Singular Value Decomposition

By Maeve and Alex

Learning Plan

- Singular Value Decomposition Basics
    - SVD on an image
    - Data
    - Why use it?
- SVD Math
- SVD Implementation
    - MatLab
    - Python
    - Wolframalpha

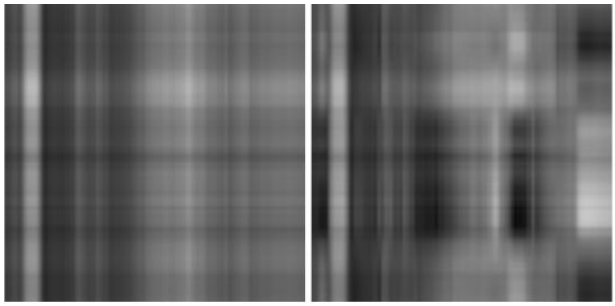
SVD: Basics

Simply put, the Singular Value Decomposition allows us to rewrite a matrix as eigenvalues and eigenvectors. We can then use the SVD to compress data by figuring out which data parts are important and which are not. Eigenvectors associated with larger eigenvalues are more important than those with lower eigenvalues.

SVD on an Image

To help better understand how the SVD works, let's observe how we can use the SVD to compress the following image.

Original Image



One Singular Value-Two Singular Values

Five Singular Values-Ten Singular Values

60 Singular Values-100 Singular Values

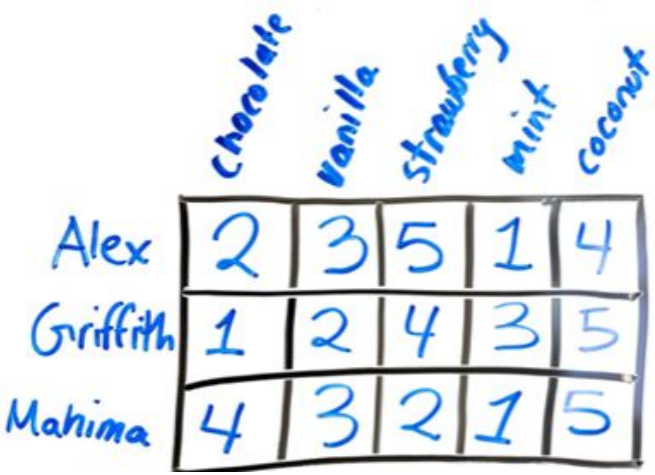
As we increase the number of singular values used, the details of the image start to take form. After using the 10 most significant singular values, we can see the general form of the image. This image takes 96% less storage space.

What data?

Matrices can be used to describe the relationship between two things.

Let's say we (the authors) and a friend rank our top 5 ice cream flavors, with 5

being our favorite and 1 being our least favorite. We could express this in a matrix:



Streaming services like Spotify can make huge matrices like this. For example, Spotify may take a look at what percent of a song you listened to before stopping to find out what keeps your attention. Imagine a matrix of every user and song on Spotify! There's a reason we often leave data analysis to the computers.

What is the SVD?

Mathematically speaking, the SVD is a way of decomposing *any* matrix into the sum of eigenvalues and eigenvectors.

When we have a symmetric matrix, this is very easy. Symmetric matrices can easily be decomposed into the following form:

$$A = \lambda_1 \vec{v}_1 \vec{v}_1^T + \lambda_2 \vec{v}_2 \vec{v}_2^T + \dots \lambda_n \vec{v}_n \vec{v}_n^T$$

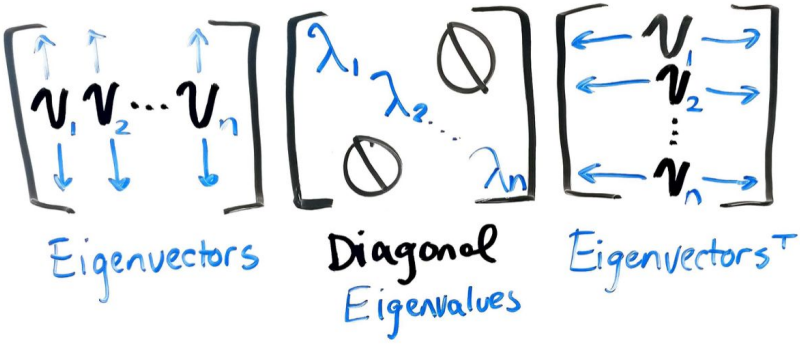
$\vec{v}$ is the eigenvector: $\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$

$\vec{v}^T$ is the transpose of the eigenvector: $[a_1, a_2 \cdots a_n]$

$\vec{v}\vec{v}^T$ forms a matrix the same dimensions as A

This is called eigendecomposition.

It can also be written as three matrices multiplied together:



The multiplication of these matrices is the same as our decomposition above!

When we do the SVD, we are essentially performing the same function on a *non-symmetric* matrix. So, our decomposed equation looks a little different:

$$A = \sqrt{\lambda_1}\,\vec{w}_1\vec{v}_1^{\,T} + \sqrt{\lambda_2}\,\vec{w}_2\vec{v}_2^{\,T} + \ldots \sqrt{\lambda_n}\,\vec{w}_n\vec{v}_n^{\,T}$$

In the SVD: Math section, we'll talk about the specifics of deriving this equation, including what *w* and *v* represent.

Why do we want to do it?

Decomposing a matrix into this equation is more intuitive to us as mathematicians. But more importantly, we can order the terms by magnitude: from the biggest numbers to the smallest ones. This means that we can see **patterns** and **relationships** in data. It also means that if we take a large amount of data, like a photo, and run SVD analysis on it, we can eliminate the least important terms and **compress** data.

SVD: Math

Earlier, we told you that the SVD is just about decomposing any matrix into the form:

$$A = \sqrt{\lambda_1}\vec{w}_1\vec{v}_1^T + \sqrt{\lambda_2}\vec{w}_2\vec{v}_2^T + \ldots \sqrt{\lambda_n}\vec{w}_n\vec{v}_n^T$$

But how and why do we do that?

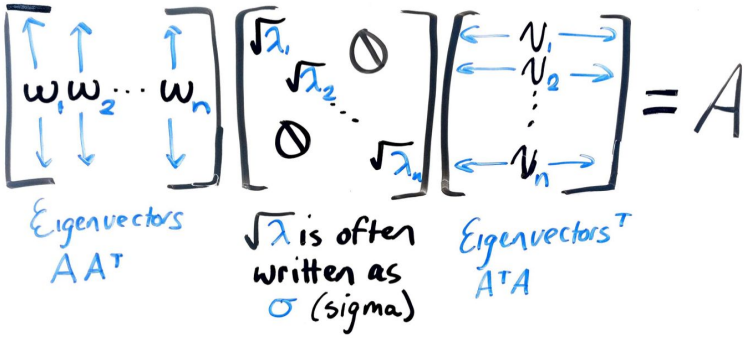
When we have a symmetric matrix A, we can decompose it easily. But when we have a non-symmetric matrix, it's a little harder.

Here's the trick: You can make a symmetric matrix by multiplying A by its transpose!

However, the order you multiply matrices in matters. We have to have a statement that holds true for both A^TA and AA^T. This is why we have two different terms: *v* and *w*. All *v* are the eigenvectors of A^TA and all

w are the eigenvectors of AA^T.
Meanwhile, A^TA and AA^T have the same eigenvalues.

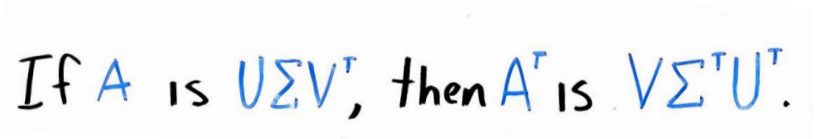
Rewriting the decomposition equation as matrices will provide insight:



These matrices are also written as $U \Sigma V^T$. (Σ is capital Sigma).

How do we prove that this decomposition of A considers both forms of symmetrizing?

Let's start by rewriting A and A^T in terms of $U \Sigma V^T$.



Since Σ is diagonal, its transpose has the same values as the original--though it may have extra 0's if it's not square. Now, let's multiply:

$$A^TA = (V\Sigma U^T)(U\Sigma V^T) = V\Sigma\Sigma V^T$$

$$U^TU = I$$

U^TU forms the identity matrix, so it doesn't affect the end result.

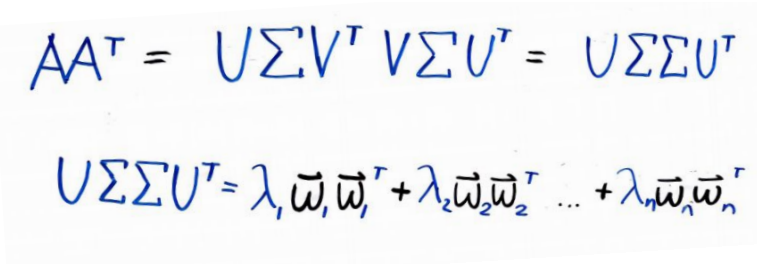
$$\Sigma\Sigma = \begin{bmatrix} \ddots & 0 \\ 0 & \sqrt{\lambda_n} \end{bmatrix} \cdot \begin{bmatrix} \ddots & 0 \\ 0 & \sqrt{\lambda_n} \end{bmatrix} = \begin{bmatrix} \ddots & 0 \\ 0 & \lambda_n \end{bmatrix}$$

Since Σ is full of the singular values $\sqrt{\lambda}$, multiplying it by itself gives us the diagonal matrix of λ.

Therefore:

$$V\Sigma\Sigma V^T = \lambda_1\vec{v}_1\vec{v}_1^T + \lambda_2\vec{v}_2\vec{v}_2^T \ldots + \lambda_n\vec{v}_n\vec{v}_n^T$$

Which is a symmetric eigendecomposition! You can trust us in saying that AA^T yields the decomposition for eigenvectors *w*, but here's a short run-through just to prove it:



So once again, our singular value decomposition of the non-symmetric matrix A is proven true as it yields the eigendecomposition of AA^T.

SVD: Implementation

Wow, that is a lot of math. Why would I ever want to do that? Well short answer, you don't.

Now that you understand some of the logic behind it, we can just use computers to do the work.

To start off with, here is how we find the Singular Value Decomposition using different tools.

You can download the sample code here: https://github.com/mstites/Linearity-Zine

MATLAB

Learn how to use the SVD in MATLAB.

```
A = [1 2 3; 4 5 6]
[U,S,V] = svd(A)
```

```
U =

   -0.3863   -0.9224
   -0.9224    0.3863

S =

    9.5080         0         0
         0    0.7729         0

V =

   -0.4287    0.8060    0.4082
   -0.5663    0.1124   -0.8165
```

```
   -0.7039   -0.5812  0.4082
```

This performs the SVD on matrix A, such that USV'=A. We can do this in MATLAB with the following command:

```
U*S*V'
```

```
ans =

    1.0000  2.0000  3.0000
    4.0000  5.0000  6.0000
```

Matlab – On Image

Now, let's learn how to use the SVD on an image.

```
img = imread('cat.jpg');
img_double = im2double(img);
```

Opens the file and converts it to double precision.

```
[U,S,V] = svd(img_double);
```

Performs the SVD on the image matrix.

```
rank = 10;
L = U(:,1:rank) * S(1:rank,
1:rank) * V(:, 1:rank)';
```

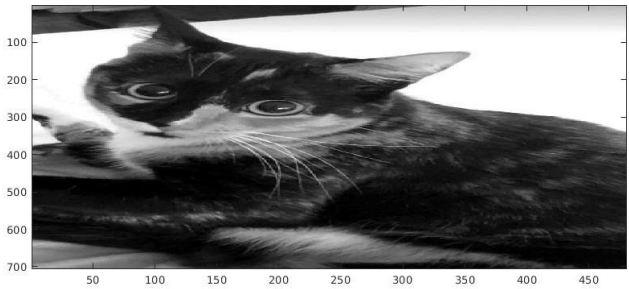
Chooses the first 10 singular vectors from the SVD breakdown.

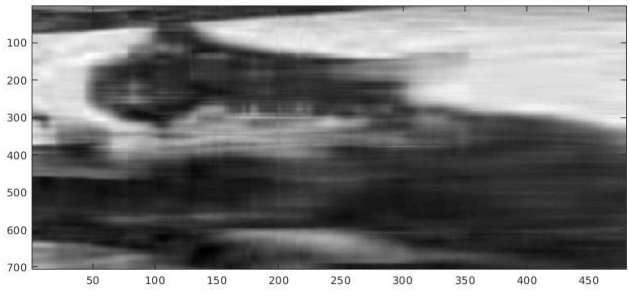
```
colormap('gray')

subplot(211)
imagesc(img) % original
image
```

```
subplot(212)
imagesc(L) % compressed
image
```

Displays both the original image and the compressed image.





Want to experiment? By changing the rank number, you can determine how

many terms you're taking in the compression. How many singular values do you need to get an image that looks like the original? How few do you need to tell it's a cat? You can also insert and compress your own images. If you download the image, you can see how much data is saved as well.

Python

Quick info: Python is free and easy to install, download here: https://www.anaconda.com/distribution/

Once you have python installed, use your favorite text editor (IE Atom) to create some Python scripts. Alternatively, you can use Jupyter notebooks (which should install with Anaconda).

```
import imageio
import numpy as np
from numpy import ndarray
```

We will be using NumPy, a package installed by default, for our matrix operations. ImageIo is used for opening an image.

```
img = Image.open('cat.jpg')
img_m = np.asarray(img,
dtype="int32")
```

Opens the grayscale image “cat.jpg” and create a matrix to represent it.

```
U, S, V =
np.linalg.svd(img_m,
full_matrices = True)
```

Performs the SVD on the image matrix.

```
u_size = np.shape(U)
s_size = np.shape(S)
v_size = np.shape(V)
print('U is:', u_size, '\nS
is:',s_size, '\nV is:',
v_size)
U is: (706, 706)
```

```
S is: (480, 480)
V is: (480, 480)
```

Checks the size of the matrices.

To multiply U, S, and V together we need S to be (706, 480). We could also change the size of U, but this is more challenging so we are opting for the simple solution.

```
S_new = np.zeros((706, 480))
S_new[:480, :480] = S
```

Adds zeros to the extra rows.

```
a = U @ S_new @ V
np.isclose(a, img_m)
```

Multiplies U, S, and V together. It uses the isclose command to check that a is equivalent (ignoring floating point precision errors) to our original image matrix.

Challenge: The above code only breaks down a matrix into the SVD, it does not compress it. Write a program to compress data using the SVD in Python.

WolframAlpha

Getting the Singular Value Decomposition for a matrix is very simple in WolframAlpha, just use the SVD command.

For example: “SVD {{1,0,-1},{-2,1,4}}”.

For further resources and questions, make sure to check out our website:

www.mstites.com/Linearity-Zine/

Thank you so much for reading!