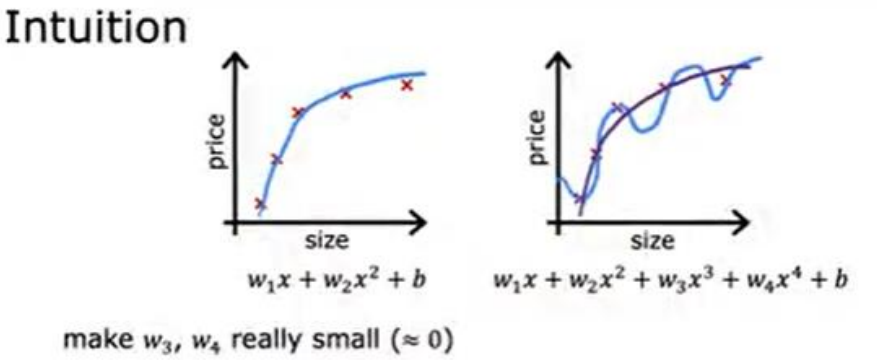
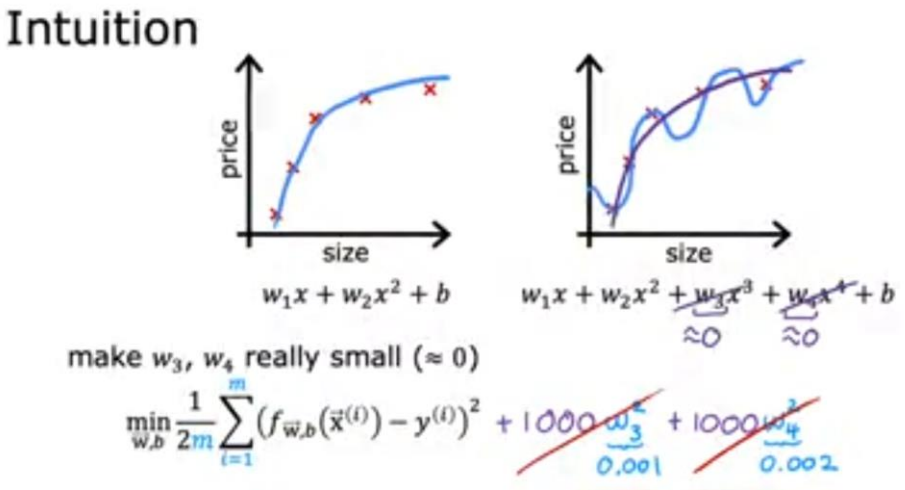
Fonction de coût avec régularisation

In the last video we saw that regularization tries to make the parental values W1 through WN small to reduce overfitting. In this video, we'll build on that intuition and developed a modified cost function for your learning algorithm that can use to actually apply regularization. Let's jump in, recall this example from the previous video in which we saw that if you fit a quadratic function to this data, it gives a pretty good fit. But if you fit a very high order polynomial, you end up with a curve that over fits the data. But now consider the following, suppose that you had a way to make the parameters W3 and W4 really, really small. Say close to 0. Here's what I mean.



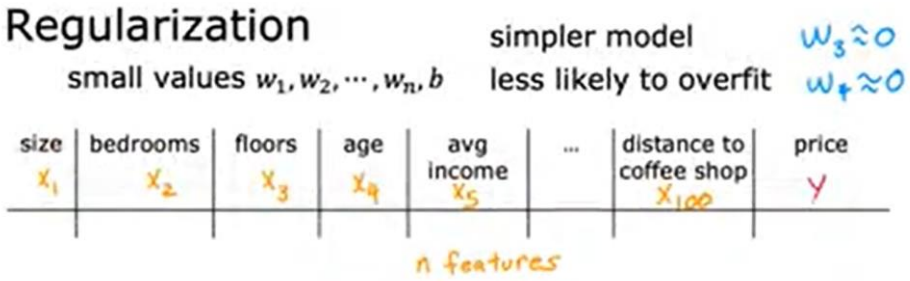
Let's say instead of minimizing this objective function, this is a cost function for linear regression. Let's say you were to modify the cost function and add to it 1000 times W3 squared plus 1000 times W4 squared. And here I'm just choosing 1000 because it's a big number but any other really large number would be okay. So with this modified cost function, you could in fact be penalizing the model if W3 and W4 are large. Because if you want to minimize this function, the only way to make this new cost function small is if W3 and W4 are both small, right? Because otherwise this 1000 times W3 squared and 1000 times W4 square terms are going to be really, really big.

So when you minimize this function, you're going to end up with W3 close to 0 and W4 close to 0. So we're effectively nearly canceling out the effects of the features execute and extra power of 4 and getting rid of these two terms over here. And if we do that, then we end up with a fit to the data that's much closer to the quadratic function, including maybe just tiny contributions from the features x cubed and extra 4. And this is good because it's a much better fit to the data compared to if all the parameters could be large and you end up with this weekly quadratic function more generally, here's the idea behind regularization.



The idea is that if there are smaller values for the parameters, then that's a bit like having a simpler model. Maybe one with fewer features, which is therefore less prone to overfitting. On the last slide we penalize or we say we regularized only W3 and W4. But more generally, the way that regularization tends to be implemented is if you have a lot of features, say a 100 features, you may not know which are the most important features and which ones to penalize.

So the way regularization is typically implemented is to penalize all of the features or more precisely, you penalize all the WJ parameters and it's possible to show that this will usually result in fitting a smoother simpler, less weekly function that's less prone to overfitting.



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

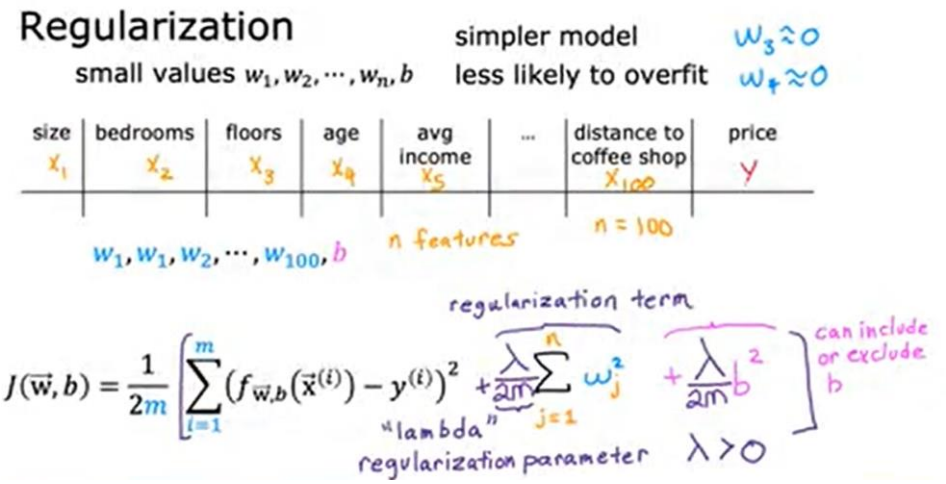
So for this example, if you have data with 100 features for each house, it may be hard to pick an advance which features to include and which ones to exclude. So let's build a model that uses all 100 features. So you have these 100 parameters W1 through W100, as well as 100 and first parameter B. Because we don't know which of these parameters are going to be the important ones.

Let's penalize all of them a bit and shrink all of them by adding this new term lambda times the sum from J equals 1 through n where n is 100. The number of features of wj squared. This value lambda here is the Greek alphabet lambda and it's also called a regularization parameter. So similar to picking a learning rate alpha, you now also have to choose a number for lambda.

A couple of things I would like to point out by convention, instead of using lambda times the sum of wj squared. We also divide lambda by 2m so that both the 1st and 2nd terms here are scaled by 1 over 2m. It turns out that by scaling both terms the same way it becomes a little bit easier to choose a good value for lambda.

And in particular you find that even if your training set size growth, say you find more training examples. So m the training set size is now bigger. The same value of lambda that you've picked previously is now also more likely to continue to work if you have this extra scaling by 2m. Also by the way, by convention we're not going to penalize the parameter b for being large. In practice, it makes very little difference whether you do or not.

And some machine learning engineers and actually some learning algorithm implementations will also include lambda over 2m times the b squared term. But this makes very little difference in practice and the more common convention which was used in this course is to regularize only the parameters w rather than the parameter b.

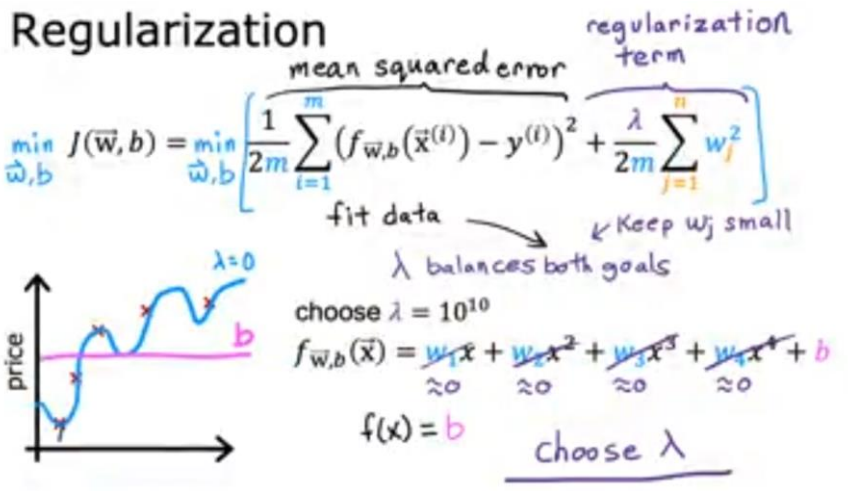


So to summarize in this modified cost function, we want to minimize the original cost, which is the mean squared error cost plus additionally, the second term which is called the regularization term. And so this new cost function trades off two goals that you might have. Trying to minimize this first term encourages the algorithm to fit the training data well by minimizing the squared differences of the predictions and the actual values.

And try to minimize the second term. The algorithm also tries to keep the parameters wj small, which will tend to reduce overfitting. The value of lambda that you choose, specifies the relative importance or the relative trade off or how you balance between these two goals.

So to summarize in this modified cost function, we want to minimize the original cost, which is the mean squared error cost plus additionally, the second term which is called the regularization term. And so this new cost function trades off two goals that you might have. Trying to minimize this first term encourages the algorithm to fit the training data well by minimizing the squared differences of the predictions and the actual values. And try to minimize the second term. The algorithm also tries to keep the parameters wj small, which will tend to reduce overfitting. The value of lambda that you choose, specifies the relative importance or the relative trade off or how you balance between these two goals.

If you said lambda to be a really, really, really large number, say lambda equals 10 to the power of 10, then you're placing a very heavy weight on this regularization term on the right. And the only way to minimize this is to be sure that all the values of w are pretty much very close to 0. So if lambda is very, very large, the learning algorithm will choose W1, W2, W3 and W4 to be extremely close to 0 and thus F of X is basically equal to b and so the learning algorithm fits a horizontal straight line and under fits.



To recap if lambda is 0 this model will over fit If lambda is enormous like 10 to the power of 10. This model will under fit. And so what you want is some value of lambda that is in between that more appropriately balances these first and second terms of trading off, minimizing the mean squared error and keeping the parameters small. And when the value of lambda is not too small and not too large, but just right, then hopefully you end up able to fit a 4th order polynomial, keeping all of these features, but with a function that looks like this.

So that's how regularization works. When we talk about model selection, later into specialization will also see a variety of ways to choose good values for lambda.