CS 6220: Data-sparse Matrix Computations Lecture 5: Least Squares Part 1

Author: Andrew Graven
ajg362

February 4, 2020

1 Wrapping Up The Randomized SVD

Recall the Randomized SVD Algorithm:

Algorithm 1 The Randomized SVD (Given $A \in \mathbb{R}^{n\times n}$, target rank k and oversampling parameter p)

1: Draw $\Omega \in \mathbb{R}^{n\times(k+p)}$ w/ iid $N(0,1)$ entries.
2: Set $Y = A\Omega$.
3: Compute $[Q, \sim] = qr(Y)$ (Reduced QR Factorization).
4: Set $B = Q^T A$.
5: Compute $B = U_B \Sigma_B V_B^T$ (Reduced SVD).
6: $U = QU_B$, $\Sigma = \Sigma_B$, $V = V_b$.
7: Return $[U, \Sigma, V]$.

$$\text{Randomized SVD Complexity} = O(T_{\text{mult}}(A)(k+p) + n(k+p)^2)$$

$$\text{Randomized SVD Error Bound: } E\|A - U\Sigma V^T\|_2 \leq \left[1 + \frac{4\sqrt{k+p}}{p-1}\sqrt{n}\right]\sigma_{k+1}$$

Side note: If we take $\gamma = k + p$ (the rank of Ω), then note that the above error bound implies the slightly stronger result:

$$E\|A - U\Sigma V^T\|_2 \leq \min_{k+p=\gamma;\ k>0,p>1}\left[1 + \frac{4\sqrt{k+p}}{p-1}\sqrt{n}\right]\sigma_{k+1} = \min_{k\in[0,\gamma-2]}\left[1 + \frac{4\sqrt{\gamma}}{\gamma-k-1}\sqrt{n}\right]\sigma_{k+1}$$

In particular, this suggests that in practice we might expect to do better than the standard error bound would indicate. For example, if the values were chosen so that $n = 1000$ $k = 10$, $p = 3$ and $\sigma_{11} = 1$, $\sigma_{12} = 1 \cdot 10^{-5}$ the naive error bound would predict:

$$E\|A - U\Sigma V^T\|_2 \leq \left[1 + \frac{4\sqrt{13}}{2}\sqrt{1000}\right] \approx 229$$

While a much tighter bound on the error can be shown by setting $k = 11$, $p = 2$:

$$E\|A - U\Sigma V^T\|_2 \leq \left[1 + \frac{4\sqrt{13}}{1}\sqrt{1000}\right] 1 \cdot 10^{-5} \approx 4.57 \cdot 10^{-3}$$

In particular, if there's a large jump in the singular values of A for some $k+1$ s.t. $k \in [0, \gamma - 2]$, then performance may be significantly better than expected.

A natural question to ask now would be: How can we improve the above error bound, perhaps while doing a bit more work? One option is to borrow an idea from power iteration: if A is applied to Ω several times, the range of the resulting Y will better approximate the range of A. Indeed, if we take an additional "iteration" parameter q and now let $Y = (AA^T)^q A\Omega$, the new algorithm becomes:

Algorithm 2 The Randomized SVD With $q-$Iteration
Given: $A \in \mathbb{R}^{n\times n}$, target rank k, oversampling parameter p and iteration parameter q

1: Draw $\Omega \in \mathbb{R}^{n\times(k+p)}$ w/ iid $N(0,1)$ entries.
2: Set $Y = (AA^T)^q A\Omega$.
3: Compute $[Q, \sim] = qr(Y)$ (Reduced QR Factorization).
4: Set $B = Q^T A$.
5: Compute $B = U_B \Sigma_B V_B^T$ (Reduced SVD).
6: $U = QU_B$, $\Sigma = \Sigma_B$, $V = V_b$.
7: Return $[U, \Sigma, V]$.

$$\text{Randomized SVD With q-Iteration Complexity} = O(T_{\text{mult}}(A)(k+p)q + n(k+p)^2)$$

$$\text{Randomized SVD With q-Iteration Error Bound: } E\|A - U\Sigma V^T\|_2 \leq \left[1 + \frac{\sqrt{k}}{p+1} + \frac{e\sqrt{k+p}}{p}\sqrt{n}\right]^{\frac{1}{\sqrt{2q+1}}} \sigma_{k+1}$$

Note that one major caveat of the $q-$Iteration method is that the conditioning of Y becomes progressively worse. That is, for Algorithm 1, $\mathcal{K}(Y) = \mathcal{K}(A\Omega) = \mathcal{K}(A)$. But for Algorithm 2, $\mathcal{K}(Y) = \mathcal{K}((AA^T)^q A\Omega) = \mathcal{K}(A)^{2q+1}$. A potential solution to this issue would be to add an orthogonalization step after each iteration.

Another note on the computational complexity of the Randomized SVD: One of the potentially dominating terms in the computational complexity of the SVD is the $T_{\text{mult}}(A)(k+p)$ term resulting from the application of A to Ω. In particular, if A isn't amenable to fast matrix-vector multiplication, then this term can be problematic. A potential solution to this problem is to change the choice of Ω s.t. $A\Omega$ can be computed more quickly. This, of course comes with the caveat that we lose many of the nice properties of Ω being chosen from an iid normal distribution - in particular, it makes clean error analysis more difficult.

There are many different choices of Ω that are used in practice, but one particularly popular choice is the "Sub-sampled Randomized Fourier Transform", where:

$$\Omega = DFS,\ D \in \mathbb{R}^{n\times n},\ F \in \mathbb{R}^{n\times n},\ S \in \mathbb{R}^{n\times(k+p)}$$

Where:
D = the identity matrix with random sign flips on the diagonal.
F = the Discrete Fourier Transform Matrix.
S = a random sub-sampling of the Fourier Coefficients (ie. exactly one non-zero entry per column chosen uniformly at random).
D and S can clearly be applied in $O(n)$ and $O(k+p)$ respectively because they each have at most one non-zero entry per column. Furthermore, the Discrete Fourier Transform Matrix can be applied in $O(n\log(n))$ time by applying the Discrete Fast Fourier Transform.

2 Introduction to Randomized Methods for Least Squares

Recall from the lectures on the Randomized SVD that $Q^T A$ had meaning to the extent that $\|A - QQ^T A\|_2$ was small. For least squares, instead we can think of $Q^T A$ as a dimension reduction on the column space of A. ie. $Q^T Ae_i$ (where e_i is some standard basis vector) is a low-dimensional $(k+p)-$dimensional embedding of Ae_i, containing some information about the structure of A.

- Question: Can we use an embedding of the column space of A which is agnostic to the structure of A?
- Definition: A matrix $G \in \mathbb{R}^{n\times k}$ is called an $\epsilon-$accurate subspace embedding for A if:

$$(1-\epsilon)\|Ax\|_2 \leq \|G^T Ax\|_2 \leq (1+\epsilon)\|Ax\|_2\ \forall x \in \mathbb{R}^n$$

- Question: When is having such a matrix useful?

2.1 Case 1: Least Squares

If $A \in \mathbb{R}^{m \times n}$, $m >> n$, and we want so solve:

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2^2$$

Then, if G is an $\epsilon-$accurate subspace embedding for A, we have that if $\|G^T(A\hat{x}-b)\|_2$ is small, then $\|A\hat{x}-b\|_2$ is also small with a factor of ϵ. In fact, one possible approach to solving such least squares methods is to use randomness to generate a right-preconditioner (see below for formal definition) which is simultaneously a "good" $\epsilon-$subspace embedding. Thus significantly reducing the cost per iteration, while simultaneously increasing the convergence rate - with some qualifications pertaining to the ϵ factor, and probabilistic bounds on the condition number of the new preconditioned problem. In fact, this is similar to the approach used by LSRN, outlined in section 3.2.

2.2 Case 2: Estimating Matrix-Matrix Products

Using G, we can approximate $A^T B \approx A^T G G^T B$.

- In fact, G can be chosen such that $E(A^T G G^T B) = A^T B$. From there, properties such as variance can be considered.

(For more on $\epsilon-$accurate subspace embeddings and matrix multiplication, see Woodruff[2])

3 Randomized Methods for Least Squares

Problem Statement: Given $A \in \mathbb{R}^{m \times n}$, $m >> n$, $b \in \mathbb{R}^m$, want to solve:

$$\min_{x \in \mathbb{R}^n} \frac{1}{2}\|Ax - b\|_2^2$$

(We assume that A has full column rank. Note that this can easily be extended to the case of column rank-deficient A, but the analysis is cleaner in the first case).

Two major randomized algorithms for solving randomized least squares:

1. "Blendnpik" (Avron, Maymounkov and Toledo)[1]
2. "LSRN" (Meng, Saunders and Mahoney)[3]

3.1 An Aside on CG (Conjugate Gradient Decent) and MINRES

Conjugate Gradient (CG) is a Krylov method for solving $Mx = b$ when M is symmetric positive definite (SPD). MINRES is another such algorithm. The primary difference between these two algorithms is that CG minimizes with respect to the norm defined by M, whereas MINRES minimizes residual error.

Mathematically, we can exactly solve:

$$\min_{x \in \mathbb{R}^n} \frac{1}{2}\|Ax - b\|_2^2$$

Using the normal equations:

$$A^T A x = A^T b$$

- Caveat: Use of the normal equations will typically require solving a more poorly conditioned problem than the original because $\mathcal{K}(A^T A) = \mathcal{K}(A)^2$.

Krylov methods for solving Least Squares, such as CG or MINRES, try to solve this problem without directly applying the normal equations. Several variants include:

- LSQR: Mathematically equivelent to CG on the normal equations, without explicitly using them, and thus avoiding many of the numerical issues posed by direct use of the normal equations.
- LSMR: Mathematically equivelent to running MINRES on the normal equations, without explicitly using them, and thus avoiding many of the numerical issues posed by direct use of the normal equations.

(Specifically, these methods avoid the issue mentioned above that $\mathcal{K}(A^TA) = \mathcal{K}(A)^2$).

3.2 LSRN

Big idea: LSRN is an iterative procedure utilizing preconditioning on LSQR to accelerate convergence. In exact arithmetic, LSQR has the following error bound at the kth iterate:

$$\frac{\|x^{(k)} - x^*\|_{A^TA}}{\|x^{(0)} - x^*\|_{A^TA}} \leq \left(\frac{\mathcal{K}(A) - 1}{\mathcal{K}(A) + 1}\right)^k$$

Where:

- $x^{(k)}$ is the kth iterate.
- $x*$ is the true solution.
- $\|x\|_{A^TA} = x^T(A^TA)x$, the norm induced by the inner product defined by A^TA.

There are two primary types of preconditioning for solving systems of the form $Bx = d$:

1. Left Preconditioning: Solve instead $M^TBx = M^Td$.
2. Right Preconditioning; Solve $BNy = d$, then take $x = Ny$.

These techniques can be extended to Least Squares problems (and thus algorithms such as LSRN):

- Theorem: If $x^* = Ny^*$, when y^* solves $\min_y \|ANy - b\|$, then if range(N) =range(A^T), then x^* is the solution to $\min_x \|Ax - b\|_2$.

The question then becomes: How do we construct a preconditioner such that it simultaneously reduces the condition number of A, well approximates the range of A^T and is fast to apply vectors to?

References

[1] Avron H., Maymounkov P., and Toledo S. Blendenpik: Supercharging lapack's least-squares solver. *SIAM Journal on Scientific Computing*, 2010.

[2] David P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science, Vol 10, Issue 1–2, 2014, pp 1–157*, 2014.

[3] Meng X., Saunders M., and Mahoney M. Lsrn: Strongly over- and under-determined systems. *SIAM Journal on Scientific Computing*, 2014.