The Random SVD: Part I (1/28/20)

Lecturer Anil Damle, Scribe Misha Padidar

Data Sparse Matrix Computations, Cornell University

1 Introduction

The Random SVD is a method for using randomness to approximately compute the Singular Value Decomposition (SVD). This is particularly useful when a matrix has a numerical rank much smaller than either of its dimensions, such as matrices that appear in Principal Component Analysis, Least Squares, and PDE solvers based on the Fast Multipole Method, to name a few. In these applications, matrix factorizations such as the SVD can significantly speed up or stabilize solvers; so it is particularly desirable to have an efficient SVD on hand. To this end, the Random SVD ameliorates many of the scalability problems associated with the traditional SVD.

Throughout these notes we will discuss a prototype method for computing the approximate SVD of a large matrix A using randomness. By approximate we mean that the approximate factorization has the same form as the SVD, $A \approx U\Sigma V^T$, as well as a similar approximation quality. On 1/24/20 not only did we find a method to do this, but we also found an interesting property, namely, that the components of the approximate SVD need not resemble the components of the true SVD.

After outlining the algorithm, we will discuss the complexity of the Random SVD, provide a partial proof of an error bound, and then comment on practical implementations. For a thorough and digestible treatment of this topic we direct the reader to [3].

2 Fixed Rank Algorithm

In this section we describe an algorithm for using randomness to compute a rank k approximate SVD of a matrix $A \in \mathbb{R}^{n\times n}$ where n is large. For simplicity we only consider squares matrices, however the techniques and proofs generalize to rectangular matrices as well. This *fixed rank method* is seldom used on it's own, as it is rare to a priori know the rank of a massive matrix. More generally, one desires an approximation of A to desired precision ϵ; this is known as the *fixed precision problem*. The fixed precision problem can be solved by sequentially iterating fixed rank algorithms, at a minor additional computational expense [3].

The method for computing the approximate factorization proceeds in two steps. First, we compute a low-dimensional subspace that captures the range of A. Secondly, we project A onto the subspace and compute the factorization [3]. The first step is a technique that is commonly used in subspace iteration the primary difference being that the use of randomness guarantees (probabilistically) that the subspace will be the desired dimension [2, 3]. This condition is also ensured by the use of an oversampling factor p.

Typically p is chosen within $5 - 10$, as the probability converges super-exponentially in p [3]. A safe upper bound is $p \leq k$, however the choice of p is generally dependent on matrix size and the decay rate of singular values.

Given a matrix $A \in \mathbb{R}^{n \times n}$, a target rank k and an oversampling factor p we compute the following steps

Algorithm 1: Fixed Rank Random SVD

Result: Approximate rank $k+p$ factorization $A \approx \tilde{U}\tilde{\Sigma}\tilde{V}^T$
Input: matrix $A \in \mathbb{R}^{n \times n}$, target rank $k > 0$, oversampling factor $p > 0$
1. Draw $\Omega \in \mathbb{R}^{n \times (k+p)}$ i.i.d. samples from the standard normal distribution.
2. Compute the product $Y = A\Omega$.
3. Compute the reduced factorization $Y = QR$, with $Q \in \mathbb{R}^{n \times (k+p)}$ and $R \in \mathbb{R}^{(k+p) \times (k+p)}$ (see [2] for reduced factorizations).
4. Compute the product $B = Q^T A$.
5. Compute the reduced SVD $B = U_B \Sigma_B V_B^T$. with $U_B \in \mathbb{R}^{n \times (k+p)}$, $\Sigma_B \in \mathbb{R}^{(k+p) \times (k+p)}$, and $V_B^T \in \mathbb{R}^{(k+p) \times (k+p)}$.
6. Set $\tilde{U} = QU_B$, $\tilde{\Sigma} = \Sigma_B$, $\tilde{V} = V_B$.
return $\tilde{U}, \tilde{\Sigma}, \tilde{V}^T$

Algorithm 1 is similar to the true SVD twofold. Firstly it has the same form $A \approx \tilde{U}\tilde{\Sigma}\tilde{V}^T$ where $\tilde{U}$ and $\tilde{V}$ are orthonormal and $\tilde{\Sigma}$ is diagonal. Secondly, as we will see in section 4, the approximation error is a function of the singular value of A, σ_{k+p+1}, just as in the truncated SVD.

This prototype algorithm serves as the base for constructing more sophisticated and robust algorithms for matrix factorizations. With that in mind we do not recommend Algorithm 1 for general application as there are many instances where it is insufficient. For instance, when the singular values of A decay slowly this algorithm suffers from inaccuracy. The reasoning is that to achieve minimum error the matrix Q should resemble the first $k+p$ left singular vectors of A. Consequently, the product $A\Omega$ must *reveal* the singular vectors, which requires the singular values to be largely different.

The next algorithm presents a quick fix to this problem through a minor adaptation of Algorithm 1. By raising our matrix to a power q via $(AA^T)^q A$ prior to applying it to Ω we can spread out the singular spectrum of our matrix without effecting the direction of the eigenvectors. Typically a small power of q (say 2) suffices. In this scenario it is also beneficial to increase our oversampling factor p to ensure sufficient coverage. The following algorithm solves the Fixed Rank problem for slowly decaying singular values

Algorithm 2: Fixed Rank Random SVD for Slowly Decaying Singular Spectrum

Result: Approximate rank $k+p$ factorization $A \approx \tilde{U}\tilde{\Sigma}\tilde{V}^T$

Input: matrix $A \in \mathbb{R}^{n\times n}$, target rank $k > 0$, oversampling factor $p > 0$, exponent $q \geq 1$

1. Draw $\Omega \in \mathbb{R}^{n\times(k+p)}$ i.i.d. samples from the standard normal distribution and format as a matrix.
2. Compute the product $Y = (AA^T)^q A\Omega$.
3. Compute a matrix $Q \in \mathbb{R}^{n\times(k+p)}$ whose columns form an orthonormal basis for the range of Y.
4. Compute the product $B = Q^T A$.
5. Compute the reduced SVD $B = U_B \Sigma_B V_B^T$ with $U_B \in \mathbb{R}^{n\times(k+p)}$, $\Sigma_B \in \mathbb{R}^{(k+p)\times(k+p)}$, and $V_B^T \in \mathbb{R}^{(k+p)\times(k+p)}$.
6. Set $\tilde{U} = QU_B$, $\tilde{\Sigma} = \Sigma_B$, $\tilde{V} = V_B$.

return $\tilde{U}, \tilde{\Sigma}, \tilde{V}^T$

In practice to ensure stability and high accuracy, orthonormalization is required between every application of A or A^T. In step 3 the Rank-Revealing QR can yield highly accurate results. This modification of Algorithm 1 will vastly outperform it when the singular spectrum of A decays slowly and is thus suited for more general application.

In the following section we will flush out the computational costs of both of these methods. Significantly the second is not significantly more costly than the first.

3 Complexity

In our complexity analysis we will ignore the cost of sampling from the normal distribution. In general it is approximately some constant times $n(k+p)$, but this number is difficult to pin down due to the variety of sampling techniques. Depending on the structure of the given operator or matrix, matrix multiplication can have vastly different costs as well. Rather than assuming we define the cost for a matrix vector multiplication with A as τ_A. The Random SVD (Algorithm 1) has the following complexity:

- $\tau_A(k+p)$ cost for multiplication $A\Omega$.
- $\mathcal{O}(n(k+p)^2)$ cost for factorization $Y = QR$.
- $\tau_{A^T}(k+p)$ cost for multiplication $Q^T A$. Often this is the same as $\tau_A(k+p)$.
- $\mathcal{O}(n(k+p)^2)$ cost for SVD.
- $\mathcal{O}(n(k+p)^2)$ cost for the matrix-matrix multiplication QU_B

This brings us to the total cost of $\tau_A(k+p) + \mathcal{O}(n(k+p)^2)$. By a large margin, the most significant cost in this method is the product $A\Omega$ in step 2. In practice this cost can be lowered, particularly if A is structured or sparse, i.e. matrices for which matrix-vector products can be evaluated rapidly. However, even without sparsity or fast matrix product schemes we can decrease the cost of a matrix product by inducing structure in Ω [3]. For instance, a good choice is to sample Ω from the Discrete Fourier

Transform Matrix (DFT)

$$\Omega = \sqrt{\frac{n}{k+p}} DFR \tag{1}$$

where D is signed $n \times n$ identity matrix with uniformly random signs, $F \in \mathbb{R}^{n \times n}$ is the DFT matrix, and $R \in \mathbb{R}^{n \times k+p}$ is a sub-sampling matrix with one nonzero entry per column. This decreases the cost of $A\Omega$ to $\mathcal{O}(n^2 \log(k+p))$ from the traditional $\mathcal{O}(n^2(k+p))$, and can decrease the cost of the QR as well [3,6].

In algorithm 2, The Random SVD of A with Slowly Decreasing Singular Spectrum, there is an additional cost associated with taking the product $(AA^T)^q A$. Through a similar complexity analysis the total cost of this algorithm is $(2q+1)(k+p)\tau_A + \mathcal{O}(n(k+p)^2)$. For small q this is not a burdensome additonal cost to the original algorithm, particularly when considering the increase in accuracy.

4 Error Bounds

We now present two theorems for the approximation error of this routine. They both measure the error in the approximation of QQ^TA to A, which is the primary source of error in this method.

Theorem 1 (Expected Accuracy)**.** *In expectation, the accuracy of the approximation for $k, p \geq 2$ is given by*

$$\mathbb{E}\left[\left\|(I - QQ^T)A\right\|_2\right] \leq \left[1 + \frac{4\sqrt{(k+p)n}}{p-1}\right] \sigma_{k+1} \tag{2}$$

Theorem 2. *With probability $1 - 6p^{-p}$*

$$\left\|(I - QQ^T)A\right\|_2 \leq \left[1 + 11\sqrt{(k+p)n}\right] \sigma_{k+1} \tag{3}$$

These two error bounds show that in expectation, or with high probability, that the Random SVD can perform with near optimal accuracy(optimal being $\leq \sigma_{k+1}$). These bounds are fairly tight; in practice the random SVD performs very well near the expected outcome [3].

4.1 Notation

In the proof of theorem 1 we will need the following notation.

$$A = U \begin{bmatrix} \Sigma_1 & \\ & \Sigma_2 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} \tag{4}$$

In this decomposition the matrix Σ_1 is $k \times k$, Σ_2 is $(n-k) \times (n-k)$, V_1^T is $k \times n$ and V_2^T is $(n-k) \times n$. We also will set $\Omega_i = V_i^T \Omega$, and $Y = A\Omega$ where the entries of Ω are from $N(0,1)$. The matrix P_Y will be the orthogonal projector onto the range of Y. Orthogonal projectors are unique so, in this case, $P_Y = QQ^T$.

4.2 Partial Proof of theorem 1

For this theorem we assume that Ω_1 is full row rank. This assumption is rather mild and can be justified by the fact that Gaussian random matrices have probability zero of being singular. Our goal will be first to show that

$$\|(I - P_Y)A\|_2^2 \leq \|\Sigma_2\|_2^2 + \left\|\Sigma_2\Omega_2\Omega_1^\dagger\right\|_2^2 \tag{5}$$

If we define $\tilde{A} = U^T A$ and let $\tilde{Y} = \tilde{A}\Omega$ we can put our bound in terms of $\tilde{A}$

$$\|(I - P_Y)A\|_2 = \left\|U^T(I - P_Y)U\tilde{A}\right\|_2 = \left\|(I - P_{\tilde{Y}})\tilde{A}\right\|_2 \tag{6}$$

This first equality holds because our norm is unitarily invariant. Meanwhile the second equality substitutes the orthogonal projector $P_{\tilde{Y}} = U^T P_Y U$ [3]. Now we do a tricky transformation of $\tilde{Y}$. Set $Z = \tilde{Y}\Omega_1^\dagger\Sigma^{-1} = [I\ F]^T$ where $F = \Sigma_2\Omega_2\Omega_1^\dagger\Sigma_1^{-1}$. With this transformation $range(Z) \subseteq range(\tilde{Y})$ which implies that

$$\left\|(I - P_{\tilde{Y}})\tilde{A}\right\|_2^2 \leq \left\|\tilde{A}^T(I - P_Z)\tilde{A}\right\|_2 = \left\|V\Sigma(I - P_Z)\Sigma V^T\right\|_2 = \|\Sigma(I - P_Z)\Sigma\|_2 \tag{7}$$

This form of the bound is far easier to analyze than our original form. Now if we expand the form of the orthogonal projector P_Z we can bound it by a positive definite matrix. The matrix P_Z has the expansion

$$P_Z = \begin{bmatrix} I \\ F \end{bmatrix} (I + F^T F)^{-1} \begin{bmatrix} I \\ F \end{bmatrix}^T \tag{8}$$

As P_Z is a spectral projector, the difference $I - P_Z$ is positive definite. Furthermore this difference can be bounded by

$$0 \prec I - P_Z \preceq \begin{bmatrix} F^T F & B \\ B^T & I \end{bmatrix} \tag{9}$$

where $B = -(I + F^T F)^{-1} F^T$. Multiplying both sides by Σ we get

$$\Sigma(I - P_Z)\Sigma \preceq \begin{bmatrix} \Sigma_1 F^T F \Sigma_1 & \Sigma_1 B \Sigma_2 \\ \Sigma_2 B^T \Sigma_1 & \Sigma_2\Sigma_2 \end{bmatrix} \tag{10}$$

Finally we can use this fact to complete our bound

$$\begin{aligned} \|(I - P_Y)A\|_2^2 &= \left\|(I - P_{\tilde{Y}})\tilde{A}\right\|_2^2 && (11)\\ &\leq \|\Sigma(I - P_Z)\Sigma\|_2 && (12)\\ &\leq \left\|\Sigma_1 F^T F \Sigma_1\right\|_2 + \left\|\Sigma_2^2\right\|_2 && (13)\\ &\leq \left\|\Sigma_2\Omega_2\Omega_1^\dagger\right\|_2^2 + \|\Sigma_2\|_2^2 && (14) \end{aligned}$$

This is the deterministic bound that we were looking for. The above statement holds for any matrix Ω. In the next lecture we will finish the proof by analyzing the case when the entries of Ω are drawn from a standard normal distribution. Together these two results culminate to Theorem 1.

5 Code packages

Matlab, Julia, Python and a few other major programming languages have practical implementations of the Random SVD. In Matlab, the Random SVD can be found on the file exchange [4]. The code is an implementation of Algorithm 1, so it does not include *any* of the robust improvements, i.e. choice of p, exponentiation of A, structured Ω. Thus this code is useful for some basic tests, but is not recommended for general application. The code is as follows

```
function [U,S,V] = rsvd(A,K)
%----------------------------------------------------------------------------
% random SVD
% Extremely fast computation of the truncated Singular Value Decomposition, using
% randomized algorithms as described in Halko et al. 'finding structure with
   randomness
%
% usage :
%
%  input:
%  * A : matrix whose SVD we want
%  * K : number of components to keep
%
%  output:
%  * U,S,V : classical output as the builtin svd matlab function
%----------------------------------------------------------------------------
% Antoine Liutkus  (c) Inria 2014

[M,N] = size(A);
P = min(2*K,N);
X = randn(N,P);
Y = A*X;
W1 = orth(Y);
B = W1'*A;
[W2,S,V] = svd(B,'econ');
U = W1*W2;
K=min(K,size(U,2));
U = U(:,1:K);
S = S(1:K,1:K);
V=V(:,1:K);
```

Scikitlearn has developed the Python implementation with a larger suite of adaptive sub-methods such as the Power Iteration Normalizer, and Randomized Range Fider [3, 5]. This implementaion is more robust than Matlab's and easy to use.

```
sklearn.utils.extmath.randomized_svd(A)
```

Julia has implemented nearly all of the algorithms from [3] with all of the bells and whistles in the IterativeSolvers.jl package [1]. The recommended algorithm call is

```
rsvd_fnkz(A, k)
```

By far Julia has the most sophisticated implementation of the three softwares.

6 Concluding Remarks

The Random SVD is a powerful algorithm which uses randomness to compute the approximate SVD of large matrices. The speed increase from this method is substantial, especially when considering the minor decrease in accuracy. Prior to using it we direct the reader to the friendly, yet thorough, review article [3]. This should provide a more complete lesson to all of the extensions that will supercharge your implementation, and bring insight to application specific decisions.

References

[1] Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B Shah. Julia: A fresh approach to numerical computing. *SIAM review*, 59(1):65–98, 2017.

[2] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Matrix Computations. Johns Hopkins University Press, 2012.

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

[4] Antoine Liutkus. Matlab central file exchange, randomized singular value decomposition. Retrieved February 5, 2020.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[6] Franco Woolfe, Edo Liberty, Vladimir Rokhlin, and Mark Tygert. A fast randomized algorithm for the approximation of matrices. *Applied and Computational Harmonic Analysis*, 25(3):335 – 366, 2008.