Design and Analysis of Algorithms
Massachusetts Institute of Technology 6.046J/18.410J
Debayan Gupta, Aleksander Madry, and Bruce Tidor Handout 10

Recitation 10: Continuous optimization

1 Gradient Descent

As you saw in the lecture, the *gradient descent* algorithm is an optimization algorithm via greedy local search. This algorithm is relatively simple and scales well to large data sets. Therefore, it has lots of application in machine learning and other fields.

In general, the problem we are trying to solve is the following: Assume we have a function $f : \mathbb{R}^n \to \mathbb{R}$. We are trying to find x^* such that $f(x^*)$ is the minimum of f.

2 Gradient Descent Algorithm

GRADIENTDESCENT(f) returns approximation of the minimum of f.

GRADIENTDESCENT(f, x_0)

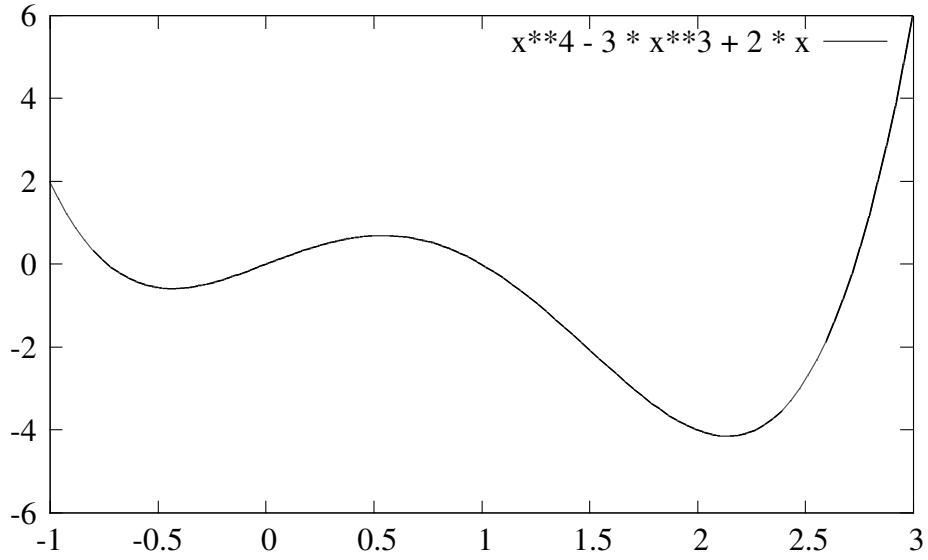
1: $x^{(0)} = x_0$
2: **for** i in $1 \dots T$ **do**
3: $\quad x^{(i)} = x^{(i-1)} - \eta \cdot \nabla f(x^{(i-1)})$
4: **end for**

Intuitively, in each iteration, the algorithm goes a little bit downhill, so eventually it should reach the bottom of the hill. "Downhill" is always in the direction of $-\nabla f$. When we can't go downhill any farther, $\nabla f = 0$, and we've found a (local) minimum.

As a reminder of what the gradient is, $\nabla f(x)_i = \partial f(x)/\partial x_i$. This is simply the multi-dimensional analouge of the derivative.

We'd like to analyze when this algorithm works well, and well it doesn't work well. We're going to do this with four example functions, each with an input in one dimension and each with the parameters $x_0 = 0$ and $\eta = 1$. As we go, we'll put stronger and stronger conditions on our function, until we can guarantee that the algorithm works well. This will tell us what properties the function needs to have for this algorithm to work well. On other functions, we'd need a different algorithms.

2.1 Convexity



The first way that this algorithm can go wrong is that it converge, but not to the point we want it to converge to. For instance, on the function above, the gradient descent algorithm with the starting point $x_0 = 0$ will converge to a local minimum near $x = -0.5$, not the desired global minimum near $x = 2$.

This happens because the only thing that gradient descent knows how to do is go downhill, so it goes down the hill to the left, instead of to the point we want it to go to.

To avoid this sort of problem, we're only going to run gradient descent on functions that don't have local minima which aren't their global minimum. Unfortunately, there isn't a good way to tell whether this is the case for a given function without doing time-consuming computation. Therefore, we're going to require something stronger, namely that the function is convex.

There are two definitions of convex that we will think about. The standard definition, which is often the easiest to prove, is that f is convex if

$$\forall x, y, 0 \leq \lambda \leq 1, f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(x)$$

This intuitively says that f is convex if the value of the function at a point between two other points must be less than the weighted average of the function values at those points. The function must always lie below a line drawn between any two points on the function. The function given above is not convex, and that's why gradient descent fails on it.

In analyzing gradient descent, we're going to be making use of the Taylor series expansion of f:

$$f(x + \delta) = f(x) + \nabla f(x)^T \delta + \delta^T \nabla^2 f(x) \delta \ldots$$

From this, we can focus on the linear approximation of $f(x)$:

$$f(x+\delta) \approx f(x) + \nabla f(x)^T \delta$$

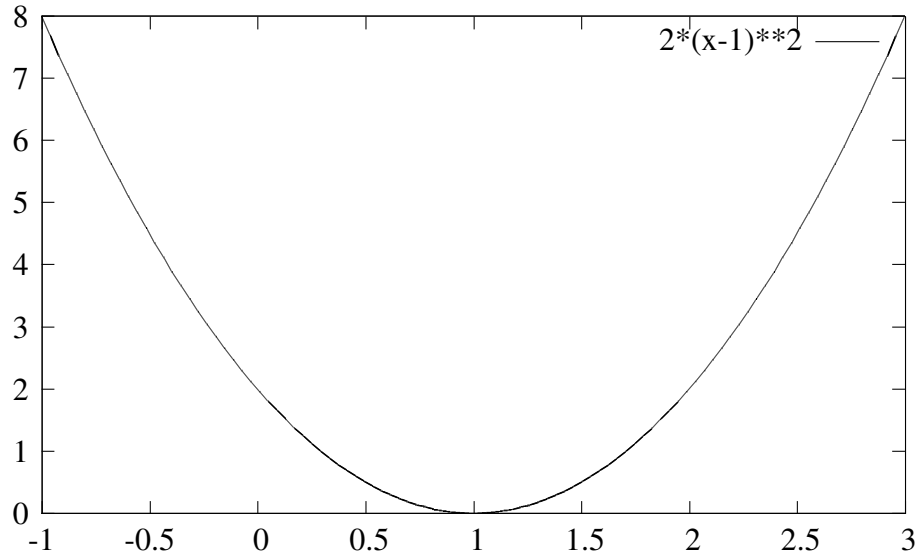
A function is convex if this linear approximation is always a lower bound on the function:

$$\forall x, \delta, f(x+\delta) \leq f(x) + \nabla f(x)^T \delta$$

This means that the function lies above the line tangent to it at any point, in any direction.

If a function is convex, gradient descent will only converge to the global minimum, if it converges at all.

2.2 Smoothness (Doesn't bend up too fast)



Let's take a look at what happens when we run gradient descent on this function, again with the parameters $x_0 = 0, \eta = 1$. $\nabla f(x) = \frac{df(x)}{dx} = 4 * (x-1)$.

i	x_i	$\nabla f(x_i)$
0	0	-4
1	4	12
2	-8	-36
3	28	108
4	-90	-364

As you can see, the algorithm is not converging at all, and is in fact getting farther and farther away from the minimum, which is $x^* = -1$.

The reason the algorithm fails in this case is that it keeps overshooting the minimum. This happens because the steps that we are taking are too big. We can't solve this problem with just a linear approximation, so we'll think about a quadratic approximation instead.

Based on our Taylor series from before, the quadratic approximation is

$$f(x+\delta) = f(x) + \nabla f(x)^T \delta + \delta^T \nabla^2 f(x) \delta$$

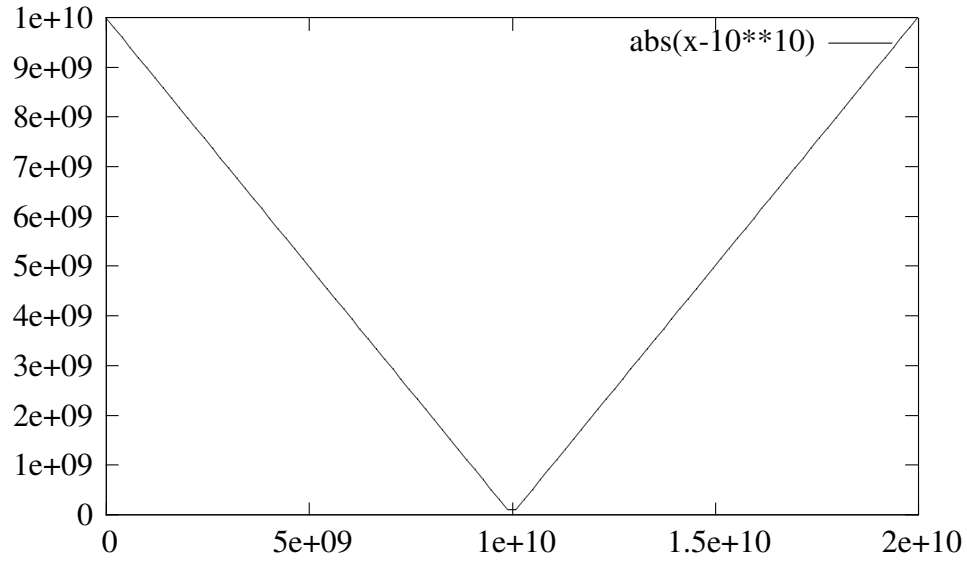
This time, we're ignoring all terms larger than the quadratic one. If this quadratic approximation was exactly correct, and we were working in one dimension, then the perfect choice of η would be $1/(\nabla^2 f(x)$. For instance, in the example above, $\frac{d^2 f(x)}{dx^2} = 4$, and the value of η that would bring us exactly to the minimum is $1/4$.

Of course, not all functions that we consider will be exactly quadratic, and we never want to run into this divergent behavior. So we'll find an upper bound on how quickly the function can slope upwards. The upper bound will look like the following:

$\delta^T \nabla^2 f(x) \delta \leq \beta ||\delta||^2$.

This says that at any starting point, and along any direction, the function is always upper bounded by a quadratic function, with leading coefficient no more than $\beta/2$. This condition is called β-smoothness, and with the convexity guarantee, will ensure that our algorithm always converges to an absolute minimum, given that one exists.

2.3 Strongly-Convex (Not a line)



We've guaranteed that the algorithm will converge, but we'd like it to do so quickly. In particular, we'd like it take a polynomial amount of time in the size of the input. Since our minimum can be as big as a number written in the input, and it takes as many bits to write a number as the logarithm of the size of number, this means that our algorithm needs to run in an amount of time logarithmic in the size of the minimum.

In the example, that is not the case. Let's take a look at what happens when we run gradient descent on this function, again with the parameters $x_0 = 0, \eta = 1$. $\nabla f(x) = \frac{df(x)}{dx} = -1$. We're not going to look at what happens near the minimum itself, assume the function changes a bit there to remain smooth.

i	x_i	$\nabla f(x_i)$
0	0	-1
1	1	-1
2	2	-1
3	3	-1
4	4	-1

As you can see, this algorithm will always take a step of size 1, and so will take 10^{10} steps to reach the minimum. This is not acceptable, this is another type of function we shouldn't be running this algorithm on.

While in this specific case, we can increase the step size to converge faster, in general we might not be able to do that because increasing the step size might run afoul of the previous β-smoothness rule. In particular, a function might look like a steep quadratic in one dimension, and like a line in another dimension, and so it will just take forever.

To get around this problem, we're going to fall back on the situation we know how to solve really well, the quadratic case. In the quadratic case, a single step of size $1/\nabla^2 f(x)$ got us exactly to the minimum. In general, things won't be that good, but a step of that size will allow us to make significant progress, and in general we'll only need to make a logarithmic number of steps of that size.

If we can put a lower bound on the quadratic-ness of the function, we'll be able to say how big the steps we need to take a logarithmic number of are. The bound looks like the following:

$\delta^T \nabla^2 f(x) \delta \geq \alpha ||\delta||^2$.

This is called being α-strongly convex when $\alpha > 0$. If $\alpha = 0$, this is simply the convexity constraint, which is enough to prove convergence, but not logarithmic converge. With $\alpha > 0$, we can say that after a logarithmic number of steps of size $1/\alpha$, we'll reach the minimum. However, we're not taking steps of size $1/\alpha$, we're taking steps of size $1/\beta$. The ratio of these two step sizes is β/α, and β/α times more steps will be enough to compensate for the fact that our steps are α/β times smaller. Since β/α comes up so often, we given a name, the condition number, and a symbol, κ.

Now we can give the actual effectiveness guarantee of the function:

If we set T, the number of steps we're going to take, to $\kappa \log \frac{f(x_0)-f(x^*)}{\epsilon}$, then x_T, the output, will have a value within ϵ of minimal. $f(x_T) - t(x^*) \leq \epsilon$.

3 Guarantees, visualized