B553 Lecture 4: Gradient Descent

Kris Hauser

January 24, 2012

The first multivariate optimization technique we will examine is one of the simplest: gradient descent (also known as steepest descent). Gradient descent is an iterative method that is given an initial point, and follows the negative of the gradient in order to move the point toward a critical point, which is hopefully the desired local minimum. Again we are concerned with only local optimization, because global optimization is computationally challenging.

Gradient descent is popular for very large-scale optimization problems because it is easy to implement, can handle "black box" functions, and each iteration is cheap. Its major disadvantage is that it can take a long time to converge. We will also describe a discrete descent method often used to solve large combinatorial problems.

1 Gradient descent

Given a differentiable scalar field $f(\mathbf{x})$ and an initial guess $\mathbf{x}_1$, gradient descent iteratively moves the guess toward lower values of f by taking steps in the direction of the negative gradient $-\nabla f(\mathbf{x})$. Locally, the negated gradient is the steepest descent direction, i.e., the direction that $\mathbf{x}$ would need to move in order to decrease f the fastest. The algorithm typically converges to a local minimum, but may rarely reach a saddle point, or not move at all if $\mathbf{x}_1$ lies at a local maximum.

1.1 The gradient is the steepest descent direction

The first order Taylor approximation of $f(\mathbf{x})$ about $f(\mathbf{x}_1)$ is

$$f(\mathbf{x}) = f(\mathbf{x}_1) + \nabla f(\mathbf{x}_1) \cdot (\mathbf{x} - \mathbf{x}_1) + O(||\mathbf{x} - \mathbf{x}_1||^2). \tag{1}$$

Consider moving from $\mathbf{x}_1$ a small amount h in a unit direction $\mathbf{u}$. We want to find the $\mathbf{u}$ that minimizes $f(\mathbf{x}_1 + h\mathbf{u})$. Using the Taylor expansion, we see that

$$f(\mathbf{x}_1 + h\mathbf{u}) - f(\mathbf{x}_1) = h\nabla f(\mathbf{x}_1) \cdot \mathbf{u} + h^2 O(1). \tag{2}$$

If we make the h^2 term insignificant by shrinking h, we see that in order to decrease $f(\mathbf{x}_1 + h\mathbf{u}) - f(\mathbf{x}_1)$ the fastest we must minimize $\nabla f(\mathbf{x}_1) \cdot \mathbf{u}$. The unit vector that minimizes $\nabla f(\mathbf{x}_1) \cdot \mathbf{u}$ is $\mathbf{u} = -\nabla f(\mathbf{x}_1)/||\nabla f(\mathbf{x}_1)||$ as desired.

1.2 Algorithm

The algorithm is initialized with a guess $\mathbf{x}_1$, a maximum iteration count N_{max}, a gradient norm tolerance ϵ_g that is used to determine whether the algorithm has arrived at a critical point, and a step tolerance ϵ_x to determine whether significant progress is being made. It proceeds as follows.

```
1. For $t = 1, 2, \ldots, N_{max}$:
2.      $\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t - \alpha_t \nabla f(\mathbf{x}_t)$
3.      If $||\nabla f(\mathbf{x}_{t+1})|| < \epsilon_g$ then return “Converged on critical point”
4.      If $||\mathbf{x}_t - \mathbf{x}_{t+1}|| < \epsilon_x$ then return “Converged on an $x$ value”
5.      If $f(\mathbf{x}_{t+1}) > f(\mathbf{x}_t)$ then return “Diverging”
6. Return “Maximum number of iterations reached”
```

The variable α_t is known as the *step size*, and should be chosen to maintain a balance between convergence speed and avoiding divergence. Note that α_t may depend on the step t.

1.3 Choosing a step size using line search

To a first-order approximation, each step decreases the value of f by approximately $\alpha_t||\nabla f(\mathbf{x}_0)||^2$. If α_t is too small, then the algorithm will converge very slowly. On the other hand, if the step size α_t is not chosen small enough, then the algorithm may fail to reduce the value of f. (Because the first order approximation is valid only locally.)

One approach is to *adapt* the step size α_t in order to achieve a reduction in f while still making sufficiently fast progress. This procedure is known as a *line search* and is employed in many multivariate optimization algorithms. The input to a line search is a function f, an initial point $\mathbf{x}$ and direction $\mathbf{d}$, initial candidate step size γ_1. The output is a step size $\gamma > 0$ such that $f(\mathbf{x} + \gamma\mathbf{d}) < f(\mathbf{x})$, or failure if the step size falls below some threshold.

The first step is to check whether $\mathbf{x} + \gamma_1\mathbf{d}$ decreases the value of f. If so, then we can either terminate the line search, or search for a larger valid γ by repeated doubling: $\gamma_n = \gamma_1 2^n$. If a step of size γ_1 increases f, then we search through repeated halving: $\gamma_n = \frac{\gamma_1}{2^n}$. This continues until a reduction in f is found, or minimum step size tolerance is reached.

You might be thinking, "wouldn't it make sense for line search to find the *optimal* step size rather than any one that reduces the function value?" It turns out that in practice it is usually not worthwhile to do extra work to obtain an optimal line search because moving along the current line typically doesn't get you much closer to the local minimum. More often it is a better use of time to take a step and get a new gradient direction at the next point. Nevertheless it is sometimes useful during analysis to assume that optimal line searches are being performed.

1.4 Pathological cases

It can be proven, through a relatively technical proof, that gradient descent with an optimal line search obtains a linear convergence. However, there are cases in which the convergence constant is very bad, that is, the error ratio E_{t+1}/Et is close to 1. This occurs when the eigenvalues of f's second derivative matrix, known as the Hessian matrix, is poorly "scaled" at the minimum. We will learn more about the Hessian matrix in future lectures, but here we will provide some intuition for the causes of slow convergence.

Consider a very simple quadratic function in a two dimensional Euclidean space: $f(x_1, x_2) = ax_1^2 + bx_2^2$, with positive constants a and b. The gradient of f is $(2ax_1, 2bx_2)$, and of course the minimum of f is at the origin.

If $a = b$, then f increases isotropically from the origin in every direction. Another way to think of this is that the level sets of f are circles. At a point (x_1, x_2), observe that $-\nabla f(x_1, x_2)$ points directly toward the origin. Hence, gradient descent moves any initial point directly toward the origin, and it will perform well.

Now consider the case where $b >> a$, say $b = 100a$. Now the level sets of f are ellipses that are thinner in the x_1 direction. At a point (x_1, x_2), the descent direction $-\nabla f(x_1, x_2) = (-2ax_1, -2bx_2)$ is steeper along the x_2

axis than the x_1 axis. So, the line search will not be directed toward the origin. This gives gradient descent a characteristic zig-zagging trajectory that takes longer to converge to the minimum. In general, slow convergence holds when the "bowl" around the local minimum is much thinner in one direction than another. Problems like this are said to have an *ill-conditioned* Hessian matrix. We will make this more precise in future lectures.

2 Variants

2.1 Steepest descent in discrete spaces

Gradient descent can be generalized to spaces that involve a discrete component. The method of steepest descent is the discrete analogue of gradient descent, but the best move is computed using a local minimization rather rather than computing a gradient. It is typically able to converge in few steps but it is unable to escape local minima or plateaus in the objective function.

A discrete search problem is defined by a discrete state space S and a set of edges $E \subseteq S \times S$. This induces a graph $G = (S, E)$ that must be searched in order to find a state that minimizes a given function $f(s)$. A steepest descent search begins at a state s_0 and takes steps on G that descend $f(s)$ with the maximum rate of descent.

The algorithm repeatedly computes each transition $s \to s'$ by performing the local minimization $\arg\min_{s \to s' \in E} f(s')$, and terminates when $f(s') \geq f(s)$. This approach is typically much faster than an exhaustive search if S is large or possibly infinite but the degree of G is small.

2.2 Hill Climbing

Hill climbing is a similar approach to steepest descent that is used for large discrete problems in which the state space is combinatorial. (Although the name is hill *climbing* the approach can be applied to either minimization or maximization.) If the state is composed of several attributes $s = (s_1, \ldots, s_n)$, an optimization can be formulated over the composite state space $S = S_1 \times \cdots \times S_n$. In each iteration, each of the attributes s_i is locally optimized in S_i using a single step of steepest descent. These sweeps continue until convergence or a termination condition is reached.

2.3 Coordinate Descent

Using a similar approach as to hill climbing, coordinate descent (CD) is a method that performs line searches along each axis of a Euclidean space. The algorithm sweeps through all the axes in a round robin fashion. This approach is most helpful when the function f admits exist fast analytical expressions for solving the optimal line searches. CD methods are used in inverse kinematics for robotics and character animation, as well as in the expectation-maximization algorithm that we will see later in the course. CD can, however converge slowly when f has channels in non-axis aligned directions.

2.4 Nonlinear Conjugate Gradient Method

The *nonlinear conjugate gradient method* is a technique that chooses better line search directions than steepest descent, under the assumption that the function to be optimized is nearly quadratic at the minimum. The algorithm is only slightly harder to implement than gradient descent, but explaining its derivation is rather complex. Briefly, it is based on the conjugate gradient method for finding the solution to the matrix equation $A\mathbf{x} = \mathbf{b}$, where A is a positive semidefinite matrix. It tends to work better than gradient descent but not as well as the Newton-type methods we will present in future lectures.

2.5 Random Restarts

Local optimization techniques are not guaranteed to find the global minimum unless it can be proven that the function under question has only a single minimum or that the initial guess is within the *basin of attraction* of the global minimum. Moreover, hard optimization problems are often riddled with local minima. Nevertheless it is possible to improve the chance of finding a global minimum by performing many local optimizations with randomly chosen initial points, and picking the locally optimized point with the lowest f value. This strategy is known as steepest descent with random restarts. Quantifying the probability that random restarts find a global minimum is left as an exercise.

3 Exercises

1. Show that for a decomposible function $f(x_1, x_2) = g(x_1) + h(x_2)$, the $(x_1^\star, x_2^\star)$ that optimizes f can be computed by optimizing $g(x_1)$ and $h(x_2)$ individually. If each evaluation of g, ∇g, h, and ∇h has cost 1, determine the number of operations saved by performing individual gradient descents rather than an overall gradient descent (make assumptions about the number of steps taken until convergence).

2. For the quadratic function $f(x_1, x_2) = ax_1^2 + bx_2^2$, find an exact expression for the next $\mathbf{x}_{t+1}$ computed by an optimal line search starting from $\mathbf{x}_t$. Determine the rate of convergence from this expression.

3. If p is the probability of sampling an initial guess in the basin of attraction of the global minimum, then what is the probability that one of n random restarts reach a global minimum?