Some properties of various types of matrix factorization

Wei Shean Ng[1*], and *Wei Wen* Tan[1]

[1]Department of Mathematical and Actuarial Sciences, Lee Kong Chian Faculty of Engineering and Science, Universiti Tunku Abdul Rahman, Malaysia

Abstract. Matrix factorizations or matrix decompositions are methods that represent a matrix as a product of two or more matrices. There are various types of matrix factorizations such as LU factorization, Cholesky factorization, singular value decomposition etc. Matrix factorization is widely used in pattern recognition, image denoising, data clustering etc. Motivated by these applications, some properties and applications of various types of matrix factorizations are studied. One of the purposes of matrix factorization is to ease the computation. Thus, comparisons in term of computation time of various matrix factorizations in different areas are carried out.

1 Introduction

Matrix factorizations or matrix decompositions are methods that represent a matrix as a product of two or more matrices. Matrix factorization is usually used to simplify the computations in solving a problem which is relatively difficult to solve in its original form. We list the notations and definitions used in this paper in the next section.

1.1 Notations and definitions

Let m and n be positive integers and let $\mathcal{M}_{m,n}(\mathbb{F})$ denote the linear space of $m \times n$ matrices over a field $\mathbb{F}$. We use $\mathcal{M}_n(\mathbb{F})$ for $\mathcal{M}_{n,n}(\mathbb{F})$. A^t stands for the transpose of A. If $A = (a_{ij}) \in \mathcal{M}_n(\mathbb{C})$, $\bar{A} = (\overline{a_{ij}})$. Here, $\bar{z}$ is the conjugate of any $z \in \mathbb{C}$. A matrix $A \in \mathcal{M}_n(\mathbb{C})$ is said to be *Hermitian* if $\bar{A}^t = A$, and $A \in \mathcal{M}_n(\mathbb{R})$ is *symmetric* if $A^t = A$. $A = (a_{ij}) \in \mathcal{M}_{m,n}(\mathbb{F})$ is *lower triangular* if $a_{ij} = 0$ for all $i < j$. A is *upper triangular* if $a_{ij} = 0$ for all $i > j$. $A \in \mathcal{M}_n(\mathbb{F})$ is *diagonal* if $a_{ij} = 0$ for all $i \neq j$. A matrix $A \in \mathcal{M}_n(\mathbb{C})$ is a *positive definite Hermitian matrix* if for any nonzero vector $\mathbf{x}$, $\bar{\mathbf{x}}^t A\mathbf{x} > 0$. In particular, $A \in \mathcal{M}_n(\mathbb{R})$ is a *positive definite symmetric matrix* if for any nonzero vector $\mathbf{x}$, $\mathbf{x}^t A\mathbf{x} > 0$. Frobenius norm of matrix $A = (a_{ij}) \in \mathcal{M}_{m,n}(\mathbb{R})$ is defined as

*Corresponding author: ngws@utar.edu.my

$$\|A\|_F = \left(\sum_{i=0}^{m}\sum_{i=0}^{m} a_{ij}^2\right)^{\frac{1}{2}}.$$

Next, we state the definitions of various types of matrix factorizations. An *LU factorization* of a matrix A is to represent $A \in \mathcal{M}_{m,n}(\mathbb{F})$ as a product of a lower triangular matrix, $L \in \mathcal{M}_m(\mathbb{F})$ and an upper triangular matrix, $U \in \mathcal{M}_{m,n}(\mathbb{F})$, i.e. $A = LU$. Cholesky factorization is a factorization of a positive-definite Hermitian matrix or a positive-definite symmetric matrix A into the form $A = L\bar{L}^t$, where L is a lower triangular matrix with nonnegative diagonal entries. QR factorization is a factorization of a matrix $A \in \mathcal{M}_{m,n}(\mathbb{C})$ of rank r in the form $A = QR$, where $Q \in \mathcal{M}_m(\mathbb{C})$ such that $\bar{Q}^t Q = I_m$ and $R \in \mathcal{M}_{m,n}(\mathbb{C})$ is an upper triangular matrix. A *singular value decomposition* (SVD) is a factorization of $A \in \mathcal{M}_{m,n}(\mathbb{C})$ in the form $A = U\Sigma\bar{V}^t$ where $V \in \mathcal{M}_{m,k}(\mathbb{C})$ such that $\bar{U}^t U = I_k$, $V \in \mathcal{M}_{n,k}(\mathbb{C})$ such that $\bar{V}^t V = I_k$ and Σ is a $k \times k$ diagonal matrix with all positive entries. Nonnegative matrix factorisation (NMF) is a group of algorithms where a nonnegative matrix, $A \in \mathcal{M}_{m,n}(\mathbb{R})$, is factorized into $A = WH$ where $W \in \mathcal{M}_{m,k}(\mathbb{R})$, $H \in \mathcal{M}_{k,n}(\mathbb{R})$ are two nonnegative matrices by minimizing the distance between A and WH, $\min\|A - WH\|_F^2$.

2 Some properties and applications of matrix factorizations

LU factorization method was introduced by a mathematician named Tadeusz Banachiewicz in 1938 [1]. In 2014, Sudipto and Anindya [2] proved that a non-singular matrix $A \in \mathcal{M}_n(\mathbb{R})$ has an LU factorization if and only if all its leading principal submatrices are non-singular. They also proved that LU factorization of a non-singular matrix A is unique. LU factorization is useful in solving a linear system. As stated by Molala [3], matrix factorization helps to ease certain operations of the matrix by breaking down the matrix into smaller pieces. Let A be a non-singular matrix where $A = LU$. Then, $\det(A) = \det(LU) = \det(L)\det(U)$. Since L and U are both triangular matrices, determinants of L and U can be found by finding the product of the diagonal entries of the respective matrix. Similarly, if the inverse of A is to be found, instead of finding the inverse of A directly, one may find the inverses of L and U which are much easier. Hence, $A^{-1}=U^{-1}L^{-1}$.

Cholesky factorization was developed by André-Louis Cholesky who was a French military officer and mathematician [4]. He used the factorization in his surveying work. Cholesky factorization is a factorization of a positive-definite Hermitian matrix or a positive-definite symmetric matrix A into $A = L\bar{L}^t$. [2] mentioned that if A is a positive definite symmetric matrix, Cholesky factorization of A always exists. In many econometric contexts, matrices used are mostly positive definite symmetric matrices. An algorithm for finding the inverse of a matrix by using Cholesky factorization was proposed by Krishnamoorthy and Menon in 2013 [5]. The algorithm reduces the number of operations by 16% to 17%. It is achieved by avoiding computation of some known intermediate results. [5] proposed to find the inverse of a positive definite matrix $A \in \mathcal{M}_n(\mathbb{C})$, let $X \in \mathcal{M}_n(\mathbb{C})$ such that $X = A^{-1}$. Then $AX = I$. By Cholesky factorization, $A = L\bar{L}^t$. By letting $U = \bar{L}^t$, $A = \bar{U}^t U$. Thus, $\bar{U}^t UX = I$. If $UX = B$ then $\bar{U}^t B = I$. Since $X = A^{-1}$, $X = U^{-1}(\bar{U}^t)^{-1}$ and hence $B = UU^{-1}(\bar{U}^t)^{-1} = (\bar{U}^t)^{-1} = L^{-1}$. Thus, matrix B is a lower triangular matrix with diagonal elements as reciprocals of the diagonal elements of L. The authors constructed a matrix $S = (s_{ij})$ where

$$s_{ij} = \begin{cases} \frac{1}{l_{ii}}, & \text{if } i = j, \\ 0, & \text{otherwise.} \end{cases}$$

The matrix S is the solution of the upper diagonal elements of matrix B and backward substitution can be used to solve for x_{ij} using the equation $Ux_i = s_i$.

Ye and Qin [6] use QR factorization to determine the number of hidden nodes in generalized single-hidden-layer feedforward networks. In [7, 8], parallel QR factorization algorithms are used in shared memory, synchronous message passing, and asynchronous message passing. QR factorization is also used in subspace based blind channel identification [9, 10]. According to the paper, the QR factorization method is computationally more efficient because the method requires fewer computations compared to SVD.

In 2009, Konda and Nakamura [11] introduced an algorithm for SVD in parallel computing. Besides parallel computing, SVD is also used in compression to obtain low bit rate and good quality image coding [12]. Sudipto and Anindya [2] proved that it is always possible to factorize a matrix $A \in \mathcal{M}_{n,r}(\mathbb{C})$ in the form $A = U\Sigma V^t$ where U, Σ and V are unique, U and V are real orthogonal matrices such that $UU^t = I$ and $VV^t = I$, and Σ is a diagonal matrix with the diagonal entries are in decreasing order along the diagonal. In the example below, each row of matrix A contains the ratings (1 – 5) of a person on five different movies. If a movie is rated zero, it means the person has never watched the movie. The matrix that we obtain is

$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \\ e \\ f \\ g \end{array} \begin{array}{c} \begin{array}{ccccc} \text{Taken} & \text{TheMatrix} & \text{Avengers} & \text{Camille} & \text{Casablance} \end{array} \\ \begin{pmatrix} 2 & 2 & 2 & 0 & 1 \\ 2 & 2 & 2 & 0 & 0 \\ 4 & 4 & 4 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 5 & 5 \\ 0 & 2 & 0 & 2 & 2 \\ 0 & 0 & 0 & 3 & 3 \end{pmatrix} \end{array}$$

and hence SVD of $A = U\Sigma V^t$, where

$$U = \begin{pmatrix} 0.29 & 0.05 & 0.05 & 0.95 & 0 \\ 0.28 & -0.03 & 0.01 & -0.09 & 0 \\ 0.57 & -0.06 & 0.03 & -0.18 & 0 \\ 0.71 & -0.08 & 0.03 & -0.22 & 0 \\ 0.05 & 0.81 & 0.26 & -0.07 & -0.51 \\ 0.11 & 0.32 & -0.95 & 0 & 0 \\ 0.03 & 0.48 & 0.16 & -0.04 & 0.86 \end{pmatrix}, \Sigma = \begin{pmatrix} 12.2 & 0 & 0 & 0 & 0 \\ 0 & 8.74 & 0 & 0 & 0 \\ 0 & 0 & 1.54 & 0 & 0 \\ 0 & 0 & 0 & 0.68 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\text{and } V = \begin{pmatrix} 0.57 & -0.07 & 0.41 & -0.02 & -0.71 \\ 0.59 & 0 & -0.81 & 0.01 & 0 \\ 0.57 & -0.07 & 0.41 & -0.02 & 0.71 \\ 0.04 & 0.7 & 0.02 & -0.71 & 0 \\ 0.07 & 0.71 & 0.06 & 0.7 & 0 \end{pmatrix}.$$

In this example, the first three movies are action movies while the other two are romance. The first column of matrix U represents the action concept of the movies as we can see that the first four persons tend to watch action movies. The second column represents the romance concept of the movies, thus the last three rows of the second column have higher values. Matrix Σ represents the strength of the concepts. We can see the fourth and fifth concepts have a very low strength and thus we can ignore them in our data. Matrix V is a movie to concept similarity matrix. We can see that the first three movies correspond heavily to the action concept. From this example, we can see that SVD is good at reducing the dimension of a matrix while keeping the important information.

In data mining applications, the matrices obtained are often nonnegative. Thus, NMF is used to factorize a nonnegative matrix into two nonnegative matrices. NMF is not unique and convergence is not guaranteed for all NMF algorithms. In 1999, Lee and Seung [13] investigated the properties of the algorithm for finding NMF and hence NMF has become more widely known. NMF and principal component analysis (PCA) are used to extract facial features [13]. A face is represented by a matrix A and factorize $A = WH$ where W represents the basis components of a face and H represents the weightage of each components. They have also published some simple and useful algorithms in [14]. They showed that NMF is able to learn to represent an object as a combination of various parts. [15] mentioned that NMF is extensively used in machine learning. It is used to analyze high-dimensional data because it can extract sparse and meaningful features from nonnegative data vectors. A local coordinate-based graph regularized NMF method (LCGNMF) was proposed [16]. The sparse coefficients are induced by the proposed LCGNMF. In addition, the geometric structure of dataspace was considered.

3 Comparisons

In this section, we compare the effectiveness of various matrix factorizations in solving system of linear equations, finding the inverse of a matrix and image processing. We have used Python to carry out these comparisons. Matrices are generated randomly by using Python.

3.1 Solving system of linear equations

Consider a system of linear equations with n equations and n unknowns, $A\mathbf{x} = \mathbf{b}$ where $A \in \mathcal{M}_n(\mathbb{F})$ is non-singular. There are many ways to solve such linear system. The linear system can be solved by finding the inverse of A, A^{-1} and compute $\mathbf{x} = A^{-1}\mathbf{b}$. It can also be solved by using LU factorization, QR factorization, Cholesky decomposition, and singular value decomposition (SVD). We compare the efficiency of the above-mentioned methods in solving the linear system. We have applied the methods on matrix A when it is a real matrix, Hermitian matrix, and complex matrix.

The time (in second) taken to find $\mathbf{x}$ in the equation $A\mathbf{x} = \mathbf{b}$ by using different types of matrix factorization is shown in Table 1. We first use Python to randomly generate a 1000×1000 non-singular matrix A and a column vector $\mathbf{b}$ with 1000 entries. Then, we use different methods to solve the system of linear equations and the computation time is recorded. The same process is repeated for other types of matrices which include Hermitian matrix and complex matrix.

Table 1. Computation time (in second) when n is 1000.

Types of factorization	**Real**	**Hermitian**	**Complex**	**Complexity**
LU	0.0210	0.0198	0.0801	$O(\frac{2}{3}n^3)$
QR	0.1344	0.1446	0.3619	$O(\frac{4}{3}n^3)$
Inverse	0.0866	0.0935	0.2268	$O(n^3)$
Cholesky	N/A	0.0152	N/A	$O(\frac{1}{3}n^3)$
SVD	0.6369	0.6240	1.5364	N/A

Table 2 shows the results when A is a 1500×1500 non-singular matrix whereas Table 3 shows the results when A is a 2000×2000 non-singular matrix.

Table 2. Computation time (in second) when n is 1500.

Types of factorization	**Real**	**Hermitian**	**Complex**	**Complexity**
LU	0.0857	0.0649	0.2004	$O(\frac{2}{3}n^3)$
QR	0.4237	0.4265	1.0117	$O(\frac{4}{3}n^3)$
Inverse	0.2481	0.2251	0.6243	$O(n^3)$
Cholesky	N/A	0.0467	N/A	$O(\frac{1}{3}n^3)$
SVD	2.1644	2.1896	4.9181	N/A

Table 3. Computation time (in second) when n is 2000.

Types of factorization	**Real**	**Hermitian**	**Complex**	**Complexity**
LU	0.1507	0.1470	0.4573	$O(\frac{2}{3}n^3)$
QR	0.7993	0.7782	2.2404	$O(\frac{4}{3}n^3)$
Inverse	0.5523	0.5026	1.5084	$O(n^3)$
Cholesky	N/A	0.0971	N/A	$O(\frac{1}{3}n^3)$
SVD	5.3346	5.2530	11.6344	N/A

As mentioned in Section 1, Cholesky factorization is a factorization of Hermitian matrices. Thus, the method is used in the comparison for Hermitian matrices only. From the results shown in the three tables above, we found that Cholesky factorization is the most efficient way in solving the linear system when A is Hermitian. However, when Cholesky factorization is not applicable, *LU* factorization is the fastest among the methods used.

The *time complexity* as shown in the three tables above is defined to be the computational complexity that describes the amount of time taken to run an algorithm. An algorithm with time complexity $O(n^k)$ for some constant k is a *polynomial time algorithm*. If *LU* factorization is computed using Gaussian elimination, the time complexity is $O(\frac{2}{3}n^3)$. If *QR* factorization is computed using householder reflections, the time complexity is $O(\frac{4}{3}n^3)$ which is two times slower than LU factorization. However, the time taken shown in the tables are not only two times slower. This is because there is a built-in function in Python that solve a linear system by using LU factorization with partial pivoting. If a linear system is solved by using the inverse, it is much slower than solving the linear system by using LU factorization as shown in the results obtained. The reason is to find the matrix, A^{-1} is similar to solving the linear system $AX = I$ for $X \in \mathcal{M}_n(\mathbb{F})$ where more unknowns are involved. Therefore, more processing time is needed. The time complexity for Cholesky factorization is $O(\frac{1}{3}n^3)$ which is two times faster than LU factorization.

3.2 Matrix inversion

In this section, we study the efficiency of LU factorization, QR factorization and SVD in computing the inverse of $A \in \mathcal{M}_n(\mathbb{F})$ when A is real or Hermitian. The following three tables show the time taken (in second) to find the inverse of a matrix A. Table 4 shows the results when A is 1000 × 1000. Table 5 shows the results when A is 1500 × 1500 whereas Table 6 shows the results when A is 2000 × 2000.

Table 4. Computation time (in second) for 1000 × 1000 matrix.

Types of factorization	**Real**	**Hermitian**
LU	0.1736	0.1982
QR	0.2213	0.2364
Cholesky	N/A	0.0900
SVD	0.7984	0.8419

Table 5. Computation time (in second) for 1500 × 1500 matrix.

Types of factorization	**Real**	**Hermitian**
LU	0.5236	0.5239
QR	0.6411	0.6324
Cholesky	N/A	0.2259
SVD	2.8032	2.7690

Table 6. Computation time (in second) for 2000 × 2000 matrix.

Types of factorization	**Real**	**Hermitian**
LU	1.1448	1.2217
QR	1.2414	1.3603
Cholesky	N/A	0.6075
SVD	5.7349	6.4952

Again, we observe that Cholesky factorization is the fastest among the tested factorization methods in computing the inverse of a matrix whenever it is applicable. One of the reasons is that Cholesky factorization is two times faster than LU factorization based on the complexity of the two methods. Another reason is if we factorized A by using Cholesky factorization, i.e. $A = L\bar{L}^t$, only L^{-1} needs to be found to obtain $A^{-1} = (\bar{L}^{-1})^t L^{-1}$. However, if $A = LU$, L^{-1} and U^{-1} need to be found in order to obtain $A^{-1} = U^{-1}L^{-1}$. When comparing with QR factorization, LU factorization is slightly faster in finding the inverse of A. Q is an orthogonal matrix where $Q^{-1} = Q^t$. This should have eased the computation of finding $A^{-1} = R^{-1}Q^{-1}$. However, based on the complexity of LU and QR factorizations, computation of QR factorization requires longer time than LU factorization. The time taken for SVD is much longer and hence SVD is not a good way in finding the inverse of a matrix.

3.3 Image processing

SVD and NMF are commonly used in image processing. One of the reasons is when a matrix is factorized using SVD and NMF, the sizes of the factorized matrices are smaller compared to the original matrix. However, in LU and QR factorizations, the sizes of the factorized matrices remain the same. Thus, we only compare the speed of SVD and NMF in compressing an image in this section.

Every image is made up of pixels and each pixel of an RGB image is made up of three colors, red, green and blue. An RGB image is split into three matrices, each represents one color. Since each color is 8-bit, each entry ranges from 0 to 255. NMF is used as all the entries are nonnegative. The entries of each matrix indicate the intensity of the color in the respective pixel. We consider an image with 512 × 512 pixels as shown in Fig. 1. This implies that the matrix size is 512 × 512. After the matrices of the image are extracted, we apply SVD or

Fig. 1. Original image.

NMF to compress the matrix. For comparison, the matrix is compressed to the same size in pixels. The number of components in NMF is the same with the rank of matrix in SVD. The compressed images with rank 80 are shown in Fig. 2.

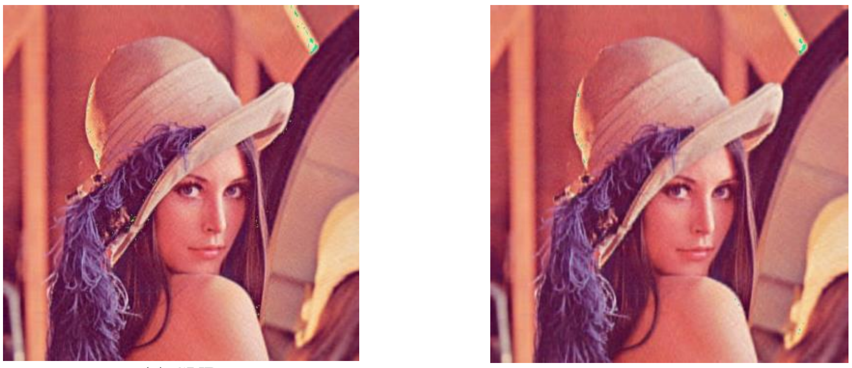

Fig. 2. Compressed images with rank 80.

The results in Table 7 show that NMF needs more time than SVD to compress an image. In addition, the time taken for NMF increase much faster comparing with SVD when the number of components increases.

Table 7. Processing time (in second) of 512 × 512 pixel image.

Rank	10	20	40	80	160
Size in pixels	30750	61500	123000	246000	492000
SVD	0.20616	0.21229	0.20856	0.21763	0.21533
NMF	0.43383	0.77515	1.80757	5.60797	22.66388

Frobenius norm is used to compute the distance of two matrices are. Therefore, the results in Table 8 show that SVD has lower approximation errors than NMF. However, SVD does not always preserve the nonnegativity of the matrix. Hence, NMF is one of the most popular low-rank approximations used for image processing, text mining etc.

Table 8. Frobenius norm.

Rank / **Frobenius norm**	**10**	**20**	**40**	**80**	**160**
SVD	149148	148286	147186	143387	132159
NMF	149870	149666	148761	146099	139626

4 Conclusion

This paper is an overview of various matrix factorizations. We have done three types of comparisons. These comparisons are the preliminary results of our future project. We shall continue this research by performing more comparisons.

The authors would like to express their sincere appreciation to Universiti Tunku Abdul Rahman (Malaysia) for all kind of support in working out this paper.

References

1. A. Schwarzenberg-Czerny, Astron. Astrophys. **110**, 405 (1995)
2. B. Sudipto, R. Anindya, Linear algebra and matrix analysis for statistics (CRC Press, 2014)
3. R. Molala, accessed 20 Feb 2020, https://blog.clairvoyantsoft.com/eigen-decompositionand-pca-c50f4ca15501 (2019)
4. J.J. O'Connor, E.F. Robertson, MacTutor history of mathematics archive, University of St Andrews (2003)
5. A. Krishnamoorthy, D. Menon, *Matrix inversion using Cholesky decomposition*, in Proceedings of the International Conference on SPA, 70 – 72 (2013)
6. Y. Ye, Y. Qin, Pattern Recognit. Lett. **65**, 177 – 183 (2015)
7. I.N. Dunn, G.G.L. Meyer, J. Franklin Inst. **338**, 601 – 613 (2001)
8. I.N. Dunn, G.G.L. Meyer, Parallel Comput. **28**, 1507 – 1530 (2002)
9. X. Li, H. Fan, IEEE Trans. on Signal Processing **48**, 60 – 69 (2000)
10. C.Y. Chen, J.S. Yu, Procedia Eng. **29**, 3413 – 3417 (2012)
11. T. Konda, Y. Nakamura, Parallel Comput. **35**, 331 – 334 (2009)
12. J.F. Yang, C.L. Lu, IEEE Trans. Image Processing **4**(8) 1141 – 1146 (1995)
13. D. Lee, H. Seung, Nature **401**, 788 – 791(1999)
14. D. Lee, H. Seung, Adv. Neural Inf. Process. Syst. **13**, 535–541 (2001)
15. N. Gillis, Reg, Opt, Kernels, and SVM **12**, 257 – 291 (2014)
16. Q. Liao, Q. Zhang, Signal Process. **124**, 103 – 114 (2016)