MATH 690: Topics in Probablity Theory November 9, 2017

Fast Randomized SVD

Lecturer: Xiuyuan Cheng *Scribe: Rohith Kuditipudi*

1 Introduction

1.1 Motivation

Many modern applications of machine learning require computations over very large matrices (e.g. Laplacian of a social network graph, gene expression arrays, etc.)—so large in fact that even storing them in memory can be non-trivial. Fortunately however, in many cases these matrices are either sparse (e.g. a typical person is friends with a very small fraction of the world's population) or otherwise endowed with some structure (e.g. genes in particular pathways tend to be co-expressed) that enables them to be significantly compressed.

One approach that has proven particulary effective for compressing a large matrix is decomposing the matrix as the product of two (or more) smaller matrices. In particular, suppose A is an $m \times n$ matrix that can be approximated as the product of an $m \times k$ matrix B and an $k \times n$ matrix C, where $k << m, n$. Then instead of naively requiring $O(mn)$ memory to store A, we need only use $O(km + kn)$ memory.

As an additional benefit, such *low-rank decompositions* often capture useful structural information about the original matrix and therefore tend to be useful for purposes beyond compression. For instance, suppose some entries of A are missing. In principle, a missing entry a_{ij} can be recovered as the product of the i^{th} row of B and the j^{th} column of C.

1.2 Problem Statement

The goal of this lecture will be to develop a fast algorithm for computing low-rank decompositions of large matrices. Specifically, given an $m \times n$ matrix A as input, we desire an algorithm that yields a rank-k factorization $A_{m\times n} = B_{m\times k} \cdot C_{k\times n}$ for which $\|BC - A\|_2$ is sufficiently small.

1.3 Overview of Approach

Our approach can be condensed to the following steps:

1. Given $A_{m\times n}$ as input, draw k' samples from the range of A
   - speciically, let $Y_{m\times k'} = A\Omega_{n\times k'}$ where $k' \approx k$ and Ω_{ij} are drawn i.i.d. $\mathcal{N}(0, 1)$
2. Compute an orthonormal basis $Q_{m\times k'}$ for Y (e.g. via QR-decomposition)
3. Compute the singular value decomposition of $Q^T A$ and through a change of basis obtain an approximately rank k factorization of A
   - specifically, if $Q^T A = U\Sigma V^T$ then it follows that $A \approx QU\Sigma V^T$

So long as Y captures most of the action of A, the low-rank factorization output by our algorithm will be a good approximation of A. Computationally, the most expensive step in our algorithm is computing the product $Y = A\Omega$. As we will see later however, in practice the computation of this product can be sped up dramatically (while still preserving the quality of Y) by imposing some additional structure on Ω.

2 Preliminaries

2.1 Properties of tall Gaussian matrices

The only source of randomness in our algorithm is the tall Gaussian matrix Ω (tall in the sense that $n \geq k'$). The focus of this section will be to establish properties of tall Gaussian matrices that will allow us to later establish error bounds on the quality of the low-rank factorization output by our algorithm. In particular, we want to demonstrate that tall Gaussian matrices tend to be well-conditioned.

Definition 1. *Let $G_{n\times k}$, where $n \geq k$, be a matrix whose elements are drawn i.i.d. $\sim \mathcal{N}(0,1)$. Then G is a* ***tall Gaussian matrix****.*

Definition 2. *Let $\{\sigma_1, ..., \sigma_k\}$ denote the singular values of G. Then $\kappa(G) = \frac{\sigma_1}{\sigma_k}$ is the* ***condition number*** *of G.*

Recall that in a previous lecture (11/7), we discussed the asymptotic behavior of σ_1 and σ_k as n and k grow large. In particular, as a consequence of the Marchenko-Pastur Law (presented in 9/7's lecture), we have that $\sigma_1 \approx \sqrt{n} + \sqrt{k}$ and $\sigma_k \approx \sqrt{n} - \sqrt{k}$. As it turns out, we can actually obtain a non-asymptotic bound on the expectations of σ_1 and σ_k.

Theorem 1. *(Gordon's Theorem) Let $G_{n\times k}$ be a tall Gaussian matrix with singular values $\{\sigma_1, ..., \sigma_k\}$. Then*

$$\sqrt{n} - \sqrt{k} \leq \mathbb{E}\sigma_k \leq \mathbb{E}\sigma_1 \leq \sqrt{n} + \sqrt{k}$$

From the fact that $\sigma_1, ..., \sigma_k$ are all 1-Lipschitz functions of G (see Corollary 8.6.2 from [4]), we can combine a concentration argument with Gordon's theorem to obtain the following bound:

Corollary 1. *(Proposition 10.3 from [1]) Let G be defined as in Theorem 1. Then for $1 \leq i \leq k$,*

$$Pr(|\sigma_i - \mathbb{E}\sigma_i| > \alpha) \leq e^{-\frac{\alpha^2}{2}}$$

And so with high probability for $t_1, t_2 = O(1)$, we have that

$$\sqrt{n} - \sqrt{k} - t_1 \leq \sigma_k \leq \sigma_1 \leq \sqrt{n} + \sqrt{k} + t_2$$

It immediately follows that with high probability,

$$\kappa(G) \leq \frac{\sqrt{n} + \sqrt{k} + t_2}{\sqrt{n} - \sqrt{k} - t_1}$$

And so as long as $n >> k$, the condition number of G will be close to 1.

2.2 Why Oversample?

The focus of Section 2.1 was presenting properties of Gaussian matrices that later on will allow us to draw conclusions about the quality of Y (and thus the quality of the orthonormal basis Q obtained for Y). Recall that once we obtain Q, we deterministically compute the singular value decomposition of $Q^T A$, from which we obtain a low-rank decomposition of our original matrix A through left-multipying by Q.

At this point, an attentive reader might have noticed that although we purportedly want a rank k decomposition of A, the actual output of our algorithm is a rank k' decomposition. As illustrated by the following theorem, the reason we oversample from the range of A is to ensure that Y captures enough of the action of A so as to make $QQ^T A$ a meaningful approximation of A. We will elaborate on this point in Section 3. Since in any case we assume $k << n$, in practice a rank k' decomposition is just as good as a rank k decomposition so long as $k' \approx k$ we

Theorem 2. *(Theorem 1.1 from [1]) Let A and Q be defined as in Section 1.3, and suppose $k' = k + p$. Denote the singular values of A as $\{\sigma_1, ..., \sigma_n\}$. Then*

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

3 Performance Analysis

The focus of this section will be to develop bounds on the performance and runtime of our algorithm.

3.1 How good is the best possible decomposition?

Unless A itself is of rank k or lower, finding a perfect rank k decomposition of A is impossible. In fact, the best we can do is decompose A as the sum of the first k components of its singular value decomposition.

Theorem 3. *(Eckart-Young-Mirsky Theorem) Let $A_{m\times n} = U\Sigma V^T$ denote the singular value decomposition of A, with left singular vectors $\{u_1, ..., u_n\}$, right singular vectors $\{v_1, ..., v_n\}$ and singular values $\{\sigma_1, ..., \sigma_n\}$. Then with respect to both $\|\cdot\|_2$ and $\|\cdot\|_F$, the best rank k decomposition of A is given by*

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T$$

Note that $\|A_k - A\|_2 = \sigma_{k+1}$, implying that σ_{k+1} lower bounds the performance of our algorithm (recall from Section 1.2 that we are using $\|\cdot\|_2$ error as our performance metric).

3.2 How good is our decomposition?

Theorem 3 gives us a recipe for when our algorithm performs best. In particular, if we so happen to get incredibly lucky and draw Ω such that the columns of Ω are precisely the first k right singular vectors $\{v_i\}_{i=1}^k$ of A, then the output of our algorithm will be A_k. To the extent that Range(Ω) is

a poor approximation of the span of these k vectors, the performance of our algorithm will be suboptimal.

But what does it really mean for Range(Ω) to be a "good approximation" of the span of $\{v_i\}_{i=1}^k$? One heuristic is the condition number of $[v_1, ..., v_k]^T\Omega$, where $[v_1, ..., v_k]$ denotes the $n \times k$ matrix with columns $\{v_i\}_{i=1}^k$. In particular, a condition number close to 1 implies that $[v_1, ..., v_k]^T\Omega$ is of full rank, and moreover that none of the first k right singular vectors of A are orthogonal (or even close to orthogonal) to Range(Ω).

Theorem 4. *(Theorem 9.1 from [1]) Let A, Ω, Y and Q be defined as in the description of our algorithm (see Section 1.3). Furthermore, let $A = U\Sigma V^T$ denote the singular value decomposition of A. We can partition this decomposition as follows:*

$$A = U \begin{array}{c} \begin{matrix} k & n-k \end{matrix} \\ \begin{bmatrix} \Sigma_1 & \\ & \Sigma_2 \end{bmatrix} \end{array} \begin{array}{c} \begin{matrix} n \end{matrix} \\ \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix} \end{array} \begin{array}{l} k \\ n-k \end{array}$$

Define Ω_1 and Ω_2 such that $\Omega_1 = V_1^T\Omega$ and $\Omega_2 = V_2^T\Omega$. Then so long as Ω_1 has full row-rank,

$$\|A - QQ^TA\|_2^2 \leq \|\Sigma_2\|_2^2 + \|\Sigma_2\Omega_2\Omega_1^+\|_2^2$$

where Ω_1^+ denotes the pseudoinverse of Ω_1.

Theorem 4 formalizes our intuition that our algorithm performs well when $V_1^T\Omega$ is well-conditioned. Of course, this begs the question: can we actually expect $V_1^T\Omega$ to be well-conditioned? It turns out we can. In particular, from the fact that Gaussian matrices are rotationally invariant, it follows that if the elements of Ω are i.i.d $\sim \mathcal{N}(0, 1)$ then so are the elements of $V_1^T\Omega$ (recall that the columns of V_1 are orthonormal). Moreover, $V^T\Omega$ is "fat" in the sense that it is a $k \times k'$ matrix, where $k' > k$ (recall that we chose to oversample from the range of A). Thus, we can adapt the results for "tall" Gaussian matrices presented Section 2.1 to obtain that $V_1^T\Omega$, a "fat" Gaussian matrix, is also well-conditioned. Finally, recall that our algorithm returns the exact singular value decomposition of QQ^TA. Thus, the performance of our algorithm is a direct function of $\|A - QQ^TA\|$.

Theorem 5. *(presented in lecture) Let $QQ^TA = U\Sigma V^T$ denote the singular value decomposition of QQ^TA. Then*

$$\mathbb{E}\|A - U\Sigma V^T\|_2 = \mathbb{E}\|A - QQ^TA\|_2 \leq \left(1 + 4\sqrt{\frac{2n}{k-1}}\right)\sigma_{k+1}(A)$$

where $\sigma_{k+1}(A)$ denotes the $(k+1)^{st}$ singular value of A.

Theorem 5 gives us another nice bound on the performance of our algorithm—or more specifically, the expected performance. Recall from Section 3.1 that in terms of $\|\cdot\|_2$ error, no rank k decomposition of A can do better than $\sigma_{k+1}(A)$. Theorem 5 tells us that if the singular values of A decay nicely (i.e. $\sigma_{k+1}(A) \leq \epsilon$), then the rank k decomposition produced by our algorithm will be a pretty good approximation of A. It turns out that Theorem 5 can be combined with a concentration argument to obtain the existence of a constant c such that $\|A - U\Sigma V^T\|_2 \leq c\left(1 + 4\sqrt{\frac{2n}{k-1}}\right)\sigma_{k+1}(A)$ with high probability.

A complete proof of Theorem 5 (and its corresponding concentration result) is beyond the scope of this lecture. Curious readers are invited to refer to Part III of [1] for an in-depth presentation.

3.3 Runtime Analysis

Compared to returning the first k components of the singular value decomposition of A, our algorithm will almost certainly produce a less accurate decomposition. However, what we get in return is a faster runtime.

Our algorithm can be thought of as a two-stage process, where the output of the first stage is Q and the output of the second stage is the singular value decomposition of QQ^TA. Whereas the classical complexity of computing the singular value decomposition of an $m \times n$ matrix is mn^2, the standard (naive) approach to multiplying an $l \times m$ matrix with an $m \times n$ matrix takes $O(lmn)$ time. And so because in the second stage we need only actually compute the singular value decomposition of the $k' \times n$ matrix Q^TA, which as it turns out can be done without actually having to explicitly compute Q^TA, the overall runtime of our algorithm is dominated by the computation of the matrix product $A\Omega$ in the first stage of the algorithm.

The naive complexity of computing $A\Omega$ is $O(kmn)$, which in fact is the same as the classical complexity of directly computing the first k components of the singular value decomposition of A. However, it turns out that in practice, Ω does not strictly have to be a Gaussian random matrix; so long as Ω is "random enough", the performance of our algorithm won't suffer much. Consequently, through imposing some mild structure on Ω, we can not only maintain the performance of our algorithm but also dramatically improve its complexity.

One approach for constructing Ω that works particularly well is to randomly subsample and scale the discrete Fourier transform, making Ω what is known as a *Subsampled Random Fourier Transform*, or SRFT for short. Specifically, we can sample Ω according to $\Omega \sim D_{n\times n}F_{n\times n}T_{n\times k'}$, where D is a random diagonal matrix, F is a Fourier matrix and T is a column selection matrix (for specifics, see Section 4.6 and Algorithm 4.5 in [1]). By defining Ω as the product of these three matrices, we can speed up the computation of $A\Omega$ to a complexity of $O(mn \log k)$.

TABLE 7.1
Computational times for a partial SVD. *The time, in seconds, required to compute the ℓ leading components in the SVD of an $n \times n$ matrix using each of the methods from §7.4. The last row indicates the time needed to obtain a full SVD.*

	$n = 1024$			$n = 2048$			$n = 4096$		
ℓ	direct	gauss	srft	direct	gauss	srft	direct	gauss	srft
10	1.08e-1	5.63e-2	9.06e-2	4.22e-1	2.16e-1	3.56e-1	1.70e 0	8.94e-1	1.45e 0
20	1.97e-1	9.69e-2	1.03e-1	7.67e-1	3.69e-1	3.89e-1	3.07e 0	1.44e 0	1.53e 0
40	3.91e-1	1.84e-1	1.27e-1	1.50e 0	6.69e-1	4.33e-1	6.03e 0	2.64e 0	1.63e 0
80	7.84e-1	4.00e-1	2.19e-1	3.04e 0	1.43e 0	6.64e-1	1.20e 1	5.43e 0	2.08e 0
160	1.70e 0	9.92e-1	6.92e-1	6.36e 0	3.36e 0	1.61e 0	2.46e 1	1.16e 1	3.94e 0
320	3.89e 0	2.65e 0	2.98e 0	1.34e 1	7.45e 0	5.87e 0	5.00e 1	2.41e 1	1.21e 1
640	1.03e 1	8.75e 0	1.81e 1	3.14e 1	2.13e 1	2.99e 1	1.06e 2	5.80e 1	5.35e 1
1280	—	—	—	7.97e 1	6.69e 1	3.13e 2	2.40e 2	1.68e 2	4.03e 2
svd	1.19e 1			8.77e 1			6.90e 2		

Figure 1: Table 7.1 from [1] provides a fairly extensive empirical comparison of the runtimes of three algorithms for computing low-rank decompositions—"direct" refers to deterministically compute the first l components of the exact SVD, "gauss" corresponds to running our algorithm as described in Section 1.3, and "srft" is the same as "gauss" but with Ω drawn as an SRFT instead of a Gaussian random matrix.

Thus, the overall complexity of our algorithm is roughly $O(mn \log k)$[1], which compares favorably with the $O(mnk)$ complexity of explicitly computing the first k components of the singular

[1]Technically the true complexity is $O(mn \log k + (m+n)k^2)$, but since it's typically the case that $k << n, m$, the second term ends up not being that important.

value decomposition of A. Another benefit of our algorithm, even if we naively draw Ω as a random Gaussian matrix, is that it is more amenable to parallelization since the primary bottleneck is the computation of a matrix product (i.e. $A\Omega$). For more details on optimizing the runtime of the algorithm, refer to [2] and Part II of [1].

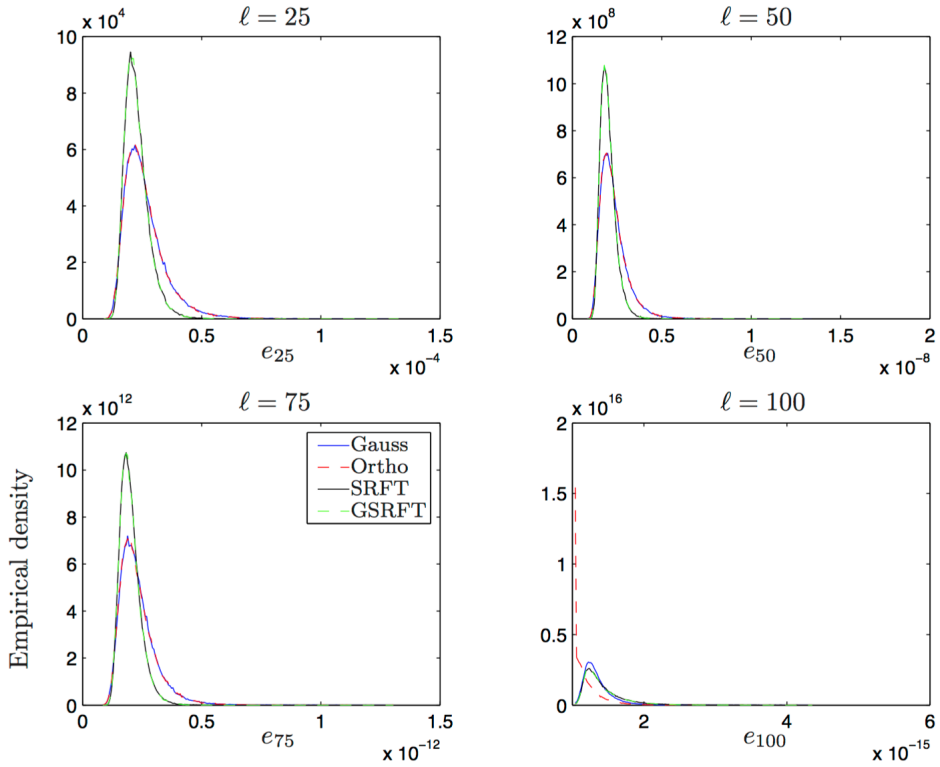

FIG. 7.8. Empirical probability density functions for the error in Algorithm 4.1. *As described in §7.4, the algorithm is implemented with four distributions for the random test matrix and used to approximate the* 200×200 *input matrix obtained by discretizing the integral operator* (7.1). *The four panels capture the empirical error distribution for each version of the algorithm at the moment when* $\ell = 25, 50, 75, 100$ *random samples have been drawn.*

Figure 2: Figure 7.8 from [1] illustrates the typical error obtained by our algorithm given different distributions for Ω—"Gauss" refers to the standard Gaussian distribution, "Ortho" to the uniform distribution on orthonormal matrices, "SRFT" to the SRFT distribution and "GSRFT" to a slightly modified version of the SRFT distribution. The key takeaway is that Ω need not necessarily be a Gaussian random matrix in order for our algorithm to perform well.

References

[1] Halko, Nathan, Per-Gunnar Martinsson, and Joel A. Tropp. "Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions." SIAM review 53.2 (2011): 217-288.

[2] Woolfe, Franco, et al. "A fast randomized algorithm for the approximation of matrices." Applied and Computational Harmonic Analysis 25.3 (2008): 335-366.

[3] Martinsson, Gunnar. "Enabling very large-scale matrix computations via randomization." https://amath.colorado.edu/faculty/martinss/Talks/2010_banff.pdf

[4] Golub, Gene H. and Charles F. Van Loan. "Matrix Computations" (3rd ed.). *Johns Hopkins University Press* (1996).