CS 388R: Randomized Algorithms, Fall 2021

Lecture 24: Randomized Numerical Algebra II

Prof. Eric Price *Scribe: Giannis Daras and Connor L Colombe*

NOTE: THESE NOTES HAVE NOT BEEN EDITED OR CHECKED FOR CORRECTNESS

1 Overview

As before we would like to solve efficiently the simple linear regression problem:

$$\min_x ||Ax - b||, \tag{1}$$

where A is a $n \times d$ matrix for where we are assuming that n is significantly larger than d.

We argued last class that (1) could be solved naively in $O(nd^2)$ time. Then, we showed how to use the Sketch and Solve method to solve (1) in $O(nd \log n + d^3/\epsilon^2 \log^2 d)$ where the key idea was to choose some $m \times n$ matrix S that embeds the problem in a smaller dimension while still approximately maintaining the norms. The last algorithm we discussed in last class was the Conjugate Gradient Method and we showed it runs in $O(nd \log(n/\epsilon)\kappa(A))$, where $\kappa(A)$ is the "condition number" of the matrix A, defined in (2).

In this class, we show how we can improve upon the Conjugate Gradient Method when the condition number, $\kappa(A)$, is bad. We also show how we can further improve our complexity bound when A is sparse.

Notation Throughout this doc, if ommited, $||\cdot||$ denotes the l_2 norm of a vector.

2 Preconditioning to improve the Conjugate Gradient Method

2.1 Preconditioning to improve condition number

The condition number of a matrix is defined as following:

$$\kappa(A) := \frac{\max_{||x||=1} ||Ax||}{\min_{||x||=1} ||Ax||}. \tag{2}$$

Intuitively, the condition number measures how much the output vector of a linear transformation can change for a small change in the input vector. Matrices with large condition number might cause the conjugate gradient method to converge more slowly as they make little progress per iteration in their search. It would be better if we could pick a matrix with a low condition number.

Suppose we had some invertible matrix $R \in \mathbb{R}^{d \times d}$. Instead of solving the problem defined in (1), we could solve for y^* the problem:

$$y^* = \text{argmin}_y ||ARy - b||_2. \tag{3}$$

Observe that if we could solve for y^*, the solution x^* of (1) could be derived by:

$$x^* = R^{-1}y^*. \tag{4}$$

The runtime of conjugate gradient method for (3) now has a dependence on $\kappa(AR)$ instead of $\kappa(A)$. The central question becomes how can we pick an R that results in a low condition number.

We can do a singular value decomposition (SVD) and write A as:

$$A = U\Sigma V^T$$

where U is an $n \times d$ is a unitary matrix [1], Σ is a $d \times d$ diagonal matrix where the entries are the singular values of A, and V is a $d \times d$ unitary matrix.

Note that by setting $R = V\Sigma^{-1}$, we get $\kappa(AR) = 1$. Alternatively, we could compute R using QR Decomposition.

Unfortunately, computing R requires $O(nd^2)$ time and hence we are no better off than we were before. Could we potentially use something like sketch and solve to get an approximately good condition number? Indeed we can!

2.2 Sketch and Solve for an approximately good condition number

The idea is to pick some matrix $S \in \mathbb{R}^{m \times n}$ where S is a d-dimensional embedding such that

1. $$\text{For all } y,\ ||SARy||_2 = ||y||_2$$

2. $$\text{For all } x,\ ||SAx||_2 = (1 \pm \epsilon)||Ax||_2$$

If we find such an S, when we go to compute the condition number for AR,

$$\kappa(AR) = \frac{\max_{||y||=1} ||ARy||_2}{\min_{||y||=1} ||ARy||_2}$$

we can note that $ARy = x$ for some x, and use properties 1 and 2 above, to find that

$$||ARy||_2 = \frac{1}{1 \pm \epsilon}||SARy||_2 = \frac{1}{1 \pm \epsilon}||y||_2.$$

[1] A matrix U is unitary if its conjugate transpose is also its inverse $U^*U = I$.

Which then implies we that we can bound $\kappa(AR)$ by

$$\kappa(AR) = \frac{1/(1-\epsilon)}{1/(1+\epsilon)} \leq 2.$$

With the matrix preconditioned, the overall runtime is now

- Compute SA: $\tilde{O}(nd)$
- QR or Singular value decompostion to find R: $\tilde{O}(d^3)$
- Apply Conjugate Gradient method on AR: $O((nd + d^2)\log(n/\epsilon))$

2.3 Using a simpler algorithm than Conjugate Gradient

For such a small κ, it turns out that we do not even need the Conjugate Gradient Algorithm. Suppose:

$$||Ax||_2^2 = (1 \pm \epsilon)||x||_2^2 \text{ for all } x. \tag{5}$$

This is equivalent to

$$||A^T A - I||_2^2 \leq \epsilon. \tag{6}$$

We want to iteratively solve $Ax \approx b$. How would we start?

Let $x^{(1)} = A^T b$. We can improve on the next iteration by using the residual $x^{(2)} = x^{(1)} + A^T(b - Ax^{(1)})$. In general, on the k-th iteration we use

$$x^{(k)} = x^{(k-1)} + A^T(b - Ax^{(k-1)}).$$

Note that the actual solution is given by

$$\begin{aligned} x^* &= A^+ b \\ &= (A^T A)^{-1} A^T b. \end{aligned}$$

Therefore,

$$A^T A x^* = A^T b = x^{(1)}$$

which implies

$$\begin{aligned} ||x^{(1)} - x^*||_2 &= ||(A^T A - I)||_2 \\ &\leq ||A^T A - I||_2 \cdot ||x^*||_2 \\ &\leq \epsilon ||x^*||. \end{aligned}$$

That is, the first step has error of order ϵ and it decreases geometrically in ϵ as the algorithm progresses.

3 Benefiting from sparsity

3.1 Preliminaries

With the modifications we did in the previous sections, the overall time complexity is:

$$\tilde{O}(\underbrace{nd}_{\text{Computing SA}} + d^2 + \underbrace{nd\log(n/\epsilon)}_{\text{Conj. Gradient}}) \tag{7}$$

For $n >> d$, the complexity is dominated by the factor nd. This factor appears in two places, in the first and in the last term. Now, assume that A is sparse, i.e. a certain number of entries of A are zero. The complexity then becomes:

$$\tilde{O}(\underbrace{nd}_{\text{Computing SA}} + d^2 + \underbrace{\text{nnz}(A)\log(n/\epsilon)}_{\text{Conj. Gradient}}) \tag{8}$$

where nnz denotes the number of non-zero entries of A.

Hence, even though the sparsity helped us remove the nd dependence in the last term, the problematic nd term remained, since the matrix product SA is dense, even if A is sparse.

The central question becomes whether we can find a subspace embedding $S \in \mathbb{R}^{m\times n}$ such that SA can be computed in $\tilde{O}(\text{nnz}(A))$ time.

3.2 A decent attempt

Algorithm We will first try to understand how many non-zeros we can afford in any column of S such that SA can be computed in $\text{nnz}(A)$ time.

Suppose that we have k nonzeros per column of S. Then, the computation of SA will take $k\cdot\text{nnz}(A)$ time. To achieve the desired result, we can only afford a constant k.

We examine the simple case where $k = 1$ and if S_{ij} is the non-zero entry of the j-th column of S, then S_{ij} is a Rademacher R.V., i.e. with probability $p = 1/2$ it is 1 and with prob. $p = 1/2$ is -1. Ideally, we would like $||Sx||^2 \approx ||x||^2,\ \forall x \in \mathbb{R}^n$ [2].

Expectation analysis For any $x \in \mathbb{R}^n$, we can view S as a hashing of n balls (co-ordinates of x) to m bins, where the value of each bin is the value of the sum of the values of the balls allocated to the bin, but with the signs of the ball values being chosen at random.

Formally, let y_j be the sum of bin j and $h(i)$ be the bin that i-th coordinate of x is mapped to. Let R_i be a Rademacher R.V. (that reflects the sign of non-zero element of the i-th column of S). Then,

$$(Sx)_j = y_j = \sum_{i:h(i)=j} x_i \cdot R_i. \tag{9}$$

[2] This is an abuse of notation, where x is actually Ax if we want to be consistent with the notes so far.

We will now show that $||Sx||^2$ is an unbiased estimator of $||x||^2$. Specifically, we have:

$$\mathop{\mathbb{E}}_{R}[||y||^2] = \sum_j \left[\left(\sum_i x_i R_i \right)^2 \right] = ||x||^2, \tag{10}$$

since the cross-terms cancel out on expectation for random signs.

Concentration We will now study the concentration of $||Sx||^2$ around $||x||^2$. We would like to use the idea of ϵ-nets that we used last class and see what is the dependence between m and d.

Observe that for $x = e_1 + e_2$, there is a decent chance, $\frac{1}{m}$, that the two ones of x will be mapped to the same bin (collision), in which case, $||Sx||^2$ will be a bad approximation of $||x||^2$. If we were to use the ϵ-nets idea, we would get $m = 2^d$, i.e. exponential dependence on d instead of polynomial. Hence, this approach won't work in the general case.

Easier case The problem with the previous analysis appeared because x had big value spikes in its co-ordinates, which causes a big problem when the spikes are mapped to the same bin. A more uniform x would have much better properties. Specifically, if $||x||_\infty = \max_{i \in [n]} x_i$ is small, e.g. $\approx \frac{||x||_2}{\sqrt{n}}$, then the analysis works and we get polynomial dependence on d.

3.3 A sketch of a proof

We will know show that it is possible to get polynomial dependence on d.

We will start by understanding what's possible. Let:

$$A = \begin{bmatrix} e_1 & e_1 & ... & e_1 \end{bmatrix}. \tag{11}$$

Observe that if we have any collisions, then $||SAx||^2$ will be a bad approximation of $||Ax||^2$. From the Birthday Paradox, we know that we need $m \geq d^2$ to have hope that there won't be any collisions. Suprisingly, it turns out that $m = \Omega(d^2/\epsilon^2)$ is sufficient. The rest of this seciton is a sketch of a proof, as presented in class.

Let A have orthonormal columns and $x = Ay$ for some $y \in \mathbb{R}^d$. Let a_i^T be the i-th row of A. Observe that:

$$x_i^2 = (a_i^T y)^2 \leq ||a_i||^2, \quad \text{for } ||y|| = 1. \tag{12}$$

Also, see that for $||a_i|| = 1$, we have that:

$$||x||^2 = ||Ay||_2^2 = \sum_i ||a_i||^2 = ||A||_F^2 = d. \tag{13}$$

Hence, even if an individual co-ordinate might be big, the total amount is not that big.

The set T with norm $Ua_iU \geq \frac{1}{d}$ is of size $\leq d^2$. For $m > d^5$, likely all of T is separately embedded. We can do Chernoff to get a bound on the probability of error, but details will not be specified here.

With this approach, we calculate SA in time:

$$\tilde{O}(\text{nnz}(A) + d^4 + nd \log n/\epsilon). \tag{14}$$

This is pretty good, but in some cases, d^4 can be annoying. There is a way to reduce this by using Fast Johnson Lindenstrauss Transform (FJLT). The idea is that first we do a very fast embedding in a very large direction that helps us avoid collisions in the next step. The overall scheme will look like this[3]:

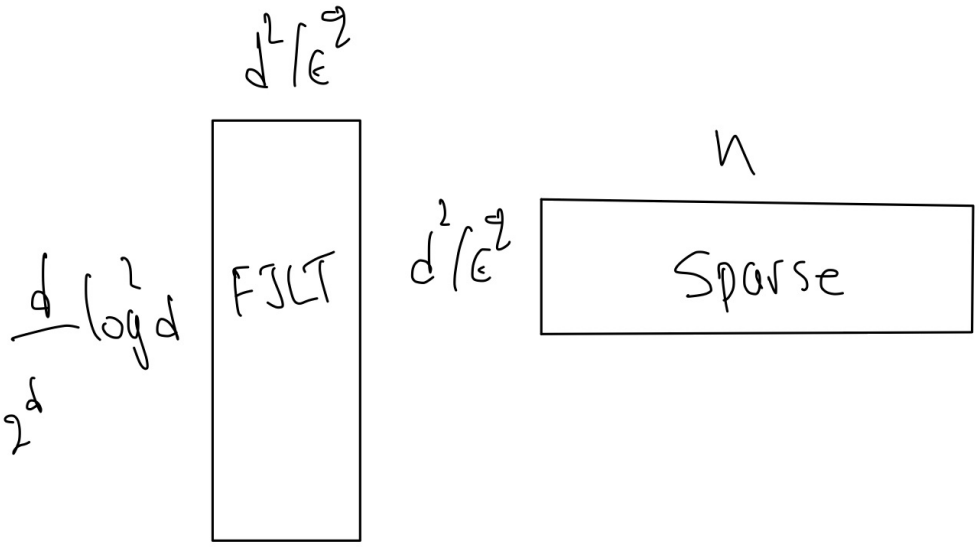

Figure 1: Sketch of overall scheme.

With this approach, we get $m = \frac{d^{1+\gamma}}{\epsilon^2}$ for sparsity poly$(1/\gamma)$.

References

[1] Dimitris Achlioptas. Database-friendly random projections: Johnson-lindenstrauss with binary coins. *Journal of computer and System Sciences*, 66(4):671–687, 2003.

[2] Per-Gunnar Martinsson and Joel Tropp. Randomized numerical linear algebra: Foundations & algorithms, 2021.

[3] Sometimes, we also multiply with a Gaussian matrix $\in \mathbb{R}^{d/\epsilon^2 \times d/\epsilon^2 \log^2 d}$.

[3] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge university press, 1995.