Software Project

EE25BTECH11012-BEERAM MADHURI

Computing SVD of matrix A

Summary of Strang's video

Goal: $A = U\Sigma V^\top$
Express matrix A as product of U, Σ, $V^\top$ matrices
Where,

$$U = Orthogonalmatrix$$
$$V = Orthogonalmatrix$$
$$\Sigma = diagonalmatrix$$

$$\Sigma = \begin{bmatrix} \sigma_1 & & \\ & \sigma_2 & \\ & & \ddots \end{bmatrix}$$

Computation:
let $u_1, u_2 \dots u_m$ be the unit vectors in the column space of A
$v_1, v_2 \dots v_n$ are the unit vectors in row space of A

then, we are taking the typical vectors

$$\sigma_1 u_1 = Av_1$$
$$\sigma_2 u_2 = Av_2$$
$$\vdots$$
$$\sigma_r u_r = Av_r$$

$u_1, u_2 \dots u_r$ are orthogonal to each other.
$v_1, v_2 \dots v_r$ are orthogonal to each other.

$$A\begin{bmatrix} v_1 & v_2 & v_3 & \dots & v_r \end{bmatrix} = \begin{bmatrix} u_1 & u_2 & \dots & u_r \end{bmatrix}\begin{bmatrix} \sigma_1 & 0 & 0 & \dots \\ 0 & \sigma_2 & & \dots \\ \vdots & & \ddots & \end{bmatrix}$$

$\therefore$ We have to find the orthonormal basis in the row space and orthonormal basis in the column space of A such that

$$AV = U\Sigma$$

as V and U are Orthogonal Matrices

$$UU^\top = I$$
$$VV^\top = I$$
$$AVV^\top = U\Sigma V^\top$$
$$A = U\Sigma V^\top$$

Multiplying $A^\top$ with A

$$A^\top A = (V\Sigma^\top U^\top)(U\Sigma V^\top)$$
$$A^\top A = V\Sigma^\top (U^\top U)\Sigma V^\top$$
$$A^\top A = V\Sigma^\top \Sigma V^\top$$
$$A^\top A = V \begin{pmatrix} \sigma_1^2 & 0 & 0\cdots \\ 0 & \sigma_2^2 & 0\cdots \\ 0 & 0 & \sigma_2^2\cdots \\ \vdots & & \ddots \end{pmatrix} V^\top$$

let $A^\top$A =B and $\Sigma^\top\Sigma = M$
$B = VMV^\top$
In the same way

$$AA^\top = U \begin{pmatrix} \sigma_1^2 & 0 & 0\cdots \\ 0 & \sigma_2^2 & 0\cdots \\ 0 & 0 & \sigma_2^2\cdots \\ \vdots & & \ddots \end{pmatrix} U^\top$$

$\therefore$ V is the Eigen vector matrix of matrix $A^\top$A
$\therefore$ Σ is the diagonal matrix with eigen values of $A^\top A$ as it's diagonal entries.
$\therefore$ U is the Eigen vector matrix of matrix $AA^\top$
here,
$V_1, V_2, \cdots V_r$ are the orthonormal basis for rowspace of A.
$U_1, U_2, \cdots U_r$ are the orthonormal basis for columnspace of A.
$V_(r+1), \cdots V_n$ are the orthonormal basis for nullspace of A.
$U_(r+1), \cdots U_m$ are the orthonormal basis for nullspace of $A^\top$.

Explaination of Implemented Algorithm

Power iteration with matrix deflation is done by svd.c code.

1) The inner loop that calculates $v_new = matrix_multiply(A_T Av)$ is the power iteration. It's a method to find the dominant eigenvector (the v_1 corresponding to the largest singular value) of the matrix $A^\top A$.

2) After finding the top singular triplet (σ_1, u_1, v_1), we do two things:
Reconstruct: We add this rank-1 matrix ($A_1 = \sigma_1 u_1 v_1^T$) to our final image, A_k.

Deflate: We subtract that same rank-1 matrix from our temporary matrix ($A_temp = A_temp - A_1$).

3) The main for loop repeats this k times. On the second iteration, the power method automatically finds the next largest singular value/vector triplet because we've already "deflated" (removed) the first one.

Why Power Iteration Algorithm over other algorithms

I studied about the following algorithms:

1) Jacobi SVD Algorithm
2) Golub-Kahan Algorithm
3) Power Iteration

1. Jacobi SVD Algorithm:

This algorithm applies a series of rotations to diagonalize the matrix
in this first we compute $A^\top A$
then apply Jacobi rotation :

$$G^\top (A^\top A) G = diagonal$$

$$\text{where, } G = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

equate non diagonal elements to zero to find the value of theta.
then the root of diagonal elements of this matrix gives σ_1, σ_2 etc.

$$V = G$$

$$U = AV\Sigma^-1$$

It computes all singular values even if we require only some k values.
We have to write code to "sweep" through the matrix, calculate the correct rotation angles for each step, and update two other matrices (U and V) at every single step.

2. Golub-Kahan Algorithm:

Instead of directly computing U, Σ, V, we do it in two major steps:
1. Bidiagonalization using Householder transformations
We reduce A to a bidiagonal matrix B, such that:

$$A = PBQ^\top$$

where:
P and Q are orthogonal,
B has nonzero entries only on its main diagonal and superdiagonal.
2.Compute the SVD of the bidiagonal matrix B
Once A is reduced to B, we use a Golub-Kahan SVD iteration (a stable QR-like method) to diagonalize B to find
U,Σ, V.

uses implicit QR steps on $B^\top B$,
converges to the diagonal matrix of singular values Σ After Iteration we get:

$$\begin{aligned} B &= U_B \Sigma V_B^\top \\ A &= (PU_B)\Sigma(QV_B) \\ A &= U\Sigma V \\ \text{where,} U &= PU_B \\ V &= QV_B \end{aligned}$$

it is extremely complex.
Just like Jacobi, it's designed to compute the Full SVD, which is not our goal.

3. Power Iteration Algorithm:

In this process, we find the largest eigen value and eigen vector of $A^{\vec{\top}}A$ For Example: let

$$\vec{A} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

The power iteration finds the largest eigenvector of A^TA. **Calculate A^TA:**

$$A^T = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \quad A^TA = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}\begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

Pick a random starting vector $\mathbf{v}_0$: Let's start with

$$\mathbf{v}_0 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}.$$

Iterate: $\mathbf{v}_{\text{new}} = (A^TA) \cdot \mathbf{v}_{\text{old}}$

- **Iteration 1:** $\mathbf{v}' = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}\begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$ Normalize it:

$$\|\mathbf{v}'\| = \sqrt{2^2 + 3^2} = \sqrt{13} \approx 3.605$$

$$\mathbf{v}_1 = \begin{bmatrix} 2/3.605 \\ 3/3.605 \end{bmatrix} = \begin{bmatrix} 0.555 \\ 0.832 \end{bmatrix}$$

- **Iteration 2:** $\mathbf{v}' = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}\begin{bmatrix} 0.555 \\ 0.832 \end{bmatrix} = \begin{bmatrix} 1.387 \\ 2.219 \end{bmatrix}$ Normalize it:

$$\|\mathbf{v}'\| = \sqrt{1.387^2 + 2.219^2} \approx 2.617$$

$$\mathbf{v}_2 = \begin{bmatrix} 1.387/2.617 \\ 2.219/2.617 \end{bmatrix} = \begin{bmatrix} 0.530 \\ 0.848 \end{bmatrix}$$

- **Iteration 3:**

$$\mathbf{v}' = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}\begin{bmatrix} 0.530 \\ 0.848 \end{bmatrix} = \begin{bmatrix} 1.378 \\ 2.226 \end{bmatrix} \quad \text{Normalize it:} \quad \|\mathbf{v}'\| = \sqrt{1.378^2 + 2.226^2} \approx 2.618$$

$$\mathbf{v}_3 = \begin{bmatrix} 1.378/2.618 \\ 2.226/2.618 \end{bmatrix} = \begin{bmatrix} 0.526 \\ 0.850 \end{bmatrix}$$

The vector is converging! We can stop here. This is our first right singular vector, $\mathbf{v}_1$.

$$\mathbf{v}_1 = \begin{bmatrix} 0.526 \\ 0.850 \end{bmatrix}$$

- $\mathbf{u}_1' = A\mathbf{v}_1 = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 0.526 \\ 0.850 \end{bmatrix} = \begin{bmatrix} 1.376 \\ 0.850 \end{bmatrix}$
- $\sigma_1 = \|\mathbf{u}_1'\| = \sqrt{1.376^2 + 0.850^2} \approx \sqrt{2.616} \approx 1.618$
- $\mathbf{u}_1 = \mathbf{u}_1'/\sigma_1 = \frac{1}{1.618} \begin{bmatrix} 1.376 \\ 0.850 \end{bmatrix} = \begin{bmatrix} 1.376/1.618 \\ 0.850/1.618 \end{bmatrix} = \begin{bmatrix} 0.850 \\ 0.525 \end{bmatrix}$

We have found the largest singular triplet: $(\sigma_1, \mathbf{u}_1, \mathbf{v}_1)$

Hence I chose this method as we can find the top k eigen values and eigen vectors by applying power iteration and matrix deflation k times. which results in largest k eien values and forms new A_k matrix. (Matrix deflation: Subtract this triplet's information from the matrix, creating a new, "deflated" matrix.)
($A_{new} = A - A_1$
$A_1 = \sigma_1 u_1 v_1^\top$)

Error Analysis

for error analysis I used Frobenuis norm.
For an $m \times n$ matrix A, with elements a_{ij} the Frobenius norm is defined as the square root of the sum of the absolute squares of all its elements:

$$\|A\|_F = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{n} |a_{ij}|^2}$$

we calculate the Forbenius norm of the actual image matrix($\|A\|_F$) and the matrices generated for different k values($\|A_k\|_F$) then error is:

$$error = \|A - A_k\|_F$$
$$\text{Relative error} = (\|A - A_k\|_F)/\|A\|_F$$
$$\text{Percentage error} = \text{Relative error} * 100$$

k values and Forbenius Norm	error
10, 157594.4621	15060.9479
20, 157954.7769	10636.4985
30, 158081.8578	8542.4176
50, 158191.4815	6188.8484

TABLE 3
Observations and errors for globe.jpg

k values and Forbenius Norm	error
5, 21293.2517	4715.3304
10, 21564.3793	3257.9831
20, 21705.0692	2127.6465
40,21777.9139	1165.9213

TABLE 3
OBSERVATIONS AND ERRORS FOR EINSTEIN.JPG

k values and Forbenius Norm	error
5, 193238.0008	11156.6352
10, 193426.0230	7195.0763
20, 193522.3057	3809.5483
50, 193556.3033	1163.1619

TABLE 3
OBSERVATIONS AND ERRORS FOR GREYSCALE.PNG

TRADE-OFFS AND REFLECTIONS ON IMPLEMENTATION CHOICE-POWER ITERATION ALGORITHM WITH MATRIX DEFLATION

Pros:

Simple to Implement:

The code that I used just uses the for loop containing a matrix-vector multiplication so it's easier to implement.

Efficient for this Project:

As our goal is to find different matrices with top k eigen values but not Full SVD, power iteration fits very good. It's fast if k is much smaller than the image dimensions than that of calculating whole SVD.

Cons:

Can have numerical stability issues as errors accumulate during deflation the matrix.

OBSERVATION

1. As the value of k increases image quality increases.
2. As the value of k decreases image compression increases.

APPROXIMATE IMAGES FOR DIFFERENT VALUES OF K

Fig. 3.1. given Globe Image

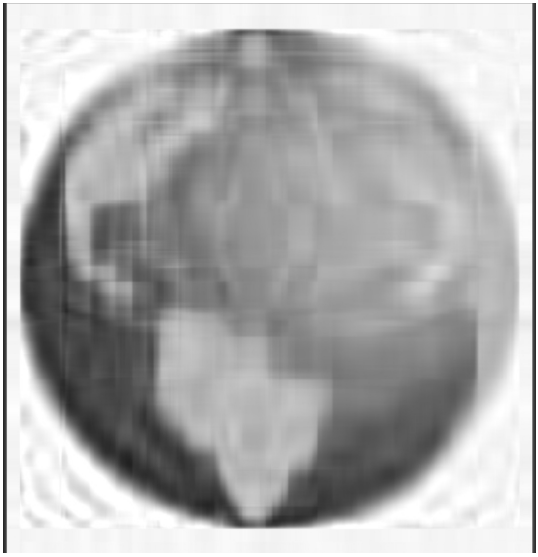

Fig. 3.2. k=10

Fig. 3.3. k=20

Fig. 3.4. k=30

Fig. 3.5. k=50

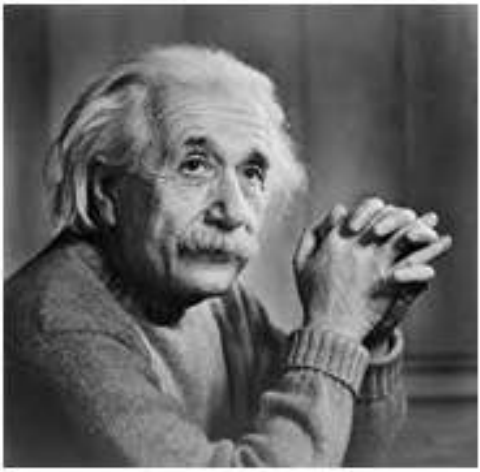

Fig. 3.6. Given Einstein Image

Fig. 3.7. k=5

Fig. 3.8. k=10

Fig. 3.9. k=20

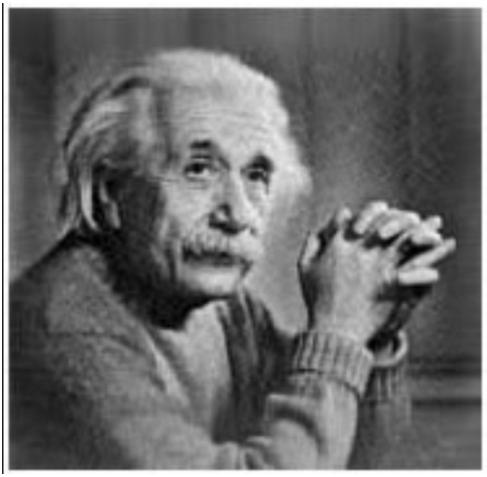

Fig. 3.10. k=40

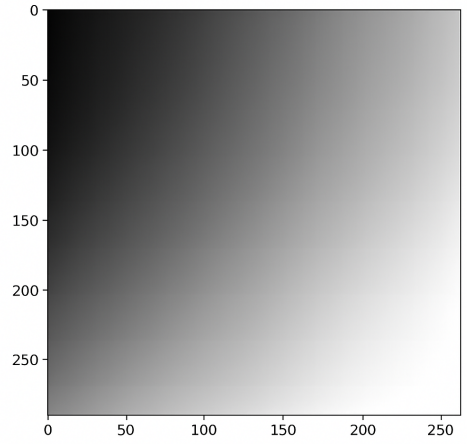

Fig. 3.11. Given Greyscale image

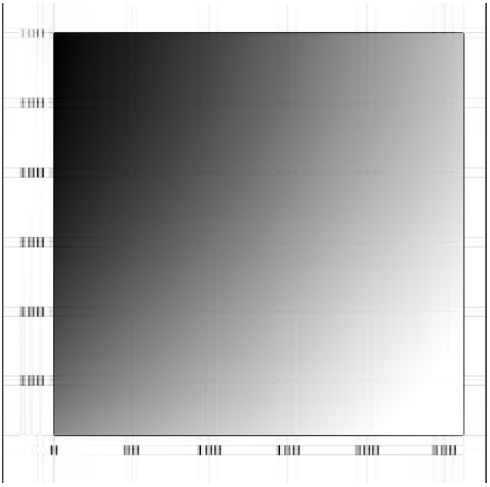

Fig. 3.12. k=5

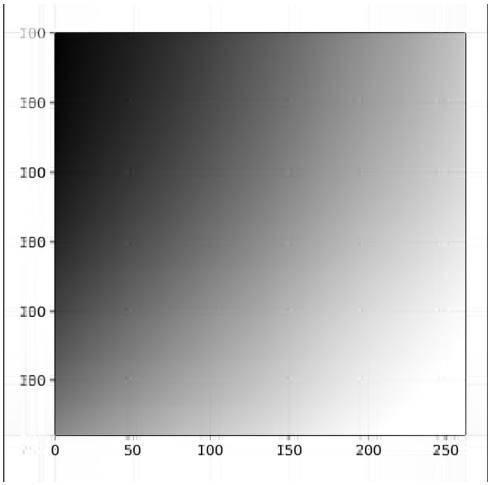

Fig. 3.13. k=10

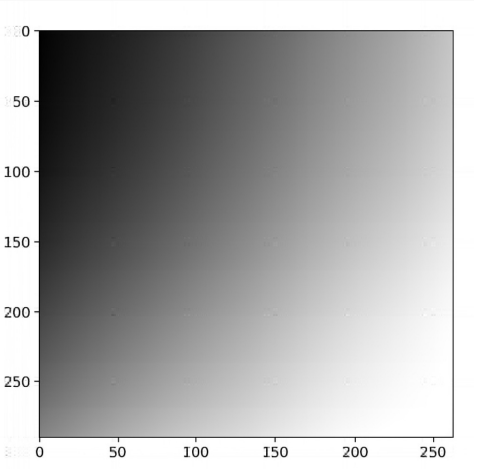

Fig. 3.14. k=20

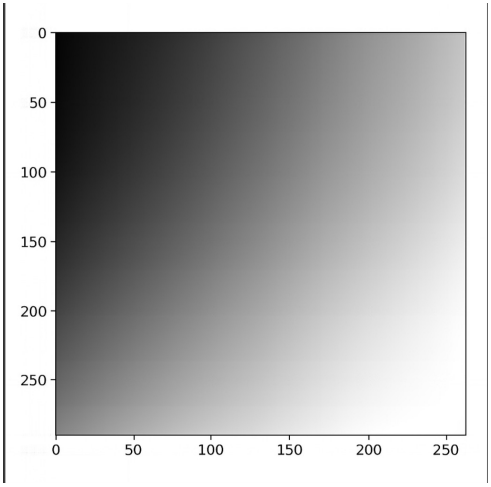

Fig. 3.15. k=50