Stat260/CS294: Randomized Algorithms for Matrices and Data

Lecture 19 - 11/06/2013

Lecture 19: Randomized Low-rank Approximation in Practice, Cont.

Lecturer: Michael Mahoney *Scribe: Michael Mahoney*

Warning: these notes are still very rough. They provide more details on what we discussed in class, but there may still be some errors, incomplete/imprecise statements, etc. in them.

19 Randomized Low-rank Approximation in Practice, Cont.

We continue with the discussion from last time. There is no new reading, just the same as last class.

In particular, today we will cover the following topics.

- Practical random projection algorithms.

19.1 Practical Random Projections

Let's revisit random projections in light of the main structural result for low rank approximation. We will be particularly interested in modifications to what we have been discussing to make random projection algorithms implementable. As a trivial but important point, we can't write code to perform a loop over a dummy variable from 1 to $O\left(k\log(k)/\epsilon^2\right)$, where the constant in the big-O is left unspecified, and we don't want to choose the upper index arbitrarily. Also, in practice, we don't want to be quite so cavalier about constant factors as we typically are in theory. As it turns out, the same structural result that is used in the CSSP can also be used to parameterize random projection algorithms in a manner that make them more easily implementable, and it also provides finer control over several other issues of interest in practice. We turn to that now.

Recall that random projections can be used to get additive-error low-rank approximations. At root, this was accomplished by applying subspace JL or approximate matrix multiplication ideas to the columns/rows of A. Relatedly, observe that random projections rotate the input data to a random basis where the squared-norms of the columns are approximately uniform, and thus where uniform sampling can be used. We also saw that by using a more sophisticated importance sampling distribution the random sampling algorithms can lead to relative-error guarantees, when parameters are chosen appropriately. In addition, since it was in the homework, we didn't go through the details in class, but random projection algorithms also lead to relative-error low-rank approximation. If you recall, at root this was accomplished by applying subspace JL or approximate matrix multiplication ideas to the rows of the truncated singular vectors. Thus, both sampling and projection algorithms can be improved from additive-error to relative-error—the former are improved by improving the algorithm, and the latter are improved by improving the analysis. In particular, the latter happened since random projections uniformize a lot of things (both row norms squared as well as leverage scores), and thus the original analysis was weak.

Given this, one might wonder whether random projections uniformize other things related to the basic structural result underlying low-rank matrix approximation. Not unrelatedly, in NLA and scientific computing, one often thinks of ϵ as machine precision, and one is more interested in obtaining results with respect to the spectral norm than with respect to the Frobenius norm or trace norm, and one often uses iterative algorithms to accomlish this. As with iterative RandNLA LS solvers, here too for low-rank matrix approximation, we can couple basic random projections with traditional numerical methods to lead to good implementations. There are several other related issues that we will discuss today, but the best implementations exploit the basic structural result in important ways. We will review this here.

Most high-quality implementations for low-rank matrix approximation have been done in the context of scientific computing, where the end goal is to obtain a basis. The reason for that is that, once one has an exact or approximate basis Q for the range space of the matrix (think: operator), one can do a lot of other things of interest with it. (It is really still unresolved whether this will be as useful a primitive in building a foundation of linear algebraic code for machine learning and data analysis applications, since in many of those applications the requirements are very different.) Here are several matrix decompositions of interest that can be computed with such a basis.

- Pivoted QR decomposition.
- Eigenvalue decomposition.
- SVD.

All of these deal with the (numerical) rank/range of a matrix. Often one uses a *truncated* form of these decompositions to get low-rank matrix approximations of the general form

$$\underbrace{A}_{m\times n} \approx \underbrace{B}_{m\times k}\underbrace{C}_{k\times n},$$

where k is the numerical rank. (Here, B and C are some matrices of the given dimension, with no necessary relationship to columns of rows or the way we used these letters before.) If $k \ll \min\{m,n\}$, then these decompositions allow A (essentially *all* of A, with no loss, since k is often chosen to be the the numerical rank) to be stored cheaply and/or to be multiplied quickly with other vectors/matrices.

One way these algorithms and decompositions are used in scientific computing is via a two step procedure as follows.

1. Step 1: Compute an approximate basis for the range of A, i.e., compute an orthogonal matrix A such that $A \approx QQ^TA$ (where, again, the notion of approximation captured by $\approx$ is often very strong, e.g., that they are the same up to numerical precision).
2. Step 2: Given this orthogonal matrix Q, use Q to compute QR, SVD, etc. for A.

As before with the LS problem, where we did one of two general approaches (use the sketch directly by solving a subproblem on the sketch, or use the sketch indirectly to construct a preconditioner to solve the original problem), so too here for the low rank approximation problem we can either work directly with the sketch (what we have been discussing so far) or use the sketch to couple to other traditional numerical algorithms (as we will describe now). Here are several differences to keep in mind.

- We will allow Q to have "extra" dimensions, i.e., if we want to have a target numerical rank of k, then we will allow Q to have $\ell = k + p$, where p is a small positive integer, columns, and we will not "filter" through the rank k space. This corresponds to the "easier" case we saw before of not filtering the low-rank approximation through an exactly rank k space. Importantly, though, keeping those extra dimensions will provide us with big gains in ways that aren't immediately obvious.
- Since we will call traditional iterative algorithms on fat matrices, we need to be more careful about how the top and bottom parts of the spectrum interact, i.e., how the subspaces corresponding to the top and bottom parts of the spectrum interact. In particular, this means that we need to go beyond just reproducing the error with respect to the top of the spectrum, instead taking into account the other criteria we discussed. The basic structural result will make this easier.

An issue we will discuss now is problem parameterization. (We saw an analogous issue of problem parameterization with the LS algorithms.)

- In TCS, it is more common to do a *fixed rank k approximation*, where one fixes the rank to be k and then asks for the best or a good approximation with respect to that value of k. For example: given a matrix $A \in \mathbb{R}^{m\times n}$, and numbers $k \in \mathbb{R}^+$ and $p \in \mathbb{Z}^+$, construct a matrix $Q \in \mathbb{R}^{m\times(k+p)}$ such that

  $$\left\|A - QQ^TA\right\|_\xi \approx \min_{\mathrm{rank}(B)\leq k} \left\|A - B\right\|_\xi ,$$

  where $\|\cdot\|_\xi$ is some norm, e.g., the spectral or Frobenius or trace norm. In TCS, k is assumed to be part of the input. In machine learning and data analysis applications, it is typically determined by some sort of model selection rule. In either case, there is no expectation that it captures the full numerical rank of the matrix.
- In NLA and scientific computing, it is more common to do a *fixed precision ϵ approximation*, where one fixed the acceptable value of the error, e.g., to be machine precision, and then chooses k to get below that error level. For example: given a matrix $A \in \mathbb{R}^{m\times n}$ and an error parameter $\epsilon > 0$, construct a matrix A consisting of $k = k(\epsilon)$ columns such that

  $$\left\|A - QQ^TA\right\|_\xi \leq \epsilon,$$

  where in these applications (especially if ϵ is set to machine precision) $\|\cdot\|_\xi$ refers typically to the spectral norm.

One way to deal with the difference is problem parameterization is to see that algorithms for the fixed rank approximation can often be adapted to solve the fixed precision approximation problem. Importantly, though, just as we saw with the LS problem, the precise form of the original worst-case bounds typically do *not* go through to the new problem parameterization, but it is typically the case that some variant of those bounds can be established.

That being said, the basic idea to convert a fixed rank approximation problem to a related fixed precision approximation problem is to build the basis Q incrementally (which explains the difficulty of obtaining worst-case bounds). Here is an example of such an algorithm. AlgorithmFixedRank takes as input A, k, and ℓ or p. Then, it does the following.

- Let Π be a random projection matrix consisting of Gaussian random variables.
- Let $C = A\Pi \in \mathbb{R}^{m\times\ell}$.
- Form $Q \in \mathbb{R}^{m\times\ell}$, an orthonormal basis for the span of C.

It is not essential that Π consist of Gaussian random variables, but among scientific computing implementations it is most common, so we have stated it that way. (All the expected consequences follow—it is worse on worst-case FLOPS, it may or may not be faster on particular matrices depending on the size, aspect ratio, pipelineing issues, etc.—but it is more convenient to work with when the matrix A is only implicitly represented and/or can be applied quickly to an arbitrary vector.) One reason it is common to use Gaussian random variables is that in very successful scientific computing applications of RandNLA, e.g., geophysics and certain areas of pdes, one only has an implicit representation of the matrix A, but that representation can be quickly applied to arbitrary vectors, and thus much of the benefit of input-sparsity-time or fast Hadamard-based projections is lost.

To make this work, one uses the fixed precision ϵ approximation, and thus one constructs the basis incrementally. HMT describe a probabilistic error estimator (that was introduced by LWMR, WLR, TR) that can be used as part of the incremental iteration. If the exact approximation error is $\left\|\left(I - QQ^T\right)A\right\|_2$, then the algorithm is the following.

- Draw a sequence of r $N(0,1)$ random vectors, call them $g^{(i)}$, where r is chosen empirically to balance the tradeoff between additional computational cost and the reliability of the estimator.

Then, one can establish the following lemma.

Lemma 1

$$\left\|I - QQ^TA\right\|_2 \leq 10\sqrt{\frac{2}{\pi}}\max_{i\in[r]}\left\|\left(I - QQ^T\right)Ag^{(i)}\right\|_2 .$$

Proof: The lemma follows easily from the following lemma.

Lemma 2 *Let $B \in \mathbb{R}^{m\times n}$, and fix $r \in \mathbb{Z}^+$ and $\alpha \in (0,1)$. If we draw $\{g^{(i)}, i \in [r]\}$ Gaussian vectors, then with probability $\geq 1 - \alpha^r$, we have that*

$$\|B\|_2 \leq \frac{1}{\alpha}\sqrt{\frac{2}{\pi}}\max_{i\in[r]}\left\|Bg^{(i)}\right\|_2$$

We won't go though the details of the proof of either of these.

$\diamond$

Here are several remarks.

- To do this error estimate requires a small number of additional matrix-vector products (which is often relatively cheap).
- In practice, the way this would be implemented is as follows: one would make an underestimate of the rank and then add more samples as necessary.

- BTW, I have not worked with this error estimator myself, and I have heard mixed reviews about it, in particular that it's variability might practically be too large for most applications of interest, especially outside scientific computing applications. So, if you want to use it, then look into how it performs for you.

The last comment aside, we could implement this probabilistic error estimator as part of the main algorithm. Here is the combined algorithm. AlgorithmCombined takes as input a matrix A and an error parameter ϵ, and it does the following steps.

1. Let $Q^{(0)}$ be an empty basis matrix.
2. For $i = 1, 2, 3, \ldots$, do the following.
   (a) Draw $n \times 1$ Gaussian random vector $g^{(i)}$ and set $y^{(i)} = Ag^{(i)}$.
   (b) Compute $\hat{q}^{(i)} = \left(I - Q^{(i-1)} Q^{(i-1)^T}\right) y^{(i)}$.
   (c) Normalize $q^{(i)} = \hat{q}^{(i)} / \left\|\hat{q}^{(i)}\right\|_2$ and form the matrix $Q^{(i)} = \left[Q^{(i-1)} q^{(i)}\right]$.

To determine when to stop, note that the vectors $\hat{q}^{(i)}$ are vectors that appear in the bound of Lemma 1, and so one stopping rule is to stop the look when the error $\epsilon' = \frac{\epsilon}{10\sqrt{2\pi}}$. (There are some numerical issues that we are ignoring, but this captures the main ideas.)

Recall that, in the motivating scientific computing applications where these implemented algorithms have been most fully developed, one wants to use the basis Q to construct other decompositions. Then, given a matrix Q of size $n \times (k + p)$, we want to get various decompositions with it. So, assume that we are given matrices B and C such that

$$\|A - BC\|_2 \leq \epsilon,$$

where $\text{rank}(B) = \text{rank}(C) = k$. For example, this could be gotten with a fixed precision variant of a random projection algorithm, as just described. Then, how can we use it to compute other factorizations, with comparable additional error? Here is how to do it.

- **Pivoted QR.** Given the matrix $A = \mathbb{R}^{m \times n}$, there is the decomposition $A = QR$, where $Q \in \mathbb{R}^{m \times \ell}$ is orthogonal, and R is—up to a permutation—an upper triangular matrix. In some cases, this process is stopped early, and we keep fewer than all the columns.

  To construct a partial QR decomposition, do the following.

  1. Compute QR factorization of C, i.e., $C = Q_1 R_1$.
  2. Form the product $D = R_1 B$, and compute the QR factorization of D, i.e., $D = Q_2 R$.
  3. Form the product $Q = Q_1 Q_2$.

  The result of this is an orthogonal Q and a matrix R that is—up to permutation—upper-triangular such that $\|A - QR\|_2 \leq \epsilon$.

- **SVD.** $A = U \Sigma V^T$. To construct a partial SVD, do the following.

  1. Compute QR factorization C such that $C = Q_1 R_1$.
  2. Form the product $D = R_1 B$, and do an SVD to get $D = U_2 \Sigma V^T$.

3. Form the product $U = Q_1 U_2$.

The result of this are matrices U, Σ, and V^T such that $\left\|A - U\Sigma V^T\right\|_2 \leq \epsilon$.

- **Interpolative Decomposition.** Given the matrix $A = \mathbb{R}^{m\times n}$, there is an index set $J = [j_1, \cdots, j_k]$ such that $A = A_{(:,J)}X$, with $X \in \mathbb{R}^{k\times n}$, where the $k \times n$ matrix X contains an $k \times k$ identity matrix I, i.e., $X_{(:,J)} = I$. Note that this is NP-hard to compute (I think, still to check), but there exist algorithms to compute a relaxation of it such that X has entries with magnitude bounded by 2 (and this leads to good conditioning properties). This interpolative decomposition can also be computed from $A \approx BC$, but we won't describe it here.

Another issue is that for some matrices the spectrum might decay rather slowly, and this can slow down Krylov-based iterative methods. In this case, RST suggested essentially to do a "power method" to enhance the decay of the spectrum. This reduces the weight of the small singular values, without changing the singular vectors and singular subspaces, and so it gets better convergence with iterative methods. So, we can apply random sampling/projection to $B = \left(AA^T\right)^q A$, observing that

$$B = \left(AA^T\right)^q A = U\Sigma^{2q+1}V^T.$$

Note that the matrix B has the same singular vectors as A, but its singular values decay faster as $\sigma_i(B) = \sigma_i(A)^{2q+1}$. So, this method requires doing more matrix-vector multiplications, but it is much more accurate. Thus, if the original basis is within a factor, call it γ, of optimum, then this gives a basis that is within a factor of $\gamma^{1/(2q+1)}$ of optimum, i.e, it is exponentially fast with the number of iterations.

We will analyze this below, and this will make use of our basic structural result in an important way. In particular, the main structural result suggests that the performance of algorithms depends on how the top and bottom part of the spectrum of A interact with the sketching matrix, in the way just described. Here is a lemma.

Lemma 3 *Let $A \in \mathbb{R}^{m\times n}$, and let $S \in \mathbb{R}^{n\times \ell}$ be a sampling matrix, and let $Z = BS$. Then,*

$$\left\|(I - P_Z)\, A\right\|_2 \leq \left\|(I - P_Z)\, B\right\|_2^{1/(2q+1)}.$$

Proof: We will skip the proof, but it is based on a variant of the spectral radius framework.

$\diamond$

To see how this lemma *applies* to the previous result, recall that

$$\left\|I - P_{AS}A\right\|_\xi \leq \left\|A - A_k\right\|_\xi + \left\|\Sigma_2\Omega_2\Omega_1^+\right\|_\xi,$$

where recall that $\Omega_2 = V_2^T S$ and $\Omega_1 = V_1^T S$. Thus,

$$\left\|I - P_{AS}A\right\|_\xi \leq \left(1 + \left\|\Omega_1\Omega_2^+\right\|_\xi\right)\sigma_{k+1},$$

where this is true for the spectral norm (still to check: is it true for other norms). But from the above lemma, we have that (and still to check: is it true for other norms):

$$\begin{aligned}
\left\|I - P_{BS}A\right\|_\xi &\leq \left\|I - P_{BS}B\right\|_\xi^{1/(2q+1)} \\
&\leq \left\|I + \Omega_2\Omega_1^+\right\|_\xi^{1/(2q+1)} \sigma_{k+1}^{1/(2q+1)}(B) \\
&= \left\|I + \Omega_2\Omega_1^+\right\|_\xi^{1/(2q+1)} \sigma_{k+1}(A)
\end{aligned}$$

So, in particular, the use of the power method drives down the sub-optimality of the additional error exponentially fast as the power q increases.

Finally, let's conclude with a comment about projecting onto extra dimensions and "keeping" them, i.e., not filtering them through the rank k space. Perhaps surprisingly, this is helpful to improve the failure probability. (This is somewhat different than with Blendenpik in the LS case.) In particular, it can be used to make the failure probability very small as a function of p, the number of extra samples/dimensions. For Gaussian projections, this holds true, and the result takes a particularly simple form, as is given in the following lemma.

Lemma 4 *Let $A \in \mathbb{R}^{m \times n}$, fix a rank parameter k, and let $p \geq 2$ be an oversampling factor. Then,*

$$\begin{aligned}
\mathbf{E}\left[\, \|(I - P_{AS})\, A\|_F \right] &\leq \left(1 + \frac{k}{p-1}\right)^{1/2} \|A - A_k\|_F \\
\mathbf{E}\left[\, \|(I - P_{AS})\, A\|_2 \right] &\leq \left(1 + \frac{k}{p-1}\right)^{1/2} \|A - A_k\|_2 + \frac{e\sqrt{k+p}}{p} \|A - A_k\|_F
\end{aligned}$$

Proof: The proof is omitted, but it uses the main structural result in an essential manner.

$\diamond$

Here are some final comments.

- The proof for this last result makes use of straightforward result, including the basic structural result, and the analysis is sufficiently fine that we can't get bounds for this by looking at just the top part of the spectrum. So, both the improved failure probability as well as the improved convergence rate rely on the same structural result that gave improved results for the CSSP.
- We can get similar bounds with the expectation removed, and there the failure probability decreases exponentially in p as the extra dimensions increases.
- This last result shows that there are several regimes of interest for p:
  - $p = 0$: this was in the CSSP
  - $p = 5$ or so, which is a modest oversampling which is used in random projections in practice.
  - $p \gtrsim k$, in which case the multiplicative factors start to become small
  - $p = \Theta\left(k \log(k)\right)$ or $p = \Theta\left(k \log(k)/\epsilon^2\right)$, which is the regime where the worst-case analysis is applied.