Comparison of Fast Low-rank Approximation and Approximate SVD Methods

Roman Vakhrushev
Department of Computer Science
New York University, Tandon School of Engineering, NY
`rv1057@nyu.edu`

Abstract

Low Rank Approximation and Singular Value Decomposition are powerful tools used in many applications. However, computing them via traditional methods is expensive. In this paper we analyze 4 different fast approximation algorithms that can be used for that purpose.

1 Introduction

The Singular Value Decomposition (SVD) is a factorization technique that decomposes a matrix into three components: U, Σ, and V. Given a matrix $A \in \mathbb{R}^{m \times n}$, the SVD of A is written as $A = U\Sigma V^T$, where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are orthogonal matrices, and $\Sigma \in \mathbb{R}^{m \times n}$ is a diagonal matrix containing the singular values of A.

A low-rank approximation of a matrix A can be obtained by retaining only the top k singular values and the corresponding singular vectors. This is known as a rank-k approximation, where $k \ll \min(m, n)$. The resulting approximation is given by $A_k = U_k \Sigma_k V_k^T$, where $U_k \in \mathbb{R}^{m \times k}$, $\Sigma_k \in \mathbb{R}^{k \times k}$, and $V_k \in \mathbb{R}^{n \times k}$ are the truncated versions of U, Σ, and V, respectively. Depending on the application, we may decide to store U_k and $\Sigma_k V^T$ (as a product). The matrices would have dimensions $m \times k$ and $k \times m$ and their product would produce a valid low-rank approximation.

Moreover, the rank-k approximation A_k is the best possible approximation of A in the Frobenius norm and Spectral norm, and is given by the Eckart-Young theorem Eckart and Young [1936].

2 Problem Statement

Although the SVD produces the best rank-k approximation, its naive computation is prohibitively expensive. A straightforward approach would be to compute the full SVD and then truncate it at k to obtain U_k, Σ_k, and V_k^T. As previously discussed, this method yields a valid low-rank approximation. However, the time complexity of computing the SVD is typically $O(m \times n^2)$ (assuming $n \leq m$ throughout this paper), which means the running time is independent of k. Consequently, this method becomes impractical for large matrices.

A more viable idea is to compute A_k approximately, satisfying conditions such as $||A - \hat{A}|| \leq (1 + \epsilon)||A - A_k||$ (where the norm is usually Frobenius or spectral), among others. Fortunately, there exist approximation algorithms with improved time complexities, including non-iterative methods that run in $O(mnk)$ time and iterative methods that run in $O(mnkt)$ time (where t denotes the number of iterations). The primary objective of this paper is to investigate some of these methods and provide a comprehensive comparison, both theoretical and empirical, of their performance.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

3 SVD-based Approximation Algorithms

In this section we discuss the algorithms we would study and compare in the later sections. See Appendix A for the pseudocode of the described methods. All of the algorithms were implemented in Python from scratch.

3.1 rSVD

Randomized Singular Value Decomposition (rSVD) is a low-rank approximation technique that has gained significant attention in recent years due to its efficiency and accuracy. The rSVD algorithm, as introduced by Halko et al. [2011], is a randomized version of the traditional SVD algorithm. It approximates a large matrix by projecting it onto a lower-dimensional space using random matrices, and then computes the SVD of the resulting smaller matrix. This approach allows for a significant reduction in computational cost, making it particularly useful for large-scale datasets where traditional SVD algorithms can be computationally prohibitive. Our implementation uses simplified rSVD (without oversampling), which allows for easier comparison with other methods.

3.2 Power Iteration with Deflation

Power Iteration with Deflation is a low-rank approximation technique that builds upon the basic power iteration algorithm. Instead of computing a single largest singular value (and its vectors), we compute all top-k triples one by one. Specifically, the method deflates the original matrix by subtracting the previously computed low-rank approximation from it, effectively removing the already captured dominant subspace and allowing the algorithm to focus on the remaining residual. One of the key strengths of Power Iteration with Deflation is its simplicity and ease of understanding, it can be implemented very easily with basic power iteration as a black-box.

3.3 Block Power Iteration

Block Power Iteration is a low-rank approximation technique that extends both the basic power iteration and the rSVD algorithm to improve its efficiency and accuracy. Specifically, we first draw a random matrix, multiply it by input matrix A. And then iteratively left-multiply the resulting matrix by AA^T, resulting in a matrix block. After each step we orthogonolize the block. Our implementation of Block Power Iteration mostly follows the formulation presented in the work of Musco and Musco [2015]. We also do postprocessing similar to that in the paper, although it is possible to it differently (for example, similar to our implementation of rSVD). For reconstruction, we compute UU^TA.

3.4 Block Krylov Iteration

The implementation for Block Krylov Method is also very similar to Block Power Iteration. While computing blocks for Power Iteration, we basically compute the Krylov subspace, which can be used for more efficient reconstruction. As we will show this results in more space used, but also often requires less iterations. The postprocessing and reconstruction step are also similar to that in our implementation of Block Power Iteration.

4 Theoretical Analysis of the Algorithms

In this section we briefly analyze the runtime and space complexity of the discussed algorithms.

4.1 Runtime

The rSVD algorithm consists of 3 main steps. Computation of product between an input matrix A ($m \times n$) and random matrix Ω ($n \times k$), which has time complexity of $O(mnk)$, then computing QR decomposition of the result ($O(mk^2)$) and SVD of projection into Q ($O(nk^2)$). So, the total runtime is $O(mnk + mk^2 + nk^2)$. Assuming $k \ll m$ and $k \ll n$, we can state the runtime as $O(mnk)$.

The Power Iteration with Deflation for each triple of singular value and vectors computes t iterations, during which we do matrix-vector multiplications ($O(mn)$) and some matrix subtractions ($O(mn)$). Since there are a total of k triples to compute, we can do it in $O(mnkt)$ time. Notice however, that we do a lot of computations in nonparallel way, computing vectors separately. This often results in slower runtime in practice.

The Block Power Iteration consists of two parts: block computation and postprocessing. During block computation we compute the following products: AQ (once), $T = A^T Q$, and AT. The runtimes for all of these are $O(mnk)$. We also compute QR decomposition during each iteration, which is $O(mk^2)$. So, the runtime for this part is $O(mnkt + mtk^2)$. For postprocessing, we compute $Q^T AA^T Q$, which can be done in $O(mnk + mk^2)$ time, an SVD on a $k \times k$ matrix, which is $O(k^3)$, and also a multiplication between $n \times k$ and $k \times k$ matrix, which results in $O(nk^2)$. Assuming $k \ll n$, the total runtime is still $O(mnkt + mtk^2) = O(mnkt)$, meaning it is heavily dominated by the iterative part.

The Block Krylov Iteration consists of the same parts as Block Power Iteration. The runtime for computation of Krylov subspace is essentially the same (the only difference is that we now **store** each block in memory), so it is again $O(mnkt + mtk^2) = O(mnkt)$. The main difference comes in the second part. Here, instead of a small matrix Q ($n \times k$), we now have larger matrix of size $m \times qk$, which represents Orthogonolized Krylov subpace. The runtime for QR is $O(mq^2k^2)$, then we compute $Q^T AA^T Q$ in time $O(mntk + mk^2t^2)$, compute SVD of a $tk \times tk$ matrix in time $O(k^3t^3)$ and the final result is $O(mnkt + mk^2t^2 + k^3t^3)$. Here, we cannot make assumptions about t, since it can be very large theoretically depending on the desired error for approximation. However, in practice, we often select the number of iterations to be very small, so in certain scenarios the runtime can be seen as $O(mnkt)$.

4.2 Number of Iterations for Block Methods

The result by Musco and Musco [2015] also provides us with a number of iterations for block methods. In fact, the bound even does not depend on singular value gaps. For Block Power Iteration, $t = \Theta(\frac{\log n}{\epsilon})$ results in approximation $\hat{A}$ that satisfies $||A - \hat{A}||_F \leq (1 + \epsilon)||A - A_k||_F$ and also $||A - \hat{A}||_2 \leq (1 + \epsilon)||A - A_k||_2$. Similarly, setting $t = \Theta(\frac{\log n}{\sqrt{\epsilon}})$ for Block Krylov Iteration satisfies $\hat{A}$ that satisfies $||A - \hat{A}||_F \leq (1 + \epsilon)||A - A_k||_F$ and also $||A - \hat{A}||_2 \leq (1 + \epsilon)||A - A_k||_2$. We set our number of iterations according to this result in our experiments as well.

With this results, the runtime for Block Power Iteration adn Block Krylov Iteration can be described accordingly as $O(mnk\frac{\log n}{\epsilon})$ and $O(mnk\frac{\log n}{\sqrt{\epsilon}} + mk^2\frac{\log^2 n}{\epsilon} + k^3\frac{\log^3 n}{\epsilon^{1.5}})$

4.3 Space Complexity

Space complexity for rSVD is very small, we only need to store a few matrices of size $O(mk)$ and $O(nk)$, so the space complexity is $O(mk)$.

For our implementation of Power Iteration with Deflation, we need to maintain copy of A and also we need another copy of size of A to perform deflection. So, in total, the required space is $O(mn)$, or more precisely dominated by $2mn$ (there are also some smaller matrices we need to store).

For Block Iteration, we only need to store small matrices ($O(mk)$ and $O(nk)$), assuming all computations are performed correctly. So, the space complexity is $O(mk)$.

For Krylov Block Iteration, we need to store Krylov subpace of size $O(mtk)$, so in this case, the space complexity depends on t.

5 Experiments

5.1 Image Compression

One possible application of low-rank approximation is data compression in general, and image compression in particular. One obvious benefit is the saved space: instead of storing one $m \times n$ image, we store two $m \times k$, $k \times n$ images, with $k \ll m$. The images is then reconstructed as a matrix product of these matrices. Although usually not used in practice due to the existence of better compression algorithms for images specifically, this is still a very simple and very general idea that can be applied to any data that can be represented as a matrix.

Table 1: Performance comparison of different SVD methods, $k = 10$, $\epsilon = 0.5$

Metric	Truncated SVD	rSVD	Power Iteration (deflation)	Block Power Iteration	Block Krylov Iteration
SSIM	0.626114	0.464286	0.62587	0.625921	0.626114
Relative Frobenius norm error	0.166893	0.245896	0.167037	0.167044	0.166893
Relative Spectral norm error	0.0526419	0.122124	0.0531099	0.0531292	0.0526419
Time (s)	0.0852879	0.00214573	0.0878975	0.0104674	0.0136205

Table 2: Performance comparison of different SVD methods, $k = 50$, $\epsilon = 0.5$

Metric	Truncated SVD	rSVD	Power Iteration (deflation)	Block Power Iteration	Block Krylov Iteration
SSIM	0.847862	0.688497	0.84759	0.847542	0.847862
Relative Frobenius norm error	0.0619878	0.102122	0.0620491	0.0620594	0.0619878
Relative Spectral norm error	0.0121688	0.0315171	0.0123818	0.0124058	0.0121688
Time (s)	0.0844845	0.00913211	0.446623	0.0247392	0.0533206

To test how the algorithms work on images, we used Flower 102 Dataset. The Flower 102 dataset (Nilsback and Zisserman [2008]) is a multi-class image classification dataset consisting of 102 different categories of flowers. The dataset contains 8,189 images, with an average of 82 images per class. The images are of varying sizes (usually around 500x500, but can be larger or smaller than that) and are represented in RGB color space, capturing the natural diversity of flowers in different environments. Unlike many other image datasets, the images in Flower 102 are not uniformly cropped or resized, reflecting their original dimensions and aspect ratios. The dataset is commonly used in the computer vision community for benchmarking image classification models.

We used this dataset by first selecting 1000 images at random, converting the images to grayscale and processing them as matrices. After that we ran all discussed methods on it, setting number of iterations as discussed in section 4, and computing different metrics (averaged).

One metric is SSIM (structural similarity index measure), which is a common way in computer vision to measure similarity between two images, we measured SSIM between original image and each of the resulting low-rank approximations. SSIM ranges from 0 to 1 with 1 being the best score. We also measured relative Frobenius and Spectral norm errors (relative are due to different size of images) between original and approximations and runtime. The results are summarized in the figures and tables below.

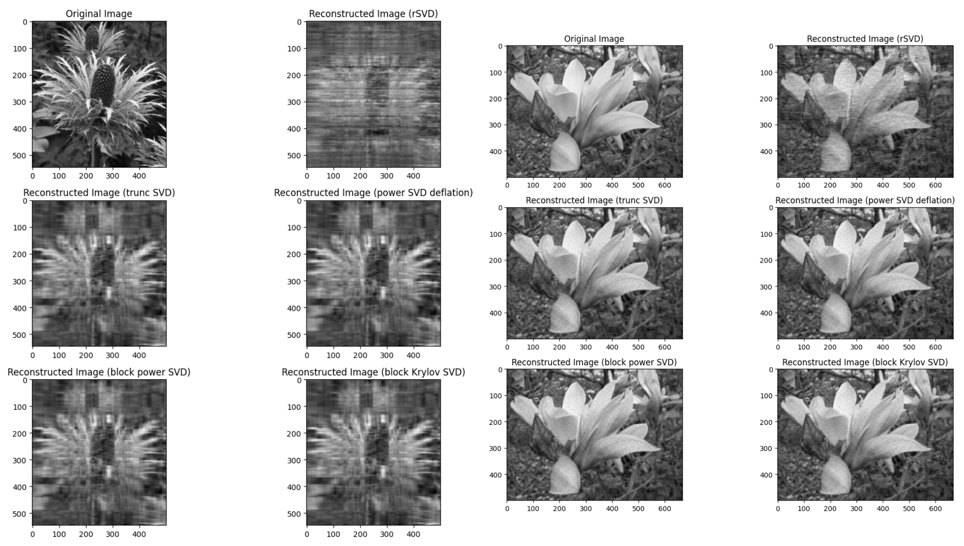
Original Image
Reconstructed Image (rSVD)
Reconstructed Image (trunc SVD)
Reconstructed Image (power SVD deflation)
Reconstructed Image (block power SVD)
Reconstructed Image (block Krylov SVD)
Original Image
Reconstructed Image (rSVD)
Reconstructed Image (trunc SVD)
Reconstructed Image (power SVD deflation)
Reconstructed Image (block power SVD)
Reconstructed Image (block Krylov SVD)

Figure 1: The results of image reconstructions by different methods. Left is for $k = 10, \epsilon = 0.5$, right is for $k = 50, \epsilon = 0.5$

.

Figure 1 shows original image and its reconstructions with parameters $k = 10, \epsilon = 0.5$ and $k = 50, \epsilon = 0.5$. As you can see, even with $k = 10$, most reconstructions, except for rSVD, look rather similar to the original image, although we see some some distortions. For $k = 50$, the results are

Table 3: Performance comparison of different SVD methods on Large Dataset. k = 10, $\epsilon = 0.5$

Method	Frobenius Norm Error	Spectral Norm Error	Runtime (sec)	RAM Used (MB)
rSVD	233.175	35.1933	0.0576109	3.49462
Truncated SVD	221.368	17.6278	91.2	887.069
Power Iteration (deflation)	221.375	17.7188	4.85592	888.394
Block Power Iteration	221.386	17.8391	0.777639	3.52305
Block Krylov Iteration	221.368	17.6278	0.990578	32.2679

Table 4: Performance comparison of different SVD methods on Large Dataset. k = 50, $\epsilon = 0.5$

Method	Frobenius Norm Error	Spectral Norm Error	Runtime (sec)	RAM Used (MB)
rSVD	225.266	26.4561	0.197349	17.5027
Truncated SVD	206.469	10.3101	90.5827	887.068
Power Iteration (deflation)	206.494	10.4405	26.4946	893.05
Block Power Iteration	206.497	10.4563	2.89315	17.5216
Block Krylov Iteration	206.469	10.3101	3.66485	166.424

even better from human eye perspective. For Truncated SVD, Power Iteration with Deflation, block power SVD, and block Krylov SVD, all 4 images visually look very similar, but in fact they are not the same. There are slight differences, which results in different values for different metrics as shown in tables 1 and 2.

Table 1 supports this findings. We observe that rSVD performs the worst across all metrics except for time, which makes sense since it is the only non-iterative method (somewhat equivalent to a single iteration of Block methods). In terms of errors and SSIM score, all 4 remaining methods perform mostly the same, with Truncated SVD performing better than any other method in terms of Frobenius and Spectral errors, which supports the theoretical results. It also often achieves the best SSIM scores (although not always). The time for both Truncated SVD and Power Iteration with deflation is relatively high compared to block methods as expected. Block Power Iteration runs slightly faster compared to Block Krylov, this is because the number of iterations for both methods is often the same or almost the same (due to matrix size and rounding), but as it was shown in section 4, in this setting, Block Krylov method would run slightly slower.

Table 2 shows similar results. For $k = 50$, the algorithms produce better reconstructions in terms of SSIM and relative norm errors. Interestingly, the time for power iteration with deflation is already larger than for truncated SVD, which makes it almost useless in this setting. Overall, we see a slowdown in terms of time for all methods, except for truncated SVD, which also makes sense. The runtime for Krylov method is slower than that of block power iteration and is approaching time for truncated SVD.

5.2 Large Matrix Compression

To properly test the described methods, we ran the code on a large matrix, corresponding to a user dataset. LastFM asia (Rozemberczki and Sarkar [2020]) is a social network of LastFM users which was collected from the public API in March 2020. Nodes are LastFM users from Asian countries and edges are mutual follower relationships between them. We generated an adjacency matrix of this graph, which ended up being a large 7624×7624 matrix.

We then ran the described methods on this matrix, measuring Frobenius norm error, Spectral norm error, Peak RAM usage, and runtime. We ran block methods and rSVD 5 times and computed the averages. For truncated SVD and Power Iteration with deflation, we only made 1 run. The results are summarized in the tables 3 and 4.

As you can see from tables 3 and 4, the results for norms on the larger dataset are similar to those for smaller. The RAM space used supports our theoretical results. The dataset matrix consumed 440 MB of memory, and we observe double that number for truncated SVD and Power Iteration (deflation). For Krylov method, we see significant consumption of space, for $k = 50$, it is already 166 MB, which is already about 19% of that used by SVD. In terms of time, the truncated SVD is completely outperformed by approximation methods. Among the iterative approximation methods, block power iteration demonstrates slightly faster time, but Krylov method is comparable with it.

References

Carl Eckart and Gale Young. Approximation of matrices by sums of orthogonal matrices. *Psychometrika*, 1(3):211–218, 1936.

Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53 (2):217–288, 2011.

Cameron Musco and Christopher Musco. Randomized block krylov methods for stronger and faster approximate singular value decomposition. *NeurIPS*, 2015.

Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. *Proceedings of the Indian Conference on Computer Vision, Graphics and Image Processing*, 2008.

Benedek Rozemberczki and Rik Sarkar. Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*, page 1325–1334. ACM, 2020.

A Appendix

Algorithm 1 Simplified Randomized SVD (rSVD)

```
**Require:** Matrix $A \in \mathbb{R}^{m \times n}$, desired rank $k$
**Ensure:** Approximate SVD of $A$: $U \in \mathbb{R}^{m \times k}$, $\Sigma \in \mathbb{R}^{k \times k}$, $V \in \mathbb{R}^{n \times k}$
  Generate random matrix $\Omega \in \mathbb{R}^{n \times k}$
  $Y = A\Omega$
  Compute QR decomposition of $Y$: $Y = QR$
  $B = Q^T A$
  Compute SVD of $B$: $B = U_B \Sigma_B V_B^T$
  $U = QU_B$
  **return** $U, \Sigma_B, V_B$
```

Algorithm 2 Power Iteration with Deflation for Approximate SVD

```
**Require:** Matrix $A \in \mathbb{R}^{m \times n}$, desired rank $k$, number of iterations $t$
**Ensure:** Approximate SVD of $A$: $U \in \mathbb{R}^{m \times k}$, $\Sigma \in \mathbb{R}^{k}$, $V \in \mathbb{R}^{n \times k}$
  Initialize $U \in \mathbb{R}^{m \times k}$, $\Sigma \in \mathbb{R}^{k}$, $V \in \mathbb{R}^{n \times k}$
  **for** $i = 1, \ldots, k$ **do**
    Initialize random vectors $u \in \mathbb{R}^{m}$, $v \in \mathbb{R}^{n}$
    **for** $j = 1, \ldots, t$ **do**
      $u = \frac{Av}{||Av||_2}$
      $v = \frac{A^T u}{||A^T u||_2}$
    **end for**
    $\Sigma_i = u^T A v$
    $U[:, i] = u$
    $V[:, i] = v$
    $A = A - \Sigma_i u v^T$
  **end for**
  **return** $U, \Sigma, V$
```

Algorithm 3 Block Power Iteration

```
**Require:** Matrix $A \in \mathbb{R}^{m \times n}$, desired rank $k$, number of iterations $t$
**Ensure:** Low-rank approximation of $A$: $U \in \mathbb{R}^{m \times k}$
  Generate random matrix $\Omega \in \mathbb{R}^{n \times k}$
  Compute QR decomposition of $\Omega$: $\Omega = QR$
  **for** $i = 0$ to $t - 1$ **do**
    **if** $i = 0$ **then**
      $Block = AQ$
    **else**
      $T = A^T Q$
      $Block = AT$
    **end if**
    Compute QR decomposition of $Block$: $Block = QR$
  **end for**
  Compute $B = Q^T A$
  Compute $B = BA^T Q$
  Compute SVD of $B$: $B = U_B \Sigma_B V_B^T$
  $U = QU_B$
  **return** $U$
```

Algorithm 4 Block Krylov Iteration

```
Require: Matrix $A \in \mathbb{R}^{m \times n}$, desired rank $k$, number of iterations $t$
Ensure: Low-rank approximation of $A$: $U \in \mathbb{R}^{m \times k}$
  Generate random matrix $\Omega \in \mathbb{R}^{n \times k}$
  Compute QR decomposition of $\Omega$: $\Omega = QR$
  Compute Krylov subspace:
  for $i = 0$ to $t - 1$ do
    if $i = 0$ then
      $Block = AQ$
    else
      $T = A^T Q$
      $Block = AT$
    end if
    Compute QR decomposition of $Block$: $Block = QR$
    Append $Q$ to Krylov subspace $K$
  end for
  Stack Krylov subspace $K$: $K = [Q_0, Q_1, ..., Q_{t-1}]$
  Orthogonalize columns of $K$ via QR decomposition: $K = QR$
  Compute $B = Q^T A$
  Compute $B = BA^T Q$
  Compute SVD of $B$: $B = U_B \Sigma_B V_B^T$
  $U = QU_B$
  Cut $U$ to top-$k$: $U = U[:, : k]$
  return $U$
```