Intelligent Data Analysis 19 (2015) 743–760
DOI 10.3233/IDA-150743
IOS Press

Effective algorithms of the Moore-Penrose inverse matrices for extreme learning machine

Shuxia Lu[a,b,*], Xizhao Wang[c], Guiqiang Zhang[a] and Xu Zhou[a]
[a]*College of Mathematics and Information Science, Hebei University, Baoding, Hebei, China*
[b]*Key Laboratory of Machine Learning and Computational Intelligence of Hebei Province, Baoding, Hebei, China*
[c]*College of Computer Science and Software, Shenzhen University, Shenzhen, China*

Abstract. Extreme learning machine (ELM) is a learning algorithm for single-hidden layer feedforward neural networks (SLFNs) which randomly chooses hidden nodes and analytically determines the output weights of SLFNs. After the input weights and the hidden layer biases are chosen randomly, ELM can be simply considered a linear system. However, the learning time of ELM is mainly spent on calculating the Moore-Penrose inverse matrices of the hidden layer output matrix. This paper focuses on effective computation of the Moore-Penrose inverse matrices for ELM, several methods are proposed. They are the reduced QR factorization with column Pivoting and Geninv ELM (QRGeninv-ELM), tensor product matrix ELM (TPM-ELM). And we compare QRGeninv-ELM, TPM-ELM with the relational algorithm of Moore-Penrose inverse matrices for ELM, the relational algorithms are: Cholesky factorization of singular matrix ELM (Geninv-ELM), QR factorization and Ginv ELM (QRGinv-ELM), the conjugate Gram-Schmidt process ELM (CGS-ELM). The experimental results and the statistical analysis of the experimental results both demonstrate that QRGeninv-ELM, TPM-ELM and Geninv-ELM are faster than other kinds of ELM and can reach comparable generalization performance.

Keywords: Extreme learning machine, tensor product matrix, cholesky factorization of singular matrix, conjugate gram-schmidt process, QR factorization

1. Introduction

Extreme learning machine (ELM) is one of the most popular neural networks for its simple structure and powerful approximation capability [1,2]. For function approximation in a finite training set, Huang and Babri [3] shows that a single-hidden layer feedforward neural network (SLFNs) with at most N hidden nodes and with almost any nonlinear activation function can exactly learn N distinct observations. Different from traditional learning algorithms the ELM not only tends to reach the smallest training error but also the smallest norm of weights. Bartlett [4] pointed out that for feedforward neural networks, the smaller the norm of weights and training error is, the better generalization performance the networks tend to have. Thus, in order to obtain the better generalization performance, we find the minimum norm least squares solution.

*Corresponding author: Shuxia Lu, College of Mathematics and Information Science, Hebei University, Baoding, Hebei 071002, China. E-mail: cmclusx@126.com.

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 [5–9]. For example online sequential extreme learning machine [10], a structure-adjustable online learning ELM [11], Bidirectional extreme learning machine [12] is quicker in learning speed. There have been some studies on the approximation capabilities of SLFNs [13,14]. The number of hidden nodes is determined in an adaptive way, there are normally two heuristic approaches to modify the structure of SLFNs: constructive methods (or growing methods) and destructive methods (or pruning methods) [15–19].

ELM can be simply considered as a linear system. However, the learning time of ELM is mainly spent on calculating the Moore-Penrose generalized inverse matrices of the hidden layer output matrix. We focus on effective computation of the Moore-Penrose inverse matrices for ELM. There are several methods for computing the Moore-Penrose inverse matrices [20]. These may include orthogonal projection, orthogonalization method, iterative method, and singular value decomposition (SVD) [21–28]. The orthogonalization method and iterative method have their limitations since searching and iteration are used. 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, $\mathbf{H}^T\mathbf{H}$ may not always nonsingular or may tend to be singular and thus orthogonal projection method may not perform well, and matrix decomposition techniques can be used. The SVD can be used to calculate the Moore-Penrose inverse of the hidden layer output matrix $\mathbf{H}$ in ELM. SVD is very accurate but also time-consuming since it requires a large mount of computational resources, especially in the case of large matrices.

Also, in the recent work, there are several methods for computing the Moore-Penrose inverse matrices by matrix decomposition techniques [29–33]. Horata, Chiewchanwattana, and Sunat [29] gave a comparative study of pseudo-inverse computing for the extreme learning machine classifier. Katsikis [30] provided a fast and reliable method to calculate the Moore-Penrose inverse and can be used for large sparse matrices. P. Courrieu [31] proposed a fast computation of Moore-Penrose inverse matrices based on Cholesky factorization of singular matrix. Toutounian and Ataei [32] presented the CGS-MPi algorithm based on the conjugate Gram-Schmidt process and the Moore-Penrose inverse of partitioned matrices, and they concluded that this algorithm is a robust and efficient tool for computing the Moore-Penrose inverse of large sparse and rank deficient matrices. Katsikis [33] proposed a method based on QR factorization in order to calculate the Moore-Penrose inverse of singular square matrices and of rectangular matrices.

In this paper, we focus on effective computation of the Moore-Penrose inverse matrices for ELM, several methods are proposed. They are QRGeninv-ELM and TPM-ELM. We compare QRGeninv-ELM, TPM-ELM with the relational algorithm of Moore-Penrose inverse matrices for ELM, the relational algorithms are: Geninv-ELM, QRGinv-ELM, CGS-ELM.

This paper is organized as follows. Section 2 outlined the related work of ELM and Moore-Penrose inverse matrices. Relational algorithms of Moore-Penrose inverse matrices for ELM are presented in Section 3. Then the proposed effective algorithms are described in Section 4. Performance evaluation is presented in Section 5. Conclusions are given in Section 6.

2. Background

This section will briefly describe the ELM and Moore-Penrose inverse matrices.

2.1. Extreme learning machine (ELM)

The extreme learning machine (ELM) is a single hidden layer feed forward network where the input weights and the biases are chosen randomly and the output weights are calculated analytically [1].

For N arbitrary distinct samples $(\mathbf{x}_i, \mathbf{t}_i)$, where $\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots x_{in}]^T \in \mathbb{R}^n$ and $\mathbf{t}_i = [t_{i1}, t_{i2}, \ldots t_{im}]^T \in \mathbb{R}^m$, standard SLFNs with L hidden nodes with activation function $g(x)$ can approximate these N distinct samples with zero error are modeled as

$$\sum_{i=1}^{L} \beta_i g(\mathbf{w}_i \cdot \mathbf{x}_j + b_i) = \mathbf{t}_j \quad j = 1, \ldots, N. \tag{1}$$

where $\mathbf{w}_i = [w_{i1}, w_{i2}, \ldots w_{in}]^T$ is the weight vector connecting the ith hidden node and the input nodes, $\beta_i = [\beta_{i1}, \beta_{i2}, \ldots, \beta_{im}]^T$ is the weight vector connecting the ith hidden node and the output nodes, and b_i is the threshold of the ith hidden node, $\mathbf{w}_i \cdot \mathbf{x}_j$ denotes the inner product of $\mathbf{w}_i$ and $\mathbf{x}_j$. L is the number of hidden nodes, activation function $g(x)$ is infinitely differentiable in any interval, These N equations can be written compactly as

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

where

$$\mathbf{H}(\mathbf{w}_1, \ldots, \mathbf{w}_L, b_1, \ldots, b_L, \mathbf{x}_1, \ldots, \mathbf{x}_N) = \begin{bmatrix} g(\mathbf{w}_1 \cdot \mathbf{x}_1 + b_1) & \ldots & g(\mathbf{w}_L \cdot \mathbf{x}_1 + b_L) \\ \ldots & \ldots & \ldots \\ g(\mathbf{w}_1 \cdot \mathbf{x}_N + b_1) & \ldots & g(\mathbf{w}_L \cdot \mathbf{x}_N + b_L) \end{bmatrix}_{N \times L}, \tag{3}$$

$\boldsymbol{\beta} = [\beta_1, \ldots, \beta_L]^T \in \mathbb{R}^{L \times m}$ and $\mathbf{T} = [\mathbf{t}_1, \ldots, \mathbf{t}_N]^T \in \mathbb{R}^{N \times m}$.

where $\mathbf{H}$ is called the hidden layer output matrix of the neural network.

The ELM algorithm of SLFNs can be summarized as the following three steps.

Algorithm ELM (*ELM* is denoted by *SVD-ELM*)

Given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) \mid \mathbf{x}_i \in \mathbb{R}^n, \mathbf{t}_i \in \mathbb{R}^m, i = 1, \ldots, N\}$, activation function $g(x)$, and hidden node number L,

Step 1: Randomly assign input weight $\mathbf{w}_i$ and bias $b_i (i = 1, \ldots, L)$. For any weights and biases are randomly chosen from any intervals of $\mathbb{R}^n$ and $\mathbb{R}$, respectively, according to any continuous probability distribution in Matlab (using Matlab *rand* function).

Step 2: Compute the hidden layer output matrix of the network denoted by $\mathbf{H} \in \mathbb{R}^{N \times L}$.

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

$$\boldsymbol{\beta} = \mathbf{H}^{\dagger}\mathbf{T}, \tag{4}$$

where $\mathbf{T} = [\mathbf{t}_1, \ldots, \mathbf{t}_N]^T \in \mathbb{R}^{N \times m}$. $\mathbf{H}^{\dagger}$ is the Moore-Penrose generalized inverse of matrix $\mathbf{H}$. $\mathbf{H}^{\dagger}$ can be obtained from the SVD (using Matlab *pinv* function).

2.1.1. Singular value decomposition (SVD)

The SVD was established for real square matrices in the 1870s by Beltrami and Jordan. We now state the SVD theorem and the reduced SVD [21].

Theorem 1. (SVD) Let $\mathbf{H} \in \mathbb{R}^{N\times L}$ and $r = rank(\mathbf{H})$. Then there exist orthogonal matrices $\mathbf{U} \in \mathbb{R}^{N\times N}$ and $\mathbf{V} \in \mathbb{R}^{L\times L}$ such that $\mathbf{H} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$, where $\mathbf{\Sigma} = \begin{bmatrix} \mathbf{S} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}$ and $\mathbf{S} = diag(\sigma_1, \sigma_2, \ldots, \sigma_r)$ with $\sigma_1 \geqslant \sigma_2 \geqslant \ldots \geqslant \sigma_r > 0$.

The numbers $\sigma_1, \sigma_2, \ldots, \sigma_r$ together with $\sigma_{r+1} = 0, \ldots, \sigma_L = 0$ are called the singular values of $\mathbf{H}$ and they are the positive square roots of the eigenvalues (which are nonnegative) of $\mathbf{H}^T\mathbf{H}$. The columns of $\mathbf{U}$ are called the left singular vectors of $\mathbf{H}$ (the orthonormal eigenvectors of $\mathbf{H}\mathbf{H}^T$) while the columns of $\mathbf{V}$ are called the right singular vectors of $\mathbf{H}$ (the orthonormal eigenvectors of $\mathbf{H}^T\mathbf{H}$).

$$\mathbf{H}^\dagger = \mathbf{V}\mathbf{\Sigma}^\dagger\mathbf{U}^T.$$

2.1.2. The reduced SVD

Let $\mathbf{H} \in \mathbb{R}^{N\times L}$ and $rank(\mathbf{H}) = L$. Then there exist orthogonal matrices $\mathbf{U} \in \mathbb{R}^{N\times N}$ and $\mathbf{V} \in \mathbb{R}^{L\times L}$ such that

$$\mathbf{H} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T = [\mathbf{U}_1\mathbf{U}_2]\begin{bmatrix} \mathbf{\Sigma}_1 \\ \mathbf{0} \end{bmatrix}\mathbf{V}^T = \mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}^T.$$

where $\mathbf{U}_1 \in \mathbb{R}^{N\times L}$ and $\mathbf{\Sigma}_1 \in \mathbb{R}^{L\times L}$ are nonsingular, this is called the reduced SVD of $\mathbf{H}$.

From above ELM (or SVD-ELM) algorithm, if we want to get good generalization performance, the most important step is to accurately calculate the Moore-Penrose generalized inverse matrix $\mathbf{H}^\dagger$.

2.2. Moore-Penrose inverse matrices

Moore defined a new inverse of matrix by projection matrices, which is now called as Moore-Penrose generalized inverse. Penrose gave an equivalent definition of Moore-Penrose generalized inverse [34].

Definition 1. ([34]) For $\mathbf{H} \in \mathbb{R}^{N\times L}$, if $\mathbf{G} \in \mathbb{R}^{L\times N}$ satisfies the following four equations:

$$\mathbf{HGH} = \mathbf{H},\ \ \mathbf{GHG} = \mathbf{G},\ \ (\mathbf{HG})^T = \mathbf{HG},\ \ (\mathbf{GH})^T = \mathbf{GH}.$$

Then matrix $\mathbf{G}$ is called the Moore-Penrose generalized inverse of matrix $\mathbf{H}$, denoted by $\mathbf{G} = \mathbf{H}^\dagger$.

Theorem 2. ([35,36]) Let there exist a matrix $\mathbf{G}$ such that $\mathbf{GT}$ is a minimum norm least squares solution of a linear system $\mathbf{H}\beta = \mathbf{T}$. Then it is necessary and sufficient that $\mathbf{G} = \mathbf{H}^\dagger$, the Moore–Penrose generalized inverse of matrix $\mathbf{H}$.

Remark 1. Seen from Theorem 2, we have the following property: the special solution $\boldsymbol{\beta} = \mathbf{H}^\dagger\mathbf{T}$ is one of the least squares solutions of a general linear system $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$; the special solution $\boldsymbol{\beta} = \mathbf{H}^\dagger\mathbf{T}$ has the smallest norm among all the least-squares solutions of $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$; the minimum norm least-squares solution of linear system is unique, which is $\boldsymbol{\beta} = \mathbf{H}^\dagger\mathbf{T}$. For the over-determined linear system $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$, if $\mathbf{H}$ has full rank, least squares solution of linear system would be unique, if $\mathbf{H}$ is rank deficient, the least squares solution of linear system must still exist, it cannot be unique in this case.

3. Relational algorithms of the Moore-Penrose inverse matrices for ELM

In this section, the matrix $\mathbf{H}^\dagger$ is calculated by matrix decomposition techniques. These methods may include Singular Value Decomposition (SVD), Cholesky factorization of singular matrix, and $\mathbf{QR}$ factorization. The relational algorithms are mainly based on the several methods for computing the Moore-Penrose inverse matrix [30–33], and we apply these methods to the extreme learning machine (ELM). They are the Cholesky factorization of singular matrix ELM (Geninv-ELM), QR factorization and Ginv ELM (QRGinv-ELM), the conjugate Gram-Schmidt process ELM (CGS-ELM).

3.1. Cholesky factorization of singular matrix ELM (Geninv-ELM)

Geninv-ELM provides a rapid method, which is based on a known reverse order law and Cholesky factorization of singular matrix. The computation time is substantially shorter, particularly for large least square 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 Courrieu [31].

The Geninv-ELM algorithm is described as follows.

Algorithm 1. Geninv-ELM

Given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) \mid \mathbf{x}_i \in \mathbb{R}^n, \mathbf{t}_i \in \mathbb{R}^m, i = 1, \ldots, N\}$, activation function $g(x)$, and hidden node number L, $L < N$.

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

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

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

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

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

(4) Calculate

$$(\mathbf{H}^T\mathbf{H})^\dagger = (\mathbf{R}^T\mathbf{R})^\dagger = \mathbf{R}^T(\mathbf{R}\mathbf{R}^T)^{-1}(\mathbf{R}\mathbf{R}^T)^{-1}\mathbf{R}. \tag{5}$$

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

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

$$\boldsymbol{\beta} = \mathbf{H}^\dagger\mathbf{T} = (\mathbf{H}^T\mathbf{H})^\dagger\mathbf{H}^T\mathbf{T}. \tag{6}$$

where $\mathrm{T} = [\mathbf{t}_1, \ldots, \mathbf{t}_N]^T \in \mathbb{R}^{N \times m}$.

Remark 2. *The Cholesky factorization* ([21]) If the matrix $\mathbf{A}$ is symmetric positive definite, then $\mathbf{A} = \mathbf{L}\mathbf{L}^T$, where $\mathbf{L}$ is lower triangular matrix and has positive diagonal entries (but not, in general, a unit diagonal).

The Cholesky factorization of singular matrix ([37]) Let $\mathbf{A}$ be a symmetric, possibly singular, semi-positive definite matrix of order $n \times n$. Then there is an upper triangular matrix X such that $\mathbf{A} = \mathbf{X}^T\mathbf{X}$, $x_{ii} \geqslant 0, 1 \leqslant i \leqslant n$, and if for an index i one has $x_{ii} = 0$, then $x_{ij} = 0, 1 \leqslant j \leqslant n$. Moreover, the matrix $\mathbf{X}$ with these properties is unique.

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 ($r \leqslant L$).

Matlab code of the "*geninv*" function can be found in Courrieu [31].

3.2. QR factorization and ginv ELM (QRGinv-ELM)

QRGinv-ELM provides a method for computing generalized inverses, which is based on the $\mathbf{QR}$ factorization and *ginv* function. The method of computing the Moore-Penrose inverse matrix is based on the work of Katsikis, Pappas, and Petralias [33].

Let $\mathbf{H} \in \mathbb{R}^{N\times L}$ be a matrix, with *rank*$(\mathbf{H}) = r > 0$. Then there exist matrices $\mathbf{M}$, $\mathbf{Q}$, $\mathbf{R}$ such that $\mathbf{M}$ is obtained by $\mathbf{H}$ by permuting its columns ($\mathbf{M} = \mathbf{HP}$, where $\mathbf{P} \in \mathbb{R}^{L\times L}$ is a permutation matrix), $\mathbf{Q} \in \mathbb{R}^{N\times N}$ is orthogonal, $\mathbf{R} = \begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \in \mathbb{R}^{N\times L}$, $\mathbf{R}_{11} \in \mathbb{R}^{r\times r}$ is nonsingular and upper triangular and $\mathbf{M} = \mathbf{QR}$, this is called the QR factorization.

$$\mathbf{H}^{\dagger} = \mathbf{P}\mathbf{R}^{\dagger}\mathbf{Q}^{T} \in \mathbb{R}^{L\times N}. \tag{7}$$

Algorithm 2. QRGinv-ELM

Given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) \mid \mathbf{x}_i \in \mathbb{R}^n, \mathbf{t}_i \in \mathbb{R}^m, i = 1, \ldots, N\}$, activation function $g(x)$, and hidden node number L.

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

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

(1) Using the QR factorization, we have that $\mathbf{HP} = \mathbf{QR}$, where $\mathbf{P} \in \mathbb{R}^{L\times L}$ is a permutation matrix, $\mathbf{Q} \in \mathbb{R}^{N\times N}$ is orthogonal, $\mathbf{R} = \begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \triangleq \begin{bmatrix} \mathbf{R}_1 \\ \mathbf{0} \end{bmatrix} \in \mathbb{R}^{N\times L}$, $\mathbf{R}_{11} \in \mathbb{R}^{r\times r}$ is nonsingular and upper triangular, $\mathbf{R}_1 \in \mathbb{R}^{r\times L}$.

(2) Calculate $\mathbf{R}_1^{\dagger} \in \mathbb{R}^{L\times r}$ by using *ginv* function, $\mathbf{R}_1^{\dagger} = ginv(\mathbf{R}_1)$.

(3) Add $N - r$ zero columns to $\mathbf{R}_1^{\dagger} \in \mathbb{R}^{L\times r}$, and get $\mathbf{R}^{\dagger} = \begin{bmatrix} \mathbf{R}_1^{\dagger} & \mathbf{0} \end{bmatrix} \in \mathbb{R}^{L\times N}$.

(4) We have that $\mathbf{H}^{\dagger} = \mathbf{P}\mathbf{R}^{\dagger}\mathbf{Q}^{T} \in \mathbb{R}^{L\times N}$.

Step 3: Calculate the output weight $\boldsymbol{\beta} \in \mathbb{R}^{L\times m}$. $\boldsymbol{\beta} = \mathbf{H}^{\dagger}\mathbf{T}$, where $\mathrm{T} = [\mathbf{t}_1, \ldots, \mathbf{t}_N]^T \in \mathbb{R}^{N\times m}$.

Matlab code of the "*ginv*" and "*qrginv*" functions can be found in [30,33], respectively.

3.3. The conjugate Gram-Schmidt process ELM (CGS-ELM)

The CGS-ELM algorithm is based on the conjugate Gram-Schmidt process and the Moore-Penrose inverse of partitioned matrices (Toutounian, and Ataei [32]). The method is a robust and efficient for large sparse and rank deficient matrices.

Definition 2. By assuming $\mathbf{H} \in \mathbb{R}^{N\times L}$ has full column rank, the $L\times L$ matrix $\mathbf{C} = \mathbf{H}^T\mathbf{H}$ is symmetric positive definite and therefore for two vectors $\mathbf{x}$ and $\mathbf{y}$ it defines the following $\mathbf{C}$-inner products:

$$\langle \mathbf{x}, \mathbf{y} \rangle_{\mathbf{C}} = \langle \mathbf{C}\mathbf{x}, \mathbf{y} \rangle_2, \tag{8}$$

where $\langle\cdot\rangle_2$ denotes the usual inner product in $\mathbb{R}^L$.

Theorem 3. Let the $r \times L$ matrix $\mathbf{W}$ be partitioned as $\mathbf{W} = [\mathbf{U} \;\; \mathbf{V}]$, where $\mathbf{U}$ is an $r \times r$ nonsingular matrix and $\mathbf{V}$ is an $r \times (L - r)$ matrix, then

$$\mathbf{W}^{\dagger} = \begin{bmatrix} \mathbf{I}_r - \mathbf{B}\mathbf{K}\mathbf{B}^T \\ \mathbf{K}\mathbf{B}^T \end{bmatrix} \mathbf{U}^{-1}, \tag{9}$$

where $\mathbf{B} = \mathbf{U}^{-1}\mathbf{V}$, $\mathbf{K} = (\mathbf{I}_{L-r} + \mathbf{B}^T\mathbf{B})^{-1}$ and $\mathbf{I}_r$ is an $r \times r$ unit matrix.

We can build a $\mathbf{C}$-orthogonal set of vectors $\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_L \in \mathbb{R}^L$ by a conjugate Gram-Schmidt process.

Let $\mathbf{z}_j^{(0)} = \mathbf{e}_j, j = 1, 2, \ldots, L$, where $\mathbf{e}_1, \mathbf{e}_2, \ldots, \mathbf{e}_L \in \mathbb{R}^L$ is the set of unit basis vectors,

$$\mathbf{z}_i^{(j)} \leftarrow \mathbf{z}_i^{(j-1)} - \frac{\langle \mathbf{z}_j^{(j-1)}, \mathbf{z}_i^{(j-1)} \rangle_{\mathbf{C}}}{\langle \mathbf{z}_j^{(j-1)}, \mathbf{z}_j^{(j-1)} \rangle_{\mathbf{C}}} \mathbf{z}_j^{(j-1)}, \quad j = 1, 2, \ldots, L-1 \text{ and } i = j+1, \ldots, L. \tag{10}$$

$\mathbf{Z} = (\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_L)$ is unit upper triangular, where $\mathbf{z}_i = \mathbf{z}_i^{(i-1)}$, and $\mathbf{D} = diag(d_1, d_2, \ldots, d_L)$ with $d_j = \langle \mathbf{z}_j, \mathbf{z}_j \rangle_C = \|\mathbf{H}\mathbf{z}_j\|_2^2 > 0$. If $\mathbf{H}$ has full column rank, then $\mathbf{H}^\dagger = (\mathbf{Z}\mathbf{D}^{-1}\mathbf{Z}^T)\mathbf{H}^T$.

Algorithm 3. CGS-ELM

Given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) \mid \mathbf{x}_i \in \mathbb{R}^n, \mathbf{t}_i \in \mathbb{R}^m, i = 1, \ldots, N\}$, activation function $g(x)$, and hidden node number L, $L < N$.

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

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

(1) If $\mathbf{H}$ has full column rank, then $\mathbf{H}^\dagger = (\mathbf{Z}\mathbf{D}^{-1}\mathbf{Z}^T)\mathbf{H}^T$, where the matrix $\mathbf{Z} = (\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_L) \in \mathbb{R}^{L \times L}$ is unit upper triangular, which computed by the conjugate Gram-Schmidt process.

(2) If the matrix $\mathbf{H}$ is rank deficient, $rank(\mathbf{H}) = r < L$

a. Utilize the Cholesky factorization of $\mathbf{P}^T\mathbf{H}^T\mathbf{H}\mathbf{P}$, one obtains $\mathbf{P}^T\mathbf{H}^T\mathbf{H}\mathbf{P} = \mathbf{R}^T\mathbf{R}$, where $\mathbf{R} = \begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} \triangleq \begin{bmatrix} \mathbf{W} \\ \mathbf{0} \end{bmatrix}$, $\mathbf{R}_{11}$ is $r \times r$ upper triangular with positive diagonal elements, $\mathbf{W} \in \mathbb{R}^{r \times L}$, $rank(\mathbf{W}) = r$, $\mathbf{P} \in \mathbb{R}^{L \times L}$ is a permutation matrix.

b. $\mathbf{P}^T\mathbf{H}^T\mathbf{H}\mathbf{P} = \mathbf{W}^T\mathbf{W}$.

c. $\mathbf{H}^\dagger = \mathbf{P}\mathbf{W}^\dagger(\mathbf{W}^\dagger)^T\mathbf{P}^T\mathbf{H}^T$, where the computation of $\mathbf{W}^\dagger$ is by using theorem 3.

Step 3: Calculate the output weight $\boldsymbol{\beta} \in \mathbb{R}^{L \times m}$. $\boldsymbol{\beta} = \mathbf{H}^\dagger\mathbf{T}$, where $\mathbf{T} = [\mathbf{t}_1, \ldots, \mathbf{t}_N]^T \in \mathbb{R}^{N \times m}$.

4. The proposed algorithms of the Moore-Penrose inverse matrices for ELM

In this section, we propose the effective learning algorithm of Moore-Penrose inverse matrices for ELM. They are the reduced QR factorization with column Pivoting and Geninv ELM (QRGeninv-ELM), tensor product matrix ELM (TPM-ELM). The proposed QRGeninv-ELM is based on the reduced QR factorization with column Pivoting and Geninv method. The TPM-ELM is the revised method (Katsikis, and Pappas [30]) for computing the Moore-Penrose inverse of rank deficient matrix.

4.1. The reduced QR factorization with column pivoting and geninv ELM (QRGeninv-ELM)

The proposed QRGeninv-ELM method is based on the reduced $\mathbf{QR}$ factorization with column Pivoting and *Geninv* method.

Suppose $\mathbf{H} \in \mathbb{R}^{N \times L}$ and $N \geqslant L$ with $rank(\mathbf{H}) = r \leqslant L$. QR with column Pivoting produces the factorization $\mathbf{H}\mathbf{P} = \mathbf{Q}\mathbf{R}$, where $\mathbf{P} \in \mathbb{R}^{L \times L}$ is a permutation matrix, $\mathbf{Q} \in \mathbb{R}^{N \times N}$ is orthogonal matrix, $\mathbf{R} \in \mathbb{R}^{N \times L}$ is an upper trapezoidal matrix. Assume that $\mathbf{P}$ is chosen so that $\mathbf{Q}$ and $\mathbf{R}$ can be partitioned as $\mathbf{Q} = [\mathbf{Q}_1 \ \ \mathbf{Q}_2]$, $\mathbf{R} = \begin{bmatrix} \mathbf{R}_{11} & \mathbf{R}_{12} \\ \mathbf{0} & \mathbf{0} \end{bmatrix} = \begin{bmatrix} \mathbf{R}_1 \\ \mathbf{0} \end{bmatrix}$, where $\mathbf{Q}_1 \in \mathbb{R}^{N \times r}$ is the first r columns

of $\mathbf{Q}$ and $\mathbf{R}_1 \in \mathbb{R}^{r\times L}$ is the first r rows of $\mathbf{R}$, $\mathbf{R}_{11} \in \mathbb{R}^{r\times r}$ is nonsingular and upper triangular. Then $\mathbf{HP} = \mathbf{QR} = \mathbf{Q}_1\mathbf{R}_1$, this is called the reduced (or "economy size") QR factorization.

$$\mathbf{H}^\dagger = \mathbf{P}\mathbf{R}_1^\dagger\mathbf{Q}_1^T \in \mathbb{R}^{L\times N}. \tag{11}$$

$\boldsymbol{\beta} = \mathbf{H}^\dagger\mathbf{T}$ minimizes $\|\mathbf{H}\boldsymbol{\beta} - \mathbf{T}\|_2$ and has the minimum-norm least squares solution of linear system $\mathbf{H}\boldsymbol{\beta} = \mathbf{T}$.

Algorithm 4. QRGeninv-ELM

Given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) \mid \mathbf{x}_i \in \mathbb{R}^n, \mathbf{t}_i \in \mathbb{R}^m, i = 1, \ldots, N\}$, activation function $g(x)$, and hidden node number L.

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

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

(1) Using the reduced QR factorization, we have that $\mathbf{HP} = \mathbf{Q}_1\mathbf{R}_1$, where $\mathbf{P} \in \mathbb{R}^{L\times L}$ is a permutation matrix, $\mathbf{Q} = [\mathbf{Q}_1, \mathbf{Q}_2] \in \mathbb{R}^{N\times N}$ is orthogonal, $\mathbf{R} = \begin{bmatrix}\mathbf{R}_1 \\ \mathbf{0}\end{bmatrix} \in \mathbb{R}^{N\times L}$ is an upper trapezoidal matrix, $\mathbf{Q}_1 \in \mathbb{R}^{N\times r}$ is the first r columns of $\mathbf{Q}$ and $\mathbf{R}_1 \in \mathbb{R}^{r\times L}$ is the first r rows of $\mathbf{R}$.

(2) Calculate $\mathbf{R}_1^\dagger \in \mathbb{R}^{L\times r}$ by using the method of calculating the Moore-Penrose generalized inverse in Geninv-ELM. $\mathbf{R}_1^\dagger = \textit{geninv}(\mathbf{R}_1)$

(3) Calculate $\mathbf{H}^\dagger = \mathbf{P}\mathbf{R}_1^\dagger\mathbf{Q}_1^T \in \mathbb{R}^{L\times N}$.

Step 3: Calculate the output weight $\boldsymbol{\beta} \in \mathbb{R}^{L\times m}$. $\boldsymbol{\beta} = \mathbf{H}^\dagger\mathbf{T}$, where $\mathbf{T} = [\mathbf{t}_1, \ldots, \mathbf{t}_N]^T \in \mathbb{R}^{N\times m}$.

Matlab code of the "*qrgeninv*" function is listed in the appendix.

4.2. Tensor product matrix ELM (TPM-ELM)

Katsikis and Pappas [30] provides a fast computational method in order to calculate the Moore-Penrose inverse of full rank matrices and of square matrices with at least one zero row or column. However, this method (using *ginv* function) is invalid when matrix is rank deficient. We revised the method (Katsikis and Pappas [30]) for computing the Moore-Penrose inverse of rank deficient matrix. The revised method is called the tensor product matrix ELM (TPM-ELM). TPM-ELM provides a rapid and reliable method and can be used for large matrices.

Algorithm 5. TPM-ELM

Given a training set $\aleph = \{(\mathbf{x}_i, \mathbf{t}_i) \mid \mathbf{x}_i \in \mathbb{R}^n, \mathbf{t}_i \in \mathbb{R}^m, i = 1, \ldots, N\}$, activation function $g(x)$, and hidden node number L,

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

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

if $\mathbf{H}$ has full rank,

if $N > L$, then $\mathbf{H}^\dagger = (\mathbf{H}^T\mathbf{H})\backslash\mathbf{H}^T$

if $L > N$, then $\mathbf{H}^\dagger = (\mathbf{H}\mathbf{H}^T)\backslash\mathbf{H}$.

if $\mathbf{H}$ is rank deficient,

if $N > L$, then $\mathbf{H}^\dagger = (\mathbf{H}^T\mathbf{H} + eye(size(\mathbf{H}, 2))/D)\backslash\mathbf{H}^T$, where D is a parameter

if $L > N$, then $\mathbf{H}^\dagger = (\mathbf{H}\mathbf{H}^T + eye(size(\mathbf{H}, 1))/D)\backslash\mathbf{H}$.

Step 3: Calculate the output weight $\boldsymbol{\beta} \in \mathbb{R}^{L\times m}$. $\boldsymbol{\beta} = \mathbf{H}^\dagger\mathbf{T}$, where $\mathbf{T} = [\mathbf{t}_1, \ldots, \mathbf{t}_N]^T \in \mathbb{R}^{N\times m}$.

Remark 3. The operator "\" means matrix left-division (which is an operator in MATLAB7.1). $\mathbf{C}\backslash\mathbf{H}$ is the matrix division of $\mathbf{C}$ into $\mathbf{H}$, which is roughly the same as $inv(\mathbf{C}) \cdot \mathbf{H}$, except it is computed in a different way. If $\mathbf{C}$ is an L-by-L matrix and $\mathbf{H}$ is a column vector with L components, or a matrix with several such columns, then $\mathbf{Y} = \mathbf{C}\backslash\mathbf{H}$ is the solution to the linear system $\mathbf{CY} = \mathbf{H}$ computed by Gaussian Elimination. If $\mathbf{C}$ is badly scaled or nearly singular. $\mathbf{C}\backslash$*eye* (*size* ($\mathbf{C}$)) produces the inverse of $\mathbf{C}$.

Matlab code of the "*TPM*" function is listed in the appendix.

5. Performance evaluation

In this section, we compare the performance of the six methods for the computation of Moore-Penrose inverse matrices used in ELM on a set of random matrices, random singular matrices (Higham [38]) and the twelve datasets from the UCI machine learning repository [39] and database [40]. All the simulations are carried out in MATLAB7.1 environment running in Core 2 Quad, 3.09 GHz CPU with 3.16 GB RAM. In our experiment, the activation function which has better performance is selected from among the *sine*, *sigmoid*, and *RBF* functions. The *sigmoid* function $g(\mathbf{x}) = 1/(1 + \exp(-\mathbf{x}))$ is used as an activation function for all methods.

In the real-world problems, we preprocess these datasets according to the following procedures.

1) Delete the nominal condition attributes.
2) Fill in the missing attribute values. We use the unsupervised filter named ReplaceMissingValues in WEKA [42] to fill in all the missing attribute values in each dataset.
3) All the inputs (attributes) have been normalized into the range [0, 1], while the outputs (targets) have been normalized into $[-1, 1]$.

5.1. Robustness

The robustness performance of the six methods was checked on a set of random matrices, random singular matrices, Page and Segment datasets. The random matrices are obtained using Matlab *rand* function. The random singular matrices (namely Chow, Gearmat and Lotkin) are taken from the function *matrix* in the Higham's Matrix Computation Toolbox (mctoolbox) (Higham [38]). The random singular matrices are of size 200×200 with large condition numbers (ill-conditioned) and rank deficient. The Page and Segment datasets are taken from the UCI repository of machine learning databases [39]. The number of Page and Segment sample N is 1155 and 2736 respectively, and the number of hidden nodes in the hidden layer L is 200 in ELM methods. The hidden layer output matrix $\mathbf{H}$ is of size 1155×200 and 2736×200 respectively. The six methods can be used to calculate the Moore-Penrose inverse of the hidden layer output matrix $\mathbf{H}$ in ELM.

The six methods for the computation of Moore-Penrose inverse matrices are: the QR factorization with column Pivoting and Geninv (QRGeninv), the tensor product matrix (TPM), the Cholesky factorization of singular matrix (Geninv), the QR factorization and Ginv (QRGinv), the conjugate Gram-Schmidt process (CGS), and the Singular Value Decomposition (SVD) (Matlab *pinv* function).

In addition, the accuracy of the results is examined with the matrix 2-norm in error matrices corresponding to the four properties characterizing the Moore-Penrose inverse. The accuracy of the results is verified using the following four relations which characterize the Moore-Penrose inverse:

$$\mathbf{H}\mathbf{H}^{\dagger}\mathbf{H} = \mathbf{H}, \quad \mathbf{H}^{\dagger}\mathbf{H}\mathbf{H}^{\dagger} = \mathbf{H}^{\dagger}, \quad (\mathbf{H}\mathbf{H}^{\dagger})^{T} = \mathbf{H}\mathbf{H}^{\dagger}, \quad (\mathbf{H}^{\dagger}\mathbf{H})^{T} = \mathbf{H}^{\dagger}\mathbf{H}$$

In Table 1, the accuracy of the six methods is tested based on the 2-norm errors. The worst error and

Table 1
Error and computational time results: Matrices

Matrix	Method	Time	$\|\mathbf{HH}^\dagger\mathbf{H}-\mathbf{H}\|_2$	$\|\mathbf{H}^\dagger\mathbf{HH}^\dagger-\mathbf{H}^\dagger\|_2$	$\|\mathbf{HH}^\dagger-(\mathbf{HH}^\dagger)^T\|_2$	$\|\mathbf{H}^\dagger\mathbf{H}-(\mathbf{H}^\dagger\mathbf{H})^T\|_2$
Rand	QRGeninv	0.65625	1.107e-013	1.4508e-016	7.6306e-015	2.9367e-015
(5000 × 200)	TPM	**0.26563**	5.0208e-013	1.3353e-015	8.3997e-016	1.3234e-014
rank = 200	Geninv	0.29688	2.6039e-011	2.979e-015	3.7493e-014	1.1837e-012
cond = 30.60668	QRGinv	28.9844	2.4777e-011	2.7778e-015	4.5354e-014	1.2071e-012
	CGS	1.5313	2.0464e-012	1.3386e-015	2.5514e-015	8.7002e-014
	SVD	0.875	8.7312e-013	4.5767e-016	4.3165e-014	2.735e-014
Chow	QRGeninv	0.0625	3.0915e-013	1.7863e-014	4.1056e-013	2.415e-014
(200 × 200)	TPM	**0.015625**	**0.01923**	**0.073958**	2.8981e-015	4.9614e-012
rank = 199	Geninv	0.046875	1.0327e-009	3.4006e-012	4.7847e-012	3.8534e-010
cond	QRGinv	0.0625	3.0915e-013	1.7863e-014	4.1056e-013	2.415e-014
= 3.7908e+017	CGS	1.1406	**253.5073**	**3.5192**	**10.1016**	**10.1016**
	SVD	0.09375	5.7239e-013	6.5282e-014	1.712e-013	1.6634e-013
Gearmat	QRGeninv	0.0625	3.1255e-015	1.5293e-013	8.3603e-014	1.7612e-014
(200 × 200)	TPM	**0.015625**	**0.049911**	**3.2298**	2.4622e-016	1.87e-014
rank = 199	Geninv	0.046875	5.5754e-012	1.8232e-011	3.3859e-012	2.4882e-011
cond	QRGinv	0.0625	3.1255e-015	1.5293e-013	8.3603e-014	1.7612e-014
= 5.3667e+016	CGS	1.125	9.2847e-014	4.2644e-013	2.0529e-016	7.4349e-013
	SVD	0.10938	1.6939e-014	2.838e-013	6.6898e-014	6.4645e-014
Lotkin	QRGeninv	**0.015625**	8.347e-006	3.5206e-009	0.04629	2.9699e-012
(200 × 200)	TPM	**0.015625**	**0.049781**	**2.7219**	1.1881e-015	2.0678e-013
rank = 19	Geninv	**0.015625**	**0.037166**	**7714.2718**	0.00056153	**1054.6227**
cond	QRGinv	**0.015625**	9.7623e-006	**5472091.6085**	**0.095973**	0.0001883
= 9.6941e+020	CGS	No results	No results	No results	No results	No results
	SVD	0.0625	3.0803e-005	**119555442.363**	0.00065397	0.00037043
Page	QRGeninv-ELM	0.29688	7.0864e-010	2.8684e-007	6.9357e-010	1.2828e-010
$N = 2736$, $L = 200$	TPM-ELM	**0.14063**	**0.15811**	**1.0266**	8.6231e-015	4.1631e-011
H(2736 × 200)	Geninv-ELM	0.21875	**2.4706**	**24.095**	0.000525	**0.24911**
rank(H) = 200	QRGinv-ELM	8.3281	1.2303e-009	5.3354e-007	8.6712e-009	9.9916e-011
cond(H)	CGS-ELM	1.6875	0.022683	**11.4082**	7.1211e-005	0.01641
= 1.4788e+007	SVD-ELM	0.59375	1.7051e-009	3.0629e-007	2.4082e-009	1.2121e-010
Segment	QRGeninv-ELM	0.10938	2.7835e-012	1.3133e-011	1.8313e-012	7.2352e-013
$N = 1155$, $L = 200$	TPM-ELM	**0.0625**	1.9471e-010	1.1222e-006	4.1617e-013	2.6133e-008
H(1155 × 200)	Geninv-ELM	0.10938	8.1325e-006	5.2527e-007	4.0931e-009	6.1584e-007
rank(H) = 200	QRGinv-ELM	1.1875	2.8998e-007	5.1662e-007	1.1844e-009	1.771e-007
cond(H)	CGS-ELM	1.8281	3.0501e-007	3.7765e-007	4.4997e-010	1.4709e-007
= 3.3722e+004	SVD-ELM	0.23438	4.5478e-012	1.0073e-011	6.8231e-012	5.7238e-013

the shortest time are given in boldface in Table 1. It is evident that the six methods are all accurate on random matrices and Segment dataset, and the TPM method is the fastest algorithm. As observed from the Table 1, the TPM, Geninv and CGS methods are less reliable than the QRGeninv method on the random singular matrices and Page dataset. The QRGinv method is faster than the SVD method on the random singular matrices (namely Chow, Gearmat and Lotkin). The QRGeninv method is fast and accurate computation of Moore-Penrose inverse matrices on all kinds of matrices.

5.2. Performance in the real-world problems

In this section, we conduct the performance comparison of the six methods for twelve real problems: Digit, DNA, Letter, Magic, Mushroom, Page, Sat, Segment, Shuttle, Spambase, Usps and Vehicle. Most of the datasets are taken from the UCI machine learning repository [39]. Usps is taken from

Table 2
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
Magic	10	16	9510	9510	400
Mushroom	22	2	3762	1882	400
Page	10	5	2736	2737	250
Sat	36	7	3217	3218	400
Segment	19	7	1155	1155	250
Shuttle	9	7	29000	29000	250
Spambase	57	2	2300	2301	400
Usps	256	10	6198	3100	400
Vehicle	18	4	423	423	150

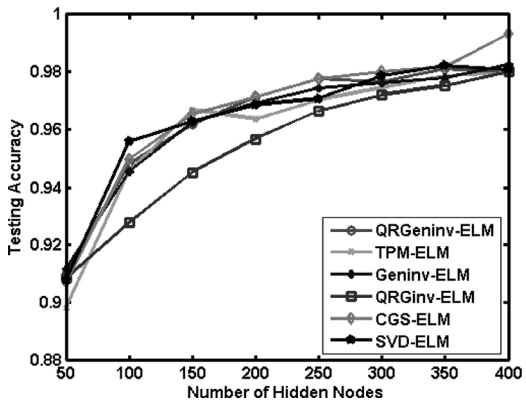

Fig. 1. The testing accuracy of the six algorithms in Digit with different number of hidden nodes.

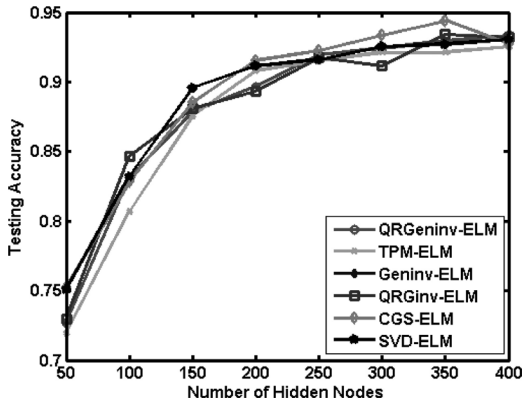

Fig. 2. The testing accuracy of the six algorithms in DNA with different number of hidden nodes.

database [40]. The numbers of attributes, classes, samples for training and testing, and hidden nodes are shown in Table 2.

For the QRGeninv-ELM, TPM-ELM, Geninv-ELM, QRGinv-ELM, CGS-ELM, and SVD-ELM methods, the number of hidden nodes L is searched by means of trial and error method with grid search technique. L is set to be 50, 100, 150, 200, 250, 300, 350, and 400. The nearly optimal numbers of hidden nodes for the six methods are then selected based on thirty trials. The testing accuracy of the six algorithms in twelve datasets with different number of hidden nodes is reported in Figs 1–12. From the fluctuating curves, we obtain values of L selected for each application. The value of L for Vehicle is 150. The value of L for Segment is 250. The values of L for Shuttle and Page are 200. The values of L for Digit, DNA, Letter, Magic, Mushroom, Sat, Spambase, and Usps are 400. The nearly optimal numbers of hidden nodes for the six methods are shown in Table 3.

Take Shuttle (large number of training samples) and Usps (medium size of dataset with high input dimensions) datasets as examples:

1) For Shuttle datasets, in Fig. 9, as the number of hidden nodes is gradually increased, the testing accuracy of the QRGeninv-ELM, Geninv-ELM, QRGinv-ELM, and SVD-ELM methods increases. However, the testing accuracy of the TPM-ELM and CGS-ELM increases at the beginning and then starts to decrease. The deviating range is small and relatively stable, ranging from 0.975 to 0.995. The value of L for Shuttle is 200.

Table 3
Comparison of testing accuracy of the six methods in the real-world problems

Data sets	# hidden nodes	QRGeninv-ELM	TPM-ELM	Geninv-ELM	QRGinv-ELM	CGS-ELM	SVD-ELM
Digit	**400**	0.97943	0.98032	0.98093	0.9822	0.9804	0.97993
DNA	**400**	0.93031	0.93493	0.93112	0.93245	0.93245	0.93169
Letter	**400**	0.87231	0.87247	0.87308	0.87151	0.87246	0.87248
Magic	**400**	0.86145	0.86239	0.8621	0.8639	0.86199	0.8623
Mushroom	**400**	0.99989	1	1	1	1	1
Sat	**400**	0.89018	0.88748	0.88943	0.8872	0.8900	0.89068
Spambase	**400**	0.89535	0.89431	0.8857	0.8905	0.8966	0.89444
Usps	**400**	0.95271	0.95142	0.95094	0.9552	0.9548	0.95203
Vehicle	**150**	0.81868	0.81939	0.81135	0.82128	0.81962	0.81915
Segment	**250**	0.93983	0.94156	0.94338	0.94459	0.94165	0.94277
Shuttle	**200**	0.99622	0.99588	0.99347	No results	0.99531	0.99628
Page	**200**	0.95086	0.95301	0.95342	0.95294	0.95411	0.95228

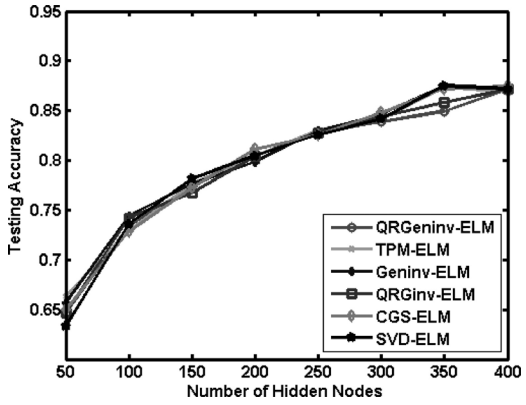

Fig. 3. The testing accuracy of the six algorithms in Letter with different number of hidden nodes.

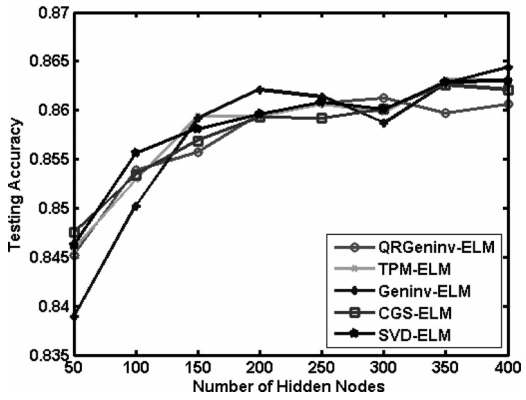

Fig. 4. The testing accuracy of the five algorithms in Magic with different number of hidden nodes.

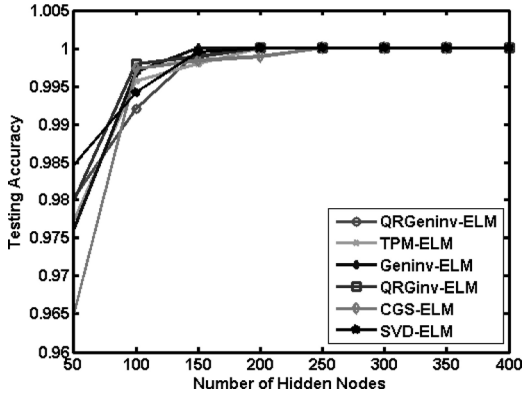

Fig. 5. The testing accuracy of the six algorithms in Mushroom with different number of hidden nodes.

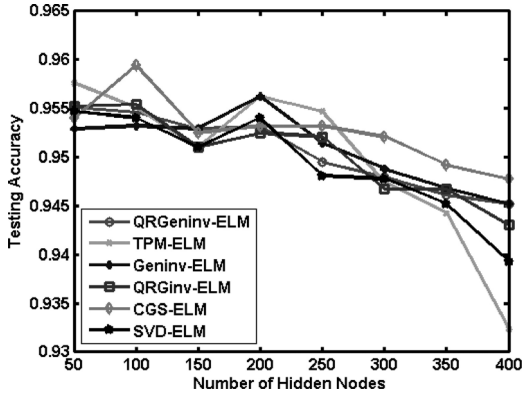

Fig. 6. The testing accuracy of the six algorithms in Page with different number of hidden nodes.

2) For Usps datasets, in Fig. 11, as the number of hidden nodes is gradually increased, the testing accuracy of the six methods increases. The value of L for Usps is 400.

As observed from Figs 1–12, the generalization performance of ELM is relatively stable on a wide

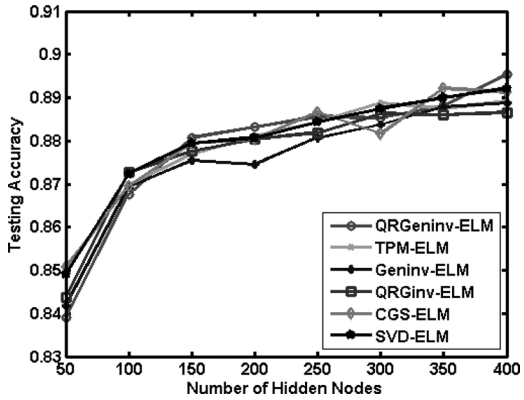

Fig. 7. The testing accuracy of the six algorithms in Sat with different number of hidden nodes.

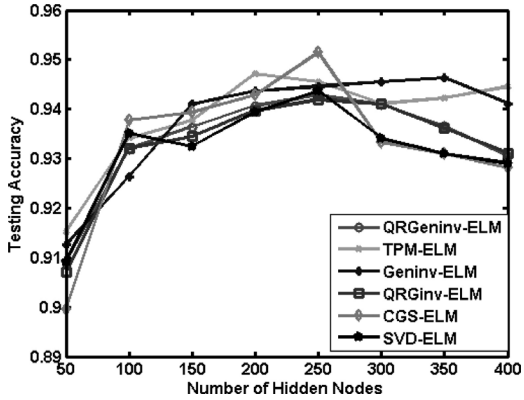

Fig. 8. The testing accuracy of the six algorithms in Segment with different number of hidden nodes.

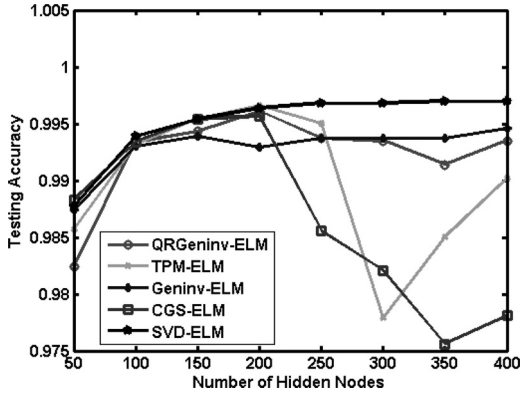

Fig. 9. The testing accuracy of the five algorithms in Shuttle with different number of hidden nodes.

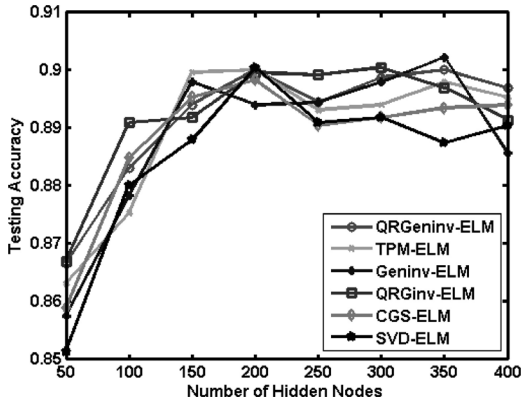

Fig. 10. The testing accuracy of the six algorithms in Spambase with different number of hidden nodes.

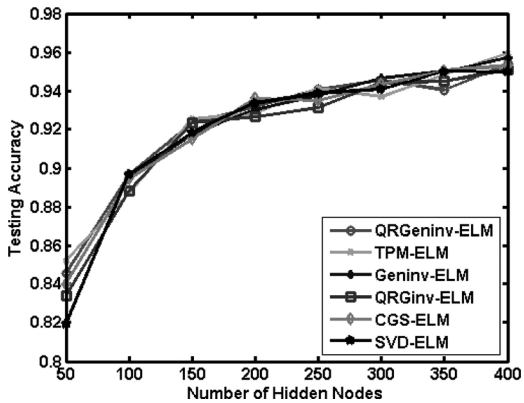

Fig. 11. The testing accuracy of the six algorithms in Usps with different number of hidden nodes.

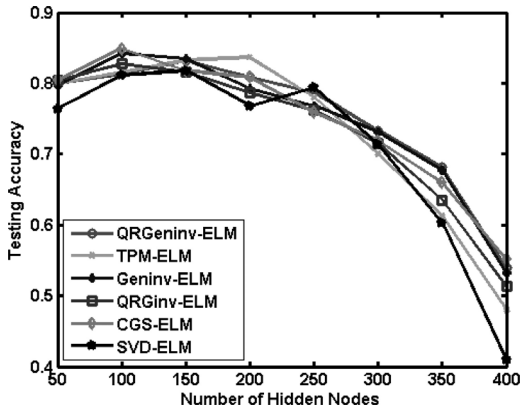

Fig. 12. The testing accuracy of the six algorithms in Vechicle with different number of hidden nodes.

Table 4
Comparison of average training time of the six methods in the real-world problems

Data sets	# hidden nodes	QRGeninv-ELM	TPM-ELM	Geninv-ELM	QRGinv-ELM	CGS-ELM	SVD-ELM
Digit	400	0.67812	**0.21563**	0.58281	6.7344	8.9219	1.4484
DNA	400	0.96562	**0.34219**	0.76406	10.3219	0.76719	1.7891
Letter	400	2.6367	**0.68672**	1.6195	92.007	10.4383	3.9148
Magic	400	2.7031	**0.60781**	1.5094	79.8281	10.4375	3.7625
Mushroom	400	0.95781	**0.2625**	0.70625	12.0625	9.2813	1.7047
Sat	400	0.78281	**0.22969**	0.63438	8.7813	8.9531	1.5063
Spambase	400	0.65938	**0.175**	0.50313	4.3750	8.8438	1.1234
Usps	400	1.7938	**0.68594**	1.3172	37.5156	9.8906	2.9328
Vehicle	150	0.01875	**0.010937**	0.025	0.046875	0.28594	0.045312
Segment	250	0.095312	**0.042188**	0.11563	0.71094	1.5922	0.24688
Shuttle	200	3.0438	**0.6**	1.2813	No results	2.4992	3.3484
Page	200	0.15313	**0.059375**	0.14531	3.8922	0.87344	0.29531

range of number of hidden nodes although the generalization performance tends to become worse when too few or too many nodes are randomly generated.

We compare the performance of six methods (QRGeninv-ELM, TPM-ELM, Geninv-ELM, QRGinv-ELM, CGS-ELM, and SVD-ELM) in the real-world problems. The *sigmoid* function is used as an activation function for all methods. For any weights and biases are randomly chosen from any intervals of $\mathbb{R}^n$ and $\mathbb{R}$, respectively, according to any continuous probability distribution in Matlab (using Matlab *rand* function). The learning times of the six methods are mainly spent on calculating the Moore-Penrose inverse of the hidden layer output matrix. Thirty trials have been conducted for all the ELM algorithms and the average results are shown in Tables 3 and 4. The best results are given in boldface in Tables. Table 3 shows the performance comparison of testing accuracy of the six methods in the real-world problems. As observed from the Table 3, general speaking, all of methods obtain similar testing accuracy.

For Shuttle datasets, the QRGinv-ELM algorithm fails to produce a numerical result (Matlab produced an "out of memory" message), but the other algorithms obtain similar testing accuracy. For Usps datasets, all of methods obtain similar testing accuracy.

Whenever the activation function is taken as *sigmoid*, the deduced hidden layer output matrix is of full column rank; therefore the generalized inverse technique can be efficiently applied to yield the solution an ELM-like system.

Table 4 shows the performance comparison of average training time of the six methods in the real-world problems. As observed from the Table 4, 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 learns up to 5–7 times faster than SVD-ELM. Geninv-ELM learns up to 2–3 times faster than SVD-ELM. QRGeninv-ELM learns up to 1.5–2.6 times faster than SVD-ELM. QRGinv-ELM is slower than SVD-ELM.

For Shuttle dataset, QRGinv-ELM algorithm fails to produce a numerical result (Matlab produced an "out of memory" message). TPM-ELM and Geninv-ELM learn up to 2–5 times faster than the other ELM methods. For Usps datasets, TPM-ELM learns up to 2 times faster than Geninv-ELM and QRGeninv-ELM. Geninv-ELM and QRGeninv-ELM learn up to 2–18 times faster than SVD-ELM, CGS-ELM, and QRGinv-ELM. CGS-ELM is slower than SVD-ELM on all the datasets except for DNA (medium size of data set with high input dimensions) and Shuttle (large number of training samples) datasets.

QRGeninv-ELM, TPM-ELM and Geninv-ELM are fast and accuracy methods on the real word benchmark datasets (medium and large datasets).

Table 5
h-value and p-value of the pairwise T-test comparisons between the algorithms on average training time

Datasets	QRGeninv-ELM SVD-ELM	TPM-ELM SVD-ELM	Geninv-ELM SVD-ELM	QRGeninv-ELM TPM-ELM	TPM-ELM Geninv-ELM
	h/p value	h/p value	h/p value	h/p value	h/p value
Digit	1/1.6112e-028	1/4.3901e-033	1/6.3310e-030	1/6.5212e-027	1/1.2450e-030
DNA	1/2.8931e-028	1/1.5877e-034	1/7.4455e-029	1/1.7292e-028	1/2.9452e-022
Letter	1/7.8508e-069	1/9.0341e-095	1/3.7301e-086	1/1.0336e-051	1/2.6144e-078
Magic	1/1.8193e-028	1/7.2457e-042	1/4.7787e-039	1/3.1332e-035	1/7.2342e-033
Mushroom	1/4.8115e-027	1/5.8550e-034	1/9.0921e-031	1/3.4780e-028	1/6.6760e-033
Sat	1/5.0753e-037	1/1.3289e-040	1/1.6942e-035	1/3.3145e-034	1/5.0098e-029
Spambase	1/3.0107e-008	1/1.9737e-036	1/9.6126e-033	1/3.5533e-024	1/2.7802e-030
Usps	1/6.0846e-030	1/2.0068e-038	1/1.1900e-037	1/3.7383e-031	1/1.0300e-028
Vehicle	1/4.0465e-010	1/2.9967e-008	1/6.1790e-005	**0/0.1285**	**1/0.0147**
Segment	1/1.5293e-020	1/1.3408e-020	1/1.1833e-019	1/3.4446e-009	1/1.8522e-012
Shuttle	**1/0.0015**	1/1.1383e-040	1/3.0105e-036	1/5.4694e-018	1/1.5850e-029
Page	1/1.4199e-021	1/6.0316e-026	1/3.9005e-020	1/1.5501e-015	1/2.8412e-016

Table 6
h-value and p-value of the pairwise T-test comparisons between the algorithms on testing accuracy

Datasets	QRGeninv-ELM SVD-ELM	TPM-ELM SVD-ELM	Geninv-ELM SVD-ELM	QRGeninv-ELM TPM-ELM	TPM-ELM Geninv-ELM
	h/p value	h/p value	h/p value	h/p value	h/p value
Digit	0/0.4449	0/0.5520	0/0.1364	0/0.3271	0/0.4222
DNA	0/0.5801	0/0.1440	0/0.8032	0/0.5802	0/0.0840
Letter	0/0.8064	0/0.9830	0/0.4901	0/0.9830	0/0.4901
Magic	0/0.0190	0/0.8744	0/0.6868	0/0.8637	0/0.6281
Mushroom	0/0.2805	0/0.3293	0/0.3293	0/0.3293	0/0.3293
Sat	0/0.6513	**1/0.0029**	0/0.2742	0/0.2004	0/0.0566
Spambase	0/0.5522	0/0.9322	**1/0.0089**	0/0.8657	**1/0.0089**
Usps	0/0.5537	0/0.6448	0/0.3598	0/0.3810	0/0.7034
Vehicle	0/0.9398	0/0.9686	0/0.1314	0/0.8140	0/0.2323
Segment	0/0.0836	0/0.3764	0/0.6848	0/0.0740	0/0.2034
Shuttle	0/0.6087	1/0.0213	**1/2.1572e-010**	0/0.2377	**1/8.9866e-009**
Page	0/0.1046	0/0.4006	0/0.0955	0/0.4320	0/0.6248

5.3. Statistical comparisons of classifiers

To verify whether the improvements are statistically significant, we perform the pairwise T-test (Matlab *ttest2* function) [41]. The null hypothesis of the pairwise T-test is that the average training times of the algorithms come from distributions with equal means, the testing accuracies of the algorithms come from distributions with equal means. The h-value and p-value of the pairwise T-test are reported as measures of statistical significance. $h = 1$ indicates that the null hypothesis can be rejected at the 0.05 or 0.01 significance level (one algorithm is significantly different from the other). A small p-value implies that one algorithm is significantly different from the other. The pairwise T-test is used as shown in Tables 5 and 6. The bad results are given in Tables 5 and 6. Table 5 shows h-value and p-value of the pairwise T-test comparisons between the QRGeninv-ELM, TPM-ELM, Geninv-ELM and SVD-ELM algorithms on average training time. Therefore, it is obvious that the performance of QRGeninv-ELM, TPM-ELM and Geninv-ELM algorithms on average training time is significantly different from that of the SVD-ELM. The performance of TPM-ELM algorithms on average training time is significantly different from that of the Geninv-ELM. Table 6 shows h-value and p-value of the pairwise T-test comparisons between the QRGeninv-ELM, TPM-ELM, Geninv-ELM and SVD-ELM algorithms on testing accuracy.

The performance of QRGeninv-ELM, TPM-ELM and Geninv-ELM algorithms on testing accuracy is not significantly different from that of the SVD-ELM in many cases.

6. Conclusions

This paper focuses on effective computation of the Moore-Penrose inverse matrices for ELM, several methods are proposed. They are QRGeninv-ELM and TPM-ELM. The QRGeninv-ELM is based on the reduced QR factorization with column Pivoting and Geninv method. The TPM-ELM is the revised method for computing the Moore-Penrose inverse of rank deficient matrix. And we compare QRGeninv-ELM, TPM-ELM with the relational algorithm, the relational algorithms are: Geninv-ELM, QRGinv-ELM, CGS-ELM. The experimental results and the statistical analysis of the experimental results both demonstrate that QRGeninv-ELM, TPM-ELM and Geninv-ELM are faster than other kinds of ELM and can reach comparable generalization performance, which could greatly speed up the application running time in many problem domains, so these three methods will be extensively used for ELM training. However, QRGeninv-ELM is more robust than TPM-ELM and Geninv-ELM.

Acknowledgments

This research is supported by the National Natural Science Foundation of China (61170040 and 71371063), by the Natural Science Foundation of Hebei Province (F2015201185, F2013201110, F2013201220), and by the Key Scientific Research Foundation of Education Department of Hebei Province (ZD20131028).

References

[1] G.B. Huang, Q.Y. Zhu and C.K. Siew, Extreme learning machine: Theory and applications, *Neurocomputing* **70** (2006), 489–501.
[2] G.B. Huang, D.H. Wang and Y. Lan, Extreme learning machines: A survey, *Int J Mach Learn Cybern* **2**(2) (2011), 107–122.
[3] G.B. Huang and 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.
[4] 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.
[5] G.B. Huang, H.M. Zhou, X.J. Ding and 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] E.S. Olivas, J.G. Sanchis, J.D. Martín et al., BELM: Bayesian extreme learning machine, *IEEE Trans Neural Networks* **22**(3) (2011), 505–509.
[7] W.Y. Deng, Q.H. Zheng and L. Chen, Regularized extreme learning machine, in: *Computational Intelligence and Data Mining*, 2009, CIDM'09, IEEE Symposium on 60825202 (2009), 389–395.
[8] Y. Wang, F. Cao and Y. Yuan, A study on effectiveness of extreme learning machine, *Neurocomputing* **74**(16) (2011), 2483–2490.
[9] G.B. Huang, X.J. Ding and H.M. Zhou, Optimization method based extreme learning machine for classification, *Neurocomputing* **74**(1–3) (2010), 155–163.
[10] N.Y. Liang, G.B. Huang, P. Saratchandran and N. Sundararajan, A fast and accurate online sequential learning algorithm for feedforward networks, *IEEE Trans Neural Netw* **17**(6) (2006), 1411–1423.
[11] G.H. Li, M. Liu and M.Y. Dong, A new online learning algorithm for structure-adjustable extreme learning machine, *Computers and Mathematics with Applications* **60**(3) (2010), 377–389.
[12] Y.M. Yang, Y.N. Wang and X.F. Yuan, Bidirectional extreme learning machine for regression problem and its learning effectiveness, *IEEE Trans Neural Networks* **23**(9) (2012), 1498–1505.

[13] G.B. Huang, L. Chen and C.K. Siew, Universal approximation using incremental constructive feed forward networks with random hidden nodes, *IEEE Trans Neural Network* **17**(4) (2006), 879–892.
[14] R. Zhang, Y. Lan, G.B. Huang and 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.
[15] G.B. Huang and L. Chen, Convex incremental extreme learning machine, *Neurocomputing* **70**(16–18) (2007), 3056–3062.
[16] G.B. Huang and L. Chen, Enhanced random search based incremental extreme learning machine, *Neurocomputing* **71** (2008), 3460–3468.
[17] G.R. Feng, G.B. Huang, Q.P. Lin and R. Gay, Error minimized extreme learning machine with growth of hidden nodes and incremental learning, *IEEE Trans Neural Netw* **20**(8) (2009), 1352–1357.
[18] H.J. Rong, Y.S. Ong, A.H. Tan and Z.X. Zhu, A fast pruned-extreme learning machine for classification problem, *Neurocomputing* **72**(1–3) (2008), 359–366.
[19] Y. Miche, A. Sorjamaa, P. Bas, O. Simula, C. Jutten and A. Lendasse, OP-ELM: Optimally pruned extreme learning machine, *IEEE Trans Neural Networks* **21**(1) (2010), 158–162.
[20] A. Ben-Israel and T.N.E. Grenville, Generalized inverses: Theory and applications, Springer-Verlag. Berlin 2002.
[21] G.H. Golub and C.F.V. Loan, Matrix computations, thirded, Johns Hopkins University Press, MD, 1996.
[22] S. Ferrari and R.F. Stengel, Smooth function approximation using neural networks, *IEEE Trans Neural Networks* **16**(1) (2005), 24–38.
[23] A. Ben-Israel and D. Cohen, On iterative computation of generalized inverses and associated projections, *SlAM J Numer Anal* **3**(3) (1966), 410–419.
[24] L. Chen, E.V. Krishnamurthy and I. Macleod, Generalized matrix inversion and rank computation by successive matrix powering, *Parallel Comput* **20**(3) (1994), 297–311.
[25] Y. Wei, J. Cai and M.K. Ng, Computing moore-penrose inverse of toeplitz matrices by Newton's iteration, *Mathematical and Computer Modeling* **40**(1–2) (2004), 181–191.
[26] P.S. Stanimirović and D.S. Cvetković-Ilić, Successive matrix squaring algorithm for computing outer inverses, *Appl Math Comput* **203**(11) (2008), 19–29.
[27] F. Huang and X. Zhang, An improved Newton iteration for the weighted moore-penrose inverse, *Applied Mathematics and Computation* **174**(2) (2006), 1460–1486.
[28] M.D. Petkovic and P.S. Stanimirovic, Iterative method for computing the Moore-Penrose inverse based on Penrose equations, *Journal of Computational and Applied Mathematics* **235**(6) (2011), 1604–1613.
[29] P. Horata, S. Chiewchanwattana and K. Sunat, A comparative study of pseudo-inverse computing for the extreme learning machine classifier, in: *Proceedings – 3rd International Conference on Data Mining and Intelligent Information Technology Applications*, ICMIA 2011, (2011) 40–45. (URL: http://tinyurl.com/bhflyqx)
[30] V.N. Katsikis and D. Pappas, Fast computing of the moore-penrose inverse matrix, *Electronic Journal of Linear Algebra* **17** (2008), 637–650.
[31] P. Courrieu, Fast computation of moore-penrose inverse matrices, in: *Neural Information Processing – Letters and Reviews* **8**(2) (2005), 25–29.
[32] F. Toutounian and A. Ataei, A new method for computing moore-penrose inverse matrices, *Journal of Computational and Applied Mathematics* **228**(1) (2009), 412–417.
[33] V.N. Katsikis, D. Pappas and A. Petralias, An improved method for the computation of the moore-penrose inverse matrix, *Applied Mathematics and Computation* **217** (2011), 9828–9834.
[34] R. Penrose, A generalized inverse for matrices, *Proc Camb Phil Soc* **51** (1955), 406–413.
[35] D. Serre, Matrices: Theory and applications, Springer, New York, 2002.
[36] C.R. Rao and S.K. Mitra, Generalized inverse of matrices and its applications, Wiley, New York, 1971.
[37] P. Courrieu, Straight monotonic embedding of data sets in euclidean spaces, *Neural Networks* **15** (2002), 1185–1196.
[38] N.J. Higham, The Matrix computation toolbox, <http://www.ma.man.ac.uk/higham/mctoolbox>.
[39] A. Frank and A. Asuncion, UCI machine learning repository, 2010. URL <http://archive.ics.uci.edu/ml>.
[40] J.J. Hull, A database for handwritten text recognition research, *IEEE Trans Pattern Anal Mach Intell* **16**(5) (1994), 550–554.
[41] J. Demsar, Statistical comparisons of classifiers over multiple data sets, *J Mach Learn Res* **7**(1) (2006), 1–30.
[42] I.H. Witten and E. Frank, Data mining: Practical machine learning tools and techniques, San Mateo, CA: Morgan Kaufmann, 2005.

Appendix

A. The qrgeninv function

```
function  Y = qrgeninv(H)
[m,n] = size(H); [Q,R,P] = qr(H,0);
r = sum(any(abs(R) >1e-5,2));
if r == n;
Y(P,:)=R(1:n,:)\ Q';
else % When H is rank deficient.
R2=geninv(R(1:r,:));
Y(P,:)=R2* Q(:,1:r)';
End
```

B. The TPM function

```
function TPM = TPM(H)
[n,m] = size(H);
r=rank(H);
if r<min(m,n);
    if m<n;
       TPM=(eye(size(H,2))/C+ H'*H)\H';
    else
        TPM=(eye(size(H,1))/C+ H*H')\H;
    end
else
    if m<n;
        TPM=(H'*H)\H';
    else
        TPM=(H*H')\H;
    end
end
```