Software Project

AI25BTECH11024 - Pratyush Panda

1 Professor Gilbert Strang's Video Summary

Professor Gilbert Strang's video on Singular Value Decomposition (SVD) is a very basic description of the concept. Basically, SVD is used to express any matrix $\mathbf{A}$ as a product of three special matrices as;

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{0.1}$$

where, $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices and $\mathbf{\Sigma}$ is a diagonal matrix.
In the video it is said that $\mathbf{V}$ can be found out by getting the eigen vectors of the matrix $\mathbf{A}^T\mathbf{A}$. Since $\mathbf{A}^T\mathbf{A}$ is a symmetric matrix, its eigen vectors are orthogonal.

$$\mathbf{A}^T\mathbf{A}\mathbf{v}_k = \sigma_k^2\mathbf{v}_k \tag{0.2}$$

And $\mathbf{U}$ can be found by finding the eigen vectors of $\mathbf{A}\mathbf{A}^T$.

$$\mathbf{A}\mathbf{A}^T\mathbf{u}_k = \sigma_k^2\mathbf{u}_k \tag{0.3}$$

And the singular values, which are the entries of the diagonal matrix are the square root of the eigen values of either $\mathbf{A}^T\mathbf{A}$ or $\mathbf{A}\mathbf{A}^T$ since they are the same. Using these results given in the video, we can find the SVD of any matrix.

2 Explanation of the Implementation of Algorithm

In the project, we are considering the given gray-scale image as a matrix. Then we compress it by expressing the matrix as the SVD of its first k most dominant eigen vectors. In this project, we are approximating the SVD by using the Power Iteration method.

Power Iteration is a method by which we can find the most dominant eigen vector in a given matrix. The algorithm that power iteration follows is:

$$\mathbf{v}_(\mathbf{k}+\mathbf{1}) = \mathbf{A}\mathbf{v}_\mathbf{k} \tag{0.4}$$

$$\mathbf{v}_(\mathbf{k}+\mathbf{1}) = \mathbf{v}_(\mathbf{k}+\mathbf{1})/|\mathbf{v}_(\mathbf{k}+\mathbf{1})| \qquad \textit{normalizing after each step} \tag{0.5}$$

The more times we iterate this, the better approximation we can get of the eigen vector. Thus, in the code the number of iterations is set to 200. Now, after getting the eigen vector.

We now find the eigen values using the relation:

$$\lambda = \|\mathbf{Av}\| \tag{0.6}$$

To get the SVD, we apply the concept of power iteration to the matrix $\mathbf{A}^T\mathbf{A}$. Where every eigen vector ($\mathbf{v}_k$) are the right singular vectors. And the singular values (σ) are $\sqrt{\lambda_k}$ and the left singular values are computed as:

$$\mathbf{u}_k = \frac{\mathbf{Av}_k}{\sigma_k} \tag{0.7}$$

Now to find k distinct eigen vectors, we deflate the matrix $\mathbf{A}^T\mathbf{A}$ after each iteration:

$$\mathbf{A}^T\mathbf{A} = \mathbf{A}^T\mathbf{A} - \sigma_k^2\mathbf{v}_k\mathbf{v}_k^T \tag{0.8}$$

Thus we can get all the three matrices for the SVD.

In the code, we have first defined some basic matrix functions which are used repeatedly. The power iteration algorithm is made using simple matrix multiplication and normalization methods. And the SVD method is implemented using simple array logic following the steps given above.

3 Comparing different algorithms with the Power Iteration method

The algorithm of my choice for this project is the Power Iteration method. Other than that there are several other methods for doing the same task such as Inverse Iteration, Jacobi method, QR algorithm etc. While all of these attain the same objective and some algorithms like QR algorithm or Inverse Iteration might be more efficient but are much more complex to make and require much more advanced concepts, and some like Jacobi method have constraints like it can only find eigen vectors for symmetric matrices.

So, considering the complexity of the algorithm and other constraints, Power Iteration is quite balanced in terms of complexity and efficiency. Thus, I decided of implementing it in the code.

4 Reconstructed image for different k values

For each test image, four reconstructed images are made with k values as $k = 50$, $k = 90$, $k = 150$, and $k = 200$. The images are given below:

The first image is the globe.

Fig. 0.1: img1

For k=50

Fig. 0.2: k=50

For k=90

Fig. 0.3: k=90

For k=150

Fig. 0.4: k=150

For k=200

Fig. 0.5: k=200

The second image is the gray-scale.

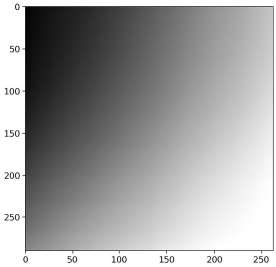

Fig. 0.6: img2

For k=50

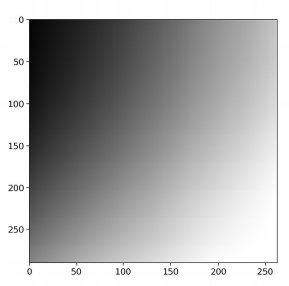

Fig. 0.7: k=50

For k=90

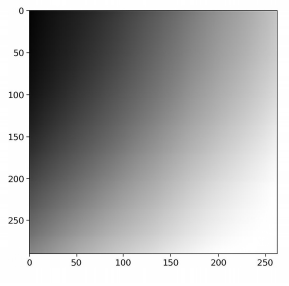

Fig. 0.8: k=90

For k=150

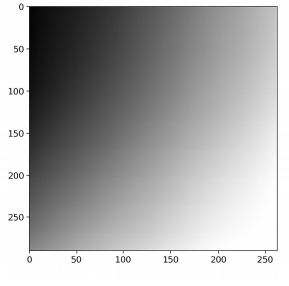

Fig. 0.9: k=150

For k=200

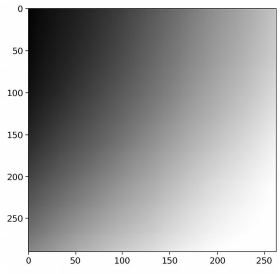

Fig. 0.10: k=200

The third image is of Einstein.

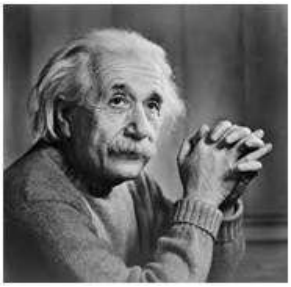

Fig. 0.11: img3

For k=50

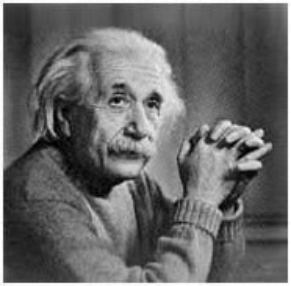

Fig. 0.12: k=50

For k=90

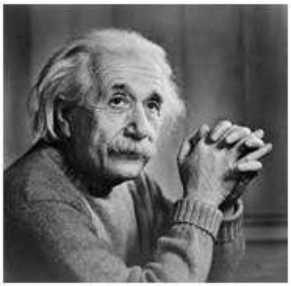

Fig. 0.13: k=90

For k=150

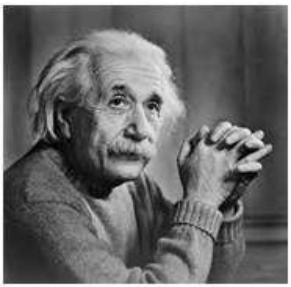

Fig. 0.14: k=150

For k=200

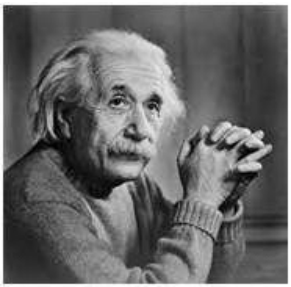

Fig. 0.15: k=200

5 Error Analysis

The error being calculated are Frobenius Error which is defined as:
For the given matrix $\mathbf{A}_i$, and the resultant matrix $\mathbf{A}_f$ the error can be written as:

$$e = \|\mathbf{A}_f - \mathbf{A}_i\|_F \tag{0.9}$$

Error for the globe picture:

k Value	Frobenius Error
50	24.550934
90	16.140119
150	10.328848
200	7.526983

TABLE 0: img 1

Error for the gray-scale image:

k Value	Frobenius Error
50	4.666718
90	1.982175
150	1.170241
200	0.789041

TABLE 0: img 2

Error for the Einstein image:

k Value	Frobenius Error
50	3.699289
90	1.067723
150	0.039846
200	0.176103

TABLE 0: img 3

In general, after high values in k, there was not a lot of changes in the image, but for small values of k, the image quality did become very poor. Frobenius Error is a good way to judge the difference between the image quality. We can see that when we move from k=50 to k=90 the error decreases quite a lot, while when we go from k=150 to k=200, the change in error is not that much. Thus we can say, for large values of k, the image quality does not improve much and converges to the original image.

6 Discussion and trade-offs and reflections on the algorithm

In short, the chosen algorithm proved to be quite difficult to implement and had quite a large time complexity, making the code inefficient

But it was good for learning such approximation methods and increased my understanding. I feel that the choice was not that bad and could have been implemented in the code better. But the code is still not very slow and gives decent output.