# The Fast Computation Methods for Extreme Learning Machine

Tao Dou
Dalian University of Technology, Faculty of Electronic Information and Electrical Engineering, Computer Application Technology
Dalian City, Liaoning Province Postal Code: 116024

Xu Zhou
Agricultural University of Hebei, the ministry of basic course
Huanghua City, Hebei Province Postal Code: 061100

***Abstract*—The extreme learning machine (ELM) that is proposed by Huang is designed based on single-hidden layer feedforward neural networks (SLFNs), which can randomly choose the parameters of hidden nodes and the output weights gotten analytically. So it can get the solution fastly. However, the learning time of ELM is mainly spent on calculating the Moore-Penrose generalized inverse matrices of the hidden layer output matrix. This paper mainly focuses on the effective computation of the Moore-Penrose generalized inverse matrices for ELM. Moreover, several methods are proposed, which are tensor product matrix ELM (TPM-ELM), Geninv ELM Numerical experiments show that both Geninv-ELM and TPM-ELM are faster than other kinds of ELM and can reach comparable generalization performance.**

## I. INTRODUCTION

Recently Huang Guang-bin et al. proposed extreme learning machine that is a new single-hidden layer feedforward neural networks (SLFNs). Because it can randomly chooses hidden nodes and get the output weights of SLFNs analytically. Theory and experiments prove that the Extreme learning machine (ELM) has simple structure and powerful approximation capability [1,3]. Huang Guang-bin et al. show that ELM is one of the most popular neural networks especially it runs extremely fast. ELM not only can get the smallest training error but also the get good generalization performance and it obtains the smallest norm of weights when it compared to the traditional learning algorithm on feedforward network. ELM can get well function approximation from any finite set, Huang and Babri [4] shows that a single-hidden layer feedforward neural network (SLFNs) can learn N different observations if the network has N hidden nodes whose activation function must be nonlinear. Huang Guang-bin et al. prove that when the activation functions are infinitely differentiable in the hidden notes, we can randomly choose the hidden layer biases and the input weights of the single-hidden layer feed forward neural network (SLFNs).

However, when dealing with large datasets, certain problems arise, including the huge amount of memory required for storing the weights and bias matrix, so a lot of varieties of ELM have been proposed by researchers for solving both generalization performance and learning speed problems in the past few years [2,5]. Such as error minimized extreme learning machine, on line sequential extreme learning machine, a structure-adjustable online learning ELM are with quicker learning speed [7]. In the past two decades, much work has been done about the approximation capabilities of SLFNs [6]. A series of learning algorithms have been proposed recently that referred to as incremental extreme learning machines. The parameters of hidden node and the input weights in the network are randomly chosen. The output weights of the network can be determined analytically [8]. There are normally two heuristic approaches to adaptive the number of hidden nodes in the network, either destructive nodes method (usually called pruning nodes method) [9]. Modify the structure of SLFNs: constructive methods (usually called growing nodes methods)

Extreme learning machine (ELM) is a fast learning algorithm that can be considered as a linear system. However, calculating the hidden layer output matrix which actually is the Moore-Penrose generalized inverse matrices that need spending much time to get the solution. So this paper focus on the learning speed of ELM about the computation of the Moore-Penrose generalized inverse matrices. There are several methods for computing the Moore-Penrose inverse matrix [10], these methods may include orthogonalization method, orthogonal projection, iterative method, and singular value decomposition (SVD) [11]. As we know, both the orthogonalization and iterative method have their limitations when using the searching and iteration in the learning algorithm. The orthogonal projection method can be used when $\mathbf{H}^T\mathbf{H}$ is nonsingular and $\mathbf{H}^\dagger = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T$ which is also used. However, the orthogonal projection method usually perform worse, when the $\mathbf{H}^T\mathbf{H}$ is not nonsingular or tend to be singular, thus the matrix decomposition techniques can be used. The SVD has been used by some expert to calculate the Moore-Penrose generalized inverse of the hidden layer output matrix $\mathbf{H}$ in ELM. The SVD is very accurate method has been proved but also time-consuming since it requires a large mount of computational resources, especially in the case of dealing with large matrices.

Also, several methods for computing the Moore-Penrose generalized inverse matrix by matrix decomposition techniques have been generated in the recent contribution[12]. V.-N. Katsikis [14] provided a fast and reliable method to calculate the Moore-Penrose generalized inverse and that also can be used for large sparse matrices. Courrieu [15] proposed a fast computation of Moore-Penrose generalized inverse matrices based on a full rank Cholesky factorization. Toutounian and Ataei [13] presented the CGS-MPi algorithm which is based on parting the Moore-Penrose inverse matrices, and the method of conjugate Gram-Schmidt process. They proved that this algorithm is efficient tool for computing the Moore-Penrose inverse and it is a robust algorithm especially dealing with rank deficient and large sparse matrices. Vasilios [17] proposed a method based on QR factorization, which can effective calculate the Moore-Penrose inverse when the matrices are singular square matrices or rectangular matrices.

In this paper, we focus on effective computation of the Moore-Penrose generalized inverse matrices for ELM. Several methods are proposed, which are tensor product matrix ELM (TPM-ELM), QR factorization and Geninv ELM (QR-ELM).

## II. Extreme Learning Machine

As we know that the single-hidden-layer neural networks are consisted of three layers: input nodes, hidden layer and output nodes. The input nodes' role is to pass signals to the hidden layer. And the hidden layer nodes are consisted of activation function; The output layer nodes are usually can be considered a simple linear function; In the single hidden layer feed forward neural network, from input layer to hidden layer are non-linear transformation and from the hidden layer to the output layer transformation is linear, that is, the output of the network hidden nodes are a linear weighted sum.

The training data usually are $(x_i, t_i)$, $i = 1, 2, \cdots, N$ and the samples are arbitrary distinct, where $t_i = [t_{i1}, t_{i2}, \cdots, t_{im}]^T \in R^m$ $m$ is the number of output nodes, $x_i = [x_{i1}, x_{i2}, \cdots, x_{in}]^T \in R^n$ $n$ is the number of the sample attributes. $\tilde{N}$ is the number of the hidden nodes and the activation function is $g(x)$. We usually use mathematically modeled as $\sum_{i=1}^{\tilde{N}} \beta_i g_i(x_i) = \sum_{i=1}^{L} \beta_i g(w_i \cdot x_j + b_i) = o_j$. $j = 1, \cdots, N.$ the weight vector $w_i = [w_{i1}, w_{i2}, \cdots, w_{in}]^T$ which connecting the input nodes with the $i$ th hidden node. The weight vector $\beta_i = [\beta_{i1}, \beta_{i2}, \cdots, \beta_{im}]^T$ which connecting the $i$ th hidden node with the output nodes, and the threshold of the $i$ th hidden node is $b_i$. $w_i \cdot x_j$ is the inner product of the weight $w_i$ and the threshold $b_i$. That standard SLFNs with $\tilde{N}$ hidden nodes and the activation function is $g(x)$. The SLFNs can approximate these arbitrarily $N$ samples with zero error, that is, $\sum_{j=1}^{\tilde{N}} \left\| T_j - t_j \right\| = 0$ i.e. It also can be showed as $\sum_{i=1}^{\tilde{N}} \beta_i g(w_i \cdot x_j + b_i) = t_j$. $j = 1, \cdots, N.$ These $N$ equations can be written compactly as

$H\beta = T$ Where

$$H(w_1, \cdots, w_{\tilde{N}}, b_1, \cdots, b_{\tilde{N}}, x_1, \cdots, x_N)$$

$$= \begin{bmatrix} g(w_1 \cdot x_1 + b_1) & \cdots & g(w_{\tilde{N}} \cdot x_1 + b_{\tilde{N}}) \\ \vdots & \cdots & \vdots \\ g(w_1 \cdot x_N + b_1) & \cdots & g(w_{\tilde{N}} \cdot x_N + b_{\tilde{N}}) \end{bmatrix}_{N \times \tilde{N}}$$

$$\beta = \begin{bmatrix} \beta_1^T \\ \vdots \\ \beta_{\tilde{N}}^T \end{bmatrix}_{\tilde{N} \times m} \text{ and } T = \begin{bmatrix} t_1^T \\ \vdots \\ t_{\tilde{N}}^T \end{bmatrix}_{N \times m}$$

Generally, training an SLFNs, we need find the optimal $\hat{w}_i, \hat{b}_i \ (i = 1, \cdots, \tilde{N}), \hat{\beta}$ from the fellow:

$$\left\| H(\hat{w}_1, \cdots, \hat{w}_{\tilde{N}}, \hat{b}_1, \cdots, \hat{b}_{\tilde{N}})\hat{\beta} - T \right\|$$
$$= \min_{w_i, b_i, \beta} \left\| H(w_1, \cdots, w_{\tilde{N}}, b_1, \cdots, b_{\tilde{N}})\beta - T \right\|$$

$H$ can be get from the gradient-based learning algorithms which are usually searching the minimum value of the formula $\left\| H\beta - T \right\|$. The input weights and the output weights $(w_i, \beta_i)$ and the biases parameters of the hidden nodes, are usually get the best solution by iteratively adjusted just as follow: $W_K = W_{K-1} - \eta \frac{\partial E(W)}{\partial (W)}$. In this equation $\eta$ is the learning rate. Generally, The users get the best solution by the BP learning algorithm.

ELM is proposed by Huang et al. based on SLFNs, however, unlike the traditional SLFNs, when the activation function in the hidden nodes is infinitely differentiable, the input weights and the hidden nodes biases usually get randomly. Differently, the traditional neural network, all of the parameters of SLFNs need to be adjusted, but, in the ELM learning algorithm, all of the parameters need not turn, such as the input weights and the biases in the hidden nodes. $H$ is the hidden layer output matrix, which can be determined randomly at the beginning of learning, and it need not be turned in the all of learning process. And the output weights of the network is $\hat{\beta}_i$ which is usually get from the smallest norm least squares solution of the formula $H\beta = T$, the output weights $\hat{\beta} = H^{\dagger}T$, and $H^{\dagger}$ is inverse of matrix $H$, which is the Moore-penrose generalized inverse.

The ELM algorithm is simply summarized as follows:

Algorithm ELM: Generally the training set is:

$X = \{(x_i, t_i) \mid x_i \in R^n,\ t_i \in R^m,\ i = 1, \cdots, N\}$, $N$ is the number of the samples, $g(x)$ is the activation function, $\tilde{N}$ is the number of hidden nodes.

Step 1: Get the input weights $w_i$ and the bias

$b_i, (i = 1, \cdots, \tilde{N})$ which are all Randomly determined.

Step 2: Calculate $H$ that is the hidden layer output matrix.

Step 3: Solute the output weight $\beta$. Form calculating the formula $\beta = H^{\dagger}T. and\ T = [t_1, \cdots, t_N]^T$.

Finding the minimum norm least squares solution is equivalent to calculating the Moore-Penrose generalized inverse of matrix.

## III. Fast Computation Methods For Elm

In this section, the matrix $\mathbf{H}^{\dagger}$ is calculated by matrix decomposition techniques. We will propose several learning algorithms, which are tensor product matrix ELM (TPM-ELM), QR factorization and Geninv ELM (QR-ELM) Our work is mainly based on the the traditional methods to calculate the Moore-Penrose generalized inverse matrix [30-33].

### *a) The product matrix ELM (TPM-ELM )*

**Definition 3.1.** For each $\mathbf{x} \in R^N$, and usually assume that $\{\mathbf{e}_1, ..., \mathbf{e}_L\}$ and $\{\mathbf{h}_1, ..., \mathbf{h}_L\}$ are two collections of orthonormal vectors and they are linearly independent vectors of $R^N$, $L < N$, respectively. The mapping $\mathbf{e} \otimes \mathbf{h} : R^N \to R^N$ with $(\mathbf{e} \otimes \mathbf{h})(\mathbf{x}) = \langle \mathbf{x}, \mathbf{e} \rangle \mathbf{h}$ .If every rank-$L$ operator $F$ can be written in the form $F = \sum_{i=1}^{L} \mathbf{e}_i \otimes \mathbf{h}_i$ . Then, $F$ is called the tensor product of the collections $\{\mathbf{e}_1, ..., \mathbf{e}_L\}$ and $\{\mathbf{h}_1, ..., \mathbf{h}_L\}$ . The corresponding matrix $\mathbf{F}$ can be written in the form $\mathbf{F} = [\mathbf{h}_1, \cdots, \mathbf{h}_L, \mathbf{0}, \cdots, \cdots \mathbf{0}] \in R^{N \times N}$ . $\mathbf{F}$ is called the tensor-product matrix of the given collections.

**Theorem 3.1.** Let $\mathcal{H}$ be a Hilbert space. If $F = \sum_{i=1}^{L} \mathbf{e}_i \otimes \mathbf{h}_i$ is a rank-$L$ operator then its generalized inverse is also a rank-L operator and for each $\mathbf{x} \in \mathcal{H}$, it is defined by the relation $\mathbf{F}^{\dagger}\mathbf{x} = \sum_{i=1}^{L} \lambda_i(\mathbf{x})\mathbf{e}_i$ ,

Where the functions $\lambda_i$ are the solution of an appropriately defined $L \times L$ linear system.We proceed with computing the generalized inverse of a tensor-product matrix as follows:

Assume that $\{\mathbf{h}_1, ..., \mathbf{h}_L\}$ are collections of linearly independent vectors of $R^N$, $L < N$,

$\mathbf{H} = [\mathbf{h}_1, \cdots, \mathbf{h}_L] \in R^{N \times L}$

$\mathbf{F} = [\mathbf{h}_1, \cdots, \mathbf{h}_L, \mathbf{0}, \cdots, \cdots \mathbf{0}] = [\mathbf{H} \quad \mathbf{0}]$ . $\in R^{N \times N}$ For each $l = 1, \cdots, N$ , we solve the linear system $\langle \mathbf{e}_l, \mathbf{h}_i \rangle = \sum_{j=1}^{L} \lambda_j(\mathbf{e}_l)\langle \mathbf{h}_i, \mathbf{h}_j \rangle, \quad i = 1, \cdots, L$

Have $\lambda_1(\mathbf{e}_l), \lambda_2(\mathbf{e}_l), ..., \lambda_L(\mathbf{e}_l)$ as a solution from the above linear system. On the basis of Theorem 3.1 $\mathbf{F}^{\dagger}\mathbf{e}_l = \sum_{j=1}^{L} \lambda_j(\mathbf{e}_l)\mathbf{e}_j, . \quad l = 1, \cdots, N$ Then the generalized inverse $\mathbf{F}^{\dagger}$ has the following form

$$\mathbf{F}^{\dagger} = \begin{pmatrix} \lambda_1(\mathbf{e}_1) & \lambda_1(\mathbf{e}_2) & \dots & \lambda_1(\mathbf{e}_N) \\ \lambda_2(\mathbf{e}_1) & \lambda_2(\mathbf{e}_2) & \dots & \lambda_2(\mathbf{e}_N) \\ \vdots & \vdots & \vdots & \vdots \\ \lambda_L(\mathbf{e}_1) & \lambda_L(\mathbf{e}_2) & \dots & \lambda_L(\mathbf{e}_N) \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 \end{pmatrix} = \begin{bmatrix} \mathbf{H}^{\dagger} \\ \mathbf{0} \end{bmatrix} \in R^{N \times N} .$$

The TPM-ELM algorithm is summarized as follows:

Generally the training set is:

$X = \{(x_i, t_i) \mid x_i \in R^n,\ t_i \in R^m,\ i = 1, \cdots, N\}$, $N$ is the number of the samples, $g(x)$ is the activation function, $L$ is the number of hidden nodes.

Step 1: Get the input weights $w_i$ and the bias

$b_i, (i = 1, \cdots, L)$ which are all Randomly determined.

Step 2: Calculate the Moore-Penrose generalized inverse of matrix $\mathbf{H} \in R^{N \times L}$

if $N > L$, then $\mathbf{C} = \mathbf{H}^T\mathbf{H}$, $\mathbf{H}^{\dagger} = \mathbf{C} \setminus \mathbf{H}^T$

if $L > N$, then $\mathbf{C} = \mathbf{H}\mathbf{H}^T$, $\mathbf{H}^{\dagger} = \mathbf{C} \setminus \mathbf{H}$

Step 3: Solute the output weight $\beta$. Form calculating the formula $\beta = H^{\dagger}T$. and $\mathbf{T} = [\mathbf{t}_1,...,\mathbf{t}_N]^T \in R^{N\times m}$.

$\mathbf{Y} = \mathbf{C} \setminus \mathbf{H}^T$ is the solution to the linear system $\mathbf{CY} = \mathbf{H}^T$ computed by Gaussian Elimination.

$$\mathbf{Y} = \mathbf{H}^{\dagger} = \begin{bmatrix} \lambda_1(\mathbf{e}_1) & \lambda_1(\mathbf{e}_2) & \cdots & \lambda_1(\mathbf{e}_N) \\ \lambda_2(\mathbf{e}_1) & \lambda_2(\mathbf{e}_2) & \cdots & \lambda_2(\mathbf{e}_N) \\ \vdots & \vdots & \vdots & \vdots \\ \lambda_L(\mathbf{e}_1) & \lambda_L(\mathbf{e}_2) & \cdots & \lambda_L(\mathbf{e}_N) \end{bmatrix} \in R^{L\times N}.$$

*b) The Full rank Cholesky factorization ELM (Geninv-ELM)*

Geninv-ELM provides a rapid method, which is based on a known reverse order law and a full rank Cholesky factorization. The computation time is substantially shorter, particularly for large systems, and Geninv-ELM can be used for rank deficient matrices. The method of computing the Moore-Penrose inverse matrix is based on the work of P. Courrieu [14].

The Geninv-ELM algorithm is summarized following:

The training set is $X = \{(\mathbf{x}_i, \mathbf{t}_i) \mid \mathbf{x}_i \in R^n, \mathbf{t}_i \in R^m, i = 1,...,N\}$, the activation function we select the function $g(x)$, and the number of hidden nodes is $L$, and $L < N$.

Step 1: Randomly assign input weight $\mathbf{w}_i$ and bias $b_i$ $(i = 1,...,L)$.

Step 2: Calculate the Moore-Penrose generalized inverse of the matrix $\mathbf{H} \in R^{N\times L}$.

(1) $\mathbf{H}^T\mathbf{H} \in R^{L\times L}$, $rank(\mathbf{H}^T\mathbf{H}) = r \le L$.

(2) Utilize the full rank Cholesky factorization of $\mathbf{H}^T\mathbf{H}$, obtains a matrix $\mathbf{S}$ which satisfies $\mathbf{H}^T\mathbf{H} = \mathbf{S}^T\mathbf{S}$, where $\mathbf{S} \in R^{L\times L}$ is a unique upper triangular matrix with $L - r$ zero rows.

(3) Remove the zero rows from $\mathbf{S}$, obtains a matrix $\mathbf{R}^T \in R^{r\times L}$, $rank(\mathbf{R}) = r$, which satisfies $\mathbf{S}^T\mathbf{S} = \mathbf{RR}^T$, thus $\mathbf{H}^T\mathbf{H} = \mathbf{S}^T\mathbf{S} = \mathbf{RR}^T$.

(4) Calculate $(\mathbf{H}^T\mathbf{H})^{\dagger} = (\mathbf{RR}^T)^{\dagger} = \mathbf{R}(\mathbf{R}^T\mathbf{R})^{-1}(\mathbf{R}^T\mathbf{R})^{-1}\mathbf{R}^T$.

Where $\mathbf{R}^T\mathbf{R} \in R^{r\times r}$.

Step 3: Calculate the output weight $\beta \in R^{L\times m}$.

$\beta = \mathbf{H}^{\dagger}\mathbf{T} = (\mathbf{H}^T\mathbf{H})^{\dagger}\mathbf{T} = \mathbf{R}(\mathbf{R}^T\mathbf{R})^{-1}(\mathbf{R}^T\mathbf{R})^{-1}\mathbf{R}^T\mathbf{H}^T\mathbf{T}$, where $\mathbf{T} = [\mathbf{t}_1,...,\mathbf{t}_N]^T \in R^{N\times m}$.

The advantage of Geninv-ELM is that computation of the $L\times L$ inverse matrix is changed into one of the $r\times r$ inverse matrix ($r \le L$).

## IV. Performance Evaluation

In this part, we compare the performance of the computation of Moore-Penrose generalized inverse matrices used in ELM on the eight datasets from the UCI. The environment of implementing of all these algorithm is: in MATLAB7.1, double 2.8 GHz, 1G memory and the CPU is Pentium 4. The activation function for all models is sigmoid function $g(\mathbf{x}) = 1/(1+\exp(-\mathbf{x}))$, In the real-world problems, the attributes of all the training set and testing set were scaled to [-1, 1]. We conduct the performance comparison of the methods for eight real problems: Digit, DNA, Vehicle, Page, Sat, Shuttle, Usps and Letter. All the datasets are from the UCI databases[17]. The numbers of attributes, classes, samples for training and testing, and hidden nodes are shown in Table I.

Table I. Specification of the real-world problems.

| Datasets | # attributes | # Classes | # Training | # Testing | # hidden nodes |
|---|---|---|---|---|---|
| Digit | 64 | 10 | 2810 | 2810 | 400 |
| DNA | 180 | 3 | 3457 | 1729 | 400 |
| Letter | 16 | 26 | 10000 | 10000 | 400 |
| Page | 10 | 5 | 2736 | 2737 | 250 |
| Sat | 36 | 7 | 3217 | 3218 | 400 |
| Shuttle | 9 | 7 | 29000 | 29000 | 250 |
| Usps | 256 | 10 | 6198 | 3100 | 400 |
| Vehicle | 18 | 4 | 423 | 423 | 150 |

For the TPM-ELM, Geninv-ELM methods, the number of hidden nodes was set 50, 100, 150, 200, 250, 300, 350, and 400. We will present the average learning results on these benchmark datasets by using the above algorithms with 10-fold cross-validation. The testing accuracy of the algorithms in eight datasets with different hidden nodes is reported in Fig. 1-8 From the fluctuating curves, we have obtained values of L selected for each application. As observed from Fig. 1-8, we can get the conclusion if the hidden nodes are too few or too much the generalization performance of ELM will tend to be worse. Otherwise, the ELM algorithm get better perform on the moderate hidden nodes.

We compare the performance of methods (TPM-ELM, Geninv-ELM) in the real-world problems. The attributes of datasets were scaled to [-1, 1]. The sigmoid function was used as an activation function for the all algorithm. With 10-fold cross-validation were conducted for all the ELM algorithms and the average results are shown in Tables 2 and Table III. Table III shows the performance comparison of testing accuracy of the six methods in the real-world problems. As observed from the Table II, general speaking, all of methods obtain similar testing accuracy, Geninv-ELM is slightly lower than TPM-ELM and SVD-ELM in many cases.

TABLE II. THE AVERAGE TESTING ACCURACY OF THE METHODS IN THE REAL-WORLD PROBLEMS.

| Data sets | hidden nodes | TPM-ELM | Geninv-ELM | SVD-ELM |
|---|---|---|---|---|
| Digit | 400 | 0.9779 | 0.9826 | 0.98084 |
| DNA | 400 | 0.9410 | 0.9371 | 0.9323 |
| Letter | 400 | 0.8734 | 0.8749 | 0.8752 |
| Sat | 400 | 0.8875 | 0.8888 | 0.8900 |
| Usps | 400 | 0.9484 | 0.9500 | 0.9500 |
| Vehicle | 150 | 0.8014 | 0.8061 | 0.8180 |
| Shuttle | 250 | 0.7301 | 0.6736 | 0.7366 |
| Page | 250 | 0.9536 | 0.9532 | 0.9529 |

TABLE III THE AVERAGE TRAINING TIME OF THE METHODS IN THE REAL-WORLD PROBLEMS.

| Data sets | hidden nodes | TPM-ELM | Geninv-ELM | SVD-ELM |
|---|---|---|---|---|
| Digit | 400 | 1.2172 | 1.0313 | 2.5313 |
| DNA | 400 | 0.6240 | 0.7956 | 1.8876 |
| Letter | 400 | 3.3750 | 3.5781 | 11.875 |
| Sat | 400 | 0.9063 | 1.1875 | 2.7500 |
| Usps | 400 | 2.1250 | 2.4063 | 5.2656 |
| Vehicle | 150 | 0.0156 | 0.0313 | 0.0781 |
| Shuttle | 250 | 2.0748 | 4.0781 | 10.5313 |
| Page | 250 | 0.5781 | 0.5938 | 1.9531 |

Table III shows the performance comparison of average training time of the methods in the real-world problems. From the Table 3, the learning speed is significantly different, both TPM-ELM and Geninv-ELM obtain comparable performance to other methods with much faster learning speed in all cases. TPM-ELM and Geninv-ELM learn up to 2-5 times faster than SVD-ELM.

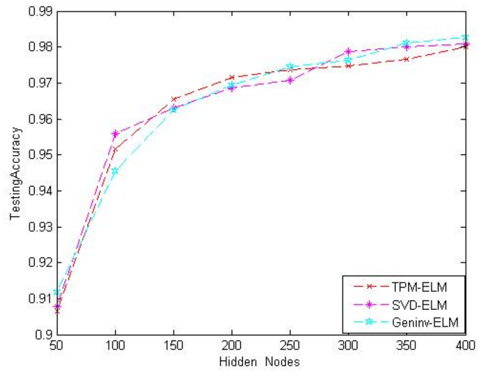

Figure 1. The testing accuracy of the three algorithms in Digit with different hidden nodes

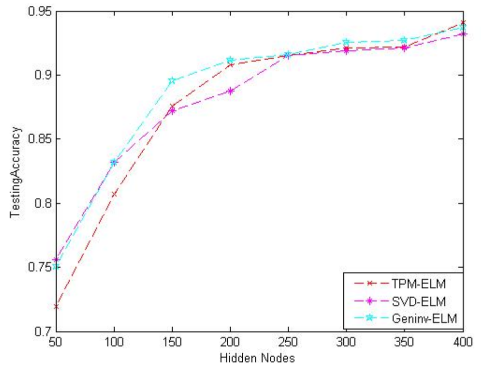

Figure 2. The testing accuracy of the three algorithms in DNA with different hidden nodes.

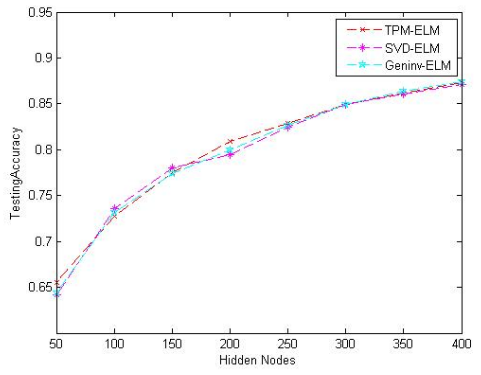

Figure 3. The testing accuracy of the three algorithms in Letter with different hidden nodes.

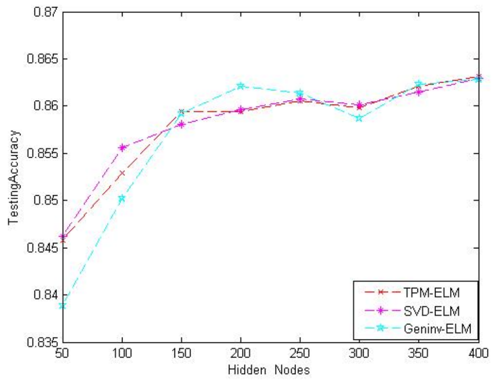

Figure 4. The testing accuracy of the three algorithms in Sat with different hidden nodes.

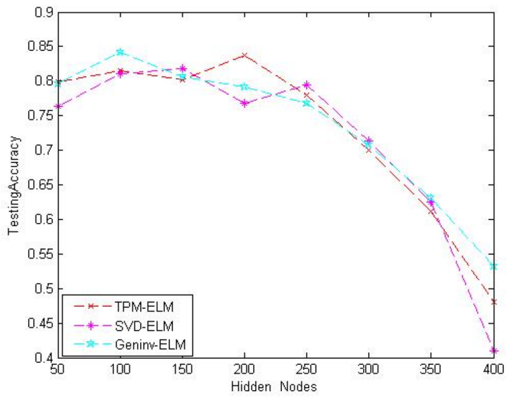

Figure 7. The testing accuracy of the three algorithms in Page with different hidden nodes.

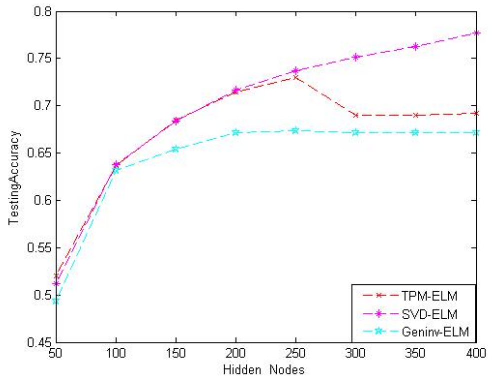

Figure 5. The testing accuracy of the three algorithms in Shuttle with different hidden nodes.

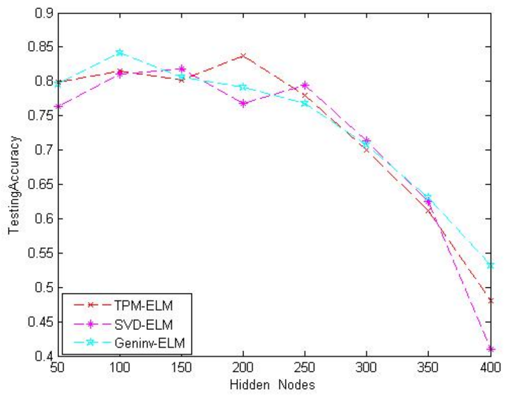

Figure 8. The testing accuracy of three algorithms in Vechicle with different hidden nodes.

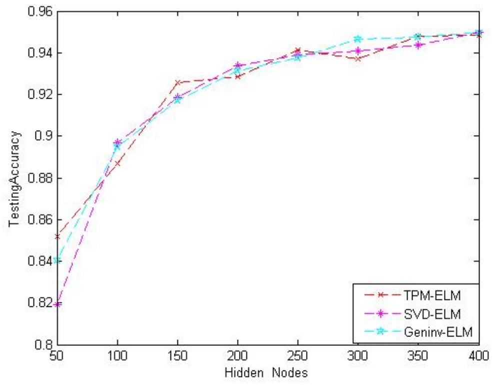

Figure 6. The testing accuracy of the three algorithms in Usps with different hidden nodes.

## REPFERENCES

[1] G.-B. Huang, Q.-Y. Zhu, C.-K. Siew, Extreme learning machine: theory and applications, Neurocomputing 70 (1–3) (2006) 489–501.

[2] P.-L. Bartlett, The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network, IEEE Trans. Inf. Theory 44 (2) (1998) 525–536.

[3] G.-B. Huang, D.-H. Wang, Y. Lan, Extreme learning machines: a survey, Int J Mach Learn Cybern 2(2) (2011) 107–122.

[4] G.-B. Huang, H.-A. Babri, Upper bounds on the number of hidden neurons in feedforward networks with arbitrary bounded nonlinear activation functions, IEEE Trans. Neural Networks 9 (1) (1998) 224–229.

[5] G.-B. Huang, H.-M. Zhou, X.-J. Ding, R. Zhang, Extreme Learning Machine for Regression and Multiclass Classification, IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics 42(2) (2012) 513-529.

[6] R. Zhang, Y. Lan, G.-B. Huang, Z.-B. Xu, Universal Approximation of Extreme Learning Machine with Adaptive Growth of Hidden Node, IEEE Transactions on Neural Networks and Learning Systems 23(2) (2012) 365–371.

[7] N.-Y. Liang, G.-B. Huang, P. Saratchandran, N. Sundararajan, A Fast and Accurate Online Sequential Learning Algorithm for Feedforward Networks, IEEE Trans. Neural Netw 17(6) (2006) 1411–1423.

[8] G.-B. Huang, L. Chen, Enhanced random search based incremental extreme learning machine, Neural Computing 71(2008) 3460–3468.

[9] G.-B. Huang, L. Chen, Convex incremental extreme learningmachine. Neurocomputing 70(16–18) (2007) 3056–3062

[10] A. Ben-Israel, T.-N.-E. Grenville, Generalized Inverses: Theory and Applications. Springer-Verlag. Berlin 2002.

[11] G.-H. Golub, C.-F.-V. Loan, Matrix Computations, thirded., Johns Hopkins University Press, MD, 1996.

[12] W. Guo, T. Huang, Method of elementary transformation to compute Moore-Penrose inverse, Applied Mathematics and Computation 216(5) (2010)1614–1617.

[13] F. Toutounian, A. Ataei, A New Method for Computing Moore-Penrose Inverse Matrices, Journal of Computational and Applied Mathematics 228(1) (2009)412–417.

[14] V.-N. Katsikis, D. Pappas, Fast computing of the Moore-Penrose inverse matrix, Electronic Journal of Linear Algebra 17 (2008) 637–650.

[15] P. Courrieu, Fast Computation of Moore-Penrose Inverse Matrices, In: Neural Information Processing- Letters and Reviews 8(2) (2005) 25–29.

[16] A. Frank, A. Asuncion,UCI Machine Learning Repository, 2010. URL <http://archive.ics.uci.edu/ml>.

[17] N. Vasilios, Katsikis, P. Dimitrios, P. Athanassios, An improved method for the computation of the Moore-Penrose inverse matrix, Applied Mathematics and Computation 217 (2011) 9828–9834.