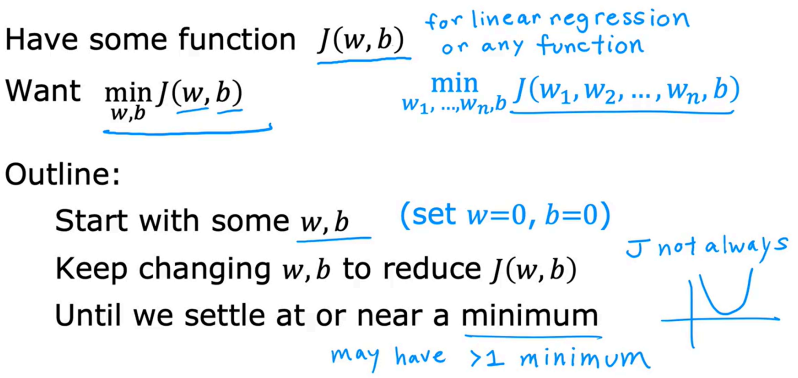
Train the model with gradient descent

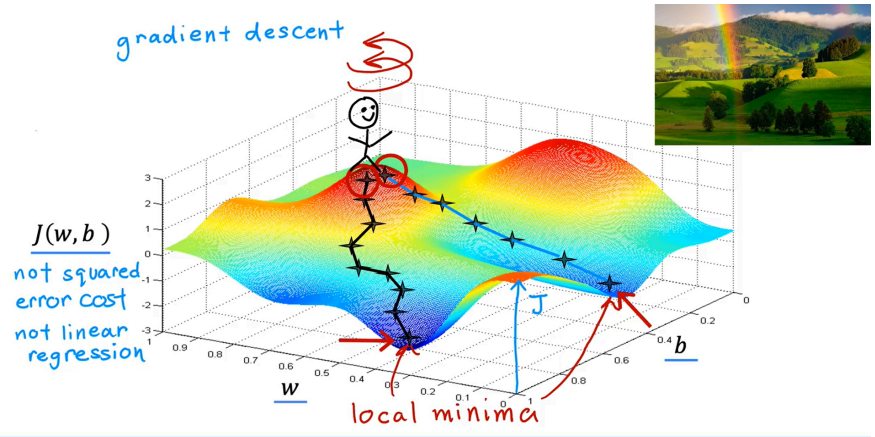
Gradient descent

- Gradient descent is used all over the place in machine learning, not just linear regression, but for training for example some of the most advanced neural network models, also called deep learning models.
- Here's an overview of what we'll do with gradient descent.



- You have the cost function j of w, b right here that you want to minimize.
- In the example we've seen so far, this is a cost function for linear regression,
- but it turns out that gradient descent is an algorithm that you can use to try to minimize any function, not just a cost function for linear regression.
- Just to make this discussion on gradient descent more general, it turns out that gradient descent applies to more general functions, including other cost functions that work with models that have more than two parameters.
- It turns out that gradient descent is an algorithm that you can apply to try to minimize this cost function j as well.
- What you're going to do is just to start off with some initial guesses for w and b. In linear regression, it won't matter too much what the initial value are, so a common choice is to set them both to 0

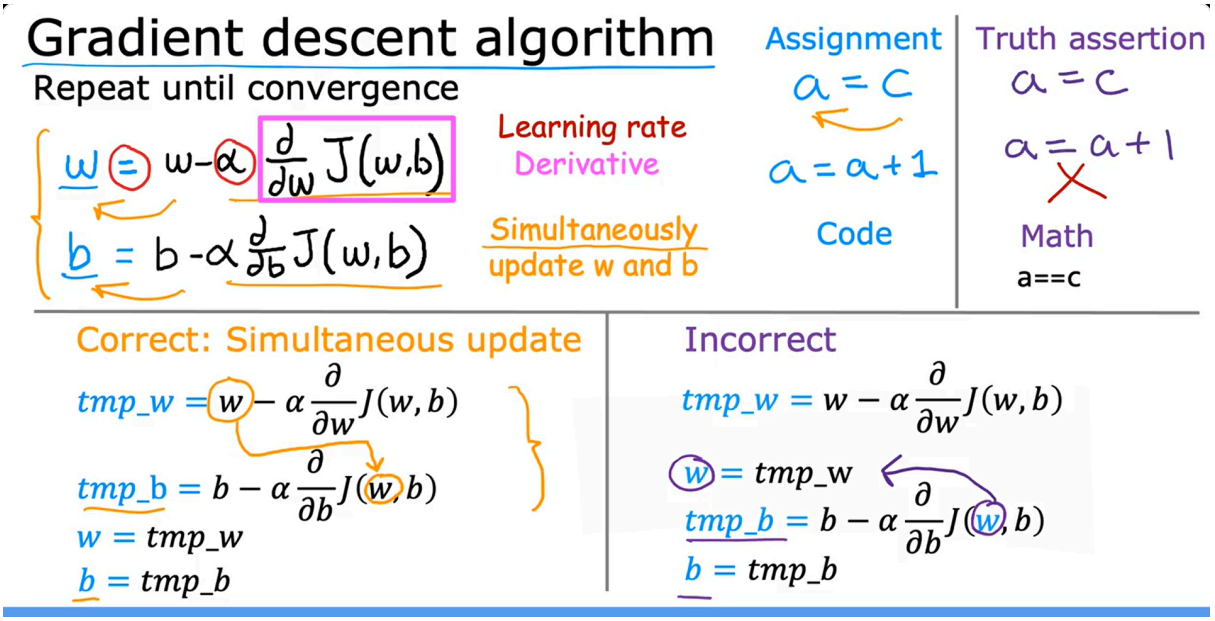
- For example, you can set w to 0 and b to 0 as the initial guess. With the gradient descent algorithm, what you're going to do is, you'll keep on changing the parameters w and b a bit every time to try to reduce the cost j of w, b until hopefully j settles at or near a minimum.
- One thing I should note is that for some functions j that may not be a bow shape or a hammock shape, it is possible for there to be more than one possible minimum.



- If it helps you to relax, imagine that there's lots of really nice green grass and butterflies and flowers is a really nice hill.
- Your goal is to start up here and get to the bottom of one of these valleys as efficiently as possible.
- What the gradient descent algorithm does is, you're going to spin around 360 degrees and look around and ask yourself, if I were to take a tiny little baby step in one direction, and I want to go downhill as quickly as possible to or one of these valleys. What direction do I choose to take that baby step?
- Well, if you want to walk down this hill as efficiently as possible, it turns out that if you're standing at this point in the hill and you look around, you will notice that the best direction to take your next step downhill is roughly that direction.
- Mathematically, this is the direction of steepest descent. It means that when you take a tiny baby little step, this takes you downhill faster than a tiny little baby step you could have taken in any other direction. After taking this first step, you're now at this point on the hill over here.

- Now let's repeat the process. Standing at this new point, you're going to again spin around 360 degrees and ask yourself, in what direction will I take the next little baby step in order to move downhill? If you do that and take another step, you end up moving a bit in that direction and you can keep going. From this new point, you can again look around and decide what direction would take you downhill most quickly.
- Take another step, another step, and so on, until you find yourself at the bottom of this valley, at this local minimum, right here. What you just did was go through multiple steps of gradient descent.
- It turns out, gradient descent has an interesting property. Remember that you can choose a starting point at the surface by choosing starting values for the parameters w and b. When you perform gradient descent a moment ago, you had started at this point over here.
- Now, imagine if you try gradient descent again, but this time you choose a different starting point by choosing parameters that place your starting point just a couple of steps to the right over here. If you then repeat the gradient descent process, which means you look around, take a little step in the direction of steepest ascent so you end up here. Then you again look around, take another step, and so on. If you were to run gradient descent this second time, starting just a couple steps in the right of where we did it the first time, then you end up in a totally different valley.
- This different minimum over here on the right. The bottoms of both the first and the second valleys are called local minima. Because if you start going down the first valley, gradient descent won't lead you to the second valley, and the same is true if you started going down the second valley, you stay in that second minimum and not find your way into the first local minimum.

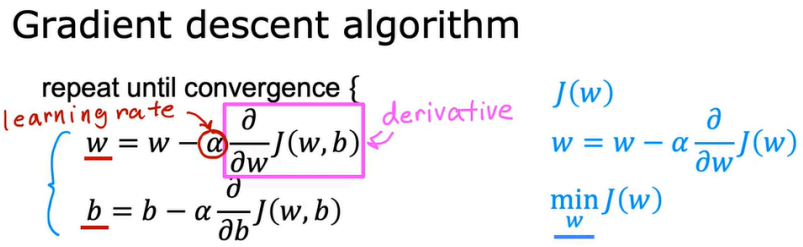
Implementing gradient descent

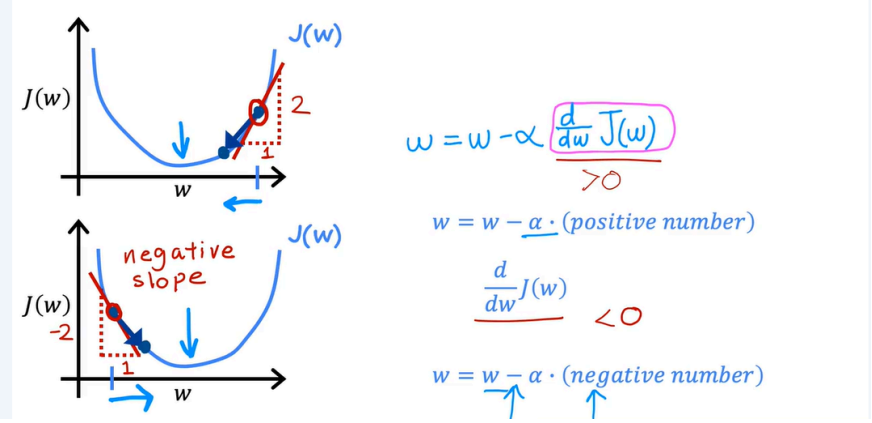


- Now, this dive more deeply into what the symbols in this equation means. The symbol here is the Greek alphabet Alpha (α). In this equation, Alpha is also called **the learning rate**.
- The learning rate is usually a **small positive number between 0 and 1 and it might be say, 0.01.** What Alpha does is, it basically controls how big of a step you take downhill.
- If Alpha is very large, then that corresponds to a very aggressive gradient descent procedure where you're trying to take huge steps downhill. If Alpha is very small, then you'd be taking small baby steps downhill.
- We'll come back later to dive more deeply into how to choose a good learning rate Alpha.

Gradient descent intuition

- In order to do this let's use a slightly simpler example where we work on minimizing just one parameter. Let's say that you have a cost function J of just one parameter w with w is a number.





repeat until convergence: {

$$w = w - \alpha \frac{\partial J(w,b)}{\partial w}$$

$$b = b - \alpha \frac{\partial J(w,b)}{\partial b}$$

}

Learning rate (alpha)

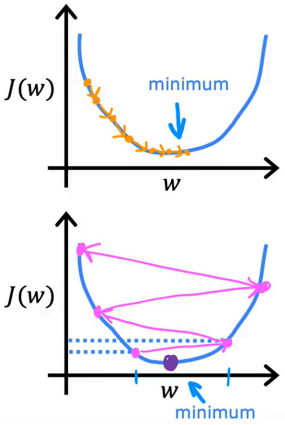
- The choice of the learning rate α will have a huge impact on the efficiency of your implementation of gradient descent.
- And if α, the learning rate is chosen poorly rate of descent may not even work at all.

$$w = w - \alpha \frac{d}{dw} J(w)$$

If α is too small...
Gradient descent may be slow.

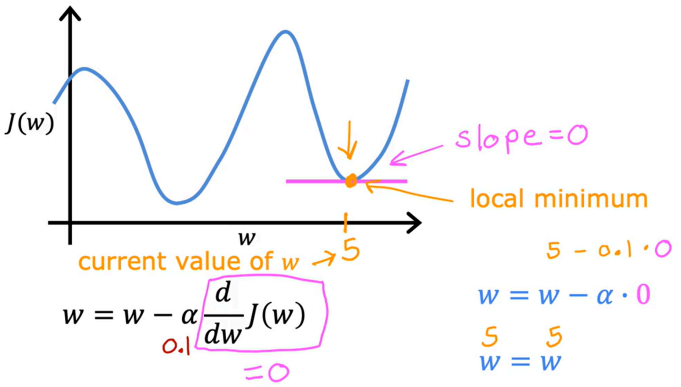
If α is too large...

Gradient descent may:
- Overshoot, never reach minimum
- Fail to converge, diverge



→ So to summarize if the learning rate is too small, then gradient descents will work, but it will be slow.

→ if the learning rate is too large, then creating the sense may overshoot and may never reach the minimum.



- if your parameters have already brought you to a local minimum, then further gradient descent steps to absolutely nothing.

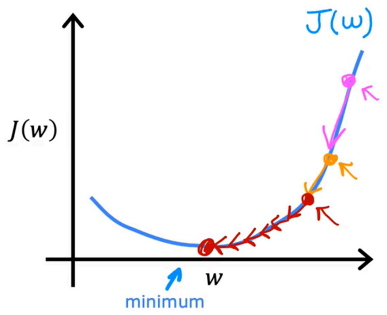
Can reach local minimum with fixed learning rate

$$w = w - \alpha \frac{d}{dw} J(w)$$

smaller

not as large

large



Near a local minimum,
- Derivative becomes smaller
- Update steps become smaller

Can reach minimum without decreasing learning rate α

Gradient descent for linear regression

Linear regression model

$$f_{w,b}(x) = wx + b$$

Cost function

$$J(w,b) = \frac{1}{2m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)})^2$$

Gradient descent algorithm

repeat until convergence {

$$w = w - \alpha \frac{\partial}{\partial w} J(w,b) \rightarrow \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)}) x^{(i)}$$

$$b = b - \alpha \frac{\partial}{\partial b} J(w,b) \rightarrow \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)})$$

}

explaination (calculus related):

(Optional)

$$\frac{\partial}{\partial w} J(w,b) = \frac{d}{dw} \frac{1}{2m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right)^2 = \frac{d}{dw} \frac{1}{2m} \sum_{i=1}^{m} \left(wx^{(i)} + b - y^{(i)}\right)^2$$

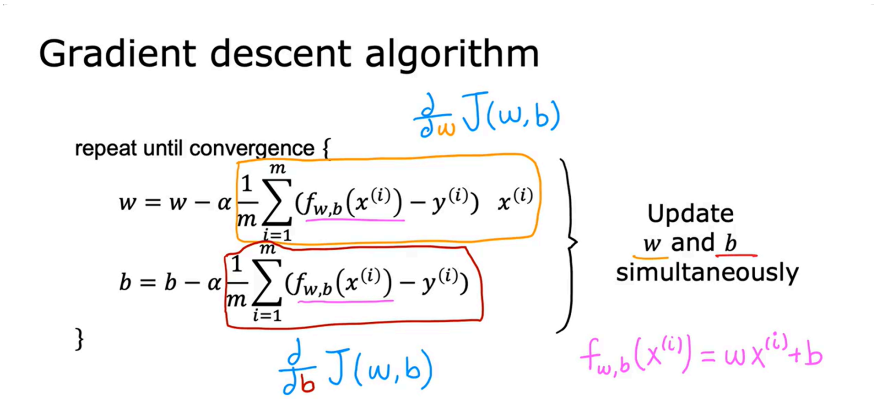
$$= \frac{1}{2m} \sum_{i=1}^{m} \left(wx^{(i)} + b - y^{(i)}\right) 2x^{(i)} = \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)}) x^{(i)}$$

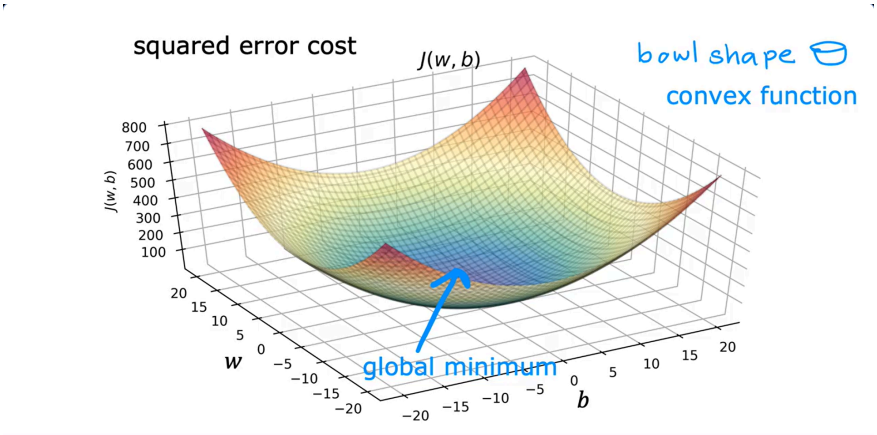
$$\frac{\partial}{\partial b} J(w,b) = \frac{d}{db} \frac{1}{2m} \sum_{i=1}^{m} \left(f_{w,b}(x^{(i)}) - y^{(i)}\right)^2 = \frac{d}{db} \frac{1}{2m} \sum_{i=1}^{m} \left(wx^{(i)} + b - y^{(i)}\right)^2$$

$$= \frac{1}{2m} \sum_{i=1}^{m} \left(wx^{(i)} + b - y^{(i)}\right) 2 = \frac{1}{m} \sum_{i=1}^{m} (f_{w,b}(x^{(i)}) - y^{(i)})$$

no $x^{(i)}$

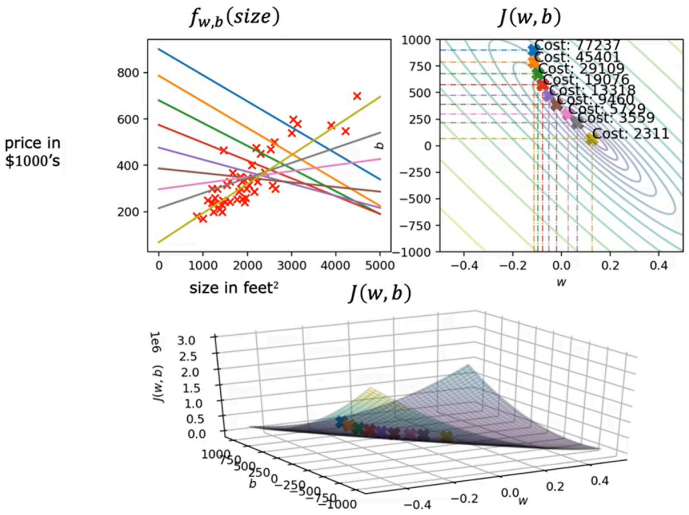
Apply Gradient descent algorithm to Cost function





- One the shoe we saw with gradient descent is that it can lead to a local minimum instead of a global minimum. Whether global minimum means the point that has the lowest possible value for the cost function J of all possible points. You may recall this surface plot that looks like an outdoor park with a few hills with the process and the birds as a relaxing Hobo Hill. This function has more than one local minimum.
- When you're using a squared error cost function with linear regression, the cost function does not and will never have multiple local minima. It has a single global minimum because of this bowl-shape.
- The technical term for this is that this cost function is a convex function.
    - Informally, a convex function is of bowl-shaped function and it cannot have any local minima other than the single global minimum.
- When you implement gradient descent on a convex function, one nice property is that so long as you're learning rate is chosen appropriately, it will always converge to the global minimum.

Running gradient descent



- Often w and b will both be initialized to 0, but for this demonstration, lets initialized w = -0.1 and b = 900. So this corresponds to f(x) = -0.1x + 900. (the blue line)
- Now, if we take one step using gradient descent, we ended up going from this point of the cost function out here to this point just down and to the right and notice that the straight line fit is also changed a bit.
- The cost function has now moved to this third and again the function f(x) has also changed a bit.
- As you take more of these steps, the cost is decreasing at each update. So the parameters w and b are following this trajectory.

“Batch” Gradient Descent

"Batch" gradient descent

DeepLearning.AI
THE BATCH

"Batch": Each step of gradient descent uses all the training examples.

other gradient descent: subsets

	x size in feet²	y price in $1000's
(1)	2104	400
(2)	1416	232
(3)	1534	315
(4)	852	178
...	...	...
(47)	3210	870

$m = 47$

$$\sum_{i=1}^{m} \left(f_{w,b}\left(x^{(i)}\right) - y^{(i)}\right)^2$$