Murat Aydos
Jan 28, 2019 · 6 min read

Image Processing with Singular Value Decomposition

Singular Value Decomposition (SVD) is used widely in signal processing. Noise reduction and image compression are some of the applications of SVD. I'd like to explain the underlying concepts of SVD which can be applied to different problems and then give some examples of noise reduction and image compression.

As a starting point, I am obligated to write the formula. This is to understand the structure; we will come to the meaning of it later.

$$A = U \Sigma V$$

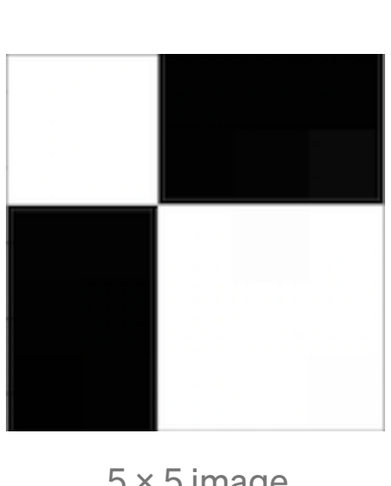
A is an *m x n* matrix you originally have either from an image or some other data source. *U* and *V* are orthogonal matrices and *Σ* is a diagonal matrix.

OK, we are mostly done with formality.

SVD Example

Let's apply SVD to a very simple matrix and discuss what's happening.

We have a *5 x 5* grayscale image with these pixel values:



5 × 5 image

```
255, 255,   2,   2,   2
255, 255,   3,   5,  10
  4,   4, 255, 253, 255
  3,   2, 255, 255, 255
  1,   3, 255, 255, 254
```

And here is the calculated *U* matrix:

```
-0.01,  0.70,  0.67, -0.19,  0.02

-0.02,  0.70, -0.67,  0.19, -0.03

-0.57, -0.01, -0.16, -0.48,  0.63

-0.57, -0.02, -0.04, -0.29, -0.75

-0.57, -0.02,  0.21,  0.77,  0.12
```

Imagine columns of *U* as concepts and rows of *U* as the relations between *A*'s rows' and these concepts. For example, we can easily say that there are two concepts in this picture:

1. Black -> White
2. White -> Black

The first column of *U* is for the 1st concept since the larger values only appear for the last 3 columns all of which start with black and continues with white. And the second column of U is for the 2nd concept for the same reason.

Σ values are as below:

```
764.29,      0,    0,    0,    0
     0, 509.74,    0,    0,    0
     0,      0, 3.72,    0,    0
     0,      0,    0, 1.62,    0
     0,      0,    0,    0, 1.21
```

You can interpret the values of *Σ* as the magnitude of concepts. This matrix tells us that the first two concepts are the most important which means the other concepts are just noise. We will come to this noise part again.

Finally the *V* values:

```
-0.02, -0.02, -0.57, -0.57, -0.57

 0.70,  0.70, -0.02, -0.01, -0.01

-0.06,  0.07,  0.53,  0.26, -0.79

-0.56,  0.56, -0.40,  0.43, -0.03

-0.41,  0.41,  0.46, -0.64,  0.17
```

You can think of *V* values as similar to *U* but this time for *A*'s columns instead of rows. So each row of *V* represents a concept and columns of *V* are the relations between A's columns and the concepts.

In this example, you can easily see the last 3 columns of A belong to the 1st concept and first 2 columns of A belong to the 2nd concept.

Noise Reduction with SVD

We have all the decomposed matrices; *U*, *Σ* and *V*. Now what?

As I mentioned above, *Σ* tells us how important a concept is in the given matrix *A*. What if we remove all the unimportant concepts simply by changing the values of *Σ* to 0 which are smaller than some threshold?

So just update the *Σ* as:

```
764.29,      0,    0,    0,    0
     0, 509.74,    0,    0,    0
     0,      0,    0,    0,    0
     0,      0,    0,    0,    0
     0,      0,    0,    0,    0
```

Then calculate the A again with our magic formula: $A = U \Sigma V$.

Here is the result:

```
254, 254,   0,   1,   4

255, 255,   4,   5,   8

  3,   4, 254, 253, 254

  2,   2, 255, 254, 255

  1,   2, 254, 254, 254
```

We got almost the same matrix back, just a little bit smoother. However, in this example, we didn't have so much noise so the result is not too different. But this example clearly shows that we can get the same looking image by omitting the small *Σ* values.

Let's see some examples with Python so we can apply this method to large images.

Denoising with SVD in Python (SciPy)

Now let's work on a larger image to reduce noise with Python — SciPy. First I will add some noise to a photo that I took in Morocco.

Gray Original (1000 × 667)

3% Noised (1000 × 667)

Now we need to calculate *U*, *Σ* and *V*. Then, we will choose a threshold to discard the rest of the *Σ*. For this example, I chose the *Σ* threshold as 250 since it seems like giving the best result but you can play with it and observe the results.

```
import numpy as np
from scipy import misc

# Load the image
img = misc.imread('morocco_noised.jpg', flatten=True)

# Calculate U (u), Σ (s) and V (vh)
u, s, vh = np.linalg.svd(img, full_matrices=False)

# Remove sigma values below threshold (250)
s_cleaned = np.array([si if si > 250 else 0 for si in s])

# Calculate A' = U * Σ (cleaned) * V
img_denoised = np.array(np.dot(u * s_cleaned, vh), dtype=int)

# Save the new image
misc.imsave('morocco_denoised.jpg', img_denoised)
```

And below is the resulting image which is a little smoother than the noised but of course not as good as the original one.

Denoised (1000 × 667)

Image Compression

If you check the *Σ* in the desert image example, you'd see that its size is 667. And by omitting everything under 250, we basically discard 583 columns since only the first 83 columns are above 250.

```
>>> u.shape
(667, 667)

>>> vh.shape
(667, 1000)

>>> s.shape
(667,)
```

If we wanted to store all *U*, *Σ* and *V*, we would need 1,112,556 bytes which is almost twice the image size (1000 * 667 = 667,000).

- *U*: 667 * 667 = 444,889
- *V*: 667 * 1000 = 667,000
- *Σ*: 667 * 1 = 667
- 444,889 + 667,000 + 667 = 1,112,556

Now that we discarded most of the *Σ*, we only need 138,444 with SVD image compression method.

- U: 667 * 83 = 55,361
- V: 1000 * 83 = 83,000
- *Σ*: 83 * 1 = 83
- 55,361 + 83,000 + 83 = 138,444

Let's apply the same method to the original image. However, the *Σ* values are different for the original image so I choose 200 as the threshold.

```
import numpy as np
from scipy import misc

# Load image
img = misc.imread('morocco_bw.jpg', flatten=True)

# Calculate U (u), Σ (s) and V (vh)
u, s, vh = np.linalg.svd(img, full_matrices=False)

# Remove sigma values below threshold (200)
s_cleaned = np.array([si if si > 200 else 0 for si in s])

# Calculate A' = U * Σ (cleaned) * V
img_compressed = np.array(np.dot(u * s_cleaned, vh), dtype=int)

# Save the new image
misc.imsave('morocco_compressed.jpg', img_compressed)
```

Compressed (1000 × 667)

The first 97 values of *Σ* are greater than 200 so we can save this image in 161,796 bytes with a decent quality instead of 667,000 bytes.

- U: 667 * 97 = 64,699
- V: 1000 * 97 = 97,000
- *Σ*: 97 * 1 = 97
- 64,699 + 97,000 + 97 = 161,796

If we increased the compression and chose a higher threshold like 500, the result would be as following and it could be stored in 63,384 bytes.

Even though the last image seems quite distorted compared to the original image, its size is less than 10% of the original size and it is still able to give good amount of details. However, it's also adding some details that doesn't exist in the original image :)