ECE 490: Introduction to Optimization **Spring 2022**

Supplementary Material for Note 8

Extensions of Newton's Method: Quasi-Newton, BFGS, Trust-Region

Lecturer: Bin Hu, Date:02/22/2022

Newton's method has some disadvantages. Now we discuss some potential fix.

8.1 A New Interpretation of Newton's Method

We start from a new interpretation of Newton's method. First, we consider the steepest descent method $x_{k+1} = x_k - \alpha \nabla f(x_k)$, which can be interpreted as follows. At each step k, we are actually solving a quadratic minimization problem

$$x_{k+1} = \arg\min_{x \in \mathbb{R}^p} \left\{ f(x_k) + \nabla f(x_k)^\mathsf{T}(x - x_k) + \frac{1}{2\alpha}\|x - x_k\|^2 \right\}$$

The quadratic cost $\left\{ f(x_k) + \nabla f(x_k)^\mathsf{T}(x - x_k) + \frac{1}{2\alpha}\|x - x_k\|^2 \right\}$ is just the sum of the Taylor expansion of f at x_k and an ℓ_2 regularizar. If we know f is L-smooth, then we know

$$f(x) \leq f(x_k) + \nabla f(x_k)^\mathsf{T}(x - x_k) + \frac{L}{2}\|x - x_k\|^2$$

So the gradient method with $\alpha = \frac{1}{L}$ is actually minimizing the above quadratic upper bound for f at each k.

Can we improve the optimization process by minimizing a better quadratic estimation of f at each k? This natural question leads to Newton's method, which iterates as

$$x_{k+1} = \arg\min_{x \in \mathbb{R}^p} \left\{ f(x_k) + \nabla f(x_k)^\mathsf{T}(x - x_k) + \frac{1}{2}(x - x_k)^\mathsf{T} \nabla^2 f(x_k)(x - x_k) \right\} \tag{8.1}$$

At each step k, we estimate f by its second-order Taylor expansion $f(x_k) + \nabla f(x_k)^\mathsf{T}(x - x_k) + \frac{1}{2}(x - x_k)^\mathsf{T} \nabla^2 f(x_k)(x - x_k)$ and then minimize this quadratic estimation. Intuitively, the Taylor expansion gives a good local estimate for f but may not give a good global estimation. Consequently, Newton's method sometimes does not even converge if the iterates are too far away from the optimal points.

8.2 Quasi-Newton Methods

Quasi-Newton Methods are a family of methods that follow the idea of Newton's Method but estimate the Hessian $\nabla^2 f(x_k)$ with some simpler matrix H_k. Specifically, Quasi-Newton methods have the iteration form:

$$x_{k+1} = x_k - \alpha_k H_k^{-1} \nabla f(x_k)$$

where H_k is some estimated version of $\nabla^2 f(x_k)$, and the stepsize α_k is typically determined by Armijo rule.

Recall that the idea of Newton's method is based on approximating the objective function $f(x)$ as a quadratic function via Taylor expansion:

$$f(x) \approx f(x_k) + \nabla f(x_k)^\mathsf{T}(x - x_k) + \frac{1}{2}(x - x_k)^\mathsf{T}\nabla^2 f(x_k)(x - x_k)$$

What if we estimate $\nabla^2 f(x_k)$ with some simpler matrix H_k? Specifically, we define

$$g(x) = f(x_k) + \nabla f(x_k)^\mathsf{T}(x - x_k) + \frac{1}{2}(x - x_k)^\mathsf{T} H_k(x - x_k)$$

Then we hope $g(x) \approx f(x)$ and optimize g for this step. What properties should H_k have such that g is a good estimate for f? It is reasonable to just enforce $\nabla f(x_k) = \nabla g(x_k)$ and $\nabla f(x_{k-1}) = \nabla g(x_{k-1})$. The condition $\nabla f(x_k) = \nabla g(x_k)$ is automatically satisfied. What about $\nabla f(x_{k-1}) = \nabla g(x_{k-1})$? This is equivalent to

$$H_k(x_k - x_{k-1}) = \nabla f(x_k) - \nabla f(x_{k-1}) \tag{8.2}$$

The above condition is called the secant equation. Therefore, we should choose H_k based on this condition. There are infinitely many H_k satisfying this condition. Various choices of H_k lead to different Quasi-Newton methods. We will talk about the most popular one, i.e. the BFGS method.

8.3 BFGS Method

We need H_k to be constructed in a way that it can be efficiently computed. It will be nice if H_k can be computed by some iterative formula $H_k = H_{k-1} + M_{k-1}$. Another nice property we want H_k to have is the positive definiteness. If H_k is positive definite, we can at least guarantee that the BFGS method is a decent method, i.e. $f(x_{k+1}) \leq f(x_k)$. Suppose we choose $H_0 > 0$ and then guarantee $M_k \geq 0$. Then by induction we have the positive definiteness of H_k. So one reasonable thing to do is to set up $\{H_k\}$ using the following iterative formula:

$$H_{k+1} = H_k + a_k v_k v_k^\mathsf{T} + b_k u_k u_k^\mathsf{T} \tag{8.3}$$

where $v_k \in \mathbb{R}^p$ and $u_k \in \mathbb{R}^p$ are some vectors. If $H_0 > 0$, the above iterative formula just guarantees H_k to be positive definite. How can we choose v_k and u_k to guarantee the secant equation $H_{k+1}(x_{k+1} - x_k) = \nabla f(x_{k+1}) - \nabla f(x_k)$? Let's denote $s_k = x_{k+1} - x_k$ and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$. The secant equation becomes $H_{k+1}s_k = y_k$. Substituting this into (8.3) leads to

$$y_k = H_{k+1}s_k = H_k s_k + a_k v_k v_k^\mathsf{T} s_k + b_k u_k u_k^\mathsf{T} s_k$$

Since $v_k v_k^\mathsf{T} s_k = v_k(v_k^\mathsf{T} s_k) = (v_k^\mathsf{T} s_k)v_k$ and $u_k u_k^\mathsf{T} s_k = u_k(u_k^\mathsf{T} s_k) = (u_k^\mathsf{T} s_k)u_k$, the above equation is just equivalent to

$$y_k - H_k s_k = a_k(v_k^\mathsf{T} s_k)v_k + b_k(u_k^\mathsf{T} s_k)u_k$$

How can we choose v_k, u_k, a_k, and b_k such that the above equation is satisfied? We can choose $v_k = y_k$ and $a_k(v_k^\mathsf{T} s_k) = 1$ so that the first terms on the left and right sides exactly match. Similarly, we can choose $u_k = H_k s_k$ and $b_k(u_k^\mathsf{T} s_k) = -1$ so that the second terms on the left and right sides exactly match. Therefore, we have $v_k = y_k$, $u_k = H_k s_k$, $a_k = \frac{1}{y_k^\mathsf{T} s_k}$, and $b_k = -\frac{1}{s_k^\mathsf{T} H_k s_k}$. The iteration formula (8.3) becomes

$$H_{k+1} = H_k + \frac{y_k y_k^\mathsf{T}}{y_k^\mathsf{T} s_k} - \frac{H_k s_k s_k^\mathsf{T} H_k}{s_k^\mathsf{T} H_k s_k} \tag{8.4}$$

where $s_k = x_{k+1} - x_k$ and $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$. This is exactly the BFGS method. When implementing the BFGS method $x_{k+1} = x_k - \alpha_k H_k^{-1} \nabla f(x_k)$, it will be better to directly update H_k^{-1} other than first obtaining H_k and then solving $H_k^{-1}\nabla f(x_k)$. Based on (8.4), one can use the matrix inversion lemma to show

$$H_{k+1}^{-1} = \left(I - \frac{s_k y_k^\mathsf{T}}{y_k^\mathsf{T} s_k}\right) H_k^{-1} \left(I - \frac{y_k s_k^\mathsf{T}}{y_k^\mathsf{T} s_k}\right) + \frac{s_k s_k^\mathsf{T}}{y_k^\mathsf{T} s_k} \tag{8.5}$$

You will be asked to show the above formula in Homework 3. Therefore, for the BFGS method, the computation cost is mainly required for updating (8.5). At each iteration, the main computation is doing the matrix multiplication twice and the cost scales with $O(p^2)$ if $x \in \mathbb{R}^p$. In contrast, Newton's method requires computing the Hessian $\nabla^2 f(x_k)$ and then solving the linear equation $\nabla^2 f(x_k) d_k = \nabla f(x_k)$. The cost for solving the linear equation $\nabla^2 f(x_k) d_k = \nabla f(x_k)$ scales with $O(p^3)$ in general [1]. Therefore, the per iteration computation cost for Newton's method is the cost for computing Hessian plus some value scaling with $O(p^3)$. This is much higher than the per iteration cost for the BFGS method which roughly scales with $O(p^2)$.

Locally, the BFGS method also achieves superlinear convergence. This is similar to Newton's method. One interpretation for the BFGS update (8.5) is that H_{k+1}^{-1} is chosen to be as close to H_k^{-1} as possible for some appropriate metric quantifying the distance between two matrices. We skip the details of these interpretations.

It is worth mentioning that the BFGS method requires storing H_k^{-1} in memory. When p is large, this could be an issue. Therefore, the limited-memory BFGS (L-BFGS) method is developed. We will not talk about L-BFGS in details in this course.

Compared with the gradient method, Newton's method typically requires much less iterations but the per iteration cost is significantly higher. The BFGS method can be viewed as an interpolation of the gradient method and Newton's method. There is another method

[1] When there is some sparsity in the Hessian matrix, one can solve this equation much faster. But in general, $O(p^3)$ is the required cost.

called the conjugate gradient method which can also be viewed as some interpolation of the gradient method and Newton's method. Due to the time constraint, we will not cover this method in the class.

8.4 Trust-Region Method and Cubic Regularization

We briefly mention a few variants for Newton's method. One issue for Newton's method is that the quadratic function $f(x_k) + \nabla f(x_k)^\mathsf{T}(x - x_k) + \frac{1}{2}(x - x_k)^\mathsf{T}\nabla^2 f(x_k)(x - x_k)$ may only be a good estimate for f when x is not far from x_k. What if we enforce x_{k+1} to be not far from x_k in the update? This is the idea of the trust region method. At each step k, the trust region method updates x_{k+1} as

$$x_{k+1} = \underset{\|x - x_k\| \le \Delta_k}{\arg\min} \left\{ f(x_k) + \nabla f(x_k)^\mathsf{T}(x - x_k) + \frac{1}{2}(x - x_k)^\mathsf{T}\nabla^2 f(x_k)(x - x_k) \right\} \tag{8.6}$$

So we restrict x_{k+1} to be in a trust region $\|x - x_k\| \le \Delta_k$. The parameter Δ_k can be tuned. When Δ_k is large, the trust region update behaves more similarly to Newton's method. The trust region method fixes the global convergence issue of Newton's method to some extent. It also gets a lot of recent attention due to its ability to escape saddle points. One can actually show that the trust region method can escape strict saddle points under some assumptions.

One can also add higher order term $\|x - x_k\|^3$ to the quadratic estimation $f(x_k) + \nabla f(x_k)^\mathsf{T}(x - x_k) + \frac{1}{2}(x - x_k)^\mathsf{T}\nabla^2 f(x_k)(x - x_k)$. This is the idea of cubic regularization.