New York University Tandon School of Engineering
Computer Science and Engineering

CS-GY 9223D: Homework 3.
Due Wednesday, November 25th, 2020, 11:59pm.

Collaboration is allowed on this problem set, but solutions must be written-up individually. Please list collaborators for each problem separately, or write "No Collaborators" if you worked alone.

Problem 1: Acceleration Through the Polynomial Lens

(15 pts) In Lecture 7, we saw how to analyze gradient descent for $f(\mathbf{x}) = \|\mathbf{A}\mathbf{x} - \mathbf{b}\|_2^2$, which has gradient $\nabla f(\mathbf{x}) = 2\mathbf{A}^T\mathbf{A}\mathbf{x} - 2\mathbf{A}^T\mathbf{b}$. The dominant cost for each gradient descent iteration is multiplying $\mathbf{x}$ by $\mathbf{A}^T\mathbf{A}$ to compute the gradient, which takes $O(nd)$ time when A is $n \times d$.

We obtained a convergence bound depending on the largest and smallest eigenvalues of $\mathbf{A}^T\mathbf{A}$, which we denote λ_1 and λ_d respectively. We did so by rearranging the gradient descent update rule:

$$\begin{aligned}
\mathbf{x}^{(i)} &= \mathbf{x}^{(i-1)} - \eta\left(2\mathbf{A}^T\mathbf{A}\mathbf{x}^{(i-1)} - 2\mathbf{A}^T\mathbf{b}\right) \\
\mathbf{x}^{(i)} - \mathbf{x}^* &= \mathbf{x}^{(i-1)} - \eta\left(2\mathbf{A}^T\mathbf{A}\mathbf{x}^{(i-1)} - 2\mathbf{A}^T\mathbf{A}\mathbf{x}^*\right) - \mathbf{x}^* \qquad \text{since } \nabla f(\mathbf{x}^*) = \mathbf{0}, \text{ so } \mathbf{A}^T\mathbf{A}\mathbf{x}^* = \mathbf{A}^T\mathbf{b} \\
\mathbf{x}^{(i)} - \mathbf{x}^* &= (\mathbf{I} - 2\eta\mathbf{A}^T\mathbf{A})(\mathbf{x}^{(i-1)} - \mathbf{x}^*).
\end{aligned}$$

By induction, it follows that the error $\mathbf{x}^{(i)} - \mathbf{x}^*$ equals $\mathbf{x}^{(i)} - \mathbf{x}^* = (\mathbf{I} - 2\eta\mathbf{A}^T\mathbf{A})^i(\mathbf{x}^{(0)} - \mathbf{x}^*)$. This allowed us to obtain a convergence bound by arguing that, if we set $\eta = 1/2\lambda_1$ where λ_1 is the largest eigenvalue of $\mathbf{A}^T\mathbf{A}$, then $(\mathbf{I} - \frac{1}{\lambda_1}\mathbf{A}^T\mathbf{A})^i$ has top eigenvalue $< \epsilon$ after $i = O(\frac{\lambda_1}{\lambda_d}\log(1/\epsilon))$ iterations. In this problem you will prove an "accelerated" version of this bound that only requires $O(\sqrt{\frac{\lambda_1}{\lambda_d}}\log(1/\epsilon))$ iterations.

1. Let p be a degree q polynomial. I.e. $p = c_0 + c_1 x + \ldots + c_q x^q$. Show that, for any p with $c_0 + c_1 + \ldots + c_q = 1$ and any starting vector $\mathbf{x}^{(0)}$, we can compute in $O(ndq)$ time a vector $\mathbf{x}^{(q)}$ such that:

$$\mathbf{x}^{(q)} - \mathbf{x}^* = p\left(\mathbf{I} - \frac{1}{\lambda_1}\mathbf{A}^T\mathbf{A}\right)(\mathbf{x}^0 - \mathbf{x}^*).$$

2. Prove that for $q = O(\sqrt{\frac{\lambda_1}{\lambda_d}}\log(1/\epsilon))$, there exists a polynomial p with coefficients $c_0 + c_1 + \ldots + c_q = 1$ such that the top eigenvalue of $p\left(\mathbf{I} - \frac{1}{\lambda_1}\mathbf{A}^T\mathbf{A}\right) \leq \epsilon$. **Hint:** What does this requirement on the coefficients tell us about the value of $p(0)$ and $p(1)$?

3. **Quick answer:** Write down a convergence bound for your new algorithm based on the above result.

Problem 2: Non-convex Optimization

(10 pts) Consider the problem of computing the top right singular vector $\mathbf{v}_1$ of a matrix $\mathbf{A} \in \mathbb{R}^{n \times d}$. As mentioned, it is possible to frame this problem as an optimization problem and solve it with gradient descent. As discussed in class, a benefit of doing so is that it makes it easier to introduce stochastic and online methods, and possible use projection to add additional constraints.

1. **Quick answer:** Assume we have know some coarse upper bound $\tilde{\lambda} \geq \lambda_1$. Let $f(\mathbf{x}) = \tilde{\lambda} \cdot \mathbf{x}^T\mathbf{x} - \mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x}$. It is easy to check that $\mathbf{v}_1 = \arg\min_{\mathbf{x} \in \mathcal{S}} f(\mathbf{x})$ where $\mathcal{S} = \{\mathbf{y} : \|\mathbf{y}\|_2^2 \geq 1\}$. Prove that $f(\mathbf{x})$ is a convex function, but $\mathcal{S}$ is not a convex set.

2. **Quick answer:** Since $\mathcal{S}$ is not convex, our analysis of projected gradient descent will give no guarantees for this problem. However, we could try using it anyway. Prove that power method is *exactly equivalent* to using projected gradient descent to solve $\arg\min_{\mathbf{x} \in \mathcal{S}} f(\mathbf{x})$ with a specific learning rate η.

3. Consider an alternative approach through unconstrained optimization. Let $g(\mathbf{x}) = -\frac{\mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x}}{\mathbf{x}^T\mathbf{x}}$. Now we have that $\mathbf{v}_1 \in \arg\min g(\mathbf{x})$. Prove that g is non-convex and derive an expression for its gradient $\nabla g(\mathbf{x})$. Show that $c \cdot \mathbf{v}_i$ is a statitionary point of g for any right singular vector $\mathbf{v}_i$ and scaling c.

4. While g's non-convexity also rules out a direct convergence bound: in theory gradient descent could converge to any singluar vector of $\mathbf{A}$, not the top one. However, we can argue this is unlikely to happen. In particular, we claim that for any $i \neq 1$, $\mathbf{v}_i$ is actually just a *saddle point* of g, not a local minimum. To prove this, it suffices to show that for any such $\mathbf{v}_i$, and any $t > 0$,

$$\text{There exists a perturbation } t\mathbf{z} \text{ with } \|\mathbf{z}\|_2 = 1 \text{ such that} \qquad g(\mathbf{v}_i + t\mathbf{z}) < g(\mathbf{v}_i).$$

   Prove the above. You can assume that $\mathbf{A}$ has unique singular values – i.e., $\sigma_1 > \sigma_2 > \ldots, \sigma_d$.

   If you are interested, you can find a some work on proving gradient methods won't get stuck at saddle points here https://arxiv.org/abs/1703.00887.

Problem 3: Locating Points via the SVD

(15 pts) Suppose you are given all pairs distances between a set of points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$. You can assume that $d \ll n$. Formally, you are given an $n \times n$ matrix $\mathbf{D}$ with $\mathbf{D}_{i,j} = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2$. You would like to recover the location of the original points, at least up to possible rotation and translation (which do not change pairwise distances).

Since we can only recover up to a translation, it may be easiest to assume that the points are centered around the origin. I.e. that $\sum_{i=1}^n \mathbf{x}_i = \mathbf{0}$.

1. Under this assumption, describe a polynomial time algorithm for learning $\sum_{i=1}^n \|\mathbf{x}_i\|_2^2$ from $\mathbf{D}$. Hint: expand $\|\mathbf{x}_i - \mathbf{x}_j\|_2^2$ as $(\mathbf{x}_i - \mathbf{x}_j)^T(\mathbf{x}_i - \mathbf{x}_j)$ and go from there.

2. Next, describe a polynomial time algorithm for learning $\|\mathbf{x}_i\|_2^2$ for each $i \in 1, \ldots, n$.

3. Finally, describe an algorithm for recovering a set of points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ which realize the distances in $\mathbf{D}$. Hint: This is where you will use the SVD! It might help to know (and prove to yourself) that $\mathbf{D}$ has rank $\leq d + 2$.

4. Implement your algorithm and run it on the U.S. cities dataset provided in `UScities.txt`. Note that the distances in the file are unsquared Euclidean distances, so you need to square them to obtain $\mathbf{D}$. Plot your estimated city locations on a 2D plot and label the cities to make it clear how the plot is oriented. Submit these images and your code with the problem set (in the same file, as plaintext) – I don't need to be able to run the code.

Problem 4: Faster Trace Estimation

(10 pts) A common task in linear algebra is to approximately compute the trace $\text{tr}(\mathbf{A})$ of a matrix $\mathbf{A}$ that you do not have explicit access to, but can multiply vectors by efficiently. For example, in machine learning, the trace of the Hessian $\mathbf{H}$ is often useful to compute. While constructing $\mathbf{H}$ and then computing its trace directly would be very expensive, the cost of computing $\mathbf{Hx}$ for any vector $\mathbf{x}$ is the same as the cost of two gradient evaluations – do you see why? (Look up "Pearlmutter's trick" if you don't.)

On the midterm, we analyzed a randomized algorithm for approximating the trace of any matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ using a small number of matrix-vector multiplication with $\mathbf{A}$. In particular, let $\mathbf{x}_1, \ldots, \mathbf{x}_m$ be vectors with uniformly random ± 1 entries and consider the estimator $\tilde{t} = \frac{1}{m}\sum_{i=1}^m \mathbf{x}_i^T(\mathbf{A}\mathbf{x}_i)$. We proved that:

$$\mathbb{E}[\tilde{t}] = \text{tr}(\mathbf{A}) \qquad \text{Var}[\tilde{t}] \leq \|\mathbf{A}\|_F^2.$$

We concluded via Chebyshev's that if $m = O(1/\epsilon^2)$, then with probability $9/10$, $\left|\text{tr}(\mathbf{A}) - \tilde{t}\right| \leq \epsilon\|\mathbf{A}\|_F$.

1. Assume $\mathbf{A}$ is symmetric and positive semi-definite with eiganvalues $\lambda_1 \geq \ldots \geq \lambda_n \geq 0$. For example, it might be a Hessian. Show that the above implies a relative error approximation:

$$(1 - \epsilon)\,\text{tr}(\mathbf{A}) \leq \tilde{t} \leq (1 + \epsilon)\,\text{tr}(\mathbf{A}).$$

   Hint: Use the fact that $\|\mathbf{A}\|_F^2 = \sum_{i=1}^n \lambda_i^2$ and $\text{tr}(\mathbf{A}) = \sum_{i=1}^n \lambda_i$.

2. **Quick answer:** Let $k = 1/\epsilon$ and let $\mathbf{V}_k$ be a span for the top eigenvectors of $\mathbf{A}$ (which are the same as its top right singular vectors since A is symmetric PSD). Prove that:

$$\text{tr}(\mathbf{A}) = \text{tr}(\mathbf{A} - \mathbf{A}\mathbf{V}_k\mathbf{V}_k^T) + \text{tr}(\mathbf{V}_k^T\mathbf{A}\mathbf{V}_k).$$

   Hint: You might want to use cyclic property of the trace.

3. Prove that $\|\mathbf{A} - \mathbf{A}\mathbf{V}_k\mathbf{V}_k^T\|_F^2 \leq \epsilon\,\text{tr}(\mathbf{A})^2$ for $k = 1/\epsilon$. **Hint:** Write both sides as sums involving eigenvalues.

4. Argue that using $m = O(1/\epsilon)$ matrix vector multiplications with $\mathbf{A}$, it is possible to find some estimate $\tilde{z}$ such that with probability 9/10:

$$\left|\tilde{z} - \text{tr}(\mathbf{A} - \mathbf{A}\mathbf{V}_k\mathbf{V}_k^T)\right| \leq \epsilon\,\text{tr}(\mathbf{A}).$$

5. **Quick answer:** $\mathbf{V}_k$ can be computed with roughly $O(k) = O(1/\epsilon)$ matrix vector multiplications using e.g block power method.[1] Conclude from the pieces above that with $O(1/\epsilon)$ matrix-vector multiplications total we can find some $\tilde{t}$ such that with probability 9/10 $(1-\epsilon)\,\text{tr}(\mathbf{A}) \leq \tilde{t} \leq (1+\epsilon)\,\text{tr}(\mathbf{A})$.

Note that this approach beats the naive estimator by a factor of $1/\epsilon$, which is quite a lot!

[1] For the sake of this problem, assume you get an exact answer in that time. It is possible to rigorously address the case when $\mathbf{V}_k$ is only computed approximately, but I'm not asking you to do that here.