CS 542G: Understanding SD, Newton

Robert Bridson

October 27, 2008

1 Understanding Steepest Descent

Last time we finished with a basic Steepest Descent algorithm using backtracking line search and a basic convergence test. This is a solid, reliable algorithm that is expected to converge to a local minimum at least for any well-posed convex problem, and probably many more general cases too, though proving that isn't trivial.

However, it would serve us very well to take a closer look at analyzing how fast SD might be, and what factors might slow it down. We won't do this too rigourously, but will try to do it in a way that exposes both intuition for what's going on and more importantly demonstrates the general process of taking a reasonable-looking algorithm for a hard problem and trying to evaluate and/or improve it. This process is absolutely critical to scientific computing and its applications.

1.1 Model Problems

The first step, generally, is to replace the hard problem (minimizing a general function f) with a **model problem**: something simpler to understand and work with analytically which nevertheless captures at least some of the essence of the original hard problem. We did this before when talking about interpolation, trying out a 1D version of the general high dimension problem first. It's always a good idea.

In this case, we'll use Taylor series to replace the general f with something easier: a polynomial. The simplest polynomial which can have a well-defined minimum is a quadratic, so that's where we'll stop the series:

$$f(x + \Delta x) \approx f(x) + \nabla f(x)\Delta x + \frac{1}{2}\Delta x^T H(x)\Delta x + O(\Delta x^3)$$

To understand convergence, obviously we should expand the Taylor series around a point x which is a minimum of f, which means $\nabla f(x) = 0$. We can also subtract off the value $f(x)$ at the minimum without changing the problem, and translate our coordinate system so the minimum is in fact at $x = 0$. This leaves us with the following model problem, after relabeling Δx to x:

$$\min_x \frac{1}{2} x^T H x$$

where H is a symmetric matrix representing the Hessian of f at a minimum. Note that the gradient of this model function is just Hx.

Of course, H can't be any old symmetric matrix: it can be proven that the Hessian of a smooth function at a local minimum must be symmetric positive semi-definite (if there were negative eigenvalues, a perturbation in those eigenvector directions would find even smaller values of f than the local minimum which is impossible). In fact, if we want our model problem to be well-posed, we have to strengthen this to SPD—ruling out zero eigenvalues. However, we need to keep in mind that a real-world f may be both well-posed and have a singular Hessian at the minimum, so we'll be particularly interested in how SD copes with an ill-conditioned H.

The next difficulty we face is that the line search methods we discussed aren't easy to boil down to a simple formula: they involved loops with logic tests etc. Instead we'll model this as well by assuming a "perfect" line search, that exactly minimizes the objective along the search direction—which isn't too hard at all for the quadratic we have. We need to find step size α minimizing:

$$\frac{1}{2}(x + \alpha d)^T H (x + \alpha d)$$

Expanding this out and setting the derivative with respect to α eqal to zero gives:

$$\begin{aligned} \frac{\partial}{\partial \alpha}\left[\frac{1}{2} x^T H x + \alpha d^T H x + \frac{1}{2}\alpha^2 d^T H d\right] &= 0 \\ d^T H x + \alpha d^T H d &= 0 \\ \alpha &= -\frac{d^T H x}{d^T H d} \end{aligned}$$

Putting this together with the gradient above, our Steepest Descent model looks like this:

- Start with initial guess $x^{(0)}$.
- For $k = 0, 1, \ldots$
  - Pick $d = -Hx^{(k)}$, $\alpha = -\frac{d^T H x}{d^T H d}$
  - Update $x^{(k+1)} = x^{(k)} + \alpha d$

The question to answer is how fast this converges to $x = 0$.

We can actually simplify our model a bit further, without loss of generality assuming we've rotated coordinates along eigenvectors, diagonalizing H. In particular, let's assume $H = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$, with $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_n > 0$. In this case the i'th entry of the update boils down to:

$$x_i^{\text{new}} \leftarrow x_i - \frac{\sum_{j=1}^n \lambda_j^2 x_j^2}{\sum_{j=1}^n \lambda_j^3 x_j^2} \lambda_i x_i$$

We can get a feel for what's going on by ignoring the possible variations in the components of x: then all components of x are reduced, but the component for the biggest eigenvalue will be reduced λ_1/λ_n more than the component for the smallest eigenvalue. With some more effort that we won't go through, it can eventually be proven that error will (worst case) be reduced linearly at a rate bounded by

$$1 - \frac{\lambda_n}{\lambda_1} = 1 - \frac{1}{\kappa(A)}$$

That is, SD slows down according to how ill-conditioned the Hessian matrix is, with the components in the smallest eigenvalue spaces converging slowest. If the condition number is huge, those components will converge very slowly indeed.

Some further investigation, for example experimenting with normalizing x after each SD step above, shows that as x converges to zero, in the limit the components corresponding to λ_1 and λ_n dominate all others and are in the ratio $\lambda_n : \lambda_1$. Try plugging in $x = (s\lambda_n, 0, \ldots, 0, \pm s\lambda_1)$ to the update above to see how this is a fixed point (up to a reduction of s) of the iteration. This means that even the convergence of the component of the largest eigenvalue eventually is stalled by the effect of the smallest eigenvalue. Initially, of course, the large eigenvalue components converge very nicely—the steepest descent direction $-Hx$ clearly responds best to errors in those components, and most weakly to the small eigenvalue components.

The summary of all this is that Steepest Descent is reliable—and can make fast progress initially—but is slow to finish converging when the Hessian is ill-conditioned.

2 Newton's Method

In the model problem above we could see the convergence problems are related to the fact that the optimal direction to the minimum at zero is $-x$, but the steepest descent direction is instead $-Hx$ which unfortunately scales down the small eigenvalue components. In a real problem we of course don't know the exact direction to the true minimum (we don't have direct access to "x")—but we can evaluate the Hessian and then form $-H^{-1}\nabla f$ which reduces to $-x$ in the model problem. This is the heart of **Newton's method**: picking a smart direction by using the Hessian. This puts it in the third category of solvers, where we assume we do have access to the second derivatives of the objective function f.

Let's derive Newton more formally. We'll take the same approach as we did for analyzing SD, modeling the objective using a quadratic polynomial from the Taylor series, but we'll build the method out of this model. That is, since minimizing a general objective $f(x)$ is hard, we'll temporarily replace $f(x)$ with a **model objective** function which is just quadratic (approximating $f(x)$ with a Taylor series) and easier to minimize. With guess $x^{(k)}$ in Newton, the new model function is:

$$f(x^{(k)} + \Delta x) \approx f(x^{(k)}) + \nabla f(x^{(k)})\Delta x + \frac{1}{2}\Delta x^T H(x^{(k)})\Delta x$$

and we want to find a Δx that minimizes it, giving the next Newton guess $x^{(k+1)} = x^{(k)} + \Delta x$.

Setting the derivative w.r.t. Δx of the model objective to zero gives:

$$\begin{aligned} \nabla f(x^{(k)}) + H(x^{(k)})\Delta x &= 0 \\ \Leftrightarrow \qquad H(x^{(k)})\Delta x &= -\nabla f(x^{(k)} \end{aligned}$$

We thus have a symmetric linear system to solve for Δx, with formal solution $\Delta x = H^{-1}\nabla f$.

This, then, is the plain Newton algorithm:

- Start with guess $x^{(0)}$
- For $k = 0, 1, 2, \ldots$ until convergence
  - Evaluate $\nabla f(x^{(k)}$ and $H(x^{(k)}$
  - Solve the linear system $H\Delta x = -\nabla f$
  - Update $x^{(k+1)} = x^{(k)} + \Delta x$

Once this starts to converge, it converges extremely efficiently: the $O(\Delta x^3)$ error in our model function means the Newton guess is extremely accurate, and we get **quadratic convergence**. That is, for k large enough and for some constant C, the error satisfies

$$\|e_{k+1}\| \leq C\|e_k\|^2$$

This is not quite as fast as the cubic convergence we saw with Rayleigh Quotient Iteration, but it still means we can expect to get double the number of accurate digits each iteration in the convergence regime—which for double precision means we'll be finished in 3–5 iterations.

The main problem with Newton, however, is that getting to the point of convergence is not at all guaranteed. Unlike Steepest Descent, there is no guarantee we'll make any improvement, and in fact a Newton step far from the minimum might take us even further away. There's also the issue that the linear system might not even be well-posed: H could be singular. All this means we have to do some work making Newton more robust before we can use it as a general-purpose method.

We'll skip over the question of a bad H for now, and assume the H we get is in fact SPD (so minimizing the model objective is well-posed); we'll return to this next lecture. The other problem is that far from convergence, there's no guarantee that $x^{(k+1)}$ is going to be closer to the solution than $x^{(k)}$; even in 1D it's simple to construct an example where it's drastically further away.

We can significantly improve the reliability of Newton by instead thinking of the solution Δx as a suggested search direction, with $\alpha = 1$ the natural step size, and then use that to start off line search. In the convergence regime, line search will return immediately because $\alpha = 1$ will make an improvement; far away, α will be reduced until we do get an improvement.