Gradient descent Overview

Today we'll be reviewing gradient descent and getting some intuition behind what it does, why it works, what are some caveats in its implementation, and how we can improve the basic algorithm.

In general, gradient descent seeks to solve the following problem. Suppose we have a function,$f(\mathbf{x})$ over a vector of inputs (or, as will often be referred to also as parameters), $\mathbf{x} = [x_0, x_1..., x_m]$. The goal of gradient descent is to find the setting of this vector that minimizes the function, that is

$$\mathbf{x}^* = \arg\min_x f(\mathbf{x}) \tag{1}$$

The general insight behind gradient descent is that if one always travels in the direction of fastest descent, that is, changes $\mathbf{x}$ in a way that is guaranteed to reduce value of $f(\cdot)$, one will always converge to a local minimum. There is the caveat that $f(\cdot)$ must satisfy certain properties. For this course we will impose the overly strict conditions that $f(\cdot)$ is infinitely differentiable, which implies that it is also smooth and continuous.

Gradient descent as a specific approach to continuous optimization is omnipresent, particularly within the realm of machine learning. Many of the optimization techniques that allow complicated neural networks to detect unique faces or beat Go grandmasters are surprisingly simple refinements to the basic idea of gradient descent, so it behooves those of you interested in machine learning to learn it well now!

The basic algorithm is exceedingly simple. It looks like this:

```
GRADIENT_DESCENT ( $f(\cdot)$ ):
    initialize $\mathbf{x}$ arbitrarily
    while $|\nabla f(x)| > \epsilon$:
        $x_{old} = x$
        $x = x_{old} - \eta \nabla f(x_{old})$
    return $x$
```

Three new symbols have arisen above. The first, ϵ, gives us our stopping condition. If we find that our gradient has essentially leveled off, that is less than ϵ, we say that the θ we found is close enough to optimal and return that. The second symbol is η, the learning rate. There's a lot of subtlety in choosing the right learning rate which will be explored shortly. Last is $\nabla f(\mathbf{x})$, the gradient of the function at the current parameter setting. Recall that the gradient is

$$\nabla f(\mathbf{x}) = \left[\frac{\partial f(\mathbf{x})}{\partial x_0}, \frac{\partial f(\mathbf{x})}{\partial x_1} ..., \frac{\partial f(\mathbf{x})}{\partial x_m}\right] \tag{2}$$

In one dimension, it reduces to just the derivative. Anyhow, with those clarifications, that's it! Plug in some arbitrary values for the constants and if you're lucky it will converge sometime within your lifespan.

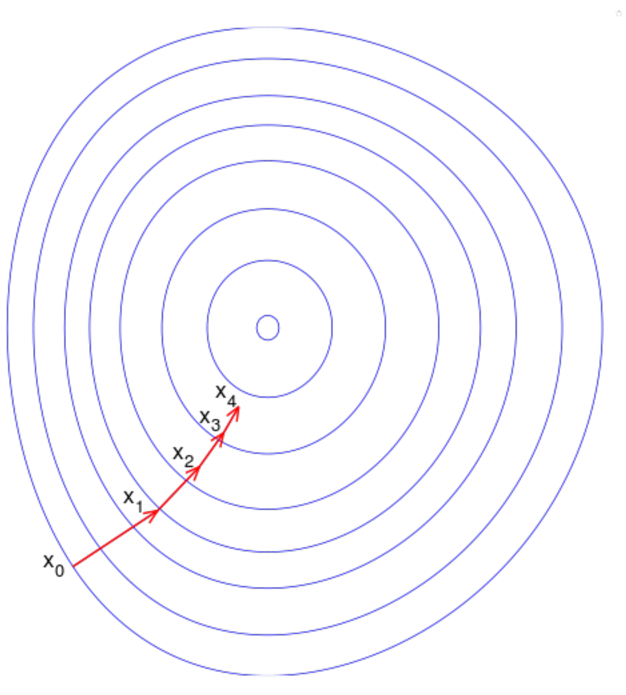

Figure 1: A successful run of gradient descent for a function of two parameters, $f(x, y)$ shown from a bird's eye view. The arrows represent steps of the algorithms where the oblong shapes are contour lines.

Gradient Refresher

Before launching ahead, its helpful to remind ourselves what the gradient is outside of its definition in equation 2. In dimensions greater than 1, it's a vector that points in the direction of maximum change. We can confirm this by recalling the directional derivative $u \cdot \nabla f(\theta)$ where u is a unit vector tangent to the function f. The directional derivative tells us the infinitesimal change in the value of the function given the direction in which we toggle our input values, u. Thus, we can immediately see that this dot product's value is maximized when u is parallel to $\nabla f(\theta)$ and minimized when anti parallel. Another way to think about them is that they are perpendicular to contour lines as shown in figure 1.

Convexity, convergence, and fun!

A class of functions that arises quite often in machine learning are convex functions. Better yet, gradient descent performs quite nicely on such functions! There are two equivalent definitions of a convex function. Restricting ourselves to one dimension for clarity, a function $f(\cdot)$ is convex if for any λ, $0 \leq \lambda \leq 1$

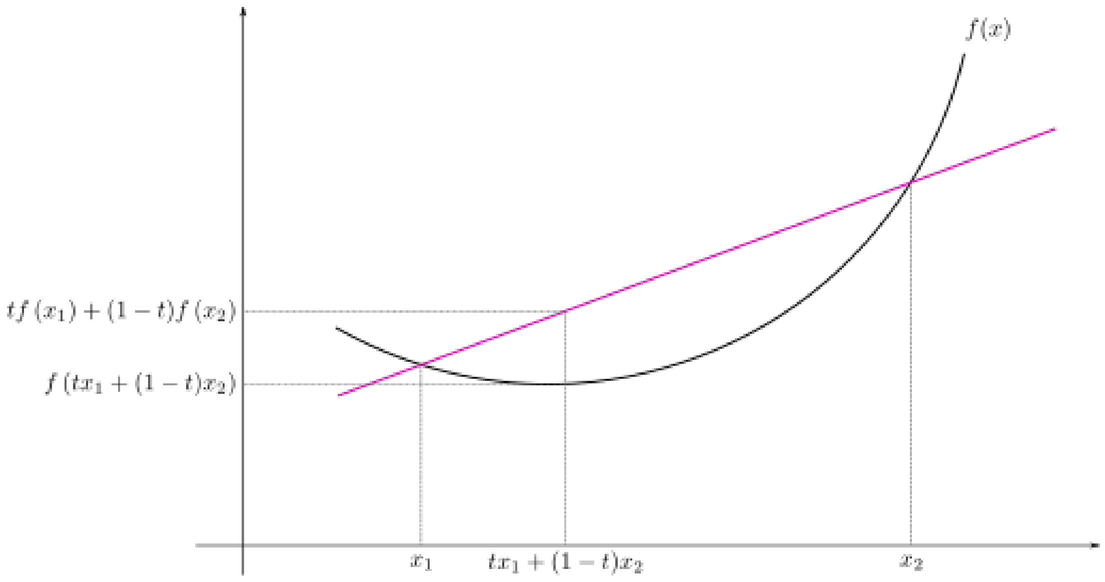

Figure 2: The red line depicts all possible values for the left side of equation 3 as we adjust λ. For a particular value of lambda, the equation describes the relationship between a point on the red line (the left hand side of the equation) and the point directly below (the right hand side).

$$\lambda f(x) + (1-\lambda)f(y)) \geq f(\lambda x + (1-\lambda)y) \tag{3}$$

Where x and y are in this case just real numbers (though this definition extends directly to vectors as well). This is more naturally conveyed in figure 2.

Another definition for a convex function, one that you may remember from 18.01, is a function such that for all x

$$\frac{d^2 f(x)}{dx^2} \geq 0. \tag{4}$$

Convex functions have some useful properties. For one, if there's a minimum, it's a global minimum and its derivative is zero. Furthermore, as we approach the minimum of a convex function, the gradient becomes more shallow. If we were to choose a fixed learning rate η, then the algorithm might move quickly to the vicinity of the minimum but then slow down as we approach it. This is also somewhat demonstrated in figure 1.

Supplementary (optional) section: Why are local optimum global optimums for Convex functions

One of the greatest strengths of convex functions for modeling problems is that once we arrive at any minimum, we know we are at the best minimum. This is a great property to have because it means that once we know we have reached a minimum, we do not have to keep searching. This can save us a lot of time! intuitively, since the function is shaped like a bowl where points can only roll down the hill and since this happens at **all** points in the function, it must mean that wherever we are, we must fall to a minimum (which makes it a global minimum by definition. Notice that all global minimum are local minimums too by definition). In this section we will prove this important property of convex functions and show formally that the above intuition is correct.

First recall from 18.02 (or linear algebra or high school) that the size size of a vector x is defined as:

$$\|x\|^2 = \sum_{d=1}^{D} |x_d|^2$$

this specific size is often called the 2-norm (though there are many other types of norms. Look some of them up for fun!).

Theorem 1 ***Local minimum are global minimums***

If we have a function $f : R^D \to R$ that is convex then the local minimum is the global minimum.
Proof.

Define a ball of radius r centered around a vector u (i.e. the set of all the points at a distance r from some center u):

$$B(u,r) = \{v : \|v-u\| \leq r\}$$

We say that $f(u)$ is a local minimum of f at u if for some (non-zero) radius around u, $f(u)$ is smaller than all its neighboring points. Formally, $f(u)$ is a local minimum of f at u if there exists some $r > 0$ such that for all $v_B \in B(u,r)$ we have $f(v_B) \geq f(u)$. It follows that for any v (not necessarily in B), there is a small enough $\alpha > 0$ such that $u + \alpha(v-u) \in B(u,r)$ and therefore:

$$f(u) \leq f(u + \alpha(v-u))$$

If f is convex, we also have that:

$$f(u + \alpha(v-u)) = f(\alpha v + (1-\alpha)u) \leq \alpha f(v) + (1-\alpha) f(u)$$

Combining these two inequalities and rearranging terms we get:

$$f(u) \leq f(u + \alpha(v-u)) = f(\alpha v + (1-\alpha)u) \leq \alpha f(v) + (1-\alpha) f(u) = \alpha f(v) + f(u) - \alpha f(u)$$

writing down the first term and last term of the above inequality and proceeding to simplify it, we get the following implication:

$$f(u) \leq \alpha f(v) + f(u) - \alpha f(u) \implies 0 \leq f(v) - f(u)$$

which gives us the last crucial inequality:

$$f(u) \leq f(v)$$

which means that for any v that we choose, we have $f(u) \leq f(v)$ which is the definition of a global minimum. So even if we try to choose a vector outside the ball around the local minimum, all other vectors end up being above the local minimum.

Intuitively, the proof can be summarized (nearly in a proof by contradiction framework) as following: if you try to choose a vector v outside of the radius ball of the local minimum, because of the bowl (convex) shape of our curve, it has to be above that local minimum and since this is true for all points (because of the shape of our function) then the local minimum must be a global minimum!

As an exercise there is another proof you can formulate to show local minimums are global minimums for convex functions. Start the proof just as above with u as a local minimum and with some other vector v (far away) from u such that $v \notin B(u, r)$. Then try to assume (by contradiction) that that point resulted in a value of f that was bellow the local minimum (this is an extreme case but it conveys the idea). Can you conclude using the definition of convexity, that this is not possible? Why? There are some details to straighten out the precise proof but the intuition should be clear after drawing a picture of what is going on.

The shortcomings of a single learning rate

A single learning rate can lead to different kinds of complications. Too large, and you can diverge, slingshotting back and forth across the minimum. Too small and you can take a really long time to converge. Just right and you get fast and convergent behavior. Unfortunately, it's hard to tell what's going to happen a priori and sometimes there are situations where there is no single best learning rate for the entire space: a rate that may lead to fast convergence from one direction may lead to vast overshooting in another. This behavior is summarized in figure 3.

Taylor Series and Adapting the Learning Rate

As we just saw, picking a fixed learning rate can be suboptimal at times, so there may be a way for us to adapt the learning rate based on where we are in the function. A tidbit from calculus that's useful to recall when thinking about this issue is the Taylor series expansions. We'll use the single-dimension version to offer intuition about how to choose η and then we'll extend this to multiple variables. Given a starting value x, a offset, ϵ, and a function $f(x)$, we can get the exact value of the function ϵ away as

$$f(x+\epsilon) = f(x) + \epsilon f'(x) + \frac{\epsilon^2}{2!} f''(x) + \frac{\epsilon^3}{3!} f'''(x) ... \tag{5}$$

When the steps ϵ become sufficiently small, all terms except the first two shrink away and a linear approximation using just the derivative turns out to be very close to the actual function value. In gradient descent, all we really are doing is making a linear approximation at each step, traveling some distance along it, then revising our linear approximation when the one we just used no longer approximates the true function well. Well, that prompts the question of how the linear

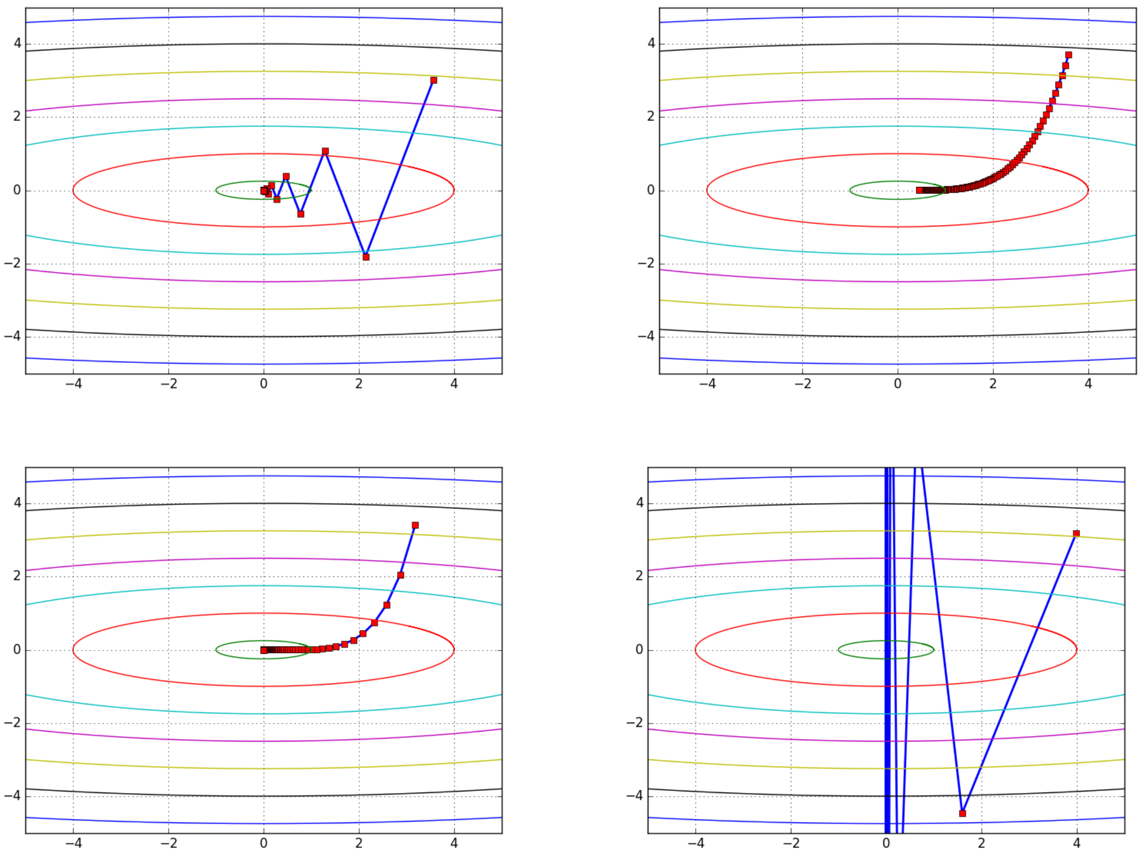

Figure 3: Different kinds of behavior for gradient descent with various learning rates. The top left has a rate that's a bit too large and overshoots. The top right has one that's too small and takes many iterations to get close to the minimum. The bottom left is well tuned, and the bottom right is much too large, leading to divergent behavior.

approximation differs from the actual function. If we take ϵ to still be small but large enough to make the third term of the above relevant, we can approximate our function more closely as the first three terms of the expansion. In other words, we are now making a quadratic local approximation to the function. We have

$$f(x+\epsilon) \approx f(x) + \epsilon f'(x) + \frac{\epsilon^2}{2!} f''(x) \tag{6}$$

Now, with this approximation, let's see which choice of ϵ leads to a minimum of the local parabola! We take the derivative with respect to ϵ and set to 0 to obtain

$$\begin{align} 0 &= f'(x) + \epsilon f''(x) \tag{7} \\ \epsilon &= -\frac{f'(x)}{f''(x)} \tag{8} \\ & \tag{9} \end{align}$$

This should look familiar! Recall that in gradient descent we choose our ϵ to be $-\eta f'(x)$, and so if we pick our η to be $\frac{1}{f''(x)}$.

We can extend the Taylor series expansion to multiple dimensions for a vector $\mathbf{x}$ and vector perturbation ϵ and redo the analysis. A truncated multidimensional Taylor series looks like

$$f(\mathbf{x}+\epsilon) = f(\mathbf{x}) + \nabla f(\mathbf{x})^\top \epsilon + \epsilon^\top H(f(\mathbf{x}))\epsilon \tag{10}$$

Where $H(f(\theta))$ is the Hessian and looks like

$$\begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_1 \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_m} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_m \partial x_1} & \frac{\partial^2 f}{\partial x_m \partial x_2} & \frac{\partial^2 f}{\partial x_m \partial x_3} & \cdots & \frac{\partial^2 f}{\partial x_m \partial x_m} \end{bmatrix}$$

and our η becomes $H(f(\mathbf{x}))^{-1}$, the inverse of the Hessian. In real settings, the Hessian or approximations to the Hessian are often used to adjust the learning parameter so we can descend quickly without fear over overshooting.

Campaign Madness

Ben Bitdiddle, starting from his humble beginnings at MIT, is now running for president of the United States under the "Fewer Psets Party". In order to most effectively campaign, Ben wants to identify swing voters, that is voters who have a decent chance of voting for Ben given some more directed campaigning. To this end, Ben has examined polling data very closely, trying to figure out how factors such as age or income influence how positively voters view him. Using this data, he wants to come up with a prediction how much a previously-unseen voter may approve of Ben.

Explicitly, Ben is given a $n \times m$ matrix, θ which contains m different pieces of demographic information about n individuals. For instance, the first row of θ will consist of m different non-negative real numbers that correspond to some quantitative information (like age, income, number of children, years of formal education, hand size, etc.) about a particular person. For instance, if Ben polled three voters about just their age and handsize in centimeters, the matrix might look like this:

$$\begin{bmatrix} 21 & 20 \\ 44 & 15 \\ 64 & 17 \end{bmatrix}$$

Ben also has a length n vector $\mathbf{y}$ of real numbers, the approval scores, which represent how much a person approves of Ben. The more positive the score, the more a person approves of Ben. For instance, the i^{th} element of $\mathbf{y}$ corresponds to the approval score of the i^{th} person.

In order to predict the approval score of a new voter, Ben wants to construct a model that generates a guess, $\hat{y}$ based on the demographic information of that voter. He does this for the i^{th} voter as

$$\hat{y}(\theta_{\mathbf{i}}) = \sum_{j=1}^{m} \theta_{ij} x_j = \theta_{\mathbf{i}} \cdot \mathbf{x} \tag{11}$$

Where $\theta_{\mathbf{i}}$ denotes the i^{th} row of θ and θ_{ij} the element in the j^{th} column of the i^{th} row. In other words, we compute the approval score as the inner product between a voter's attributes and some vector of parameters $\mathbf{x}$. Note that for our input we will set all $\theta_{i1} = 1$, so that the x_1 term functions as a constant offset for our model.

Ben wants to build the best model possible and wants to do so by picking the vector of parameters $\mathbf{x}$ that minimize the mean-squared error, $J(\mathbf{x})$, that is he wants to find a $\mathbf{x}^*$ that minimizes

$$J(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \theta_{\mathbf{i}} \cdot \mathbf{x})^2 = \frac{1}{n} (\mathbf{y} - \theta\mathbf{x})^\top (\mathbf{y} - \theta\mathbf{x}) \tag{12}$$

Once Ben finds the optimal $\mathbf{x}$ for this cost function, he will generate guesses of the approval score of brand new voters and in turn decide whether or not to devote resources campaigning for those people. You won't have to make the decision of where Ben has to campaign, but he needs your help to build this model!

In order to perform the gradient descent algorithm, we first need an update equation. Analytically derive the update step for the gradient descent algorithm. You may leave the step size as η for now.

Solution: This is a straightforward matter of taking the gradient and plugging it into the general update for gradient descent. In particular, we have

$$\mathbf{x}^{t+1} = \mathbf{x}^t - \eta \cdot \nabla J(\mathbf{x}^t) \tag{13}$$

Where the j^{th} component of the gradient is

$$\nabla J(\mathbf{x}^t)_j = \frac{-2}{n}\sum_{i=1}^{n}(y_i - \theta_i \mathbf{x})\theta_{\mathbf{ij}} \tag{14}$$

Fitting a Plane to Data

Imagine we have a set of n points of the form $\theta^i = [\theta_1^i, \theta_2^i]$ in addition to values y^i corresponding to those points. To make this situation concrete, let θ_1^i be the latitude of a location represented by the ith point, θ_2^i be its distance to an ocean in kilometers, and y^i be the mean temperature during the month of May. Suppose we want to fit a planar model to this data in an attempt to predict the mean temperature in May of some other locations. A plane can be specified with three parameters, $[x_0, x_1, x_2]$ and can yield a prediction about a location as

$$\hat{y} = x_0 + x_1\theta_1 + x_2\theta_2 \tag{15}$$

Still, we want this planar model to accurately represent the source data. We can specify what it means to have a good fit/represent the data in a variety of ways, but a common metric one can use is the mean-squared error, MSE, of the data.

$$J(x) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (x_0 + x_1\theta_1^i + x_2\theta_2^i))^2 = \tag{16}$$

Notice that we can think of the MSE as a function of our parameters and in turn the best fit will be the one that leads to the minimum MSE. Looks like we have a continuous optimization problem on our hands and can use gradient descent! Let's first find the gradient with respect to the MSE, $J(x)$:

$$\frac{\partial J(x)}{\partial x_0} = \frac{1}{n}\sum_{i=1}^{n} 2(y_i - (x_0 + x_1\theta_1^i + x_2\theta_2^i)) \tag{17}$$

$$\frac{\partial J(x)}{\partial x_1} = \frac{1}{n}\sum_{i=1}^{n} 2(y_i - (x_0 + x_1\theta_1^i + x_2\theta_2^i))\theta_1 \tag{18}$$

$$\frac{\partial J(x)}{\partial x_2} = \frac{1}{n}\sum_{i=1}^{n} 2(y_i - (x_0 + x_1\theta_1^i + x_2\theta_2^i))\theta_2 \tag{19}$$

Now that we have all the components of our gradient, we can just plug it into our algorithm from the very beginning and proceed with some fixed learning rate. If we want to be particularly clever, we could compute the Hessian and invert it to find a optimal learning rate, but we leave that as an exercise to the student.