Software Assignment

EE25BTECH11004 - Aditya Appana

November 7th, 2025

Analysis of Professor Strang's Lecture

The aim of the singular value decomposition is to be able to decompose a matrix of any dimensions into the product of an orthogonal matrix, a diagonal matrix, and another orthogonal matrix. We want to find such a decomposition that is similar to the spectral decomposition of positive definite matrices.
We want to find an orthogonal basis, on which if the matrix transformation $\mathbf{A}$ is applied, we obtain an orthogonal basis in the column space of $\mathbf{A}$ as well. Fundamentally,
$\mathbf{v_1}, \mathbf{v_2} \dots \mathbf{v_r}$ is an orthonormal basis in the row space (which can be seen by the transpose on vector $\mathbf{V}$)
$\mathbf{u_1}, \mathbf{u_2} \dots \mathbf{u_r}$ is an orthonormal basis in the column space.
$\mathbf{v_{r+1}}, \mathbf{v_{r+2}} \dots \mathbf{v_n}$ is an orthonormal basis in the null space of $\mathbf{A}$.
$\mathbf{u_{r+1}}, \mathbf{u_{r+2}} \dots \mathbf{u_n}$ is an orthonormal basis in the null space of $\mathbf{A^T}$.

Algorithm

For the implementation of the Singular Value Decomposition, I chose the **QR Iteration with Gram-Schmidt Orthogonalization** algorithm. This algorithm involves the repeated QR decomposition of a symmetric matrix till we converge on a diagonal matrix. The diagonal elements of the diagonal matrix will be the eigenvalues of the symmetric matrix.

0.1 Explanation

Let the image be represented as a matrix X. Since this algorithm operates on a square matrix, we can use the matrices $\mathbf{X^TX}$ and $\mathbf{XX^T}$. We will demonstrate the algorithm on an arbitrary symmetric matrix $\mathbf{A_k}$. We start by decomposing $\mathbf{A_0}$:

$$\mathbf{A_0} = \mathbf{Q_0R_0} \tag{1}$$

Here $\mathbf{Q_k}$ and $\mathbf{R_k}$ are an orthogonal and a diagonal matrix respectively.
We now form a new matrix $\mathbf{A_1}$,

$$\mathbf{A_1} = \mathbf{R_0Q_0} \tag{2}$$

and then further decompose it as follows:

$$\mathbf{A_1} = \mathbf{Q_1R_1} \tag{3}$$

We iterate through this algorithm until we converge to a diagonal matrix $\mathbf{A_k}$ of satisfactory precision. We don't obtain the perfect eigen-values from the diagonal matrix, but simply approximated eigen-values.
The mathematical explanation of this algorithm is below.

We converged on $\mathbf{A_k}$ after the iteration:

$$\mathbf{A_k} = \mathbf{Q_kR_k} \tag{4}$$

However, we firstly obtained $\mathbf{A_k}$ as:

$$\mathbf{A_k} = \mathbf{R_{k-1}Q_{k-1}} \tag{5}$$

and $\mathbf{A_{k-1}}$ as

$$\mathbf{A_{k-1}} = \mathbf{Q_{k-1}R_{k-1}} \tag{6}$$

Since Q is orthogonal, its inverse will simply be its transpose. We can therefore write:

$$\mathbf{R_{k-1}} = \mathbf{Q_{k-1}^{-1}A_{k-1}} = \mathbf{Q_{k-1}^{T}A_{k-1}}. \tag{7}$$

Substituting equation **(7)** in equation **(5)**:

$$\mathbf{A_k} = \mathbf{Q_{k-1}^T A_{k-1} Q_{k-1}} \tag{8}$$

We can perform a similar set of operations on $\mathbf{A_{k-1}}$ to obtain:

$$\mathbf{A_k} = (\mathbf{Q_{k-1}^T Q_{k-2}^T})\mathbf{A_{k-2}}(\mathbf{Q_{k-2} Q_{k-1}}) \tag{9}$$

This will eventually become:

$$\mathbf{A_k} = (\mathbf{Q_{k-1}^T Q_{k-2}^T} \dots \mathbf{Q_0^T})\mathbf{A_0}(\mathbf{Q_0} \dots \mathbf{Q_{k-2} Q_{k-1}}) \tag{10}$$

$$\mathbf{A_k} = (\mathbf{Q_0} \dots \mathbf{Q_{k-2} Q_{k-1}})^T \mathbf{A_0}(\mathbf{Q_0} \dots \mathbf{Q_{k-2} Q_{k-1}}) \tag{11}$$

$$\mathbf{A_k} = \Pi^T \mathbf{A_0} \Pi \tag{12}$$

Since Π is a product of orthogonal matrices, it will itself be an orthogonal matrix. Therefore we can rearrange the equation to the form:

$$\Pi \mathbf{A_k} \Pi^T = \mathbf{A_0} \tag{13}$$

where Π is an orthogonal matrix, $\mathbf{A_k}$ is (approximately) a diagonal matrix, and $\mathbf{A_0}$ is a symmetric matrix. **This is essentially the spectral decomposition of the symmetric matrix.** Therefore, Π is the eigen-vector matrix and $\mathbf{A_k}$ is the eigen-value matrix. Now, for the image matrix $\mathbf{X}$, we can express it in the form of its singular decomposition as:

$$\mathbf{X} = \mathbf{U \Sigma V^T} \tag{14}$$

Therefore, the symmetric matrix $\mathbf{X^T X}$ can be expressed as:

$$\mathbf{X^T X} = \mathbf{V \Sigma^T U^T U \Sigma V^T} \tag{15}$$

$$\mathbf{X^T X} = \mathbf{V \Sigma^2 V^T} \tag{16}$$

Similarly, the symmetric matrix $\mathbf{X X^T}$ can be expressed as:

$$\mathbf{X X^T} = \mathbf{U \Sigma^2 U^T} \tag{17}$$

It can be shown that $\mathbf{\Sigma}$ will be the same for both symmetric matrices, and the diagonal elements of $\mathbf{XX^T}$ or $\mathbf{X^TX}$ are the squares of the singular values of the matrix $\mathbf{X}$. Therefore the $\mathbf{\Pi}$ of $\mathbf{XX^T}$ and $\mathbf{X^TX}$ will be the $\mathbf{U}$ and $\mathbf{V}$ of the SVD of $\mathbf{X}$, respectively.

We can now perform the k-rank approximation of $\mathbf{X}$ by using the below formula:

$$\mathbf{X_{approximated}} = \sum_{i=1}^{k} \sigma_{\mathbf{i}} \mathbf{u_i} \mathbf{v_i^T} \tag{18}$$

0.2 Pseudocode

1. Input the image into the python code using the matplotlib command *imread*

2. A three-dimensional array will be returned, which we can convert to a one-dimensional array by taking the mean of the three elements in each row.

3. Send this matrix to the C object
   (a) Define functions for matrix multiplication, vector inner product, vector outerproduct, scalar multiplication, QR decomposition, transpose, vector addition, vector subtraction, and converting between a one-dimensional contiguous array and a pointer to two-dimensional array
   (b) Apply the Gram-Schmidt algorithm for orthogonalisation, and return the **Q** and **R** to the iterative function which performs **QR** iteration.
   (c) Apply the **QR** iteration as discussed iin the mathematical explanation, while simultaneously multiplying the obtained **Q's** together to get the eigen-vector matrix.
   (d) After obtaining the singular value decomposition, take the input of the rank for the desired rank-approximation
   (e) Perform equation (18) with k = rank
   (f) Convert the array into a one dimensional continguous form and return the array to Python.

4. Display the image using the matplotlib command *imshow*

Reconstructed Images

For *greyscale.png*, the rank-5 approximated image looks like this:

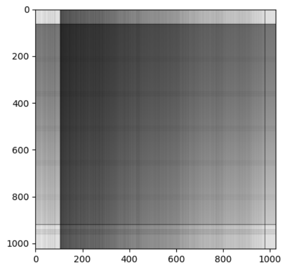

Figure 1: greyscale.png

For *globe.jpeg*, the rank-5 approximated image looks like this:

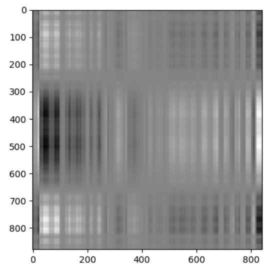

Figure 2: globe.jpeg

For *einstein.jpg*, the rank-5 approximated image looks like this:

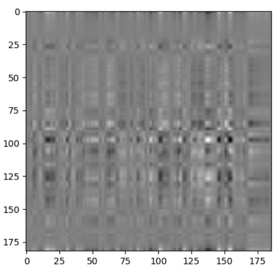

Figure 3: einstein.jpeg

Reasons for Choosing the Algorithm

I chose this particular algorithm since it seemed the most mathematically feasible out of all the algorithms I researched about. The algorithm seemed very straightforward and easy to implement, and was based on linear algebra concepts which I was already aware of and had a good understanding of. I was not able to mathematically understand other algorithms involving processes like bidiagonalisation, householder reflections, etc. so I implemented this algorithm.
However, though this algorithm is mathematically very simple, it is hard to implement it accurately in code as it is plagued with errors related to arithmetic operations.

Tradeoffs

After executing the program, I realised there were several shortcomings and tradeoffs associated with this particular algorithm.

1. The Gram-Schmidt orthogonalization is not necessarily the best method for the orthogonalization of the given basis. It is the easiest way to find an orthogonal basis on paper, but it fails during the implementation of the program due to the drawbacks of floating point arithmetic.
   If the columns in the original matrix are very closely linearly dependent, we end up subtracting almost identical vectors during the orthogonalisation process, and due to the errors associated with floating point values, most of the significant digits of the elements are wiped out, and the resulting vector

will not be orthogonal. **In short, the method is not sustainable in code due the errors and volatility associated with floating point values.**

2. Some of the processes involved in the method I used for applying **QR** iteration are particularly expensive and time consuming. The repeated looping for calculating the singular values, and the repeated looping for sorting the singular values increase the time complexity of the algorithm, ultimately making it a poor choice to apply.

3. As seen in the reconstructed images, the drawbacks associated with the algorithm result in very poorly reconstructed images.

4. I was misled into thinking that since the Gram-Schmidt is extremely simple and easy to apply on paper, it would be the same case while performing it through code. However, due to the very large time complexity of the processes involved and the floating point errors, it is not a suitable method.