Power Method for Computing SVD of a Matrix

Yiwen Zhang (yiwenz2) and John Zhang (tianyuz2)

Carnegie Mellon University

Abstract. In this paper, we will use power method to calculate the eigenvalues of the matrix A^TA, which is then used to compute the singular values of the original matrix A. The procedure, advantages and applications of power method are briefly discussed, and the process of using power method to perform Principal Component Analysis (PCA) on a large image is presented.

Keywords: Linear Algebra · Matrix · SVD · Power Method · PCA

1 Introduction

Eigenvalues and singular values play an important role in the application of linear algebra. As we have seen in class, they can be very useful to understand a matrix. For now, our naive method of finding the singular values of a matrix A involves finding the eigenvalues of the matrix A^TA. This theorem will be proven in the Definitions and Notations section. To find the eigenvalues of a matrix, we need to find the roots of its characteristic polynomial. This method may be feasible for a matrix with lower rank but would be extremely complicated for an industrial sized matrix. Thus, the singular values of a matrix need to be obtained by other methods. Fortunately, there exists some other methods to calculate the singular values of a matrix A with power method being one of them. Power method works by refining its approximation of the eigenvalues of a matrix iteratively, until the error of the approximation is below a certain threshold. With the application of power method, we can approximate the dominant eigenvalue of the matrix A^TA easily, which is the square of the singular value of matrix A. This method can significantly simplify our calculation of singular values, and it makes the calculation of singular values of industrial size matrices more efficient. With power method, we can now perform PCA on a large-size image more easily.

In this paper, we will examine the reliability of power method and summarize its derivations, procedures, advantages and drawbacks. This paper is meant to be a record of our exploration of the feasibility and application of power method.

In Section 2 of this paper, we will provide a brief introduction of the definitions and notations that will be used throughout this paper. In Section 3, we will present the main theorems of power method and study it in detail. We will show how to use power method to calculate the singular values of a matrix step by step. After that, in Section 4, we will present the result we get after applying power method to the PCA on an image and the calculation of the singular values of a matrix. Finally, in Section 5, we will conclude the whole study process,

the profits and drawbacks of power method and the reflection over our research progress.

2 Definitions and Notations

Before starting to use power method to calculate the singular values of a matrix A, we need to go over some definitions and notations that will be frequently used in the following proof and calculation process.

Theorem 1. *Every matrix A over $\mathbb{R}$ has a singular value decomposition. $A = U\Sigma V^T$ [1]*

Definition 1. *Let matrix $A \in \mathbb{R}^{n\times n}$. A nonzero vector $v \in \mathbb{R}^n$ is called an eigenvector of A with corresponding eigenvalue λ if $Av = \lambda v$.*

Late in this paper, we will frequently use eigenvalues and eigenvectors to calculate singular values.

Definition 2. *Let matrix A over $\mathbb{R}$. Let v_1 be the norm-one vector v maximizing $||Av||$, let v_2 be the norm-one vector v orthogonal to v_1 that maximizes $||Av||$, let v_3 be the norm-one vector v orthogonal to v_1 and v_2 that maximizes $||Av||$, and so on. $\sigma_i = ||Av_i||$. The vectors $v_1, v_2, \ldots, v_r$ are the right singular vectors of A, and the corresponding real numbers $\sigma_1, \sigma_2, \ldots, \sigma_r$ are the singular values of A.*

Definition 3. *The vectors $u_1, \ldots, u_r$ such that $\sigma_j u_j = Av_j$ are the left singular vectors of A.*

Theorem 2. *The eigenvalues of the square matrix A^TA are the square of the singular values of matrix A.*

Proof. Let $A = U\Sigma V^T$. $A^T = (U\Sigma V^T)^T = V\Sigma^T U^T$. Writing A^T in the summation form, we have $A^T = \sum_i \sigma_i v_i u_i^T$. Consider a new matrix B

$$\begin{aligned} B &= A^TA \\ &= (\sum_i \sigma_i v_i u_i^T)(\sum_i \sigma_j u_j v_j^T) \\ &= \sum_{i,j} \sigma_i\sigma_j v_i (u_i^T u_j) v_j^T \\ &= \sum_i \sigma_i^2 v_i v_i^T \end{aligned}$$

We can see that B is square and symmetric. Moreover, the eigenvalues of B are σ_i^2, which are the square of the singular values of A.

Notation 1. For a square matrix A^TA over $\mathbb{R}$, λ_i is its i-th eigenvalue and v_i is the corresponding eigenvector.

Notation 2. For a matrix A over $\mathbb{R}$, u_i is its i-th left singular vector, σ_i is its i-th singular value and v_i is its i-th singular vector.

3 Main Theorems

3.1 A First Step

As we have proved in Theorem 2.2, we can obtain the singular values of a matrix by computing the eigenvalue of that matrix transpose multiplied by itself.

Let's again have $B = A^T A$. Consider the computation of B^2

$$\begin{aligned} B^2 &= (\sum_i \sigma_i^2 v_i v_i^T)(\sum_j \sigma_j^2 v_j v_j^T) \\ &= \sum_{i,j} \sigma_i^2 \sigma_j^2 v_i (v_i^T v_j) v_j^T \end{aligned}$$

When $i \neq j$, $v_i^T v_j = 0$ because V has orthogonal columns. Thus $B^2 = \sum_{i=1}^r \sigma_i^4 v_i v_i^T$. Moreover, if we keep multiplying B onto this matrix, all the cross products will be zero because of orthogonality and hence

$$B^k = \sum_i \sigma_i^{2k} v_i v_i^T$$

If $\sigma_1 > \sigma_2$, this means the first term will dominate as we take the limit on $k \to \infty$. The summation will thus converge to the first term, giving us $B^k \to \sigma_{2k} v_1 v_1^T$. This essentially means we can obtain a close estimation of v_1 by simply normalizing the first column of B^k.

However, a problem with this approach is that in many applications of SVD like performing PCA on an image, the original matrix A is very big, which causes B to be huge as well. Computing matrix multiplication can be very costly. A naive algorithm will have $O(n^3)$ time complexity and the fastest known algorithm, the Coppersmith-Winograd algorithm [2], with $O(n^{2.3737})$. Hence computing B^k for very large k might not be very realistic. This motivates us to develop the faster method in the next section.

3.2 An Improved Approach

In this improved version, instead of computing the high powers of B and the resulted matrix multiplications, we wish to reduce it to matrix-vector multiplications which have significantly lower time complexity. We first select a random vector x and compute $B^k x$. Recall that the matrix V in the SVD of $B = U\Sigma V^T$ has orthonormal columns that we can use as a basis of the vector space. Hence we write $x = \sum_{i=1}^d c_i v_i$. Computing $B^k x$ would then give us

$$\begin{aligned} B^k x &\approx (\sigma_1^{2k} v_1 v_1^T)(\sum_{i=1}^d c_i v_i) \\ &= \sigma_1^{2k} c_1 v_1 \end{aligned}$$

Note that again we are using the orthogonality of v_i and v_j for $i \neq j$.

Normalizing the resulting vector would give us v_1, the first singular vector of A. Using $\sigma_1 = ||Av_1||$ and $u_1 = \frac{1}{\sigma_1}Av_1$ would give us the first singular value and left singular vector.

Note that $B^k x$ is computed by a series of matrix-vector products, rather than matrix products described in the previous section. Concretely,

$$B^k x = (A^T A)^k x = A^T A A^T A \dots A^T A x$$

which is computed from right to left.

Since we have computed (σ_1, u_1, v_1) we can extend this algorithm to a full SVD of the matrix by repeating the above procedure for

$$A - \sigma_1 u_1 v_1^T = \sum_{i=2}^{n} \sigma_i u_i v_i^T$$

and of which the singular value and vectors will be (σ_2, u_2, v_2). The full play out of this algorithm will be seen in the `PowerMethod` object in our Python script. We have added a decent amount of comments in the script for the reader to map the algorithm described here to the corresponding codes.

Because of the rather complicated probability theory involved in calculating the number of iterations until the vector converges, rigorous time complexity analysis of this algorithm is beyond the scope of this paper. However, we would like to mention the complexity of this algorithm is $O(mnks)$ for a matrix of dimension $m \times n$ and computing the first k singular values and vectors. The s here the number of iterations performed. We refer to this article which delves deeper into the value of s for interested readers. [3] Nevertheless, we will show empirically the lower time complexity of this method comparing to traditionally SVD methods in the next section.

4 Results

4.1 A Simple Test

Now we have developed power method to compute the singular value decomposition of a matrix A. We first want to apply this method to calculate the SVD of matrices with higher dimensions, such as 10×10 matrices. We feed the below matrix into our python script:

```
[[ 0.041,0.815,0.245,0.054,0.249,0.534,0.753,0.307,0.877,0.429],
 [ 0.918,0.846,0.249,0.262,0.133,0.32, 0.446,0.122,0.164,0.711],
 [ 0.139,0.701,0.726,0.094,0.036,0.695,0.325,0.29, 0.373,0.692],
 [ 0.644,0.067,0.032,0.896,0.047,0.55, 0.062,0.568,0.204,0.275],
 [ 0.631,0.412,0.232,0.415,0.335,0.508,0.393,0.549,0.076,0.698],
 [ 0.231,0.231,0.452,0.567,0.453,0.677,0.434,0.887,0.323,0.677],
 [ 0.918,0.846,0.249,0.262,0.133,0.32, 0.446,0.122,0.164,0.711],
 [ 0.139,0.701,0.726,0.094,0.036,0.695,0.325,0.29, 0.373,0.692],
 [ 0.644,0.067,0.032,0.896,0.047,0.55, 0.062,0.568,0.204,0.275],
 [ 0.631,0.412,0.232,0.415,0.335,0.508,0.393,0.549,0.076,0.698]]
```

We can obtain the singular value decomposition of this matrix with power method in 0.00444912 seconds. It takes 0.01702308 seconds to obtain the SVD of same matrix using the `numpy.linalg.svd()` function in Numpy. We can see the significant difference between the run time between these two approaches.

However, one important thing to note is that this discrepancy in time our implementation takes and numpy takes is not conclusive: numpy's function calculates the full SVD while ours only calculate the first k. Numpy performs better when we increase k because numpy itself is written in C/C++ and FORTRAN which is thousands times more efficient than python. There is virtually no way for us to make an exact comparison except re-implementing the numpy library ourselves in python. However, our implementation provides a way to specify the parameter k which numpy cannot.

Then we try to examine the precision of the singular values our method outputs. Since the rank of the matrix is 6, we should get 6 non-negative singular values. The output singular values of this matrix are:

```
4.283450426807314,
1.639836158688936,
1.213184990284024,
0.7440354654912632,
0.6373832397083439,
0.11328647280325838
```

To validate our output, we calculate the singular values of this matrix again with `numpy.linalg.svd()`. The singular values we got are:

```
4.28345043,
1.63983719,
1.21318394,
0.744039950,
0.637377309,
0.113286465
```

The MSE of our result to numpy's calculation is $9.5751e-12$. Obviously, we can find all these six values are very close to the singular values we obtained by using power method. Thus, we can find that using power method to calculate singular value decomposition of a matrix is both efficient and precise.

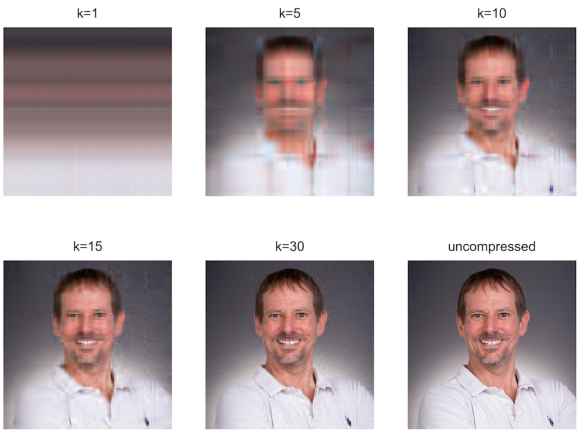
4.2 PCA on Images

Apart from using power method to compute the singular value decomposition of matrices with higher dimension, we also try to find the real-world application of power method. Inspired by the lecture on PCA, we decide to use power method to compress large images more efficiently.

However, as noted in the previous subsection, we came to realize there is no way to rigorously compare the performance of power method with the traditional one quantitatively. However, when k is small, using our implementation of power

method to compute PCA on a large image still takes less time than using the full SVD provided by numpy. We ran a PCA on Prof. Mackey's profile image using both our implementation of power method and the numpy function. This gives us the below result:

Fig. 1. Result produced by `main.py`



Below are some statistics calculated by the `timing.py` script.

```
Numpy full SVD: 0:00:12.449012
Power method:
(k = 1) 0:00:03.050974
(k = 5) 0:00:03.587041
(k = 10) 0:00:04.386675
(k = 15) 0:00:06.206003
(k = 30) 0:00:14.576710
```

As we can see, our implementation's run time exceeds the Numpy full SVD somewhere between $k = 15$ and $k = 30$. This is still a decent run-time performance, considering the natural disadvantage of the performance of python.

5 Conclusion

The purpose of this paper is to present the derivation, procedure, advantages, drawbacks and applications of power method to obtain the singular value decomposition of a potentially large-size matrix. This method is based on the fact that

while we raise the power of A^TA to a certain level and multiply it by a vector x, we can get the approximation of the dominant eigenvalue and corresponding eigenvector. Hence, we can obtain the singular value and the left singular vector of the original matrix A out of the above computation. This method can be very efficient when dealing with matrices with higher dimensions. Most real-world matrices are of large dimensions, so power method can be really useful in daily life. We have already shown power method can efficiently compress an image.

However, even power method is more efficient than the normal method, it is still not efficient enough. During our process of research, we have seen many probable ways that we can use in the future to refine the algorithm so that the implementation of the method can speed up. For example, power method is more efficient when applied to a matrix with linear independent columns. Since we do not know the rank of the matrix beforehand, the output may contain a large number of singular values that are actually 0. If we could know the rank of the matrix A in advance, assume $rank(A) = k$, we can ask the algorithm to output only the first k singular values. This can save a large amount of time, especially in solving real world problems. Also, how precise our approximation of the dominating eigenvalue of matrix A^TA remains to be a problem. The precision of our approximation of the eigenvalues of A^TA will affect the precision of the singular values we get. For now, we will output the eigenvalue of A^TA if the difference of the previous eigenvalue and the current eigenvalue is within 0.00001. Because of that, the margin of errors of the singular values we obtain is about 0.00002. This may seem trivial, but this kind of error can be extremely dangerous while in real world situations. So, we could improve our precision of calculating the eigenvalues of A^TA .

As with all the other iterative algorithms, power method can always be improved in its efficiency and precision. While we are exploring the applications of the algorithm, we find that it may be hard to keep a balance between efficiency and precision, but we also see the huge potential of power method. Power method can be extremely suitable for sparse matrices and we can design more advanced eigenvalue and singular values computing algorithm based on power method. Therefore, we will keep exploring the application of power method in various subjects and fields.

6 Acknowledgement

We would like to thank all the TAs for their help and advice throughout the process of designing algorithms and coding. Their feedback and suggestions are really valuable to us. Also, we would also like to thank to Prof. Radcliffe for introducing us to the study of linear algebra.

References

1. Lecture 3: Introduction to singular value decomposition. 18-335J Introduction to Numerical Methods, Massachusetts Institute of Technology Page 4-9.

2. Coppersmith, Don; Winograd, Shmuel (1990), "Matrix multiplication via arithmetic progressions", Journal of Symbolic Computation, 9 (3): 251, doi:10.1016/S0747-7171(08)80013-2
3. Lecture 7: Singluar Value Decomposition. 0368-3248-01 Algorithms in Data Mining, Yale University Page 5-6.
4. Blum, A., J. Hopcroft, and R. Kannan (2018): Foundations of data science, Cornell University Section 3.7