Overview of RandNLA: Randomized Numerical Linear Algebra

Michael W. Mahoney

ICSI and Dept of Statistics, UC Berkeley

*(For more info, see:
http://www.stat.berkeley.edu/∼mmahoney/
or Google on "Michael Mahoney")*

February 2015

Outline

RandNLA: Randomized Numerical Linear Algebra

Matrices provide a natural structure with which to **model data**.

- $A \in \mathbb{R}^{m \times n}$ can encode information about m objects, each of which is described by n *features*; etc.
- A positive definite $A \in \mathbb{R}^{n \times n}$ can encode the correlations/similarities between all pairs of n objects; etc.

Motivated by data problems, recent years have witnessed **many exciting developments** in the theory and practice of matrix algorithms.

- Particularly remarkable is the use of *randomization*.
- Typically, it is assumed to be a property of the input data due (*e.g.*, to noise in the data generation mechanisms).
- Here, it is used as an algorithmic or computational resource.

RandNLA: Randomized Numerical Linear Algebra

RandNLA: an interdisciplinary research area that exploits randomization as a computational resource to develop improved algorithms for large-scale linear algebra problems.

- Foundational perspective: roots in theoretical computer science (TCS); deep connections with convex analysis, probability theory, and metric embedding theory, etc.; and strong connections with scientific computing, signal processing, and numerical linear algebra (NLA).
- Implementational perspective: well-engineered RandNLA algorithms beat highly-optimized software libraries for problems such as very over-determined least-squares and scale well to parallel/distributed environments.
- Data analysis perspective: strong connections with machine learning and statistics and many "non-methodological" applications of data analysis.

Moreover, there is a growing interest in providing an *algorithmic and statistical foundation for modern large-scale data analysis*.

An historical perspective

Linear algebra has had a long history in large-scale (by the standards of the day) statistical data analysis.

- Method of least-squares (LS): due to Gauss, Legendre, and others; and used in early 1800s for fitting linear equations to determine planetary orbits.
- Principal Component Analysis (PCA) and low-rank approximations: due to Pearson, Hotelling, and others, and used in early 1900s for exploratory data analysis and predictive analytics.

These and related methods are of interest since, *e.g.*, if there is noise or randomness *in the data* then the leading principle components tend to capture the signal and remove the noise.

An historical perspective

Advent of the digital computer in the 1950s:

- Proto computer science and early applications of linear algebra focused on scientific computing problems (where computation was an essential tool)
- Even for "well-posed" problems, many algorithms perormed very poorly in the presence of the finite precision.
- Work by Turing, von Neumann, and others laid much of the foundations for scientific computing and NLA: this led to problem-specific complexity measures (*e.g.*, the condition number) that characterize the behavior of an input for a specific class of algorithms (*e.g.*, iterative algorithms).

But . . . (for various technical and nontechnical reasons), there then occured a split in the nascent field of computer science:

- Continuous linear algebra became the domain of applied mathematics.
- Computer science theory and practice became discrete and combinatorial.

An historical perspective

Linear algebra became the domain of continuous applied mathematics; and it focused on scientific applications.

- Nearly all work in scientific computing and NLA has been deterministic; this led to high-quality codes in the 1980s/1990s, *e.g.*, LAPACK.
- Most work focused on optimizing FLOPS—matrix-vector multiplies on dense matrices—in shared memory environments on matrices that arise in structured scientific computing applications.
- This code is now widely-used in NLA and scientific computing as well as in machine learning, statistics, data analysis, etc.

An historical perspective

Computer science became discrete and combinatorial; and it focused on business and commerce applications.

- Turing, Church, and other studied computation *per se*.
- Seemingly-different approaches (recursion theory, the λ-calculus, and Turing machines) defined the same class of functions
- Belief arose that the concept of computability is formally captured in a qualitative and robust way by these three equivalent processes, *independent of the input data*.
- Randomization (where the randomness is *inside the algorithm*, and the algorithm is applied to arbitrary or worst-case data) was introduced and exploited as a powerful computational resource.

An historical perspective: now and going forward . . .

Recently, a **convergence of these two very different perspectives**.

- Motivated by scientific, Internet, social media, financial, etc. applications.
- Computation *per se* is necessary but very insufficient.
- Most people want to *obtain insight* and/or *make predictions* from the data they generate to make downstream claims about the world.

Central to these developments RandNLA, including:

- Randomness in the data versus randomness in the algorithm.
- Continuous (mathematics) versus discrete (computer science).
- Worst-case algorithms versus problem-specific complexity measures.
- Scientific versus business/commerce applications.

Good "hydrogen atom" to consider algorithmic and statistical foundations of modern large-scale data analysis.

Outline

Basic RandNLA Principles

Basic RandNLA method: given an input matrix:

- Construct a "sketch" (a smaller or sparser matrix matrix that represents the essential information in the original matrix) by random sampling.
- Use that sketch as a surrogate to compute quantities of interest.

Basic design principles[*] underlying RandNLA:

- Randomly sample (in a careful data-dependent manner) a small number of elements to create a much sparser sketch of the original matrix.
- Randomly sample (in a careful data-dependent manner) a small number of columns and/or rows to create a much smaller sketch of the original matrix.
- Preprocess an input matrix with a random-projection-type matrix and then do uniform sampling of rows/columns/elements in order to create a sketch.

[*] The first two principles deal with identifying nonuniformity structure. The third principle deals with preconditioning the input (*i.e.*, uniformizing nonuniformity structure) s.t. uniform random sampling performs well.

Element-wise Sampling

- An $m \times n$ matrix A is an array of numbers, $A_{ij}, \forall i \in [m], \forall j \in [n]$.
- Randomly sample a small number of entries, each w.r.t. importance sampling probability distribution p_{ij}.
- Return a sparse matrix $\tilde{A}$ that contains precisely the (rescaled) entries.
- Uniform sampling easily leads to poor results; but non-uniform sampling w.r.t. magnitudes or element-wise leverage scores gives nontrivial results.
- Thm [AM01/AM07/DZ11]: If sample s elements with $p_{ij} = \frac{A_{ij}^2}{\sum_{i,j} A_{ij}^2}$, then

$$\|A - \tilde{A}\|_2 \leq O\left(\sqrt{\frac{(m+n)\ln(m+n)}{s}}\right)\|A\|_F.$$

  This gives *"additive-error" bounds* for low-rank matrix approximation.
- Proof method: $A - \tilde{A}$ is a random matrix; use random matrix theory, combinatorial moment methods, matrix measure concentration bounds.

Row/column Sampling

- An $m \times n$ matrix A is a linear operator, with column/row spaces.
- Randomly sample a small number of rows, each w.r.t. importance sampling probability distribution $\{p_i\}_{i=1}^m$.
- Return $s \times n$ matrix $\tilde{A}$, an approximation to A, containing s (rescaled) rows.
- Uniform sampling easily leads to poor results; but non-uniform sampling w.r.t. magnitudes or leverage scores gives nontrivial results.
- Thm [FVK97/DKM05/RV06]: If sample s rows with $p_i = \frac{\|A_{(i)}\|^2}{\sum_{i,j} A_{ij}^2}$, then

  $$\|A^T A - \tilde{A}^T \tilde{A}\|_F \leq \frac{1}{\sqrt{s}} \|A\|_F^2 .$$

  This gives *"additive-error" bounds* for low-rank matrix approximation.
- Proof method: expectations and variances for $\|\cdot\|_F$; Khintchine inequality or matrix-Bernstein inequalities for $\|\cdot\|_2$ extension.

Row/column Sampling

- Norm-squared sampling does only comparable to element-wise sampling.
- Leverage score sampling does better: say $m \gg n$, then let

$$p_i = \frac{1}{n} \left(P_A\right)_{ii} = \frac{1}{n} \|U_{(i)}\|_2^2,$$

  where U is any $m \times n$ orthogonal matrix spanning the column space of A.
- These *statistical leverage scores*
  - are useful in regression diagnostics to identify outliers
  - approximatable without computing U in "random projection time"
  - give *"relative-error" bounds* for least-squares & low-rank approximation
  - provide data-aware subspace embedding: fix $\epsilon \in (0,1)$, $s \gtrsim \frac{n \log(n)}{\epsilon}$ then

$$\|U^T U - (SU)^T SU\|_2 = \|I - (SU)^T SU\| \leq \epsilon.$$

  (For NLA, this is an acute perturbation; for TCS this is a subspace JL.)

Random Projections as Preconditioners[†]

- **Main challenge** for uniform sampling: relevant information could be *localized* on a small number of rows/columns/elements.
- Main challenge for non-uniform sampling: construct sampling probabilities.
- **One solution**: *spread out* this information, so uniform sampling does well.
- Bicriteria:
  - Preprocessed matrix should be similar to the original matrix.
  - Preprocessing should be computationally efficient to perform.
- Do this preconditioning with random projections:
  - Pre-/post-multiply by appropriately-scaled random matrix (i.i.d. Gaussians, i.i.d. Rademacher, Hadamard-based constructions, etc.)
  - Can get data-oblivious subspace embedding: fix $\epsilon \in (0,1)$, then

$$\|U^T U - (\Pi U)^T \Pi U\|_2 = \|I - (\Pi U)^T \Pi U\| \leq \epsilon.$$

(For NLA, this is an acute perturbation; for TCS this is a subspace JL.)

[†] Preconditioners: a transformation that converts a problem instance into another instance that is more-easily solved by a given class of algorithms.

Outline

Least-squares approximation

Least-squares (LS) : given $m \times n$ matrix A and m-dimensional vector b, solve

$$x_{opt} = \arg\min_{x \in \mathbb{R}^n} \|Ax - b\|_2.$$

- If $m \gg n$, it is overdetermined/overconstrained.
- Compute solution in $O(mn^2)$ time (in RAM model) with one of several methods: computing the normal equations; QR decompositions; or SVD.
- RandNLA provides faster algorithms for this ubiquitous problem.
    - **TCS**: faster in terms of low-precision asymptotic worst-case theory.
    - **NLA**: faster in terms of high-precision wall-clock time.
    - **Implementations**: (in Spark) can compute low, medium, and high precision solutions on up to terabyte-sized data.
- *The basic RandNLA approach extends to many other matrix problems.*

Least-squares approximation: leverage and condition

- **Statistical leverage.** *(Think: eigenvectors. Important for low-precision.)*
  - The statistical leverage scores of A (assume $m \gg n$) are the diagonal elements of the projection matrix onto the column span of A.
  - They equal the ℓ_2-norm-squared of any orthogonal basis spanning A.
  - They measure:
    - how well-correlated the singular vectors are with the canonical basis
    - which constraints have largest "influence" on the LS fit
    - a notion of "coherence" or "outlierness"
  - Computing them exactly is as hard as solving the LS problem.

- **Condition number.** *(Think: eigenvalues. Important for high-precision.)*
  - The ℓ_2-norm condition number of A is $\kappa(A) = \sigma_{\max}(A)/\sigma^{+}_{\min}(A)$.
  - $\kappa(A)$ bounds the number of iterations; for ill-conditioned problems (e.g., $\kappa(A) \approx 10^6 \gg 1$), the convergence speed is very slow.
  - Computing $\kappa(A)$ is generally as hard as solving the LS problem.

Least-squares approximation: Meta-algorithm (1 of 2)

```
1: Using the $\ell_2$ statistical leverage scores of $A$, construct an importance
   sampling distribution $\{p_i\}_{i=1}^m$.
2: Randomly sample a small number of constraints according to $\{p_i\}_{i=1}^m$
   to construct a subproblem.
3: Solve the $\ell_2$-regression problem on the subproblem.
```

A naïve version of this meta-algorithm gives a $1+\epsilon$ relative-error approximation—on both the objective function and the certificate/vector achieving the optimum—in roughly $O(mn^2/\epsilon)$ time. (Ugh.)

Least-squares approximation: Meta-algorithm (2 of 2)

(Drineas, Mahoney, etc., 2006, 2008, etc., starting with SODA 2006; Mahoney FnTML, 2011.[¶])

But, **we can make this meta-algorithm "fast" in RAM**:[‡]

- This meta-algorithm runs in $O(mn \log n/\epsilon)$ time in RAM if:
  - we perform a Hadamard-based random random projection and sample uniformly sampling in the randomly rotated basis, or
  - we quickly computing approximations to the statistical leverage scores and using those as an importance sampling distribution.
- Can be improved to run in almost $O(\mathbf{nnz}(A))$ time.

And, **we can make this meta-algorithm "high precision" in RAM**:[§]

- This meta-algorithm runs in $O(mn \log n \log(1/\epsilon))$ time in RAM if:
  - we use the random projection/sampling basis to construct a preconditioner and couple with a traditional iterative algorithm.
- See Blendenpik/LSRN for NLA-style wall-clock time comparisons.
- Can also be improved to run in almost $O(\mathbf{nnz}(A))$ time.

[‡] (Sarlós 2006; Drineas, Mahoney, Muthu, Sarlós 2010; Drineas, Magdon-Ismail, Mahoney, Woodruff 2011.)

[§] (Rokhlin & Tygert 2008; Avron, Maymounkov, & Toledo 2010; Meng, Saunders, & Mahoney 2011.)

[¶] (Mahoney, "Randomized Algorithms for Matrices and Data," FnTML, 2011.)

Extensions to Low-rank Matrix Approximation

What is your objective?

- *In NLA*: deterministic algorithms & greedy pivot rule decisions; choose exactly k columns; strong connections with QR/RRQR; focus on $\|\cdot\|_2$.
- *In TCS*: randomized algorithms, that might fail; select more than k columns, e.g., $\Theta\left(k \log(k)\right)$ columns; focus on $\|\cdot\|_F$.
- *In ML/data applications*: low-rank approximations an intermediate step.

Best algorithms: exploit the following **structural condition underlying randomized low-rank algorithms**: If $V_k^T Z$ has full rank, then

$$||A - P_{AZ}A||_\xi^2 \leq ||A - A_k||_\xi^2 + \left|\left|\Sigma_{k,\perp}\left(V_{k,\perp}^T Z\right)\left(V_k^T Z\right)^\dagger\right|\right|_\xi^2 .$$

This structural condition

- was introduced to solve the "column subset selection problem,"
- can be used to get $o(k\log(k))$ columns in TCS theory,
- is easy to parameterize RandNLA algorithms to choose $k+p$ columns,
- is easy to couple with various NLA iterative algorithms, and
- often leads to less variance in downstream data applications.

Matrix Completion

Given arbitrary $m \times n$ matrix A, reconstruct A by sampling $O\left((m+n)\text{poly}(\frac{1}{\epsilon^{\alpha}})\right)$ entries (α small, *e.g.*, 2, $\log(\frac{mn}{\epsilon})$ factors ok, but not all mn entries) s.t.

$$\|A - \tilde{A}\| \leq (1+\epsilon)\|A - A_k\|_F. \tag{1}$$

- **One approach** from TCS: above element-wise sampling algorithm. In two "passes," sample entries with based on their squared magnitude:

  $$\|A - \tilde{A}\| \leq \|A - A_k|_F + \epsilon\|A\|_F.$$

  Entire matrix is observed; works for worst-case input matrices.
  Additive error bound is too large to satisfy Eqn. (1).

- **Another approach**[‖] from signal processing and applied mathematics: under incoherence assumptions, give a uniform sample of $O\left((m+n)\,k\ln(m+n)\right)$ entries of A to form $\tilde{A}$, then A is the solution to:

  $$\min_{\tilde{A}\in\mathbb{R}^{m\times n}} \quad \|\tilde{A}\|_1 \quad \text{s.t.} \quad \tilde{A}_{ij} = A_{ij}, \quad \forall \text{ sampled entries } A_{ij}.$$

  Don't even observe all of A; but strong assumptions on A are allowed.
  If A is exactly low-rank and incoherent, then Eqn. (1) is satisfied.

[‖] Very different problem parameterizations: either assume worst-case input and must identify nonuniformity strucutre; or make "niceness" assumptions about input, where the worst nonuniformity structure is not present.

Solving Systems of Laplacian-based Linear Equations

Consider the problem of solving the system of linear equations $Ax = b$.

$$x_{opt} = \arg\min_{x \in \mathbb{R}^n} \|Ax - b\|_2. \tag{2}$$

- Solvable "exactly" in $O\left(n^3\right)$ time for worst-case dense input A
- Iterative techniques (*e.g.*, CG) used if A is positive definite (PD); then, running time is $O(\mathbf{nnz}(A))$ time, times $\kappa(A)$ factor.
- Important special case: A is the Laplacian matrix of an graph $G = (V, E)$. (Arises in scientific computing, machine learning, etc.)
- Then, there exist randomized, relative-error algorithms that run in $O\left(\mathbf{nnz}(A)\text{polylog}(n)\right)$ time.
- First step: randomized graph sparsification to create sparser Laplacian $\tilde{L}$.
  - sample edges of G according to leverage scores of weighted edge-incidence matrix
  - but must approximate them graph theoretically
- Second step: use $\tilde{L}$ (recursively) as a preconditioner to solve Eqn. (2).

Statistics, Machine Learning, and Data Applications

Many examples:

- Kernel-based machine learning: fast low-rank approximations via projections and Nyström method.
- CX/CUR decompositions provide scalable and interpretable low-rank approximations in genetics, astronomy, etc.
- More scalable scientific computing for classes of pdes.
- Divide-and-conquer matrix completion algorithms use similar analysis.
- Statistical aspects of this "algorithmic leveraging" approach.

Main challenges:

- Most people who use low-rank approximations use them for something else.
- Many statistics and machine learning formluations of these problems render the problem trivial (for important algorithmic-statistical reasons).
- Sometimes the methods do "better" than they "should" (implicit regularization), but sometimes they don't.

Outline

Conclusions

- RandNLA has had **several big successes** already:
  - The best works-case algorithms (TCS-style) for very overdetermined least-squares problems.
  - Implementations (NLA-style) are competative with and can beat the best high-quality NLA libraries.
  - Implementations (in Spark) can compute low, medium, and high precision solutions on up to terabyte-sized data.
  - Several big wins in statistics, machine learning, and data applications.

  Are these just "one off" successes, or just the tip of the iceberg?

- **This reading group**:
  - Go through several papers central to **Rand**NLA and Rand**NLA**
  - Learn a bit about RandNLA
  - Identify particularly promising directions to strengthen the NLA foundations of RandNLA