Low Rank Approximation Methods

December 25, 2024

1 Introduction

Low rank approximation of a matrix is approximation of a matrix by another matrix that is of lower rank than the original matrix. The goal of doing this is to obtain a more compact representation of the data with limited loss of information.

The mathematical description of finding a lower rank matrix approximation can be described as follows: Let A be a $m \times n$, then the low rank approximation (rank k) is given by

$$A_{m\times n} \approx B_{m\times k} C_{k\times n}$$

The main use of such a approximation is in algorithms and operations that utilizes the matrix. Using low rank approximation of the matrix reduces the space needed to store the matrix as well as reducing the computation time for different application.

1. **Reduced Space:** One can see from the above approximation that only k(m + n) entries have to be stored instead of mn entries of the original matrix A.
2. **Reduced Computational Complexity:** Let's consider the problem of multiplying a vector v with a $N \times N$ matrix A, which is $Ax = b$. Various computational complexities for different structures of matrix A are as follows
   - If A is a general matrix, then cost is $O(N^2)$
   - If A is a spare matrix with k elements per row, then cost is $O(kN)$
   - If A is a circulant matrix (so that $A(i,j) = a(i-j)$), then FFT (Fast Fourier Transform) renders the cost to $O(N \log N)$
   - If A is a matrix with rank k (as mentioned in equation), then cost is $O(kN)$

The low rank approximation of a matrix appears in many applications. The list of applications includes image processing [[Fri], [166]], data mining [[Eld05], [Ski07a]], noise reduction, seismic inversion, latent semantic indexing [[Ski07b]], principal component analysis (PCA) [[Jol02], [EP06]], machine-learning [[Lee13], [Mur12], [Ye05]], regularization for ill-posed problems, statistical data analysis applications, DNA microarray data, web search model and so on. The low rank approximation of matrices also plays a very important role in tensor decompositions [[MES12], [LGT13], [Kho11], [Kho14], [Kho12], [KK09], [LDLV00], [OT10]].

The two important parameters for rank approximation are the rank k and error of approximation ϵ.

2 Classical Methods

2.1 Singular Value Decomposition (SVD)

Time Complexity: $O(mn \min(m,n))$

SVD happens to be the most successful method at generating a low-rank approximation of any matrix. Singular Value Decomposition factorizes $A \in \mathbb{R}^{mxn}$ (where $m > n$), into the matrices U, S and V^T, where V^T is the transpose of a matrix V . The SVD factorization does not require square matrices, therefore m, the number of rows, does not have to equal n, the number of columns. The equation is:

$$A_{mxn} = U_{mxm} S_{mxn} V_{nxn}^T$$

S is a rectangular diagonal matrix with positive values called singular values along the diagonal which are $min(m, n)$ and they are arranged in descending order.

$$S = \begin{bmatrix} \sigma_1 & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & 0 & \dots & 0 \\ 0 & 0 & \sigma_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & 0 \\ 0 & 0 & 0 & \dots & \sigma_n \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & 0 \\ 0 & 0 & 0 & \dots & 0 \end{bmatrix}$$

where $\sigma_1 \geq \sigma_2 \geq \sigma_3 \dots \geq \sigma_n$

2.1.1 Prerequisites: Thin SVD

Since $m > n$, one can represent the SVD of A as

$$A_{m\times n} = U_{m\times n} S_{n\times n} V_{n\times n}^T,$$

where $S_{n\times n} = \operatorname{diag}(\sigma_1, \sigma_2, \dots, \sigma_n)$. This representation is called the thin SVD of A.

2.1.2 Prerequisites: Low-Rank Approximation

Singular value decomposition gives the rank of a matrix. The number of nonzero singular values of A is the rank of the matrix A. Let the rank of A be $r = \min(m, n)$, then its SVD reads

$$A = U_{m\times r} S_{r\times r} V_{r\times n}^T,$$

where

$$A = [u_1\, u_2\, \dots\, u_r] \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_r \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_r^T \end{bmatrix},$$

or equivalently,

$$A = \sum_{i=1}^{r} \sigma_i u_i v_i^T,$$

where $u_1, u_2, \dots, u_r$ are columns of $U_{m\times r}$ and $v_1, v_2, \dots, v_r$ are columns of $V_{n\times r}$. One can see that the matrix A is represented by the sum of outer products of vectors. The matrix approximation by a low-rank matrix is possible using SVD.

2.1.3 Prerequisites: Rank-k Approximation

The rank-k approximation (also called truncated or partial SVD) of A, denoted A_k where $k < r$, is given by zeroing out the $r - k$ trailing singular values of A. Thus,

$$A_k = U_{m\times k} S_{k\times k} V_{k\times n}^T = \sum_{i=1}^{k} \sigma_i u_i v_i^T,$$

where $S_k = \text{diag}(\sigma_1, \sigma_2, \ldots, \sigma_k)$, $U_{m\times k} = [u_1, u_2, \ldots, u_k]$, and

$$V_{k\times n}^T = \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_k^T \end{bmatrix}.$$

Then one can see that

$$A_k = U_{m\times k} U_{k\times m}^T A = \left(\sum_{i=1}^{k} u_i u_i^T \right) A$$

and

$$A_k = A V_{n\times k} V_{k\times n}^T = A \left(\sum_{i=1}^{k} v_i v_i^T \right),$$

i.e., A_k is the projection of A onto the space spanned by the top k singular vectors of A.

Algorithm 1 Fixed Rank-k SVD Approximation of A

```
Require: Matrix $A \in \mathbb{R}^{m\times n}$, target rank $k$
Ensure: $U_k \in \mathbb{R}^{m\times k}$, $\Sigma_k \in \mathbb{R}^{k\times k}$, $V_k^T \in \mathbb{R}^{k\times n}$
1: Compute the $k$ largest eigenvalues of $A^T A$ by evaluating $|A^T A - \lambda I| = 0$
2: Create diagonal matrix $\Sigma_k$ using square roots of $k$ largest eigenvalues: $\sigma_i = \sqrt{\lambda_i}$ for $i = 1, \ldots, k$
3: Compute unit eigenvectors for $k$ largest eigenvalues by solving $(A^T A)v = \lambda v$
4: Form $V_k^T$ using these $k$ eigenvectors as rows
5: Compute columns of $U_k$ as $u_i = \frac{Av_i}{\sigma_i}$ for $i = 1, \ldots, k$
6: $A \approx A_k = U_k \Sigma_k V_k^T$ where $\tilde{A}$ denotes low rank approximation of $A$
```

Illustrated above is the procedure to successfully decompose a matrix using SVD. It can be noted that using $A^T A$ yields the right eigenvectors V^T while using AA^T yields the left eigenvectors U. However, both methods work as the singular values obtained are the same. The additional eigenvalues due to any of the methods in regards to the other are only 0.

2.2 Rank Revealing QR Decomposition [201]

Time Complexity: $O(mnk + k^2(m + n))$

2.2.1 Prerequisites: QR Decomposition

QR decomposition is a method in linear algebra used to factor a matrix into the product of two matrices:

$$A = QR$$

where:

- A is an $m \times n$ matrix.
- Q is an orthogonal matrix (i.e., $Q^T Q = I$), where Q is of size $m \times m$.
- R is an upper triangular matrix of size $m \times n$.

The QR decomposition is useful in solving linear systems, eigenvalue problems, and least squares problems. It is often used in numerical analysis due to its stability.

QR decomposition can be computed using methods such as Gram-Schmidt process, Householder Reflections, Givens Rotations.

Gram-Schmidt Projections : Given a set of linearly independent vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$, the process constructs an orthonormal set of vectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n$.

The steps for the Gram-Schmidt process are as follows:

1. Set $\mathbf{q}_1 = \frac{\mathbf{v}_1}{\|\mathbf{v}_1\|}$

2. For each subsequent $\mathbf{v}_k$, subtract the projection of $\mathbf{v}_k$ onto the previously computed $\mathbf{q}_i$'s:

$$\mathbf{q}_k = \frac{\mathbf{v}_k - \sum_{i=1}^{k-1} \operatorname{proj}_{\mathbf{q}_i} \mathbf{v}_k}{\|\mathbf{v}_k - \sum_{i=1}^{k-1} \operatorname{proj}_{\mathbf{q}_i} \mathbf{v}_k\|}$$

where $\operatorname{proj}_{\mathbf{q}_i} \mathbf{v}_k = \left(\frac{\mathbf{v}_k \cdot \mathbf{q}_i}{\mathbf{q}_i \cdot \mathbf{q}_i}\right) \mathbf{q}_i$.

The resulting vectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_n$ form the orthogonal matrix Q, and the coefficients used in the projection form the upper triangular matrix R.

Example

Consider the matrix A:

$$A = \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{bmatrix}$$

We will compute its QR decomposition using the Gram-Schmidt process.

Step 1: Compute $\mathbf{q}_1$

First, we take the first column of A, which is $\mathbf{v}_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$. To make it a unit vector, we compute its norm:

$$\|\mathbf{v}_1\| = \sqrt{1^2 + 2^2 + 3^2} = \sqrt{14}$$

Thus, the first orthonormal vector is:

$$\mathbf{q}_1 = \frac{1}{\sqrt{14}} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Step 2: Compute $\mathbf{q}_2$

Next, we take the second column of A, which is $\mathbf{v}_2 = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$, and subtract the projection of $\mathbf{v}_2$ onto $\mathbf{q}_1$. The projection is:

$$\operatorname{proj}_{\mathbf{q}_1} \mathbf{v}_2 = \left(\frac{\mathbf{v}_2 \cdot \mathbf{q}_1}{\mathbf{q}_1 \cdot \mathbf{q}_1}\right) \mathbf{q}_1 = \left(\frac{2+6+12}{14}\right) \mathbf{q}_1 = \frac{20}{14} \mathbf{q}_1$$

Now subtract this projection from $\mathbf{v}_2$:

$$\mathbf{v}_2 - \operatorname{proj}_{\mathbf{q}_1} \mathbf{v}_2 = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} - \frac{20}{14} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} - \begin{bmatrix} \frac{20}{14} \\ \frac{40}{14} \\ \frac{60}{14} \end{bmatrix}$$

$$= \begin{bmatrix} \frac{8}{14} \\ \frac{-8}{14} \\ \frac{-4}{14} \end{bmatrix}$$

Normalize this vector to get $\mathbf{q}_2$:

$$\|\mathbf{v}_2 - \operatorname{proj}_{\mathbf{q}_1} \mathbf{v}_2\| = \sqrt{\left(\frac{8}{14}\right)^2 + \left(\frac{-8}{14}\right)^2 + \left(\frac{-4}{14}\right)^2} = \sqrt{\frac{128}{196}} = \frac{8}{14}$$

Thus, $\mathbf{q}_2 = \frac{1}{\frac{8}{14}} \begin{bmatrix} \frac{8}{14} \\ \frac{-8}{14} \\ \frac{-4}{14} \end{bmatrix} = \begin{bmatrix} \frac{1}{\sqrt{14}} \\ \frac{-1}{\sqrt{14}} \\ \frac{-0.5}{\sqrt{14}} \end{bmatrix}$.

Step 3: Construct Q and R

The matrix Q is formed by stacking $\mathbf{q}_1$ and $\mathbf{q}_2$ as columns:

$$Q = \begin{bmatrix} \frac{1}{\sqrt{14}} & \frac{1}{\sqrt{14}} \\ \frac{2}{\sqrt{14}} & \frac{-1}{\sqrt{14}} \\ \frac{3}{\sqrt{14}} & \frac{-0.5}{\sqrt{14}} \end{bmatrix}$$

The matrix R is upper triangular and can be found by computing the dot products of the columns of A with the columns of Q:

$$R = Q^T A$$

$$R = \begin{bmatrix} \frac{1}{\sqrt{14}} & \frac{2}{\sqrt{14}} & \frac{3}{\sqrt{14}} \\ \frac{1}{\sqrt{14}} & \frac{-1}{\sqrt{14}} & \frac{-0.5}{\sqrt{14}} \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 2 & 3 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} \sqrt{14} & \frac{20}{\sqrt{14}} \\ 0 & \sqrt{2} \end{bmatrix}$$

Thus, the QR decomposition of A is:

$$A = QR = \begin{bmatrix} \frac{1}{\sqrt{14}} & \frac{1}{\sqrt{14}} \\ \frac{2}{\sqrt{14}} & \frac{-1}{\sqrt{14}} \\ \frac{3}{\sqrt{14}} & \frac{-0.5}{\sqrt{14}} \end{bmatrix} \begin{bmatrix} \sqrt{14} & \frac{20}{\sqrt{14}} \\ 0 & \sqrt{2} \end{bmatrix}$$

2.2.2 Prerequisites: Inverse Iterations To Find Lowest Singular value

Suppose $A \in \mathbb{R}^{m \times n}$ be a matrix. To find its smallest singular value $\sigma_{\min}$ and corresponding singular vectors, we can use inverse iterations:

Algorithm 2 Inverse Iterations to find smallest singular value

```
Start: With a random unit vector $v_0 \in \mathbb{R}^n$
1: For $k = 0, 1, 2, \ldots$ until convergence:
       1. Solve $(A^T A) w_{k+1} = v_k$ for $w_{k+1}$
       2. Set $v_{k+1} = \frac{w_{k+1}}{\|w_{k+1}\|}$
2: The sequence $v_k$ converges to the right singular vector corresponding to $\sigma_{\min}$
3: The smallest singular value can be computed as:
       $\sigma_{\min} = \frac{\|Av_k\|}{\|v_k\|}$
4: return U, Σ, V*
```

The inverse iteration method works because:

$$(A^T A)^{-1} = \sum_{i=1}^{r} \frac{1}{\sigma_i^2} v_i v_i^T$$

where $\{v_i\}$ are the right singular vectors and $\{\sigma_i\}$ are the singular values. The largest eigenvalue of $(A^T A)^{-1}$ corresponds to $\frac{1}{\sigma_{\min}^2}$, making inverse iteration converge to the corresponding eigenvector.

Instead of explicitly forming $A^T A$ and inverting it, we typically solve the system using methods like QR decomposition or conjugate gradient to avoid numerical instability.

The convergence rate is geometric, proportional to:

$$\left| \frac{\sigma_{\min}^2}{\sigma_{\min+1}^2} \right|$$

where $\sigma_{\min+1}$ is the second smallest singular value. Proof for the same can be found here.

2.2.3 Prerequisites: Pivoted QR Decomposition

Suppose that A is close to being p rank deficient. Let's notice how this would be reflected in QR Decomposition of the matrix.

$$A = QR$$

$$Q^T A = R$$

$$\begin{bmatrix} — & q_1 & — \\ — & q_2 & — \\ & \vdots & \\ — & q_n & — \end{bmatrix} \begin{bmatrix} | & | & & | \\ a_1 & a_2 & \dots & a_n \\ | & | & & | \end{bmatrix} = \begin{bmatrix} \langle q_1, a_1\rangle & \langle q_1, a_2\rangle & \dots & \langle q_1, a_n\rangle \\ 0 & \langle q_2, a_2\rangle & \dots & \langle q_2, a_n\rangle \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \langle q_n, a_n\rangle \end{bmatrix} = R$$

If some column a_i dot product with q_i is small, this implies that a_i nearly lies in the subspace spanned by orthonormal vectors of columns before it. Thus this particular column can be nearly spanned by columns before it in the matrix. Hence one singular value is very small and matrix is one rank lower. If somehow this column can be brought as the last column of R matrix, last row and column of R can be discarded thus creating a low rank approximation. A corresponding important theorem is detailed below.
Suppose that A is close to being rank one deficient. Our goal is to identify a column permutation of A such that the resulting QR factorization has a small pivotal element r_n. It turns out that this permutation can be determined by examining the magnitudes of the components of the singular vector of A that corresponds to its smallest singular value.

Theorem:
Suppose a vector $x \in \mathbb{R}^n$ with $||x||_2 = 1$ such that $||Ax||_2 = \epsilon$ and let Π be a permutation such that if $\Pi^T x = y$, then $|y|_n = ||y||_\infty$. Then if $A\Pi = QR$ is the QR factorisation of $A\Pi$,

$$|r_{nn}| \leq \sqrt{n}\epsilon$$

Proof:
Firstly, note that since $|y_n| = ||y||_\infty$ and $||y||_2 = ||x||_2$, then $|y_n| \geq \sqrt{\frac{1}{n}}$. Now note

$$Q^T Ax = Q^T A\Pi\Pi^T x = Ry = \begin{pmatrix} \vdots \\ r_{nn} y_n \end{pmatrix}$$

$\implies$

$$\epsilon = ||Ax||_2 = ||Q^T Ax||_2 = ||Ry||_2 \geq |r_{nn} y_n|$$

From this, we obtain the result. It can be inferred that if we have a matrix A that is nearly rank-one deficient, we can find a permutation matrix Π such that the QR factorization of $A\Pi$ has a small pivot element. This pivot will be at least as small as the smallest singular value of A.In essence, the proof demonstrates that by carefully choosing a permutation Π based on the singular vectors of A, we can obtain a QR factorization with a small pivot element. This is useful in numerical computations where small pivots can lead to numerical instability.

Let $v \in \mathbb{R}^n$ with $||v_n|| = 1$ be the singular value of A corresponding to the lowest singular value σ_n. Then

$$||Av||_2 = \sigma_n$$

On applying theorem here,

$$|(\Pi^T v)|_n = ||v||_\infty$$

then $A\Pi$ has a QR-factorisation with a pivot r_{nn} at least as small as $\sqrt{n}\sigma_n$ in absolute value.
Since only the permutation is needed, it is not necessary to compute SVD of A to obtain v. A few inverse iterations can approximate v from which Π can be computed.

2.2.4 Algorithm

Now assume A is rank $n-k$ deficient, which is A is of rank k. We aim to find a permutation such that

$$A\Pi = QR \equiv Q \begin{pmatrix} R_{11} & R_{12} \\ 0 & R_{22} \end{pmatrix}$$

with $R_{22} \in \mathbb{R}^{n-k \times n-k}$ and $||R_{22}||$ is small in some norm. Using this a low rank approximation for A can be developed as follows:

$$A \approx Q_k R_{11} P^T$$

where Q_k is the $m \times k$ matrix formed by taking the first k columns of Q. This matrix product $Q_k R_{11} P^T$ provides a rank-k approximation to A.

Appropriate QR can be found by extending the one-dimensional approach repeatedly to R_{11}, the leading principal triangular part of R. Suppose, we isolated an $n-k$ by $n-k$ block of R_{22}. Now to compute an $(n-k+1)$ by $(n-k+1)$ block, we apply the one-dimensional algorithm defined before, P, such that $R_{11}P = Q_1 \tilde{R}_{11}$ is the QR-factorisation and the (k,k) element of $\tilde{R}_{11}$ is small. Then

$$\tilde{\Pi} \equiv \Pi \begin{pmatrix} P & 0 \\ 0 & I \end{pmatrix}$$

$$\tilde{Q} \equiv Q \begin{pmatrix} Q_1 & 0 \\ 0 & I \end{pmatrix}$$

it can be verified that

$$A\tilde{\Pi} = \tilde{Q} \begin{pmatrix} \tilde{R}_{11} & Q_1^T R_{12} \\ 0 & R_{22} \end{pmatrix}$$

is the QR-factorisation of $A\tilde{\Pi}$.

Algorithm 3 RRQR

```
Start: To compute a permutation as above, initialize $W \in \mathbb{R}^{n \times n-k}$ to zero.
1: for $i = n, n-1, \ldots, k+1$ do
2:     $R_{11} \leftarrow$ leading $i \times i$ block of $R$.
3:     Compute the singular vector $v \in \mathbb{R}^i$ of $R_{11}$ corresponding to $\sigma_{\min}(R_{11})$ with $\|v\|_2 = 1$, and set
       $\delta_i = \sigma_{\min}(R_{11})$. This is done using inverse iterations.
4:     Compute a permutation $P \in \mathbb{R}^{i \times i}$ such that $|(P^T v)_i| = \|P^T v\|_\infty$.
5:     Assign $\tilde{v} \equiv \begin{pmatrix} v \\ 0 \end{pmatrix} \in R^n$ to the $i$-th column of $W$.
6:     Compute $W \leftarrow \tilde{P}^T W$, where $P \equiv \begin{pmatrix} P & 0 \\ 0 & I \end{pmatrix}$.
7:     Compute the $QR$-factorization : $R_{11}P = Q_1 \tilde{R}_{11}$
8:     $\Pi \leftarrow \Pi \tilde{P}$
9:     $Q \leftarrow Q \begin{pmatrix} Q_1 & 0 \\ 0 & I \end{pmatrix}$
10:    $R \leftarrow \begin{pmatrix} \tilde{R}_{11} & Q_1^T R_{12} \\ 0 & R_{22} \end{pmatrix}$
11: end for
12: $A \approx \tilde{A} = Q_k R_{11} P^T$ where $\tilde{A}$ denotes low rank approximation of $A$
```

The matrix W is used to store the singular vectors corresponding to the smallest singular values of the successive leading triangular blocks R_11.

2.3 Randomized SVD [202]

Time Complexity: $O(mnk + k^2(m+n))$

2.3.1 Prerequisites: Randomized Numerical Linear Algebra

A growing area of research, randomized numerical linear algebra (RandNLA), combines probability theory with numerical linear algebra to develop fast, randomized algorithms with theoretical guarantees that such algorithms are, in some sense, a good approximation of some full computation ([Mah16]) ([DM16]). The main insight is that randomness is an algorithmic resource creating efficient, unbiased approximations of nonrandom operations.

A simple example of this can be thought of as sampling some columns or rows from a matrix based on a probability distribution such that most information of the matrix is preserved.

2.3.2 Prerequisites: Projection Matrix

The question asked here is how do we project the subspace formed by the range(A) of a matrix A to subspace formed by another range(Q) of a matrix Q. To build this result let's consider projection of vector a to another vector q. This is simply

$$v = (\frac{q^T x}{q^T q})q$$

Now we can represent the range(Q) as the subspace spanned by the columns of the matrix

$$Q = \begin{bmatrix} | & | & & | \\ q_1 & q_2 & \dots & q_n \\ | & | & & | \end{bmatrix}$$

Suppose we want to project a vector to the subspace formed by these vectors

$$v = (\frac{q_1^T x}{q_1^T q_1})q_1 a + (\frac{q_2^T x}{q_2^T q_2})q_2 a + \dots + (\frac{q_m^T x}{q_m^T q_m})q_m a$$

For simplicity assume that Q is a orthonormal matrix, thus $q_i^T q_i = 1$ for all $i = 1, 2..n$.

$$v = (q_1^T x)q_1 a + (q_2^T x)q_2 a + \dots + (q_m^T x)q_m a$$

$$v = \begin{bmatrix} | & | & & | \\ q_1 & q_2 & \dots & q_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} — & q_1 & — \\ — & q_2 & — \\ & \vdots & \\ — & q_m & — \end{bmatrix} \begin{bmatrix} | \\ a \\ | \end{bmatrix}$$

$$v = \begin{bmatrix} | & | & & | \\ q_1 & q_2 & \dots & q_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} | & | & & | \\ q_1 & q_2 & \cdots & q_m \\ | & | & & | \end{bmatrix}^T \begin{bmatrix} | \\ a \\ | \end{bmatrix} = QQ^T a$$

We can represent matrix A in terms of its column vectors. These column vectors span the range(A), thus projecting a matrix A to matrix Q is equivalent to projecting each column vector to span of matrix$Q$. Let this new projected matrix be P.

$$P = \begin{bmatrix} | & | & & | \\ q_1 & q_2 & \dots & q_n \\ | & | & & | \end{bmatrix} \begin{bmatrix} | & | & & | \\ q_1 & q_2 & \cdots & q_m \\ | & | & & | \end{bmatrix}^T \begin{bmatrix} | & | & & | \\ a_1 & a_2 & \dots & a_n \\ | & | & & | \end{bmatrix} = QQ^T A$$

2.3.3 Algorithm

Consider the general problem of low-rank matrix approximation. Given an $m \times n$ matrix $\mathbf{A}$, we want $m \times k$ and $k \times n$ matrices $\mathbf{B}$ and $\mathbf{C}$ such that $k \ll n$ and $\mathbf{A} \approx \mathbf{BC}$. To approximate this computation using randomized singular value decomposition, it is broken down into a two-stage computation:

- **Step 1:** Compute an approximate basis for the range of $\mathbf{A}$. We want a matrix $\mathbf{Q}$ with ℓ orthonormal columns ($k \leq \ell \leq n$) that captures the action of $\mathbf{A}$. Formally, $\mathbf{A} \approx \mathbf{QQ}^*\mathbf{A}$.
- **Step 2:** Given such a matrix $\mathbf{Q}$—which is much smaller than $\mathbf{A}$ it is used to compute SVD of $\mathbf{A}$.

Algorithm for step 2 is designed first as shown. Imagine we had access to $\mathbf{Q}$ then first matrix $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$ is computed, $\mathbf{B}$ will be of size $l \times n$ since $\mathbf{Q}*$ $l \times m$ is of size and $\mathbf{Q}$ is of size $m \times l$. The next step of computing SVD for this will be much faster since it is of less size. Finally $\mathbf{U}$ is set to $\mathbf{Q}\tilde{\mathbf{U}}$, thus returning SVD for our original matrix. The efficiency of this algorithm comes from $\mathbf{B}$ being small relative to $\mathbf{A}$. Since $\mathbf{A} \approx \mathbf{QQ}^*\mathbf{A} = \mathbf{Q}(\tilde{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^*)$, setting $\mathbf{U} = \mathbf{Q}\tilde{\mathbf{U}}$ produces a low-rank approximation, $\mathbf{A} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$.

Algorithm 4 Randomized SVD

```
Given An orthonormal matrix $\mathbf{Q}$ s.t. $\mathbf{A} \approx \mathbf{QQ}^*\mathbf{A}$
1: Form $\mathbf{B} = \mathbf{Q}^*\mathbf{A}$
2: Compute the SVD of $\mathbf{B}$, i.e. $\mathbf{B} = \tilde{\mathbf{U}}\mathbf{\Sigma}\mathbf{V}^*$
3: Set $\mathbf{U} = \mathbf{Q}\tilde{\mathbf{U}}$
4: return $\mathbf{U}$, $\mathbf{\Sigma}$, $\mathbf{V}^*$
```

Note that randomness only occurs in Step 1 and that Step 2 is deterministic given Q. Thus, the algorithmic challenge is to efficiently compute Q through randomized methods. [NH09] present several algorithms to do this, called randomized range finders.

The goal of a randomized range finder is to produce an orthonormal matrix $\mathbf{Q}$ with as few columns as possible such that

$$\|(\mathbf{I} - \mathbf{QQ}^*)\mathbf{A}\| \leq \varepsilon \tag{1}$$

for some desired tolerance ε.

[NH09] solution is motivated by Johnson and Lindenstrauss's seminal paper here that showed that pairwise distances among a set of points in a Euclidean space are roughly maintained when projected into a lower-dimensional Euclidean space.

Algorithm for step 1 involved drawing ℓ Gaussian random vectors, $\boldsymbol{\omega}^{(1)}, \boldsymbol{\omega}^{(2)}, \ldots, \boldsymbol{\omega}^{(\ell)}$ and project them using the linear map $\mathbf{A}$. Intuitively, we are randomly sampling the range of $\mathbf{A}$. And finding an orthonormal basis for these vectors gives us our desired $\mathbf{Q}$. This scheme is formally described in Algorithm 2.

Algorithm 5 Randomized range finder

```
Given Let $\ell$ be a sampling parameter indicating the number of Gaussian random vectors to draw for $\mathbf{\Omega}$.
1: Draw an $n \times \ell$ Gaussian random matrix $\mathbf{\Omega}$
2: Generate an $m \times \ell$ matrix $\mathbf{Y} = \mathbf{A\Omega}$
3: Generate an orthonormal matrix $\mathbf{Q}$, e.g. using QR factorization $\mathbf{Y} = \mathbf{QR}$.
```

The above described algorithm works when we have a fixed rank k to approx the matrix by. On the other hand if problem of fixed error is considered, a modified version of the algorithm can be implemented as follows. In this l is automatically selected based on the acceptable error or tolerance ϵ. The main idea is that the orthonormal basis Q can be constructed incrementally, and can be stopped when the error is acceptable. Let $\mathbf{Q}^(\mathbf{i})$ denote an iteration of the $\mathbf{Q}$ matrix with i columns. The the empty basis matrix is $\mathbf{Q}^(\mathbf{0})$. Using these facts the appropriate matrix $\mathbf{Q}$ is incrementally generated.

Algorithm 6 Iterative construction of Q

```
1: Draw an $n \times 1$ Gaussian random vector, $\boldsymbol{\omega}^{(i)}$, and set $\mathbf{y}^{(i)} = A\boldsymbol{\omega}^{(i)}$
2: Construct $\tilde{\mathbf{q}} = (\mathbf{I} - \mathbf{Q}^{(i-1)}(\mathbf{Q}^{(i-1)})^*)\mathbf{y}^{(i)}$
3: Set $\mathbf{q}^{(i)} = \tilde{\mathbf{q}}/\|\tilde{\mathbf{q}}\|$
4: Form $\mathbf{Q}^{(i)} = [\mathbf{Q}^{(i-1)}\mathbf{q}^{(i)}]$
```

One method to improve the approximation is to perform a few power iterations on $(\mathbf{AA^T})$ to drive the spectrum of $\mathbf{A}$ down. The key here is that if the singular values of $\mathbf{A}$ are Σ, the singular values of $(\mathbf{AA^T})^\mathbf{p}\mathbf{A}$ are Σ^{2p+1}, so the spectrum (and our approximation error) decays exponentially with the number of iterations. In practice, one or two iterations is enough (whereas $p = 0$ is generally not sufficient for noisy data).

2.4 Adaptive Cross Approximation

Time Complexity: $O(mk + nk + k^3)$
A rough intuition and algorithm for adaptive cross approximation is provided since proof of error bounding and low rank require vast math.

2.4.1 Algorithm

The basic idea of ACA is to approximate a matrix with a rank 1 outer product of one row and one column of that same matrix. And then iteratively use this process to construct an approximation of arbitrary precision. Ideally, at each step, we will choose the best fitting row and column. After the first iteration, we are no longer trying to approximate the original matrix, but instead we approximate the residual matrix formed by the difference between the original matrix and the current approximation matrix.

Given a matrix $M \in \mathbb{R}^{n\times m}$, we'll construct an approximation like $\sum_k u_k v_k^T$. The task will be to construct the row and column vectors u_k and v_k on each iteration such that the Frobenius norm error eventually converges. To do that, the key will be to iteratively form rank-1 approximations to the residual matrix. The residual matrix is the matrix forming the difference between M and the current approximation and can be written as $R_{ij} = M_{ij} - \sum_k u_k v_k$ representing the entry-wise difference between the target matrix and the current approximation. The goal will be to have R satisfy $\|R\|_2 < \epsilon$ where ϵ is a user-specified accuracy parameter. To do this, during each iteration:

Algorithm 7 Adaptive Cross Approximation

```
1. Determine a "pivot" $(i^*, j^*)$ as the indices that maximize $R_{ij}$. Intuitively, the largest rows and columns are going to be most "important".
2. Assign a new rank-1 component of the approximation:
       $u_{ki} = R_{i^*,j} / R_{i^*,j^*}$
       $v_{ki} = R_{i,j^*}$
3. Update $R$ to account for the new rank-1 update to the approximation.
4. If the magnitude of the $u_k v_k$ update is small enough, stop. Otherwise, return to step 1.
```

3 Results

Observe the error by varying the spectral decay factor of the matrix of size 800 x 800. We also see the effect of power iterations here as below. The metric of evaluation is the Frobenius norm.

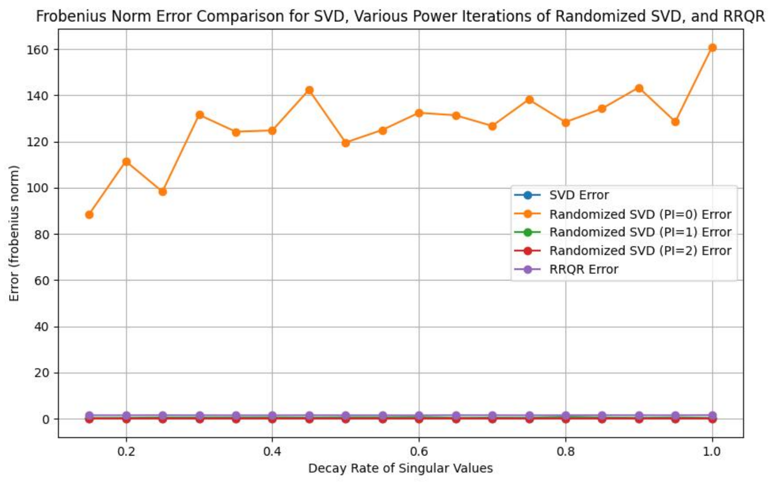

Figure 1: No PI included

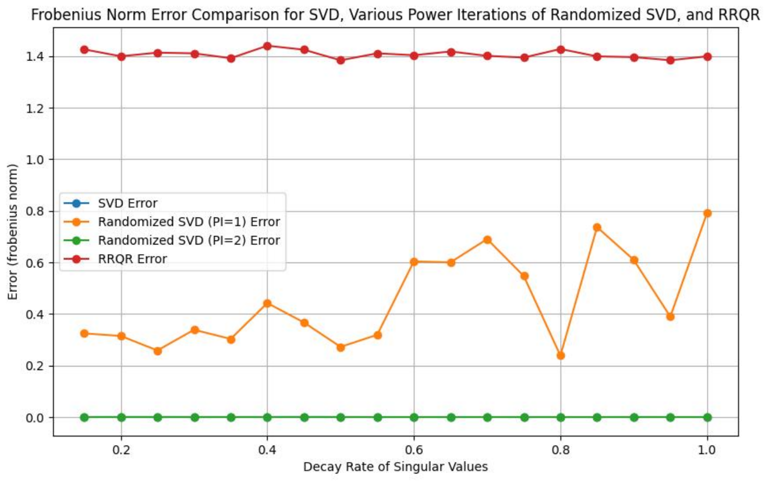

Figure 2: PI included

Observe how the use of power iterations reduces error in the randomised SVD below even RRQR! Also, increasing the decay rate generally increases error. Also find the computation time curves here.

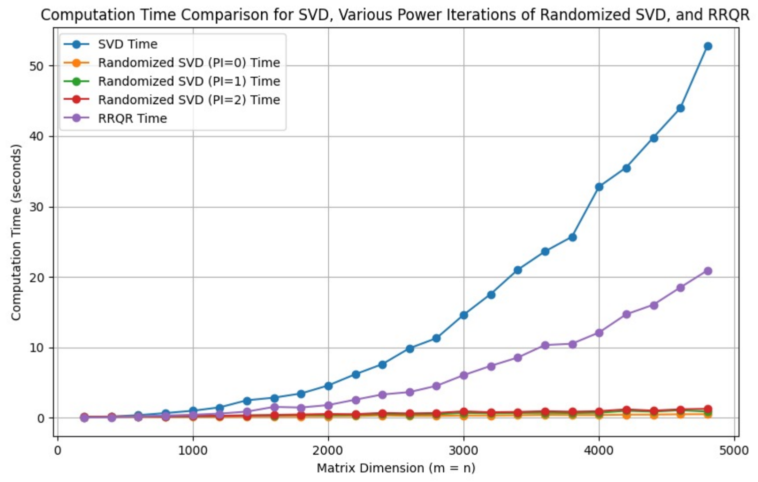

Figure 3: Computational Times

Notice how the time exponentially increases for SVD, while not so much for the others.

4 Conclusion

References

[166] http://perception.csl.illinois.edu/matrix-rank/home.html.

[201] https://apps.dtic.mil/sti/tr/pdf/ada156756.pdf.

[202] https://gregorygundersen.com/blog/2019/01/17/randomized-svd/mahoney2016lecture.

[DM16] Petros Drineas and Michael W. Mahoney. Randnla: randomized numerical linear algebra. *Commun. ACM*, 59(6):80–90, May 2016.

[Eld05] Lars Eldén. "numerical linear algebra and applications in data mining preliminary version december 20, 2005.". 2005.

[EP06] Lars Elden and Haesun Park. "matrix rank reduction for data analysis and feature extraction". 2006.

[Fri] Shmuel Friedland. "fast low rank approximations of matrices and tensors.".

[Jol02] Ian T Jolliffe. "principal component analysis for special types of data.". 2002.

[Kho11] B. N. Khoromskij. "o(d logn)quantics approximation of n d tensors in high dimensional numerical meodelling". 2011.

[Kho12] B. N. Khoromskij. "tensor structured numerical methods in scientific computing: Survey on recent advances". 2012.

[Kho14] B. N. Khoromskij. "tensor numerical methods for higher dimensional partial differential equations: Basic theory and intial applications". 2014.

[KK09] B. N. Khoromskij and V. Khoromskaia. "multigrid accerelated tensor approximation of function related multidimensional arrays". 2009.

[LDLV00] B. De. Moor L. D. Lathauver and J. Vandewalle. " a multilinear singular value decomposition". 2000.

[Lee13] Joonseok Lee. "matrix approximation under local low-rank assumption.". 2013.

[LGT13] D. Kressner L. Grasedyck and C. Tobler. "a literature survey of low rank tensor approximation techniques". 2013.

[Mah16] Michael W. Mahoney. Lecture notes on randomized linear algebra, 2016.

[MES12] N. Kishore Kumar M. Espig and J. Schneider. "a note on tensor chain approximation, computing and visualization in science". 2012.

[Mur12] K. P. Murphy. "machine learning: A probabilistic perspective". 2012.

[NH09] Joel A. Tropp Nathan Halko, Per-Gunnar Martinsson. "finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions". 2009.

[OT10] I. V. Oseledets and E. E. Tyrtyshnikov. "tt-cross approximation for multidimesnional arrays". 2010.

[Ski07a] D. Skillicorn. "understanding complex datasets, data mining with matrix decompositions". 2007.

[Ski07b] David Skillicorn. "understanding complex datasets: data mining with matrix decompositions". 2007.

[Ye05] J. Ye. "generalized low rank approximation of matrices". 2005.