lions@epfl

EE-731: ADVANCED TOPICS IN DATA SCIENCES
LABORATORY FOR INFORMATION AND INFERENCE SYSTEMS
SPRING 2016

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

INSTRUCTOR: VOLKAN CEVHER

SCRIBERS: SOROOSH SHAFIEEZADEH-ABADEH, MICHAËL DEFFERRARD

RANDOMIZED LINEAR ALGEBRA AND STOCHASTIC QUASI-NEWTON METHOD

1 Randomized Linear Algebra

1.1 Review

As real data is often noisy, it makes sense to sacrifice accuracy for speed-up by approximating the data matrix. Randomized methods offer faster and parallelizible approximate solutions that also require a lower number of passes over the data matrix.

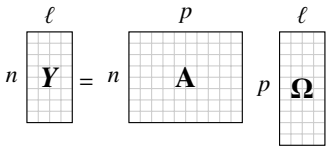
Random projections work in two steps:

1. Find an orthonormal basis $\mathbf{Q}$ that can approximate $\mathbf{A}$ well, i.e. $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{A}$.
2. Apply classical linear algebra to the smaller matrix $\mathbf{Q}^*\mathbf{A}$.

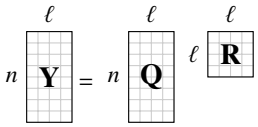
1.1.1 Decomposition

Recall that the matrix $\mathbf{A}$ can be decomposed with the following algorithm:

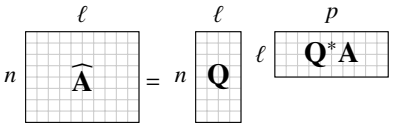
1. Multiply $\mathbf{A\Omega}$ for $\mathbf{\Omega}_{i,j} \sim \mathcal{N}(0,1)$, at cost $O(np\ell)$ (or $O(np \log \ell)$ using the FFT).



2. Compute the thin QR factorization of $\mathbf{Y}$, at a cost of $O(n\ell^2)$ (e.g. with Gram-Schmidt).



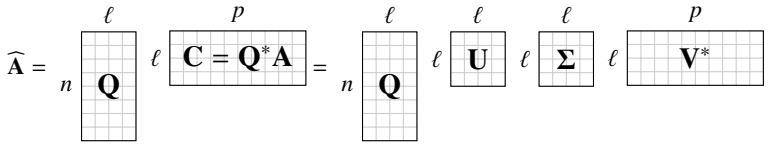
3. Finally multiply $\mathbf{C} = \mathbf{Q}^*\mathbf{A}$ at a cost of $O(np\ell)$.



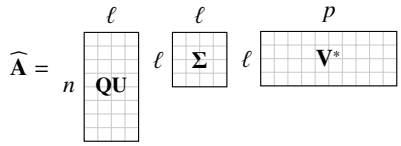
1.1.2 Randomized SVD

A small SVD on $\mathbf{C} = \mathbf{Q}^*\mathbf{A}$ then gives the randomized SVD:

1. Find SVD factors of $\mathbf{C}$ in $O(p\ell^2)$ time.

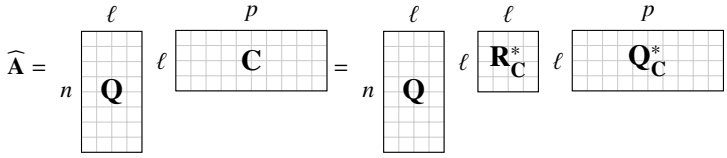


2. Multiply, in $O(n\ell^2)$ time.

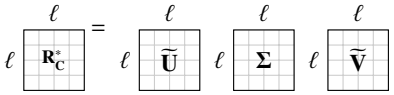


An alternative to find the randomized SVD is to compute the QR decomposition of $\mathbf{C}^*$:

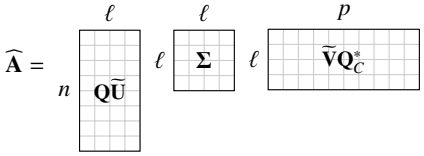
1. Find QR factors of $\mathbf{C}^*$: $\mathbf{C}^* = \mathbf{Q}_\mathbf{C}\mathbf{R}_\mathbf{C}$, in $O(p\ell^2)$ time.



2. Calculate the SVD of the small inner matrix, in $O(\ell^3)$ time.



3. Multiply in $O((n+p)\ell^2)$ time.

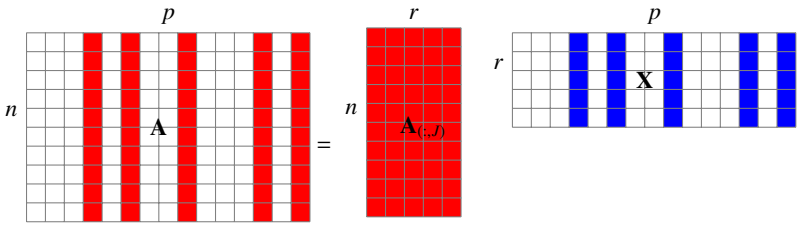


1.2 Interpolative decomposition (ID)

For a matrix $\mathbf{A} \in \mathbb{R}^{n\times p}$ of rank-r, a one-sided interpolative decomposition is defined as

$$\mathbf{A} = \mathbf{A}_{(:,J)}\mathbf{X}$$

where $J = [j_1, \ldots j_r]$ is a computed column index set and $\mathbf{X}$ is a $r \times p$ matrix with $\mathbf{X}_{(:,J)} = \mathbf{I}_r$ and $\mathbf{X}_{i,j} \leq 2\ \forall i, j$. In other words $\mathbf{A}_{(:,J)}$ is a subset of columns of $\mathbf{A}$ that spans the range of $\mathbf{A}$ with bounded coefficients.



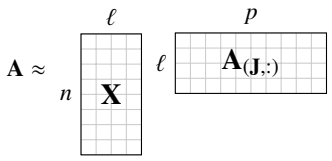
Please note that:

- There exists a decomposition of the form above with $\mathbf{X}$ whose coefficients are bounded by 1, but it is NP-hard to compute. [11]
- When the bound is 2, there are stable and efficient algorithms that compute this decomposition of a rank-r matrix $\mathbf{A} \in \mathbb{R}^{n\times p}$ at a cost of $O(nrp)$. [8]
- This decomposition can also be generalized to a two-sided form $\mathbf{A} = \mathbf{W}\mathbf{A}_{(J',J)}\mathbf{X}$ where J' is an index set of selected rows and $\mathbf{W}$ is a $n \times r$ matrix with $\mathbf{W}_{(J',:)} = \mathbf{I}_r$ and $\mathbf{W}_{i,j} \leq 2\ \forall i, j$.

1.3 Row extraction method

The problem with the aforementioned method is the $O(npl)$ cost of computing $\mathbf{C} = \mathbf{Q}^*\mathbf{A}$. We want something cheaper. Given a matrix $\mathbf{Q} \in \mathbb{R}^{n\times r}$ such that $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\| \leq \epsilon$, one can obtain the Interpolative Decomposition (ID) $\mathbf{Q} = \mathbf{X}\mathbf{Q}_{(\mathbf{J},:)}$, where J denotes the l rows of $\mathbf{Q}$ that span the rowspace of $\mathbf{Q}$ and $\mathbf{X}$ is a $n \times l$ matrix with $\mathbf{X}_{(J,:)} = \mathbf{I}_k$ and $\mathbf{X}_{i,j} \leq 2\ \forall i, j$. This costs $O(l^2 n)$ and it turns out that $\mathbf{A} \approx \mathbf{X}\mathbf{A}_{(\mathbf{J},:)}$ [9], from which we can perform a small SVD. Proof:

- As $\mathbf{X}_{(J,:)} = \mathbf{I}_k$, we have $\mathbf{A}_{(J,:)} \approx \mathbf{Q}_{(J,:)}\mathbf{Q}^*\mathbf{A}$ where $\mathbf{B} \in \mathbb{R}^{r\times p}$ consists of r rows extracted from A.
- Combining, we get $\mathbf{A} \approx \mathbf{Q}\mathbf{Q}^*\mathbf{A} = \mathbf{X}\mathbf{Q}_{(J,:)}\mathbf{Q}^*\mathbf{A}$.
- Therefore we conclude that $\mathbf{A} \approx \mathbf{X}\mathbf{B}$ with $\mathbf{B} = \mathbf{A}_{(J,:)}$.



Given $\mathbf{Q} \in \mathbb{R}^{n\times r}$ such that $\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\| \leq \epsilon$, we can compute a partial SVD at cost $O(l(n+p)^2)$ instead of $npl + O((n+p)l^2)$ (forming $\mathbf{Q}^*\mathbf{A}$ and performing SVD on it).

Algorithm: Partial SVD using row extraction
1. Form the row extraction as above: $\mathbf{Q}$: $\mathbf{A} = \mathbf{X}\mathbf{A}_{(J,:)}$ as above $O(l^2 n)$
2. Form the RQ decomposition: $\mathbf{A}_{(J,:)} = \mathbf{R}^*\mathbf{W}^*$ $O(l^2 p)$
3. Multiply $\mathbf{Z} = \mathbf{X}\mathbf{R}^*$ $O(l^2 n)$
4. Compute a classical SVD: $\mathbf{Z} = \mathbf{U}\mathbf{\Sigma}\widetilde{\mathbf{V}}^*$ $O(l^2 n)$
5. Multiply $\mathbf{V} = \mathbf{W}\mathbf{V}$ and conclude $\mathbf{A} \approx \mathbf{U}\mathbf{\Sigma}\widetilde{\mathbf{V}}^*$ $O(l^2 p)$

The decreased computational cost comes at the expense of the following worse error bound:

Lemma 1.1 (Error bound for Row Extraction [6]). *Let $\mathbf{A} \approx \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$ be the output of the algorithm produced by a given $\mathbf{Q}$ with an approximation error ϵ. Then the following error bound holds:*

$$\|\mathbf{A} - \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*\| \leq \left[\sqrt{1 + 4k(n-k)}\right]\epsilon$$

In other words, the bound is multiplied by a factor of $\sqrt{1+4k(n-k)}$ if we avoid the multiplication $\mathbf{Q}^*\mathbf{A}$ and use the row extraction technique.

1.4 Power method

The spectrum of $\mathbf{A}$ does not always decay fast, e.g. when the matrix is noisy. In those cases the basic algorithm does not work well. We therefore apply $(\mathbf{A}\mathbf{A}^*)$ several times to reduce the relative weight of the smaller singular values. Although the error guarantee is given in the spectral norm only, i.e. not in the Frobenious norm, this method works well in practice. The cost of random projections is $(2q+1)np\ell + O(q\ell^2 n)$ flops.

Algorithm: Power method
1. Draw a random matrix $\mathbf{\Omega} \in \mathbb{R}^{p\times\ell}$
2. Form the multiplication $\mathbf{Y} = (\mathbf{A}\mathbf{A}^*)^q\mathbf{A}\mathbf{\Omega}$
3. Find the orthogonal $\mathbf{Q}$ that spans the range of $\mathbf{Y}$ (e.g. using Gram-Schmidt)

Some practical considerations:

- Usually $q = 2$ or $q = 3$ is sufficient.
- $\mathbf{\Omega}$ is taken to be Gaussian. The other matrices do not work well.
- We perform QR factorization at each step.
- Apply $\mathbf{A}$ and $\mathbf{A}^*$ alternatively instead of forming $\mathbf{A}\mathbf{A}^*$.

Theorem 1.2 (Power method [6]). *Let $\mathbf{A} \in \mathbb{R}^{n\times p}$ with $n \geq p$ be the matrix that is randomly approximated using the power method. Then the following holds:*

$$\mathbb{E}\|\mathbf{A} - \mathbf{Q}\mathbf{Q}^*\mathbf{A}\|_2 \leq \left(1 + \frac{4\sqrt{r+s}}{s-1}\sqrt{p}\right)^{1/(2q+1)}\sigma_{r+1}.$$

Note that when $q = 0$, this is the original algorithm with $\mathbf{Y} = \mathbf{A}\mathbf{\Omega}$. The extra factor can be made close to 1 by increasing the number of passes q, but this is at the expense of increasing the computational cost.

1.5 Column selection methods

So far we only considered **linear combinations** of the columns for reducing the dimensionality of a matrix (SVD). Another approach is to find a **subset** of columns that could well summarize the action of the matrix. This makes it easier to interpret for data analysts although it is combinatorially hard. E.g. if the problem is to identify genes who promote a certain behavior, you would prefer to identify a subset of important genes rather than a linear combination of all of them.

Problem 1 (Column Subset Selection Problem)**.** *Given a matrix* $\mathbf{A} \in \mathbb{R}^{n\times p}$ *and a positive integer* r*, pick* r *columns of* $\mathbf{A}$ *to form a matrix* $\mathbf{C} \in \mathbb{R}^{n\times r}$ *such that the residual*

$$\|\mathbf{A} - \mathbf{P}_{\mathbf{C}}\mathbf{A}\|_\xi$$

is minimized over all possible $\binom{n}{r}$ *choices for the matrix* $\mathbf{C}$*. Here,* $\mathbf{P}_{\mathbf{C}} = \mathbf{C}\mathbf{C}^\dagger$ *denotes the projection onto the* r*-dimensional space spanned by the columns of* $\mathbf{C}$ *and* $\xi = 2$ *or* F *denotes the spectral norm or Frobenius norm. [1]*

Preliminary results:

- Uniform sampling of the columns is a bad idea in theory and in practice
- When $O(r\log(r)/\epsilon^2)$ columns are selected with probabilities proportional to the Frobenius norm of columns of A, we have

$$\|\mathbf{A} - \P_{\mathbf{C}_r}\mathbf{A}\|_F \le \|\mathbf{A} - \mathbf{A}_r\|_F + \epsilon\|\mathbf{A}\|_F$$

$$\|\mathbf{A} - \P_{\mathbf{C}_r}\mathbf{A}\|_2 \le \|\mathbf{A} - \mathbf{A}_r\|_2 + \epsilon\|\mathbf{A}\|_F$$

  with high probability [3]. $\mathbf{A}_r$ and $\mathbf{C}_r$ are the best rank-r approximations to the matrices $\mathbf{A}$ and $\mathbf{C}$ respectively. ($\mathbf{P}_{\mathbf{X}} = \mathbf{X}\mathbf{X}^\dagger$ is the projection to the column space of $\mathbf{X}$.)

Given a singular value decomposition $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$, an improved random sampling is given by [4].

- Compute the importance sampling probabilities (leverage scores)

$$p_i = \frac{1}{r}\|\mathbf{V}_r^{*(i)}\|_2^2$$

  where $\mathbf{V}_r^{*(i)}$ is the i^{th} row of matrix $\mathbf{V}_r^*$ that contains the top r right-singular vectors.
- Sample $c = O(r\log(1/\delta)r/\epsilon^2)$ columns of $\mathbf{A}$ according to this distribution to form a submatrix $\mathbf{C} \in \mathbb{R}^{n\times c}$.
- Then with probability at least $1-\delta$, the following holds:

$$\|\mathbf{A} - \P_{\mathbf{C}_r}\mathbf{A}\|_2 \le (1+\epsilon)\|\mathbf{A} - \mathbf{A}_r\|_F.$$

- However approximating the leverage scores is expensive: $O(np\log n)$.

1.6 Which method to use ?

If interpretability is important, then go for column subset selection. Keep in mind that it might be expensive due to the cost of calculating leverage scores. Otherwise use random projections, the choice of which depends on the scenarios below.

1. Matrix $\mathbf{A}$ fits in the core memory.
   - Use a structured random matrix (e.g. SRFT) for Step-1 of low rank approximation using $O(np\log\ell + \ell^2 n)$ flops.
   - For Step-2, use the row extraction technique at the cost of $O(\ell^2(n+p))$.
   - The total cost is $O(np\log l + \ell^2(n+p))$.
   - If the row extraction results in a large error, use the direct method of forming $\mathbf{Q}^*\mathbf{A}$ at Step-2 which costs $O(np\ell)$.
2. Matrix $\mathbf{A}$ can be rapidly applied to vectors.
   - This is the case for sparse matrices or structured matrices such as Toeplitz.
   - The cost of matrix-vector multiplication could be as low as $C_{mult} = O(n+p)$.
   - Step-1 to find $\mathbf{Q}$ costs $\ell C_{mult} + O(\ell^2 n)$.
   - In Step-2, form the $\mathbf{Q}^*\mathbf{A}$ as it is now cheap: $\ell C_{mult} + O(\ell^2(n+p))$.
   - In total it costs $2\ell C_{mult} + O(\ell^2(n+p))$.
   - If the singular values of $\mathbf{A}$ decays slowly, use power method with q iterations which costs $(2q+2)\ell C_{mult} + O(\ell^2(n+p))$.
   - Krylov methods would also benefit from this speed-up, but they are less robust and not as parallelizable as these random methods.
3. Matrix $\mathbf{A}$ is stored in a slow memory.
   - The computational time is dominated by the memory access, therefore classical methods which require at least r pass over the matrix are not practical.
   - One can use any of the randomized algorithms above according to the needs: e.g. if the decay of singular values is slow, use power method, as a small q would be sufficient.

2 Stochastic quasi-Newton Method

In many machine learning applications, we need to solve large scale optimization problems coming from massive amounts of training samples. The full-batch approach is feasible and appropriate if you can fulfill the computational and memory demands. However, in most large scale learning problems, it is inevitable to employ algorithms which update the prediction model based on a relatively small subset of the training data. These algorithms are known as stochastic methods which are particularly suited for settings where we access a big dataset or a data stream.

Among full-batch approaches, Newton methods are known to guarantee the fastest convergence rate. In contrast to the first order methods (such as a simple gradient decent algorithm) that only use the first order information, Newton methods exploit the second derivatives of parameters to take advantage of the curvature information. The goal of this section is to incorporate the stochastic framework into the Newton methods.

2.1 Iterative Optimization Methods

Suppose we are interested in solving the optimization problem

$$\min_{\mathbf{x}\in\mathbb{R}^p} \; f(\mathbf{x}). \tag{$\mathcal{P}$}$$

We denote the optimal value and optimal solution by $f^\star$ and $\mathbf{x}^\star$. The starting point of many optimization algorithms is to use the quadratic Taylor expansion of the objective f at iteration k

$$f(\mathbf{x}) \approx Q(\mathbf{x}, \mathbf{x}^k) := f(\mathbf{x}^k) + \langle \mathbf{x} - \mathbf{x}^k, g(\mathbf{x}^k)\rangle + \frac{1}{2}\langle \mathbf{x} - \mathbf{x}^k, \boldsymbol{H}_k(\mathbf{x}^k)(\mathbf{x} - \mathbf{x}^k)\rangle, \tag{1}$$

where $g(\mathbf{x}^k)$ and $\boldsymbol{H}_k(\mathbf{x}^k)$ are the approximations of gradient ∇f and Hessian $\nabla^2 f$ at iteration k. Notice that $Q(\mathbf{x}, \mathbf{x}^k)$ is a quadratic function which is minimized by solving $\nabla_x Q(\mathbf{x}, \mathbf{x}^k) = 0$. Thus, the minimum is attained at

$$\bar{\mathbf{x}}^k := \mathbf{x}^k - [\boldsymbol{H}_k(\mathbf{x}^k)]^{-1} g(\mathbf{x}^k). \tag{2}$$

Then, we update the current solution by

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k(\bar{\mathbf{x}}^k - \mathbf{x}^k), \tag{3}$$

where α_k is the learning rate at iteration k which can be updated by line search methods [10]. Different choices of g and $\boldsymbol{H}_k$ result in different optimization algorithms, including the well-known Newton and Gradient methods and their variant presented in Table 1. Note that using the curvature information encapsulated in the Hessian matrix and having access to full gradient lead to the fastest convergence rate in Newton method.

Table 1: Taxonomy of some methods.

	Newton	Quasi-Newton	Gradient	Stochastic Gradient
g	∇f	∇f	∇f	$\mathbb{E}[g(\mathbf{x}^k)] = \nabla f(\mathbf{x}^k)$
$\boldsymbol{H}_k$	$\nabla^2 f$	$\approx \nabla^2 f$	$L_k\mathbb{I}$	$L_k\mathbb{I}$
rate	quadratic	superlinear	linear	$O(1/k)$

2.2 Different Convergence Rates

Suppose we have a converging sequence $\lim_{k\to\infty} s_k = s^\star$, and we would like to characterize the speed, or rate, at which the iterates s_k approach the limit $s^\star$. A converging sequence of numbers $\{s_k\}$ exhibits linear convergence if for some $0 \le C < 1$,

$$\limsup_{k\to\infty} \frac{|s_{k+1} - s^*|}{|s_k - s^\star|} = C,$$

where $\limsup_{k\to\infty}$ denotes the worst-case limit points of a sequence. C in the above expression is termed as the rate constant; if $C = 0$, the sequence exhibits superlinear convergence; if $C = 1$, the sequence exhibits sublinear convergence. For instance the sequences $s_k = \left(\frac{1}{10}\right)^k$, $s_k = \frac{0.1}{k!}$, and $s_k = \frac{0.1}{k}$ have linear, superlinear, and sublinear convergence rates to zero, respectively. A sequence of numbers $\{s_k\}$ exhibits quadratic convergence if it converges to some limit $s^\star$ and

$$\limsup_{k\to\infty} \frac{|s_{k+1} - s^*|}{|s_k - s^\star|^2} = \delta \le \infty.$$

For instance, the sequence $s_k = \left(\frac{1}{10}\right)^{(2^k-1)}$ has quadratic convergence rate to zero. In optimization, we aim to determine the converging rate of the sequence $f(\mathbf{x}^k)$ to $f^\star$. In this context, algorithms with sublinear, linear, and quadratic convergence rates need at most an order of $1/\epsilon$, $\log(1/\epsilon)$, and $\log(\log(1/\epsilon))$ iterations, respectively to achieve ϵ sub-optimality (i.e., $|f(\mathbf{x}^k) - f^\star| \leq \epsilon$).

It is common to think that a linearly converging method is thought to be better than a sublinearly converging one; among two linearly converging methods the more preferable is the one with the smaller rate constant; a superlinearly converging method is preferred to a linearly converging one, etc. Of course, all these preferences are conditional, provided that there are no significant differences in the computational complexity of each step.

2.3 Newton Method and its Full-Batch Variants

As we pointed out before, Newton methods not only use the gradient information but also exploit the second derivatives of parameters. Geometrically, it means that the method takes advantage of the function curvature. In fact, information on $\nabla^2 f(\mathbf{x}^k)$ leads $Q(\mathbf{x}, \mathbf{x}^k)$ to be a better approximation of f. As a result, the Newton method uses curvature information to take a more direct route to the optimal solution in comparison to gradient methods. Figure 1 illustrates the difference in routes taken by a gradient descent method and a Newton method for minimizing a function with small step sizes.

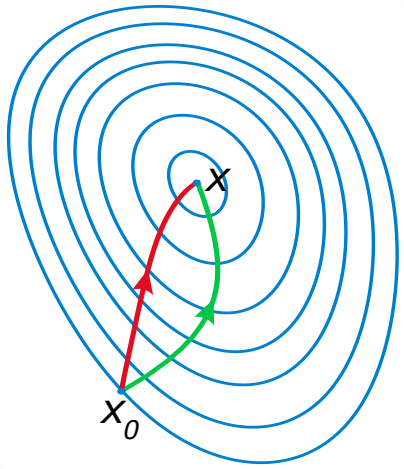


Figure 1: A comparison of gradient descent (green) and Newton's method (red).

Newton algorithm for solving $(\mathcal{P})$ can be summarized as:

Newton Algorithm
1. Initialize $\mathbf{x}^0 \in \mathbb{R}^p$.
2. For $K = 0, 1, \cdots$ performs:
$\mathbf{x}^{k+1} = \mathbf{x}^k - [\nabla^2 f(\mathbf{x}^k)]^{-1} \nabla f(\mathbf{x}^k)$

However, the Hessian in high-dimensional space can be a very dense matrix which may be computationally expensive to use. Moreover, the Hessian is close to a non-invertible matrix in most practical cases, and as a result, the inverted Hessian can be numerically unstable and the solution may diverge. In this case, we need to modify or approximate the Hessian. Quasi-Newton methods, as an alternative to Newton method, are methods used to estimate the Hessian by analyzing successive gradient vectors. The Quasi-Newton scheme can be seen as:

Quasi-Newton scheme
1. Initialize $\mathbf{x}^0 \in \mathbb{R}^p$, set $\boldsymbol{H}_0 = \mathbb{I}$, and compute $f(\mathbf{x}^0)$ and $\nabla f(\mathbf{x}^0)$.
2. For $K = 0, 1, \cdots$ performs:
2.a. Compute $\mathbf{v}^k = \boldsymbol{H}_k \nabla f(\mathbf{x}^k)$.
2.b. Compute $\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k \mathbf{v}^k$.
2.c. Compute $f(\mathbf{x}^{k+1})$ and $\nabla f(\mathbf{x}^{k+1})$.
2.d. Update $\boldsymbol{H}_{k+1}$ based on $f(\mathbf{x}^{k+1})$ and $\nabla f(\mathbf{x}^{k+1})$.

Defining $\mathbf{v}^k = \nabla f(\mathbf{x}^{k+1}) - \nabla f(\mathbf{x}^k)$ and $\mathbf{y}^k = \mathbf{x}^{k+1} - \mathbf{x}^k$, the most popular update formulas in Quasi-Newton scheme are:

- Rank-one correction:
$$\boldsymbol{H}_{k+1} = \boldsymbol{H}_k + \frac{(\mathbf{y}^k - \boldsymbol{H}_k\mathbf{v}^k)(\mathbf{y}^k - \boldsymbol{H}_k\mathbf{v}^k)^\top}{(\mathbf{y}^k - \boldsymbol{H}_k\mathbf{v}^k)^\top\mathbf{v}^k}.$$
- Davidon-Fletcher-Powell:
$$\boldsymbol{H}_{k+1} = \boldsymbol{H}_k + \frac{\mathbf{y}^k(\mathbf{y}^k)^\top}{(\mathbf{v}^k)^\top\mathbf{y}^k} - \frac{\boldsymbol{H}_k\mathbf{v}^k(\mathbf{v}^k)^\top\boldsymbol{H}_k}{(\mathbf{v}^k)^\top\boldsymbol{H}_k\mathbf{v}^k}.$$
- Broyden-Fletcher-Goldfarb-Shanno:
$$\boldsymbol{H}_{k+1} = \boldsymbol{H}_k + \frac{\boldsymbol{H}_k\mathbf{v}^k(\mathbf{y}^k)^\top + \mathbf{y}^k(\mathbf{v}^k)^\top\boldsymbol{H}_k}{(\mathbf{v}^k)^\top\boldsymbol{H}_k\mathbf{v}^k} - \beta_k\frac{\boldsymbol{H}_k\mathbf{v}^k(\mathbf{v}^k)^\top\boldsymbol{H}_k}{(\mathbf{v}^k)^\top\boldsymbol{H}_k\mathbf{v}^k},$$
where
$$\beta_k = 1 + \frac{(\mathbf{v}^k)^\top\mathbf{y}^k}{(\mathbf{v}^k)^\top\boldsymbol{H}_k\mathbf{v}^k}.$$

2.4 Stochastic Extensions

In this section, we consider the following problem:

$$f^\star = \min_{\mathbf{x}\in\mathbb{R}^p}\{f(\mathbf{x}) := \frac{1}{n}\sum_{i=1}^n f_i(\mathbf{x})\}, \tag{4}$$

where $f_i, i = 1, \cdots, n$, are twice continuously differentiable and their Hessian $\nabla^2 f_i$ are positive definite. In the high-dimensional regime where both n and p are large, evaluation of the gradient and Hessian can be computationally prohibitive. Thus, we need to exploit the hidden structure in the optimization program. Among these structures, the finite sum is the most pervasive as it appears in empirical risk minimization problems. As we discussed in lecture 5, stochastic optimization can be applied to solve such problems. To adapt this frame to Newton method, we need the following stochastic approximations:

I. sub-sampled gradient,

II. sub-sampled Hessian,

III. sub-sampled approximation of Hessian.

2.4.1 Stochastic quasi-Newton with Sublinear Convergence Rate

In this section, we present a stochastic quasi-Newton method which employs the classical BFGS update formula in its limited memory form. Instead of applying the classical approach that would compute differences of gradients at every iteration to estimate the Hessian, the Hessian approximation is based on the observation through sub-sampled Hessian-vector products. This algorithm can be summarized as:

Stochastic BFGS algorithm [2]

1. Set $t = -1$ and choose $L \in \mathbb{N}$ and $M \in \mathbb{N}$.
2. For $K = 0, 1, \cdots$ performs:
   - 2.a. Choose a sample $\mathcal{S}$.
   - 2.b. Calculate stochastic gradient $\nabla_{\mathcal{S}} f(\mathbf{x}^k)$
   - 2.c. Update:
$$\mathbf{x}^{k+1} = \begin{cases}\mathbf{x}^k - \alpha_k\nabla_{\mathcal{S}} f(\mathbf{x}^k), & \text{if } t < 1,\\ \mathbf{x}^k - \alpha_k\boldsymbol{H}_k\nabla_{\mathcal{S}} f(\mathbf{x}^k), & \text{otherwise.}\end{cases}$$
   - 2.d. When $mod(k, L) = 0$, perform:
     - 2.d.1. $t = t + 1$.
     - 2.d.2. $\bar{\mathbf{x}}^t = \frac{1}{L}\sum_{j=k-L+1}^k \mathbf{x}^j$.
     - 2.d.3. Choose a sample $\mathcal{T} \subset 1, \cdots n$.
     - 2.d.4. Compute:
$$\mathbf{y}^t = \bar{\mathbf{x}}^t - \bar{\mathbf{x}}^{t-1}, \quad \mathbf{v}^t = \nabla^2_{\mathcal{T}} f(\bar{\mathbf{x}}^t)(\bar{\mathbf{x}}^t - \bar{\mathbf{x}}^{t-1}).$$
   - 2.e. Update Hessian:
     - 2.e.1. Compute: $\boldsymbol{H} = (\mathbf{y}^t)^\top\mathbf{v}^t/((\mathbf{v}^t)^\top\mathbf{v}^t)$
     - 2.e.2. For $j = t - \min\{t, M\} + 1, \cdots, t$ perform:
$$\rho_j = 1/((\mathbf{y}^j)^\top\mathbf{v}^j), \quad \boldsymbol{H}_t = (\mathbb{I} - \rho_j\mathbf{y}^j(\mathbf{v}^j)^\top)\boldsymbol{H}(\mathbb{I} - \rho_j\mathbf{v}^j(\mathbf{y}^j)^\top) + \rho_j\mathbf{y}^j(\mathbf{y}^j)^\top$$

To approximate the Hessian, M most recent pairs of $(\mathbf{y}^l, \mathbf{v}^l)$ is used to apply the BFGS update formula. Notice that the computational cost of stochastic BFGS could be much cheaper than SGD as it is not necessary to compute the full Hessian. The directional derivative along the vector α is enough. Formally speaking, we need to compute

$$\nabla^2_S f(\mathbf{x})(\mathbf{v}) = \frac{d}{d\alpha} \nabla f_S(\mathbf{x} + \alpha \mathbf{v})\Big|_{\alpha=0}.$$

Furthermore, the quasi-Newton method can take advantage of parallelism by applying the compact form of limited memory BFGS updating presented in [10]. Now, we analyze the convergence properties of the presented stochastic quasi-Newton method.

Theorem 2.1 ($O(1/k)$ rate of stochastic BFGS [2])**.** *Suppose that:*

1. $(\forall Q \subset \{1, \cdots, n\})(\forall \boldsymbol{x} \in \mathbb{R}^p): \quad \lambda \mathbb{I} \preceq \nabla^2_Q f(\boldsymbol{x}) \preceq \Lambda \mathbb{I}, \quad$ *where* $\quad \nabla^2_Q f(\boldsymbol{x}) = \frac{1}{|Q|} \sum_{i \in Q} \nabla^2 f_i(\boldsymbol{x})$.
2. $\mathbb{E}[\|\nabla f(\boldsymbol{x}^k)\|^2] \leq \gamma^2$.

and

$$\alpha_k = \beta/k \quad \text{with} \quad \beta > 1/(2\mu_1 \lambda).$$

Then for all $k \geq 1$,

1. *there exist* (μ_1, μ_2) *such that* $\mu_1 \mathbb{I} \preceq \boldsymbol{H}_k \preceq \mu_2 \mathbb{I}$.
2. *the following holds:* $\mathbb{E}[f(\boldsymbol{x}^k) - f^\star] \leq Q(\beta)/k$, *where* $Q(\beta) = \max\{\frac{\Lambda \mu_2^2 \beta^2 \gamma^2}{2(2\mu_1 \lambda \beta - 1)}, f(\boldsymbol{x}^0) - f^\star\}$.

Note that the convergence rate does not depend on the condition number of the problem; i.e., Λ/λ. We now present a numerical experiment to compare stochastic quasi-Newton method with SGD. We first test these two algorithms on a binary classification problem through the logloss function

$$\min_{\mathbf{x} \in \mathbb{R}^p} -\frac{1}{n} \sum_{i=1}^{n} y_i \log(c(\mathbf{x}, \mathbf{x}_i)) + (1 - y_i) \log(1 - c(\mathbf{x}, \mathbf{x}_i)),$$

where $c(\mathbf{x}, \mathbf{x}_i) = 1/(1 + \exp(-\mathbf{x}_i^\top \mathbf{x}))$ and $y_i \in \{0, 1\}$. The gradient and Hessian of f_i are $\nabla f_i(\mathbf{x}) = (c(\mathbf{x}, \mathbf{x}_i) - y_i)\mathbf{x}_i$ and $\nabla^2 f(\mathbf{x})(\mathbf{v}) = c(\mathbf{x}, \mathbf{x}_i)(1 - c(\mathbf{x}, \mathbf{x}_i))(\mathbf{x}_i^\top \mathbf{v})\mathbf{x}_i$, respectively. Figure 2 reports the performance of SGD and stochastic quasi-Newton method. Both methods use a gradient batch size of $b = 50$. For stochastic quasi-Newton, we display results for two values of the Hessian batch size b_H, and set $M = 10$ and $L = 10$. The vertical axis, labeled fx, measures the value of the objective; the dotted black line marks the best function value obtained by the coordinate descent (CD) method. We observe that the stochastic quasi-Newton method with $b_H = 300$ and 600 outperforms SGD, and obtains the same or better objective value than the coordinate descent method.

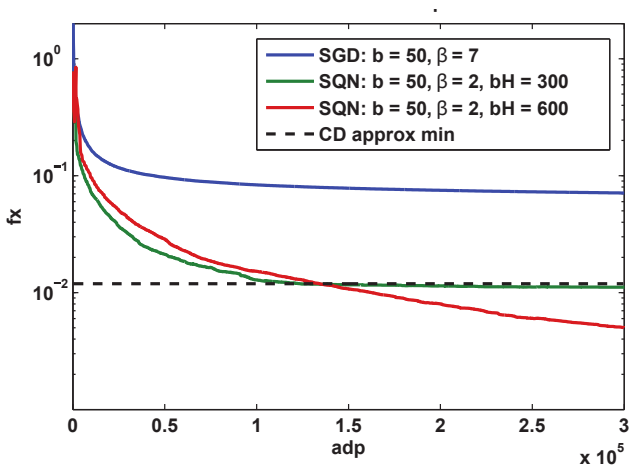


Figure 2: Illustration of SQN and SGD on the synthetic dataset.

2.4.2 Stochastic Newton Method with linear Convergence Rate

As shown in Theorem 2.1, the convergence rate of stochastic quasi-Newton method is sublinear which is similar to SGD. Recall that the converge rate of Newton and quasi-Newton methods are quadratic and sublinear. Hence, one can expect that stochastic Newton may have linear rate. Interestingly, the linear convergence rate can be attained by using sub-sampled Hessian. The challenge here is to achieve a right balance for sub-sample size as small sub-sample Hessian has not much larger computational cost than cost of gradient while using large sub-sample Hessian helps us get more curvature information. In this section, we present two stochastic second-order optimization algorithms and provide bounds on the their convergence. The first algorithm incorporates sub-sampled Hessian while using the full gradient. Defining $\nabla^2_{\mathcal{S}} f(\mathbf{x}) := \frac{1}{|\mathcal{S}|}\sum_{i\in\mathcal{S}} \lambda^2 f_i(\mathbf{x})$, the first algorithm is

Hessian sub-sampling Newton algorithm [12]
1. Choose $\mathbf{x}^0 \in \mathbb{R}^p, \beta \in (0,1)$ and $\hat{\alpha} \geq 1$ and set sample size s.
2. For $K = 0, 1, \cdots$ performs:
2.a. Select $\mathcal{S} \subset \{1, \cdots, n\}$ of size s, compute $\mathcal{S}$-sub-sampled Hessian $\nabla^2_{\mathcal{S}} f(\mathbf{x}^k)$.
2.b. Compute $\mathbf{v}^k = -[\nabla^2_{\mathcal{S}} f(\mathbf{x}^k)]^{-1}\nabla f(\mathbf{x}^k)$ and solve $\alpha_k = \arg\max \alpha \quad \text{s.t.} \quad \alpha \leq \hat{\alpha}$ and $f(\mathbf{x}^k + \alpha\mathbf{v}^k) \leq f(\mathbf{x}^k) + \alpha\beta(\mathbf{v}^k)^\top \nabla f(\mathbf{x}^k)$
2.c. Update: $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k \mathbf{v}^k$

In the sequel, we define $\kappa := \Lambda/\lambda$ and

$$\tilde{\kappa} := \begin{cases} \kappa_1, & \text{if } \mathcal{S} \text{ is drawn with replacement,} \\ \kappa_{\mathcal{S}}, & \text{if } \mathcal{S} \text{ is drawn without replacement.} \end{cases}$$

Here, given $q \in [1, n]$, $\hat{\Lambda}_q$ is the average of q largest Λ_i and $\kappa_q = \hat{\Lambda}_q/\lambda$. The following theorem provides a linear convergence rate for the first algorithm.

Theorem 2.2 (Linear rate convergence [12])**.** *Suppose that* $0 \preceq \nabla^2 f_i(\boldsymbol{x}) \preceq \Lambda_i \mathbb{I}$ *and* $\lambda\mathbb{I} \prec \nabla^2 f(\boldsymbol{x}) \prec \Lambda\mathbb{I}$ *for every* $\boldsymbol{x} \in \mathbb{R}^p$*. Given* $\epsilon \in (0,1), \delta \in (0,1)$*, and* $\boldsymbol{x} \in \mathbb{R}^p$ *and suppose that*

$$|\mathcal{S}| \geq \frac{2\kappa_1 \ln(p/\delta)}{\epsilon^2}.$$

With probability $1-\delta$*, one has* $f(\boldsymbol{x}^{k+1}) - f^\star \leq (1-\rho_k)(f(\boldsymbol{x}^k) - f^\star)$*, where* $\rho_k = \frac{2\alpha_k\beta}{\tilde{\kappa}}$*. Furthermore,* $\alpha_k \geq \frac{2(1-\beta)(1-\epsilon)}{\kappa}$.

Recall that the convergence rate of SGD is $1-\rho$ where $\rho = \Lambda/\lambda$ is the condition number of f. By choosing appropriate β and ϵ, ρ_k can be smaller than the condition number, thus the proposed quasi-newton method has faster convergence rate.

Next, in addition to Hessian sub-sampling, we can consider sub-sampling the gradient as a way to further reduce the computational complexity per iteration. To obtain proper sampling strategy, randomized numerical linear algebra is applied to approximate matrix multiplication. Defining sub-sample Hessian $\nabla^2_{\mathcal{S}} f(\mathbf{x}) := \frac{1}{|\mathcal{S}|}\sum_{i\in\mathcal{S}} \lambda^2 f_i(\mathbf{x})$ and sub-sample gradient $\nabla_{\mathcal{T}} f(\mathbf{x}) := \frac{1}{|\mathcal{T}|}\sum_{i\in\mathcal{T}} \lambda f_i(\mathbf{x})$, the second algorithm is as following:

Hessian sub-sampling Newton algorithm [12]
1. Choose $\mathbf{x}^0 \in \mathbb{R}^p, \beta \in (0,1)$ and $\hat{\alpha} \geq 1$ and set sample size s and t.
2. For $K = 0, 1, \cdots$ performs:
2.a. Select $\mathcal{S} \subset \{1, \cdots, n\}$ of size s and $\mathcal{T} \subset \{1, \cdots, n\}$ of size t.
2.b. Compute $\nabla^2_{\mathcal{S}} f(\mathbf{x}^k)$ and $\nabla_{\mathcal{T}} f(\mathbf{x}^k)$.
2.c. Compute $\mathbf{v}^k = -[\nabla^2_{\mathcal{S}} f(\mathbf{x}^k)]^{-1}\nabla_{\mathcal{T}} f(\mathbf{x}^k)$ and solve $\alpha_k = \arg\max \alpha \quad \text{s.t.} \quad \alpha \leq \hat{\alpha}$ and $f(\mathbf{x}^k + \alpha\mathbf{v}^k) \leq f(\mathbf{x}^k) + \alpha\beta(\mathbf{v}^k)^\top \nabla_{\mathcal{T}} f(\mathbf{x}^k)$
2.c. Update: $\mathbf{x}^{k+1} = \mathbf{x}^k + \alpha_k \mathbf{v}^k$

The following theorem provides a linear convergence rate for the second algorithm.

Theorem 2.3 (Linear rate convergence [12])**.** *Suppose that* $0 \preceq \nabla^2 f_i(\boldsymbol{x}) \preceq \Lambda_i \mathbb{I}$ *and* $\lambda\mathbb{I} \prec \nabla^2 f(\boldsymbol{x}) \prec \Lambda\mathbb{I}$ *for every* $\boldsymbol{x} \in \mathbb{R}^p$*. Given* $\epsilon_1 \in (0,1), \epsilon_2 \in (0, 1/2), \delta \in (0,1)$*, and* $\boldsymbol{x} \in \mathbb{R}^p$ *and suppose that*

$$|\mathcal{S}| \geq \frac{2\kappa_1 \ln(p/\delta)}{\epsilon_1^2} \quad \text{and} \quad |\mathcal{T}| \geq \frac{\max_{1\leq i\leq n} \sup_{k\in\mathbb{N}} \|\nabla f_i(\boldsymbol{x}^k)\|^2}{\epsilon_2^2}(1 + \sqrt{8\ln(1/\delta)})^2$$

With probability $1-\delta$*, one has* $f(\boldsymbol{x}^{k+1}) - f^\star \leq (1-\rho_k)(f(\boldsymbol{x}^k) - f^\star)$*, where* $\rho_k = \frac{8\alpha_k\beta}{9\tilde{\kappa}}$*. Furthermore,* $\alpha_k \geq \frac{2(1-\beta)(1-\epsilon_1)}{\kappa}$.

In both algorithms, computing the update requires solving a large scale linear program, which can be computationally expensive. As a remedy, one can solve the linear programs approximately, as presented in [13].

2.5 Stochastic Block quasi-Newton

In this section, we present a new limited-memory stochastic block BFGS update combined with the variance reduction approach of SVRG, suggested in [7], to compute batch stochastic gradients. In this method, the inverse Hessian matrix is estimated at each iteration using a sketch of the Hessian, i.e., a randomly generated compressed form of the Hessian. We propose several sketching strategies. The algorithm is as follow:

Stochastic block BFGS algorithm [5]

Inputs: $\mathbf{x}_0 \in \mathbb{R}^p$, step size $\eta > 0$, s = subsample size, q = sample action size, m = size of the inner loop, M = memory parameter.

1. Initiate: $\boldsymbol{H}_{-1} = \mathbb{I}$.
2. For $K = 0, 1, \cdots$ performs:
   - 2.a. Compute the full gradient $\nabla f(\mathbf{x}^k)$.
   - 2.b. Set $\mathbf{y}^0 = \mathbf{x}^k$.
   - 2.c. For $t = 0, \cdots, m-1$, performs:
     - 2.c.1. Sample $\mathcal{S}_t$ and $\mathcal{T}_t$ in $\{1, \cdots, n\}$, independently.
     - 2.c.2. Compute: $\mathbf{v}^t = \nabla_{\mathcal{S}_t} f(\mathbf{y}^t) - \nabla_{\mathcal{S}_t} f(\mathbf{x}^k) + \nabla f(\mathbf{x}^k)$.
     - 2.c.3. Form $\mathbf{A}_t \in \mathbb{R}^{p \times p}$ so that $\text{rank}(\mathbf{A}^t) = q$.
     - 2.c.4. Compute $\mathbf{Y}_t = \nabla^2_{\mathcal{T}_t} f(\mathbf{y}^t)\mathbf{A}_t$.
     - 2.c.5. Compute $\mathbf{A}_t^\top \mathbf{Y}_t$ and its Cholesky factorization to obtain $\Delta_t = (\mathbf{A}_t^\top \mathbf{Y}_t)^{-1}$.
     - 2.c.6. Compute $\mathbf{v}^t$

$$\begin{cases} \alpha_i = \Delta_i \mathbf{A}_i^\top \mathbf{v}^t \quad \text{and} \quad \mathbf{v}^t \leftarrow \mathbf{v}^t - \mathbf{Y}_i \alpha_i, & \text{for } i = t, \cdots, t-M+1, \\ \beta_i = \Delta_i \mathbf{Y}_i^\top \mathbf{v}^t \quad \text{and} \quad \mathbf{v}^t \leftarrow \mathbf{v}^t + \mathbf{A}_i(\alpha_i - \beta_i), & \text{for } i = t-M+1, \cdots, t. \end{cases}$$

     - 2.c.7. Set $\mathbf{y}^{t+1} = \mathbf{y}^t - \eta \mathbf{v}^t$.
   - 2.d. Update $\mathbf{x}^{k+1} = \mathbf{y}^m$

The following theorem proves that the proposed method has linear convergence.

Theorem 2.4 (Linear rate convergence of stochastic block BFGS [5])**.** *Suppose that*

$$\forall \mathcal{T} \subset \{1, \cdots, n\} (\forall \boldsymbol{x} \in \mathbb{R}^p \qquad \lambda \mathbb{I} \preceq \nabla^2_{\mathcal{T}} f(\boldsymbol{x}) \preceq \Lambda \mathbb{I},$$

where $\nabla^2_{\mathcal{T}} f(\boldsymbol{x}) = \frac{1}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} \nabla^2 f_i(\boldsymbol{x})$*. Then:*

1. *There exist* (γ, Γ) *such that* $\gamma \mathbb{I} \preceq \boldsymbol{H}_t \preceq \Gamma \mathbb{I}$.
2. *Suppose that* $\eta < \gamma\lambda/(2\Gamma^2\Lambda^2)$ *and that*

$$m \geq \frac{1}{2\eta(\gamma\lambda - \eta\Gamma^2\Lambda(2\Lambda - \lambda))}.$$

Then

$$\mathbb{E}[f(\boldsymbol{x}^k) - f^\star] \leq \rho^k (f(\boldsymbol{x}^0) - f(x^\star),$$

where

$$\rho = \frac{1/(2m\eta) + \eta\Gamma^2\Lambda(\Lambda - \lambda)}{\gamma\lambda - \eta\Gamma^2\Lambda^2}.$$

References

[1] C. Boutsidis, M. W. Mahoney, and P. Drineas. An improved approximation algorithm for the column subset selection problem. In *Proceedings of the twentieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 968–977. Society for Industrial and Applied Mathematics, 2009.

[2] R. Byrd, S. Hansen, J. Nocedal, and Y. Singer. A stochastic quasi-newton method for large-scale optimization. *SIAM Journal on Optimization*, 26(2):1008–1031, 2016.

[3] P. Drineas, R. Kannan, and M. W. Mahoney. Fast monte carlo algorithms for matrices ii: Computing a low-rank approximation to a matrix. *SIAM Journal on Computing*, 36(1):158–183, 2006.

[4] P. Drineas, M. W. Mahoney, and S. Muthukrishnan. Relative-error cur matrix decompositions. *SIAM Journal on Matrix Analysis and Applications*, 30(2):844–881, 2008.

[5] R. M. Gower, D. Goldfarb, and P. Richtárik. Stochastic block bfgs: Squeezing more curvature out of data. *arXiv preprint arXiv:1603.09649*, 2016.

[6] N. Halko, P.-G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2):217–288, 2011.

[7] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems*, pages 315–323, 2013.

[8] E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, and M. Tygert. Randomized algorithms for the low-rank approximation of matrices. *Proceedings of the National Academy of Sciences*, 104(51):20167–20172, 2007.

[9] P.-G. Martinsson, V. Rockhlin, and M. Tygert. A randomized algorithm for the approximation of matrices. Technical report, DTIC Document, 2006.

[10] J. Nocedal and S. Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[11] C.-T. Pan. On the existence and computation of rank-revealing lu factorizations. *Linear Algebra and its Applications*, 316(1):199–222, 2000.

[12] F. Roosta-Khorasani and M. W. Mahoney. Sub-sampled newton methods i: Globally convergent algorithms. *arXiv preprint arXiv:1601.04737*, 2016.

[13] F. Roosta-Khorasani and M. W. Mahoney. Sub-sampled newton methods ii: Local convergence rates. *arXiv preprint arXiv:1601.04738*, 2016.