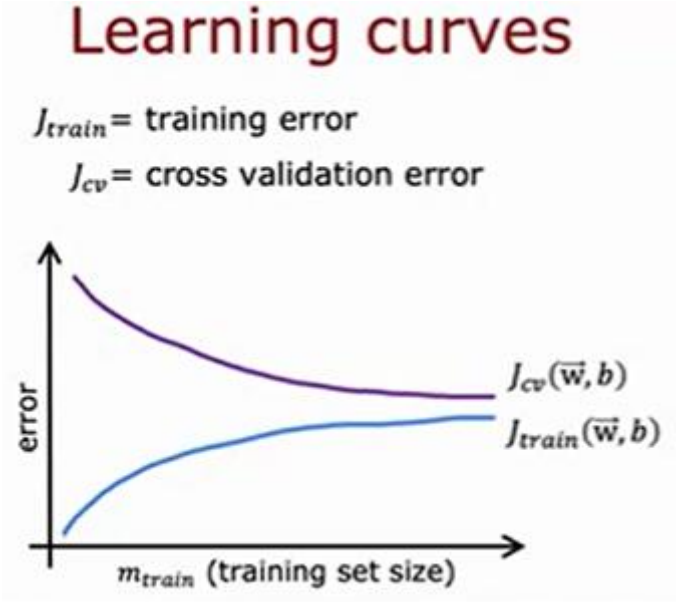
Learning curves

Learning curves are a way to help understand how your learning algorithm performs based on the amount of experience it has. By experience, I mean, for example, the number of training examples it has. Let's look at this. Let me plot the learning curves for a model that corresponds to a second-order quadratic polynomial function like this one, and I'm going to plot both JCV, the cross-validation error, and JTRAIN, the training error.

So, in this figure, the horizontal axis will be Mtrain, which is the size of the training set, or the number of examples the algorithm can learn from. And on the vertical axis, I'm going to plot the error. And by error, I mean either JCV or Jtrain. So let's start by plotting the cross-validation error.

It will look something like this. So, this is what the JCV of CV will look like. And perhaps unsurprisingly, as M trains, the size of the training set increases, and then you learn a better model, and therefore the cross-validation error decreases. Now, let's plot the J train of WB, of what the training error looks like as the size of the training set increases.

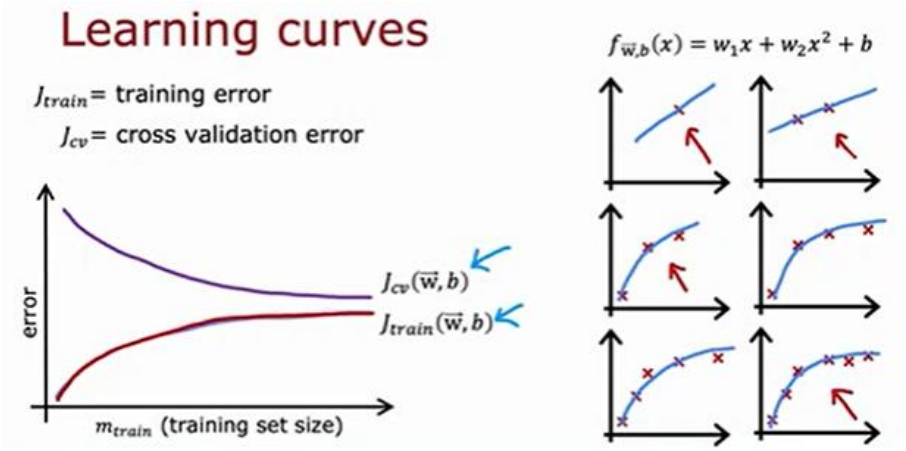
It turns out that the training error will actually look like this, that as the size of the training set increases, the training error actually increases. Let's see why this is the case. We'll start with an example where you only have one training example.



So, to summarize, when you have a very small number of training examples, such as one, two, or even three, it's relatively easy to achieve zero or very low training error. But when you have a larger training set, it becomes more difficult for a quadratic function to perfectly fit all the training examples.

Therefore, as the training set gets larger, the training error increases because it's harder to perfectly fit all the training examples. Note one more thing about these curves: the cross-validation error will generally be higher than the training error because you're fitting the parameters to the training set and therefore expect to do at least a little better, or when M is small, perhaps even much better, on the training set than on the confidence validation dataset.

Let's now look at how the learning curves of a high-bias algorithm will compare to those of a high-variance algorithm. Let's start with the case of high bias, or underfitting. Recall that an example of high bias would be fitting a linear function to a curve that looks like this.

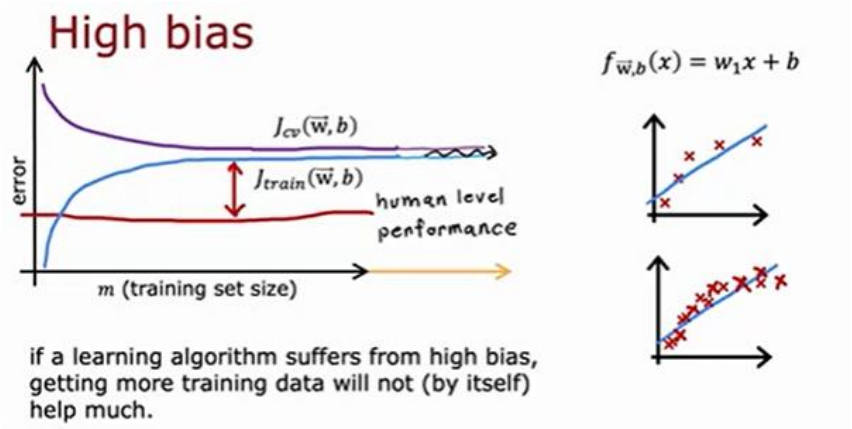


If you were to plot the training error, then the training error would increase. As expected. And in fact, this training error curve can start to flatten out, or we call it a plateau, which means to level off. After a while. This is because as you get more and more training examples, when you adjust the simple linear function, your model doesn't actually change much more. It's a straight line. And even if you get more and more examples, there isn't much more to change, which is why the average training error stabilizes after a while.

Similarly, your cross-validation error will also decrease and stabilize after a while, which is why JCV, again, is higher than JTRAIN, but JCV will tend to look like this. This is because beyond a certain point, even if you get more and more examples, little will change from the straight line you're following. It's simply too simple a model to scale to so much data, which is why both curves, JCV and JTRAIN, tend to flatten out after a while.

And if you had a measure of this baseline performance level, like human performance, then it would tend to be a lower value than your AI and CV. Thus, human-level performance might look like this, and there's a significant gap between the baseline performance level and JTRAIN, which was our indicator for this highly biased algorithm. That is, we could hope to do much better if only we could fit a more complex function than a simple straight line.

Now, an interesting thing about this graph is that you might ask, what do you think would happen if you had a much larger training set? So, what would it look like if we could increase M even further than the line on this graph and go further to the right as follows?



Well, you can imagine that if you were to extend these two curves to the right, they would both flatten out and probably remain flat that way. And no matter how far to the right you extend this graph, these two curves will never, you know, somehow, find a way to get down to that level of human performance or continue to be flat like that almost forever, regardless of the size of the training set.

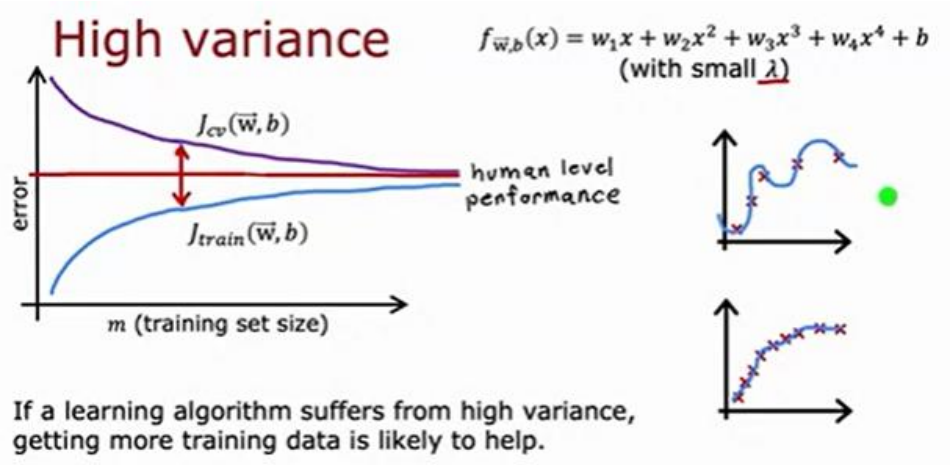
This leads us to this perhaps somewhat surprising conclusion: if a learning algorithm has a high bias, getting more training data won't help much on its own. And I know we tend to think that more data is good, but if your algorithm has a high bias, then if the only thing you do is throw more training data at it, that alone will never reduce the error rate that much.

And that's precisely why. No matter how many additional examples you add to this figure, the straight line you're adjusting won't improve significantly. That's why, before investing considerable effort in collecting more training data, it's worthwhile to check if your learning algorithm has a high bias.

Because if that's the case, you'll likely need to do more than just feed it more training data. Now let's look at what the learning curve looks like for a high-variance learning algorithm. You might remember that if you were to fit a four-part polynomial with a small lambda, for example, or even lambda equal to zero, you would get a curve that looks like this.

And while that fits the training data very well, it's not generalizable. Now let's look at what a learning curve might look like in this high-variance scenario . The JTrain will go up. As the size of the training set increases, you get a curve that looks like this, and the JCV will be much higher. So your cross-validation error is much greater than your training error.

And the fact that there's a huge gap here is what can tell you there's high variance. It performs much better on the training set than on your validation dataset. If you were to plot a baseline performance level, such as human-level performance, you might find that this is where J-train can sometimes still be lower than human-level performance.



Or perhaps human-level performance is a little lower than that. But when you overfit the training set, you might be able to fine-tune it so well that you get an unrealistic error, like zero error in this example, which is actually better than humans' ability to predict real estate prices or whatever application you're working on.

But again, the signal of high variance is whether JCV is much higher than JTRAIN. And when you have high variance, AI and UO are also important factors. Next, increasing the size of the training set could be of great help.

In particular, if we could extrapolate these curves to the right, increasing mtrain, then the training error would continue to increase, but then the cross-validation error would decrease and approach jtrain. Thus, in this scenario, it might be possible simply by increasing the size of the training set to reduce the cross-validation error and obtain even better performance from your algorithm.

This is unlike the high bias case where simply acquiring more training data won't actually improve the performance of your learning algorithm. So, to summarize, if a learning algorithm suffers from high variance, then acquiring more training data is likely beneficial. Because by extrapolating to the right of this curve, we can expect the JCV to continue to decrease.

And in this example, simply obtaining more training data allows the algorithm to move from this relatively high cross-validation error to a performance much closer to the human level. You can see that if you add many more training examples and continue fitting a four-part polynomial, you can then obtain a better four-part polynomial fit to this data than a simple, very wavy curve at the top.

So, if you're building a machine learning application, you can plot the learning curve if you want. Take subsets of AI CA UO training sets. Even if you have, say, 1,000 training examples, you could train a model on just 100 training examples and examine the training error and cross-validation error, then train a model on 200 examples, keeping 800 examples and not using them for now, and then plot JTRAIN and JCV, etc., and then repeat and plot what the learning curve looks like.

If you were to visualize it this way, it could show whether your learning curve looks more like a high-bias or high-variance curve. One of the drawbacks of plotting learning curves like this one—and something I've done—is that it's expensive to train so many different models using subsets of varying sizes from your training set. In practice, this isn't done often, but nevertheless, I find that having this mental image in my head of what the training set looks like sometimes helps me think about what I believe my learning algorithm is doing and whether it has high bias or high variance. So, I know we've talked a lot about bias and variance.