Homework 2: Solutions

January 6, 2020

Problem 1

Consider the function $f(x_1, x_2, \ldots, x_m) = \sum_{i=1}^{m} a_i \cdot (x_i - b_i)^2 + 3$

1. Implement gradient descent for this function.
2. Implement gradient descent with backtracking (backtracking parameters $\alpha = 0.5, \beta = 0.5$)
3. Assume that $m = 500$, $a_i = 1, \forall i$, and b_i is chosen randomly (and uniformly) in $[0, 100]$. (a) Run both gradient descent and gradient descent with backtracking with initial point $x_i = 0, \forall i$. (b) Explain your choise of stepsize and stopping condition; (c) compare the convergence speed of the two algorithms.
4. Assume again that $m = 500$, but pick both a_i and b_i uniformly in $[1, 100]$. (a) Run both gradient descent and gradient descent with backtracking with initial point $x_i = 0, \forall i$. (b) Explain your choise of stepsize and stopping condition; (c) compare the convergence speed of the two algorithms; (d) explain the differences in running speed with the previous question (if any).

Solution

(1) and (2) This is a sum of squares so it is clearly a (strongly) convex function. The gradient for this function is just $\sum_i 2a_i \cdot (x_i - b_i)$. These first two question are just to write a short code for the basic iteration step (with and without backtracking). Various code snippets and version abound online for different languages. (Note: we will shortly also post some of our own code, for reference)

(3) It is easy to see that the (unique) solution for this problem is just $x_i = b_i$. But we'll pretend we don't know this and let the algorithm(s) find it. However, the a_i parameters affect the eigenvalues of the Hessian, and thus the Lipschitz constant, the condition number, etc. Hence, it also affects the optimal step-size for the gradient. In this first scenario, where all a_i are equal, the function is

very well conditioned (all eigenvalues are equal, and the isotropic curves are circles). In fact, this is the most convenient scenario for such a function, and the number of iterations should be minimum.

(a) If you did things right, you should observe "linear" convergence in both cases (i.e., a line on a semi-log scale, where you have the logarithm of the distance from the optimal value on the y axis, and the number of iterations, in linear scale, on the x axis, as usual). Also, if you are choosing the step size close to it's maximum allowable value (see (c) below), the gradient-based algorithm without backtracking should take a few less iterations (but still have same convergence *rate*). This should not be a suprise! (see (c)).

(b) For gradient without backtracking the min (let m) and max (let L) eigenvalues of the Hessian are the same and are equal to 2. Hence, the condition number is 1. For any step size $t_k < 2/(m+L) = 1/2$ should lead to convergence. For backtracking, there's no other parameters to deal with. The step-size is decided automatically. In terms of stopping condition there are dos and don'ts. Here are some general hints:

- You should *not* use the number of iterations as a stopping condition. You generally don't know the optimal value, so you can't know how far you are after, say, 100 iterations. Even for this very simple problem, 100 iterations might be enough to get a small error, if the initial point is not far from the optimal and the function is well conditioned, but might need 10000 iterations to converge if you change the parameters a bit. However, it *is* common to use a "maximum" number of iterations as a backup stopping condition, in order for your algorithm to stop eventually. if it's taking too long to converge.

- In this problem you actually know the optimal value ($x_i^* = b_i$) so you could use as stopping condition the distance to this point i.e. stop if $||x^k - x^*||_2 = \sqrt{\sum_{i=1}^{m}(x_i^k - x_i^*)^2} \leq \epsilon$, for some value of ϵ that you believe is small enough for your problem. Nevertheless, again, this is *not* the right stopping method (in fact this method is almost never applicable), because you never actually know the optimal value (if you did, why would you be implementing an optimization algorithm to find it anyway).

- The only thing that you *do* know holds for *any* convex function is that it's gradient goes to 0 at the optimal. Hence, your stopping criterion should relate to how small your gradient is. E.g., assuming you are at iteration k, and your variables are x^k, you can stop if point i.e. stop if $||\nabla f(x^k)||_2 = \sqrt{\sum_{i=1}^{m}(2a_i \cdot (x_i^k - b_i))^2} \leq \epsilon$.

  For such "small" problems (just 500 variables), it is common to require high accuracy, so $\epsilon = 10^{-6}, 10^{-7}$, or even less. In the case of machine learning problems, lower accuracy (i.e., higher ϵ values) are used, as you don't

just care to minimize the training error (which is what the optimization algorithm usually does), but you want your solution to generalize well. But this is an entire discussion of its own (when to stop in machine learning problems), especially when it comes to modern methods like DNNs.

(c) As explained earlier, this is a strongly convex function, so convergence rate should be "linear", i.e. $O(\log(1/\epsilon))$ for both algorithms. However, remember that backtracking is useful when you don't know the optimal step-size to use (e.g., can't really derive the eigenvalues needed). Backtracking will make sure the step-size is "good" at each step, without such knowledge, but it might take a few extra steps per iteration, compared to standard gradient, in order to find that "good" step. The tradeoff between the two is the following: without backtracking, if you choose the step-size poorly, you might be too slow, or worse diverge! (we saw some examples in the class); with backtracking, no such dangers, but you pay the overhead of a few extra calculations per iteration (if the first backtracking step did not lead to enough progres).

(4) The difference now is that the a_i values are not equal for all i. I.e., each term in the sum has a different "weight". Nevertheless, the optimal solution is still exactly the same! $x_i^* = b_i, \forall i$. Each little square in the sum can be minimized independently and the variables are not, as we say, "coupled" (which will be the case in Problem 2). Nevertheless, the convergence time of the algorithm is affected, because the condition number will be higher. The iso-curves are now elliptic, and rather elongated, which as we discussed slows down the gradient algorithm.

(a) You should still see both algorithms have linear convergence. The slope of the curves though should be smaller than in the previous scenario.

(b) No changes needed for backtracking or for the stopping condition (both algorithms). However, to ensure convergence for the basic gradient-based algorithms, the step size should be smaller now, as the condition number is quite larger. While the exact min and max eigenvalues will depend on the specific (random) a_i you will draw, based on standard probabilities you should expect that the min value is close to 1 and the max close to 100 (I invite the probability-savvy students to calculate the expected value for this min and max of 500 random variables drawn between 1 and 100, for fun).

(c) Comments similar to the previous use case (subquestion c) apply here to.

(d) Given the higher condition number, you should expect an increase in the number of iterations, compared to the previous scenario, that is proportional to the new condition number. E.g., if $a_i^{max}/a_i^{min} = 50$, then the respective algorithms should be (roughly) 50 times slower.

Problem 2

Consider the function $f(x_1, x_2, \ldots, x_m) = \sum_{i=1}^{m} a_i \cdot (x_i - b_i)^2 + 3$, with constraints $x_i \geq 0, \forall i$, $\sum_i x_i \leq 100$.

1. Write down the KKT conditions for this problem, and derive analytical expressions for the optimal primal and dual variables (or give insights as to how these could be calculated, like the examples we did in class).

2. Solve the problem using dual ascent. Assume again that $m = 500$, and pick both a_i and b_i uniformly in $[1, 100]$. Explain again your step size choice, stopping condition, and convergence rate observed.

3. Could the solution be parallelized? If so, estimate how many iterations a fully parallel algorithm would take (based on the number of iterations of the previous (non-parallelized) question.

Solution

Generic comments: compared to problem 1, where the optimal (unconstrained) solution was obvious by inspection ($x_i = b_i$), here the constraint $\sum_i x_i \leq 100$ "couples" all variables, introducing a maximum "budget" on the total values of x_i. Given that the average value of b_i is around 50, and there's around 500 squared terms to be minimized, it is clear that the optimal uncostrained value ($x_i = b_i$) will be infeasible and violate the constraint for the vast majority of random values you'll get. Hence the optimal constrained solution will be different from the optimal unconstrained one (this is typical of constrained optimization problems). What is more, finding the optimal solution by quick inspection is not possible anymore. Two opposing forces will now decide it:

- on the one hand, a large b_i value will push that x_i to higher values to keep that square term small enough.
- on the other hand, a large a_i value implies that term is more "important" to keep low.

(1). Objective and constraints are all convex. Hence, we can work with gradients (no need to involve subgradients here). The Lagrangian of this problem is

$$L(x_i, u_i, v) = \sum_{i=1}^{m} \left[a_i \cdot (x_i - b_i)^2 + 3\right] - \sum_{i=1}^{m} u_i \cdot x_i + v \cdot \left[\sum_{i=1}^{m} x_i - 100\right]. \quad (1)$$

KKT conditions are the following:

- *Stationarity:* $2a_i \cdot (x_i - b_i) - u_i + v = 0, \forall i$ which implies that $x_i = b_i - \frac{v - u_i}{2a_i}$
- *Complementary slackness:* $u_i \cdot x_i = 0, \forall i$.

- *Complementary slackness:* $v \cdot [\sum_{i=1}^{m} x_i - 100] = 0$.
- *Dual feasibility:* $u_i, v \geq 0$.
- *Primal feasibility:* $x_i \geq 0, \forall i$, $\sum_i x_i \leq 100$.

Observe that the solution x_i is the original unconstrained solution (b_i) reduced by some amount dependent on the lagrange multipliers, i.e., the dual solution. (It is easy to see that $v \geq u_i$; it is also satisfying to notice that the higher the importance a_i of that term, the smaller the subtracted term for that x_i, everything else equal). Finally, observe that the stationarity formula suggests that, if we find the optimal dual variables (e.g., via dual ascent, as requested later), we can immediately also get the optimal primal variables, as theory suggests.

If the budget constraint is inactive, then the optimal solution is obviously the unconstrained one (i.e. $x_i = b_i$). Otherwise (which will be the case for almost most of your experiments), this constraint is active and thus $v > 0$ (i.e., strictly positive).

If $u_i > 0$, then $x_i = 0$ (from the first complementary slackness condition. However, if $x_i > 0$, then $u_i = 0$. The latter implies that for non-zero allocated x_i, the term u_i disappears from the stationarity equation which further simplifies to $x_i = b_i - \frac{v}{2a_i}$.

Finally, the budget constraint being active suggests:
$\sum_i x_i = 100 \Rightarrow \sum_{i:x_i>0} b_i - \frac{v}{2a_i} = 100$
, which provides an equation that involes v.

Nevertheless, at this point, as usual, we cannot proceed further analytically with KKT conditions, and would require either some numerical solution (e.g., bisection), or to provide a gradient-ascent algorithm for the dual variables, which is exactly what is requested next.

(2) This just requires the standard gradient (ascent) steps of the dual variables. The only point you need to be careful is that you just need to do a projection back to the feasible region for the dual variables (i.e., make sure they stay non-negative).

1. $v^{(k+1)} = \max\{0, v^{(k)} + t_k \cdot (\sum_{i=1}^{m} x_i^{(k)} - 100)\}$,
2. $u_i^{(k+1)} = \max\{0, u_i^{(k)} - t_k \cdot x_i^{(k)}\}$,

where x_i^k is calculated at each step from the KKT stationarity condition using $u_i^{(k)}$ and $v^{(k)}$.

Stopping conditions are commonly build by looking at the dual gap. (Note: I will add some more details on step size and convergence rates later).

(3) The budget constraint ($\sum_{i=1}^{m} x_i^{(k)} \leq 100$) is the only one coupling the 500 variable x_i and the separate minimiation of the 500 squared terms in the objective. However, dual decomposition allows us to de-couple these problem through the Lagrangian, as already shown in the previous question. Given the dual solution $u_i^{(k)}$ and $v^{(k)}$ at that step, one can simply minimize each term x_i independently from all other x_j $(j \neq i)$

If $u_i^{(k)} > 0$, then set $x_i^{(k}) = 0$

If $u_i^{(k)} > 0$, then set $x_i^{(k)} = b_i - \frac{v^{(k)}}{2a_i}$.

These steps can be parallelized for the 500 different i. Hence, while the centralized solution would have to make 500 computations, one after the other to get the $x_i^{(k)}$ for iteration k, now these can be run in parallel (counting for 1 computation instead of 500, time-wise), despite the coupling constraints.

After this, the values must be communicated ("gather" phase) to a centralized location that performs the dual ascent step (shown in the previous question) to derive the dual variables, and the sends them back to each processor i that is responsible to derive the next $x_i^{(k+1)}$ values. Hence, we got rid of 499 computation steps (to calculate x_i) but introduced two communication steps, per iteration. This communication-computation trade-off is a standard problem these days with respect to distributed machine learning (and distributed optimization in general).