Power Method for SVD Approximation

Allen & Jiachen (Amy) Liu

December 13th, 2018

1 Introduction

Singular value decomposition (or SVD for short) is a matrix factorization method that dates back as far as the 19th century with the first proof of SVD for rectangular matrices devised by Carl Eckart and Gale Young in 1936[4]. Applications of SVD can be found in many instances such as data compression, image processing, and noise reduction, in part due to the fact that SVD can be calculated for any rectangular matrix rather than just square matrices. While calculating SVD on smaller matrices is relatively easy, computing the singular value of an nxn matrix is $O(n^3)$. [7] Unfortunately as a result, computing singular values on a significantly larger matrix such as images will be exponentially more time consuming. The rise of big data has caused data sets for applications in machine learning to be equally as massive and improvements in image sensor technology have pushed the boundary of photo resolution. As one can see, calculating SVD for practical uses today is often simply too inefficient.

As a result of this dilemma, the majority of applications of SVD today don't actually find the singular value decomposition, but rather generates the SVD using close approximations of the singular values. The Power Method is one such method. Originally devised as a way to approximate eigenvalues, we are able to use it to approximate singular values by exploiting the relationship between the two. Depending on the implementation, the Power Method's time complexity varies based on specifications. However, even in the worst case scenario, calculating the singular value using the Power Method is only $O(n^2)$, which significantly cuts down on the computation of singular value while still producing very similar results.[4]

2 Definitions

2.1 Limits

A limit is the value that a function (or sequence) approaches as the input approaches some value. Denoted as $\lim_{x \to c} f(x) = L$ where L is the limit.

2.2 Convergence

In the context of limits, a limit is convergent if the limit exists.

2.3 Tuple

A tuple is a finite order list (known as a sequence) of elements. If the tuple has n elements, we denote it as an n-tuple.

2.4 Dominant Eigenvalue and Eigenvector

Let $\lambda_1, \lambda_2, ..., \lambda_n$ be the eigenvalues of an n x n matrix A. λ_i is called the dominant eigenvalue of A if $|\lambda_i| > |\lambda_k| \ \forall k \in 1, 2, ..., n$ where $k \neq i$. The eigenvectors corresponding to λ_i are called the dominant eigenvectors. [5]

3 Main Theorems & Ideas of Power Method

Recall the calculation of singular value decomposition and the definition of the eigenvalue & eigenvector that was detailed in 21-241 Linear Algebra course.

3.1 How & Why the Power Method works

The Power Method is an iterative method that approximates the dominant eigenvalue (which in the process calculates the corresponding dominant eigenvector) of a square matrix A. The reason why we can use it for SVD is that the singular values of A and A^TA are the same; therefore, by taking the square root of the resulting eigenvalue of A^TA from the power method, we get the singular value of A. Secondly, the eigenvector calculated from the power method is also the right singular vector. The left singular vector can be calculated through the formula $u_i = \frac{1}{\sigma_i} A v_i$ where u_i is the corresponding left singular vector, σ_i is the singular value, A is the matrix, and v_i is the right singular vector.

The power method approximates the dominant eigenvector by utilizing the fact that it is equal to $\lim_{k\to\infty}(ATA)^k x$ (see section on Convergence of Power Method below). Instead of calculating the limit, a reasonably large k is enough to give a good approximation of the dominant eigenvector. Once we have this eigenvector, we must find the corresponding eigenvalue to get the singular value. This can be achieved through the usage of Rayleigh Quotient (see section on Rayleigh Quotient below).

To find subsequent singular values and vectors, we take the original matrix A and subtract it by $\sigma_i u_i v_i$ and run the power method on the resultant matrix. Intuitively, this works because $\sigma_i u_i v_i$ produces the best rank i approximation of the matrix A. By removing this from the matrix A can be thought of "removing" this dimension. As a result, the power method will find the next dominant eigenvector and value, producing the next best rank i+1 approximation.

3.2 Rayleigh Quotient

If x is an eigenvector of a matrix A, then its corresponding eigenvalue is given by

$$\lambda = \frac{Ax \cdot x}{x \cdot x}$$

As shown in class, the singular values of a rectangular matrix A is the same as the square root of the eigenvalues of A^TA. As a result, the Rayleigh Quotient allows one to calculate the corresponding singular value after the approximation of the right singular vector.[5],

3.3 Convergence of Power Method

If A is an n x n diagonalizable matrix with a dominant eigenvalue, then there exists a nonzero vector x_0 such that the sequence of vectors given by

$$Ax_0, A^2x_0, A^2x_0, A^3x_0, ..., A^kx_0$$

approaches a multiple of the dominant eigenvector of A.[5]

The proof of this theorem is shown in Elementary Linear Algebra [5] on page 591.

4 Implementation

This section will explain the details of the cycle that finds the singular value, left singular vector, and right singular vector for the largest singular value, and repeats for the 2nd, 3rd, etc singular values and its corresponding L/R vectors. The complete set up of the entire function other than the cycle portion (such as variable naming, storage etc.) can be found on the submitted code.

4.1 Constructing A^TA

Constructs a copy of input matrix as it is: `Acopy = copy.deepcopy(A)`

Computes ATA: `ATA = np.transpose(Acopy).dot(Acopy)`

The variable ATA represents A^TA. Since we need to preserve the input matrix A for the future operation of subtracting the best rank approximation from A, we perform a deepcopy. As a part of the *copy* package, the function returns a pointer to a another distinct numpy array that contains the same values as A (in contrast to simply using = that would return an alias, a pointer pointing to the original A to execute calculation upon). In the 2nd line of code, `np.transpose(Acopy)` finds the A's transpose, and `.dot(Acopy)` multiplies the transpose with its original, yielding A^TA.

4.2 Finding eigenvalue & eigenvector

We choose a random vector x: `v = np.random.rand(ATA.shape[1])`

Algorithm of sec 3.3: `for _ in range(k):`

Matrix-vector multiplication: `temp = np.dot(ATA, v)`

Normalizes the vector each time: `v = temp / np.linalg.norm(temp)`

Calculate eigenvalue: `sval = math.sqrt(np.dot(ATA, v).dot(v) / np.dot(v,v))`

As detailed by the main theorem's section 3.3 where the $A^k x$ for a reasonably high k will yield a multiple of the dominant eigenvector, this code simulates that by initializing a random vector v and conduct $v = Av$ over and over again for k times as a for loop.

On the 5th line, the eigenvalue can be calculated through a simple process. First, we calculate the dot product of $A^T A \cdot v$. We then calculate the dot product of $(A^T A \cdot v) \cdot v$. By dividing by the $||v||^2$ which can be attained by the dot product of $v \cdot v$, we get the eigenvalue. This is known as Rayleigh Quotient outlined in section 3.2. Since we that the eigenvalue is the singular value squared, we will take the square root, hence `sval = math.sqrt(...)`.

4.3 Finding left singular vector

The left singular vector: `u = (1/sval)*A.dot(v)`

Recall from the 21-241 course, that the left singular vector, u_i is obtained by $u_i = \frac{1}{\sigma_i} \cdot A \cdot v_i$ where A is the original matrix and v is the right singular vector.

4.4 Iteration

new A: `A = A - sval*(np.dot(u.reshape(5,1),(np.transpose(v).reshape(1,10))))`

Recall that the best rank 1 approximation can be formed $A_1 = \sigma_1 u_1 v_1$. As detailed in section 3.1, by subtracting this approximation from A, we result in a A' that shifts all dimensions downwards by one rank, stripped of the previously approximated singular values and L/R vectors. Thus, yielding a matrix to perform best rank 1 approximation on again.

5 Results

5.1 Provided Test Case

$$\text{Input: } M = \begin{bmatrix} 0.041 & 0.815 & 0.245 & 0.054 & 0.249 & 0.534 & 0.753 & 0.307 & 0.877 & 0.429 \\ 0.918 & 0.846 & 0.249 & 0.262 & 0.133 & 0.32 & 0.446 & 0.122 & 0.164 & 0.711 \\ 0.139 & 0.701 & 0.726 & 0.094 & 0.036 & 0.695 & 0.325 & 0.29 & 0.373 & 0.692 \\ 0.644 & 0.067 & 0.032 & 0.896 & 0.047 & 0.55 & 0.062 & 0.568 & 0.204 & 0.275 \\ 0.63 & 0.412 & 0.232 & 0.415 & 0.335 & 0.508 & 0.393 & 0.549 & 0.076 & 0.698 \end{bmatrix}$$

The results of calling our power_method compared to an official singular value decomposition provided by Numpy [7] are shown on the next page.

Figure 1: Power Method Implementation Output

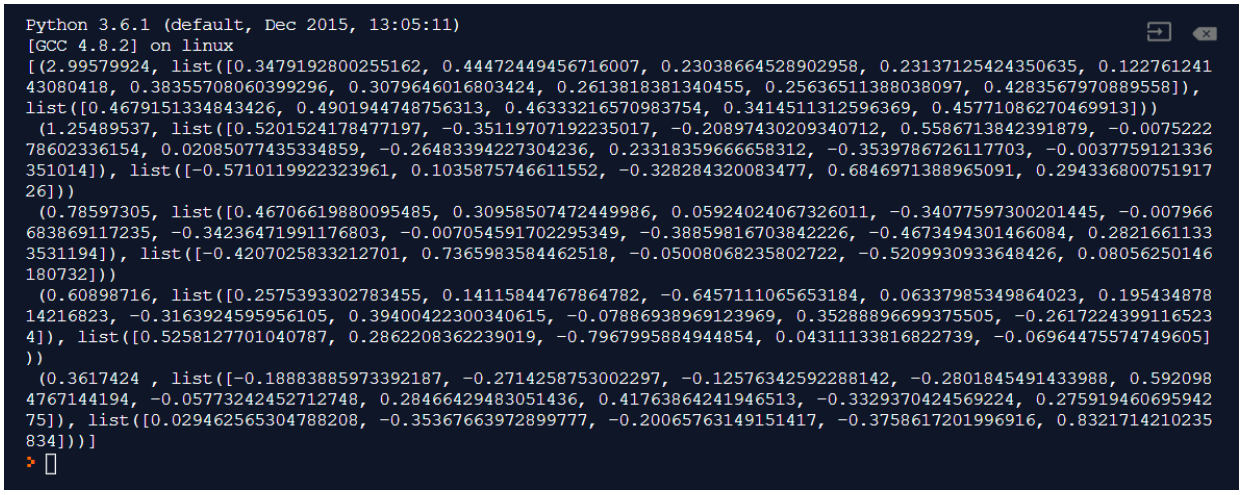
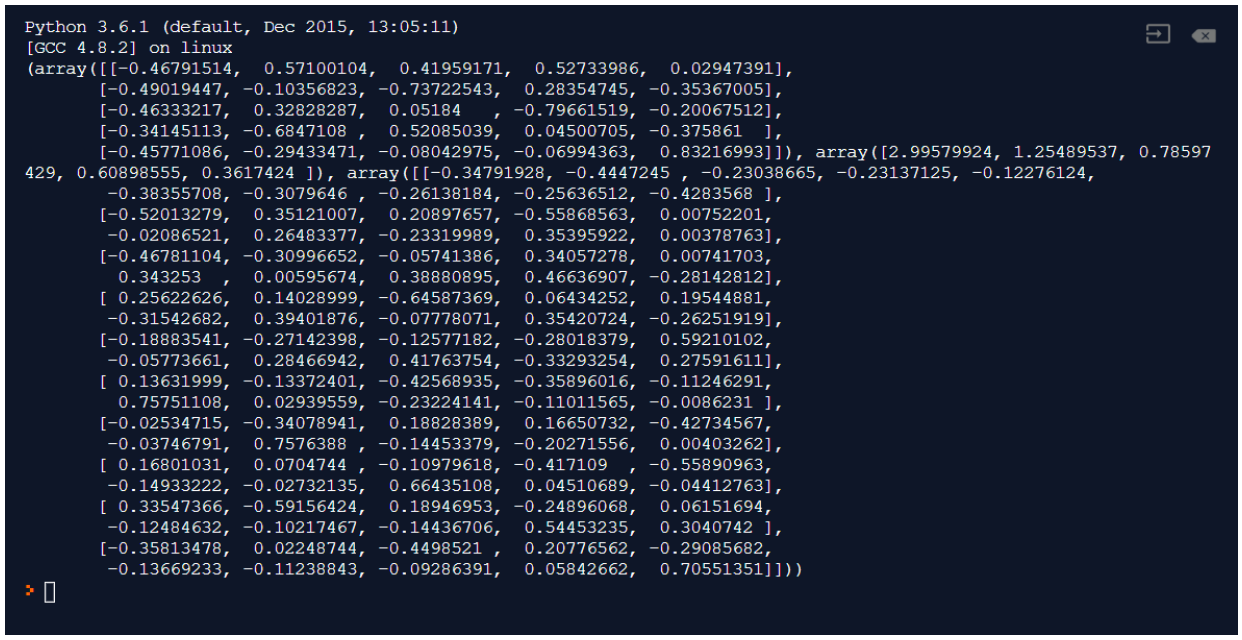


Figure 2: Numpy SVD Output



5.2 Analysis

As the outputs have shown, even with a relatively small k-value of 10, the results of our power method implementation are identical to Numpy's SVD implementation up to the 7th decimal place. Depending on the application, you could get away with even lower k-values in lower precision environments.

However, one notable difference between the two results is the sign in the first array element representing the first left and right singular vectors. Upon further research into this issue, we realized that many of today's current algorithms to find SVD (including Numpy's implementation, MATLAB, etc.) does not yield useful data from their signs. The signs of the result are simply byproducts from the numerical calculations to get the result and

not actually meaningful in describing the original matrix[3]. There is an algorithm in giving meaning to the signs from SVD created by Sandia National Laboratories; however, for the purpose of this assignment, we deemed it was not necessary to do so because of the added computational complexity required.

6 Conclusion

Mathematically, the power method performs its job in approximating the largest singular values and its corresponding left & right singular vectors. For reasonably high power value, we can calculate it with an appropriate error range but at a provably lower time-complexity cost. This is an acceptable time and precision trade-off for a majority of professional fields.

Our code successfully simulates the Power Method algorithm with the help of python's Numpy package that performs matrix-vector multiplication and transposition. The procedure repeated appropriately to find all singular values & L/R vectors for a full singular value decomposition. The program is capable of calculating such decomposition within fractions of a second as shown in our results section.

This project has shown that the power method is provenly useful and can be easily implemented for as few as a dozen lines of code. It presents a cost-efficient solution that provides comparable results to the large scale singular value decomposition that may be nearly impossible given current computational power. This is why virtually all implementation of SVD utilizes some variation of the power method to easily generate the desired values such as Google PageRank and Gumgum's natural language processing system.

7 Bibliography

References

[1] Avrim Blum, John Hopcroft, and Ravindran Kannan, *Foundations of Data Science*, Cornell CS, Ithaca, January 4, 2018.

[2] Edo Liberty, *Lecture 7: Singular Value Decomposition*, Yale University: 0368-3248-01-Algorithms in Data Mining, Fall 2013.

[3] Rasmus Bro, Evrim Acar, and Tamara Kolda, *Resolving the Sign Ambiguity in the Singular Value Decomposition*, Technical Report SAND2007-6422, Sandia National Laboratories and United States Department of Energy, New Mexico and California, October 2007.

[4] Carl Eckart and Gale Young, *The Approximation of One Matrix By Another of Lower Rank*, Psychometrika **vol.1** (September 1936), no. 3, 211–218.

[5] Ron Larson, Bruce H. Edwards, and David C. Falvo, *Elemantary Linear Algebra*, 6th ed., Houghhton Mifflin, Boston, January 1, 2008.

[6] The Scipy community, *NumPy v1.15 Manual* (August 23, 2018), `https://docs.scipy.org/doc/numpy-1.15.1/index.html`. Accessed December 14, 2018. Specifically, the numpy.array, numpy.linalg sections.

[7] Mary Radcliffe, *Math 241: Final Project Options*: *Power Method*, November 19, 2018. Cited for its claim that SVD has a complexity of $O(n^3)$.