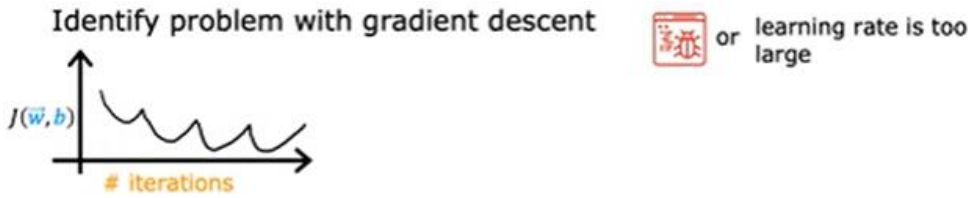
Choix du taux d'apprentissage

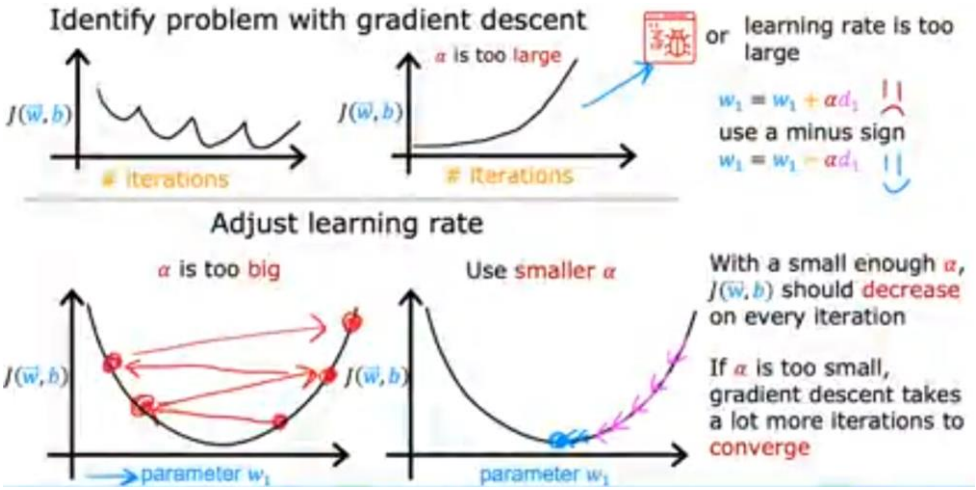
Your learning algorithm will run much better with an appropriate choice of learning rate. If it's too small, it will run very slowly and if it is too large, it may not even converge. Let's take a look at how you can choose a good learning rate for your model.



Concretely, if you plot the cost for a number of iterations and notice that the costs sometimes goes up and sometimes goes down, you should take that as a clear sign that gradient descent is not working properly. This could mean that there's a bug in the code. Or sometimes it could mean that your learning rate is too large. So here's an illustration of what might be happening.

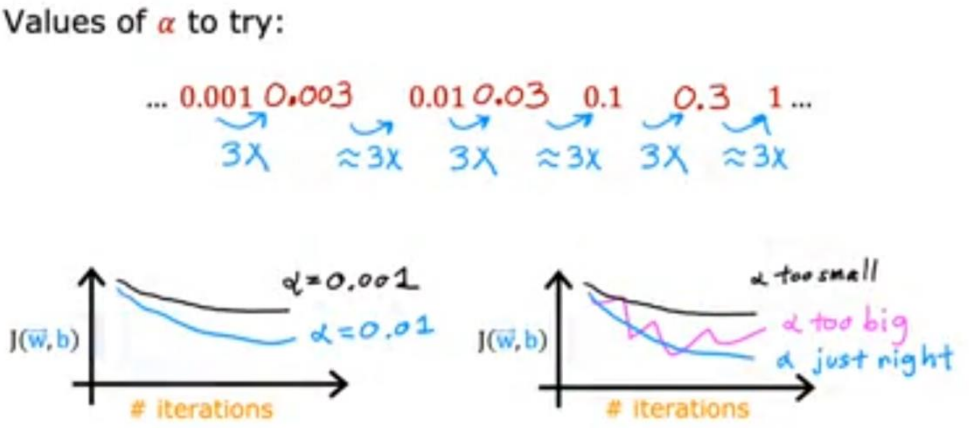
Here the vertical axis is a cost function J, and the horizontal axis represents a parameter like maybe w_1 and if the learning rate is too big, then if you start off here, your update step may overshoot the minimum and end up here, and in the next update step here, your gain overshooting so you end up here and so on. That's why the cost can sometimes go up instead of decreasing. To fix this, you can use a smaller learning rate. Then your updates may start here and go down a little bit and down a bit, and we'll hopefully consistently decrease until it reaches the global minimum.

Sometimes you may see that the cost consistently increases after each iteration, like this curve here. This is also likely due to a learning rate that is too large, and it could be addressed by choosing a smaller learning rate. But learning rates like this could also be a sign of a possible broken code. For example, if I wrote my code so that w_1 gets updated as w_1 plus Alpha times this derivative term, this could result in the cost consistently increasing at each iteration. This is because having the derivative term moves your cost J further from the global minimum instead of closer.



So remember, you want to use in minus sign, so the code should be updated w_1 updated by w_1 minus Alpha times the derivative term. One debugging tip for a correct implementation of gradient descent is that with a small enough learning rate, the cost function should decrease on every single iteration. So if gradient descent isn't working, one thing I often do and I hope you find this tip useful too, one thing I'll often do is just set Alpha to be a very small number and see if that causes the cost to decrease on every iteration.

If even with Alpha set to a very small number, J doesn't decrease on every single iteration, but instead sometimes increases, then that usually means there's a bug somewhere in the code. Note that setting Alpha to be really small is meant here as a debugging step and a very small value of Alpha is not going to be the most efficient choice for actually training your learning algorithm. One important trade-off is that if your learning rate is too small, then gradient descents can take a lot of iterations to converge.



So when I am running gradient descent, I will usually try a range of values for the learning rate Alpha. I may start by trying a learning rate of 0.001 and I may also try learning rate as 10 times as large say 0.01 and 0.1 and so on. For each choice of Alpha, you might run gradient descent just for a handful of iterations and plot the cost function J as a function of the number of iterations and after trying a few different values, you might then pick the value of Alpha that seems to decrease the learning rate rapidly, but also consistently.

In fact, what I actually do is try a range of values like this. After trying 0.001, I'll then increase the learning rate threefold to 0.003. After that, I'll try 0.01, which is again about three times as large as 0.003. So these are roughly trying out gradient descents with each value of Alpha being roughly three times bigger than the previous value.

What I'll do is try a range of values until I found the value of that's too small and then also make sure I've found a value that's too large. I'll slowly try to pick the largest possible learning rate, or just something slightly smaller than the largest reasonable value that I found. When I do that, it usually gives me a good learning rate for my model.