Singular Value Decomposition: Compression of Color Images

Bethany Adams and Nina Magnoni

Introduction

The SVD has very useful applications. It can be used in least squares approximations, search engines, and image compression. In this presentation we will show how it is applied, more specifically, to the compression of color images. We begin by decomposing a 2x2 matrix, then we'll show how matlab is useful for SVDing larger matrices, and lastly applying the matlab SVD techniques to images, grayscale and truecolor.

Definition

Singular Value Decomposition factorizes A, a rectangular matrix, into the matrices U, S, and V^T. The SVD factorization doesn't require square matrices, therefore m, the number of columns, does not have to equal n, the number of rows.

$$A_{mxn} = U_{mxm} S_{mxn} V^T_{nxn}$$

In this U and V are orthogonal matrices; therefore, all of the columns of U and V are orthogonal to one another. The S matrix, is a diagonal matrix whose entries are in descending order, $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_n \geq 0$, along the main diagonal.

$$\begin{bmatrix} \sigma_1 & 0...0 \\ 0 & \sigma_2...0 \\ 0 & 0...\sigma_n \end{bmatrix} \quad (1)$$

In order to compute the matrices U, S, and V there are several hand computations that must be completed. These include finding some special eigenvalue/eigenvector pairs.

The eigenvalues come from the relationship

$$Ax = \lambda x$$

or

$$(A - \lambda I)x = 0.$$

This shows that the eigenvectors are in the nullspace of $A - \lambda I$, therefore the determinant,

$$|A - \lambda I| = 0.$$

, must equal zero. The eigenvalues surface when the found eigenvalues are substituted into the above equation($(A - \lambda I)x = 0$) and the nullspace is found.

An Example for a 2×2 Matrix

Let $A = \begin{bmatrix} 2 & 2 \\ -1 & 1 \end{bmatrix}$. In order to complete the decomposition, we must first compute the eigenvalue eigenvector pairs for each AA^T and A^TA.

$$AA^T = \begin{bmatrix} 2 & 2 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & -1 \\ 2 & 1 \end{bmatrix} = \begin{bmatrix} 8 & 0 \\ 0 & 2 \end{bmatrix} \tag{2}$$

$$\left|AA^T - \lambda\right| = \left| \begin{bmatrix} 8 & 0 \\ 0 & 2 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right| = 0 \tag{3}$$

$$A = \begin{vmatrix} 8-\lambda & 0 \\ 0 & 2-\lambda \end{vmatrix} = 0 \tag{4}$$

To find λ,

$$(8-\lambda)(2-\lambda) - 0 = 0$$

$$(8-\lambda)(2-\lambda) = 0$$

$$\lambda_1 = 8$$

$$\lambda_2 = 2$$

When finding the eigenvalues of $A^T A$ we use the following computation:

$$A^T A = \begin{bmatrix} 2 & -1 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 2 & 2 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} 5 & 3 \\ 3 & 5 \end{bmatrix} \tag{5}$$

$$\left| A^T A - \lambda \right| = \left| \begin{bmatrix} 5 & 3 \\ 3 & 5 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right| = 0 \tag{6}$$

$$A = \begin{vmatrix} 5-\lambda & 3 \\ 3 & 5-\lambda \end{vmatrix} = 0 \tag{7}$$

To find λ,

$$(5-\lambda)(5-\lambda) - (3)(3) = 0$$

$$\lambda^2 - 10\lambda + 16 = 0$$

$$\lambda_1 = 8$$

$$\lambda_2 = 2$$

Finding The Eigenvectors

The corresponding eigenvectors to these eigenvalues are in the nullspace of $(A - \lambda I)$ or in our case the nullspace of $(AA^T - \lambda I)$ or $A^T A - \lambda I)$. Let's begin with the eigenvectors of AA^T which will eventually make up the columns of the matrix U.

$$
\begin{aligned}
(AA^T - \lambda I)\mathbf{x} &= 0 \\
\left(\begin{bmatrix} 8 & 0 \\ 0 & 2 \end{bmatrix} - \lambda_1 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \mathbf{x}_1 &= 0 \\
\begin{bmatrix} 8-8 & 0 \\ 0 & 2-8 \end{bmatrix} \mathbf{x}_1 &= 0 \\
\begin{bmatrix} 0 & 0 \\ 0 & -6 \end{bmatrix} \mathbf{x}_1 &= 0 \\
\mathbf{x}_1 &= \begin{bmatrix} -1 \\ 0 \end{bmatrix}
\end{aligned}
\tag{8}
$$

$$
\begin{aligned}
(AA^T - \lambda I)\mathbf{x} &= 0 \\
\left(\begin{bmatrix} 8 & 0 \\ 0 & 2 \end{bmatrix} - \lambda_2 \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right) \mathbf{x}_2 &= 0 \\
\begin{bmatrix} 8-2 & 0 \\ 0 & 2-2 \end{bmatrix} \mathbf{x}_2 &= 0 \\
\begin{bmatrix} 6 & 0 \\ 0 & 0 \end{bmatrix} \mathbf{x}_2 &= 0 \\
\mathbf{x}_2 &= \begin{bmatrix} 0 \\ 1 \end{bmatrix}
\end{aligned}
\tag{9}
$$

A similar computation can be used for determining the eigenvectors of the matrix $A^T A$.

$$
\begin{aligned}
\mathbf{x}_1 &= \begin{bmatrix} -1 \\ -1 \end{bmatrix} \\
\mathbf{x}_2 &= \begin{bmatrix} -1 & 1 \end{bmatrix}
\end{aligned}
$$

Finding the full Decomposition

The matrices in the singular value decomposition require that the columns are of length one. Therefore, we must divide each eigenvector by its magnitude. Also, the singular values are not equivalent to the eigenvalues, we must take the square root of each eigenvalue in order to find a singular value. The decompression then becomes

$$\begin{aligned} A &= USV^T \\ &= \begin{bmatrix} -1/\sqrt{1} & 0 \\ 0 & 1/\sqrt{1} \end{bmatrix} \begin{bmatrix} \sqrt{8} & 0 \\ 0 & \sqrt{2} \end{bmatrix} \begin{bmatrix} -1/\sqrt{2} & -1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix}^T \\ &= \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{8} & 0 \\ 0 & \sqrt{2} \end{bmatrix} \begin{bmatrix} -1/\sqrt{2} & -1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{bmatrix} \end{aligned}$$

If you would like, it may be helpful to multiply these matrices together to see that they truly do equal our original matrix $A = \begin{bmatrix} 2 & 2 \\ -1 & 1 \end{bmatrix}$

Decomposition of Larger Matrices In Matlab

As the matrices get larger and larger, it becomes too challenging to do all these computations via brain, so Matlab comes in handy for decomposing the larger, more heinous, matrices.

Doing Singular Value Decomposition using Matlab is quite simple. We need not compute eigenvalues, eigenvectors, unit eigenvectors, or singular values for our matrix A because Matlab does this for us. Notice the SVD of our sample matrix A

```
A=[2 2;-1 1]

A =

     2     2
    -1     1
```

```
U =
   -1.00000000000000    0.00000000000000
    0.00000000000000    1.00000000000000

S =
    2.82842712474619                    0
                   0    1.41421356237310

V =
   -0.70710678118655   -0.70710678118655
   -0.70710678118655    0.70710678118655
```

The singular values or the square roots of our eigenvalues, are arranged in descending order down the diagonal of S, and the corresponding unit eigenvectors are in the correct positions of our matrices U and V. The typeset is equally as simple for matrices thousands of times this large. It is here that the SVD of image matrices is possible.

Quick Description of True Color Versus Gray Scale

With gray scale images we have values for each pixel between 0 and 1 which represent the intensity of light emitted from the pixel. For the Color images we have the intensities for each color and there are three layers, each layer either being red, green, or blue in the RGB spectrum. To SVD the color image, we must strip away and SVD each color layer, and then recombine the decomposed layers to produce the whole decomposed true color image.

Decomposition of a Gray Scale Image

Singular Value Decomposition for gray scale images in Matlab is a simple task. Because grayscale images are represented by matrices containing only numbers between one and zero, the SVD is a very popular and easy method for their compression.

```
close all
imfinfo('puppy.jpg')
[A,map]=imread('puppy.jpg');

B=im2double(A);
imshow(B)
[u,s,v]=svd(B);
C=zeros(size(B));
for
k=5:5:35
    for j=1:k
        C=C+s(j,j)*u(:,j)*v(:,j).';
```


```
    end
end
figure
imshow(C,[])
drawnow
set(gcf,'Unit','inches','Paperposition',[0,0,2,2.7])
filename=['puppy',num2str(k),'.jpg'];
print('-djpeg90',filename)
```

Decomposition of a True Color Image

When decomposing a grayscale image, only one step is needed. Because the grayscale image is only one layer thick it is much easier to manage. Color images, however, are much more common, and are also much more difficult to decompose. Because truecolor images are composed of a red, green, and blue layer for each pixel, we cannot simply compute the SVD of the original matrix. In order to get around this, we have written a code in Matlab that peels off the layers of the image and stores them separately in different matrices. In this fashion the matrices can be decomposed individually and then later stacked back on top of one another to create a new SVDed truecolor image.

Matlab Code

```
close all iptsetpref('ImshowBorder','tight') clear Q
L=imread('buildings.jpg');
imshow(L)

L1=L(:,:,1);
L2=L(:,:,2);
L3=L(:,:,3);

I1=im2double(L1); I2=im2double(L2); I3=im2double(L3);
figure
imshow(I1)
figure
imshow(I2)
figure
imshow(I3)

[u1,s1,v1]=svd(I1);
```







```
[u2,s2,v2]=svd(I2);
[u3,s3,v3]=svd(I3);

C1=zeros(size(I1));
C2=zeros(size(I2));
C3=zeros(size(I3));

k=15;
for j=1:k
    C1=C1+s1(j,j)*u1(:,j)*v1(:,j).';
end
for j=1:k
    C2=C2+s2(j,j)*u2(:,j)*v2(:,j).';
end
for j=1:k
    C3=C3+s3(j,j)*u3(:,j)*v3(:,j).';
end
figure
imshow(C1)
```

```
figure
imshow(C2)
figure
imshow(C3)

C1(k)=1;
C2(k)=1;
C3(k)=1;

R1=im2uint8(C1);
R2=im2uint8(C2);
R3=im2uint8(C3);

Q(:,:,1)=R1;
Q(:,:,2)=R2;
Q(:,:,3)=R3;
figure
imshow(Q,[])
```

The resulting images

Original Image

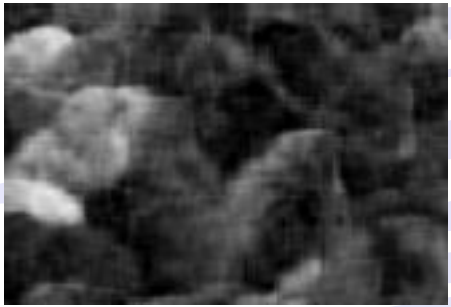
Red Layer

Green Layer

Blue Layer

SVD Red Layer

SVD Green Layer

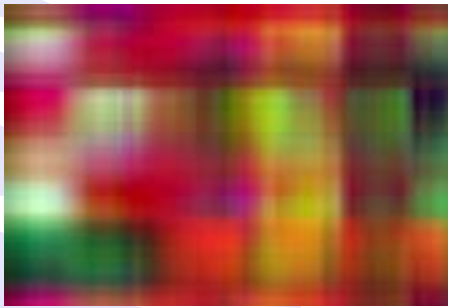


SVD Blue Layer

Recompiled Image

The previous images illustrate the individual steps that the Matlab routine executes during the SVD code. The first four images have not been decomposed and retain their original crispness, while the final four images are the decomposed red, green, and blue layers of the original image and the recompiled layers after the final iteration. These decomposed images have rank 15.

$k = 1$



$k = 2$

$k = 5$

$k = 7$

$k = 10$

$k = 12$

$k = 15$

$k = 20$

$k = 30$

$k = 50$

$k = 75$

$k = 100$

Notice that the clarity of the image improves as the k values of the previous Matlab code are increased. Eventually, however, the image is so clear that it is almost indistinguishable from the original image.

Original Image

Rank 100

See here that at rank 100 the produced image is almost identical to the original image. It is notable that the original image has rank 300, meaning that in order to actually reproduce this image, we would need 300 iterations. By using only the first 100 singular values, we can cut off an amazing amount of storage space.

In conclusion

In conclusion, the singular Value Decomposition is a very useful tool for image compression. We can take an image which originally is rank 300 and store it in a reasonably good representation matrix that has only rank 100. Sometimes, we can even cut off more of the original image without losing clarity. In a time when computer imaging and web browsing demand the use of many different images, it is very important for us to be able to store all the image types in reasonably sized matrices. The Singular Value Decomposition makes this possible.