Gradient descent

Google Classroom

Gradient descent is a general-purpose algorithm that numerically finds minima of multivariable functions.

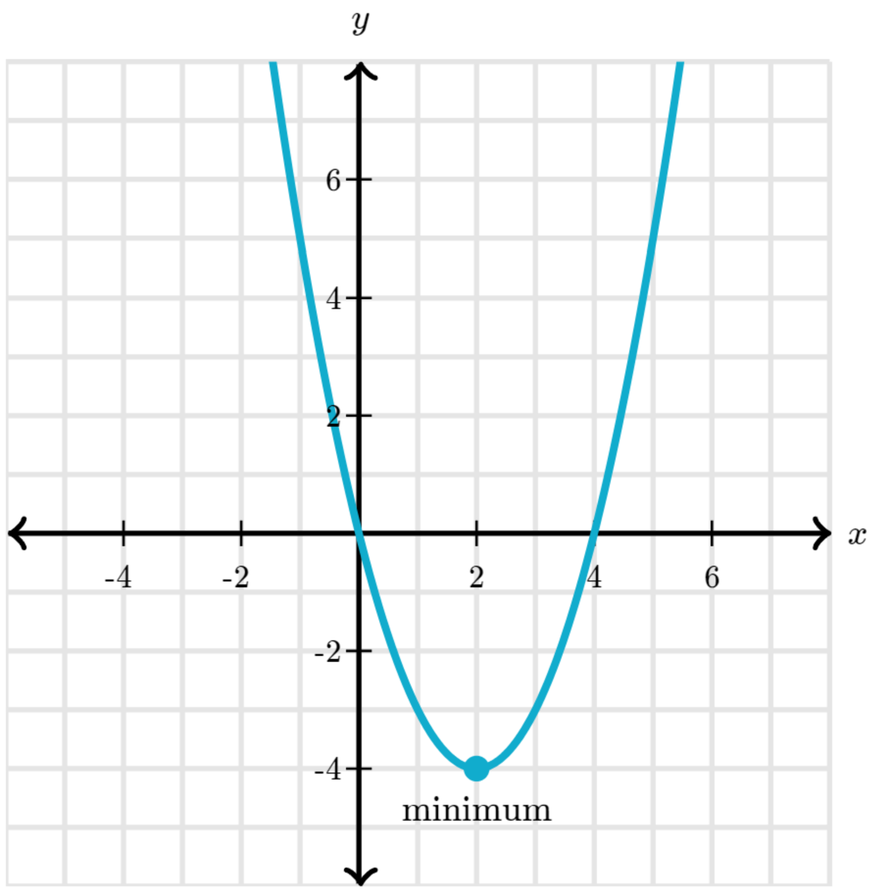
Background

- Gradient
- Maxima and minima

So what is it?

Gradient descent is an algorithm that numerically estimates where a function outputs its lowest values. That means it finds local minima, but not by setting $\nabla f = 0$ like we've seen before. Instead of finding minima by manipulating symbols, gradient descent approximates the solution with numbers. Furthermore, all it needs in order to run is a function's numerical output, no formula required.

This distinction is worth emphasizing because it's what makes gradient descent useful. If we had a simple formula like $f(x) = x^2 - 4x$, then we could easily solve $\nabla f = 0$ to find that $x = 2$ minimizes $f(x)$. Or we could use gradient descent to get a numerical approximation, something like $x \approx 1.99999967$. Both strategies arrive at the same answer.

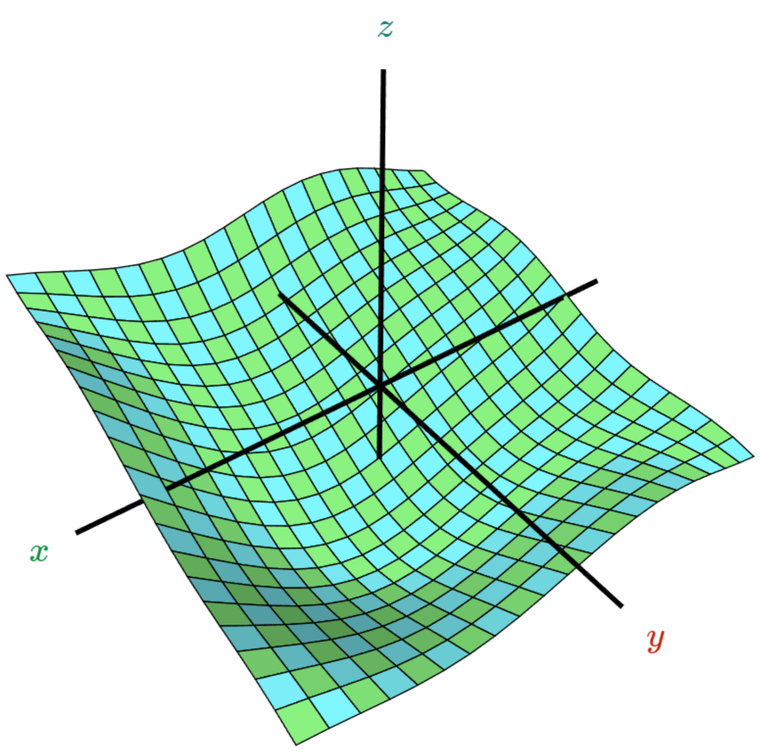


But if we don't have a simple formula for our function, then it becomes hard or impossible to solve $\nabla f = 0$. If our function has a hundred or a million variables, then manipulating symbols isn't feasible. That's when an approximate solution is valuable, and gradient descent can give us these

estimates no matter how elaborate our function is.

How gradient descent works

The way gradient descent manages to find the minima of functions is easiest to imagine in three dimensions.



Think of a function $f(x, y)$ that defines some hilly terrain when graphed as a height map. We learned that the gradient evaluated at any point represents the direction of steepest ascent up this hilly terrain. That might spark an idea for how we could *maximize* the function: start at a random input, and as many times as we can, take a small step in the direction of the gradient to move uphill. In other words, walk up the hill.

To *minimize* the function, we can instead follow the negative of the gradient, and thus go in the direction of steepest descent. This is gradient descent. Formally, if we start at a point x_0 and move a positive distance α in the direction of the negative gradient, then our new and improved x_1 will look like this:

$$x_1 = x_0 - \alpha \nabla f(x_0)$$

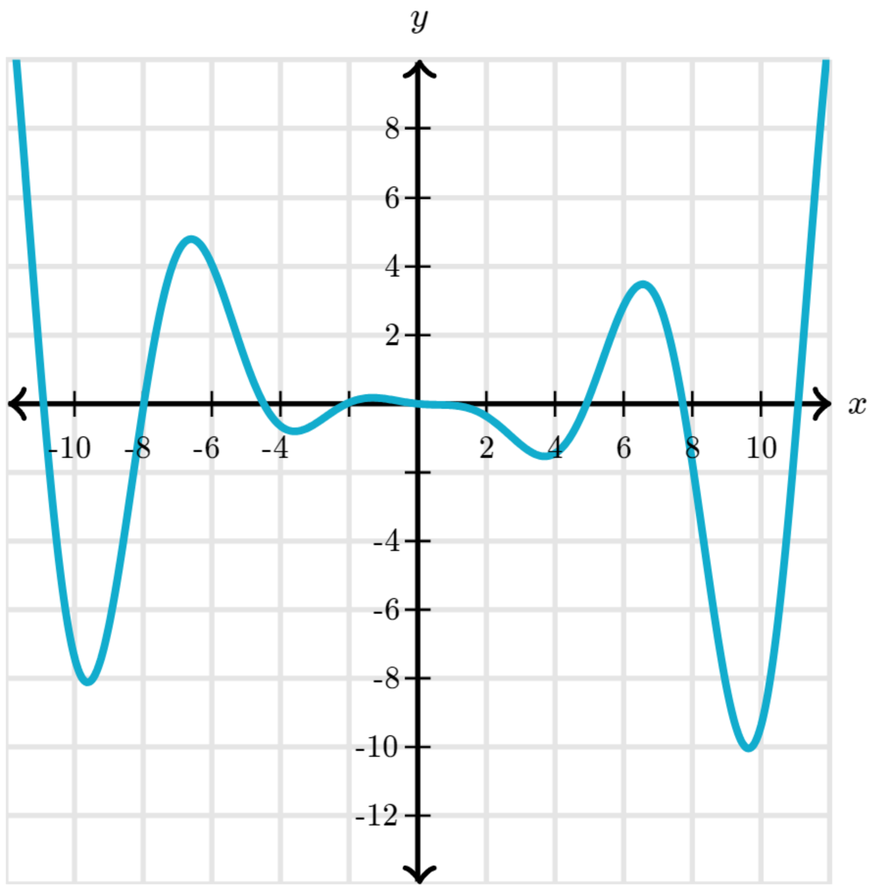
More generally, we can write a formula for turning x_n into x_{n+1}:

$$x_{n+1} = x_n - \alpha \nabla f(x_n)$$

Starting from an initial guess x_0, we keep improving little by little until we find a local minimum. This process may take thousands of iterations, so we typically implement gradient descent with a computer.

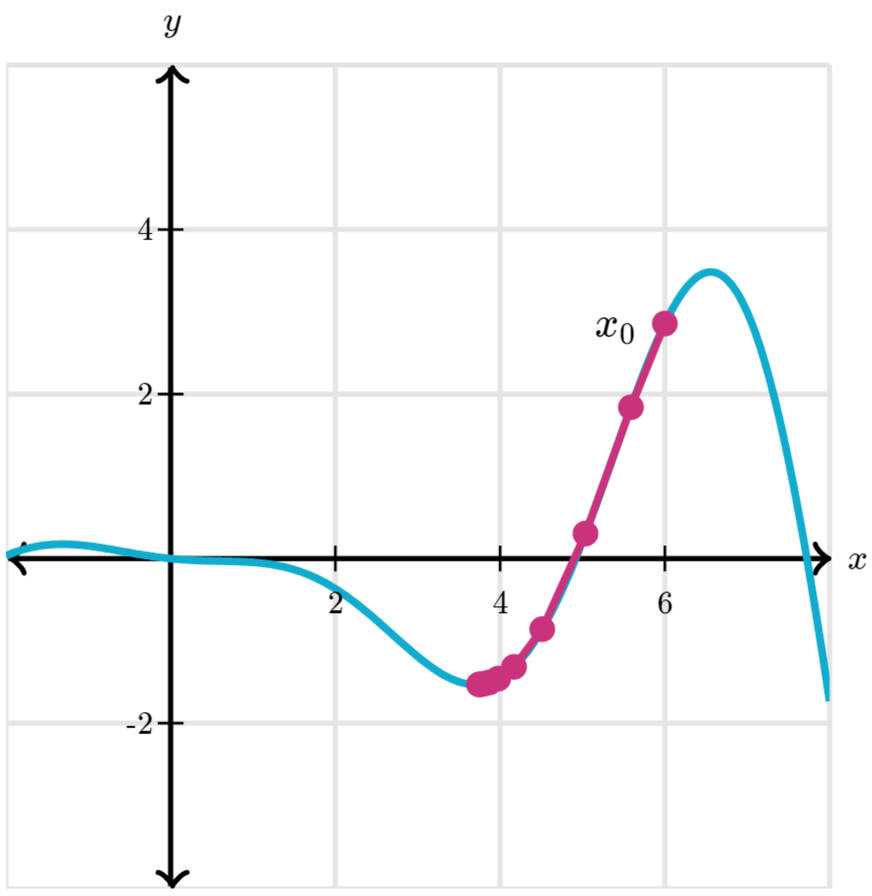
Example 1

Consider the function $f(x) = \dfrac{x^2 \cos(x) - x}{10}$.

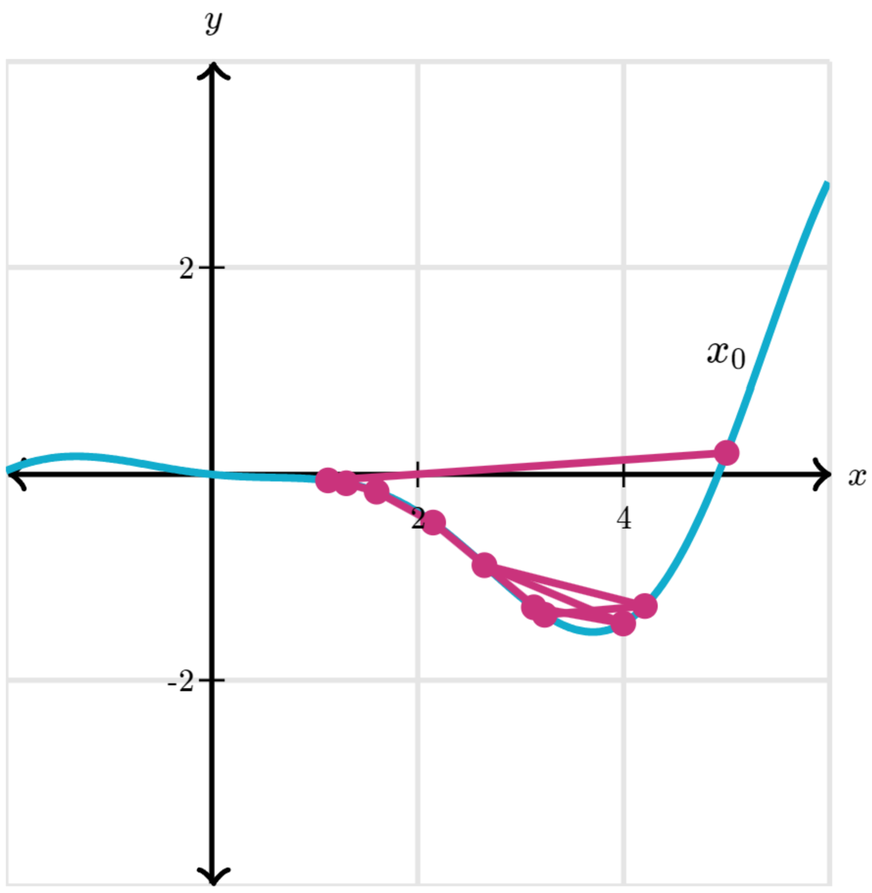


As we can see from the graph, this function has many local minima. Gradient descent will find different ones depending on our initial guess and our step size.

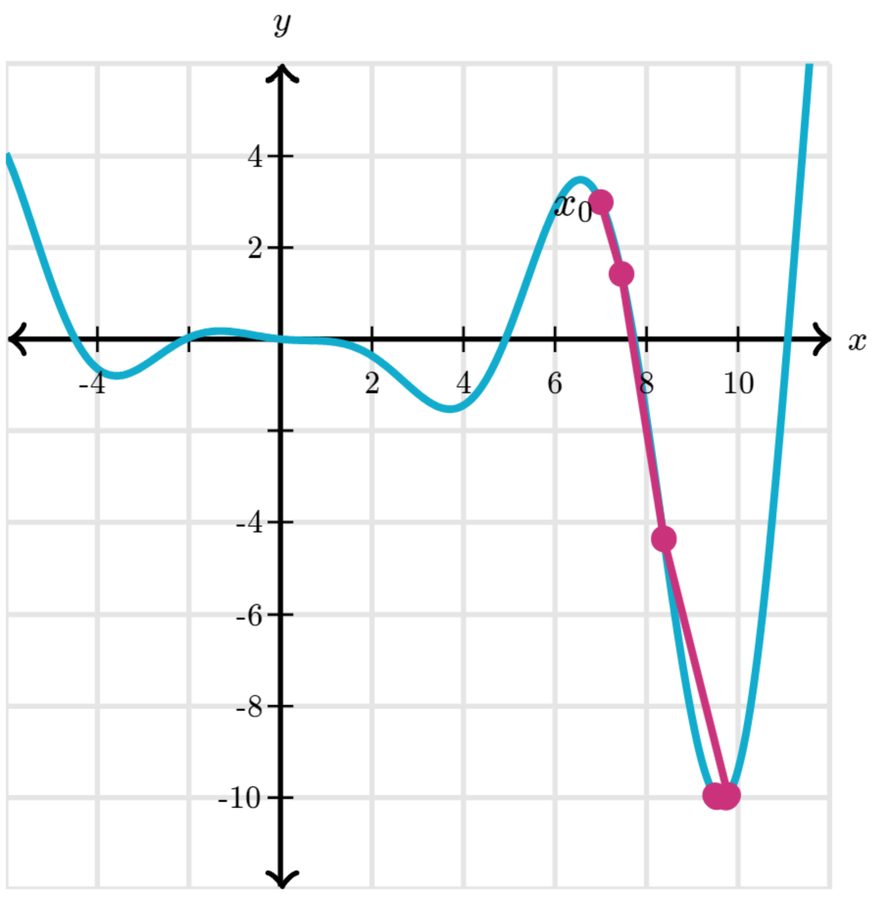
If we choose $x_0 = 6$ and $\alpha = 0.2$, for example, gradient descent moves as shown in the graph below. The first point is x_0, and lines connect each point to the next in the sequence. After only 10 steps, we have converged to the minimum near $x = 4$.



If we use the same x_0 but $\alpha = 1.5$, it seems as if the step size is too large for gradient descent to converge on the minimum. We'll return to this when we discuss the limitations of the algorithm.



If we start at $x_0 = 7$ with $\alpha = 0.2$, we descend into a completely different local minimum.



Example 2

Let's use gradient descent to solve the following problem: *how can we best approximate* $\sin(x)$ *with a degree* 5 *polynomial within the range* $-3 < x < 3$*?*

$$p(x) = a_0 + a_1 x + \cdots + a_5 x^5$$

In order to use gradient descent, we need to phrase the problem in terms of minimizing a function f. Intuitively, our goal is to find the coefficients $a_0, \ldots, a_5$ that make $p(x)$ as close to $\sin(x)$ as possible while x is between -3 and 3. There are many ways we can turn this idea into a function to minimize. One is called least squares:

$$f(a_0, \ldots, a_5) = \int_{-3}^{3} (p(x) - \sin(x))^2 \, dx$$

In short, we define our f as the sum of how incorrect $p(x)$ is at each point. For example, if $p(x) = 2$ near $x = 0$, then f will increase by a lot because $\sin(0) = 0$ not 2. Gradient descent will try to get f as low as possible, which is the same as making $p(x)$ closer to $\sin(x)$. We square the difference so the integral is always positive.

Here's what happens if we start with $a_0, \ldots, a_5$ as random numbers and then move along the negative gradient. The number in the top left shows how many steps we've taken so far, where we use a step size of $\alpha = 0.001$.

We get a pretty good approximation of $\sin(x)$!

$$p(x) = -0.00177 + 0.88974x + 0.00287x^2$$

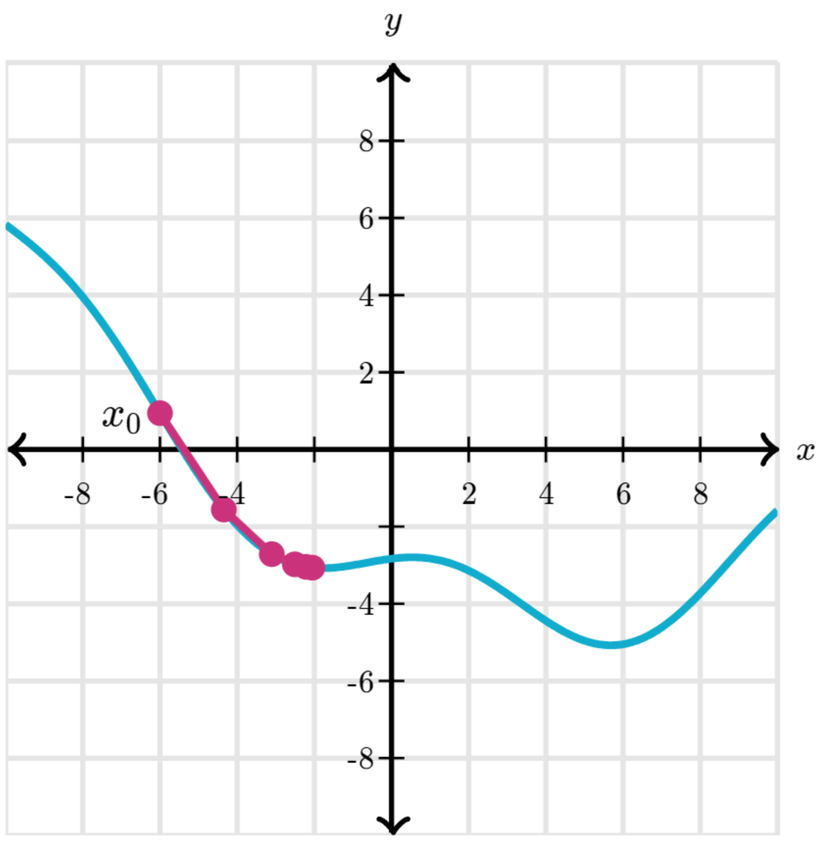
$$- 0.11613x^3 - 0.00048x^4 + 0.00224x^5$$

Technically, the animation above uses something called momentum gradient descent, which is a variant that helps it avoid getting stuck in local minima.

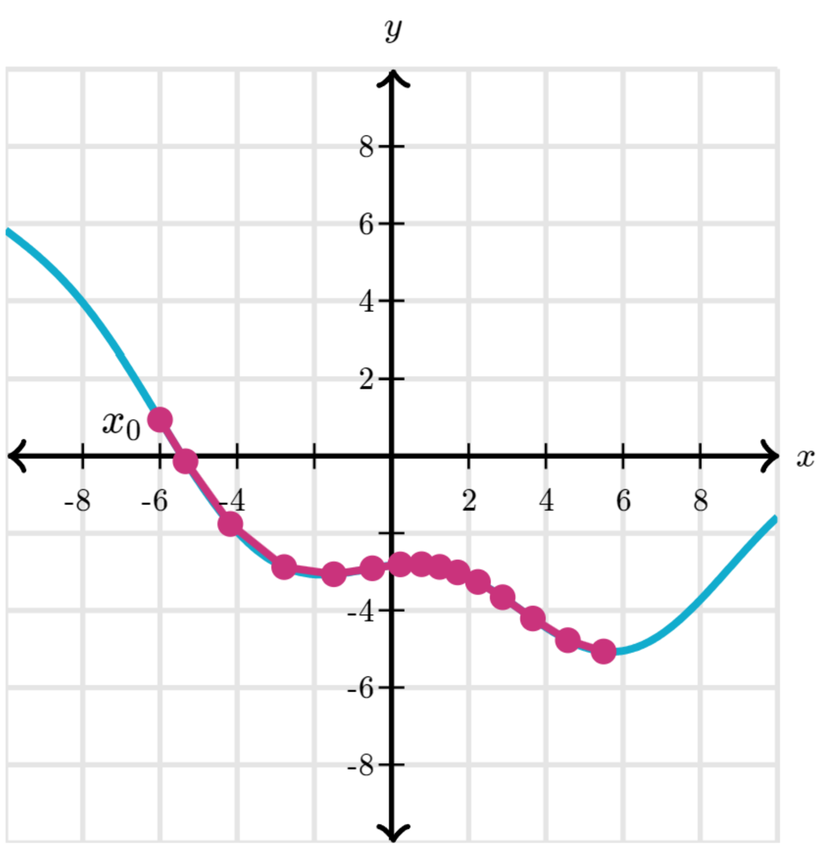
[Hide explanation]

The idea is to imagine we are a ball rolling down a hill. As we move, we go faster and faster. If we encounter a small enough local minimum, we will go straight over it because we have the momentum to carry us through it.

Without momentum, we get stuck in a local minimum:



With momentum, we find the global minimum:



Formally, we start with a position x_0 and a velocity $v_0 = 0$. Every step, we add a fraction λ (usually around 0.9) of the old velocity to the new gradient. That way we keep some of our old momentum each step.

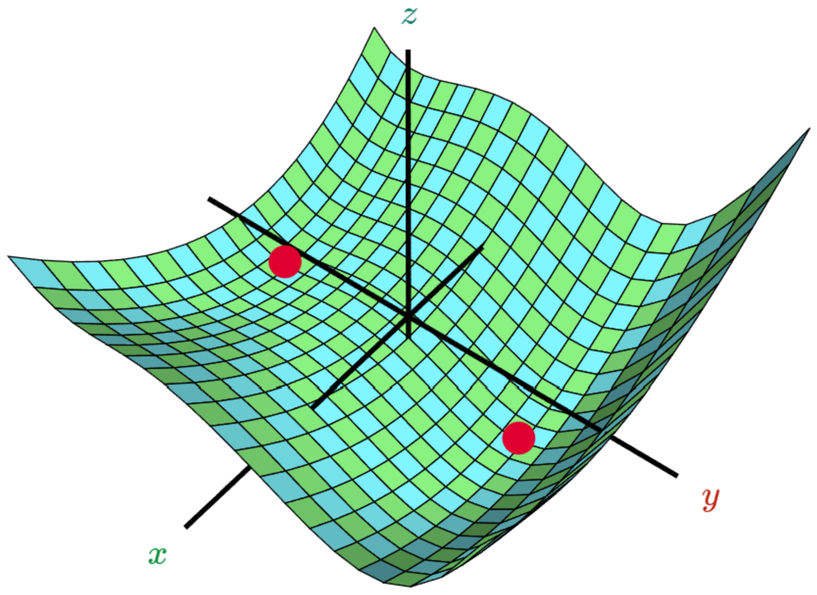
$$v_{n+1} = \lambda v_n - \alpha \nabla f(x_n)$$

$$x_{n+1} = x_n + v_{n+1}$$

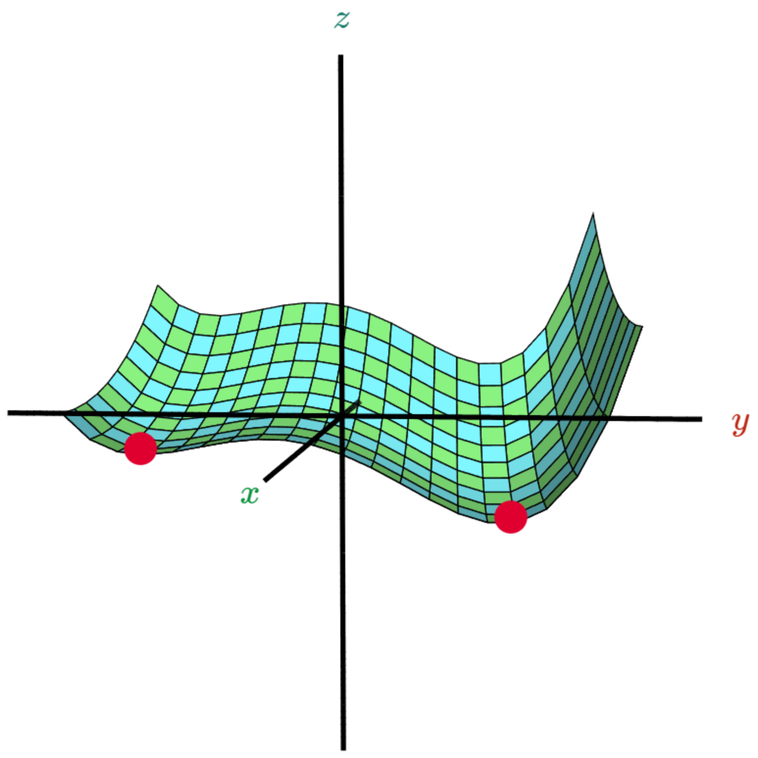
Limitations

Gradient descent has applications whenever we have a function we want to minimize, which is common in machine learning, for example. But it's important to know its shortcomings so that we can plan around them.

One of its limitations is that it only finds local minima (rather than the global minimum). As soon as the algorithm finds some point that's at a local minimum, it will never escape as long as the step size doesn't exceed the size of the ditch.



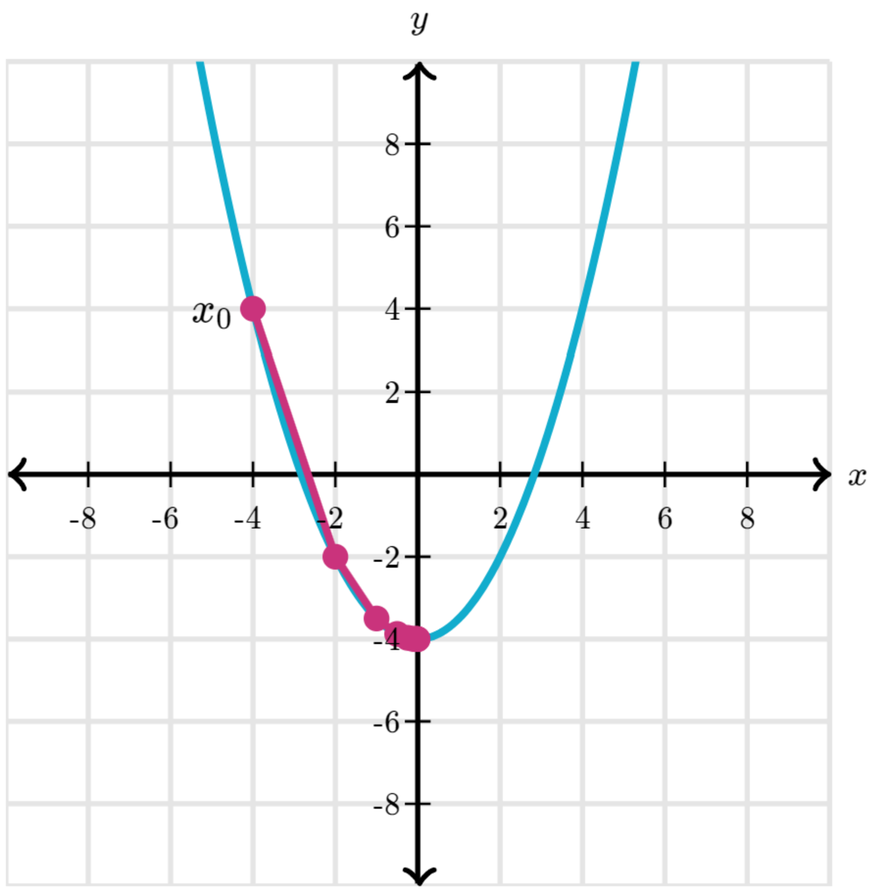
In the graph above, each local minimum has its own valley that would trap a gradient descent algorithm. After all, the algorithm only ever tries to go down, so once it finds a point where every direction leads up, it will stop. Looking at the graph from a different perspective, we also see that one local minimum is lower than the other.



When we minimize a function, we want to find the global minimum, but there is no way that gradient descent can distinguish global and local minima.

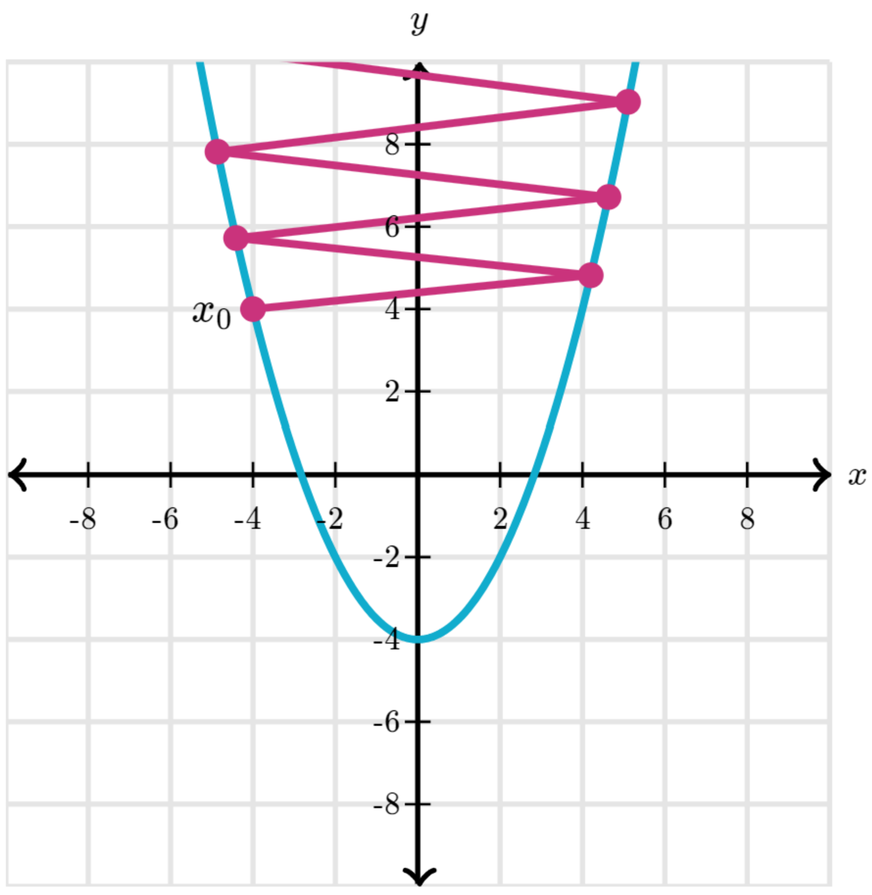
Another limitation of gradient descent concerns the step size α. A good step size moves toward the minimum rapidly, each step making substantial

progress.



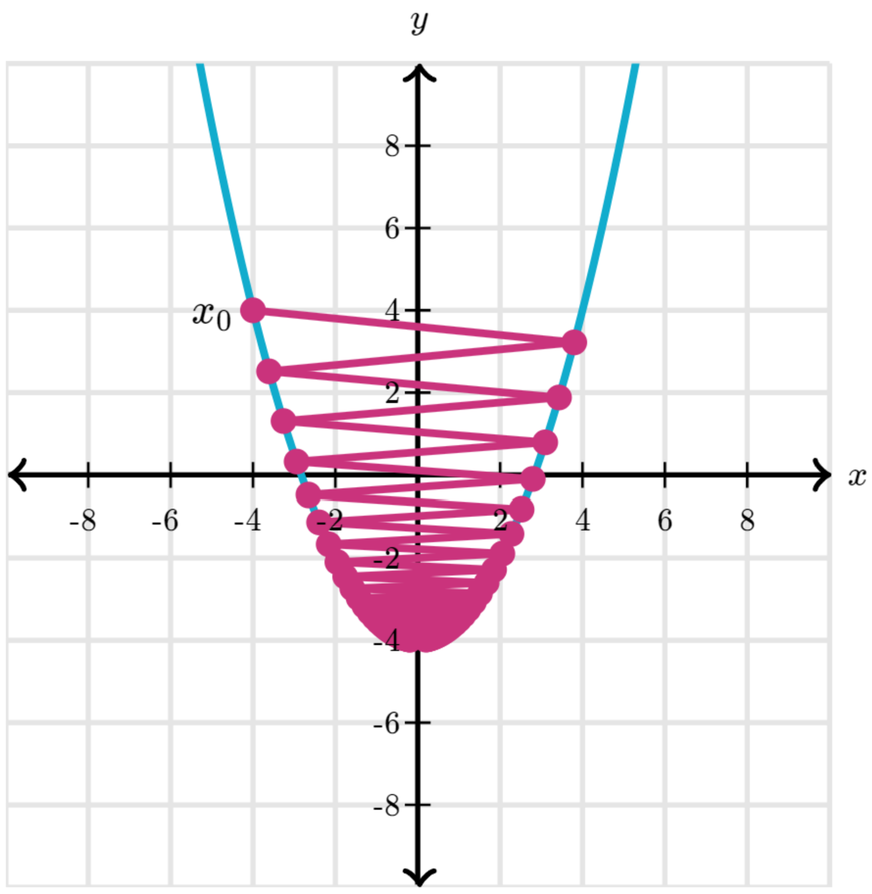
Good step size converges quickly.

If the step size is too large, however, we may never converge to a local minimum because we overshoot it every time.



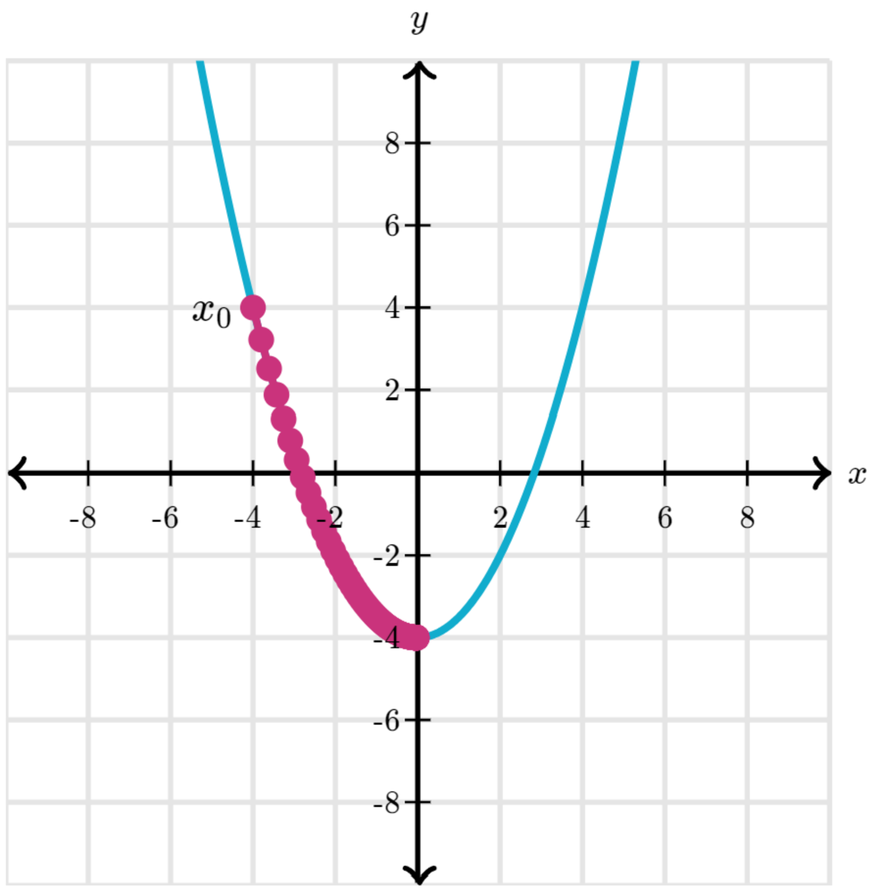
Large step size diverges.

If we are lucky and the algorithm converges anyway, it still might take more steps than it needed.



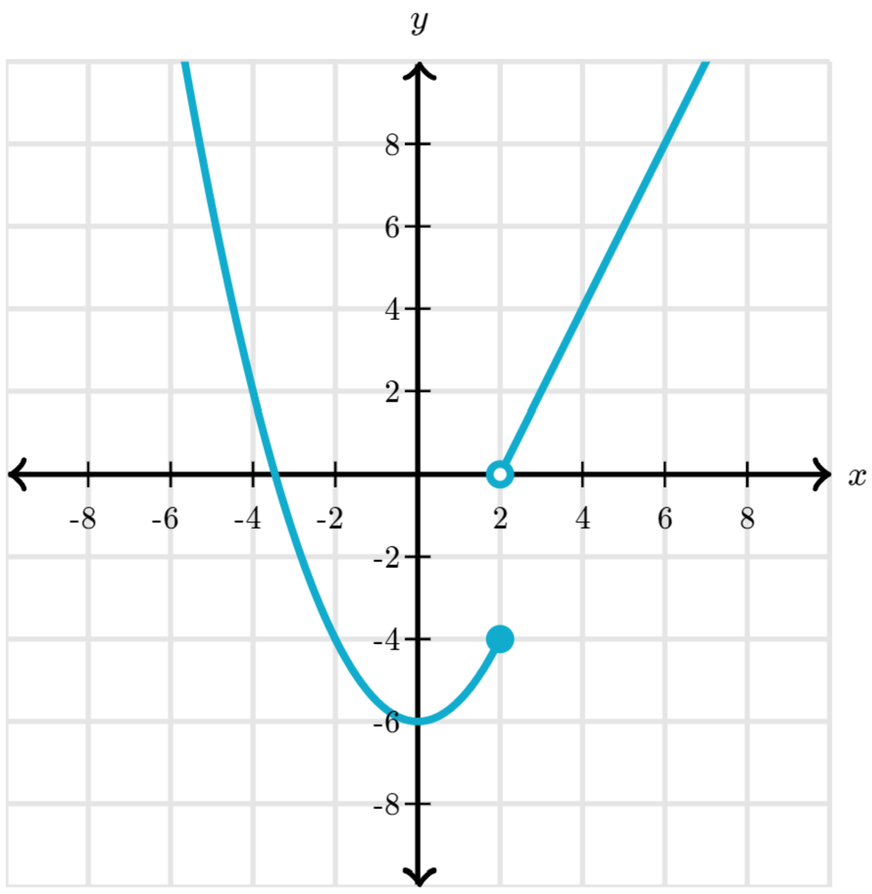
Large step size converges slowly.

If the step size is too small, then we'll be more likely to converge, but we'll take far more steps than were necessary. This is a problem when the function we're minimizing has thousands or millions of variables, and evaluating it is cumbersome.



Tiny step size converges slowly.

A final limitation is that gradient descent only works when our function is differentiable everywhere. Otherwise we might come to a point where the gradient isn't defined, and then we can't use our update formula.



Gradient descent fails for non-differentiable functions.

Summary

- Gradient descent minimizes differentiable functions that output a number and have any amount of input variables.
- It does this by taking a guess x_0 and successively applying the formula $x_{n+1} = x_n - \alpha \nabla f(x_n)$. In words, the formula says to take a small step in the direction of the negative gradient.
- Gradient descent can't tell whether a minimum it has found is local or global.
- The step size α controls whether the algorithm converges to a minimum quickly or slowly, or whether it diverges.
- Many real world problems come down to minimizing a function.