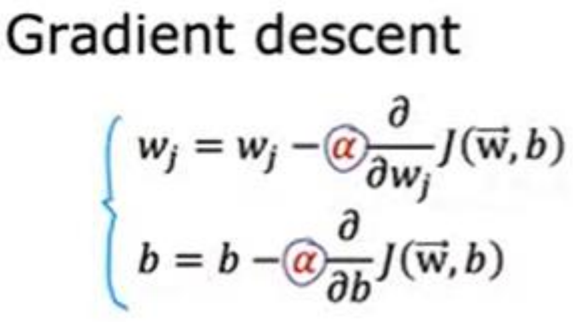
Vérification de la convergence de la descente de gradient

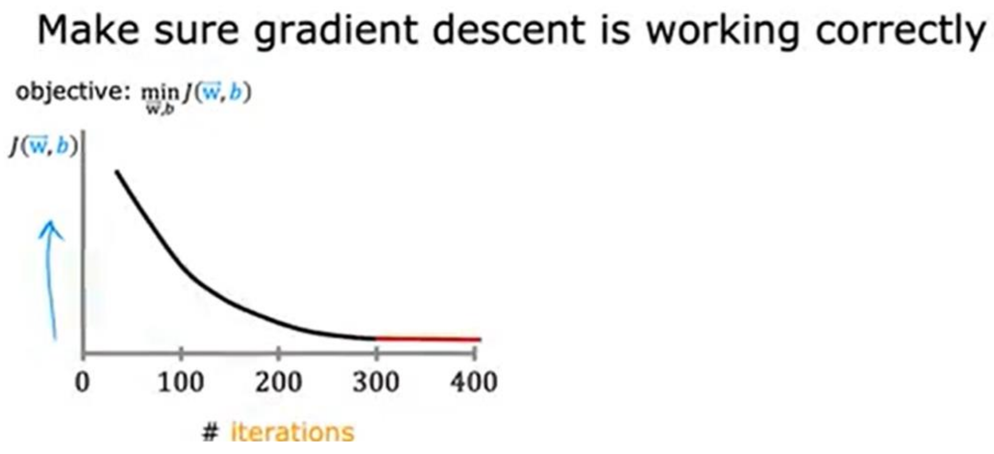
When running gradient descent, how can you tell if it is converging? That is, whether it's helping you to find parameters close to the global minimum of the cost function. By learning to recognize what a well-running implementation of gradient descent looks like, we will also, in a later video, be better able to choose a good learning rate Alpha. Let's take a look. As a reminder, here's the gradient descent rule. One of the key choices is the choice of the learning rate Alpha.



Gradient descent

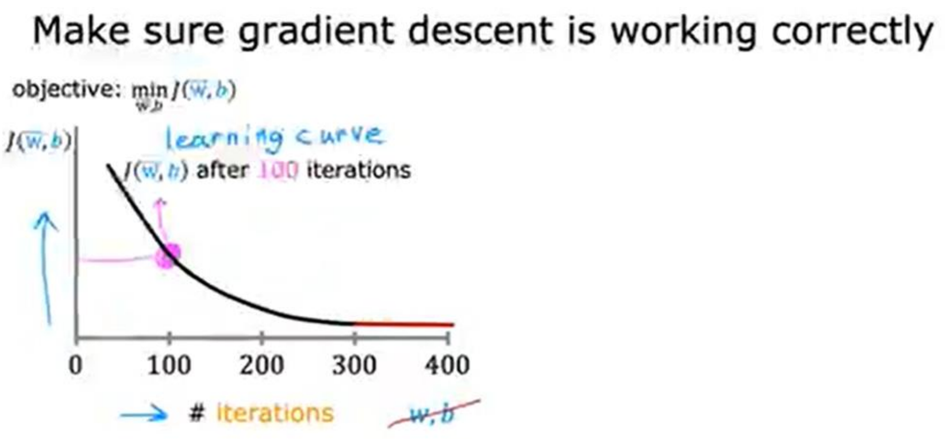
$$\begin{cases} w_j = w_j - \enclose{circle}{\alpha}\dfrac{\partial}{\partial w_j}J(\vec{\mathrm{w}},b) \\ b = b - \enclose{circle}{\alpha}\dfrac{\partial}{\partial b}J(\vec{\mathrm{w}},b) \end{cases}$$

Here's something that I often do to make sure that gradient descent is working well. Recall that the job of gradient descent is to find parameters w and b that hopefully minimize the cost function J. What I'll often do is plot the cost function J, which is calculated on the training set, and I plot the value of J at each iteration of gradient descent. Remember that each iteration means after each simultaneous update of the parameters w and b. In this plot, the horizontal axis is the number of iterations of gradient descent that you've run so far. You may get a curve that looks like this.

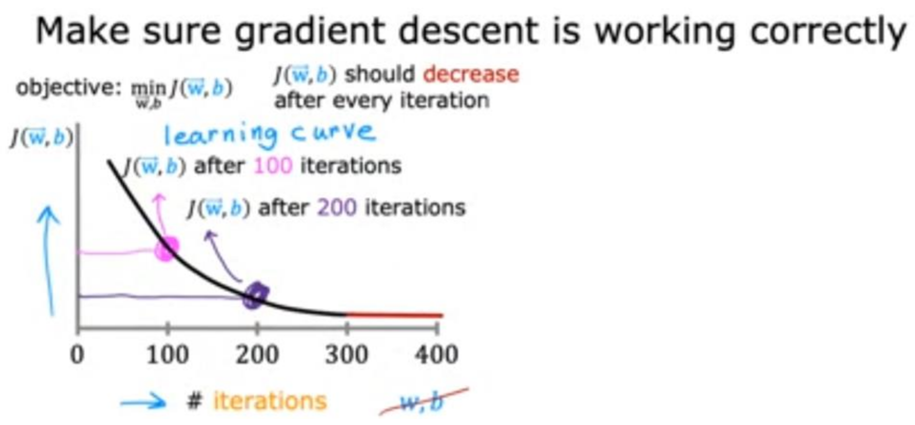


Notice that the horizontal axis is the number of iterations of gradient descent and not a parameter like w or b. This differs from previous graphs you've seen where the vertical axis was cost J and the horizontal axis was a single parameter like w or b. This curve is also called a learning curve. Note that there are a few different types of learning curves used in machine learning, and you see some of the types later in this course as well.

Concretely, if you look here at this point on the curve, this means that after you've run gradient descent for 100 iterations, meaning 100 simultaneous updates of the parameters, you have some learned values for w and b. If you compute the cost J, w, b for those values of w and b, the ones you got after 100 iterations, you get this value for the cost J. That is this point on the vertical axis.

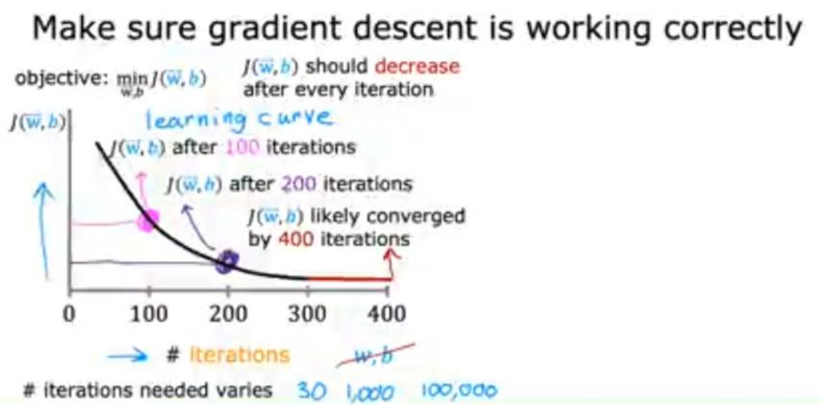


This point here corresponds to the value of J for the parameters that you got after 200 iterations of gradient descent. Looking at this graph helps you to see how your cost J changes after each iteration of gradient descent. If gradient descent is working properly, then the cost J should decrease after every single iteration. If J ever increases after one iteration, that means either Alpha is chosen poorly, and it usually means Alpha is too large, or there could be a bug in the code.



Another useful thing that this part can tell you is that if you look at this curve, by the time you reach maybe 300 iterations also, the cost J is leveling off and is no longer decreasing much. By 400 iterations, it looks like the curve has flattened out. This means that gradient descent has more or less converged because the curve is no longer decreasing.

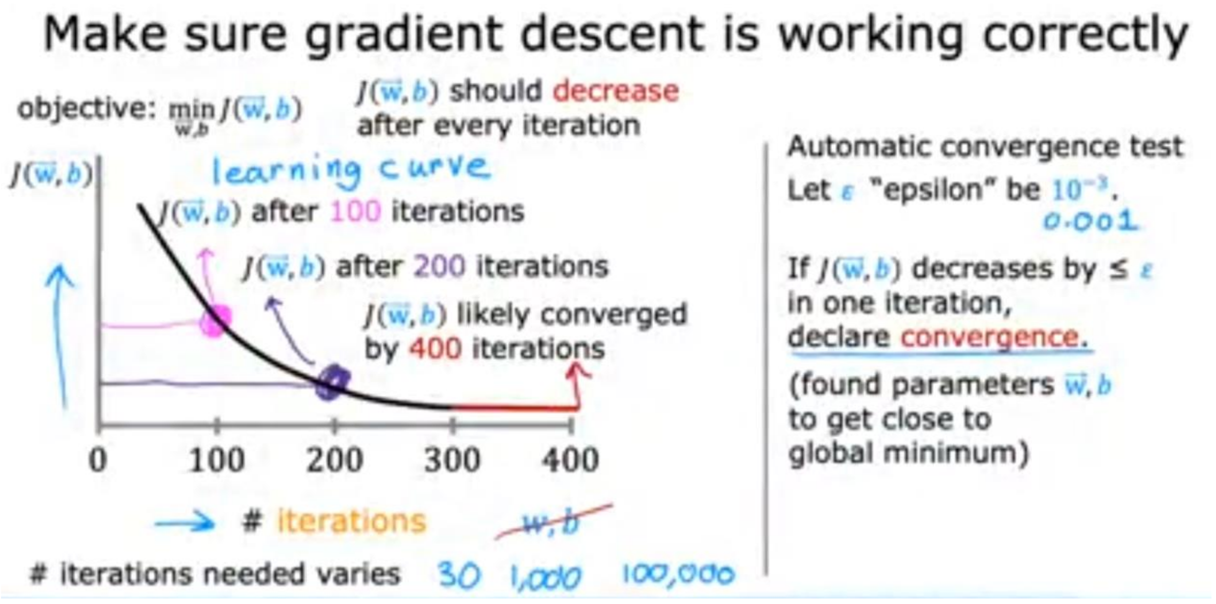
Looking at this learning curve, you can try to spot whether or not gradient descent is converging. By the way, the number of iterations that gradient descent takes a conversion can vary a lot between different applications. In one application, it may converge after just 30 iterations. For a different application, it could take 1,000 or 100,000 iterations.



It turns out to be very difficult to tell in advance how many iterations gradient descent needs to converge, which is why you can create a graph like this, a learning curve. Try to find out when you can start training your particular model. Another way to decide when your model is done training is with an automatic convergence test.

Here is the Greek alphabet epsilon. Let's let epsilon be a variable representing a small number, such as 0.001 or 10^-3. If the cost J decreases by less than this number epsilon on one iteration, then you're likely on this flattened part of the curve that you see on the left and you can declare convergence. Remember, convergence, hopefully in the case that you found parameters w and b that are close to the minimum possible value of J.

I usually find that choosing the right threshold epsilon is pretty difficult. I actually tend to look at graphs like this one on the left, rather than rely on automatic convergence tests. Looking at the solid figure can tell you, I'll give you at some advanced warning if maybe gradient descent is not working correctly as well.


Make sure gradient descent is working correctly
objective: $\min_{\vec{w},b} J(\vec{w},b)$
$J(\vec{w},b)$ should decrease after every iteration
$J(\vec{w},b)$
learning curve
$J(\vec{w},b)$ after 100 iterations
$J(\vec{w},b)$ after 200 iterations
$J(\vec{w},b)$ likely converged by 400 iterations
→ # iterations
~~w, b~~
# iterations needed varies 30 1,000 100,000
Automatic convergence test
Let ε "epsilon" be 10^{-3}.
If $J(\vec{w},b)$ decreases by $\leq \varepsilon$ in one iteration, declare convergence.
(found parameters $\vec{w}, b$ to get close to global minimum)