Singular Value Decomposition (SVD)

Jabed Umar (2011072), P-346

School of Physical Sciences, National Institute of Science Education and Research, HBNI, Jatni-752050, India

Oct-Nov, 2022

1 Introduction

Singular Value Decomposition is one of the most important concepts in linear algebra. It has a wide range of applications in science, engineering, and mathematics such as computing the pseudo inverse, Rank, Range, and Null space of a given matrix, data compression, Principal Components Analysis (PCA), low-rank approximation (LRA), image processing, Curve Fitting Problem etc. In linear algebra, a matrix's Singular Value Decomposition (SVD) is a factorization of that matrix into three matrices. Thus, the singular value decomposition of matrix A can be expressed in terms of the factorization of A into the product of three matrices as $\mathbf{A} = UDV^T$. It has some interesting algebraic properties and conveys important geometrical and theoretical insights about linear transformations. In this project, we shall discuss how to use SVD to solve a billion-dollar problem [1] image compression with no or very small loss in quality.

2 Singular Value Decomposition of a Matrix

2.1 Mathematical Motivation

Before discussing the SVD case, let's recall the Eigendecomposition[1] of any matrix A. By using the Eigendecomposition we can easily diagonalize any matrix A. But what if the given matrix is not square[1] (n×n) or we don't get n linearly independent eigenvector for (n×n) matrix or eigenvector are not orthogonal to each other[2]? Then we can't diagonalize the matrix A. SVD can help solve this issue.

2.2 SVD

Consider any (m×n) real matrix of rank r. Then A can be uniquely decomposed as A $= UDV^T$. Here, U's are eigenvectors of AA^T and the V's are eigenvectors of A^TA and D is the diagonal matrix. Since those matrices are both symmetric (AA^T and A^TA are always symmetric for all A), their eigenvectors can be chosen orthonormal. Let's find U, V, and D quickly.

$$A = UDV^T \implies AA^T = UDV^TVDTU^T$$

$$\implies AA^T = UD^2U^T \ (V^TV = I, D = D^T) \tag{1}$$

$\implies$ U is eigenvector of AA^T matrix.

Similarly,

$$A = UDV^T \implies A^TA = VD^TU^TUDV^T$$

$$\implies A^TA = VD^2V^T \ (U^TU = I, D = D^T) \tag{2}$$

$\implies$ V is eigenvector of A^TA matrix.

From Eqn-(1) and Eqn-(2) it's clear that D^2 matrix contains eigenvalues of the AA^T (or,A^TA) matrix. So, we got that to do SVD of any matrix A we need orthogonal eigenvectors U, V, and singular values σ_i's (square roots of eigenvalues of A^TA matrix). Since σ_i's are the square root of real symmetric matrices so they must be real and must have been positive. This implies eigenvalues of A^TA matrix must have to be non-negative. So, A^TA is a positive semi-definite matrix[3][2].

Therefore, SVD of any m×n matrix A with rank r is,

$$Av_i = \sigma_i u_i \quad ; \ i \le r \tag{3}$$

Where $\sigma_1 \ge .... \ge \sigma_r > 0$

2.3 Examples of SVD

2.3.1 A is square matrix (n×n)

In this case,

$$A_{n\times n} = U_{n\times n} D_{n\times n} V_{n\times n} \tag{4}$$

Let A $= \begin{pmatrix} 2 & 2 \\ -1 & 1 \end{pmatrix}$. Since rows(or columns) of A are linearly independent so the rank is 2. So, $A^TA = \begin{pmatrix} 5 & 3 \\ 3 & 5 \end{pmatrix}$. Clearly, A^TA is a symmetric matrix. The eigenvalues of A^TA is 8 and 2 ($\equiv$ A^TA is positive semidefinite). So, the matrix contains the singular values (σ_i's will be $\sqrt{8}$ and $\sqrt{2}$) will be D = diag($\sqrt{8}$, $\sqrt{2}$). Now the orthogonal eigenvectors(v_i's) of A^TA are $\begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$ and $\begin{pmatrix} \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$. So, V $= \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$.

Let's compute u_i's. From Eqn-3, $Av_1 = \begin{pmatrix} 2\sqrt{2} \\ 0 \end{pmatrix} = 2\sqrt{2} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 2\sqrt{2}$ u_1 and $Av_2 = \begin{pmatrix} 0 \\ \sqrt{2} \end{pmatrix} = \sqrt{2} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \sqrt{2}$ u_2. So, U $= \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$. Therefore, A $= \begin{pmatrix} 2 & 2 \\ -1 & 1 \end{pmatrix} = UDV^T = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 2\sqrt{2} & 0 \\ 0 & \sqrt{2} \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{-1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$.

Here, we have found out U from V. But we can find out U directly by using eqn-2. Now the question is what if A $= \begin{pmatrix} 2 & 2 \\ 1 & 1 \end{pmatrix}$ or the the rank of A is 1 (<2) ?

In this case, we shall get one eigenvalue to be 0 and another one is 10. So, this is perfectly fine because AA^T(or, A^TA) is symmetric and positive semidefinite.Again similarly we can find V $= \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{pmatrix}$ and U $= \begin{pmatrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} & \frac{-2}{\sqrt{5}} \end{pmatrix}$. So, A $= \begin{pmatrix} \frac{2}{\sqrt{5}} & \frac{1}{\sqrt{5}} \\ \frac{1}{\sqrt{5}} & \frac{-2}{\sqrt{5}} \end{pmatrix} \begin{pmatrix} \sqrt{10} & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{-1}{\sqrt{2}} \end{pmatrix}$

2.3.2 A is rectangular matrix (m×n)

Case I [m > n] : In this case,

$$A_{m\times n} = U_{m\times m} D_{m\times n} V_{n\times n} \tag{5}$$

Suppose, A is a 5×3 matrix so A^TA will be a symmetric and positive semi-definite 3×3 matrix. So, we shall get 3 eigenvalues ($\equiv$ singular values). But, the D matrix has to be 5×3 to make matrix multiplication compatible. Now, what should we do?

To solve this issue we shall add two zero rows to the D matrix. Therefore, D will be in the form,

[1] Eigen value equation (A u = λu) is defined for only square matrix

[2] This time we can diagonalize the matrix but we need to orthogonalize eigenvectors by Gram-Schmidt orthogonalization

[3] A positive semi-definite matrix is a symmetric matrix where every eigenvalue is non-negative

$$D = \begin{pmatrix} \sigma_{11} & 0 & 0 \\ 0 & \sigma_{22} & 0 \\ 0 & 0 & \sigma_{33} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

Case II [m $<$ n] : In this case,

$$A_{m\times n} = U_{m\times m} D_{m\times n} V_{n\times n} \tag{6}$$

Suppose, A is a 3×5 matrix so AA^T will be a symmetric and positive semi-definite 3×3 matrix. So, we shall get 3 eigenvalues ($\equiv$ singular values). But, the D matrix has to be 3×5 to make matrix multiplication compatible. Now, what should we do?

To solve this issue we shall add two zero columns to the D matrix. Therefore, D will be in the form,

$$D = \begin{pmatrix} \sigma_{11} & 0 & 0 & 0 & 0 \\ 0 & \sigma_{22} & 0 & 0 & 0 \\ 0 & 0 & \sigma_{33} & 0 & 0 \end{pmatrix}$$

So, for the rectangular case, it's clear that the dimension of D must be the same as matrix A and even we are allowed to add zero rows(or, columns) to matrix D to make matrix multiplication compatible. Also, U and V are square matrices because they are orthogonal matrices.

3 Algorithm/Pseudocode

So far, we have discussed so many things about SVD. Let's talk about the python implementation of SVD. The algorithm steps are following :

1. Take input of any m×n matrix.
2. if m >= n :
   (a) Multiply A^T with A.
   (b) Find the eigenvalues (λ_i's) of A^TA.
   (c) if $\sqrt{\lambda_i}(=\ \sigma_i)$ >= 0 : return A^TA is positive semi-definite.
      i. for i in range(n) : D = diag(σ_i) .
         if m>n: add zero rows as long as the no of rows of D == m.
      ii. Find eigenvectors of A^TA and store them in an array V.
      iii. Find eigenvectors of AA^T and store them in an array U.
      iv. , Therefore, SVD of A = UDV^T.
   (d) else: return A is not positive semi-definite so SVD of A is not possible.
3. else :
   (a) Multiply A with A^T.
   (b) Find the eigenvalues (λ_i's) of AA^T.
   (c) if $\sqrt{\lambda_i}(=\ \sigma_i)$ >= 0 : return AA^T is positive semi-definite.
      i. for i in range(m) : D = diag(σ_i) .
         Add zero columns as long as the no of columns of D == n.
      ii. Follow the same steps(ii - iv) to find U, V, and SVD of A.
   (d) else: return A is not positive semi-definite so SVD of A is not possible.

4 python Implementation

4.1 SVD of a matrix

From the algorithm section, it's clear that many steps are involved in doing the SVD of a matrix. And also whatever we have learned from the course we are not able to find eigenvalues for huge matrices. So, we shall use python in the build library NumPy to find the SVD of a matrix A. Numpy has a function **np.linalg.svd()** [see fig-1]. This function returns the 3 matrices: U, D, and V^T. In the D matrix, we get the singular values in descending order [see fig-1]. The function returns the matrix D not in the form of a diagonal matrix but as a vector of singular values. So, we will have to convert it into a diagonal matrix (by **numpy.diag()** function) before we can use it to reconstruct matrix A.

```
# importing python in built library numpy to do SVD
import numpy as np
# define a matrix(2*2) A
A = [[2,2],
     [-1,1]]
# SVD of matrix A into U, D, V^T
u , s , v = np.linalg.svd(A)
# print the decomposed matrix UDV^T
u,s,v
```

```
(array([[-1.00000000e+00,  1.11022302e-16],
        [ 1.11022302e-16,  1.00000000e+00]]),
 array([2.82842712, 1.41421356]),
 array([[-0.70710678, -0.70710678],
        [-0.70710678,  0.70710678]]))
```

Figure 1: **SVD of a matrix A = UDV^T**

This is the same as whatever we have done analytically in section 2.3.1 except for a bit of computational error.

4.2 Sigma(diagonal matrix of A) matrix

In the mathematical motiovation section I have talked the necessary of SVD to diagonalied any matrix A ($m\times n$). Now we shall see the python implementaion of this. To do computationally first we use **SVD** function then extract the sigular value matrix. Convert this to 2d array by **np.diag()** and then add suitable amount of zeroes to row or column by **sigma_matrix()** function. Let's see two different case of sigma matrix [see fig-2 & 3].

```
# define the matrix
X = [[1,2,3,],[2,3,4],[3,4,5],[4,5,6],[5,6,7]]
# calling the SVD function from library
x,y,z = d.SVD(X)
# print("The shape of the decomposed matrices is",[np.shape(x) for x in [x, np.diag(y), z]])
print("The singular values are \n",y)
```

```
The singular values are
 [1.67010311e+01 1.03709214e+00 3.62597321e-16]
```

```
# contruct sigma matrix()
print("The sigular matrix of the given matrix is:\n",d.sigma_matrix(X))
```

```
The sigular matrix of the given matrix is:
 [[1.67010311e+01 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 1.03709214e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 3.62597321e-16]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00]
 [0.00000000e+00 0.00000000e+00 0.00000000e+00]]
```

Figure 2: **Sigma matrix of A with m>n**

```
# define the matrix
A = [[3,2,2,6],
     [2,3,-2,4]]
# calling the Svd function from library
m,n,o = d.SVD(A)
# print("The shape of the decomposed matrices is",[np.shape(x) for x in [m, np.diag(n), o]])
print("The singular values are \n",n)
```

```
The singular values are
 [8.74792028 3.07796861]
```

```
# contruct sigma matrix()
print("The sigular matrix of the given matrix is:\n",d.sigma_matrix(A))
```

```
The sigular matrix of the given matrix is:
 [[8.74792028 0.         0.         0.        ]
 [0.         3.07796861 0.         0.        ]]
```

Figure 3: **Sigma matrix of A with m<n**

So far we have discussed how to do SVD of any matrix and see an example of how to implement that in python. Let's now stop this here and move to one significant application of SVD.

5 Low-rank approximation and Image Compression

5.1 Motivation

We often need to transmit and store the images in many applications. Smaller the image, less is the cost associated with transmission and storage. So we often need to apply data compression techniques to reduce the storage space consumed by the image. A digital image is a matrix of pixel values. Each little picture element or "pixel" has a grayscale number between black and white (it has three numbers for a color picture). The picture might have $512 = 2^9$ pixels in each row and $256 = 2^8$ pixels down each column. We have a 256-by-512-pixel matrix with 2^{17} entries! To store one picture, the computer has no problem. But a CT or MR scan produces an image at every cross section-a ton of data. If the pictures are frames in a movie, 40 frames a second means 144,000 images per hour. Compression is especially needed for high-definition digital TV, or the equipment could not keep up in real-time. Moreover, larger images will be read slower from the disk, and any operations done on it will be slower too. So, what will be the solution to this kind of image-storing problem?

If we could somehow replace those 2^{17} matrix entries with a smaller number, without losing picture quality. Then we can solve this issue.

5.2 What is Low-rank approximation?

So, image compression is essential to solve the storage and computational time issue. But to do image compression we need something that will replace bigger matrices with smaller matrices. To do this we shall use the concept of Low-rank approximation(LRA). Let $A \epsilon R^{m\times n}$ then the rank of A is defined as the dimension of range space of A. A matrix of rank r admits a factorization of the form, $A_{m\times n} = B_{m\times r}C^T_{r\times n}$. Clearly, the no of elements in A and BC^T is mn and $(mr+rn)$. Suppose, A is 50×100 matrix and r(A) = 20 then no of elements in A is 5000 and the no of elements in BC^T is 3000. Therefore, we are decreasing the computational cost. This is the main motivation or idea behind the LRA. And this can helps us to replace bigger matrices with smaller matrices.

5.3 How SVD helps ?

We have seen that a matrix of rank r can be decomposed into two matrices B and C. But we won't always be fortunate enough to get a matrix that can be expressed as a product of a B and C matrix and sometimes it's quite hard to find the B and C matrix. But to compress an image we need LRA. Now, we shall see how SVD provides a very simple solution to LRA. Suppose, A is a 5×5 matrix and the singular values of A^TA are $\sigma_1 = 3$, $\sigma_2 = 1$, $\sigma_3 = 0.5$, $\sigma_4 = 0.02$, $\sigma_5 = 0.05$ i.e r(A) = 5 and $A_{5\times5} = U_{5\times5}D_{5\times5}{V^T}_{5\times5}$. Now, we can see that the last two singular values are very small as compared to the first three values. So, we can remove this from the matrix D and make it 3×3. Again, to make the matrix multiplication compatible we remove the two columns(eigenvectors)of U and V related to the smaller eigenvalues[4] i.e r(A) = 3 and $A = U_{5\times3}D_{3\times3}V_{3\times5}$. Notice that A is still 5×5 but r(A) is 3 so it consumes very less storage to store almost the same data as compared to the actual A matrix.

So, SVD helps us to find an equivalent matrix A_k of A [where r(A) >> r(A_k)] which stores almost the same data as A stores within very less storage.

5.4 Error in LRA

Suppose A is m×n matrix with rank r and by using the SVD we convert it m×n matrix A_k with rank k (where k<< r). The measure of the quality of the approximation is given by :

$$\frac{||A_k||^2}{||A||^2} = \frac{\sigma_1^2 + \sigma_2^2 + ... + \sigma_k^2}{\sigma_1^2 + \sigma_2^2 + \sigma_3^2 + ... + \sigma_r^2}$$

Where |||| denotes the norm of the matrix. Therefore, the relative error in this approximation is

$$\frac{||A||^2 - ||A_k||^2}{||A||^2} = \frac{\sigma_{k+1}^2 + \sigma_{k+2}^2 + \sigma_{k+3}^2 + ... + \sigma_r^2}{\sigma_1^2 + \sigma_2^2 + \sigma_3^2 + ... + \sigma_r^2} \tag{7}$$

This implies removing smaller singular values leads to small errors and the approximate matrix A_k will be close to the actual matrix A.

5.5 Examples of Image compression

5.5.1 Grayscale Images

We will first try our hands upon grayscale images. Grayscale images are images in which every pixel has a single brightness value. So it is easier to work with. To compress a gray scale image first we need to comvert a RGB(red-green-blue) image greyscale image and then to do the compression **compress_grey( )** function has been used. Let's see how it looks on a greyscale image.

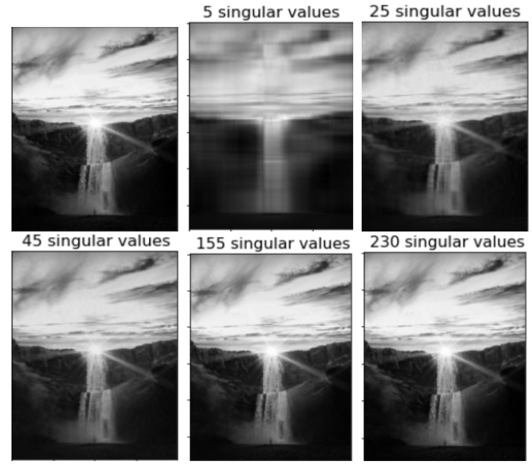

Figure 4: **GreyScale Image Compression**

Here, 1st one is the actual image. By using the **compress_grey( )** function compression has been done with different different sigular values. It's clear that 25-45 rank approximation is enough to compress this image without loosing much data. This can also be seen from the variance graph [see variance graph of singular value in **DIY.ipynb** file] of the singular values. 1st three singular values contribution to the image is almost 85%.

5.5.2 Color Images

Colour images consist of 3 channels of data: Red, Green and Blue. So, for colour images we need to perform SVD on each channel separately and then store all three channel into a single stack to return the compressed colour image. The **compress_color_a( )** and **compress_color_b( )** function does exactly that. In the later function I have used my own matrix multiplication [**matrix_product( )**] function to do low rank approximation but it takes so long time (approximately 5-7 s) so I used first function to do the compression where I have used the inbuilt matrix multiplication function of **Numpy**. Let's see how it looks on a color image.

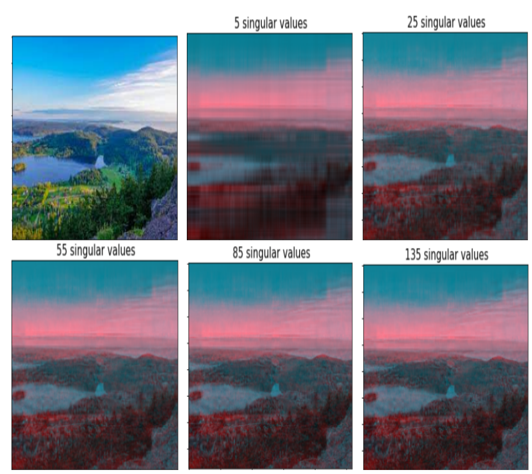

Figure 5: **GreyScale Image Compression**

[4]To minimize the representation error, we are choosing the smallest eigenvalues

Here, 1st one is the actual image. By using the **compress_color_a()** function compression has been done with different different sigular values. It's clear that 25-55 rank approximation is enough to compress this image without loosing much data. It's clear that we got more reddish color image or R channel is more present here w.r.t G and B. This is because the singular values of red is contributing more when we use low rank approximation [see variance graph of singular values for different channel in **DIY.ipynb** file]. Since, we have used **Numpy** inbuilt function to calculate eigenvalues and eigenvector of the matrix so we loss little data becuase of approximated algorithm. That's why our output is bit far from the actual image.

6 Conclusion

In this project we mainly focus on the SVD and it's application to image compression. First SVD of any matrix is done by using the **Numpy** and then we apply it to the project. I have shown two different cases of image compression. I have also shown the diagonal matrix of A [see on **DIY.ipynb** file] from where I started talking about the SVD.
In the image compression part, digital image is given to SVD. SVD refactors the given digital image into three matrices. Singular values are used to refactor the image and at the end of this process, image is represented with smaller set of values, hence reducing the storage space required by the image. We have seen how effectively and how quickly SVD compress and image. Therefore, using (SVD) for image compression can be a very useful tool to save storage space. We were able to get an image that is indistinguishable from the original image, but only using very less % of the original storage space.

7 Acknowledgement

First and foremost, I must express my sincere gratitude to the course (P-346) instructor prof. Subhasis Basak for introducing me to the beautiful project. I also acknowledge the role of the books by prof. Gilbert Strang which was my primary source for learning SVD and image compression throughout the project. Then I would like to acknowledge the book "Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control" from where I have learn different computational technique for this project. Finally, I would like to thank the IIT Roorkee for the wonderful lecture series on "essential mathematics for machine learning" from where I have learned so many important concepts of SVD and the LATEX for providing the tools with which this project has been typeset.

8 References

1. Introduction to Linear Algebra, 4th Edition by prof. Gilbert Strang

2. https://www.math.purdue.edu/eremenko/dvi/lect4.9.pdf

3. Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control 1st Edition