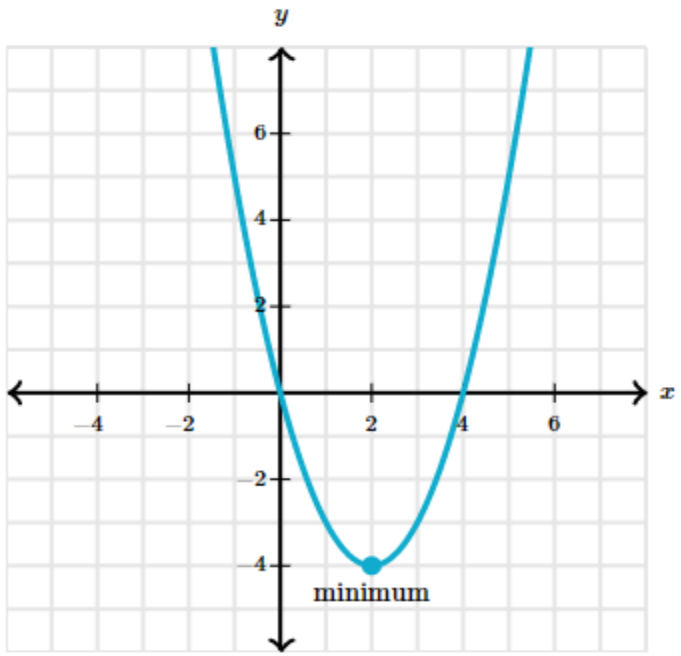
Gradient descent

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.

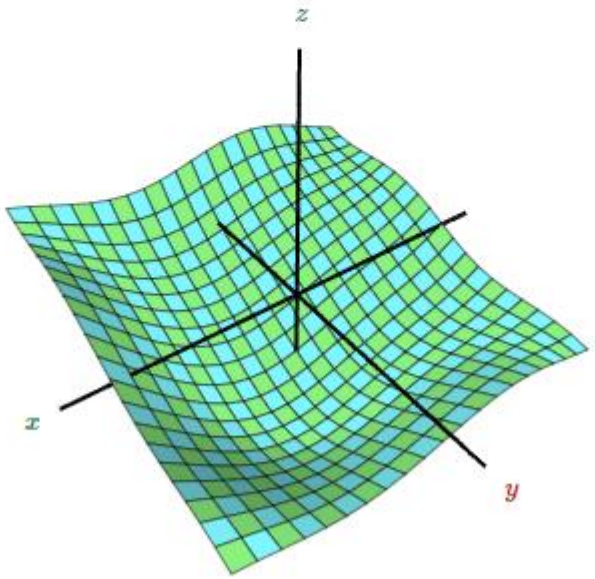
This distinction is worth emphasizing because it's what makes gradient descent useful. If we had a simple formula like $f(x) = x^2 - 4x$, then we could easily solve $\nabla f = 0$ to find that $x = 2$ minimizes $f(x)$. Or we could use gradient descent to get a numerical approximation, something like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

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

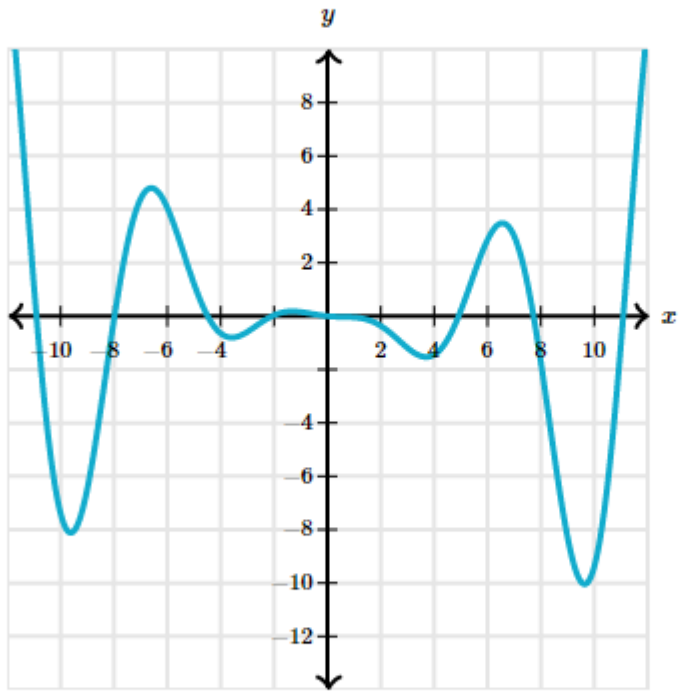
More generally, we can write a formula for turning x_n into x_{n+1}:

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

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.

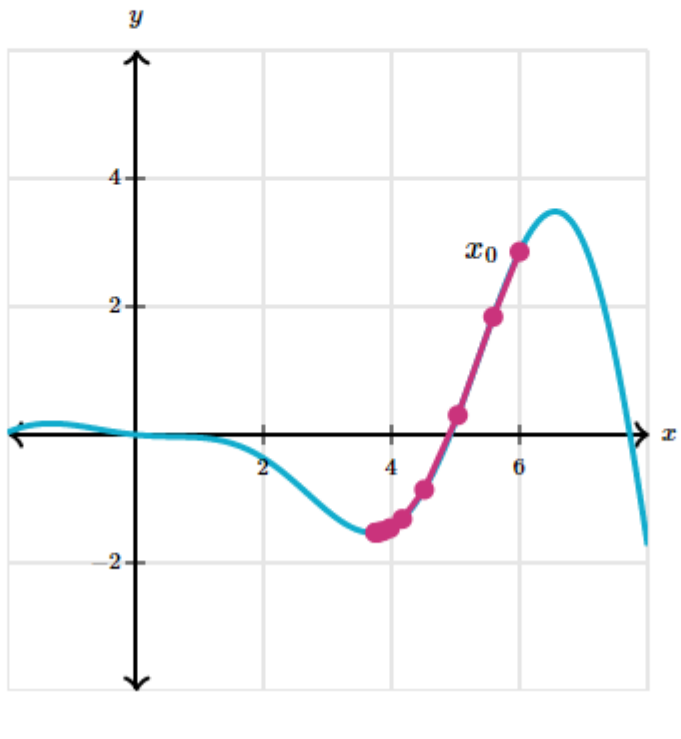
Example 1

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

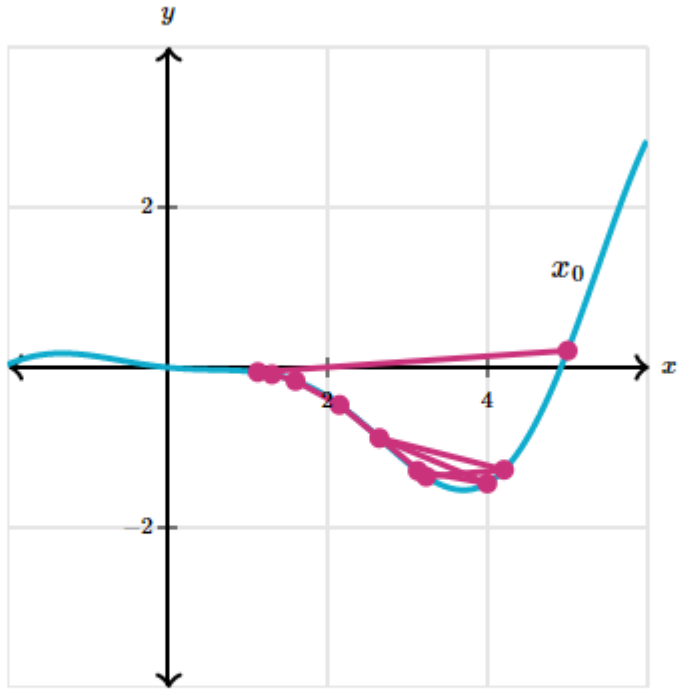


As we can see from the graph, this function has many local minima. Gradient descent will find different ones depending on our initial guess and our step size.

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.



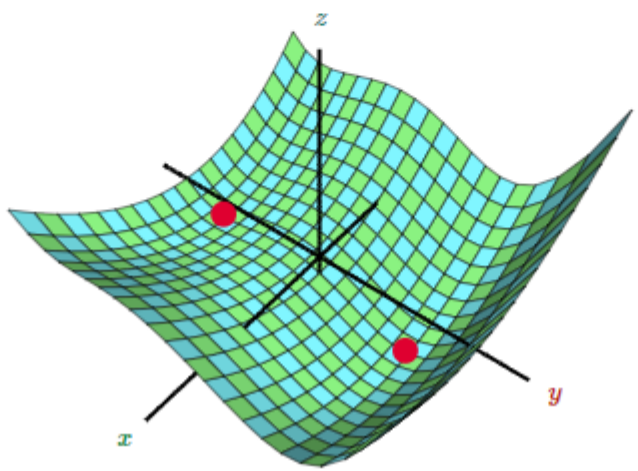
If we start at $x_0 = 7$ with $\alpha = 0.2$, we descend into a completely different local minimum.

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

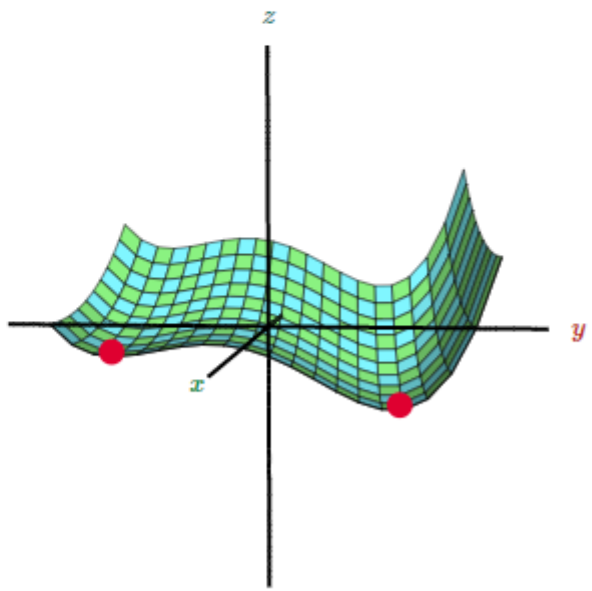
Limitations

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.

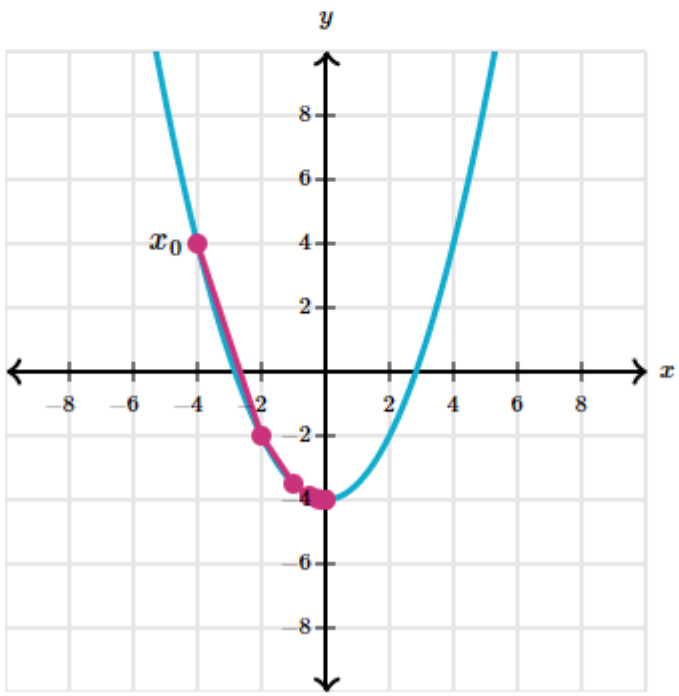


In the graph above, each local minimum has its own valley that would trap a gradient descent algorithm. After all, the algorithm only ever tries to go down, so once it finds a point where every direction leads up, it will stop. Looking at the graph from a different perspective, we also see that one local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



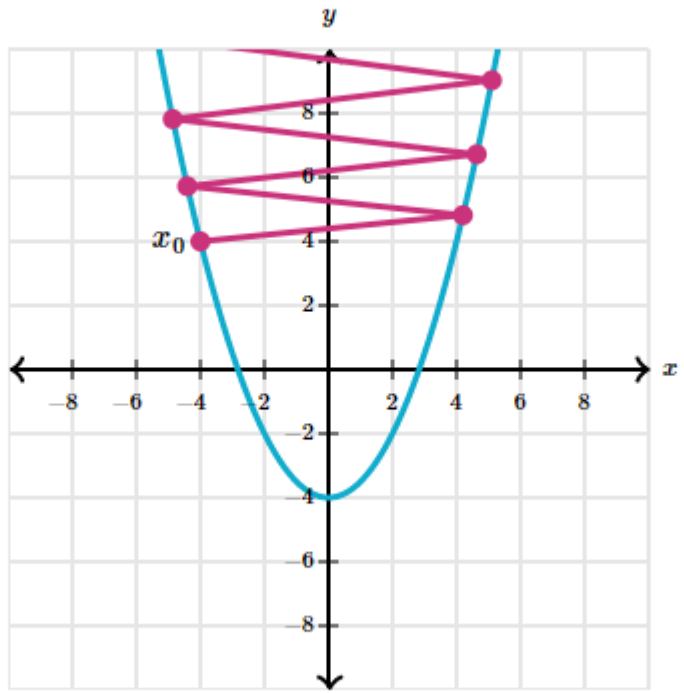
When we minimize a function, we want to find the global minimum, but there is no way that gradient descent can distinguish global and local minima.

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



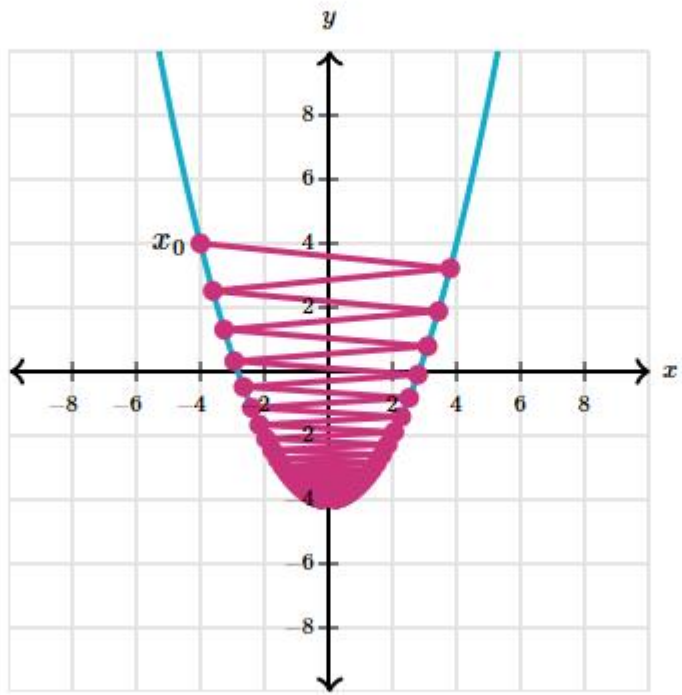
Good step size converges quickly.

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



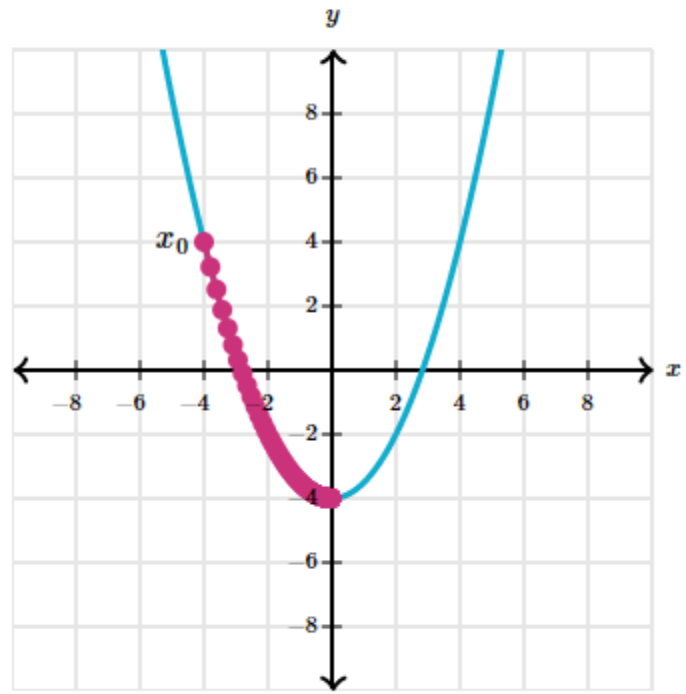
Large step size diverges.

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.

Gradient Descent Algorithm: How does it Work in Machine Learning?

What is a Cost Function?
It is a **function** that measures the performance of a model for any given data. **Cost Function** quantifies the error between predicted values and expected values and presents it in the form of a single real number.
After making a hypothesis with initial parameters, we calculate the Cost function. And with a goal to reduce the cost function, we modify the parameters by using the Gradient descent algorithm over the given data.
Here's the mathematical representation for it:

$$h_\theta = \theta_0 + \sum_{i=1}^{m} \theta_i x_i \quad if\, m{=}1\ then$$

$$consider:\ \ h_\theta = \theta_0 + \theta_1 x_i \quad parametrs \quad \theta_0, \theta_1$$

Cost function : $j(\theta_0\ , \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}(\theta_0 + \theta_1 x_i - y_i)^2$

Condition :

$$\frac{\partial j}{\partial \theta_0} = \frac{1}{\mathrm{m}}\sum_{i=1}^{m}(\theta_0 + \theta_1 x_i - y_i)$$

$$\frac{\partial j}{\partial \theta_1} = \frac{1}{\mathrm{m}}\sum_{i=1}^{m}\theta_1(\theta_0 + \theta_1 x_i - y_i)$$

What is Gradient Descent?

Gradient descent is an optimization algorithm used in machine learning to minimize the cost function by iteratively adjusting parameters in the direction of the negative gradient, aiming to find the optimal set of parameters.

The cost function represents the discrepancy between the predicted output of the model and the actual output. The goal of gradient descent is to find the set of parameters that minimizes this discrepancy and improves the model's performance.

The algorithm operates by calculating the gradient of the cost function, which indicates the direction and magnitude of steepest ascent. However, since the objective is to minimize the cost function, gradient descent moves in the opposite direction of the gradient, known as the negative gradient direction.

By iteratively updating the model's parameters in the negative gradient direction, gradient descent gradually converges towards the optimal set of parameters that yields the lowest cost. The learning rate, a hyper parameter, determines the step size taken in each iteration, influencing the speed and stability of convergence.

Gradient descent can be applied to various machine learning algorithms, including linear regression, logistic regression, neural networks, and support vector machines. It provides a general framework for optimizing models by iteratively refining their parameters based on the cost function.

Example of Gradient Descent

Let’s say you are playing a game where the players are at the top of a mountain, and they are asked to reach the lowest point of the mountain. Additionally, they are blindfolded. So, what approach do you think would make you reach the lake?

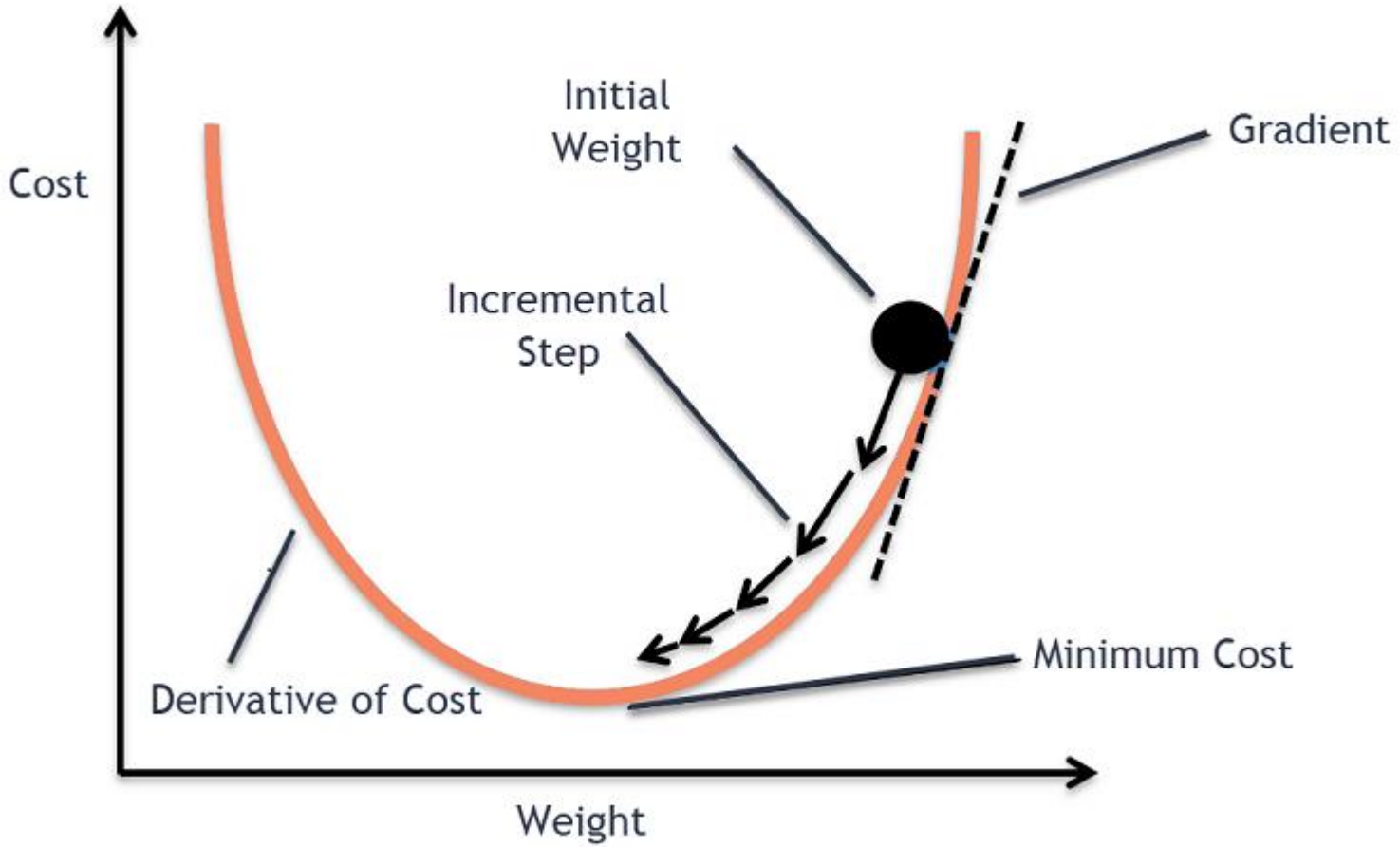
Take a moment to think about this before you read on.

The best way is to observe the ground and find where the land descends. From that position, take a step in the descending direction and iterate this process until we reach the lowest point.

Gradient descent is an iterative optimization algorithm for finding the local minimum of a function.

To find the local minimum of a function using gradient descent, we must take steps proportional to the negative of the gradient (move away from the gradient) of the function at the current point. If we take steps proportional to the positive of the gradient (moving towards the gradient), we will approach a local maximum of the function, and the procedure is called **Gradient Ascent.**

Gradient descent was originally proposed by **CAUCHY** in 1847. It is also known as steepest descent.



The goal of the gradient descent algorithm is to minimize the given function (say cost function). To achieve this goal, it performs two steps iteratively:

1. **Compute the gradient** (slope), the first order derivative of the function at that point
2. **Make a step (move) in the direction opposite to the gradient**, opposite direction of slope increase from the current point by alpha times the gradient at that point

$$\theta_{i(new)} = \theta_{i\,(old)} - \alpha_i \frac{\partial j(\theta_i)}{\partial \theta_i}$$

Alpha is called **Learning rate** – a tuning parameter in the optimization process. It decides the length of the steps.