Singular Value Decomposition: How to Store any Image
Numerical Analysis, Fall 2020

Mario Manalu December 22, 2020

Digital images are made up of the smallest graphical elements capable of storing one color at a time called pixels. The collection of pixels that make up an image are stored as a matrix where each entry contains one pixel. Grayscale images, for example, can be represented by a matrix with each entry containing a number between 0 (black) and 255 (white) inclusively. Color images, in turn, can be represented by three matrices. Each matrix represents the intensity of red, green, and blue in each pixel. Let us assume that our desktop background is an image of size 1280×1024. [1] If it was a grayscale image, our computers would need to store $1280 \times 1024 = 1310720$ pixel values. If it was a color image, our computers would need to store 1310720 red pixel values, 1310720 green pixel values, and 1310720 blue pixel values. That is a total of 3,932,160 different pixel values. If one pixel takes up one byte of memory, then our computers need about 3.93 MB of memory to store that image alone. The size of my entire numerical software package unzipped is only 45 KB, so 3.93 MB is a significant amount of memory. If your computer likes to run out of memory—I know mine certainly does—it would be wise for us to compress that background image before storing it in the memory.

Image compression minimizes the size of an image by decomposing the matrix and eliminating the entries that are relatively small. Since some of the entries are being zeroed-out, the compressed image will be a less-sharp, less-detailed version of the original image. We can tune the quality of the compressed image so that it does not exceed the amount of memory we are willing to allocate. There is a direct relationship between the size of the compressed image and its quality. Lower size will result in lower quality of the compressed image. In most cases, though, we want to compress an image in such a way that our eyes will hardly be able to distinguish the compressed image from the original image.

The idea behind image compression can be explained by the concept of Singular Value Decomposition (SVD). Before we talk about Singular Value Decomposition, let us define what Singular Values mean.

[1] It is very likely to be of that size anyway.

Definition 0.0.1. *Let $A \in \mathbb{R}^{m\times n}$ and let $\lambda_1, \lambda_2, \lambda_3, ..., \lambda_n$ be the real, non-negative eigenvalues of A^TA. The singular values of A are*

$$\sigma_i = \sqrt{\lambda_i}, \quad \textit{for } i = 1, 2, ..., n. \tag{1}$$

Singular values are important because they can be used to determine the most significant rank of a matrix. Once we have identified the most significant ranks, the less significant ranks can ignored or zeroed out. This allow us to reduce the size of a matrix containing a large number of data points into a matrix containing fewer data points that still contains data points that are of great significance. It may not be exactly the case, but you can think of singular values as the colors that are important in an image. For instance, if the image we want to compress is of a person standing in front of a wall that is quite far from the camera, then the appearance of that wall is not significant. Thus, we can blur out the colors of the wall, which is equivalent to "zeroing-out" less significant ranks, and still have a decent image of the person.

Since singular values of a matrix A are crucial for image compression, we would like to place the singular values of A in one separate matrix. This is precisely the objective of Singular Value Decomposition. It decomposes A into three matrices U, V^T, Σ with all singular values of A contained in the diagonal of Σ. Let us formally define what Singular Value Decomposition is.

Definition 0.0.2. *A Singular Value Decomposition of an $m \times n$ matrix A is a decomposition*

$$A = U\Sigma V^T \tag{2}$$

where:

- *U is an $m \times m$ orthogonal matrix,*
- *Σ is an $m \times n$ matrix whose i-th diagonal entry equals the i-th singular value of A. Other entries of Σ are zero,*
- *V^T is an $n \times n$ orthogonal matrix.*

To deepen our understanding of Singular Value Decomposition, let us go through an example.

Example 1. *Suppose that we want to decompose*

$$A = \begin{pmatrix} 2 & 2 & 0 \\ -1 & 1 & 0 \end{pmatrix}. \tag{3}$$

Recall that our objective is to construct U, Σ, V^T with $A = U\Sigma V^T$. Since we care most about singular values, we should start by finding the eigenvalues of A^TA and constructing Σ first. Notice that

$$\begin{aligned} A^TA - \lambda I &= \begin{pmatrix} 2 & -1 \\ 2 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} 2 & 2 & 0 \\ -1 & 1 & 0 \end{pmatrix} - \lambda \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, & (4) \\ &= \begin{pmatrix} 5 & 3 & 0 \\ 3 & 5 & 0 \\ 0 & 0 & 0 \end{pmatrix} - \lambda \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, & (5) \\ &= \begin{pmatrix} 5-\lambda & 3 & 0 \\ 3 & 5-\lambda & 0 \\ 0 & 0 & 0-\lambda \end{pmatrix}. & (6) \end{aligned}$$

Now, we need to compute the determinant of the matrix above. The process of computing the determinant of a 3×3 matrix is slightly tedious. I hope you are convinced that the determinant of the matrix above is given by

$$-\lambda(\lambda - 2)(\lambda - 8). \tag{7}$$

If you trust me so far, then we can conclude that the eigenvalues are $\lambda_1 = 8, \lambda_2 = 2, \lambda_3 = 0$. We want to keep these values in descending order, so that the most significant values appear first in the matrix. The corresponding singular values are

$$\begin{aligned} \sigma_1 &= \sqrt{\lambda_1} = \sqrt{8}, & (8) \\ \sigma_2 &= \sqrt{\lambda_2} = \sqrt{2}, & (9) \\ \sigma_3 &= \sqrt{\lambda_3} = \sqrt{0}. & (10) \end{aligned}$$

Notice that σ_3 does not hold significant information. So, we can ignore σ_3 and have

$$\Sigma = \begin{pmatrix} \sqrt{8} & 0 & 0 \\ 0 & \sqrt{2} & 0 \end{pmatrix}. \tag{11}$$

Next, we want to construct U. We will need to find the nullspace of the matrix corresponding to each of the eigenvectors and then normalize them to construct the columns of U. When $\lambda = 8$,

$$A^T A - 8I = \begin{pmatrix} 5-8 & 3 & 0 \\ 3 & 5-8 & 0 \\ 0 & 0 & 0-8 \end{pmatrix}, \tag{12}$$

$$= \begin{pmatrix} -3 & 3 & 0 \\ 3 & -3 & 0 \\ 0 & 0 & -8 \end{pmatrix}. \tag{13}$$

Consider $\vec{v_1} = \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}$. Note that

$$\begin{pmatrix} -3 & 3 & 0 \\ 3 & -3 & 0 \\ 0 & 0 & -8 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \tag{14}$$

We have shown that $\vec{v_1}$ is the null space of the matrix when $\lambda = 8$. We will compute the first column of U in a minute, but first, let us remind ourselves that we want the column u of U to satisfy

$A\vec{v} = \sigma u$ where $\vec{v}$ is normalized. Thus, the first column of U is given by

$$u_1 = \frac{1}{\sqrt{8}} \begin{pmatrix} 2 & 2 & 0 \\ -1 & 1 & 0 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} 1 \\ 1 \\ 0 \end{pmatrix}, \tag{15}$$

$$= \begin{pmatrix} 1 \\ 0 \end{pmatrix}. \tag{16}$$

Similarly, we will compute u_2. When $\lambda = 2$,

$$A^T A - 2I = \begin{pmatrix} 5-2 & 3 & 0 \\ 3 & 5-2 & 0 \\ 0 & 0 & 0-2 \end{pmatrix}, \tag{17}$$

$$= \begin{pmatrix} 3 & 3 & 0 \\ 3 & 3 & 0 \\ 0 & 0 & -2 \end{pmatrix}. \tag{18}$$

Consider $\vec{v_2} = \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix}$. Note that

$$\begin{pmatrix} 3 & 3 & 0 \\ 3 & 3 & 0 \\ 0 & 0 & -2 \end{pmatrix} \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \tag{19}$$

So, the second column of U is given by

$$u_1 = \frac{1}{\sqrt{2}} \begin{pmatrix} 2 & 2 & 0 \\ -1 & 1 & 0 \end{pmatrix} \frac{1}{\sqrt{2}} \begin{pmatrix} -1 \\ 1 \\ 0 \end{pmatrix}, \tag{20}$$

$$= \begin{pmatrix} 0 \\ 1 \end{pmatrix}. \tag{21}$$

With all the columns of U figured out, we have

$$U = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}. \tag{22}$$

Next, we want to construct V before constructing V^T. The first and second columns of V are $\vec{v_1}$ and $\vec{v_2}$ multiplied by the constant used to normalize them. We have not computed $\vec{v_3}$ earlier because we know we do not need u_3. Since we need $\vec{v_3}$ to fill in the third column of V, we will compute $\vec{v_3}$ now. When $\lambda = 0$,

$$A^T A - 0I = \begin{pmatrix} 5 & 3 & 0 \\ 3 & 5 & 0 \\ 0 & 0 & 0 \end{pmatrix}, \tag{23}$$

$$= \begin{pmatrix} 5 & 3 & 0 \\ 3 & 5 & 0 \\ 0 & 0 & 0 \end{pmatrix}. \tag{24}$$

Consider $\vec{v_3} = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$. *Note that*

$$\begin{pmatrix} 5 & 3 & 0 \\ 3 & 5 & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \tag{25}$$

Now that we have $\vec{v_3}$, *we can construct* V *as follows.*

$$V = \begin{pmatrix} \vec{v_1} & \vec{v_2} & \vec{v_3} \end{pmatrix}, \tag{26}$$

$$= \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{27}$$

Transposing V, *we get*

$$V^T = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{28}$$

By now, we have all the matrices we need to decompose A. *We conclude that*

$$\begin{pmatrix} 2 & 2 & 0 \\ -1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} \sqrt{8} & 0 & 0 \\ 0 & \sqrt{2} & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{29}$$

For any matrix A, if the sequence of singular values are all distinct, then the sequence of singular values and subsequently the sequence of singular vectors are unique. The following theorem proves the previous claim.

Theorem 0.0.1. ***(Singular Value Decomposition)*** *Let* $A \in \mathbb{R}^{m \times n}$, $m > n$, *with* r *nonzero singular values* $\sigma_1 \geq sigma_2 \geq \sigma_3, ..., \geq \sigma_r > 0$. *There exists* $U \in \mathbb{R}^{m \times m}, V \in \mathbb{R}^{n \times n}, \Sigma \in \mathbb{R}^{n \times m}$

such that U and V are orthogonal, Σ contains $\sigma_1, \sigma_2, ..., \sigma_r$ in its diagonal, and

$$A = U\Sigma V^T. \tag{30}$$

There is one lemma that we will be used in the proof of Singular Value Decomposition. We will prove the lemma first and then use the result to complete the proof of Singular Value Decomposition. The lemma is as follows.

Lemma 0.0.1. *The singular values σ_i of $A \in \mathbb{R}^{m\times m}$ satisfy*

$$\sigma_i = ||Av_i||_2 \tag{31}$$

where $A^T Av_i = \lambda_i v_i$ and $||v_i||_2 = 1$.

Proof. By construction, we know that V is orthonormal. So, v_i is orthogonal to itself. Let us compute $(Av_i)^2$.

$$\begin{aligned} Av_i \cdot Av_i &= (Av_i)^T \cdot Av_i, && (32)\\ &= v_i^T A^T Av_i, && (33)\\ & && (34) \end{aligned}$$

Since $A^T Av_i = \lambda_i v_i$,

$$v_i^T A^T Av_i = v_i^T \lambda_i v_i. \tag{35}$$

By Definition 0.0.2, $\lambda_i = \sigma_i^2$. So,

$$\begin{aligned} v_i^T \lambda_i v_i &= v_i^T \sigma_i^2 v_i, && (36)\\ &= \sigma_i^2 (v_i \cdot v_i), && (37)\\ &= \sigma_o^2 v_i^2. && (38) \end{aligned}$$

Since $||v_i||_2 = 1$,

$$\sigma_i^2 v_i^2 = \sigma_i^2(1), \tag{39}$$

$$= \sigma_i^2. \tag{40}$$

Connecting all the dots so far, we have $||Av_i||_2^2 = \sigma_i^2$. Taking the square of both sides, we get

$$||Av_i||_2 = \sigma_i. \tag{41}$$

□

We will proceed to prove Singular Value Decomposition now. This proof is inspired by the Singular Value Decomposition proof in Prof. Blanchard's Class Note on SVD.

Proof. Let $\{(\lambda, v_i)\}_i^r = 1$ be the set of nonzero eigenvalue-eigenvector pairs of $A^T A$ that satisfies $\sigma_i = ||Av_i||_2$. By the previous lemma, we know such pairings exist. Assume that we have removed the singular values that are insignificant. Since we order our singular values in decreasing order of magnitude, we know that at some point, the rest of the singular values are zero. That is, $\sigma_j = 0$ for $j = r+1, ..., n$. Then, for $i = 1, 2, ..., r$, define

$$u_i = \frac{1}{||Av_i||_2} Av_i = \frac{1}{\sigma_i} Av_i. \tag{42}$$

Then, $\{u_1, u_2, ..., u_r\}$ is a set of orthonormal basis of $col(A) \subset \mathbb{R}^m$. Extend $\{u_1, u_2, ...u_r\}$ to an orthonormal basis for $\mathbb{R}^m$ given by $\{u_1, u_2, ..., u_r, ...u_m\}$. Let $\{w_{r+1}, w_{r+2}, ..., w_n\}$ be an orthonormal basis for $null(A)$. Now let

$$U = \Big(u_1|u_2|...|u_m\Big) \in \mathbb{R}^{m\times m} \quad \text{and} \quad V = \Big(v_1|v_2|...|v_r|w_{r+1}|...|w_n\Big) \in \mathbb{R}^{n\times n}. \tag{43}$$

Define $\Sigma \in \mathbb{R}^{m \times n}$ as follows

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & 0 & 0 & 0 & \dots \\ 0 & \sigma_2 & 0 & 0 & 0 & \dots \\ 0 & 0 & \sigma_3 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots \end{pmatrix}. \tag{44}$$

That is, we want to place the singular values of A in the diagonal of Σ and fill in other entries with 0. Then,

$$\begin{aligned} U\Sigma &= \begin{pmatrix} u_1 | u_2 | ... | u_m \end{pmatrix} \begin{pmatrix} \sigma_1 & 0 & 0 & 0 & 0 & \dots \\ 0 & \sigma_2 & 0 & 0 & 0 & \dots \\ 0 & 0 & \sigma_3 & 0 & 0 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \dots \end{pmatrix}, \quad (45) \\ &= \begin{pmatrix} \sigma_1 u_1 | \sigma_2 u_2 | \dots | \sigma_r u_r | 0 | 0 \dots | 0 \end{pmatrix}, \quad (46) \\ &= \begin{pmatrix} Av_1 | Av_2 | Av_3 \dots | Av_r | 0 | 0 \dots | 0 \end{pmatrix}, \quad (47) \\ &= \begin{pmatrix} Av_1 | Av_2 | Av_3 \dots | Av_r | Aw_{r+1} | Aw_{r+2} \dots | Aw_n \end{pmatrix}, \quad (48) \\ &= AV. \quad (49) \end{aligned}$$

Since U and V are orthogonal,

$$\begin{aligned} A &= A(VV^T), \quad (50) \\ &= (AV)V^T, \quad (51) \\ &= U\Sigma V^T. \quad (52) \end{aligned}$$

□

We have talked about and proven the Singular Value Decomposition theorem. We should be comfortable discussing how Singular Value Decomposition allows us to compress the size of an image.

Recall that the motivation behind Singular Value Decomposition is to reduce the size of a matrix by removing singular values that are not significant. By removing some of the values, the dimension of Σ will not be the same as the dimension of A. This phenomenon is called dimensionality reduction. Let us study the following example to see how Singular Value Decomposition reduce the dimension of Σ.[2]

Example 2. *Suppose that we are able to decompose A without reducing the dimensions of U, Σ, V^T. Recall that the dimension of $A, U, \Sigma V^T$ are $m \times n, m \times m, m \times n, n \times n$ respectively. If we pay attention to the dimensions, we will see that*

$$\begin{aligned} dim(A) &= dim(U) \cdot dim(\Sigma) \cdot dim(V^T), && (53)\\ &= (m \times m)(m \times n)(n \times n), && (54)\\ &= (m \times n)(n \times n), && (55)\\ &= (m \times n) && (56) \end{aligned}$$

Now, suppose that we decompose A and use only one singular value. That is, Σ is now a 1×1 matrix. Since there is only one singular value, U will now have only one column, and consequently, V^T will have one row. Even though the dimensions of three matrices are reduced, we will see that the dimension of the product stays the same.

$$\begin{aligned} dim(A) &= dim(U) \cdot dim(\Sigma) \cdot dim(V^T), && (57)\\ &= (m \times 1)(1 \times 1)(1 \times n), && (58)\\ &= (m \times 1)(1 \times n), && (59)\\ &= (m \times n) && (60) \end{aligned}$$

We will now visit the crux of our problem: making the cost for storage cheaper. Say we have an image of size 1000×1000 pixels.[3] If we use 10 singular values for our decomposition, then the

[2]This example can be seen in the svd.py file
[3]Or more accurately pixel squared.

dimensions of our matrices are given by

$$U' = 1000 \times 10, \tag{61}$$

$$\Sigma' = 10 \times 10, \tag{62}$$

$$V^T = 10 \times 1000. \tag{63}$$

After dimensionality reduction, we only need to store $(1000 \times 10) + (10 \times 10) + (10 \times 1000) = 20100$ pixels. Compare that with the number of pixels we need to store without reduction. If we do not reduce the size of Σ, we need to store $1000 \times 1000 = 1000000$ pixels. That is a huge amount of memory saved!

Let us devote some time to talk about the implementation of Singular Value Decomposition in Python. Recall that the algorithm takes in one required input —the matrix A —, and two optional inputs: the number of singular values to determine the dimension of Σ and tolerance. Thus, our function signature is going to look like the following.

Algorithm 1 Singular Value Decomposition

```
Input: The matrix $A$, number of singular values wanted, $tol$. Output: Three matrices $U, \Sigma, V^T$.
Signature: $svd(A, n, tol)$
```

The algorithm, if written in one procedure, is going to be complex and hard to understand. In the spirit of modular programming, we will write a helper function that returns a one-dimensional array used to fill in the columns of U. Let us call it the *svdHelper* function and define it as as shown in the pseudocode under **Algorithm 3**.

Now, in the main *svd* function, we will keep calling *svdHelper* until we finish constructing the matrix U. We will store the three matrices as a tuple data structure containing three elements. We will define *svd* as shown in the pseudocode under **Algorithm 4**. We will run the algorithm against some examples in the Python implementation file.

At the end of the day, an image is just data points that are represented in the form of a matrix. The data points can be fed into the Singular Value Decomposition algorithm to minimize the number of data points that are small and insignificant. The benefit of using SVD is that we get to choose how much memory the compressed image is going to occupy. As a result, we can get rid of irrelevant information and still have the relevant information in our matrix, and this, is how we can store any image of any size.

Algorithm 2 Singular Value Decomposition Helper

Input: The matrix A, tol. **Output:** A one-dimensional array v
Signature: $svdHelper(A, tol)$

```
n,m = $A.shape$;
v = generateUnitVector(min(n,m));
prevVector = None;
currentVector = v;
if $n \geq m$ then
    $M = A^T \times A$
else
    $M = A \times A^T$
end if
while True do
    prevVector = currentVector
    currentVector = $M \times prevVector$
    currentVector = currentVector / norm(currentVector)
    if $abs(currentVector \times prevVector > 1 - tol$ then
        return currentVector
    end if
end while
```

Algorithm 3 Singular Value Decomposition

Input: The matrix A, number of singular values wanted, tol. **Output:** Three matrices U, Σ, V^T.
Signature: $svd(A, n, tol)$

```
n,m = $A.shape$;
currentSVD = EMPTYLIST;
if $n == NONE$ then
    $n = min(n, m)$
end if
for i in range(k) do
    matrixHelper = A.copy()
    for sigma, u, v in currentSVD do
        matrixHelper = matrixHelper - sigma*(outerProduct(u,v))
    end for
    if $n \geq m$ then
        columnU = svdHelper(matrixHelper, tol)
        columnVUnnormalized = dot(A, columnU)
        currentSigma = norm(columnVUnnormalized)
        v = columnVUnnormalized / currentSigma
    else
        columnV = svdHelper(matrixHelper, tol)
        columnUUnnormalized = dot($A^T$, columnV)
        currentSigma = norm(columnUUnnormalized)
        u = columnUUnnormalized / currentSigma
    end if
    currentSVD.append((currentSigma, u, v))
end for
allSigmas = store all sigmas in one one-dimensional array.
U = store all columns of $U$ in one two-dimensional array.
V = store all columns of $V$ in one two-dimensional array.
Sigma = zeros((len(singularValues), (len(singularValues))))
Sigma[:A.shape[1], :A.shape[1]] = diag(singularValues)
return U, Sigma, V
```