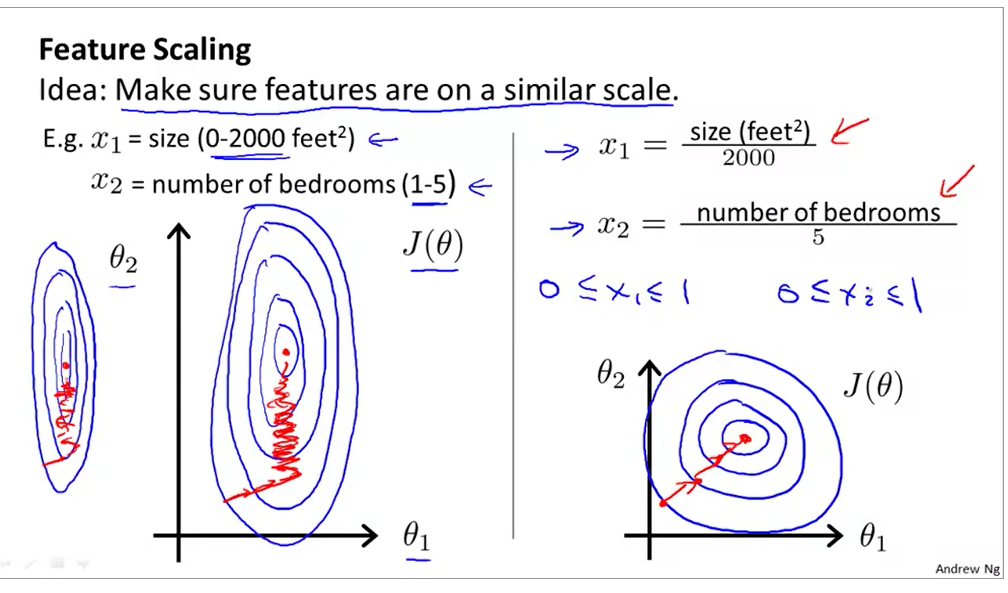


J(theta0,theta1,theta2) = theta0 \* x0 + theta1 \* x1 + theta2 \* x2

Lets ignore theta0 for now for simplification.



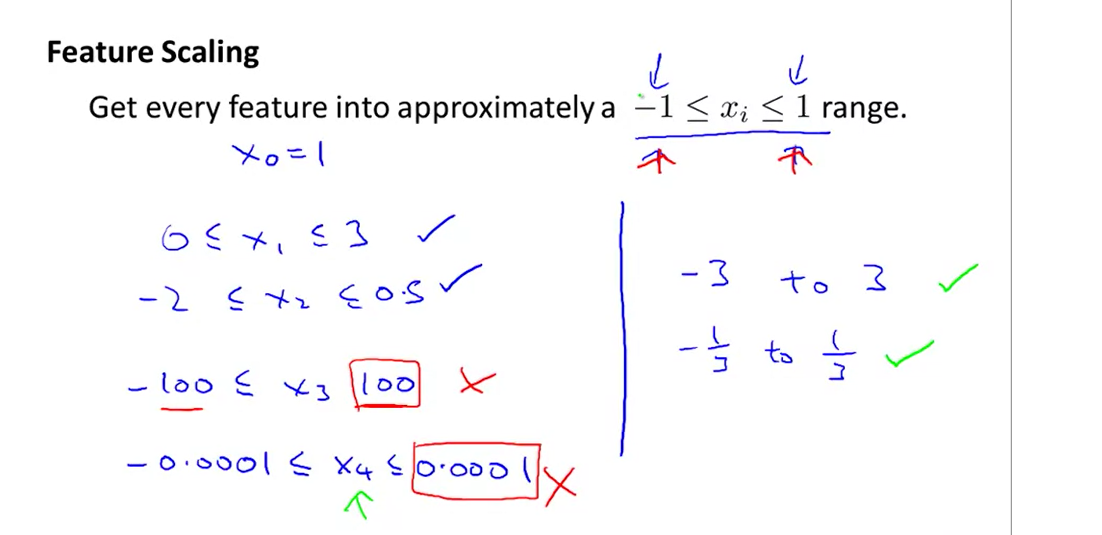
In the above slide, if x1 and x2 are on different scales, the cost function(we have ignored theta 0 for simplification) will be skewed elliptical or narrow elliptical. Gradient descent takes a longer time to find the local optima. In this case, X1 has much larger values than x2. So we do feature scaling and bring both the values on the same scale so that gradient descent will be must faster in reaching local optima.

We try to bring the features approximately in the range -1 to +1. X0 =1 is already in this range.

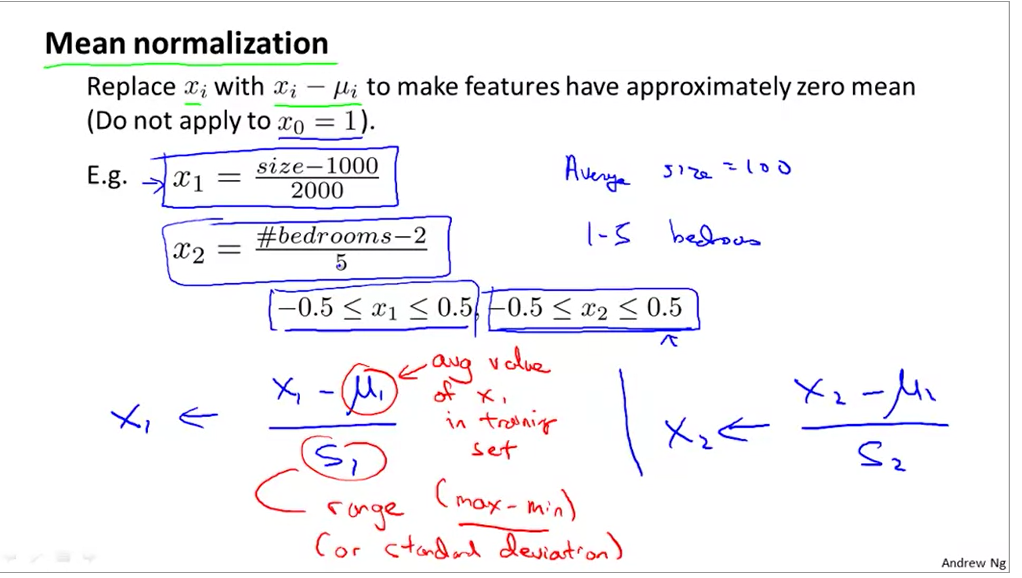
x1 that winds up being between zero and three, that's not a problem. If you end up having a different feature that winds being between -2 and + 0.5, again, this is close enough to minus one and plus one that, you know, that's fine, and that's fine.It's only if you have a different feature, say X 3

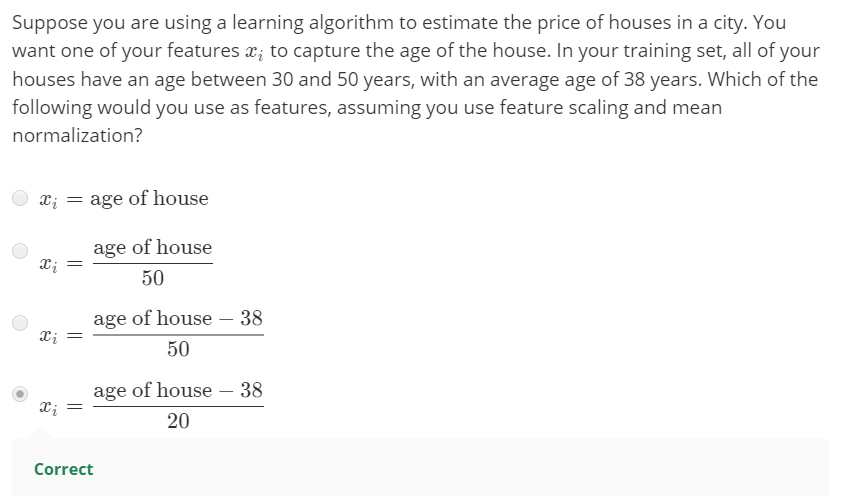
that is between, that ranges from -100 tp +100 , then, this is a very different values than minus 1 and plus 1. So, this might be a less well-skilled feature and similarly,if your features take on a very, very small range of values so if X 4 takes on values between minus 0.0001 and positive 0.0001, then again this takes on a much smaller range of values than the minus one to plus one range. And again I would consider this feature poorly scaled.

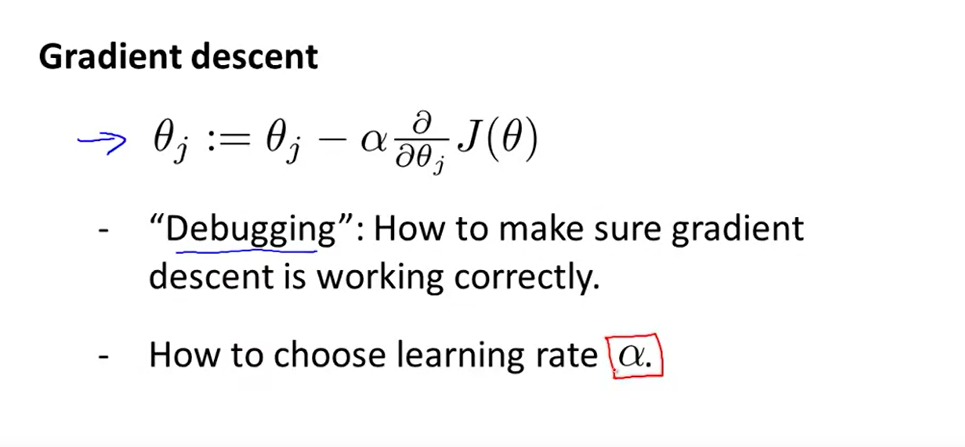
For feature scaling, -3 to +3 and -1/3 to +1/3 are valid as a thumb of rule.



For normalization, values are usually between -0.5 to 0.5, s1 and s2 can either be range or standard deviation.



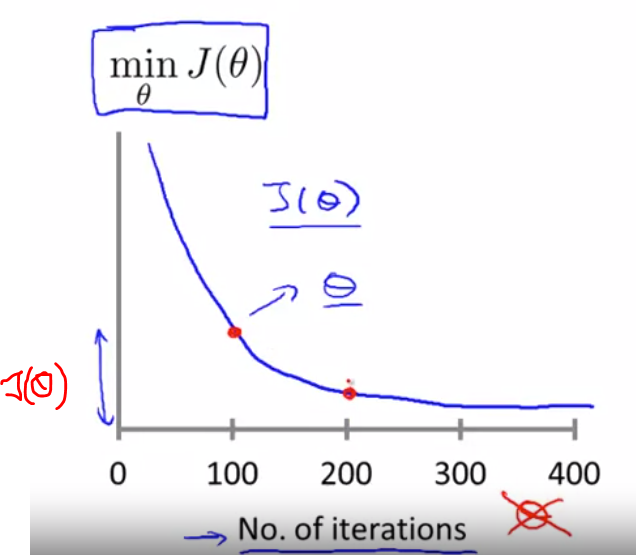




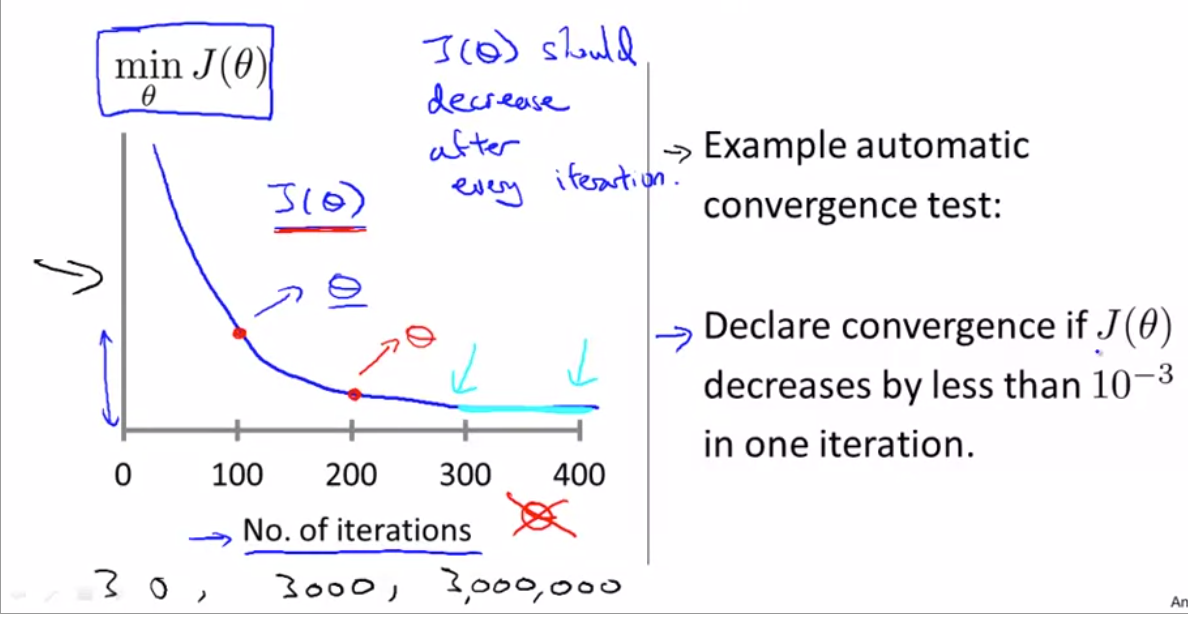
The job of gradient descent is to find the value of theta that hopefully minimizes the cost function J(theta). Plot the cost function J(theta) as gradient descent runs. So the x axis here is a number of iterations of gradient descent and as gradient descent runs you get a plot that may look like this.

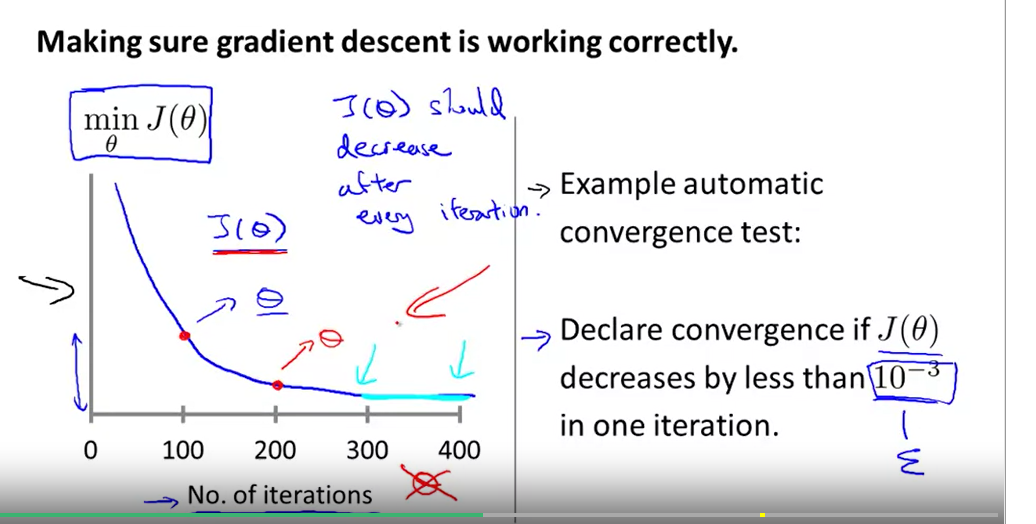
 Example, Run gradient descent for 100 iterations and we get some value of theta after 100 iterations. Evaluate the cost function J(theta) for the value of theta I get after 100 iterations,

and the vertical height is the value of J(theta).



If gradient descent is working properly, J(theta) should decrease after every iteration. After 400 iterations, it looks like gradient descent has more or less converged because your cost function isn't going down much more. So looking at this figure can also help you judge whether or not gradient descent has converged. For a different application, gradient descent may take 3,000 iterations, for another learning algorithm, it may take 3 million iterations. It turns out to be very difficult to tell in advance how many iterations gradient descent needs to converge. And is usually by plotting this sort of plot, plotting the cost function as we increase in number in iterations, is usually by looking at these plots, we can say if gradient descent has converged.



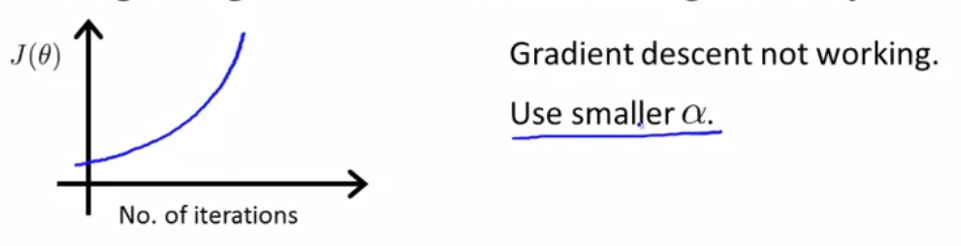


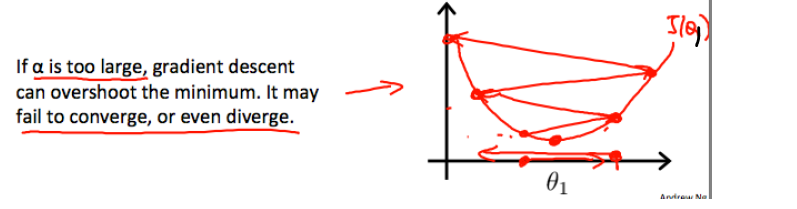
 Automatic convergence test - Such a test may declare convergence if your cost function J(theta)

decreases by less than some small value epsilon, some small value 10^-3. But usually choosing what this threshold is pretty difficult. And so in order to check your gradient descent's converge

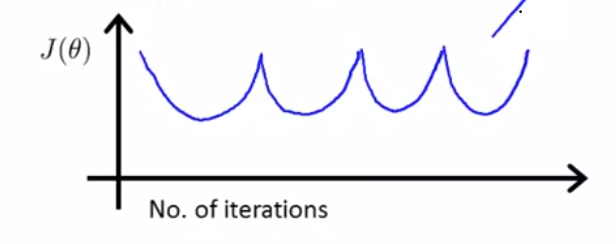
look at j(Theta) vs # of iterations plot rather than rely on an automatic convergence test.

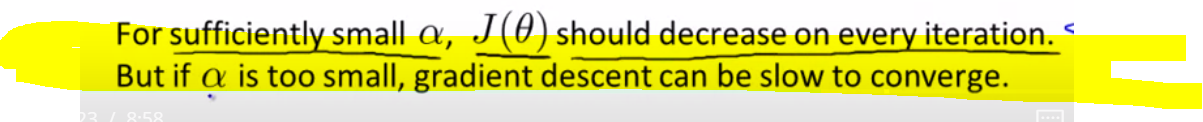
In the below graph, gradient descent is not working correctly. This happens usually when learning rate is too high that gradient descent overshoots the minimum.

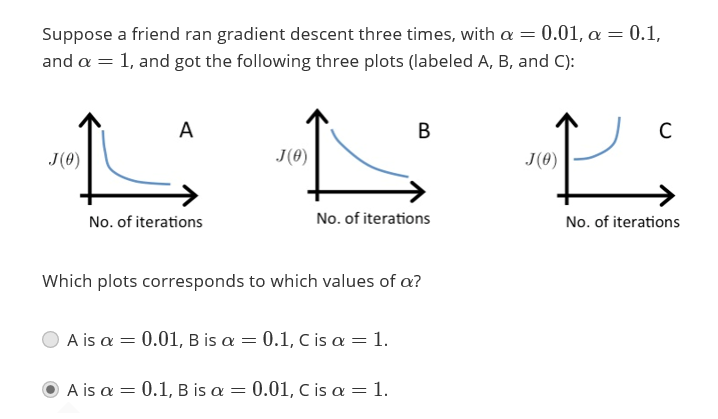


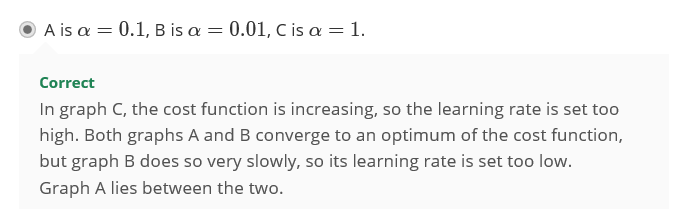


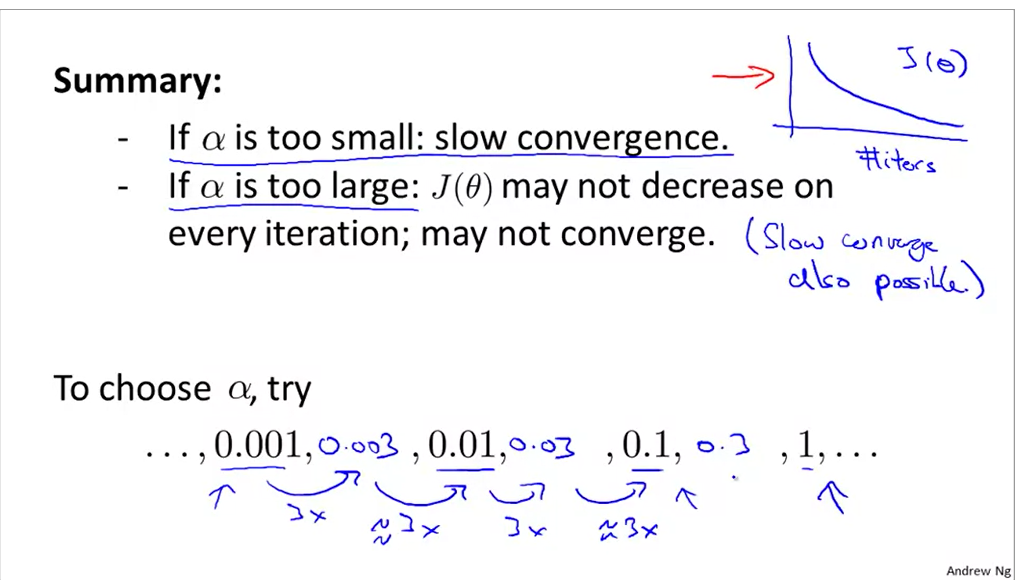
Similarly, the below graph is also an indication of gradient descent not working correctly. It is also caused due to large value of alpha , therefore recommendtion is to use small alpha





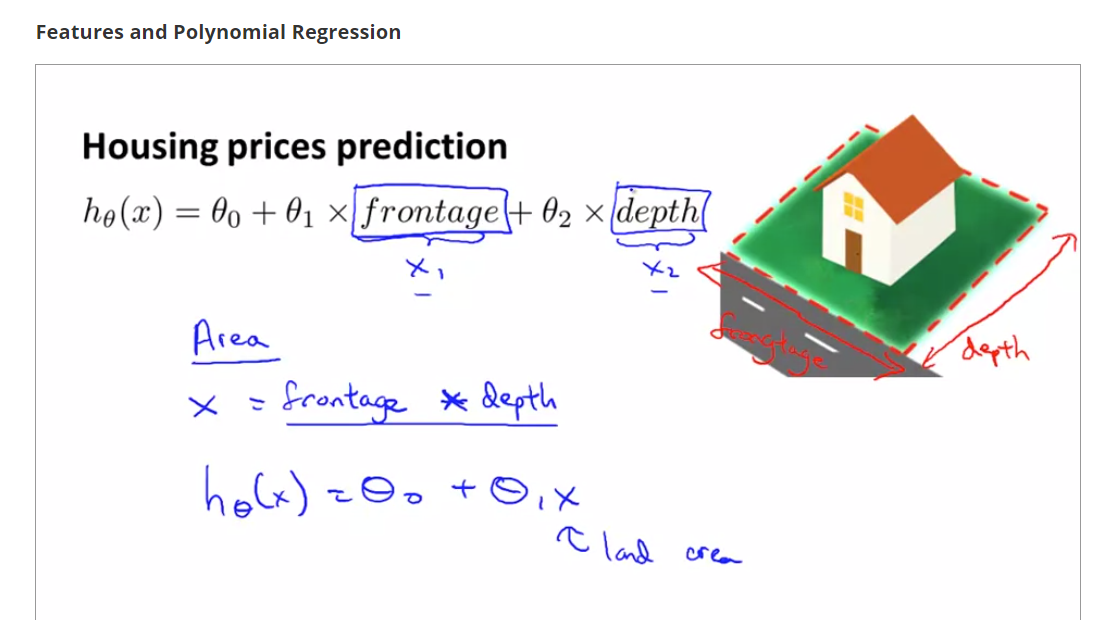






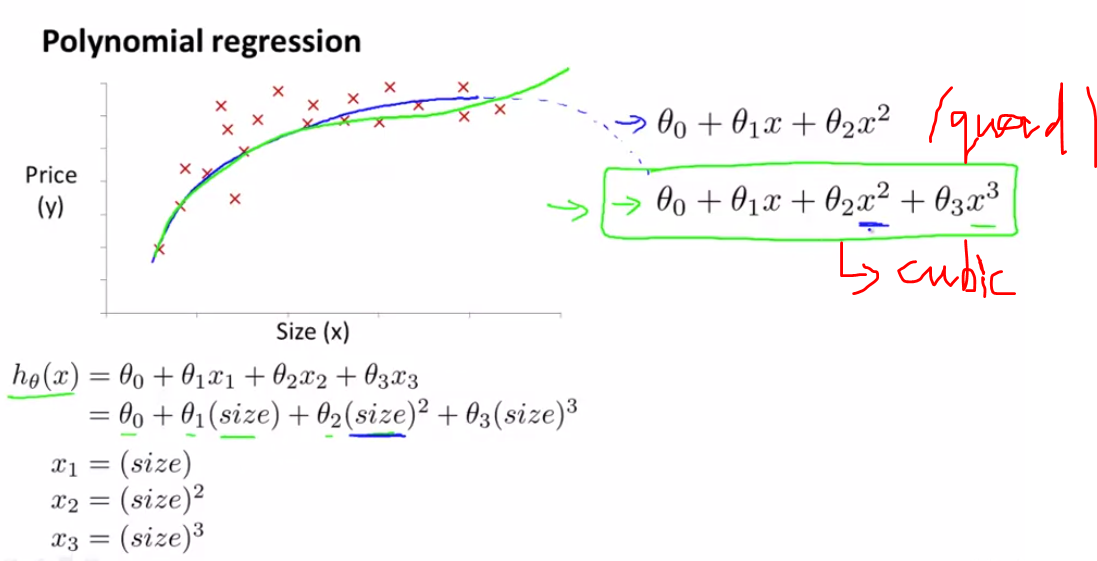
So try diff values of alpha starting with 0.001 and slowly increase it in 3 folds and try till 1

**FEATURES AND POLYNOMIAL REGRESSION**

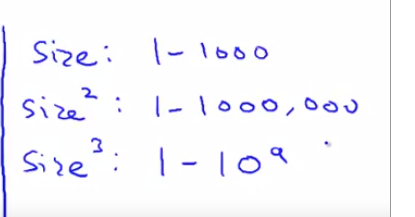


With available features frontage and depth, we can build new features such as area, and that gives us better model.

Let's say you have a housing price data set that looks like this. We start with quadratic model but no pursue that as quad model tends to droop, and we know as size increases, prices don’t go down. So we try cubic model highlighted in green box)



Feature scaling for cubic features is below

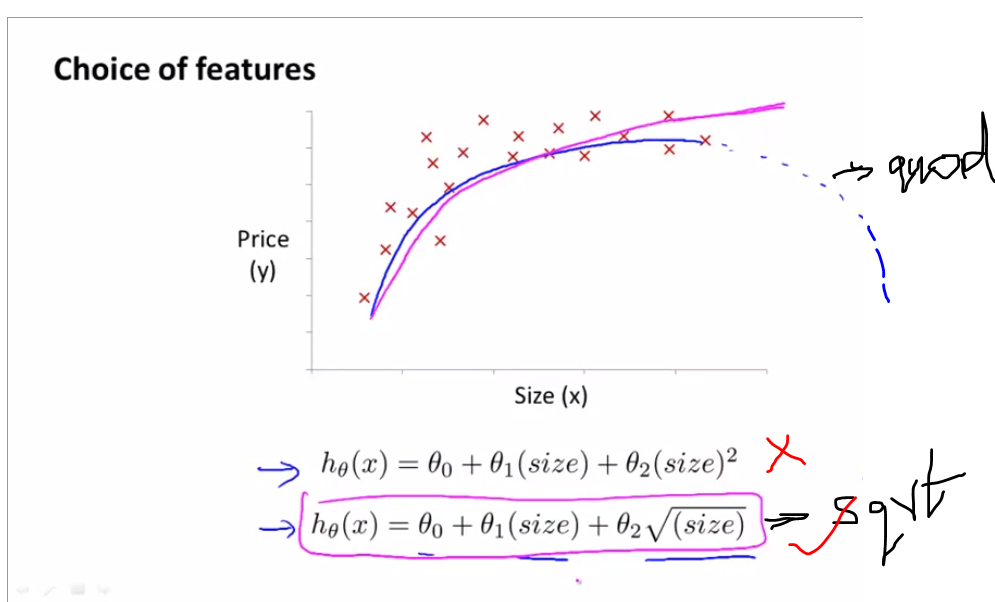


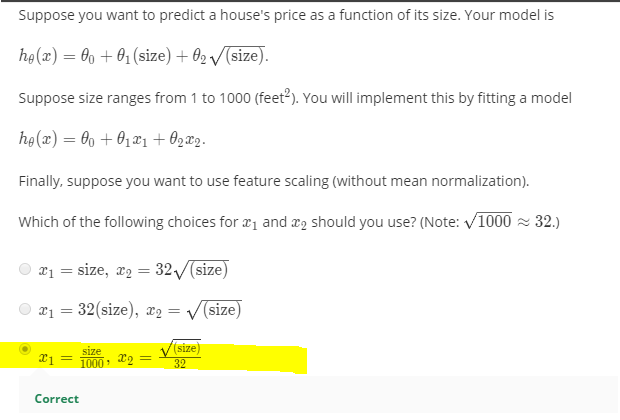
Feature scaling for x1 = size/1000

For x2 = size^2/e+6

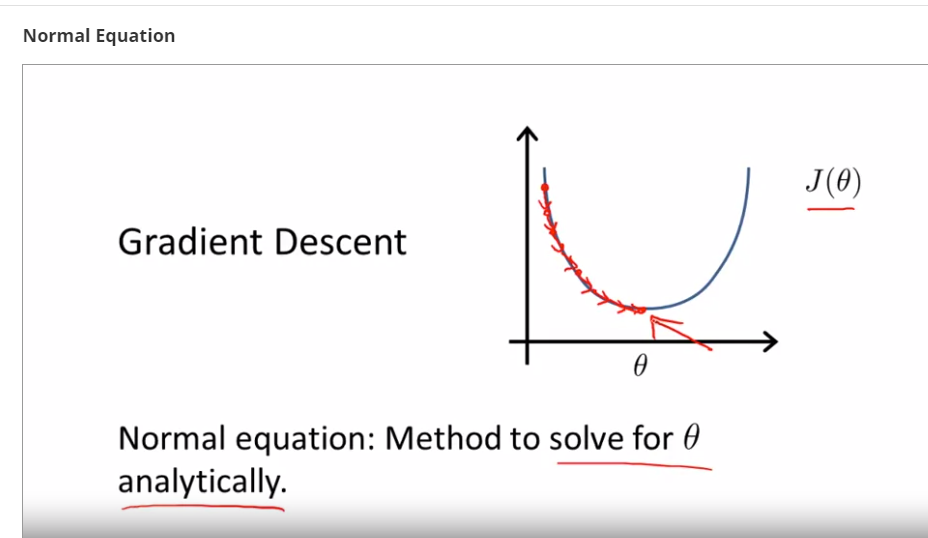
For x3 = size^3/e+9

In the belo example, quadratic model may not be best for house data as prices tend to go down with increase in size. Besides cubic model, other models such as sqrt can be used





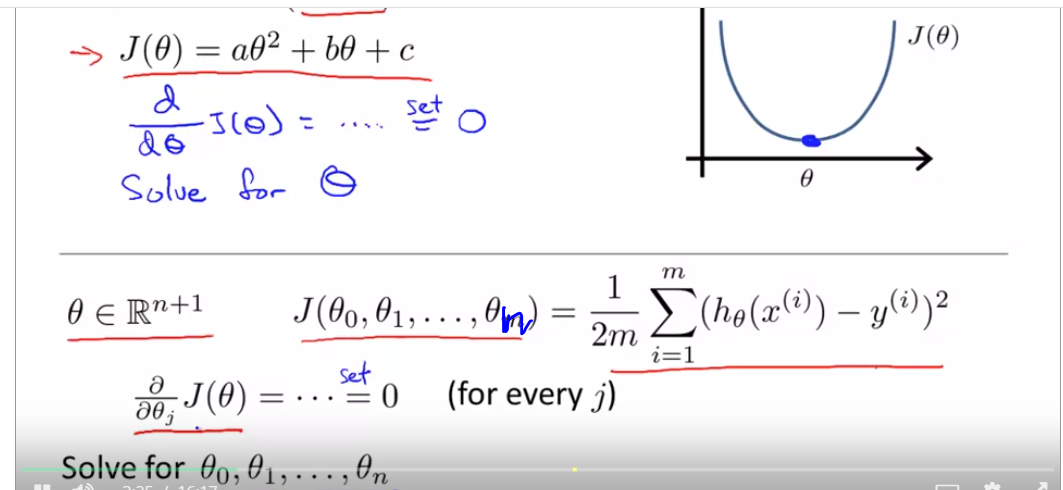
**Normal Equation**

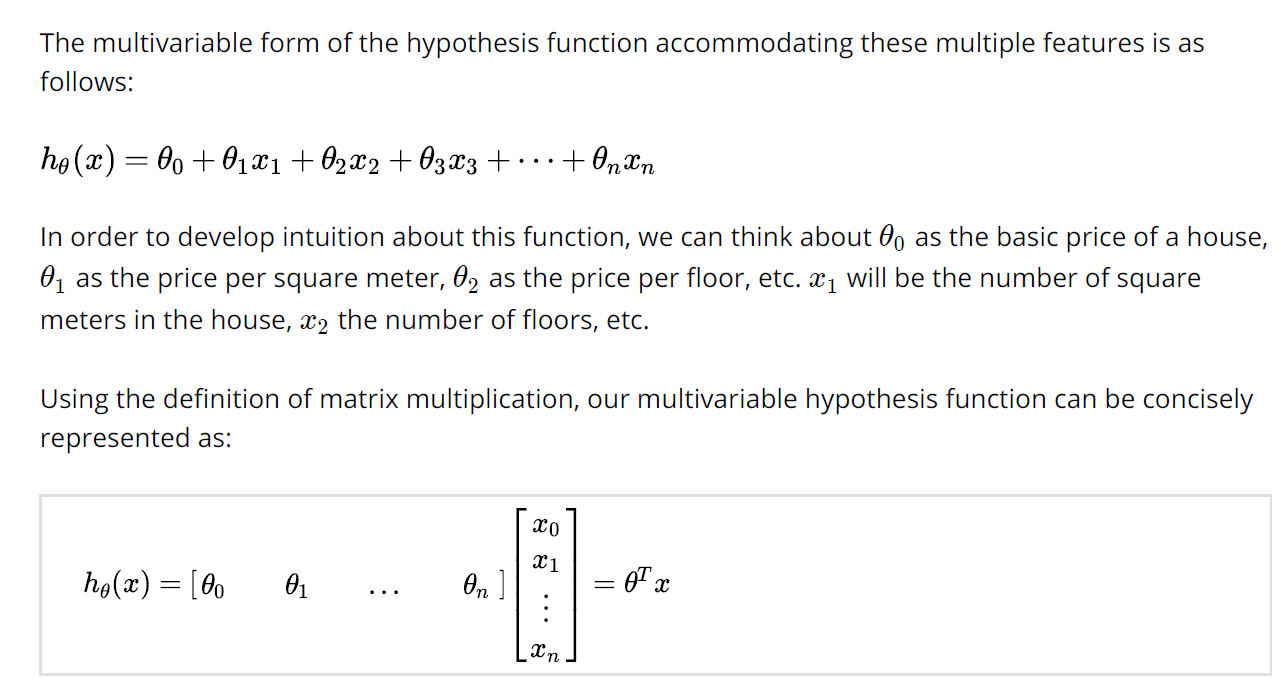


When theta is just a real number , in order to minimize J(theta), take derivative of J(theta), equate it to 0 and solve for theta

But when theta is R^(n+1) dimensional vector

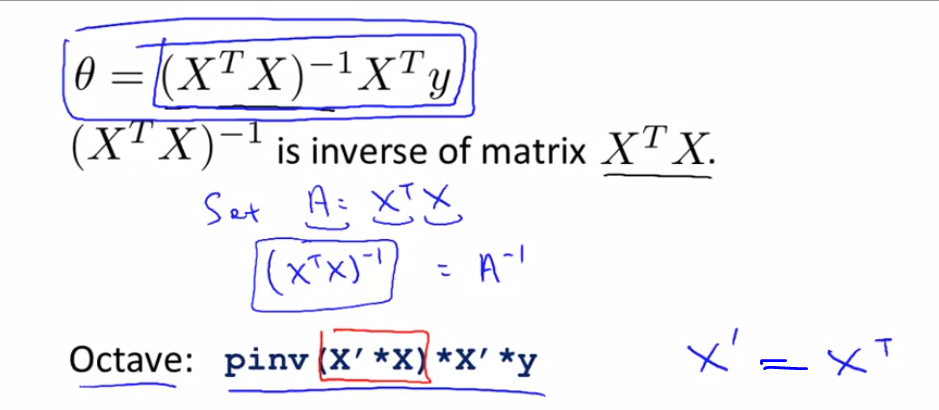
Theta = [theta0 theta1 theta2…..thetan] ~ R n+1 dimentional vector









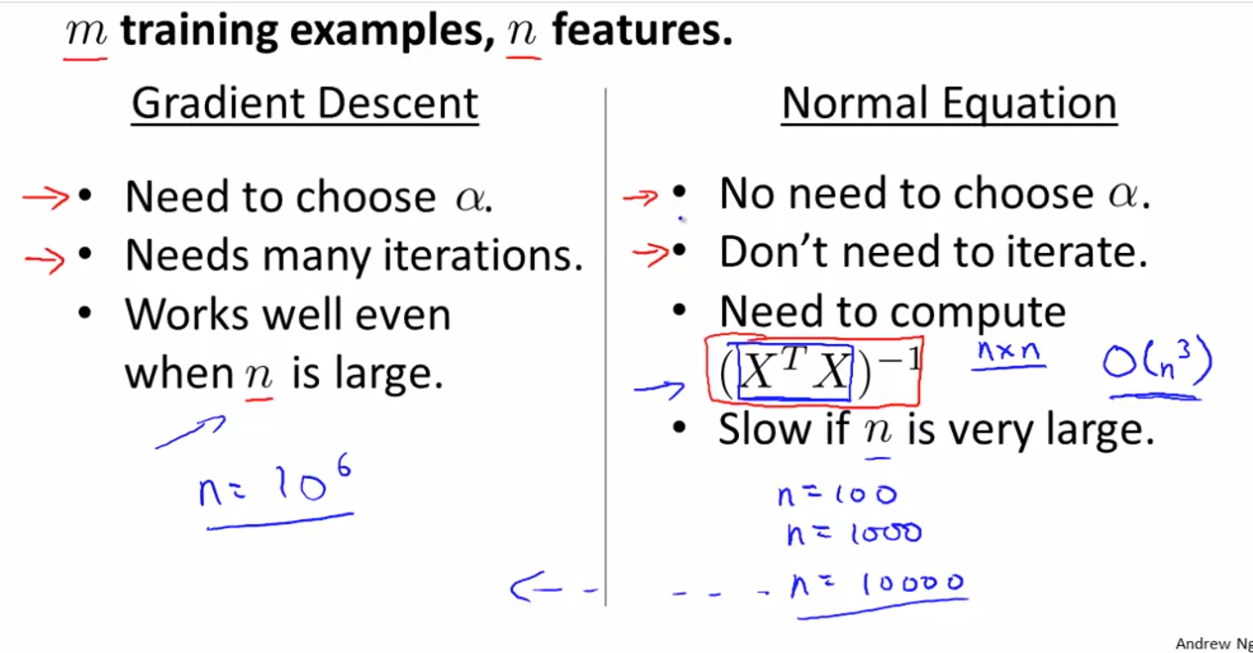


For normal equation method, feature scaling is not required

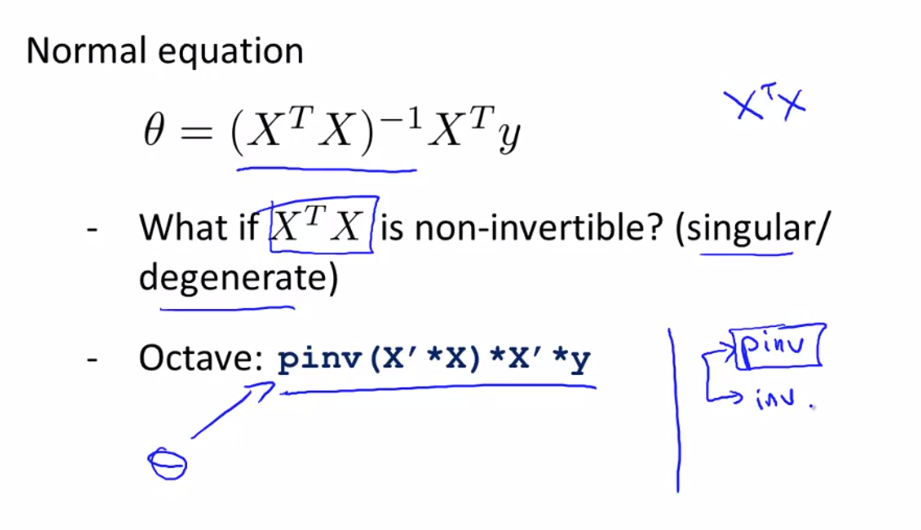
T(X) \* X ia a n \* n dimensional vector and inverting this vector could take O(n^3) computational time

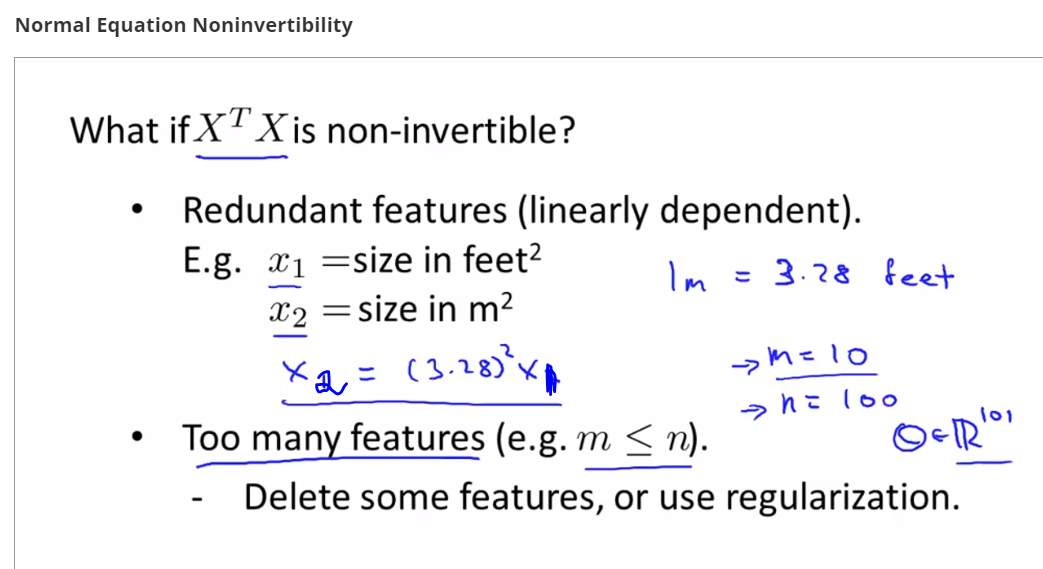
If n <= 1000, Normal eqn works better

But if n > 1000, gradient descent works best



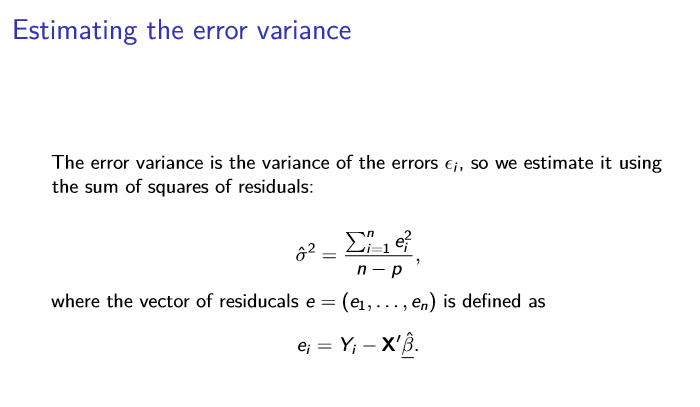
Sometimes t(X) \* X is non-invertible, pseudo inv (pinv) calculates theta even if t(X) \* X is non-invertible, where inv function doesn’t





--------------------------------------------------------------

**RECALL SECTION**



**SE for simple LR**

Beta dep only on that predictor



**SE for MULTIPLE LR**

When there are multiple predictors, the SEs depend not just on sums of squares of errors but also on the sums of cross-products of diﬀerent X variables (the oﬀ-diagonal terms of X0X). This means that the SE of a regression coeﬃcient estimate can change when a variable is added to a model. Whether or not it changes depends on the sum of squares of cross-products of the predictors and also on whether the estimate of the error variance changes. As mentioned before the coeﬃcient estimate for a predictor variable can change when we add another predictor to the model. Now we also know that the SE can change; furthermore, this can happen whether or not the coeﬃcient estimate itself changes.

Theta = Beta below

Muhat = transpose(theta) \* X

n 🡪 m training examples

y 🡪 m dim vector

X 🡪 m \* (n+1)

P 🡪 n+1 features

n – p = m – (n+1)

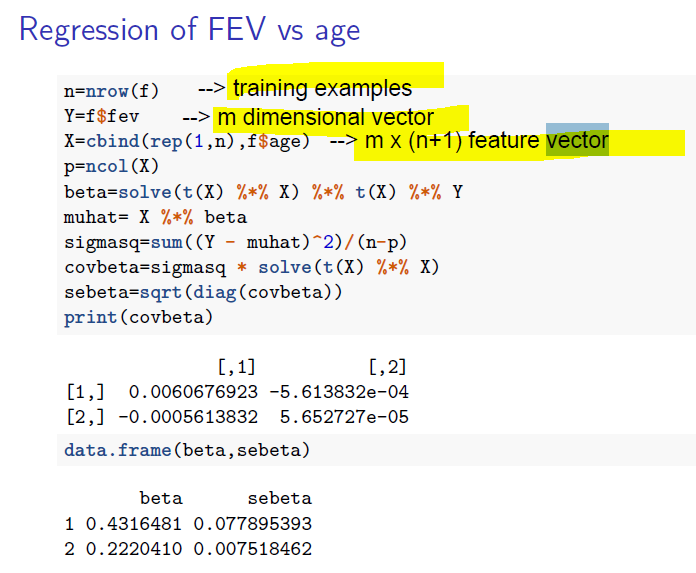
Beta = (n+1) \* 1

y = X \* Beta = m \* 1

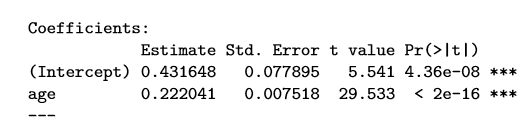
For a single training example, h(theta) or y = Theta’ \* x

For matrix of training examples, h (a vector of all the hypothesis values for the entire training set) = X \* theta

Muhat = X^t \* Beta (in the below code, X = t(x))



We also get the above results using summary(lm(fev ~ age, data=f))



Df = n-p and T statis

p = n+1 = 2

n = m

n – p = m – (n+1)

df = n-p =m – (p+1) = m -2

