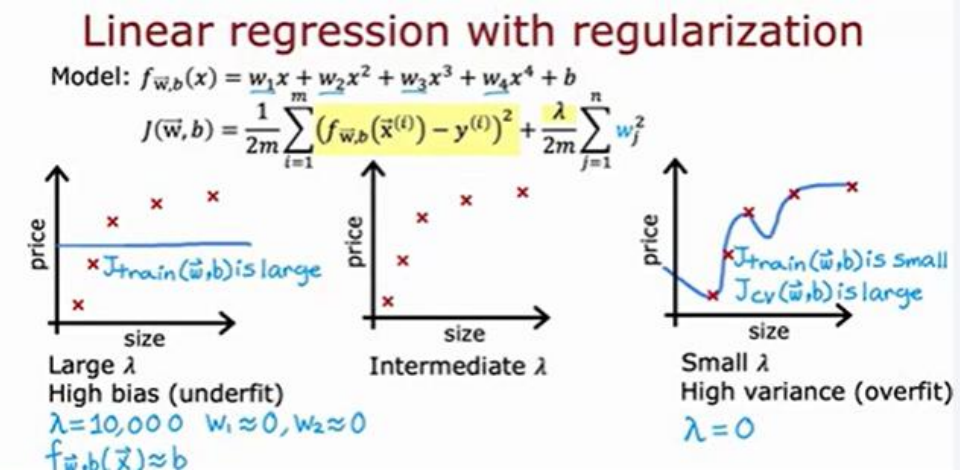
Regularization and bias/variance

variance of your learning algorithm, and therefore its overall performance. In this video, let's examine how regularization, specifically the choice of the regularization parameter lambda, affects the bias and variance, and thus the overall performance of the algorithm. This will be useful when you want to choose a good value for lambda for your algorithm's regularization parameter.

Let's take a look. In this example, I'll use a polynomial of order form, but we'll fit this model using regularization, where the value of lambda is the regularization parameter that controls the extent to which you trade off between keeping the parameters w small and fitting the training data correctly. Let's start with the example of setting lambda to a very large value (VALUE). Let's say lambda equals 10,000.

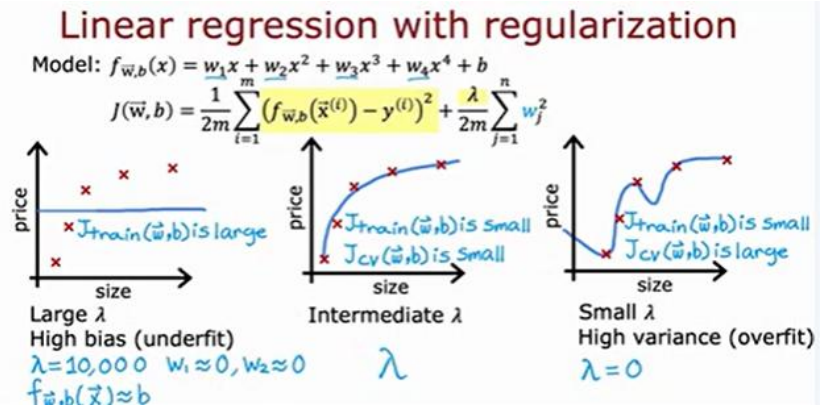
If you did that, you'd end up with a model that looks something like this. Because if lambda were very, very large, then the algorithm would be strongly incentivized to keep those parameters w very small. And so you end up with w1, w2, really all those parameters being very close to zero. And so the model ends up being that f of x is just approximately b, a constant value, which is why you end up with a model like this. And this model clearly has a high bias and it doesn't fit the training data because it doesn't even work well on the training set. And JTRAIN is large. Let's look at the other extreme.



Let's say you set lambda to a very small value (VALUE). So with a small value of lambda, in fact, let's go to the extreme by setting CA equal to zero. With this choice of lambda, there is no regularization, and so we are simply fitting a four-part polynomial without regularization, and you end up with this. And so on.

Thank you. We saw earlier that the data is overfitted, and so what we saw previously is that when you have a model like this, the J train is small, but JCV is much larger than the J train, or JCV is large, and this indicates that we have a large variance relative to the physical data.

And this would be the case if you had an intermediate value of lambda, not really far above 10,000, but not as small as zero, that you would hopefully get a model that looks like this, which is correct and fits the data well with a small J train and a small JCV. So, if you are trying to decide what a good Lambda value to use for the regularization parameter is, cross-validation also provides a way to do this.



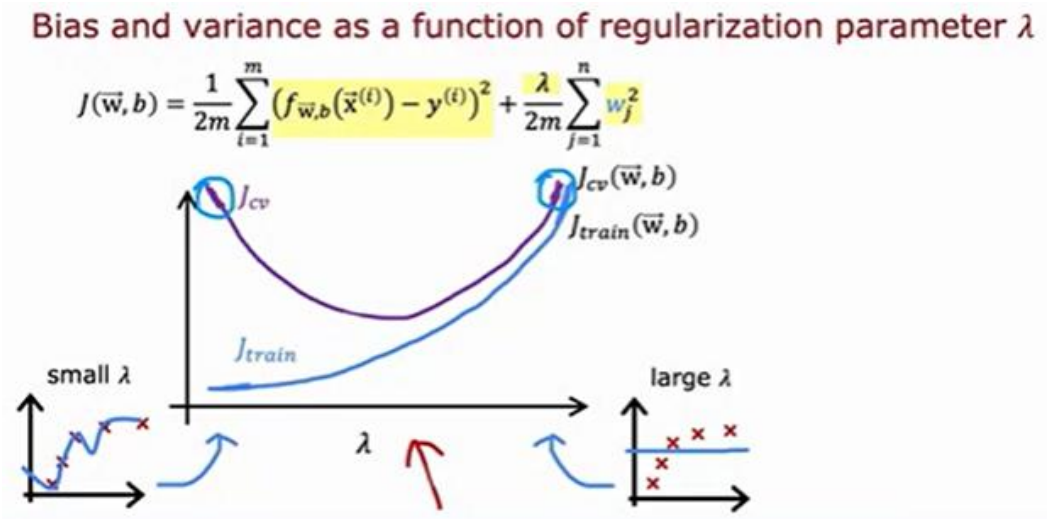
Let's see how we could achieve this. And just as a reminder, the problem we're addressing is that if you're fitting a four-part polynomial— that is, the model—and using regularization, how can you choose a good value for lambda? This would be a similar procedure to the one you saw for choosing the degree of polynomial D using cross-validation.

More precisely, let's say we're trying to fit a model using lambda equal to zero, and thus we would minimize the cost function using lambda equal to zero, and we would end up with some parameters w1, b1, and you can then calculate the cross-validation error jcv of w1, b1. Now, let's try a different value for lambda. Let's say you're trying lambda equal to 0.01.

Then, minimizing the cost function gives you a second set of parameters, W2, B2. And you can also see how well this works on the cross-validation set, and so on. Let's continue trying other Lambda values. In this example, I'll try doubling it so that Lambda equals 0.02, which would give you JCV of W3, B3, and so on.

Let's double the gain , and double the gain again. After doubling it a few times, you get a Lambda approximately equal to 10. This will give you the parameters W12, B12, and JCV, W12, B12. By trying a wide range of possible Lambda values, adjusting the parameters using these different regularization parameters, and then evaluating the performance on the cross-validation set, you can then try to choose the best value for the regularization parameter.

Specifically, if in this example you find that jcv of w5 b5 has the lowest value of all these different cross-validation errors, you can then decide to choose this value for lambda and therefore use w5 b5 as chosen parameters and finally if you wish to report and an estimate of the generalization error.



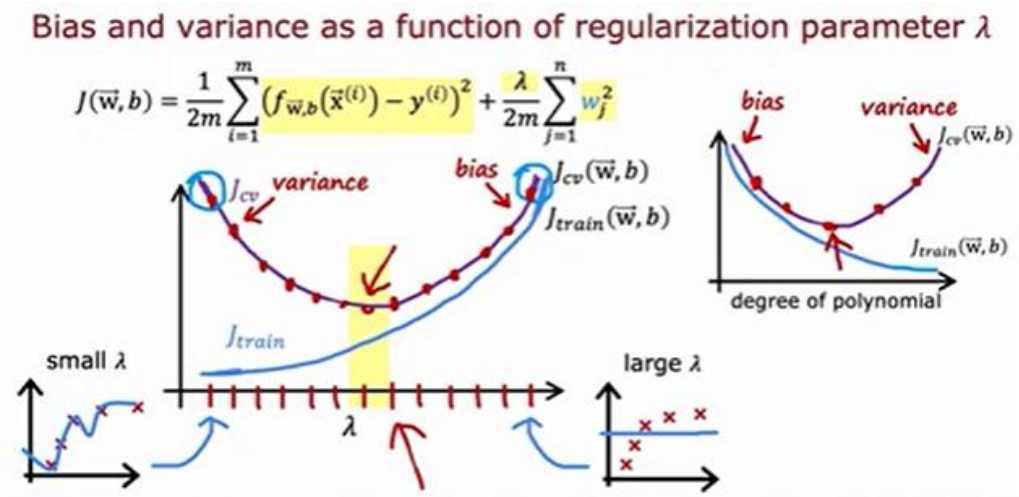
You would then report the tested error, Jtest of W5, B5. To further refine our intuition about what this algorithm does, let's examine how the training error and cross-validation error vary depending on the lambda parameter. So, in this figure, I've changed the AI axis again.

Notice that the x-axis here is labeled with the value of the regularization parameter lambda, and if we look at the extreme of lambda equal to zero here on the left, this corresponds to not using regularization. This is where we ended up with this very wavy curve if lambda was small or even zero.

In this case, we have a high-variance model, and therefore JTRAIN will be small and JCV will be large because it performs very well on the training data but performs much less well on the cross-validation data. This extreme right-hand side, with very large lambda values—let's say lambda equals 10,000—ends up fitting a model that looks like this. This therefore has a high bias; it underfits the data, and it turns out that JTRAIN will be high, and JCV will also be high. And in fact, if you look at how Jtrain varies with lambda, you'll see that Jtrain will increase like this.

Because in the optimization cost function, the larger Lambda is, the more the algorithm tries to keep W squared small—that is, the more weight it gives to this regularization term, and therefore less attention is paid to performing well on the training set. This term on the left is JTRAIN, so the more it tries to keep the parameters small, the less well it does to minimize the training error.

That's why, as Lambda increases, the AI training error tends to increase like this. And now, what about the cross-validation error? It turns out that the cross-validation error will look like this. Because we've seen that if Lambda is too small or too large, it doesn't perform well on the cross-validation set. It's either too large here on the left, or it's too small here on the right.



And there will be an intermediate value of Lambda that will make the algorithm work optimally. What cross-validation does is try a large number of different values of Lambda. This is what we saw on the last slide. Try Lambda equal to 0, Lambda equal to 0.01, Lambda equal to 0.02.

Try many different values of Lambda and evaluate the cross-validation error at several of these different points. Then, hopefully, choose a value that has a low cross-validation error, and that will hopefully provide a good model for your application.

If you compare this diagram to the one we had in the previous video where the horizontal axis was the degree of the polynomial, these two diagrams look somewhat similar—not mathematically and not formally, but they are somewhat like mirror images of each other. And that's because when you fit a degree of polynomial, the left part of this curve corresponds to an underfit and a high bias.

The right-hand side corresponded to overfitting and high variance. Whereas in this one, the high variance was on the left and the high bias was on the right. But that's why these two images are somewhat like mirror images of each other. But in both cases, cross-validation—evaluating different values—can help you choose a good t-value or a good lambda value.

So that's how the choice of the lambda regularization parameter affects the bias, variance, and overall performance of your algorithm. And you've also seen how you can use cross-validation to make a good choice for the lambda regularization parameter.

So far, we've talked about how a high training set error, a high JTRAIN, indicates high bias, and how a high cross-validation error, JCV, especially if it's much higher than JTRAIN, indicates a variance problem. But what do the words "high" or "much higher" actually mean?

Let's look at this in the next video, where we'll see how you can examine the JTrain and JCV numbers and judge whether they are high or low. It turns out that a further refinement of these ideas— establishing a baseline performance level—will allow your learning algorithm to much more easily assess these JTrain and JCV numbers and determine if they are high or lo