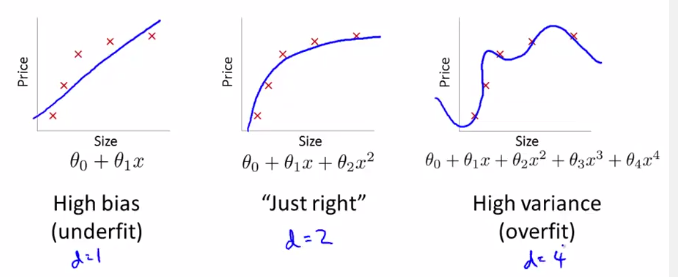
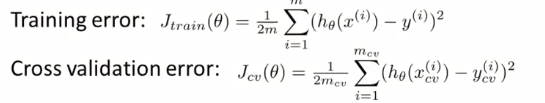
***Bias vs. Variance***

**I. DIAGNOSING BIAS VS. VARIANCE**

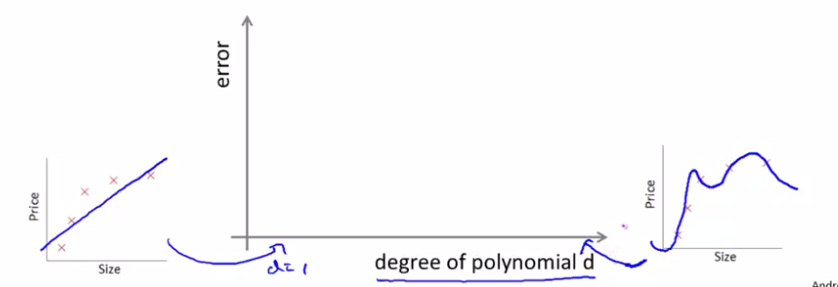
* If running a learning algorithm + it doesn't do as well as you were hoping, almost all the time it will be b/c you have either a **high bias problem** or a **high variance problem** (either an **underfitting** problem or an **overfitting** problem)
* In this case, it's very important to figure out which of these 2 problems it is, bias or variance, or a bit of both
* Knowing which of these 2 things is happening gives a very strong indicator for whether there are
* If you fit a too-simple hypothesis, it looks like a straight line through data that that underfits it.
* If you fit a too-complex hypothesis, it might fit a training set perfectly, but overfit new data
* We want a hypothesis of some intermediate level of complexity, maybe a 2 degree polynomial, not too low and not too high of a degree
* i.e. want a degree that gives you best generalization error.



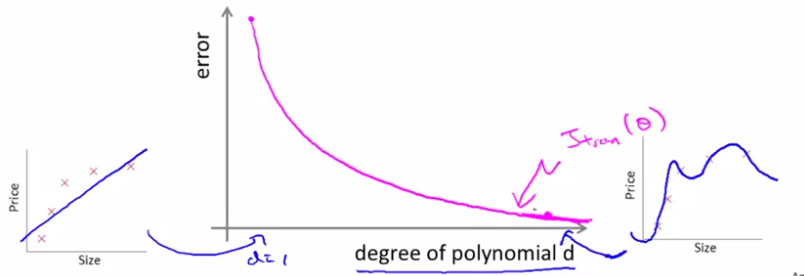
* Remember the **training error** and **cross validation error**



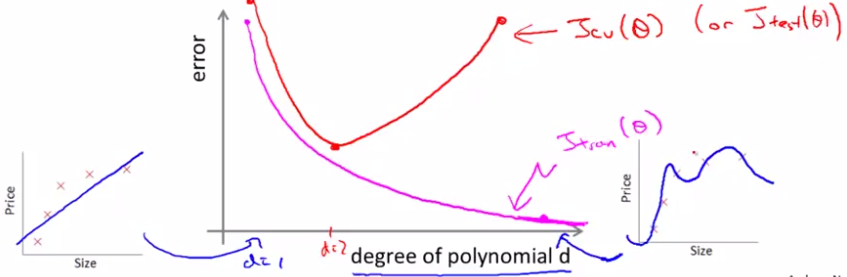
* i.e. the **average squared error** as measured on either set
* Now let's plot the hypothesis error as a function of the degree of polynomial d



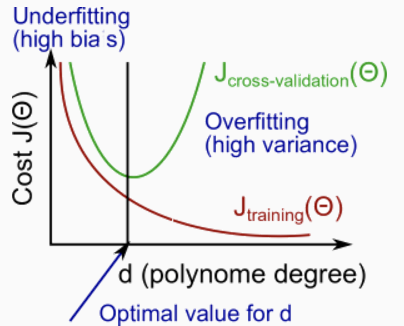
* Where maybe d = 1, we’re fitting very simple functions, when d = 4 or more, we’re fitting very complex, high-order polynomials that might fit the training set w/ much more complex functions
* Let's start with the training error.
* As we increase the degree of the polynomial, we're going to fit our training set better + better and so, if d = 1, we’ll have a very high training error.
* If we have a very high-degree polynomial, our training error is going to be very low, maybe even 0 if it fit the training set that well.
* As we increase of d, typically the training error, J\_train(Ө), decreases



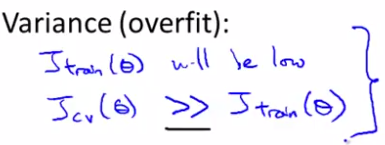
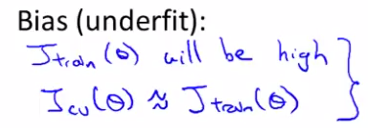
* Next, let's look at the CV error (if we were to plot the test set error, we'd get a pretty similar result to the CV error)
* If we fit an intermediate-degree polynomial, d= 2, we’re going to have a much lower CV error b/c we find a much better fit to the data.
* Conversely if d were too high, d = 4, then we're again *overfitting* + end up w/ a high value for J\_cv(Ө)



* This plot also helps us to better understand the notions of bias + variance.
* If you have a learning algorithm + the CV or test set error is high, we need to figure out if the algorithm suffering from high bias or high variance.
* When CV error is high, this corresponds to 1 of 2 regions
* The region w/ the lower degree d polynomial corresponds to a **high bias problem** (fitting an overly-low order polynomial when we really needed a higher-order polynomial to fit the data)
* The region w/ the higher degree d polynomial corresponds to a high **variance problem** (if d was too large for the data set that we have)



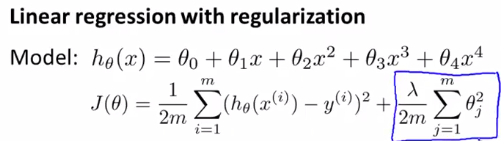
* For the high-bias case (under-fitting), what we find is that both the CV *AND* training error are going to be high.
* If you see this combo, that's a sign your algorithm may be suffering from high bias.
* If your algorithm is suffering from high variance, the training error is going to be low (fitting training set very well) but the CV error is much bigger
* The key that distinguishes these 2 cases is:
* If you have a **high bias** problem, the training set + CV set error will *both* be
* If you have a **high variance** problem, the training set error will usually be lower than the CV error



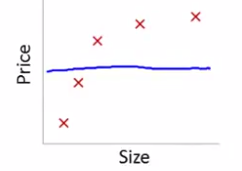
* By diagnosing whether a learning algorithm is suffering from high bias or high variance, we get much better guidance for what might be promising things to try in order to improve the performance of the learning algorithm

**II. REGULARIZATION AND BIAS/VARIANCE**

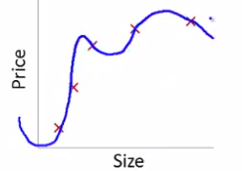
* Suppose we're fitting a high-order polynomial + to prevent over-fitting use regularization to try to keep the values of the parameters (weights) small.



* Consider 3 cases.
* Very large value of the regularization parameter λ, such as 10,000
* In this case, all parameters, Ө1-Өm, would be *heavily* penalized, so we’d end up w/ most parameter values being closer to 0 + the hypothesis hӨ(x) will be approximately = Ө0
* We end up w/ a hӨ(x) is more or less a flat, constant straight line, which high bias + badly underfits the data set



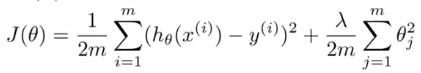
* The other extreme is if we have a very small value of λ, such as near 0
* In that case, given we're fitting a *high order* polynomial, this is typical over-fitting scenario.
* We're fitting a high-order polynomial pretty much *without regularization* or w/ very minimal regularization (penalization), so we end up w/ high-variance in an overfitting situation.



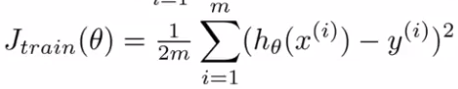
* It's only if we have some intermediate value of λ (neither too large nor too small) that we end up w/ parameters Ө that give us a reasonable fit to the data.
* So, how can we automatically choose a good value for the regularization parameter λ?
* Just to reiterate, here's our model:



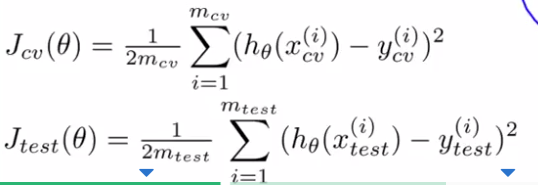
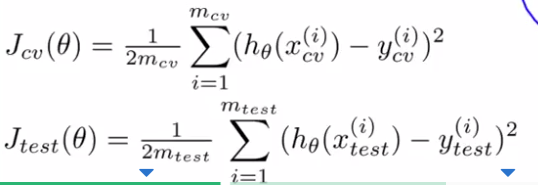
* And here's our learning algorithm's objective 🡺 to minimize:



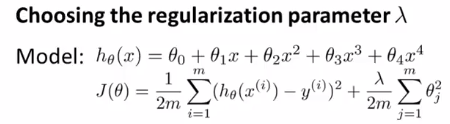
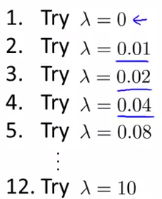
* Then define J\_train(Ө) to be different to be the optimization objective 🡪 take away the regularization term.



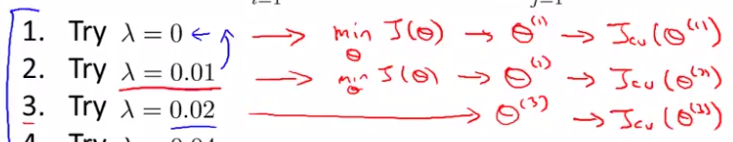
* When using regularization, J\_train(Ө) is no longer the same as J(Ө), the original cost function
* It is just the **sum of squared errors (SSE)**, or the **average squared error,** on the training set w/out taking into account regularization.
* Similarly define the CV + test set errors to be the average SSE on the CV + test sets

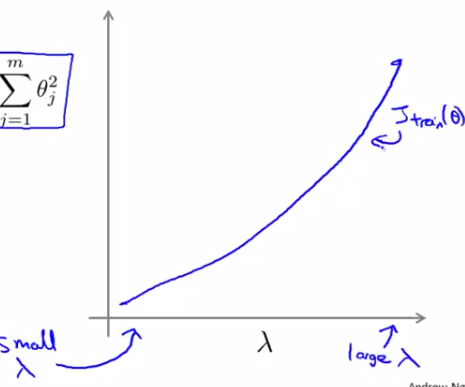
* So our 3 set errors are just 1/2 of the average squared error of each set, w/out an extra regularization term.
* We can automatically choose a regularization parameter λ is to maybe have some range of values of λ to try out, such as 0.01, 0.02, 0.04, + so on, maybe in multiples of 2, until some larger value like 10

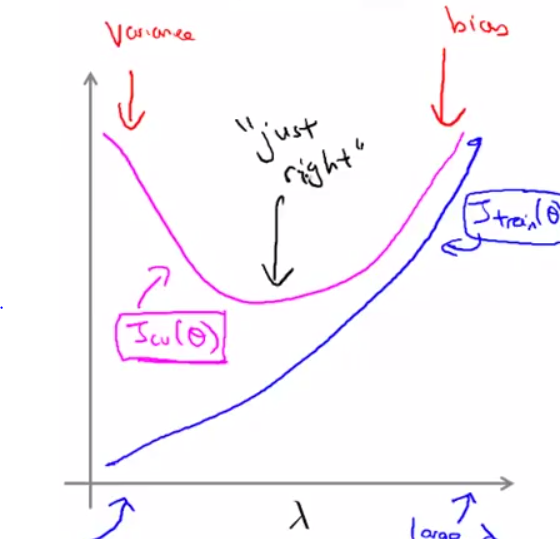
* This gives maybe 12 different model, and what we can do is take the 1st first model w/ λ = 0 + minimize the cost function J(Ө) to get some parameter Ө1
* Then take the 2nd model w/ λ = 0.01, minimize J(Ө) again, + get some different parameter vector Ө2
* Go so on up to Ө(12).
* Then take all the hypotheses these parameters + use the CV set to validate them
* We use the models fit w/ different values of the regularization parameter + evaluate them w/ the CV set to measure the average square error of each of the parameter vectors in the CV set.

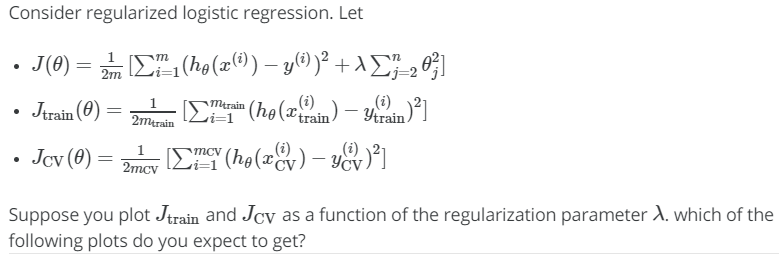
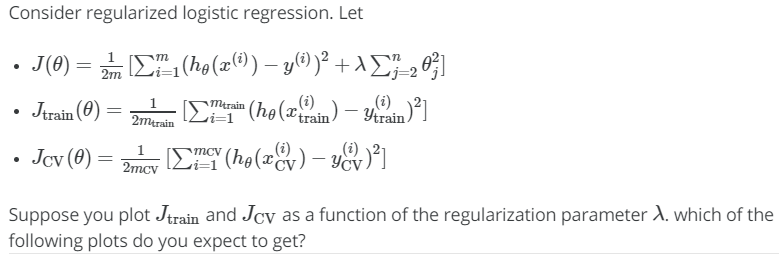
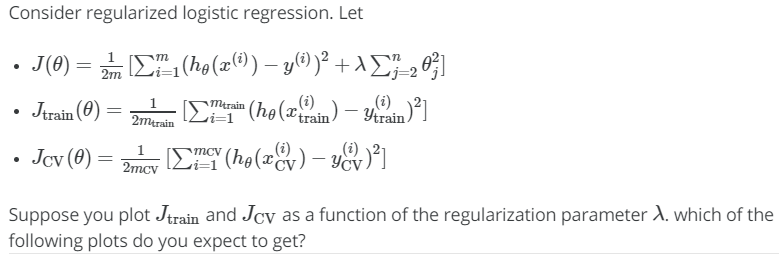


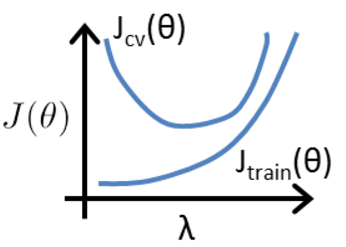
* Then pick whichever model gives the lowest error on the CV set, say Ө5,
* Finally, if we wanted to report the test set error, is to take this parameter Ө5 + look at how well it does on the test set.
* Since we fit our parameter to the CV set, we set aside a separate test set to use to get a better estimate of how well our selected parameter vector will generalize to previously unseen examples.
* So that's model selection applied to selecting the regularization parameter λ.
* Now to get a better understanding of how CV + training error vary as we vary λ.
* Plot J\_train + J\_cv as a means of seeing how well does my hypothesis does on the training + CV sets as we vary λ.
* If λ is small, we're not using much regularization + run a larger risk of overfitting, whereas if λ is large, we run the higher risk of having a biased problem
* What we find is that for small values of λ, we can fit the training set relatively way b/c we're not regularizing, as λ basically goes away you're just minimizing pretty much just gray arrows.
* So, when λ is small, you end up w/ a small value for J\_train
* Whereas if λ is large, you have a high bias problem, + you might not fit your training that well, so



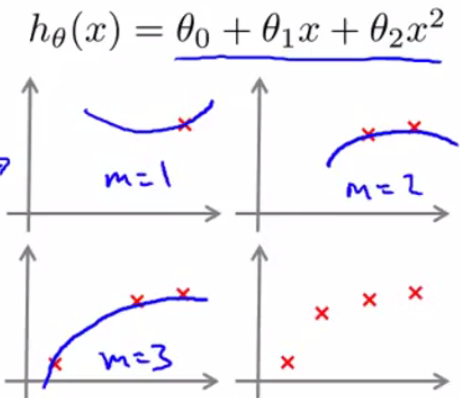
* So J\_train(Ө) will tend to increase when λ increases, b/c a large λ corresponds to high bias where we might not even fit a training set well
* A small value of λ corresponds to maybe fitting a very high degree polynomial to the data
* For the CV error, if we have a large value of λ, we may end up underfitting (high bias), + the CV error will be high.
* So, w/ high bias, we won't be doing well in either set
* Whereas w/ high variance w/ smaller λ’s, we may be overfitting data
* By overfitting, the CV error will be high.

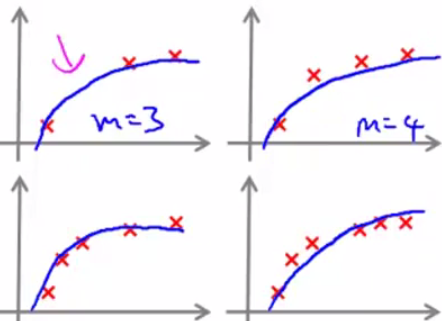
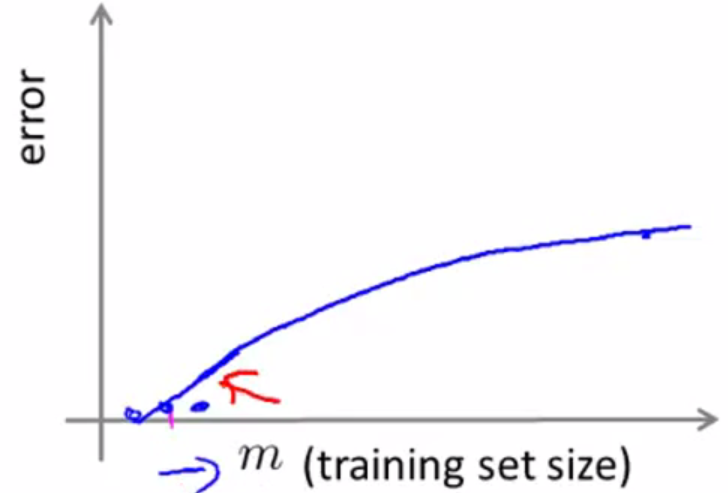


* Once again, it will often be some intermediate value of λ that is just right/works best, in terms of having a small CV error or a small test error
* On the real data set, the curves you get may end up looking more messy + noisy than above
* For some data sets, you will really see those trends
* By looking at a plot of the CV error, you can manually select a point for λ that minimizes the CV error
* When trying to pick a regularization parameter λ for a learning algorithm, often plotting a figure like this helps understand better what's going on + helps verify we are indeed picking a good value for λ.
*   

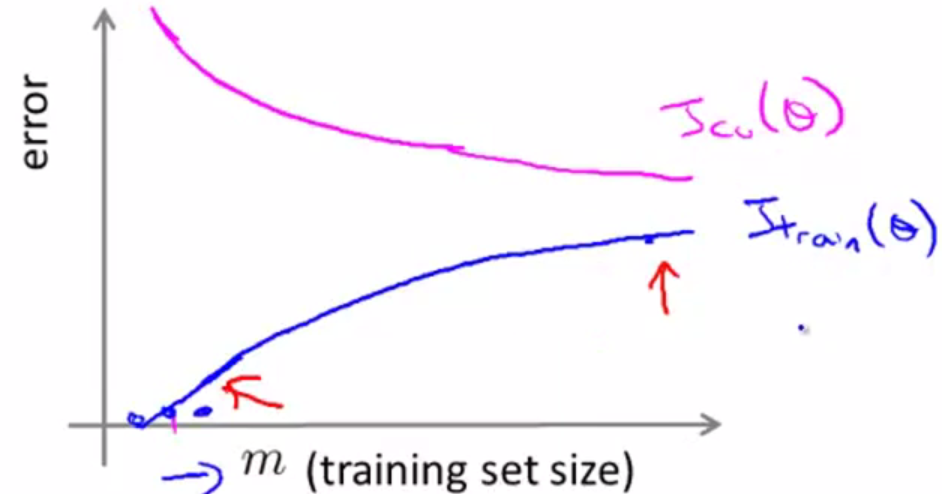


**III. LEARNING CURVES**

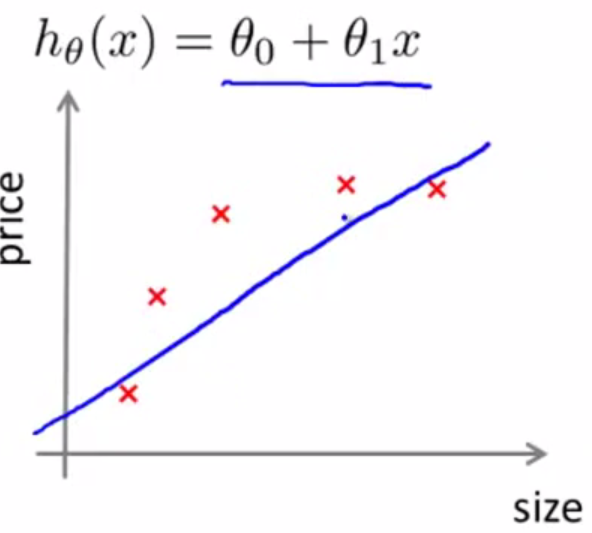
* **Learning curves** are often a very useful thing to plot if you wanted to either check that your algorithm is working correctly, or if you want to improve the performance of the algorithm.
* To plot a learning curve, plot J\_train or J\_cv (average squared error on either set) as a function of m, the number of training examples, but deliberately limit ourselves to using only 10-40 of them
* Suppose we’re fitting a quadratic equation, if we have only 1, 2, or 3 training examples we’re going to be able fit them perfectly (especially w/ regularization)
* 
* Therefore, the training error on my training set is going to be 0, assuming we’re not using regularization (or slightly larger than 0 if using regularization)
* If we have a large training set + artificial restrict the size in order to plot J\_train, we are measuring training error *only on the those plotted examples* we’ve fit the hypothesis to
* So if the training set size is small, the training error is going to be small as well b/c a small training set is going to be very easy to fit very well, maybe even perfectly
* When m = 4 and then 5, It becomes harder to ensure we can find the quadratic function that process through all the examples perfectly
* As the training set size grows, the average training error on the hypothesis increases

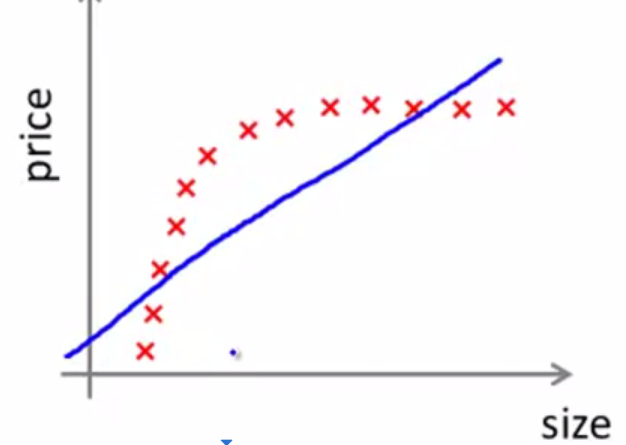
* When m is small, it's pretty easy to fit every single one of your training examples perfectly, so error is going to be small
* When m is larger, it gets harder to fit all training examples perfectly + training set error becomes larger
* When we have a very small training set, we’re not going to generalize well + the hypothesis won’t look like a good one
* It's only when we get a larger training set that we may start to get hypotheses that fit the data somewhat better.
* So the CV + test set error will tend to decrease as training set size increases, b/c the more data we have, the better we do at generalizing to new examples/the better the hypothesis we fit.



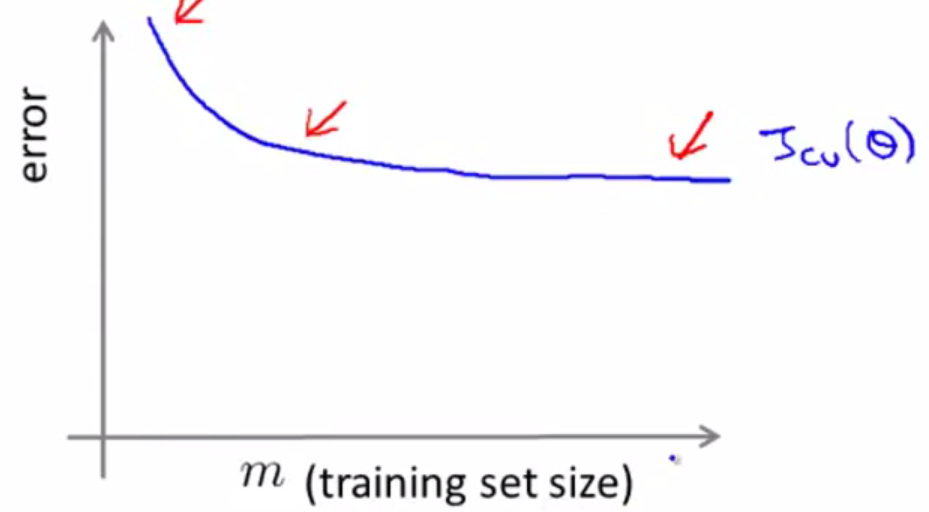
* But what about high bias or high variance problems?
* Suppose hϴ(x) has high bias (underfit) + we increase the training set size



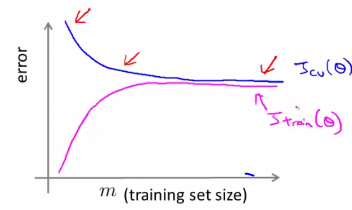
* Now, it we fit a straight line to these new data, we get pretty much the same straight line.



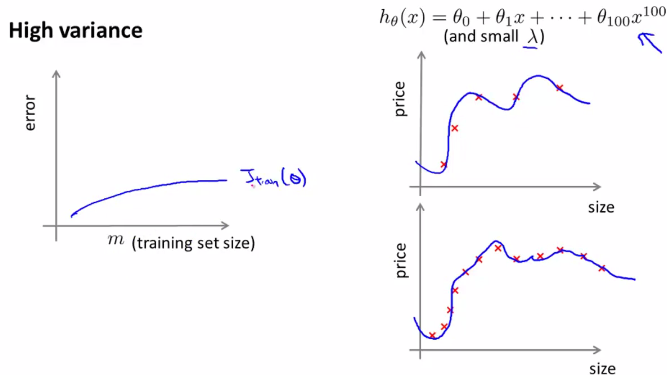
* A straight line just cannot fit this data + bringing in a more data isn't going to change that much.
* This is the *best possible* straight-line fit to this data, but the this model just can't fit this data set well.
* Remember if we you plot CV error, if you have a miniscule training set size, the model won’t do well on the CV set
* But by the time we reach a certain number of training examples, we will almost fit the best possible straight line
* Even if w/ a much larger training set size, you're basically getting the same straight line, so, the CV or test error plateaus out pretty soon (once reaching beyond a certain number of training examples, that lets us pretty much fit the best possible straight line)



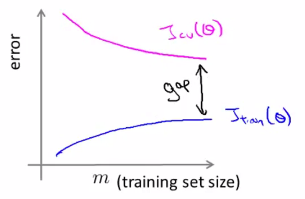
* Training error will again be small, + w/ high bias case, it will end up close to the CV error, b/c we have so few parameters + so much data (at least when m is large)
* Performance on the training set + CV set will be very similar in this case w/ high bias.



* The problem w/ high bias is reflected in the fact that both the CV + training error are high as m increases, so we end up w/ a relatively high value of both J\_cv and J\_train.
* This also implies else something interesting, which is if a learning algorithm has high bias, then as we get more training examples, we'll notice the CV error isn't going down much + basically flattens
* So if learning algorithms are really suffering from high bias, getting more training data will actually not help that much by itself to lower the CV or test set error
* Knowing if a learning algorithm is suffering from high bias is useful b/c this can prevent us from wasting a lot of time collecting more training data when it won’t end up being helpful
* W/ high variance, look at the training error for a small training set when fitting say a very high order polynomial (d =100)
* If using a fairly small value of lambda (but not 0), we'll end up fitting data very well (overfit w/ no penalization for higher order)
* So, if the training set size is small, our J\_train will be small.
* As this m increases a bit, we may still be overfitting a little, bit but it also becomes slightly harder to fit the data set perfectly, so J\_train increases
* This is b/c it’s just a little harder to fit the training set perfectly w/ more examples, but the training set error will still be pretty low.



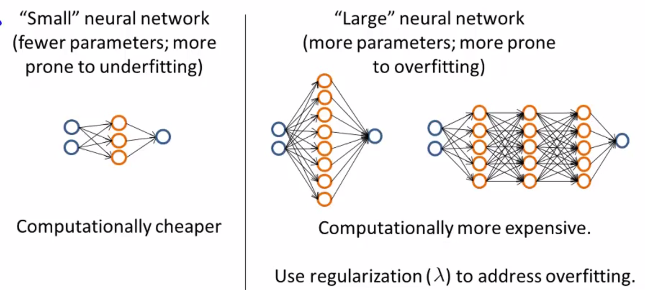
* In a high variance setting, a hypothesis is overfitting so the CV error remains even as we get a moderate number of training examples
* The **indicative diagnostic** that we have a high variance problem is the fact that there's a large gap between the training + CV error.

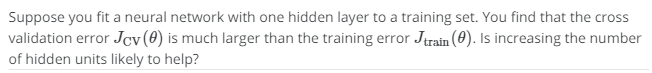


* Looking at this figure, if we think about adding more training data, the 2 curves would be converging to each other.
* It seems it likely the training error will keep on going up + the CV error would keep on going down.
* The thing we *really* care about is the CV or the test set error, right?
* So, if we keep on adding training examples, it looks like our CV error will keep coming down.
* So, in a high variance setting, getting more training data is indeed likely to help.
* Again, this is useful thing to know b/c then we know it may be worthwhile to see if you can go + get some more training data.
* If you plot these curves for an *actual* learning algorithm, sometimes you will actually see curves like these
* But mostly you see curves that are a bit noisier/messier
* Plotting learning curves like these can often help figure out if a learning algorithm is suffering from bias, or variance, or even a little bit of both.
* So when trying to improve the performance of a learning algorithm, try plotting these learning curves to get a better sense of whether there is a bias or variance problem.

**IV. DECIDING WHAT TO DO NEXT REVISITED**

* So how does all this help us figure out what are potentially fruitful + not fruitful things to try to improve performance of a learning algorithm?
* Suppose we implemented/fit a regularized linear regression to predict housing prices + find it doesn't work as well as we're hoping on new test data.
* We have a menu of options to try to improve it:
* Get more training examples
* Good to fix high variance, pointless for high bias
* Check the learning curves 🡪 higher CV w/ more m = high variance
* Try smaller sets of features
* Good to fix high variance, pointless for high bias
* Try additional features
* Usually (not always) thought of as solution to fix high bias
* Current hϴ(x) is too simple so we need more features fir a better fit
* Try adding polynomial features
* Usually (not always) thought of as solution to fix high bias
* Current hϴ(x) is too simple so we need more features fir a better fit
* Try increasing/decreasing **λ**
* This are quick + easy to try, + are less likely to be a waste of months of your life.
* Decreasing **λ** fixes high bias (penalizes less = less underfitting)
* Increasing **λ** fixes high variance (penalize more = less overfitting)
* Finally, let us take everything we have learned and relate it back to NNs
* Remember the practical advice for how to choose the **architecture**/connectivity pattern of the NNs



* 1 option = fit a relatively small NN w/ only 1 hidden layer a relatively few number of hidden units.
* A network like this might have relatively few parameters + be more prone to underfitting.
* The main advantage of these small NNs is that computation will be cheaper.
* An alternative would be to fit a relatively large NN w/ either more hidden units or w/ more hidden layers.
* These NNs tend to have more parameters + therefore be more prone to overfitting.
* 1 disadvantage, *often not a major one*, if w/ a large number of neurons in a network, it can be more computationally expensive.
* Although within reason, this is often hopefully not a huge problem.
* The *main potential problem* of these much larger NNs is that it could be more prone to overfitting
* It turns out if you're applying NN, very often using a larger one network is better, + if it's overfitting, you can then use regularization to address this
* This strategy is often more effective than using a smaller neural network
* Finally, 1 other decisions to make is the number of hidden layers to have
* Using a single hidden layer is a reasonable default
* But if you want to choose the number of hidden layers, another thing to try is get a training, CV, + test set split + try training NNs w/ 1-3 hidden layers + see which of those NNs performs best on the CV sets (compute J\_cv on each)
* 
* 
* If you understood the contents of the last few videos and if you apply them you actually be much more effective already and getting learning algorithms to work on problems and even a large fraction, maybe the majority of practitioners of machine learning here in Silicon Valley today doing these things as their full-time jobs.
* 6:35
* So I hope that these pieces of advice on by experience in diagnostics
* 6:42
* will help you to much effectively and powerfully apply learning and get them to work very well.