Assignment 3
CS289: Algorithmic Machine Learning, Fall 2016
Due: November 4, 4PM

Guidelines for submitting the solutions:

- The assignments need to be submitted on Gradescope. Make sure you follow all the instructions - they are simple enough that exceptions will not be accepted.
- To save my eyes (and perhaps, more importantly, a few of your points) please use a good scanner and/or digitize the scan as per the instructions on the class web page.
- Start each problem or sub-problem on a separate page even if it means having a lot of white-space and write/type in large font.
- The solutions need to be submitted by 4 PM on the due date. No late submissions will be accepted.
- Please adhere to the code of conduct outlined on the class page.

1. For $\mu > 0$, a smooth function $f : \mathbb{R}^n \to \mathbb{R}$ is μ-*convex* if for all $x, y \in \mathbb{R}^n$,

$$f(y) \geq f(x) + \langle \nabla f(x), y - x \rangle + \frac{\mu}{2}\|y - x\|_2^2.$$

In this exercise we will show a better bound on convergence rate of gradient descent (GD) for such functions.

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a L-Lipschitz, μ-convex function. Let x^* be an optimal minimizer for f. Consider the GD algorithm with starting point x_0 and step-size $t = \mu/L^2$. Show that after k iterations we have

$$\|x_k - x^*\|_2^2 \leq \|x_0 - x^*\|_2^2 \left(1 - \frac{\mu^2}{L^2}\right)^k.$$

In other words, the distance of the k'th iterate to the optimal decreases exponentially in the number of iterations. [2 points]

[Hint: You may use the fact that $\nabla f(x^*) = 0$.]

Remark: Note that the number of iterations to get error ε for such functions is $O(\log(1/\varepsilon))$ for fixed x_0, μ, L (assuming $\mu < L$); this is exponentially better than the bound for general covex functions which was $O(1/\varepsilon)$.

2. A common heuristic used for SGD applied to ERM is *mini-batch* where instead of estimating the gradient by sampling one point at random you sample a few points. This operation typically decreases the variance; this problem quantifies this decrease. Consider an instance of ERM given by

$$f(\Theta) = \frac{1}{n}\sum_{j=1}^{n} f_j(\Theta),$$

where $f_j(\Theta) = \ell(h(\Theta, x^j), y^j)$ for data points $(x^1; y^1), \ldots, (x^n; y^n)$, $h(\Theta, \ )$ denotes a hypothesis parameterized by Θ and ℓ is a smooth loss function. Suppose that $\sup_\Theta \|\nabla f_j(\Theta)\|_2 \leq \alpha$ for all j. Let $T \geq 1$ be some parameter and consider the following estimator for the gradient at each iteration: Pick T points $(x^{j_1}; y^{j_1}), (x^{j_2}; y^{j_2}), \ldots, (x^{j_T}; y^{j_T})$ uniformly at random (with replacement) among all the data points; given Θ, use $\boldsymbol{v} = (1/T)\sum_{\ell=1}^{T} \nabla f_{j_\ell}(\Theta)$ as the estimator for $\nabla f(\Theta)$. Bound the variance of the estimator in terms of α, T. [2 points]

3. Let $w_1, \ldots, w_m \in \mathbb{R}^d$, $A \in \mathbb{R}^{m\times d}$, $b \in \mathbb{R}^m$. Write down an algorithm to compute a subgradient of the function $f(x) = \|Ax - b\|_1 + \sum_{i=1}^{m} RELU(\langle w_i, x\rangle)$. Explain why your algorithm for the subgradient works. [2 points]

4. The goal of this exercise is to implement and compare GD, SGM, and accelerated gradient descent for LSR. An important and beautiful algorithm in numerical optimization is Nesterov's accelerated gradient descent (NAGD). Given a function $f : \mathbb{R}^d \to \mathbb{R}$ that we like to minimize, a starting vector x_0, and a step-size $t > 0$, a simple version of NAGD is the following algorithm:

   (a) Set $x_{-1} = x_0$.

   (b) For $i = 0, \ldots,$:

      i. Set $y = x_i + ((i-1)/(i+2))(x_i - x_{i-1})$.

      ii. Set $x_{i+1} = y - t\nabla f(y)$.

   (Note the similarity to GD). Let $n = 600, d = 100$. Generate a random Gaussian matrix $A \in \mathbb{R}^{n\times d}$ (in Matlab, you can do this by $\texttt{A} = \texttt{randn(n, d)}$). Generate a random unit-norm vector $x^* \in \mathbb{R}^d$. Let $b = Ax^*$ and $f(x) = (1/n)\|Ax - b\|^2$. Generate another random unit-norm vector $x_0 \in \mathbb{R}^d$; this will be the starting vector you will use in the algorithms.

   (a) Implement and run GD for f with step-size $t_1 = 0.025$, starting point x_0 for $K_1 = 50$ iterations.

   (b) Implement and run NAGD for f with step-size $t_1 = 0.025$, starting point x_0 for $K_1 = 50$ iterations.

   (c) Implement and run SGD for f with step-size $t_2 = 0.005$, starting point x_0 for $K_2 = 300$ iterations. Use the estimator we discussed in class: sample a random row of A and use that to estimate the gradient.

   (Note that you need to start all three algorithms from the same starting point). You need to submit the following. [2 points]

   (a) Screenshots of the implementations of the three algorithms (the most important parts).

(b) Plot of the values of f against the number of iteration for GD and NAGD on the same figure. The drop off should be noticeably faster for NAGD.

(c) Plot of the values of f against the number of iterations for SGD. The drop off will be slower here and the curve will not be monotonically decreasing.[1]

5. The goal of this problem is to extend the analysis of compressed sensing we saw in class to signals that are only approximately sparse. Recall the definition of a C-good subspace: A subspace $S \subseteq \mathbb{R}^n$ is C-good if for every $x \in S$, and every subset $J \subseteq [n]$,

$$\|x_J\|_1 \leq C\sqrt{\frac{|J|}{n}}\,\|x\|_1 .$$

(Here, x_J denotes x restricted to the coordinates in J.) We say $A \in \mathbb{R}^{m\times n}$ is C-good if $kernel(A)$ is C-good.

Let x^* be an *approximately*-sparse vector in the following sense: Let I denote the k largest coordinates of x^* in absolute-value and suppose that $\|x^* - x^*_I\|_1 \leq \varepsilon$. The following questions show that we can recover a very good approximation for x^* given just $b = Ax^*$. In particular, let $y^* = \arg\min_y\{\|y\|_1 : Ay = b\}$ be the solution to the LP-relaxation we saw in class. Suppose that A is $(1/4)\sqrt{n/k}$-good. Prove that $\|x^* - y^*\|_1 \leq 4\varepsilon$. [2 points]

[1] Even though SGD is not as good as GD - comparing the two as is is not fair: each iteration of SGD only looks at one row of A. That is, in crude terms, n iterations of SGD cost the same as a single iteration of GD.