10-425/625: Introduction to Convex Optimization (Fall 2023)

Lecture 7: Gradient Descent

Instructor:[1] *Matt Gormley* *September 18, 2023*

7.1 Gradient Descent

For the next couple of lectures we'll focus on a basic unconstrained optimization problem:

$$\min_{x \in \mathbb{R}^d} f(x).$$

For most of today we'll also assume that f is differentiable everywhere. A classical method to solve such optimization problems is gradient descent, i.e. we initialize at some guess x^0 and execute iterations of the form,

$$x^{t+1} = x^t - \eta \nabla f(x^t),$$

for some choice of the step-size $\eta > 0$, and until we reach some stopping condition.

Algorithm 1 Gradient Descent

1: Choose initial point $x^0 \in \mathbb{R}^n$
2: **for** $t = 1, 2, \ldots, T$ **do**
3: Compute the gradient $g = \nabla f(x^t) \in \mathbb{R}^n$
4: Update the point: $x^{t+1} = x^t - \eta_t g$
5: Stop when $\|\nabla f(x^t)\|_2^2 < \epsilon$ for some small $\epsilon > 0$

7.1.1 Motivation #0: Moving to the Nearest Valley

Gradient descent is a local optimization algorithm, which means that it converges to a nearby local minimum. Since the gradient $\nabla f(x)$ is point in the

[1] These notes were originally written by Siva Balakrishnan for 10-725 Spring 2023 (original version: here) and were edited and adapted for 10-425/625.

direction of steepest ascent, we take steps in the opposite direction $-\nabla f(x)$ and gradually move towards such a local minimum.

For a strictly convex function where the minimizer exists and is unique, gradient descent will be moving towards the same local minimum (a global minimum) regardless of where it begins. Figure 7.1 shows this case.

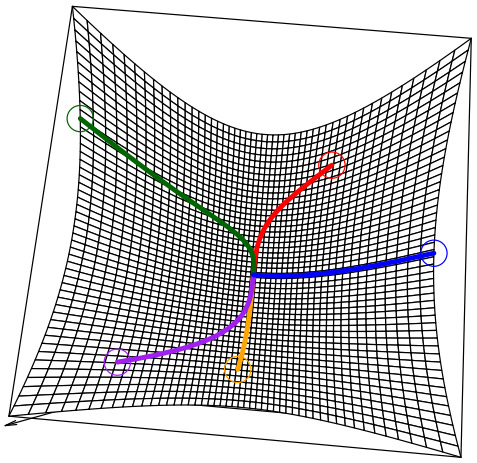

Figure 7.1: Gradient descent on a convex function with random initializations

For a nonconvex function, our choice of the initial point and step size will determine which local minimum (or saddle point) we arrive at — and there may be many, as in the example in Figure 7.2.

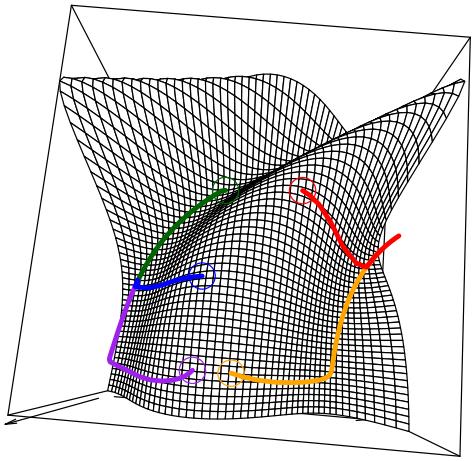

Figure 7.2: Gradient descent on a nonconvex function with random initializations

7.1.2 Motivation #1: Descent Directions

There are many ways to motivate this algorithm. One is to notice that if we were at a point x and moved in a direction v with step-size $\eta > 0$

$$f(x + \eta v) \geq f(x) + \eta v^T \nabla f(x).$$

So at the very least we'd like to ensure that the second term is negative, i.e. $v^T \nabla f(x) \leq 0$ (otherwise we're moving to a strictly worse point). Such directions (which make a larger than 90-degree angle with the gradient) are typically called *descent* directions (for f at x).

So how do we choose v s.t. $v^T \nabla f(x) \leq 0$?

It should be clear that the negative gradient $v = -\nabla f(x)$ always gives us a descent direction (and in some sense gives us one for which the term $v^T \nabla f(x)$ is most negative – amongst vectors v with a given norm). Why? Recall that for any vector w, $\|w\|_2^2 = w^T w \geq 0$. So $-\|\nabla f(x)\|_2^2 \leq 0$.

At first glance it might seem remarkable that gradient descent works at all. Note that the direction of steepest descent is not necessarily pointing towards the minimum of the function. And yet if we continue taking little steps in this direction, we will show that we eventually converge to a local minimum.

7.1.3 Motivation #2: Gradient Descent as Minimizing the Local Linear Approximation

A more interesting way to motivate GD (which will also be subsequently useful to motivate mirror descent, the proximal method and Newton's method) is to consider minimizing a linear approximation to our function (locally).

Constrained Version Suppose we approximate our true function f by the linear objective below. Then we optimize the linear objective subject to the constraint that our solution y is not too far away from our current x^t.

$$\begin{aligned} x^{t+1} = &\arg\min_{y \in \mathbb{R}^d} f(x^t) + \nabla f(x^t)^T (y - x^t) \\ &\text{s.t. } \frac{1}{2}\|y - x^t\|_2^2 \leq \epsilon, \end{aligned}$$

Unconstrained Version We could instead minimize an unconstrained version of the above problem, where we use a soft constraint. The result is a local quadratic approximation of the function. A picture will be helpful. With a picture in mind, we can view GD as solving the following local minimization problem:

$$x^{t+1} = \arg\min_{y \in \mathbb{R}^d} f(x^t) + \nabla f(x^t)^T (y - x^t) + \frac{1}{2\eta}\|y - x^t\|_2^2,$$

where the second term behaves as a regularizer to ensure that (for small η) our update remains close to our current iterate x^t. This local optimization problem has a closed form solution (take the derivative and set it to 0), and this precisely gives us our familiar GD update:

$$x^{t+1} = x^t - \eta \nabla f(x^t).$$

An example of this local minimization problem is shown in Figure 7.3, where the blue dot is our current iterate x^t and the red dot is x^{t+1}.

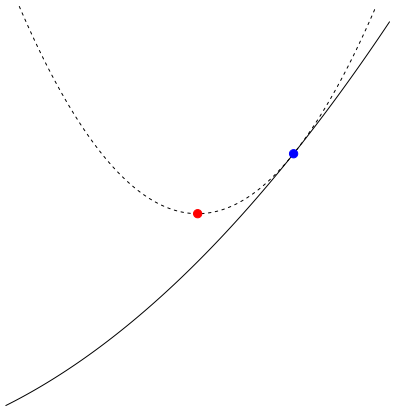

Figure 7.3: Local quadratic approximation of a function

7.2 Choosing the Step-Size

In practice, the most important choice to be made is that of the step-size. We'll see various theoretical rules/schedules that one might follow based on what we know about the objective function. Here are some natural possibilities:

1. **Fixed Step-Size:** Here we simply select a fixed step-size η and run the algorithm with that fixed step-size. An immediate problem that you will encounter (in practice) even for very benign problems is that if you select the step-size too large then GD can diverge, and if you select it too small it might take a very long time to converge.

   You will find pictures of this in the BV textbook, but here is a typical analytical example to keep in mind.

   (a) Suppose we have $f(x) = x^2/2$, initialize at $x^0 = 1$ we take our step size to be 3 (too large). Then the iterates will be $x^t = -2, 4, -8, \ldots$ (i.e. GD will diverge).

   (b) For the same function, initialization, if we take our step size to be 0.00001 then GD would take 10^5 steps to converge.

   (c) On the other hand if we picked the "correct" step-size of 1, we would converge in 1 step.

Figure 7.4 depicts three runs of gradient descent, each with a different fixed step size η. When the step size is too small, even after 100 steps it has not reached the minimum. When the step size is too large, after 8 steps it has already begun to diverge. When the step size is "just right", it converges after 40 steps.

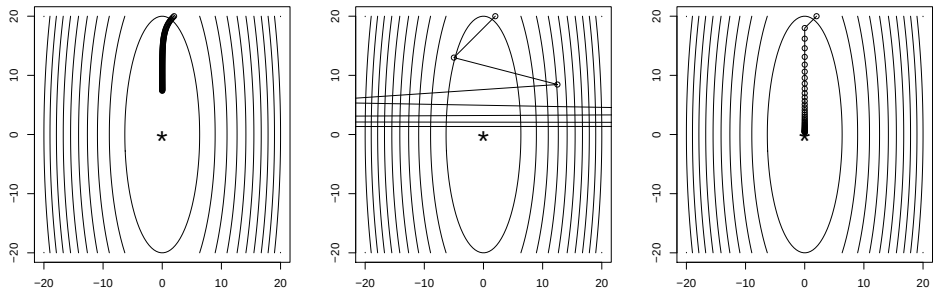

Figure 7.4: Examples of different steps sizes on the function $f(x) = (10x_1^2 + x_2^2)/2$. One is too small (left), one too large (middle), and one "just right" (right).

In theory, we'd like to understand this issue better (i.e. what properties of a function make certain step-sizes "too big", "too small", or "correct"). The correct step-size in many cases may depend on properties of the function that we don't know. In practice, it will often be useful to have at our disposal a few different ways to tune the step-size (and some understanding of how we might diagnose issues with the step-size choice).

2. **Exact Line-Search:** Once we've committed to a direction (in GD this is the direction of the negative gradient), one might consider solving the following 1D optimization problem to determine the best step-size:

$$\eta^t = \arg\min_{\widetilde{\eta} \geq 0} f(x^t - \widetilde{\eta} \nabla f(x^t)).$$

It's often computationally cumbersome to solve this optimization problem exactly, so we resort to some approximation of this idea.

3. **Backtracking Line-Search:** The idea of backtracking line-search very roughly, is to try an aggressive (large) step-size, and reduce it by some factor if it's too big.

Here is the algorithm: we pick two parameters $\alpha \in (0, 0.5)$ and $\beta \in (0, 1)$. At iteration t: initialize $\eta = 1$,

(a) If $f(x^t - \eta \nabla f(x^t)) > f(x^t) - \alpha\eta\|\nabla f(x^t)\|_2^2$, then reduce $\eta := \beta \times \eta$ and go back to step (a).

(b) Otherwise, take a step, i.e. set $x^{t+1} = x^t - \eta \nabla f(x^t)$.

Often in practice, taking $\alpha = 0.3$ and $\beta = 0.5$ work reasonably well. This method of backtracking line search uses the Armijo-Goldstein inequality to ensure that we acheive a sufficient decrease.

In Figure 7.5, gradient descent with backtracking line search is applied to the same function we examined before and it roughly seems to get the right step sizes.

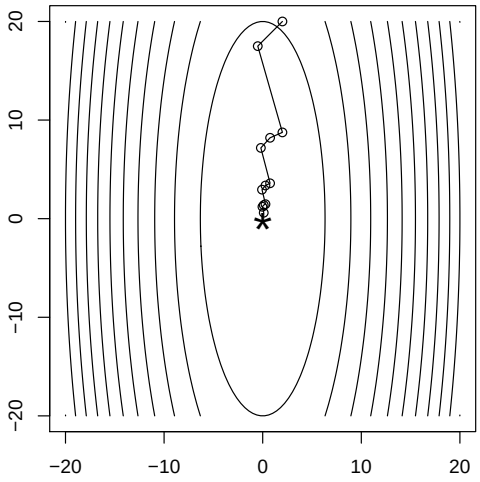

Figure 7.5: Example of gradient descent with backtracking line search. In this example, it accepts 12 steps and computes 40 steps total.

You will develop some better intuition when we study the main descent lemma for GD, but roughly if your function is nice (the Hessian term in a Taylor series is ignorable), you should expect to make about $\eta\|\nabla f(x^t)\|_2^2$ amount of progress in one step of GD if η is small enough. The backtracking line search simply says if you're making upto an α factor of this amount of progress you should be content and take a step.

Example 7.1 (Backtracking on a Linear Function)**.** As an example, consider the case where f is a linear function. Then it's clear that taking a gradient step $\eta \nabla f(x)$ should improve the function by *exactly* $f(x^{t+1}) - f(x^t) = \eta \|\nabla f(x^t)\|_2^2$. If our function is locally linear than maybe making an improvement of $\alpha\eta \|\nabla f(x^t)\|_2^2$ is good enough.

Segue... Next lecture we will look at evaluate the performance of gradient descent on two example functions and consider our first convergence results.