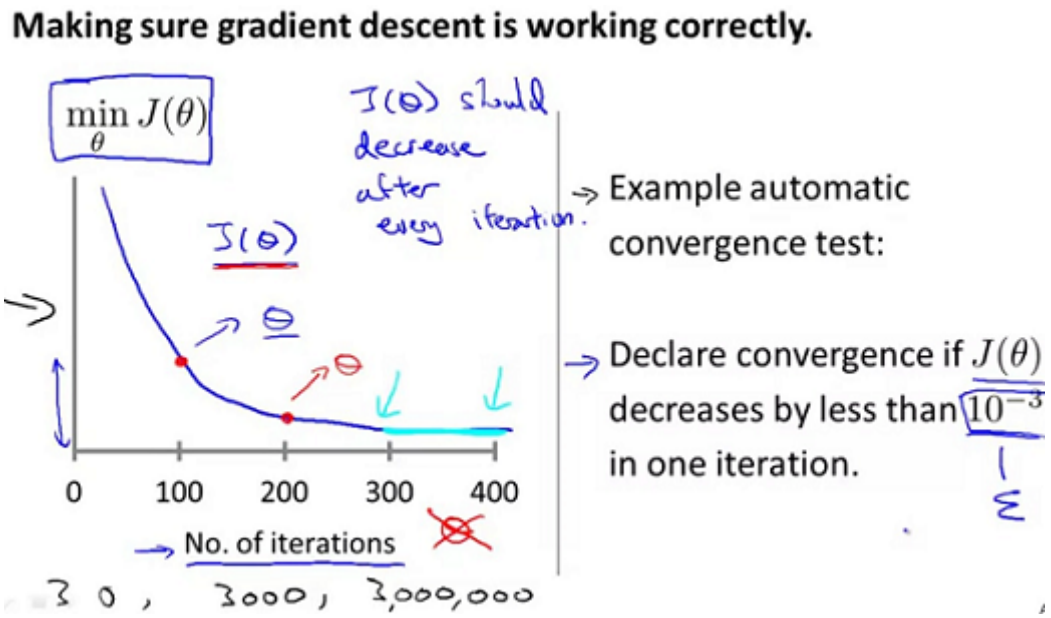
Gradient Descent in Practice II - Learning Rate

Gradient Descent update rule:

$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$

- "Debugging": How to make sure gradient descent is working correctly.
- How to choose learning rate α.

The job of gradient descent is to find the value of theta that hopefully minimizes the cost function $J(\theta)$. The x axis is a number of iterations of gradient descent and as gradient descent runs we hopefully get a plot that maybe looks like this:

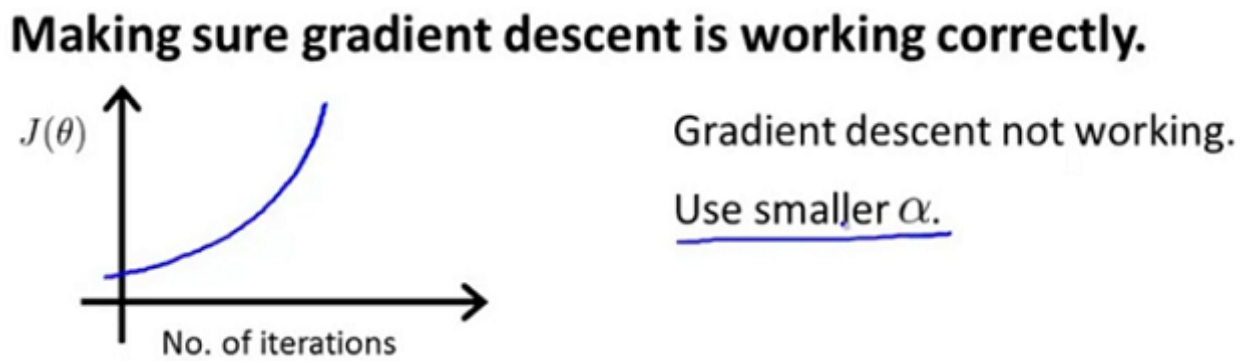


So what the plot it's showing is the value of our cost function after each iteration of gradient decent. And if gradient is working properly then $J(\theta)$ should decrease after every iteration. The number of iterations the gradient descent takes to converge for a physical application can vary a lot, so maybe for one application, gradient descent may converge after just 30 iterations.

For a different application, gradient descent may take 3,000 iterations, for another learning algorithm, it may take 3,000,000 iterations. It turns out to be very difficult to tell in advance how many iterations gradient descent needs to converge. It's also possible to come up with automatic convergence test, namely to have a algorithm try to tell if gradient descent has converged.

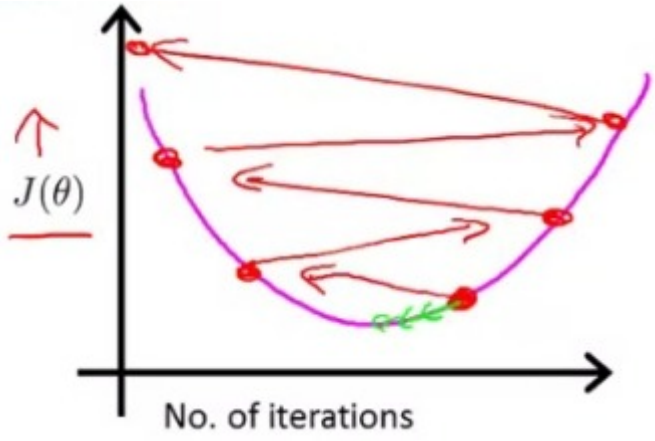
Example automatic convergence test:

- Declare convergence if $J(\theta)$ decreases by less than 10^{-3} in one iteration (It's pretty difficult what threshold choose).



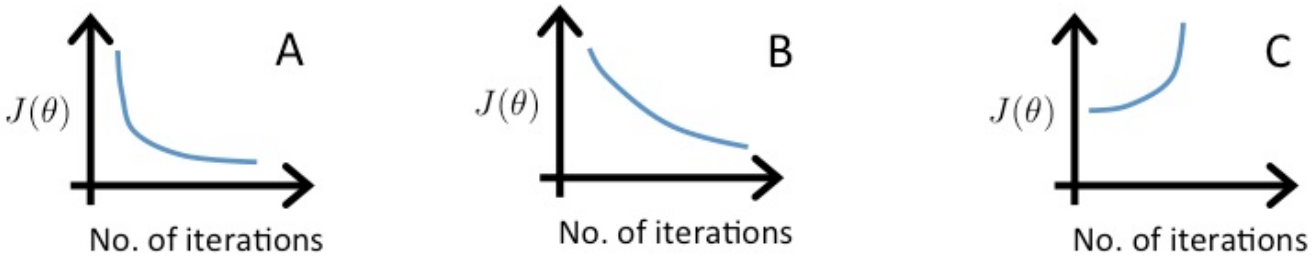
Concretely, if we plot $J(\theta)$ as a function of the number of iterations, then if we see a figure where $J(\theta)$ is actually increasing, then that gives a clear sign that gradient descent is not working.

If the learning rate is too big, then gradient descent can instead keep on overshooting the minimum.



So that actually end up getting worse, instead of getting to minimum values of the cost function $J(\theta)$. The fix is usually just to use a smaller value of alpha.

Video Question: Suppose a friend ran gradient descent three times, with $\alpha = 0.01$, $\alpha = 0.1$, $\alpha = 1$, and got the following three plots (labeled A, B, and C):



Which plots corresponds to which values of α?

- A is $\alpha = 0.01$, B is $\alpha = 0.1$, C is $\alpha = 1$.

> A is $\alpha = 0.1$, B is $\alpha = 0.01$, C is $\alpha = 1$.

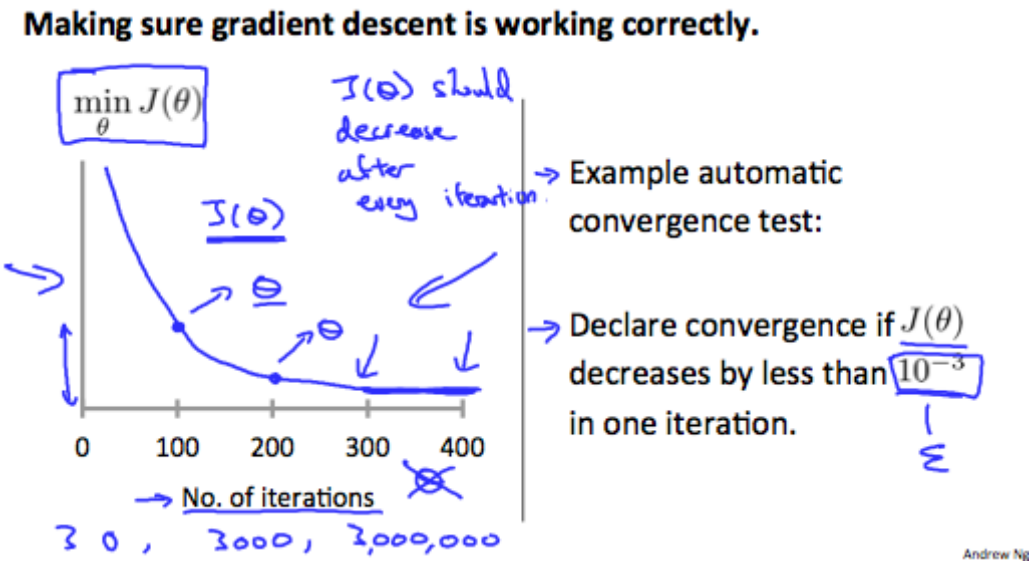
- A is $\alpha = 1$, B is $\alpha = 0.01$, C is $\alpha = 0.1$.
- A is $\alpha = 1$, B is $\alpha = 0.1$, C is $\alpha = 0.01$.

Explanation: In graph C, the cost function is increasing, so the learning rate is set too high. Both graphs A and B converge to an optimum of the cost function, but graph B does so very slowly, so its learning rate is set too low. Graph A lies between the two.

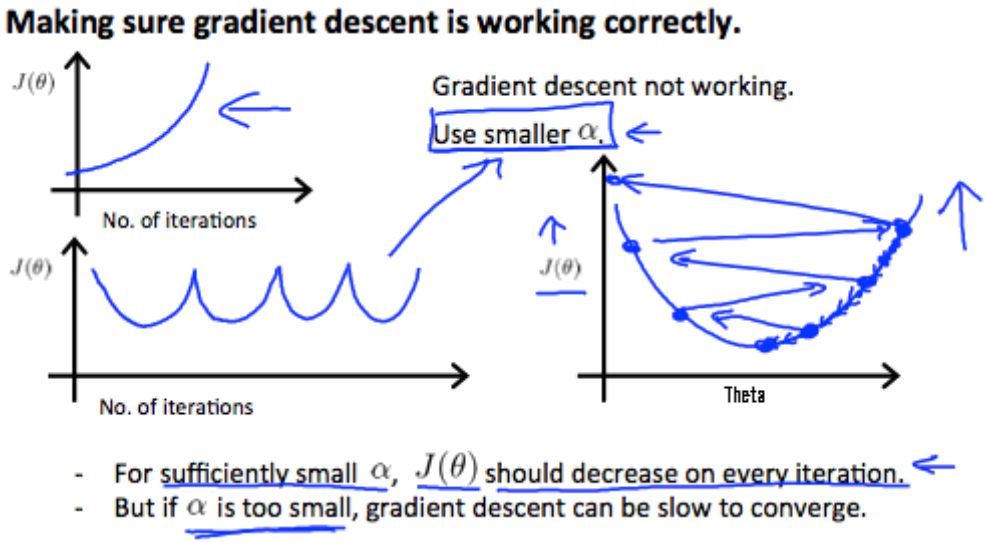
Summary

Debugging gradient descent. Make a plot with number of iterations on the x-axis. Now plot the cost function, $J(\theta)$ over the number of iterations of gradient descent. If $J(\theta)$ ever increases, then we probably need to decrease α.

Automatic convergence test. Declare convergence if $J(\theta)$ decreases by less than E in one iteration, where E is some small value such as 10^{-3}. However in practice it's difficult to choose this threshold value.



It has been proven that if learning rate α is sufficiently small, then $J(\theta)$ will decrease on every iteration.



To summarize:

- If α is too small: slow convergence.
- If α is too large: $J(\theta)$ may not decrease on every iteration; may not converge (This is the most common problem).

In order to debug all of these things, often plotting that $J(\theta)$ as a function of the number of iterations can help to figure put what's going on.

To choose α, try (choose in a scale factor of ten of each step up):
$\ldots, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, \ldots$