x}_1) \\ \vdots \\ G(\mathbf{a}_{\tilde{N}+1}, b_{\tilde{N}+1}, \mathbf{x}_{N_0}) \end{bmatrix}_{N_0 \times 1}$$

$$\mathbf{Z}_1 = \begin{bmatrix} G(\mathbf{a}_{\tilde{N}+1}, b_{\tilde{N}+1}, \mathbf{x}_{N_0+1}) \\ \vdots \\ G(\mathbf{a}_{\tilde{N}+1}, b_{\tilde{N}+1}, \mathbf{x}_{N_0+N_1}) \end{bmatrix}_{N_1 \times 1}$$

$$\hat{\boldsymbol{\beta}}^{(1)} = \begin{bmatrix} \boldsymbol{\beta}_1^T \\ \vdots \\ \boldsymbol{\beta}_{\tilde{N}}^T \\ \boldsymbol{\beta}_{\tilde{N}+1}^T \end{bmatrix}_{(\tilde{N}+1)\times \mathrm{m}}$$

Here, $\mathbf{Z}_0$ and $\mathbf{Z}_1$ are the new hidden node output matrix with respect to the two data sample set $\breve{N}_0$ and $\breve{N}_1$. The terms $\mathbf{a}_{\tilde{N}+1}$ and $b_{\tilde{N}+1}$ are the new hidden node parameters and $\boldsymbol{\beta}_{\tilde{N}+1}^T$ is the output weight vector connecting the new hidden node to the output nodes. Similar to Eqn. (18), the new output weight matrix is estimated as follows:

$$\hat{\boldsymbol{\beta}}^{(1)} = \hat{\mathbf{K}}_1^{-1} \begin{bmatrix} \mathbf{H}_0 & \mathbf{Z}_0 \\ \mathbf{H}_1 & \mathbf{Z}_1 \end{bmatrix}^T_{(N_0+N_1)\times(\tilde{N}+1)} \begin{bmatrix} \mathbf{T}_0 \\ \mathbf{T}_1 \end{bmatrix}_{(N_0+N_1)\times m} \tag{24}$$

where

$$\hat{\mathbf{K}}_1 = \begin{bmatrix} \mathbf{H}_0 & \mathbf{Z}_0 \\ \mathbf{H}_1 & \mathbf{Z}_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{H}_0 & \mathbf{Z}_0 \\ \mathbf{H}_1 & \mathbf{Z}_1 \end{bmatrix}$$

Applying the architecture rules for the block matrix and using $\mathbf{K}_1 = \mathbf{K}_0 + \mathbf{H}_1^T \mathbf{H}_1$ to substitute $\mathbf{K}_0$, we have

$$\hat{\mathbf{K}}_1 = \begin{bmatrix} \mathbf{H}_0 & \mathbf{Z}_0 \\ \mathbf{H}_1 & \mathbf{Z}_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{H}_0 & \mathbf{Z}_0 \\ \mathbf{H}_1 & \mathbf{Z}_1 \end{bmatrix}$$

$$= \begin{bmatrix} (\mathbf{K}_1)_{\tilde{N}\times\tilde{N}} & (\mathbf{H}_0^T\mathbf{Z}_0 + \mathbf{H}_1^T\mathbf{Z}_1)_{\tilde{N}\times 1} \\ (\mathbf{Z}_0^T\mathbf{H}_0 + \mathbf{Z}_1^T\mathbf{H}_1)_{1\times\tilde{N}} & (\mathbf{Z}_0^T\mathbf{Z}_0 + \mathbf{Z}_1^T\mathbf{Z}_1)_{1\times 1} \end{bmatrix}_{(\tilde{N}+1)\times(\tilde{N}+1)} \tag{25}$$

and

$$\begin{bmatrix} \mathbf{H}_0 & \mathbf{Z}_0 \\ \mathbf{H}_1 & \mathbf{Z}_1 \end{bmatrix}^T \begin{bmatrix} \mathbf{T}_0 \\ \mathbf{T}_1 \end{bmatrix} = \begin{bmatrix} \mathbf{K}_0 \boldsymbol{\beta}^{(0)} + \mathbf{H}_1^T \mathbf{T}_1 \\ \mathbf{Z}_0^T \mathbf{T}_0 + \mathbf{Z}_1^T \mathbf{T}_1 \end{bmatrix} \tag{26}$$

In order to obtain $\hat{\mathbf{K}}_1^{-1}$, the block matrix inverse theorem in [29, 30] is used here, which is given by

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}\mathbf{S}_A^{-1}\mathbf{C}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}\mathbf{S}_A^{-1} \\ -\mathbf{S}_A^{-1}\mathbf{C}\mathbf{A}^{-1} & \mathbf{S}_A^{-1} \end{bmatrix} \quad (27)$$

where $\mathbf{S}_A = \mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}$ is called the Schur complement of $\mathbf{A}$.

Here, $\mathbf{Z}_0$ is the new hidden node output matrix with respect to the original data sample set. By applying the RBF node, as long as the new node is far from other nodes, the value in $\mathbf{Z}_0$ should approach zero so that it is approximately a zero matrix. Based on the block inverse theorem and $\mathbf{P}_1 = \mathbf{K}_1^{-1}$, $\hat{\mathbf{K}}_1^{-1}$ becomes

$$\begin{aligned} \hat{\mathbf{K}}_1^{-1} &= \begin{bmatrix} (\mathbf{K}_1)_{\tilde{N}\times\tilde{N}} & (\mathbf{H}_1^T\mathbf{Z}_1)_{\tilde{N}\times 1} \\ (\mathbf{Z}_1^T\mathbf{H}_1)_{1\times\tilde{N}} & (\mathbf{Z}_1^T\mathbf{T}_1)_{1\times 1} \end{bmatrix}^{-1} \\ &= \begin{bmatrix} \mathbf{K}_1^{-1} + \mathbf{K}_1^{-1}\mathbf{H}_1^T\mathbf{Z}_1\mathbf{S}_A^{-1}\mathbf{Z}_1^T\mathbf{H}_1\mathbf{K}_1^{-1} & -\mathbf{K}_1^{-1}\mathbf{H}_1^T\mathbf{Z}_1\mathbf{S}_A^{-1} \\ -\mathbf{S}_A^{-1}\mathbf{Z}_1^T\mathbf{H}_1\mathbf{K}_1^{-1} & \mathbf{S}_A^{-1} \end{bmatrix} \\ &= \begin{bmatrix} \mathbf{P}_1 + \mathbf{P}_1\mathbf{H}_1^T\mathbf{Z}_1\mathbf{S}_A^{-1}\mathbf{Z}_1^T\mathbf{H}_1\mathbf{P}_1 & -\mathbf{P}_1\mathbf{H}_1^T\mathbf{Z}_1\mathbf{S}_A^{-1} \\ -\mathbf{S}_A^{-1}\mathbf{Z}_1^T\mathbf{H}_1\mathbf{P}_1 & \mathbf{S}_A^{-1} \end{bmatrix} \end{aligned} \quad (28)$$

where

$$\mathbf{S}_A = (\mathbf{Z}_1^T\mathbf{Z}_1)_{1\times 1} - (\mathbf{Z}_1^T\mathbf{H}_1)_{1\times\tilde{N}}(\mathbf{P}_1)_{\tilde{N}\times\tilde{N}}(\mathbf{H}_1^T\mathbf{Z}_1)_{\tilde{N}\times 1} \quad (29)$$

Note that $\mathbf{S}_A$ is a scalar value.

Combining Eqn. (26) and Eqn. (28), $\hat{\boldsymbol{\beta}}^{(1)}$ can be computed as follows:

$$\begin{aligned} \hat{\boldsymbol{\beta}}^{(1)} &= \hat{\mathbf{K}}_1^{-1}\begin{bmatrix} \mathbf{H}_0 & \mathbf{Z}_0 \\ \mathbf{H}_1 & \mathbf{Z}_1 \end{bmatrix}^T_{(N_0+N_1)\times(\tilde{N}+1)}\begin{bmatrix} \mathbf{T}_0 \\ \mathbf{T}_1 \end{bmatrix}_{((N_0+N)\times m} \\ &= \begin{bmatrix} \boldsymbol{\beta}^{(0)} + \mathbf{P}_1\mathbf{H}_1^T(\mathbf{I} + \mathbf{Z}_1\mathbf{S}_A^{-1}\mathbf{Z}_1^T(\mathbf{H}_1\mathbf{P}_1\mathbf{H}_1^T - \mathbf{I}))(\mathbf{T}_1 - \mathbf{H}_1\boldsymbol{\beta}^{(0)}) \\ -\mathbf{S}_A^{-1}\mathbf{Z}_1^T\mathbf{H}_1(\boldsymbol{\beta}^{(0)} - \mathbf{P}_1\mathbf{H}_1^T\mathbf{H}_1(\boldsymbol{\beta}^{(0)} + \mathbf{P}_1\mathbf{H}_1^T\mathbf{T}_1) + \mathbf{S}_A^{-1}\mathbf{Z}_1^T\mathbf{T}_1) \end{bmatrix} \end{aligned} \quad (30)$$

B. HOS-ELM Algorithm

As foreshadowed, the proposed HOS-ELM algorithm leverages on the advantages of MRAN and OS-ELM. In the process of adding new nodes, the three criteria listed in Eqn. (3-5) are sequentially checked. A new node will only be added when all the three criteria are met. However, in the case when the first criterion is not met, the HOS-ELM will apply the OS-ELM algorithm to adjust the output weights of the nodes in the following manner:

$$\boldsymbol{\beta}^{(k+1)} = \boldsymbol{\beta}^{(k)} + \mathbf{P}_{k+1}\mathbf{H}_{k+1}^T(\mathbf{T}_{k+1} - \mathbf{H}_{k+1}\boldsymbol{\beta}^{(k)}) \quad (31)$$

$$\mathbf{P}_{k+1} = \mathbf{P}_k - \mathbf{P}_k\mathbf{H}_{k+1}^T(\mathbf{I} + \mathbf{H}_{k+1}\mathbf{P}_k\mathbf{H}_{k+1}^T)^{-1}\mathbf{H}_{k+1}\mathbf{P}_k \quad (32)$$

Otherwise, the algorithm will check the second and third criteria. If both of them are met, a new hidden node will be added. The parameters associated with the new node are calculated based on Eqn. (6) and (7).

The output weight vector of the hidden nodes as well as the $\mathbf{P}$ value will then be updated using the formula developed in Eqn. (33) and (34).

$$\hat{\mathbf{P}}_{k+1} = \begin{bmatrix} \mathbf{P}_{k+1} + \mathbf{P}_{k+1}\mathbf{H}_{K+1}^T\mathbf{Z}_{k+1}\mathbf{S}_A^{-1}\mathbf{Z}_{k+1}^T\mathbf{H}_{k+1}\mathbf{P}_{k+1} & -\mathbf{P}_{K+1}\mathbf{H}_{K+1}^T\mathbf{Z}_{K+1}\mathbf{S}_A^{-1} \\ -\mathbf{S}_A^{-1}\mathbf{Z}_{K+1}^T\mathbf{H}_{K+1}\mathbf{P}_{K+1} & \mathbf{S}_A^{-1} \end{bmatrix} \quad (33)$$

$$\hat{\boldsymbol{\beta}}^{(k+1)} = \begin{bmatrix} \boldsymbol{\beta}^{(k)} + \mathbf{P}_{k+1}\mathbf{H}_{K+1}^T(\mathbf{I} - \mathbf{Z}_{k+1}\mathbf{S}_A^{-1}\mathbf{Z}_{k+1}^T(\mathbf{H}_{k+1}\mathbf{P}_{k+1}\mathbf{H}_{K+1}^T - \mathbf{I}))(\mathbf{T}_{K+1} - \mathbf{H}_{K+1}\boldsymbol{\beta}^{(k)}) \\ \mathbf{S}_A^{-1}\mathbf{Z}_{K+1}^T(\mathbf{T}_{K+1} - \mathbf{H}_{K+1}(\boldsymbol{\beta}^{(k)} - \mathbf{P}_{k+1}\mathbf{H}_{K+1}^T(\mathbf{H}_{K+1}\boldsymbol{\beta}^{(k)} - \mathbf{T}_{K+1}))) \end{bmatrix} \quad (34)$$

When the observation $(\mathbf{x}_n, y_n)$ does not meet the second or third criterion, the network parameters $\mathbf{w} = [\boldsymbol{\mu}_1^T, \sigma_1, \ldots, \boldsymbol{\mu}_K^T, \sigma_K]^T$ will be updated using the EKF. The procedure for adjusting $\mathbf{w}$ is as follows:

a) Calculate the gradient vector

$$\mathbf{a}_n = \left[\phi_1(\mathbf{x}_n)\frac{2\alpha_1}{\sigma_1^2}(\mathbf{x}_n - \boldsymbol{\mu}_1)^T, \phi_1(\mathbf{x}_n)\frac{2\alpha_1}{\sigma_1^3}\|\mathbf{x}_n - \boldsymbol{\mu}_1\|^2, \ldots, \phi_K(\mathbf{x}_n)\frac{2\alpha_K}{\sigma_K^2}(\mathbf{x}_n - \boldsymbol{\mu}_K)^T, \phi_K(\mathbf{x}_n)\frac{2\alpha_K}{\sigma_K^3}\|\mathbf{x}_n - \boldsymbol{\mu}_K\| \right]^T$$

b) Applying Eqn. (10) to generate the Kalman gain vector;

c) Update the parameters and the error covariance matrix using Eqn. (9) and (12);

d) Adjust the output weight in the way similar to the OS-ELM.

In the proposed HOS-ELM, the output weight vector and the $\mathbf{P}$ matrix are generated using the ELM algorithm and the pruning method used in MRAN is employed as shown in Section 2. In the next section, the HOS-ELM will be evaluated by a regression example.

V. EVALUATION AND DISCUSSIONS

In this section, two examples are used to evaluate the performance of the proposed HOS-ELM algorithm, including a Gaussian function approximation problem and a regression benchmark problem. The purpose of the evaluation focuses on a comparison of the complexity of the resulting networks and accuracy of approximation.

In the first example, the Gaussian function is governed by

$$y(\mathbf{x}) = \exp\left[-\frac{(x_1 - 0.3)^2 + (x_2 + 0.2)^2 + (x_3 - 0.6)^2 + (x_4 + 0.5)^2}{0.5}\right] \quad (35)$$

where the four-dimensional input vector $\mathbf{x}$ is randomly generated by MATLAB, and the range of each dimension is defined between (-1, 1).

The evaluation process is carried out in the following way. Firstly, the HOS-ELM was used to approximate the above objective function. For each simulation of the HOS-ELM, a set of parameters including simulation accuracy and the number of nodes will be obtained. Secondly, the ELM is employed to approximate the objective function using the same number of nodes as that of the HOS-ELM. In the experiment, a total number of 5000 samples are used for training and another 5000 samples are used for testing. Two simulation results are summarised in Table III.

Table III HOS-ELM Performance Evaluation using a Gaussian Approximation Example

S/N	Number of Nodes		Training Accuracy		Testing Accuracy	
	HOS-ELM	ELM	HOS-ELM	ELM	HOS-ELM	ELM
(a)	3	3	0.0992	0.1295	0.0967	0.1269
(b)	4	4	0.1011	0.1121	0.1026	0.1134

The results in Table III show that both training and testing accuracy of the proposed HOS-ELM are higher than that of OS-ELM when the same number of nodes is used. Further comparison between Simulations (a) and (b) shows that the performance of the OS-ELM degrades when the number of nodes decreases (e.g. from 4 to 3 in Table III). This implies that the proposed HOS-ELM algorithm is able

to reduce the number of nodes and thus simplify the network of the OS-ELM. This characteristic of the HOS-ELM will be an advantage in some online sequential learning processes where the size of the network is limited.

In order to further verify the performance of the proposed HOS-ELM algorithm, the regression benchmark problem of "Auto-MPG" is analysed as the second example. Table IV summarises a comparison of the performance of the OS-ELM, MRAN and the proposed HOS-ELM. On the one hand, the proposed HOS-ELM algorithm is able to obtain a more simplified network (i.e. reducing the number of hidden nodes from 25 to 16) while retaining comparable learning accuracy and learning speed when compared with the original OS-ELM algorithm. On the other hand, compared with the MRAN algorithm, the proposed HOS-ELM excels significantly in both learning speed and learning accuracy although the number of nodes required by the HOS-ELM is larger than that of the MRAN. The results tabulated in Table IV demonstrate that the proposed HOS-ELM algorithm has successfully leveraged on the advantages of the OS-ELM and the MRAN.

Table IV HOS-ELM Performance Evaluation using the Benchmark Example of "Auto-MPG"

Algorithms	Number of Nodes	Average Training Time	Average Testing Time	Average Training RMSE	Average Testing RMSE
OS-ELM	25	0.0531	0.0075	0.0659	0.0772
MRAN	4.7	1.3359	0.0484	0.1234	0.1339
HOS-ELM	16	0.0638	0.0081	0.0749	0.0872

VI. CONCLUSIONS

The OS-ELM is a powerful learning algorithm which is capable of generating neural networks with better generalisation performance. On the other hand, the MRAN provides a method to simulate the human being's learning process through step by step tuning. Based on a thorough analysis of the two algorithms, this paper proposed a novel algorithm known as Hybrid Online Sequential Extreme Learning Machine (HOS-ELM), leveraging on the advantages of the OS-ELM in speed and accuracy and the merits of the MRAN in node allocation, adjustment and pruning process. Results of evaluation have sown that the proposed HOS-ELM algorithm can achieve generalization performance comparable to the original OS-ELM and at the same time, the number of hidden nodes is significantly reduced resulting in a compact neural network structure.

REFERENCES

[1] R. Bustami, N. Bessaih, C. Bong, S. Suhaili, "Artificial Neural Network for Precipitation and Water Level Predictions of Bedup River", IAENG International Journal of Computer Science, vol.32, no.2, IJCS_34_2_10, 2010.

[2] T. Hacib, M.R. Mekideche, N. Ferkha, "Computational investigation on the use of FEM and RBF neural network in the inverse electromagnetic problem of parameter identification", *IAENG International Journal of Computer Science, vol. 33, no. 2, 18-24, 2007.*

[3] M.R. Jafari, T. Alizadeh, M. Gholami, A. Alizadeh, K. Salahshoor, "On-line Identification of Non-Linear Systems Using an Adaptive RBF-Based Neural Network", *World Congress on Engineering and Computer Science 2007 (WCECS 2007)*, San Francisco, USA, 2007.

[4] S. Nematipour, J. Shanbehzadeh, R.A. Moghadam, "Relevance Feedback Optimization in Content Based Image Retrieval Via Enhanced Radial Basis Function Network", *International MultiConference of Engineers and Computer Scientists 2011 (IMECS 2011), Hong Kong, pp. 539-542, 2011.*

[5] J. Platt, "A resource-allocating network for function interpolation", *Neural Computation, vol. 3, pp. 213-225, 1991.*

[6] V. Kadirkamanathan, M. Niranjan, "A function estimation approach to sequential learning with neural networks", *Neural Computation, vol. 5, pp. 954-975, 1993.*

[7] Y. Lu, N. Sundararajan, P. Saratchandran, "A sequential learning scheme for function approximation using minimal radial basis function (RBF) neural networks", *Neural Computation, vol. 9, pp. 461-478, 1997.*

[6] G.B. Huang, P. Saratchandran, N. Sundararajan, "An efficient sequential learning algorithm for growing and pruning RBF (GAP-RBF) networks", *IEEE Transactions on Systems, Man and Cybernetics, vol. 34, no. 6, pp. 2284-2292, 2004.*

[9] G.B. Huang, Q.Y. Zhu, C.K. Siew, "Extreme learning machine: theory and applications", *Neurocomputing, vol. 70, pp. 489-501, 2006.*

[10] N.Y. Liang, G.B. Huang, P. Saratchandran, "A fast and accurate online sequential learning algorithm for feedforward networks", *IEEE Transactions on Neural Netwoeks, vol. 17, no. 6, pp. 1411–1423, 2006.*

[11] D. Patra, M.K. Das, S. Pradhan, "Integration of FCM, PCA and Neural Networks for Classification of ECG Arrhythmias", IAENG International Journal of Computer Science, vol. 36, no. 3, 36_3_05, 2010.

[12] N. Indurkhya, S.M. Weiss, "Heuristic configuration of single hidden-layer feed-forward neural networks", *Applied Intelligence, vol. 2, no. 4, pp. 325-331, 1992.*

[13] E.Romero, R. Alquézar, "Heuristics for the selection of weights in sequential feed-forward neural networks: An experimental study", *Neurocomputing, vol. 70, no. 16-18, pp. 2735-2743, 2007.*

[14] H. Lappalainen, J. Miskin, "Ensemble learning", *Advances in Independent Component Analysis*, Springer-Verlag, pp.75-92, 2000.

[15] D.J.C. MacKay, *Information Theory, Inference, and Learning Algorithms*, Cambridge University Press, 2003.

[16] A.Ilin, H.Valpola, "On the Effect of the form of the posterior approximation in variational learning of ICA models", *Neural Processing Letters, vol. 22, pp.183–204, 2005.*

[17] S. Chen, S.A. Billings, C. Couan, P.M. Grant, "Practical Identification of NARMAX Models Using Radial Basis Function", *International Journal of Control, vol. 52, pp. 1327-1350, 1990.*

[18] C.L. Chen, W.C. Chen, F.Y. Chang, "Hybrid learning algorithm for Gaussian potential function networks", *Control Theory and Applications, vol. 140, no. 6, pp. 442-448, 1993.*

[19] J.Moody, C.J. Darken, "Fast learning in network of locally tuned processing units", *Neural Computation, vol. 1, pp. 281-294, 1989*

[20] D.N. Rao, Dr. M.R.K. Murthy, D.N. Harshal, S.R.M. Rao, "Computational Comparisons of GPC and NGPC Schemes", *Engineering Letters, vol. 14, no. 1, EL_14_1_19, 2007.*

[21] S. Lee, R.M.A. Kil, "Gaussian potential function network with hierarchically selforganizing learning", *Neural Networks, vol. 4, pp. 207-224, 2001.*

[22] M.T. Musavi, W. Ahmed, K.H. Chan, K.B. Faris, D.M. Hummels, "On training of Radial Basis Function classifiers", *Neural Networks, vol. 5,pp. 595-603, 1992*

[23] Y.W. Lu, N. Sundararajan, P.Saratchandran, "A sequential Minimal Radial Basis Function(RBF) neural network learning algorithm", *IEEE Transactions on Neural Networks, vol.9, no.2, pp. 308-318, 1998*

[21] S.C. Rogers, Use of MRAN adaptive neural network for control of a flexible system", *The International Society for Optical Engineering, vol. 5102, pp. 84-9, 2003.*

[25] S. Chen, S.A. Billings, P.M. Grant, "Nonlinear system identification using neural networks", *International Journal Control, vol. 51, pp. 1191–1214, 1990*

[26] T.F. Junge, H.Unbehauen, "Off-Line Identification of Nonlinear Systems Using Structurally Adaptive Radial Basis Function Networks", *The 35th Conference on Decision and Control*, pp. 943-948, Kobe, Japan, 1996

[27] T.F. Junge, H.Unbehauen, "On-Line Identification of Nonlinear Time-Variant Systems Using Structurally Adaptive Radial Basis Function Networks", *American Control Conference*, pp. 1037-1041, Albuquerque, New Mexico, 1997

[25] Y. Lu, N. Sundararajan, P. Saratchandran, "Analysis of minimal radial basis function network algorithm for real-time identification of nonlinear dynamic systems", *Control Theory Application, vol. 147, no. 4, pp. 476-484, 2000*

[29] C.R. Rao, S.K. Mitra, *Generalized inverse of matrices and its applications*. Wiley, New York, 1971
[30] G.H. Golub, C.F.V. Loan, *Matrix computations*. The Johns Hopkins Univ. Press, Baltimore, 1996.

M. J. Er (S'82-M'87-SM'07) received the B. Eng. and M. Eng. degrees in electrical engineering from the National University of Singapore, in 1985 and 1988, respectively, and the Ph.D. degree in systems engineering from the Australian National University, Canberra, Australia, in 1992. From 1987 to 1989, he was a R&D Engineer with Chartered Electronics Industries Pte Ltd and a Software Engineer in Telerate R&D Pte Ltd, respectively. He served as the Director of the Intelligent Systems Centre — a University Research Centre co-funded by Nanyang Technological University and Singapore Engineering Technologies from 2003 to 2006. He is currently a Full Professor in the School of Electrical and Electronic Engineering (EEE) and Director of Renaissance Engineering Programme, College of Engineering. He has authored 5 books, 16 book chapters, and more than 400 journal and conference papers. His research interests include fuzzy logic and neural networks, computational intelligence, robotics and automation, sensor networks, and biomedical engineering. Currently, Prof. Er is the Vice-Chairman of IEEE Computational Intelligence Society Standards Committee, Chairman of IEEE Computational Intelligence Society Singapore Chapter, and Chairman of Electrical and Electronic Engineering Technical Committee, Institution of Engineers, Singapore (IES). He serves as the Editor-in-Chief for the IES Journal B on Intelligent Devices and Systems, an Area Editor of the International Journal of Intelligent Systems Science, and an Associate Editor of twelve refereed international journals. Prof. Er was the winner of the IES Prestigious Publication (Application) Award in 1996 and IES Prestigious Publication (Theory) Award in 2001. He was awarded a Commonwealth Fellowship tenable at University of Strathclyde in 2000. He received the Teacher of the Year Award for the School of EEE in 1999, School of EEE Year 2 Teaching Excellence Award in 2008, and the Most Zealous Professor of the Year Award 2009. He also received the Best Session Presentation Award at the World Congress on Computational Intelligence in 2006. Furthermore, together with his students, he has won more than 30 awards at international and local competitions.

L. Y. Zhai received his PhD and M.Eng (both in Mechanical Engineering) from Nanyang Technological University of Singapore in 2010 and 2000, respectively. His B.Eng degree in Mechanical Engineering was awarded by Xi'an Jiaotong University (China) in 1994. He has been actively engaged in several funded research projects in manufacturing system modeling and optimisation in the past ten years. His publication includes more than 20 refereed international journal papers, book chapters and international conference papers. His main research interests include prognostic and health management, fuzzy logic and rough sets, knowledge discovery and intelligent systems.