Machine Learning
Week 6
Advice for Applying Machine Learning

Evaluating a Learning Algorithm

Deciding What to Try Next

There's a huge difference between someone who that really knows how to powerfully and effectively apply a machine learning algorithm vs. someone who isn't as familiar with the processes.

We want to know which are the most promising avenues to pursue when applying ML.

For instance, say we are continuing the regularized linear regression algorithm to predict housing prices and we see our hypothesis is performing with unacceptably large errors on new predictions. What should we try next?

- Get more training examples
- Try a smaller set of features
- Try adding more features
- Try adding polynomial features
- Try decreasing lambda (regularization term)
- Try increasing lambda

Typically people will start trying the possible steps above in a random order.

There is a very simple technique we can use to rule out some of the options above:

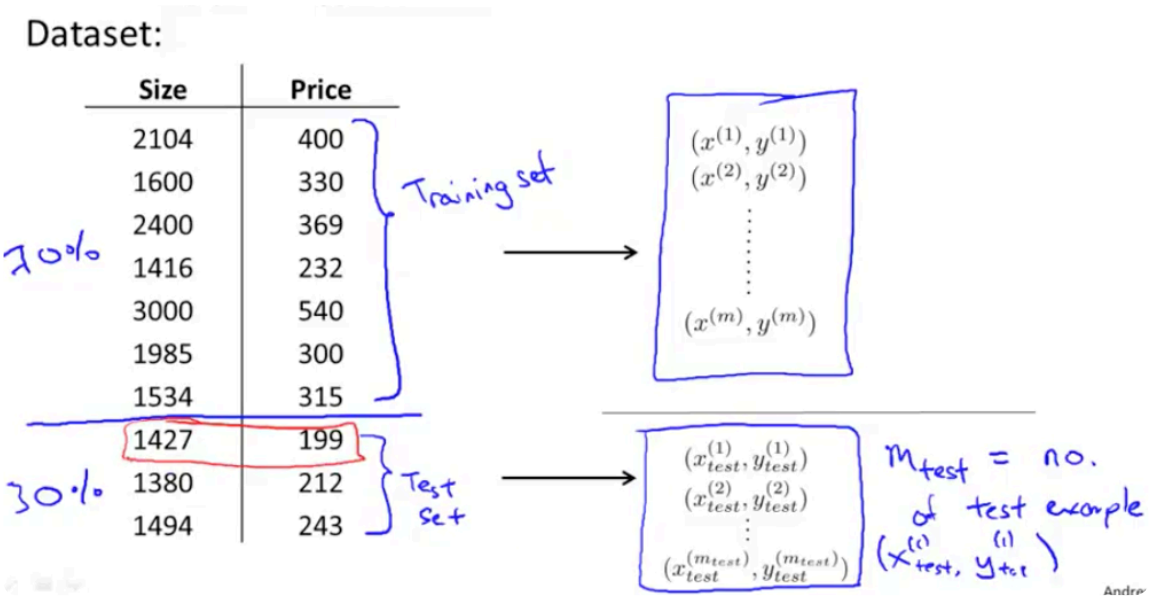
Machine Learning Diagnostic:

- A test that you can run to gain insight on what is/isn't working with a learning algorithm and give you guidance as how best to improve its performance.
- Diagnostics can take some time and effort to implement but are well worth it.

Evaluating a Hypothesis

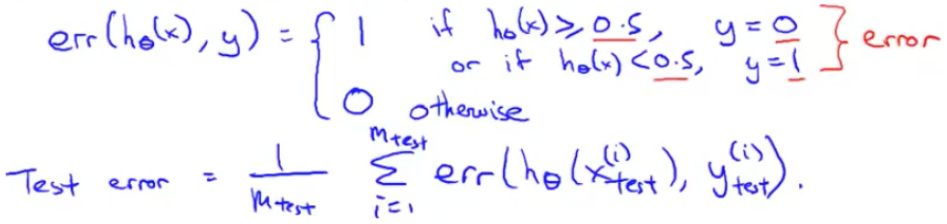
How do we evaluate the hypothesis learned by our algorithm?

How to tell if a hypothesis may be overfitting: Split the dataset into a training set and test set. (70% / 30% split)



We learn parameter theta from the training data and then compute the test set error by running the learned theta parameters w/ our hypothesis on the test set data.

We use the same process or logistic regression, however we look at the misclassification error instead for instances in which our hypothesis misclassifies it's prediction.



Once we have done some trouble shooting for errors in our predictions by:

- Getting more training examples
- Trying smaller sets of features
- Trying additional features
- Trying polynomial features
- Increasing or decreasing λ

We can move on to evaluate our new hypothesis.

A hypothesis may have a low error for the training examples but still be inaccurate (because of overfitting). Thus, to evaluate a hypothesis, given a dataset of training examples, we can split up the data into two sets: a **training set** and a **test set**. Typically, the training set consists of 70 % of your data and the test set is the remaining 30 %.

The new procedure using these two sets is then:

1. Learn Θ and minimize $J_{train}(\Theta)$ using the training set
2. Compute the test set error $J_{test}(\Theta)$

The test set error

1. For linear regression: $J_{test}(\Theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\Theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$
2. For classification ~ Misclassification error (aka 0/1 misclassification error):

$$err(h_\Theta(x), y) = \begin{matrix} 1 & \text{if } h_\Theta(x) \geq 0.5 \text{ and } y = 0 \text{ or } h_\Theta(x) < 0.5 \text{ and } y = 1 \\ 0 & otherwise \end{matrix}$$

This gives us a binary 0 or 1 error result based on a misclassification. The average test error for the test set is:

$$\text{Test Error} = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h_\Theta(x_{test}^{(i)}), y_{test}^{(i)})$$

This gives us the proportion of the test data that was misclassified.

Model Selection and Train / Validation / Test Sets

Suppose you are left to decide what degree of polynomial to fit to the data set, or which features to include, or what to choose for the regularization parameter lambda for a learning algorithm.

These types of questions are part of the model selection process.

Let us consider what degree of polynomial we should select for our model.

'd' = degree of polynomial - we want to measure how well our fitted hypothesis will generalize to new examples as we change 'd'.

We can break our data set into smaller validation sets, fit a hypothesis, and measure the error on each 'test theta' to gain insight into which degree is best for our problem.

As such we use the cross-validation data set for our model selection process.

First - minimize the cost function with parameter theta (10 iterations ranging from d 1 to 10) on the training set, then measure generalization error using the trained theta on the cross-validation data set in order to select the most appropriate model.

Lastly we then finally measure the generalization error on the test using the selected model we chose.

Just because a learning algorithm fits a training set well, that does not mean it is a good hypothesis. It could over fit and as a result your predictions on the test set would be poor. The error of your hypothesis as measured on the data set with which you trained the parameters will be lower than the error on any other data set.

Given many models with different polynomial degrees, we can use a systematic approach to identify the 'best' function. In order to choose the model of your hypothesis, you can test each degree of polynomial and look at the error result.

One way to break down our dataset into the three sets is:

- Training set: 60%
- Cross validation set: 20%
- Test set: 20%

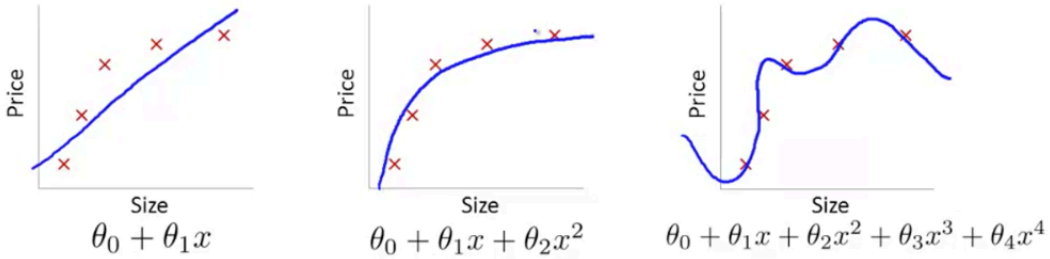
We can now calculate three separate error values for the three different sets using the following method:

1. Optimize the parameters in Θ using the training set for each polynomial degree.
2. Find the polynomial degree d with the least error using the cross validation set.
3. Estimate the generalization error using the test set with $J_{test}(\Theta^{(d)})$, (d = theta from polynomial with lower error);

This way, the degree of the polynomial d has not been trained using the test set.

Diagnosing Bias vs. Variance

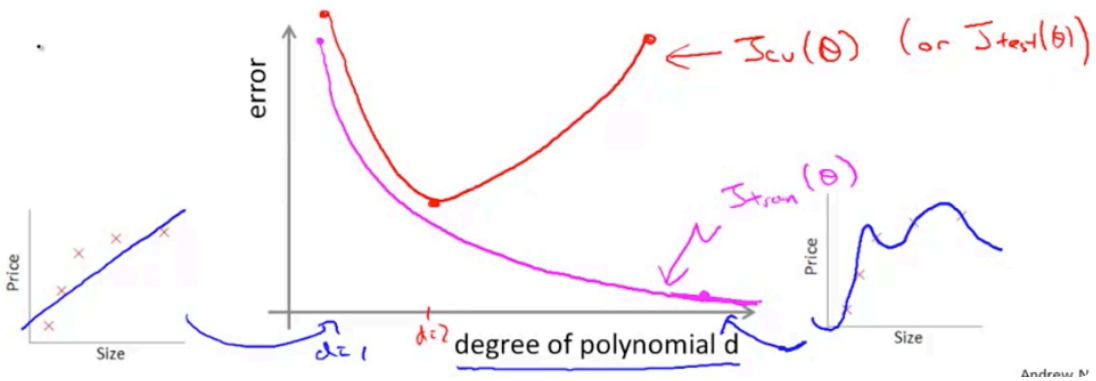
If you are training a learning algorithm and it doesn't perform as well as you are hoping it means you have either a high bias problem or a high variance problem -> either under-fitting or over-fitting.



High bias (underfit) | "Just right" | High variance (overfit)

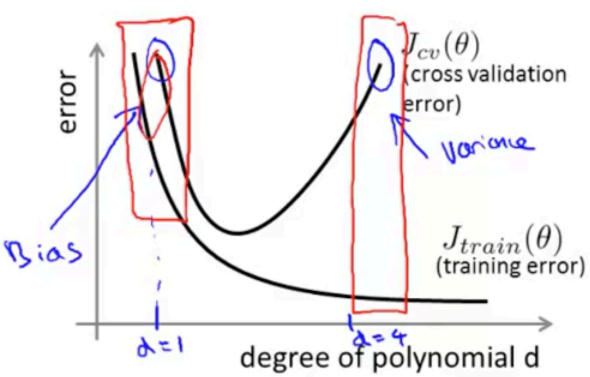
Training error: $J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

Cross validation error: $J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$ (or $J_{test}(\theta)$)



Suppose our learning algorithm is performing poorly, how can we determine if it is suffering from high bias or high variance?

Suppose your learning algorithm is performing less well than you were hoping. ($J_{cv}(\theta)$ or $J_{test}(\theta)$ is high.) Is it a bias problem or a variance problem?



Bias (underfit):
→ $J_{train}(\theta)$ will be high
$J_{cv}(\theta) \approx J_{train}(\theta)$

Variance (overfit):
→ $J_{train}(\theta)$ will be low
$J_{cv}(\theta) \gg J_{train}(\theta)$

In this section we examine the relationship between the degree of the polynomial d and the underfitting or overfitting of our hypothesis.

- We need to distinguish whether **bias** or **variance** is the problem contributing to bad predictions.
- High bias is underfitting and high variance is overfitting. Ideally, we need to find a golden mean between these two.

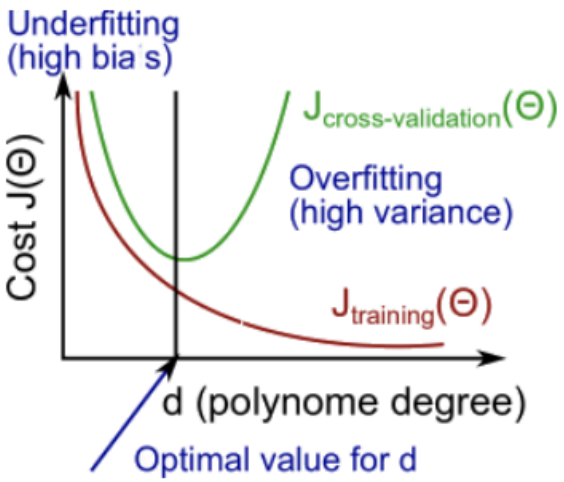
The training error will tend to **decrease** as we increase the degree d of the polynomial.

At the same time, the cross validation error will tend to **decrease** as we increase d up to a point, and then it will **increase** as d is increased, forming a convex curve.

High bias (underfitting): both $J_{train}(\Theta)$ and $J_{CV}(\Theta)$ will be high. Also, $J_{CV}(\Theta) \approx J_{train}(\Theta)$.

High variance (overfitting): $J_{train}(\Theta)$ will be low and $J_{CV}(\Theta)$ will be much greater than $J_{train}(\Theta)$.

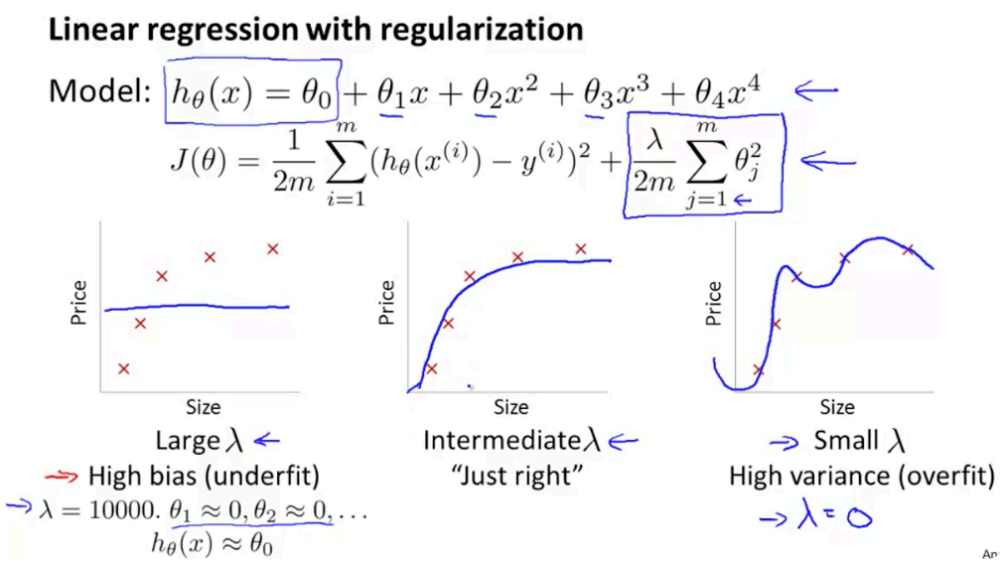
The is summarized in the figure below:



Regularization and Bias / Variance

We know regularization can help prevent over-fitting, but how does it affect the bias and variance of a learning algorithm?

Depending on the value of lambda we see the following:



How can we automatically choose a good value for lambda?

We define our cost functions as follows:

Choosing the regularization parameter λ

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

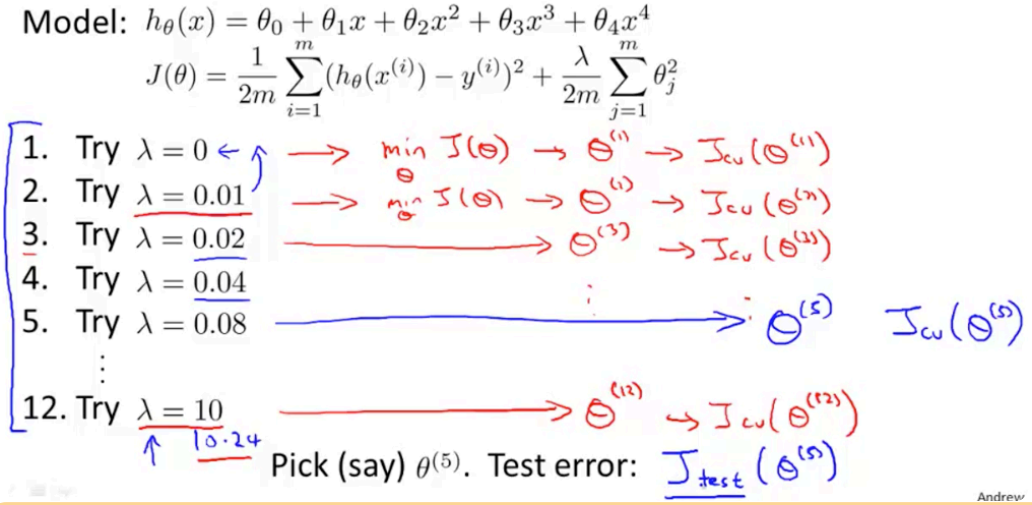
$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

$$J_{train}(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$$

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

$$J_{test}(\theta) = \frac{1}{2m_{test}}\sum_{i=1}^{m_{test}}(h_\theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

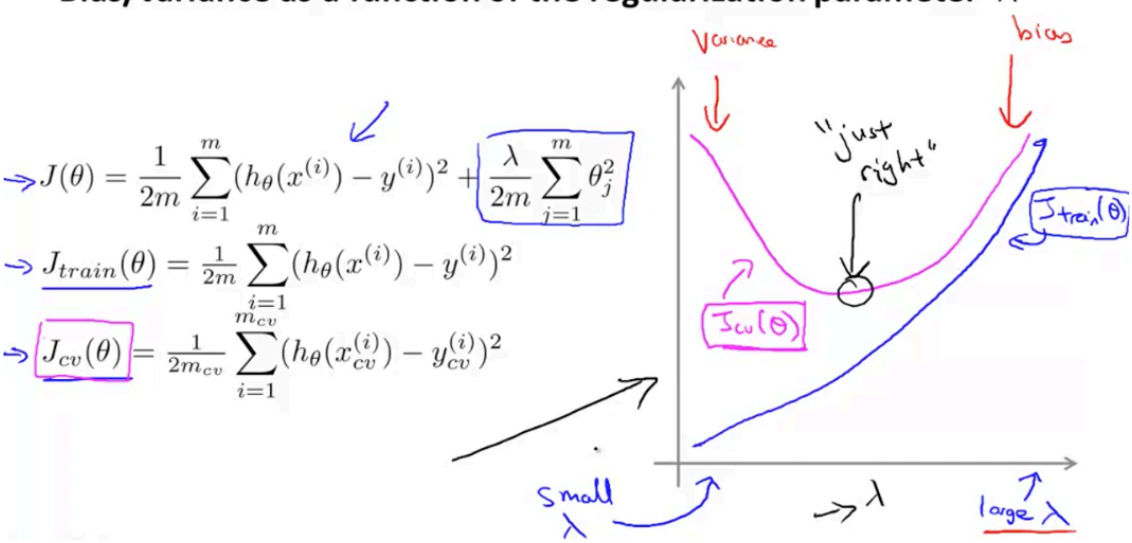
Typically you pick a range for lambda and test them iteratively:



We then finally pick the model that gives us the lowest error on the cross-validation set, then lastly check it on the test set error.

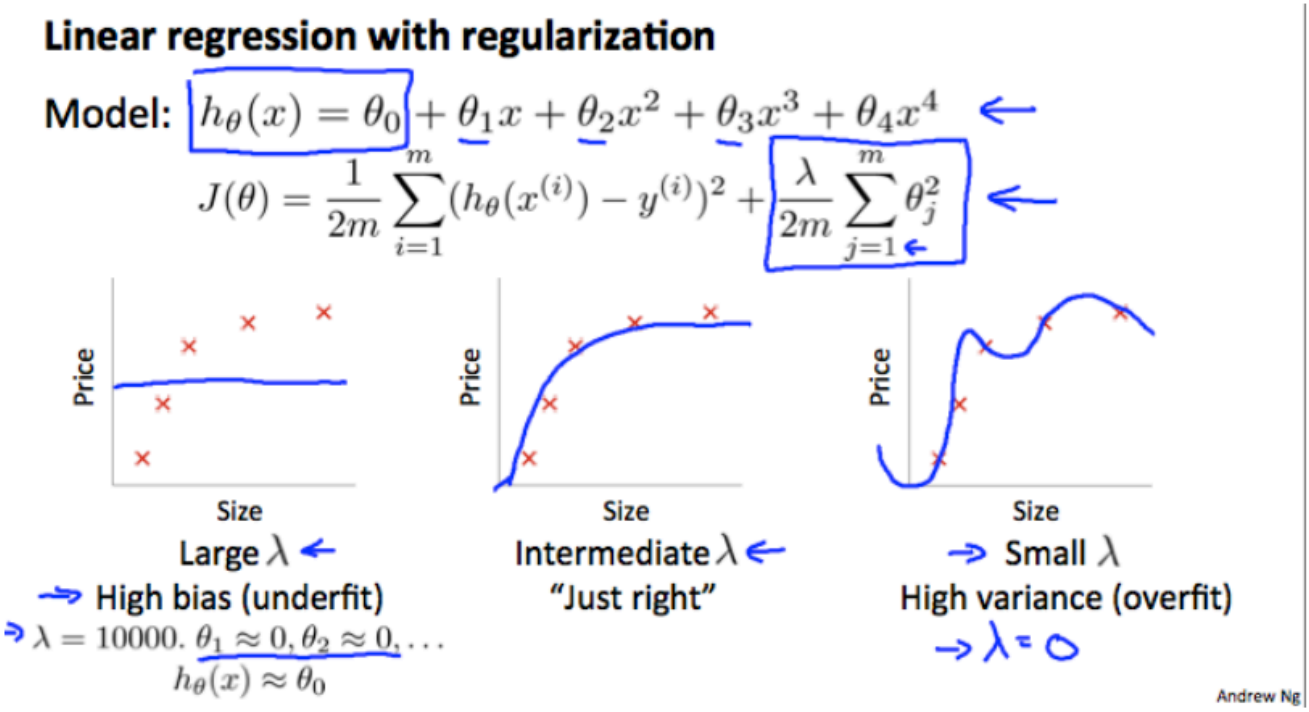
Next, let's observer how bias and variance are affected by changing the regularization parameter lambda:

Bias/variance as a function of the regularization parameter λ



Note these curves are idealized generalizations and in real world problems they may have a lot more noise or varying proportions.

Note: [The regularization term below and through out the video should be $\frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$ and **NOT** $\frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$]



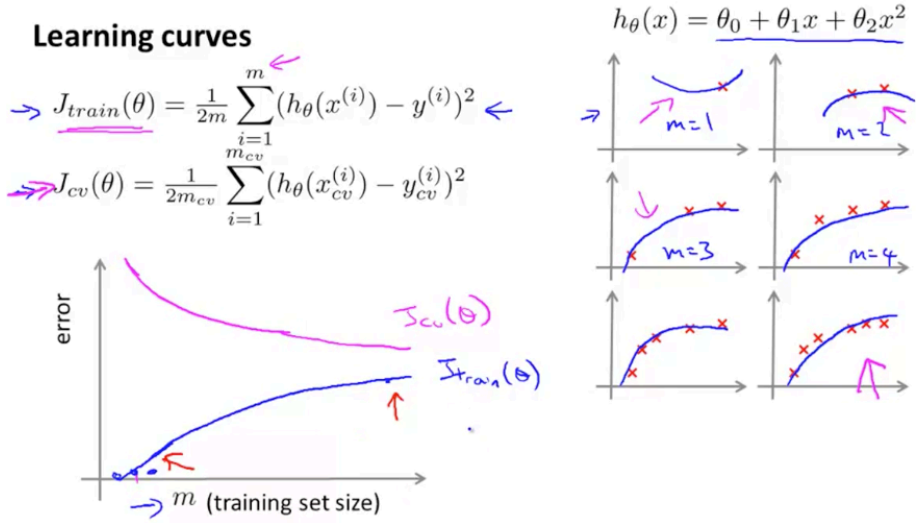
In the figure above, we see that as λ increases, our fit becomes more rigid. On the other hand, as λ approaches 0, we tend to over overfit the data. So how do we choose our parameter λ to get it 'just right' ? In order to choose the model and the regularization term λ, we need to:

1. Create a list of lambdas (i.e. λ∈{0,0.01,0.02,0.04,0.08,0.16,0.32,0.64,1.28,2.56,5.12,10.24});
2. Create a set of models with different degrees or any other variants.
3. Iterate through the λs and for each λ go through all the models to learn some Θ.
4. Compute the cross validation error using the learned Θ (computed with λ) on the $J_{CV}(\Theta)$ **without** regularization or λ = 0.
5. Select the best combo that produces the lowest error on the cross validation set.
6. Using the best combo Θ and λ, apply it on $J_{test}(\Theta)$ to see if it has a good generalization of the problem.

Learning Curves

Learning curves is a useful tool to plot to sanity check that your algorithm is working correctly and to measure performance.

Ex. For a quadratic function below:

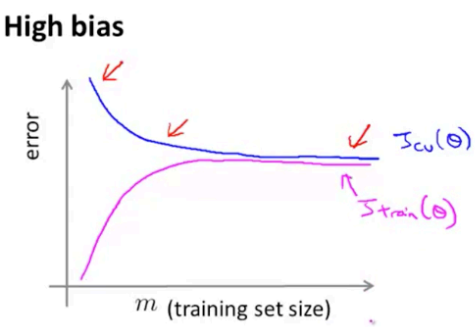


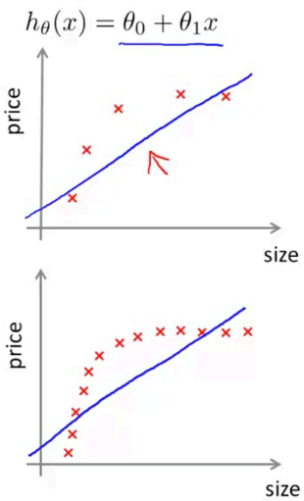
Smaller training sets are easier to fit and will have less training set error, but as the training set gets larger it gets hard to fit and will have more error.

On the flip side, our cross-validation error will start out high for smaller values of 'm' and decrease over more 'm' as it is able to generalize to new examples.

Lets take a look at how these Learning Curves will look in the cases of high bias or high variance:

Ex. For a straight line:





Notice: The Jcv plateaus out a lot sooner (remaining high), and the Jtrain is a lot higher and closer to the Jcv

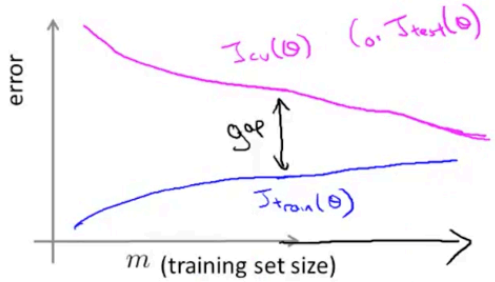
If a learning algorithm is suffering from high bias, getting more training data will not (by itself) improve the performance of the algorithm.

Next let us look at an example of a learning curve indicating high variance.

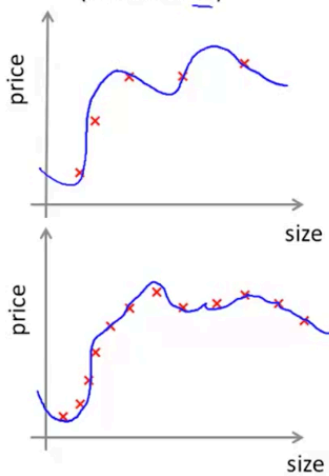
Ex. For a very high order polynomial:

Notice the training error will be low but the cross validation error will remain high. The main indicator to look for is the gap between these curves.

High variance



$h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{100} x^{100}$
(and small λ)



If a learning algorithm is suffering from high variance, getting more training data is likely to help. ⇐

** Note for all the Learning Curves above, these are all idealized generalizations, in reality they will be a lot messier and noisy.

Training an algorithm on a very few number of data points (such as 1, 2 or 3) will easily have 0 errors because we can always find a quadratic curve that touches exactly those number of points. Hence:

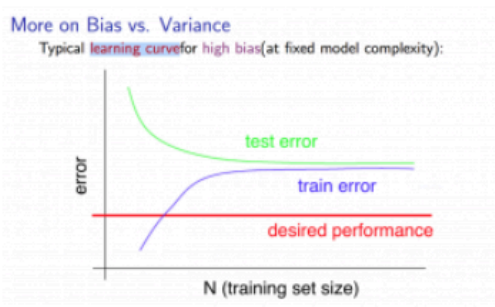
- As the training set gets larger, the error for a quadratic function increases.
- The error value will plateau out after a certain m, or training set size.

Experiencing high bias:

Low training set size: causes $J_{train}(\Theta)$ to be low and $J_{CV}(\Theta)$ to be high.

Large training set size: causes both $J_{train}(\Theta)$ and $J_{CV}(\Theta)$ to be high with $J_{train}(\Theta) \approx J_{CV}(\Theta)$.

If a learning algorithm is suffering from **high bias**, getting more training data will not **(by itself)** help much.

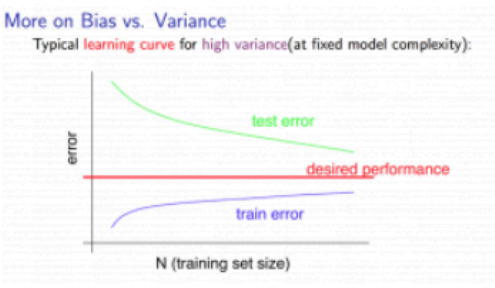


Experiencing high variance:

Low training set size: $J_{train}(\Theta)$ will be low and $J_{CV}(\Theta)$ will be high.

Large training set size: $J_{train}(\Theta)$ increases with training set size and $J_{CV}(\Theta)$ continues to decrease without leveling off. Also, $J_{train}(\Theta) < J_{CV}(\Theta)$ but the difference between them remains significant.

If a learning algorithm is suffering from **high variance**, getting more training data is likely to help.



Deciding what to try next (Revisited)

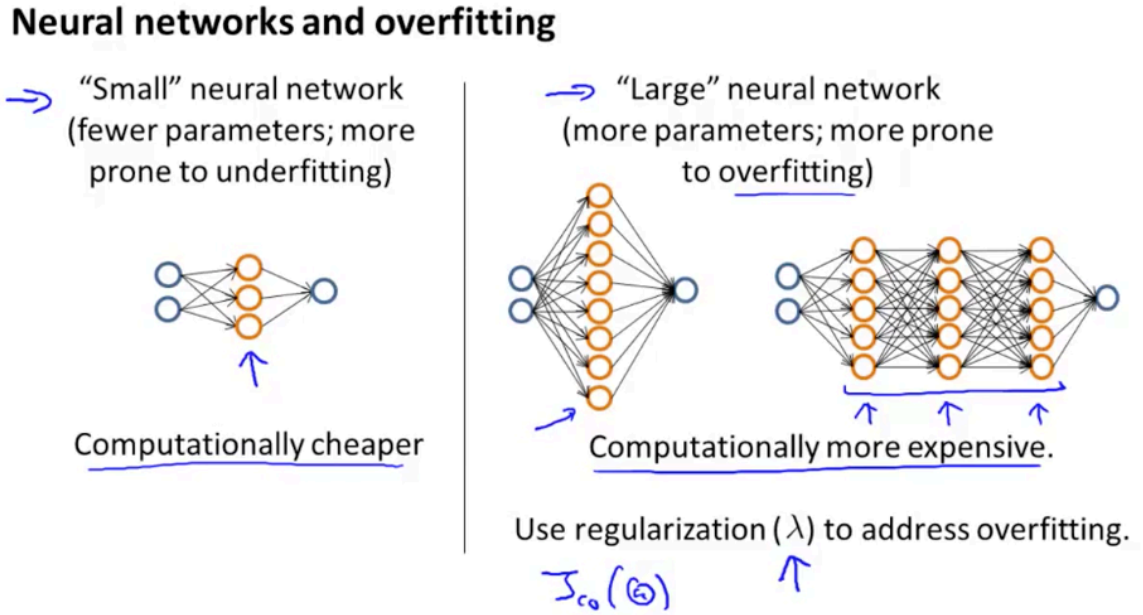
We've talked about how to evaluate learning algorithms, model selection, detecting high bias and high variant.

How do we determine which activities will be the most useful when trying to improve the performance of our learning algorithm?

Suppose you have implemented regularized linear regression to predict housing prices. However, when you test your hypothesis on a new set of houses, you find that it makes unacceptably large error on new predictions. What should you try next?

- Get more training examples -> fixes high variance
- Try smaller sets of features -> fixes high variance
- Try getting additional features -> fixes high bias
- Try adding polynomial features -> fixes high bias
- Try decreasing lambda -> fixes high bias
- Try increasing lambda -> fixes high variance

How does this relate to Neural Networks?



Adding more hidden layers fixes high variance (?)

Diagnosing Neural Networks

- A neural network with fewer parameters is **prone to underfitting**. It is also **computationally cheaper**.
- A large neural network with more parameters is **prone to overfitting**. It is also **computationally expensive**. In this case you can use regularization (increase λ) to address the overfitting.

Using a single hidden layer is a good starting default. You can train your neural network on a number of hidden layers using your cross validation set. You can then select the one that performs best.

Model Complexity Effects:

- Lower-order polynomials (low model complexity) have high bias and low variance. In this case, the model fits poorly consistently.
- Higher-order polynomials (high model complexity) fit the training data extremely well and the test data extremely poorly. These have low bias on the training data, but very high variance.
- In reality, we would want to choose a model somewhere in between, that can generalize well but also fits the data reasonably well.