The University of Texas at Austin
Department of Electrical and Computer Engineering

EE381K: Large Scale Optimization — Fall 2017

Problem Set Three

Constantine Caramanis Due: Monday, October 23, 2017.

Computational Problems

Save your completed code in a file names `hw3.py`. Don't use stock optimization code, you should develop the core part of this assignment yourself. All plots should have titles, axis labels, and lines with different colors, markers, and legend labels.

1. Stochastic Variance Reduced Gradient Descent (SVRG)

   As we discussed in class, decomposable functions of the form

   $$\min_{\omega} \left[F(\omega) = \frac{1}{n} \sum_{i}^{n} f_i(\omega) \right],$$

   are very common in statistics/ML problems. Here, each f_i corresponds to a loss for a particular training example. For example, if $f_i(\omega) = (\omega^\top x_i - y_i)^2$, then $F(\omega)$ is a least squares regression problem. The standard gradient descent (GD) update

   $$\omega_t = \omega_{t-1} - \eta_t \nabla F(\omega_{t-1})$$

   evaluates the full gradient $\nabla F(\omega) = \frac{1}{n} \sum_i^n \nabla f_i(\omega)$, which requires evaluating n derivatives. This can be prohibitively expensive when the number of training examples n is large. SGD evaluates the gradient of one (or a small subset) of the training examples–drawn randomly from $1, ...n$–per iteration:

   $$\omega_t = \omega_{t-1} - \eta_t \nabla f_i(\omega_{t-1}).$$

   In expectation, the updates are equivalent, but SGD has the computational advantage of only evaluating a single gradient $\nabla f_i(\omega)$. The disadvatage is that the randomness introduces variance, which slows convergence. This was our motivation in class to introduce the SVRG algorithm.

   Given the dataset in `digits.zip`, plot the performance of GD, SGD, and SVRG for logistic regression with $l2$ regularization in terms of negative log likelihood on the training data against the number of gradient evaluations for a single training example (GD performs n such evaluations per iteration and SGD performs 1). Choose the $l2$ parameter to optimize performance on the test set. How does the choice of m affect the performance of SVRG? There should be one plot with a title and three lines with different colors, markers, and legend labels.

2. Using any approach, optimize performance of logistic regression on the test set in `news.zip` and compare the performance of your approach to standard SGD. This dataset is the full-dimensional newsgroup dataset (as opposed to the compressed version you worked with previously). The X matrices are stored in sparse matrix format and can be read using `scipy.io.mmread`. As the dataset is large and high-dimensional, you will have to decide on how best to allocate your computational resources. Try to utilize the sparsity of the data (i.e. don't just convert it to a dense matrix and spend all your time multiplying zeros). You may use any of the techniques covered in class or ideas from outside class (e.g. momentum, variance reduction, minibatches, adaptive learning rates, preprocessing). Describe your methodology and comment on what you found improved performance and why. Plot the performance (negative log likelihood) of your method against standard SGD in terms of the number of gradient evaluations.

3. **Mirror Descent**

   Consider the same problem of robust regression that you had on the last problem set. There, you solved it using SGD. Repeat the problem, but now using Mirror Descent. Specifically:

   (a) Write down the mirror descent update. For this, we will use the mirror map $\Phi(\beta) = \sum \beta_i \log \beta_i$. Compute the Bregman divergence, $D_\phi(\mathbf{x}, \mathbf{y})$ explicitly.

   (b) Using the data in `X.npy` and `y.npy`, and using stepsizes of your choosing, compare the projected subgradient method (from the last homework) with mirror descent. What is β? Plot the objective above against iterations for both methods (in a single plot).

4. **Matrix Completion**.

   In this problem we investigate **low-rank matrix completion**, the problem of finding a low-rank matrix given only a few (randomly sampled entries). While this is (clearly) not possible in general, somewhat remarkably, it is possible once some additional assumptions are made on the problem setup (for example, for a "random" low-rank matrix and random samples). The assumptions required for guarantees on matrix completion are beyond the scope of this class. Our goal is to develop a projected subgradient algorithm to solve such a problem.

   Suppose there is a true matrix $M \in \mathbb{R}^{m \times n}$ that we want to recover, but we are only given elements in the set $\Omega \subset [m] \times [n]$ (i.e. we know the value of m_{ij} if $(i, j) \in \Omega$). We want to solve the following constrained optimization problem

$$\begin{aligned} \min_X \quad & \|X\|_* \\ s.t. \quad & x_{ij} = m_{ij} \text{ for all } (i,j) \in \Omega \end{aligned}$$

   where the variable of optimization $X \in \mathbb{R}^{m \times n}$ is a matrix. Here $\|\cdot\|_*$ is the "nuclear" norm, equal to the sum of singular values of the matrix. This norm is a convex but not smooth function of X; we will implement projected sub gradient descent for this problem.

   The sub gradient of the $\|\cdot\|_*$ function is as follows: for any matrix X, if its SVD is $U\Sigma V'$, then a matrix $Z \in \partial\|X\|_*$ is in its sub gradient if and only if

$$Z = UV' + W$$

   where W is such that *(a)* the column and row spaces of W are perpendicular to the corresponding ones of X, and *(b)* the spectral norm $\|W\|_2 \leq 1$. Recall that the spectral norm of a matrix is its maximum singular value. Also recall that if X is rank r, then the matrices U, V are of sizes $m \times r$ and $n \times r$ respectively, and have orthonormal columns.

(a) Given a matrix X, how will you generate an element $Z \in \partial\|X\|_*$ using a singular value decomposition function in Python (e.g., in np.linalg)?

(b) Given a matrix X, how will you project it onto the feasible set (i.e. the set of matrices that satisfy the constraints) ?

(c) Implement projected sub gradient descent with two choices for step sizes: $\eta_k = \frac{1}{k}$ and $\eta_k = \frac{1}{\sqrt{k}}$. You will need to use the files contained in `MatrixCompletion.zip`, which contains two 100×100 matrices: a low-rank matrix M, and the matrix O that represents the set Ω by having entries that are 0 or 1 (in particular, $O_{ij} = 1$ means $(i, j) \in \Omega$).

(d) Plot the relative error $\frac{1}{100^2}\|M - X_k\|_F^2$ between the true matrix and the k^{th} iterate, as a function of k, for both step size choices; do so on one plot.

(e) What is the rank of the intermediate iterates? Why is this the case?

5. **Conjugate Gradient Algorithm.** In class we introduced the linear conjugate gradient algorithm. Download the file `ConjugateGradient.zip`. There you will find matrices and vectors defining two equations: $M_1 x = b_1$, and $M_2 x = b_2$. The solution, x^*, is there as well, although this is easy to find since both M_1 and M_2 are invertible. Use conjugate gradient to solve these two linear systems, and plot the error, $\log(\|x^{(k)} - x^*\|_{M_i}^2)$ vs. iteration k for both.

6. **Newton's Method**. This problem will demonstrate the two convergence behaviors of Newton's method, damped and quadratic, by simulation.

   Consider $f_m(x) = \|x\|^3 + \frac{m}{2}\|x\|^2$ for $m \in \{0, 0.0001, 0.001, 0.1\}$ and $x \in \mathbb{R}^5$.

   (a) For each m, implement Newton's method on $f_m(x)$ and provide the convergence plots, i.e $\log(\|x^{(k)} - x^*\|^2)$ vs. iteration k. Use the constant step size $t = 1$.

   (b) Using the condition for quadratic convergence, explain how and why your result changes according to m.

7. **Central Path**. Consider the linear optimization problem:

$$\begin{aligned} \min: \quad & 2x_1 + 4x_2 + x_3 + x_4 \\ \text{s.t.}: \quad & x_1 + 3x_2 + x_4 \leq 4 \\ & 2x_1 + x_2 \leq 3 \\ & x_2 + 4x_3 + x_4 \leq 3 \\ & x_i \geq 0, \quad i = 1, 2, 3, 4. \end{aligned}$$

   (a) Find a function F that is a self-concordant-barrier function, such that the closure of its domain is equal to the feasible set of the problem. (Recall that $-\log(a^\top x - b)$ is a self-concordant-barrier function, as you show in an exercise below.)

   (b) Find the analytic center x_F^* using Newton's method. You can initialize at any point in the domain (e.g., $(1/2, 1/2, 1/2, 1/2)$ or any other point you like).

$$x_F^* = \arg \min_{x \in \mathrm{dom} F} F(x).$$

   (c) Now you will generate the central path:

$$x^*(t) = \arg \min_{x \in \mathrm{dom} F} f(t; x),$$

where recall:

$$f(t;x) = tc^\top x + F(x).$$

For $t = 0$, the solution, and first point of the central path, is the analytic center. At each iteration, you will compute $t_{k+1} = t_k(1+\alpha)$. Experiment with different values of α. If α is too small, progress may not be that fast as t will grow slowly. If α is too big, we might move outside the region of quadratic convergence, and although t will grow more quickly, each individual step of the central path will take longer to compute.

(d) Plot the error, $\log(\|x^{(k)} - x^*\|)$ as a function of number of iterations, for different values of α.

Note: Chapter 11 in Boyd & Vandenberghe has much information about central path and barrier methods, although the chapter also contains a lot of information, definitions and ideas we have not discussed.

8. Do the same for the (slightly larger) LP contained in `LP_centralpath.zip`. In that file you will find specified: c, A, and b, thus defining the problem:

$$\begin{aligned} \min: \quad & c^\top x \\ \text{s.t.}: \quad & Ax \leq b \\ & x \geq 0. \end{aligned}$$

Note that you can use other LP solvers to quickly solve both this LP and the previous problem, in order to have the solution.

9. **Gradient and Newton**. Consider the function (called the Rosenbrock function)

$$f(x) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2.$$

This function is not convex, however it not hard to see that it has a unique minimizer $x^* = (1,1)$, and that in a neighborhood of this point, the Hessian is positive definite. Initializing at $x_{\text{init}} = (-1.2, 1)$, implement (a) gradient descent and (b) Newton, using a back-tracking line search for both. Plot the error in each as a function of the iteration.

Written Problems

1. Prove that a matrix Z as described above in the matrix completion problem is indeed a sub gradient to the nuclear norm function at X. You can use the following fact about the nuclear norm: for any matrix $M \in \mathbb{R}^{m\times n}$, let $s = \min(m,n)$. Then for any matrices $A \in R^{m\times s}$ and $B \in R^{n\times s}$ that have orthonormal columns, we have that

$$\|M\|_* \;\geq\; \langle M, AB' \rangle$$

2. (?) Re-do the computation we did in class, showing that if we use the mirror function

$$\Phi(\mathbf{x}) = \frac{1}{2}\|x\|_2^2,$$

then the Mirror Descent update for:

$$\begin{aligned} \min_{\mathbf{x}}: \quad & f(\mathbf{x}) \\ \text{s.t.}: \quad & \mathbf{x} \in \mathfrak{X}, \end{aligned}$$

is exactly projected subgradient descent.

3. Here you will do some work that helps compute the Mirror Descent update you need for the computational problem above. As we discussed in class, and also as is explained in Section 4.2 of Bubeck's notes, the Mirror Descent update can also be obtained as:

$$x_{t+1} = \arg\min_{x \in \mathcal{X} \cap \mathcal{D}} : \ \eta \langle g_{x_t}, x \rangle + D_\Phi(x, x_t).$$

Therefore, Mirror Descent is only computationally useful if we can easily solve the problem:

$$\min_{u \in \mathcal{X}} : \ \langle z, u \rangle + \Phi(u).$$

In this problem, you will show that when $\mathcal{X}$ is the simplex, i.e., $\mathcal{X} = \Delta_n$, then this problem is indeed easy.

(a) Consider the optimization problem:

$$\begin{aligned} \min : \quad & \langle z, u \rangle + \Phi(u) \\ \text{s.t.} : \quad & \sum_i u_i = 1. \end{aligned}$$

(Note that the constraints $\{u_i \geq 0\}$ are implicitly included as they are part of domΦ.) Write the Lagrangian for the problem. The variables will be u and a single variable λ for the single constraint. Write the KKT conditions for the problem.

(b) Using the KKT conditions, derive a closed form expression for u as a function of z.

(c) Now go back to the Mirror Descent update and write explicitly the Mirror Descent update using your work above.

4. **Affine Invariance of Newton's Method**

Let $f(x)$ be a strongly convex function. Let A be some invertible matrix, and consider the change of coordinates $Ay = x$, and accordingly, the function

$$\phi(y) = f(Ay).$$

(a) Consider some starting point x_0, and the sequence $\{x_1, x_2, \dots\}$ produced by performing gradient descent on $f(x)$ starting from x_0, and using step-size η_k at iteration k. Define also y_0 given by $Ay_0 = x_0$, and consider the sequence $\{y_1, y_2, \dots\}$ produced by performing gradient descent on $\phi(y)$ starting from y_0, and with the same step size. In general, these will not be the same, i.e., we will not have $Ay_k = x_k$ for all k. Show this, by providing a specific example of a function $f(\cdot)$ and a matrix A. This shows that gradient descent *is not affine invariant.*

(b) Now repeat this for Newton, where the updates $\{x_k\}$ and $\{y_k\}$ are generated by using *undamped* Newton's method on $f(x)$ and $g(y)$, respectively. Show that $Ay_k = x_k$ for all k. This shows that a change of coordinates does not affect what Newton's algorithm is doing.

5. **Gradient descent and non-convexity**

Consider the gradient descent algorithm with fixed step size η for the function $f(x) = x'Qx$, where Q is symmetric but not positive semidefinite. (i.e., Q has some negative eigenvalues). Exactly describe the set of initial points from which gradient descent, with *any* positive step size, will diverge. What happens at the other points, if the step size is small enough ?