Singular Value Decomposition in Image Compression

Tram Ngo

November 2021

Abstract

This paper examines a matrix factorization concept: Singular Value Decomposition (SVD) in Image Compression. In practice, images usually need to be compressed. For example, images on the web are compressed for faster initial website rendering. Images are also compressed to save space on disk. This paper explains how one can use SVD to compress images by reducing the dimensionality of the data. In particular, this paper includes an overview of image compression in practice, a complete mathematical description of SVD, and the connections between the two. Examples will be included such as code, graph, and images. Finally, shortcomings of SVD will be discussed in order for us to look at SVD from all angles.

Keywords: Singular Value Decomposition, matrix factorization, image compression, machine learning, principle component analysis.

1 Introduction

From an early age, we are familiarized with the notion of factorization of an integer, e.g: 8 = 2 x 4, or of a polynomial, e.g: $x^2 - 1 = (x - 1)(x + 1)$. The purpose of such a factorization is to break a complex term, be it a number or a polynomial, into multiple simpler terms. The concept of matrix factorization (also called matrix decomposition) is not any different. A matrix, defined by Horn and Johnson as a rectangular array of numbers arranged in rows and columns [1], can also be broken up into multiple simpler matrices by a matrix factorization method.

Some notable matrix factorization methods include Lower-Upper factorization (LU factorization), QR factorization, and Singular Value Decomposition (SVD). In this paper, we discuss the latter of the three methods and how it is used to compress black-and-white (gray-scale) images. Black-and-white images can be compressed without any data loss (lossless compression), since they sometimes contain redundant (or highly correlated) data, however, in practice, this rarely happens [5]. Instead, a more popular type of image compression is used: lossy compression. SVD can be used to implement a lossy compression algorithm.

2 What is Singular Value Decomposition (SVD)?

To understand SVD as a concept, we first need to introduce a couple of important terms: orthogonal (or unitary) matrices and diagonal matrices.

Specifically, for a matrix $A = \begin{bmatrix} a_1 & a_2 & a_3 & \dots & a_n \end{bmatrix}$ with column vectors $a_k(s)$ to be called orthogonal, the inner product of any pair a_i and a_j, where $i \neq j$, has to be zero. In other words, $< a_i, a_j > = 0$, for $0 < i, j < n$ [7]. If A has complex entries, A is called a unitary matrix. Additionally, a diagonal matrix B is a highly useful concept and worth exploring. It is essentially a matrix with real or complex entries on the diagonal and 0 elsewhere.

Now that we are introduced to the terms that make up SVD, let's explore what role they play in the form of SVD. A matrix A of m by n dimensions, noted as m x n for m rows and n columns, can be decomposed into three matrices: U, Σ, and V^T. Each of these three matrices possesses special properties and represents important linear transformations. By Shores' definition, any matrix $A \in R^{m \times n}$ can be written as: $A = U\Sigma V^T$ where:

U $\in R^{m \times m}$ is an orthogonal (or unitary) matrix that acts as a rotational transformation matrix. Column vectors of U are referred to as *left singular vectors*.

$\Sigma \in R^{m \times n}$ is a diagonal matrix that acts as a scaling transformation matrix.

It has non-negative entries on the diagonal and zeros elsewhere.

$V^T \in R^{n \times n}$ is an orthogonal (or unitary) matrix that acts as a rotational transformation matrix. Column vectors of V are referred to as *right singular vectors.*

Singular values (on the diagonal) of Σ are sorted in descending order. We will denote them with $\sigma_1 > \sigma_2 > \cdots > \sigma_r > 0$ where r is the amount of non-zero entries on the diagonal (it also follows, that $r = rank(\Sigma) = rank(A)$)[7].

3 How to compute the SVD of a matrix?

In this section, let us take a simple example of A and find its SVD form.

$$A = \begin{bmatrix} 4 & 0 \\ 3 & -5 \end{bmatrix}$$

We first compute $A^T A$ as:

$$A^T A = \begin{bmatrix} 4 & 3 \\ 0 & -5 \end{bmatrix} \begin{bmatrix} 4 & 0 \\ 3 & -5 \end{bmatrix} = \begin{bmatrix} 25 & -15 \\ -15 & 25 \end{bmatrix}$$

To find the eigenvalues of $A^T A$, we find λ when $det(A^T A - \lambda I) = 0$.

$$\begin{aligned} & det(A^T A - \lambda I) = 0 \\ \Rightarrow & det(\begin{bmatrix} 25 - \lambda & -15 \\ -15 & 25 - \lambda \end{bmatrix}) = 0 \\ \Rightarrow & (25 - \lambda)(25 - \lambda) - (-15)(-15) = 0 \\ \Rightarrow & 400 - 50\lambda + \lambda^2 = 0 \\ \Rightarrow & (\lambda - 10)(\lambda - 40) = 0 \\ \Rightarrow & \lambda_1 = 40, \lambda_2 = 10 \end{aligned}$$

The singular values that form our diagonal Σ matrix are $\sigma_1 = \sqrt{\lambda_1} = \sqrt{40} = 2\sqrt{10}$ and $\sigma_2 = \sqrt{\lambda_2} = \sqrt{10}$. At this point, we have obtained our Σ to be:

$$\Sigma = \begin{bmatrix} 2\sqrt{10} & 0 \\ 0 & \sqrt{10} \end{bmatrix}$$

To find the remaining matrices U and V, we need to find the corresponding eigenvectors from the eigenvalues of $A^T A$.

For $\lambda_1 = 40$:

$$A^T A - \lambda I = \begin{bmatrix} 25 - \lambda & -15 \\ -15 & 25 - \lambda \end{bmatrix} = \begin{bmatrix} -15 & -15 \\ -15 & -15 \end{bmatrix}$$

The eigenvector $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ associated with eigenvalue $\lambda_1 = 40$ belongs to the null space $N(A^T A - \lambda_1 I)$. Hence, solving the equation $\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ yields the desired eigenvector $\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \end{pmatrix}$.

Applying the same technique gives us the second eigenvector corresponding to $\lambda_2 = 10$ to be $\begin{pmatrix} 1 \\ 1 \end{pmatrix}$. Normalizing the two eigenvectors gives us the column vectors of the matrix V:

$$\overrightarrow{v_1} = \frac{1}{\sqrt{2}} \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

and

$$\overrightarrow{v_2} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

In order to construct the last orthogonal matrix U, we find the column vectors $\overrightarrow{u}$(s) by the formula [7]:

$$\overrightarrow{u_1} = \frac{1}{\sigma_1} A \overrightarrow{v_1} = \begin{pmatrix} \frac{-2}{\sqrt{20}} \\ \frac{-4}{\sqrt{20}} \end{pmatrix}.$$

and

$$\overrightarrow{u_2} = \frac{1}{\sigma_2} A \overrightarrow{v_2} = \begin{pmatrix} \frac{4}{\sqrt{20}} \\ \frac{-2}{\sqrt{20}} \end{pmatrix}.$$

Combining all of the above yields the Singular Value Decomposition of A as:

$$A = U \Sigma V^T = \begin{bmatrix} \frac{-2}{\sqrt{20}} & \frac{4}{\sqrt{20}} \\ \frac{-4}{\sqrt{20}} & \frac{-2}{\sqrt{20}} \end{bmatrix} \begin{bmatrix} 2\sqrt{10} & 0 \\ 0 & \sqrt{10} \end{bmatrix} \begin{bmatrix} \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$$

To visualize SVD as a linear transformation, a matrix A can be transformed by a rotation V^T, followed by some scaling Σ, and then another rotation U. Figure 1, 2, and 3 below illustrate a circle after we apply each linear transformation versus after we directly apply A on it. The source code for the visual illustrations below is inspired by Luis Serrano [6].

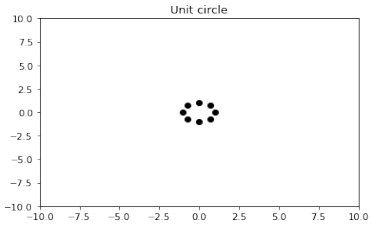
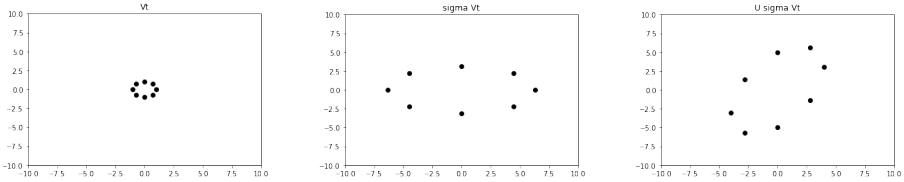


Figure 1: Original circle



(a) After rotation V^T (b) Then a scaling Σ (c) Then a rotation U

Figure 2: After SVD transformation

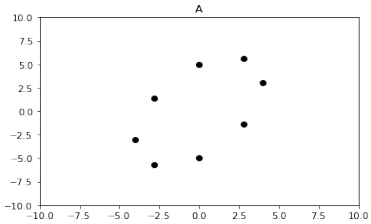


Figure 3: After original transformation A

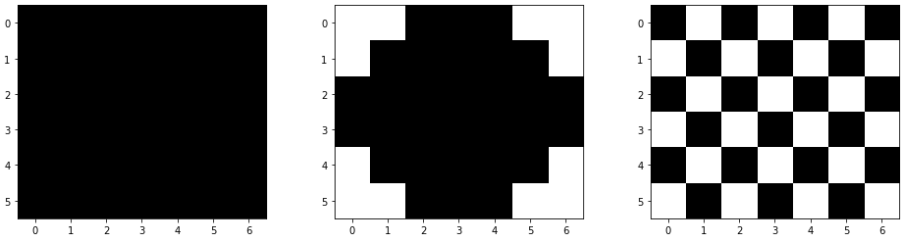
Note that we took A to be a square 2 x 2 matrix for simplicity, but it does not have to be a square matrix. In fact, this is one of the matrix factorization methods that do not require the original matrix to be square. As we saw from the example above, to work out the SVD form of A, we worked mostly with A^TA, which is a square matrix regardless of A being square or not. To confirm this, we notice that if A is an mxn matrix, its transpose A^T will be an nxm matrix; the product A^TA is an mxm matrix.

4 Why do we need to compress images?

Images are usually compressed to reduce the size of an image, which makes it more efficient to transfer and store. Let us imagine a 512 x 512 image, which means the image consists of 512^2 small elements called pixels. These 512^2 pixels

are stored in the image as a 512 x 512 matrix A. Each of these pixels, or each entry of A, takes on a real value ranging from 0 to 255. By this logic, there are 2^8 possibilities for each entry. In our example of an 512 x 512 image, we are looking at 512^2 or 262144 pixels or entries that we have to remember to render the image. A more popular dimension of images that we often see is 1024 x 1024. To store this image uncompressed on disk, our computer has to store 1048576 different values of pixels. Similarly, if a webpage has to download and render an uncompressed image, it can take a long time and use a lot of computational resources. Therefore, for practical reasons, images are almost always compressed for fast web rendering or in order to save disk space.

Another important question we might ask ourselves is: what does it mean for a pixel to take on a certain real value between 0 and 255? In fact, our computer screen cannot "make" any other colors rather than red, blue, and green. Instead, each color is a composite value of these three existing colors: each ranging from 0 to 255; 0 being "non-colored", and 255 being "fully colored" [3]. Considering the scope of this paper, we are looking only at gray-scale images (matrices with entries of 0 and 1) for illustrations. The below images correspond to a all-black (all 0 entries), a black-and-white image (entries of 0s and 1s), and black-and-white checkered image (entries of alternating 0s and 1s).



(a) Matrix with 1s (b) Matrix with 0s and 1s (c) Matrix with 0s and 1s

Figure 4: Grayscale images

5 How to use SVD to compress images?

To find the SVD form of a gray-scale image, let us take the example of image (b) from part four. We have our matrix A as:

$$A = \begin{bmatrix} 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \end{bmatrix}$$

With the help of Python, we find the below to be our matrix U, Σ, and V^T respectively. Note that we have rounded all the matrices' entries to 4 decimal places for simplicity.

$$U = \begin{bmatrix} -0.2777 & 0.5349 & 0.3698 & 0.7001 & 0.0781 & 0.061 \\ -0.4197 & 0.1597 & -0.5462 & -0.0987 & 0.5877 & 0.3806 \\ -0.4967 & -0.434 & 0.2548 & -0.0087 & -0.3854 & 0.5928 \\ -0.4967 & -0.434 & 0.2548 & 0.0087 & 0.3854 & -0.5928 \\ -0.4197 & 0.1597 & -0.5462 & 0.0987 & -0.5877 & -0.3806 \\ -0.2777 & 0.5349 & 0.3698 & -0.7001 & -0.0781 & -0.061 \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} 5.0797 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.7099 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.128 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$V^T = \begin{bmatrix} -0.1956 & -0.3608 & -0.4701 & -0.4701 & -0.4701 & -0.3608 & -0.1956 \\ -0.5076 & -0.3208 & 0.3049 & 0.3049 & 0.3049 & -0.3208 & -0.5076 \\ 0.4518 & -0.5166 & 0.1391 & 0.1391 & 0.1391 & -0.5166 & 0.4518 \\ -0.6569 & 0.2544 & -0.0713 & 0.0357 & 0.0357 & -0.2544 & 0.6569 \\ -0.1639 & -0.2695 & 0.7308 & -0.3654 & -0.3654 & 0.2695 & 0.1639 \\ 0.2042 & 0.6022 & 0.3571 & -0.1786 & -0.1786 & -0.6022 & -0.2042 \\ 0. & 0. & 0. & -0.7071 & 0.7071 & 0. & 0. \end{bmatrix}$$

The main purpose of decomposing a matrix into its SVD form is to understand its properties without having to store all of the information it carries. In this particular example, Figure 5 represents each of the five singular values of Σ: σ_1= 5.0797, σ_2= 1.7099, σ_3= 1.128, σ_4= 0, σ_5= 0, σ_6= 0. Each combination $\sigma_i \overrightarrow{u_i} \overrightarrow{v_i^T}$ has a different weight when it comes to their ability to approximate the matrix.

The first image starting from the left of Figure 5 attempts to approximate our original matrix A as A_1 by incorporating σ_1, $\overrightarrow{u_1}$, and $\overrightarrow{v_1^T}$.

$$A_1 = \sigma_1 u_1 v_1^T = \begin{bmatrix} 0.2759 & 0.509 & 0.6631 & 0.6631 & 0.6631 & 0.509 & 0.2759 \\ 0.4170 & 0.7692 & 1.0022 & 1.0022 & 1.0022 & 0.7692 & 0.4170 \\ 0.4935 & 0.9103 & 1.1861 & 1.1861 & 1.1861 & 0.9103 & 0.4935 \\ 0.4935 & 0.9103 & 1.1861 & 1.1861 & 1.1861 & 0.9103 & 0.4935 \\ 0.4170 & 0.7692 & 1.0022 & 1.0022 & 1.0022 & 0.7692 & 0.4170 \\ 0.2759 & 0.509 & 0.6631 & 0.6631 & 0.6631 & 0.509 & 0.2759 \end{bmatrix}$$

where $\sigma_1 = 5.0797, u_1 = \begin{bmatrix} -0.2777 \\ -0.4197 \\ -0.4967 \\ -0.4967 \\ -0.4197 \\ -0.2777 \end{bmatrix}$, and $v_1 = \begin{bmatrix} -0.1956 \\ -0.3608 \\ -0.4701 \\ -0.4701 \\ -0.4701 \\ -0.3608 \\ -0.1956 \end{bmatrix}$.

As we can see, A_1 is a relatively good approximation of A. The symmetrical cross shape is generally visible. Instead of storing the original matrix of 6 x 7 dimension, only three pieces of information are kept: σ_1 (scalar), u_1 (6 x 1 vector), and v_1 (1 x 7 vector). As the original matrix gets bigger in dimension (to an arbitrarily large m x n), without the help of SVD or other dimension reduction methods, the amount of data we have to store will get quadratically bigger and bigger.

Similarly, the k^{th} sub-figure below is the result of the combination σ_k, $\overrightarrow{u_k}$, and $\overrightarrow{v_k^T}$. It is not difficult to see that as we descend from the first combination (k=1) to the last combination (k=6), our ability to recognize matrix A becomes lower and lower. This is what it means for an image to be compressed with loss: we manage to approximate, or in this case, compress the image, while losing the ability to reconstruct the original image.

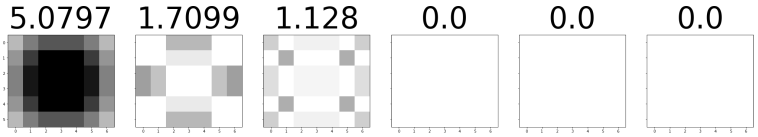


Figure 5: Dimension reduction

Now that we have compared the image quality across different combinations of σ_k, $\overrightarrow{u_k}$, and $\overrightarrow{v_k^T}$, a good question to ask outselves is: How can we get our original matrix A back from its SVD form? By Shores, we know that $A = U\Sigma V^T = \sum_{i=1}^{r} \sigma_i \overrightarrow{u_i} \overrightarrow{v_i^T}$ where r = rank(A) [7]. It follows that if we add all of the combinations that produce their own approximations of matrix A above, we should get our original matrix A back. Figure 6 below shows cumulative approximation of A from adding the first k^{th} combinations of σ_i, $\overrightarrow{u_i}$, $\overrightarrow{v_i^T}$. From adding the three combinations of non-zero singular values, or at r = 3, we already arrive at our original matrix A. This is confirmed after finding the rank of A to be 3.

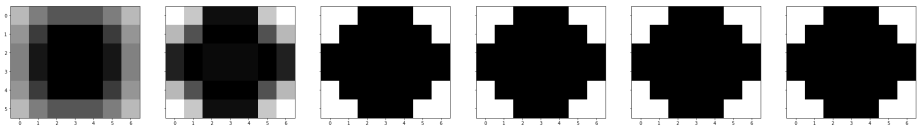


Figure 6: Approximation of a matrix

Note that while this paper applies its code and mathematical techniques to gray-scale images, all of the above could also be applied to colored images to further explore SVD-based image compression.

6 Conclusion

Image compression is one of the many applications of SVD. In practice, the amount of data we are given can be tremendous, so it is necessary to transform the given data in a way that requires the least overhead cost of memory and processing time. Through SVD-based image compression, the image is transformed into a lower-quality image for more efficient transmission and storage.

Despite being highly helpful in dimension reduction, SVD has one shortcoming that prevents it from replacing the most popular image compression method in practice: discrete cosine transform (DCT). DCT is used in the JPEG (Joint Photographic Experts Group) method to compress images. According to Connor Kuhn, we see a faster deterioration of image quality with SVD when data is lost as compared to DCT [2]. In other words, the rate in which an image gets worse in quality per unit of data loss is faster in SVD than in its competitor: DCT.

While SVD might not be the best algorithm for image compression, it does have other applications. For example, SVD can be used in an area of machine learning known as Recommender Systems. In this field, we build algorithms based on user data to predict their future preferences. In this content, we use SVD to factorize matrices of user's data into smaller matrices. The first k^{th} components of such child matrices can provide us with a significant amount of information about the users without us having to store and memorize all of the user data.

References

[1] Horn, Roger A., and Charles R. Johnson. "2. Matrices." *Matrix Analysis*, Cambridge University Press, New York, 2017.

[2] Kuhn, Connor. "SVD and DCT Image Compression." (2016).

[3] Mathews, Brady. "Image Compression using Singular Value Decomposition (SVD)." *Univ. Utah* (2014).

[4] Noble, Ben, and James W. Daniel. "8.4: Application: Singular Value Decomposition." *Applied Linear Algebra*, Pearson Education Taiwan Ltd., Taipei, 2003, pp. 343–344.

[5] Rabbani, Majid, and Paul W. Jones. "Digital Images and Image Compression." *Digital Image Compression Techniques*, SPIE Optical Engineering Press, Bellingham, 1991, pp. 5–7.

[6] Serrano Luis, Singular Value Decomposition, (2021), GitHub repository, https://github.com/luisguiserrano/singular$_v alue_d ecomposition$

[7] Shores, Thomas S. *Applied Linear Algebra and Matrix Analysis*. Springer, 2018.