Sequential Moore-Penrose Inverse Based Extreme Learning Machine with Growth of Hidden Nodes

Randriamamonjy Liantsoa Joharinirina[1], Randimbindrainibe Falimanana[2],

Robinson Matio[3]

[1]*Randriamamonjy Liantsoa Joharinirina, Student in Doctoral School of Science and Technical of Engineering and Innovation, Laboratory of Cognitives Sciences and Applications, University of Antananarivo, Madagascar*

[2] *Randimbindrainibe Falimanana, Professor in Doctoral School of Science and Technical of Engineering and Innovation, Laboratory of Cognitives Sciences and Applications, University of Antananarivo, Madagascar*

[3] *Robinson Matio, Doctor in Doctoral School of Science and Technical of Engineering and Innovation, Laboratory of Cognitives Sciences and Applications, University of Antananarivo, Madagascar*

ABSTRACT

One of the open problems in neural network research is how to automatically determine network architectures for given application. The extreme learning machine (ELM) have been proposed for generalized single-hidden-layer feedforward networks (SLFNs) which perform well in both regression and classification application. In this paper, an error-minimized incremental algorithm based on Sequential Moore-Penrose Inverse is proposed, to automatically determine the number of hidden nodes in SLFNs. This approach, Sequential Moore-Penrose inverse based ELM (SMP-ELM), is able to add random hidden nodes to SLFNs one by one or group by group. During the growth of the networks, the output weights are updated incrementally. Simulation results demonstrate and verify that our new approach can achieve more compact network structure with better generalization performance.

Keyword: - *Extreme learning machine (ELM), Growing algorithm, Incremental learning, Moore-Penrose Generalized Inverse, Sequential learning, minimizing error.*

1. Introduction

In the field of artificial neural networks, radial basis function (RBF) neural network and multilayer perceptron with one hidden layer have been regarded as a most remarkable single-hidden layer feedforward networks (SLFNs). In view of the fact that standard SLFNs with *N* hidden nodes can exactly learn *N* distinct observation, *Tamura* and *Tateishi* [1] proved that a SLFNs with N sigmoid hidden units can give any N input-target relation exactly. The capabilities of SLFNs to approximate complex nonlinear mapping directly from the input samples have been widely investigated due to their applications in various areas of scientific research and engineering [2]. Different from the conventional neural network theories where the parameters of SLFNs are well adjusted. *Huang and al* [3] [4] proposed extreme learning machine (ELM) where the hidden nodes of SLFNs need not be tuned. ELM has been successfully applied in many applications and have been shown to be an extremely fast learning algorithm and having good generalization performance. However, since ELM use batch learning scheme, the only factor that needs to be set by users is the size of the SLFNs (number of hidden nodes) before learning. Therefore, how to choose the optimal number of hidden nodes of ELM is the main objective of this paper.

This paper proposes a novel incremental extreme learning machine with sequential Moore-Penrose Inverse growth of hidden nodes and incremental updating of output weights using error-minimized based method (EM-ELM) proposed in [5] that can grow hidden nodes one by one or group by group.

The rest of this paper is organized as follows. Section 2 gives a brief review of ELM. The proposed algorithm is introduced in Section 3. Experimental results and discussion are shown in Section 4. Finally, conclusion are given in Section 5.

2. Brief review on ELM

This section will briefly reviews the ELM proposed by Huang, et al [3]. One key principle of the ELM is that one may randomly choose and fix the hidden node parameters. After the hidden node parameters are chosen randomly, SLFN becomes a linear system where the output weights of the network can be analytically determined using simple generalized inverse operation of the hidden layer output matrix.

For N arbitrary distinct samples(x_i, t_i), where $x_i = [x_{i1}, x_{i2}, \dots, x_{in}]^T \in \mathbb{R}^n$ and $t_i = [t_{i1}, t_{i2}, \dots, t_{im}]^T \in \mathbb{R}^m$ standard SLFNs can approximate these N samples with zero error means that there exist β_i, ω_i and b_i such that

$$\sum_{i=1}^{N} \beta_i \, g(\omega_i . x_j + b_i) = t_j , \qquad j = 1, \dots, N \tag{1}$$

where $\omega_i = [\omega_{i1}, \omega_{i2}, \dots, \omega_{in}]^T$ is the weight vector connecting the *i*th hidden neuron and the input neurons, $\beta_i = [\beta_{i1}, \beta_{i2}, \dots, \beta_{im}]^T$ is the weight vector connecting the *i*th hidden neuron and the output neurons, and b_i is the threshold of the *i*th hidden neuron, $\omega_i . x_j$ denotes the inner product of ω_i and x_j. The output neuron are chosen linear in this paper.

The above N equations can be written compactly as:

$$\mathbf{H}\beta = \mathbf{T} \tag{2}$$

where

$$\mathbf{H}(\omega_1, \dots, \omega_M, b_1, \dots, b_M, x_1, \dots, x_N) = \begin{bmatrix} g(\omega_1 . x_1 + b_1) & \cdots & g(\omega_L . x_1 + b_L) \\ \vdots & \ddots & \vdots \\ g(\omega_1 . x_N + b_1) & \cdots & g(\omega_L . x_N + b_L) \end{bmatrix}_{N \times L} \tag{3}$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_L^T \end{bmatrix}_{L \times m} \quad and \quad \mathbf{T} = \begin{bmatrix} t_1^T \\ \vdots \\ t_N^T \end{bmatrix}_{N \times m} \tag{4}$$

H is called the hidden layer output matrix of the neural network; the *i*th column of **H** is the *i*th hidden neuron's output vector with respect to input $x_1, x_2, \dots, x_N$.

Similar to SLFNs with additive neurons, for SLFNs with RBF kernel function $\phi(\frac{\|x-\mu\|}{\sigma})$ we have

$$\mathbf{H}\beta = \mathbf{T} \tag{5}$$

where **H** is the hidden layer output matrix of the RBF network;

$$\mathbf{H}(\mu_1, \dots, \mu_L, \sigma_1, \dots, \sigma_L, x_1, \dots, x_N) = \begin{bmatrix} \phi\left(\frac{\|x_1-\mu_1\|}{\sigma_1}\right) & \cdots & \phi\left(\frac{\|x_1-\mu_L\|}{\sigma_L}\right) \\ \vdots & \ddots & \vdots \\ \phi\left(\frac{\|x_N-\mu_1\|}{\sigma_1}\right) & \cdots & \phi\left(\frac{\|x_N-\mu_L\|}{\sigma_L}\right) \end{bmatrix} \tag{6}$$

the *i*th column of **H** is the output of the *i*th kernel with respect to inputs $x_1, x_2, \dots, x_N$.

Usually, when the number of training data is larger than the number of hidden nodes $N > L$, one cannot expect an exact solution of the system **(5)**. Fortunately, since it has been proved in theory [6] that SLFNs with random hidden nodes have the universal approximation capability, the hidden nodes can be randomly generated independent of the training data. It is proved that SLFNs' input weights ω_i and hidden neurons' biases b_i or centers and impact of RBF kernels need not be adjusted during training and one may simply randomly assign values to then. In batch ELM, the input weights and hidden biases are randomly assigned and the output weights β are estimated as:

$$\hat{\beta} = \mathbf{H}^{\dagger}\mathbf{T} \tag{7}$$

where $\mathbf{H}^{\dagger}$ is the Moore-Penrose generalized inverse (pseudo-inverse) [7], [8] of the hidden layer output matrix **H.**

The three-step simple learning algorithm [3], [9], [10] can be summarized as follow:

Algorithm ELM: Given a training set $\{(x_i, t_i)\}_{i=1}^{N} \subset \mathbb{R}^n \times \mathbb{R}^m$, activation function g or kernel function ϕ, and hidden neuron or kernel number *L*:

1) randomly assigned hidden node parameters (ω_i,b_i) or (μ_i, σ_i), $i = 1, \dots, L$
2) calculate the hidden-layer output matrix **H**
3) estimate the output weight $\beta : \hat{\beta} = \mathbf{H}^{\dagger}\,\mathbf{T}$.

As an important conclusion in the theory of ELM, interpolation theorem, presented in [4], Lemma 1, demonstrate that for a SLFNs with *N* hidden nodes can learn *N* distinct training samples with zero error. It also indicates that columns of the hidden matrix *H* satisfy column full rank when the number *N* of training samples is greater than the number *L* of hidden nodes, with probability one.

Lemma 1: Given a SLFNs with *N* hidden nodes and any activation function $g: \mathbb{R} \to \mathbb{R}$, which is infinitely differentiable in any interval, for *N* arbitrary distinct samples (x_i, t_i), where $x_i \in \mathbb{R}^n$ and $t_i \in \mathbb{R}^m$, for any a_i and b_i randomly chosen from any intervals of $\mathbb{R}^n$ and $\mathbb{R}$, respectively, according to any continuous probability distribution , then with probability one, the hidden matrix *H* is invertible, and $\|H\beta - T\| = 0$.

3. Growing Hidden Nodes using Sequential Moore-Penrose Inverse

3.1 Sequential Moore-Penrose Inverse

The procedure essentially provides a method for constructing the Moore-Penrose inverse of any columnwise partitioned matrix with the form $A = [A_1 : A_2]$ where A has n column, A_1 and A_2 has n_1, n_2 column respectively, and $n = n_1 + n_2$.

Greville's method [13] gives the Moore-Penrose inverse of any matrix with k columns, given the Moore-Penrose inverse of the sub matrix consisting of the first $k-1$ columns is known. For any $k \geq 2$, let A_k denote the matrix with k columns, $a_1, \dots, a_k$. Then A_k can be written as a partitioned form as $A_k = [A_{k-1}, a_k]$. Assuming $A_{k-1}^{\dagger}$ is known, A_k can be formed using the following formulas:
let

$$C_k = \left(I - A_{k-1}A_{k-1}^{\dagger}\right)a_k \tag{8}$$

and let

$$D_k = a_k^T (A_{k-1}^T)^{\dagger}\, A_{k-1}^{\dagger}\, a_k \tag{9}$$

Then

$$A_k^{\dagger} = \begin{bmatrix} A_{k-1}^{\dagger} - A_{k-1}^{\dagger}\, a_k\, B_k \\ B_k \end{bmatrix} \tag{10}$$

where

$$B_k = \begin{cases} C_k^{\dagger}, & if\ C_k \neq 0 \\ (I + D_k)^{-1} a_k^T (A_{k-1}^T)^{\dagger} A_{k-1}^{\dagger}, & if\ C_k = 0 \end{cases} \tag{11}$$

We extend this approach to any columnwise partitioned matrix as mentioned above. In [14, Theorem 1 & 2], two version of the general formula for Moore-Penrose inverse of matrix A partitioned as $A = [A_1 : A_2]$, have been given. These results can be summarized as:

Let $\mathrm{A} = [\mathrm{A}_1 : \mathrm{A}_2] \in \mathbb{R}_{\mathrm{m,n}}$ and let $\mathrm{P_i} \in \mathbb{R}_{\mathrm{m,m}}$ and $\mathrm{Q_i} \in \mathbb{R}_{\mathrm{m,m}}$ be the orthogonal projectors specified as: $\mathrm{P_i} = \mathrm{A_i A_i^{\dagger}}$ et $\mathrm{Q_i} = \mathrm{I_m} - \mathrm{P_i}$, $\mathrm{i} = 1,2$. then

$$A^{\dagger} = \begin{pmatrix} (Q_2 A_1)^{\dagger} \\ (Q_1 A_2)^{\dagger} \end{pmatrix} = \begin{pmatrix} A_1^{\dagger} - A_1^{\dagger} A_2 (Q_1 A_2)^{\dagger} \\ A_2^{\dagger} - A_2^{\dagger} A_1 (Q_1 A_1)^{\dagger} \end{pmatrix} \tag{12}$$

if and only $\mathcal{R}(A_1) \cap \mathcal{R}(A_2) = \{0\}$.

3.2 Proposed method for growing hidden nodes

Given a SLFNs with L_0 hidden nodes $\{(a_i, b_i)\}_{i=1}^{L_0}$, the hidden layer output matrix of this SLFNs is $\mathbf{H_1} = (a_1, \dots, a_{L_0}, b_1, \dots, b_{L_0}, x_1, \dots, x_N)$. If the network output error $E(\mathbf{H_1})$ is less than the target error $\varepsilon > 0$, no new hidden nodes will be added and the learning procedure completes. Otherwise, we can add new hidden node δL_0 to the existing SLFNs and we have the new hidden layer output matrix $\mathbf{H_2} = [\mathbf{H_1}, \boldsymbol{\delta}\mathbf{H_1}]$**.** According to the Lemma 1, the pseudo-inverse of $\mathbf{H_2}$ is obtained by using the sequential method propose in (10) and (12). Thus we can obtain a fast incremental output weight updating method for the SLFNs with the mechanism of growing hidden nodes.

Proposed algorithm: Given a set of training data $\{(x_i, t_i)\}_{i=1}^{N}$, the maximum number of hidden nodes L_{max}, and the expected learning accuracy $\epsilon > 0$, the proposed algorithm can be shown in two phases.

Initialization:

1) Initialize the SLFNs with a small randomly generated hidden nodes $(a_i, b_i)_{i=1}^{L_0}$ where L_0 is a small positive integer given by users.
2) Calculate the hidden-layer output matrix $\mathbf{H_1}$

$$\mathbf{H_1} = \begin{bmatrix} g(a_1, b_1, x_1) & \cdots & g(a_{L_0}, b_{L_0}, x_1) \\ \vdots & \ddots & \vdots \\ g(a_1, b_1, x_N) & \cdots & g(a_{L_0}, b_{L_0}, x_N) \end{bmatrix}_{N \times L_0} \tag{13}$$

3) Calculate the output error $E(\mathbf{H_1}) = \left\| \mathbf{H_1}\mathbf{H_1^{\dagger}}\mathbf{T} - \mathbf{T} \right\|$. (14)

Sequential growing hidden nodes: Let *k=0,*

While $L_k < L_{max}$ and $E(\mathbf{H_k}) > \varepsilon$:

1) $k = k + 1$
2) Randomly add new hidden node δL_k to the existing SLFNs. The total number of hidden nodes becomes $L_k = L_{k-1} + \delta L_k$ and the corresponding hidden-layer output matrix $\mathbf{H_{k+1}} = [\mathbf{H_k}, \boldsymbol{\delta}\mathbf{H_k}]$, where

$$\boldsymbol{\delta}\mathbf{H_k} = \begin{bmatrix} g(a_{L_k}, b_{L_k}, x_1) \\ \vdots \\ g(a_{L_k}, b_{L_k}, x_N) \end{bmatrix} \tag{15}$$

3) The output weights β are updated in a way as:

$$C_k = \left(I - \boldsymbol{\delta}\mathbf{H_k}\boldsymbol{\delta}\mathbf{H_k^{\dagger}}\right)\mathbf{H_k} \tag{16}$$

$$\beta_{k+1} = \mathbf{H_{k+1}^{\dagger}}\,\mathbf{T} = \begin{bmatrix} C_k \\ \boldsymbol{\delta}\mathbf{H_{k-1}^{\dagger}} - \boldsymbol{\delta}\mathbf{H_{k-1}^{\dagger}}\,\mathbf{H_k}C_k \end{bmatrix}\mathbf{T}. \tag{17}$$

End While.

The number of new hidden nodes to be added need be kept constant, i.e., the algorithm allows $\delta L_k = \delta L_{k+1}$. It is of interest that, the convergence of the proposed algorithm is given in [5].

4. Experimental verification of performance

In this section, we investigate the performance of the proposed algorithm in some real-world benchmark regression and classification data sets [12] (cf. Table-1). All the simulations are carried out in Matlab R2013a environment running on a desktop PC dual core, 2,4 GHZ CPU with 2 GB RAM. The performance of our method has been compared with the sequential/incremental/growing algorithm error-minimized extreme learning machine with growth hidden nodes and incremental learning (EM-ELM) [5]. The *sigmoid* function $g(x) = 1/(1 + \exp(-x))$ is used as an activation function for all methods.

In our experiments, all inputs (attributes) have been normalized into the range [-1, 1] while the outputs (targets) have been normalized into [0, 1]. The average results are obtained over 20 trials for all cases. For each trail of simulations, the datasets of the application was divided into training and testing datasets with the number of samples indicated in **Table-1.**

Table-1: Specification of Benchmark Data Sets

Datasets	Attributes	Classes	Types	#Training Data	#Testing Data
Machine CPU	8	-	Regression	100	109
Boston Housing	14	-	Regression	250	256
SinC	1	-	Regression	1000	1000
Diabetes	8	2	Classification	576	192
Segment	19	7	Classification	1100	1110
Satimage	36	7	Classification	3217	3218

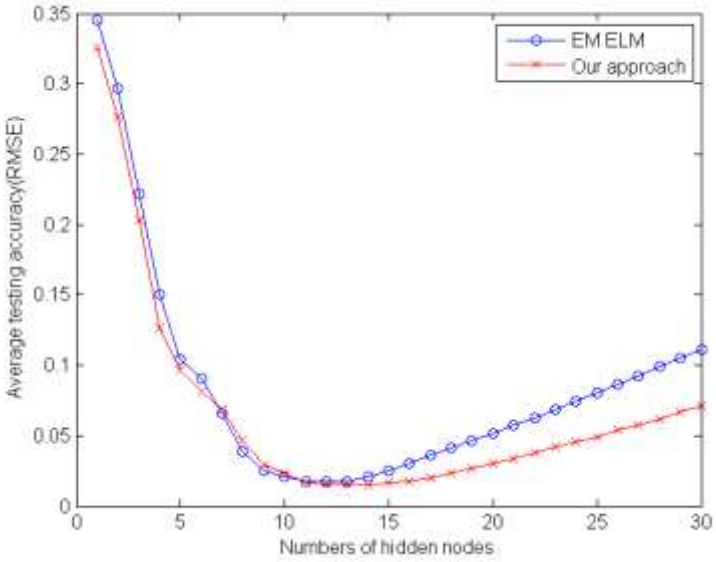

Fig-1: Average testing RMSE in SinC case.

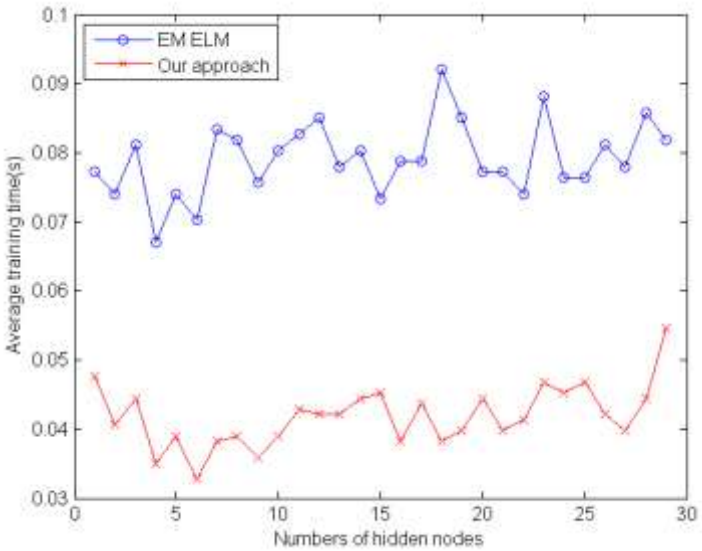

Fig-2: Average training time in SinC case.

SMP-ELM and EM-ELM have first been compared in benchmark problem: SinC function. One thousand training and testing data have been randomly generated from the interval [-10, 10] with a uniform distribution. Both algorithm increases the hidden nodes one by one along all experiments.

Fig-1 shows the testing root mean square error (RMSE) obtained by SMP-ELM and EM-ELM methods. We can observe that the testing RMSE of SMP-ELM is comparable with that of EM-ELM. It can be seen that SMP-ELM can obtain better generalization performance than EM-ELM. **Fig-2** shows the computational complexity comparisons of the two methods. We can show that our method for growing hidden nodes is much faster than EM-ELM.

Further comparison has been conducted in some real benchmark regression and classification problems shown in **Table-1**. With the same expected accuracy (or the same maximum step), the network structure and the generalization performance obtained by both SMP-ELM and EM-ELM are compared. The stopping RMSE and maximum step for all datasets cases are presented in **Table-2.a** and **Table-2.b**.The apparent better results are shown in boldface. As compared to EM-ELM, SMP-ELM always has lower network structure with less testing RMSE in most cases. The testing standard deviation (STD dev.) of SMP-ELM is better or comparable with EM-ELM, which proved the stability of our approach.

Table-3 give performance results of SMP-ELM by adding hidden nodes one by one or group by group. As observe from the results, adding hidden nodes group by group can speed up the training time of the method and the generalization performance is increased. Furthermore, in view of network structure, growing hidden nodes one by one allow to obtain lower number of hidden nodes.

Table-2.a: Performance comparison between SMP-ELM and EM-ELM (Regression)

Regression Datasets	Stop RMSE	Max Steps	Algorithms	Training time (s)	Testing RMSE	Testing Dev.	# Nodes	Nodes Dev.
Machine CPU	0.06	20	EM-ELM	0.0172	0.0758	0.0313	13.95	5.3947
	0.06	20	SMP-ELM	**0.0140**	**0.0687**	0.0213	**13.1**	5.6186
Boston housing	0,13	200	EM-ELM	0,0289	0,1293	0,0109	13.05	4.861
	0,13	200	SMP-ELM	**0,0156**	**0,1266**	0,031	**11.9**	3.2428
Abalone	0.1	200	EM-ELM	1.8229	0.0993	0.009	**8.7**	6.13
	0.1	200	SMP-ELM	**0.8588**	**0.0989**	0.0062	**8.3**	5.526

Table-2.b: Performance comparison between SMP-ELM and EM-ELM (Classification)

Classification Datasets	Stop Accuracy	Max Steps	Algorithms	Training time (s)	Testing Accuracy	Testing Dev.	# Nodes	Nodes Dev.
Diabetes	0.8	25	EM-ELM	0.5070	0.7737	0.0331	20	8.0263
	0.8	25	SMP-ELM	**0.2675**	**0.7828**	0.0272	**19.6**	7.6667
Segment	0.9	50	EM-ELM	0.8221	0.8155	0.0109	11.25	1.9160
	0.9	50	SMP-ELM	**0.617**	0.8141	0.0107	**9.7**	1.3803
Satimage	0.8	50	EM-ELM	8,8498	0.8025	0.0016	16.10	2.9231
	0.8	50	SMP-ELM	**4.5505**	0.8035	0.0037	16.6	3.0258

Datasets	#Nodes added	Training time(s)	Testing Accuracy	#Nodes
Diabetes	1-by-1	0.1903	0.76	15.1
	3-by-3	0.071	0.7656	16
	5-by-5	0.0491	0.7573	17.75
	1-by-1	0.741	0.8152	10.4

Segment	3-by-3	0.2668	0.8286	11.3
	5-by-5	0.1825	0.8425	12
Satimage	1-by-1	5.2557	0.7946	17.2
	3-by-3	1.7519	0.7979	17
	5-by-5	1.0483	0.8068	17.5

Table-3: Performance of SMP-ELM with hidden nodes added (one by one or group by group)

5. Conclusion

In this paper, we have propose an extreme learning machine with sequential Moore-Penrose Inverse (SMP-ELM) growth of hidden nodes and incremental updating of output weights using an error-minimized-based method to automatically determine the hidden nodes number in generalized SLFNs. Our approach allows the random hidden nodes to be added one by one or group by group assuming that the generalized inverse of the existing hidden nodes is known. The output weights are then updated incrementally during the growth of the networks. The simulation results on real and artificial benchmark problems show that the new approach can achieve more compact network architecture with good generalization performance than EM-ELM. Results on growing hidden nodes group by group show that our approach can significantly reduce the computation complexity of incremental ELM. The performance of our method on other type of function (geninv, qrgeninv, ginv) for computing Moore-Penrose inverse matrices will be reported in the future work.

6. References

[1]. S.Tamura and M. Tateishi, "Capabilities of a four-layered feedforward neural network: Four layers versus three," *IEEE Transaction on Neural Network*, vol.8, no. 3, 251-255, 1997
[2]. Bishop, C.M, "Neural network for pattern recognition". Oxford University Press, New York 1995
[3]. G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme Learning Machine: A new learning scheme of feedforward neural networks", in *Proceedings of International Joint Conference on Neural Networks (IJCNN 2004),* (Budapest, Hungary), 25-29 July, 2004
[4]. G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine: Theory and applications", *Neurocomputing*, vol. 70, 489-501, 2006
[5]. G. Feng, G.-B. Huang, Q. Lin and R.Gay, "Error Minimized Extreme Learning Machine with growth of hidden nodes and incremental learning", *IEEE Transactions on Neural Networks*, vol.20, No.8, august 2009
[6]. G.-B. Huang, L. Chen, and C.-K. Siew, "Universal approximation using incremental constructive networks", revised and resubmitted to *IEEE Transactions on Neural Networks*, 2005
[7]. D. Serre, Matrices: Theory and Applications, New York: Springer-Verlag, 2002
[8]. C.-R. Ras and S.-K. Mitra, Generalized inverse of matrices and its applications, New York: Wiley, 1971
[9]. G.-B. Huang, C.-K. Siew, "Extreme Learning Machine: RBF network case", in *Proceedings of the Eighth International Conference on Control, Automation, Robotics and Vision (ICARCV 2004),* (Kunming China), 6-9 Dec, 2004
[10]. G.-B. Huang, C.-K. Siew, "Extreme Learning Machine with randomly assigned RBF kernels", *International Journal of Information Technology*, vol.11, no.1, 2005
[11]. Randall E. Cline, Elements of the theory of generalized inverses for matrices, *Mathematics Department University of Tennessee*, 1979
[12]. Black, C.L., Merz, and C.J.: "UCI repository of machine learning databases", *Department of Information and Computer Sciences*, University of California, Irvine, USA (1998),http://www.ics.uci.edu/~mlearn/mlrepository.html
[13]. T.N.E. Greville, "Some applications of the pseudoinverse of a matrix," *SIAM Review*, vol.2, no.1, 15-22, 1960.