ADVICE FOR APPLYING MACHINE LEARNING

Lecture 59: Deciding What to Try Next

Suppose we've implemented regularized linear regression to predict housing prices. However, when we test our hypothesis on a new set of houses, we find that it makes unacceptably larger errors in its predictions. What should we try next?

* Get more training data
* Try smaller set of features
* Try getting additional features
* Try adding polynomial features
* Try decreasing λ (regularization parameter)
* Try increasing λ

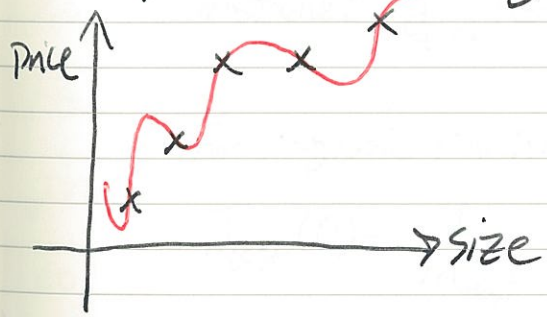
Pursuing some of these could be very time consuming & resource intensive. It would be good to have some intuition for what could potentially help, and what may not be worth trying because it's unlikely to improve the performance of the learning algorithm.

Machine learning diagnostics

These are tests we can run to gain insight into what is/isn't working with a learning algorithm, and gain guidance as to how best to improve its performance. Diagnostics often take quite some time to implement, but doing so can be a very good use of time. Andrew claims he has seen many people spend months pursuing an avenue which a diagnostic tool would've immediately been able to rule out.

Lecture 60: Evaluating a Hypothesis

How do we know our hypothesis is good? We've already seen that we cannot rely on the training error, since we may have a very low error on the training set but fail to generalize to new examples. This is why we implemented regularization.



$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x^2 + \cdots + \theta_{20} x^{20}$$

In general, how can we know our hypothesis is overfitting the data? In the example above it's obvious because we only have one

feature and can plot the hypothesis. In general we have many features and cannot rely on that. The standard way of dealing with this is to first split our data set into two parts:

(i) Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$

(ii) Test set: $(x_{test}^{(1)}, y_{test}^{(1)}), (x_{test}^{(2)}, y_{test}^{(2)}), \ldots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

Normally the split is about 70% for training & 30% for testing:

$$\frac{m_{test}}{m_{test} + m} \sim 30\%$$

Note that if there's any ordering to the data, it's better to select a random ~70% for training and ~30% for testing.

Then we do the following:

(i) Learn the parameters Θ from the training set.

(ii) Compute error on the test set. For linear regression, for instance:

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

For logistic regression:

$$J_{test}(\Theta) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} y_{test}^{(i)} \log h_\Theta(x_{test}^{(i)}) + (1 - y_{test}^{(i)}) \log(1 - h_\Theta(x_{test}^{(i)}))$$

For classification problems, another measure of error commonly used is the misclassification error.

Misclassification error reports the percentage of times the learning algorithm misclassifies on the test set:

$$\text{err}(h_\theta(x), y) = \begin{cases} 1 & \text{if } \begin{cases} h_\theta(x) \geq 0.5 \ \& \ y = 0 \\ h_\theta(x) < 0.5 \ \& \ y = 1 \end{cases} \\ 0 & \text{otherwise} \end{cases}$$

$$\text{Test error} = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} \text{err}(h_\theta(x^{(i)}_{test}), y^{(i)}_{test})$$

We can already see why the split of training/test sets can help us determine if our learning algorithm is overfitting:

If $J_{training}(\theta) \ll J_{test}(\theta)$

↑ not generalizing well to test set.

Lecture 61 & Model Selection & Training/Validation/Test Sets

Suppose we want to decide what degree polynomial to use in linear regression, or choose the optimal regularization parameter λ. How can we achieve this? These are called model selection problems. We've already seen that if we fit our parameters θ to the training set, measuring the error of prediction on the same data set is not a good indication of the error on a new dataset. This is a general principle that should be

followed: If you determine the optimal value of a set of parameters by fitting a model to a data set, do not use that same data set to evaluate how good your fit is going to be generically

Suppose we want to decide what degree polynomial to use for a linear regression problem:

* $d=1$: $h_\theta(x) = \theta_0 + \theta_1 x_1$ $\xrightarrow{\text{linear reg}}$ $\theta^{(1)}$
* $d=2$: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x^2$ $\xrightarrow{\text{linear reg}}$ $\theta^{(2)}$
* $\vdots$
* $d=10$: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_{10} x^{10}$ $\xrightarrow{\text{linear reg}}$ $\theta^{(10)}$

It's as if we're trying to fit for the optimal d. We can run linear regression on our training set using all models, and obtain the optimal parameters $\theta^{(1)}$ (for $d=1$ model), $\theta^{(2)}$ (for $d=2$ model), ..., $\theta^{(10)}$ (for $d=10$ model). Once we have the optimal parameters, we can now estimate the error of each on the test set: $J_{test}(\theta^{(1)})$, $J_{test}(\theta^{(2)})$, ..., $J_{test}(\theta^{(10)})$. We can then pick the case with the lowest test set error. Let's assume this is the $d=5$ case. Having chosen the model $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_5 x^5$, we now want to know how

well it generalizes to new examples, i.e examples that are neither in the training nor test sets. Is it fair to report $J_{test}(\theta^{(5)})$? No, because we fit "d" to the test set, $J_{test}(\theta^{(5)})$ would be an overly optimistic estimate of generalization error.

To solve this problem, we can split our data set into three different groups:

(i) Training set: $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$

(ii) Cross-Validation set: $(x_{cv}^{(1)}, y_{cv}^{(1)}), \ldots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$

(iii) Test set: $(x_{test}^{(1)}, y_{test}^{(1)}), \ldots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

The ratios are normally as follows:

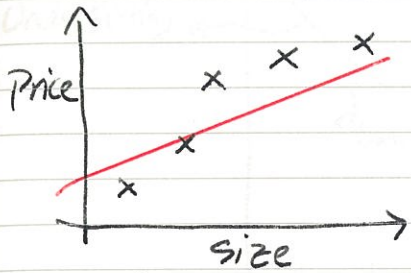
(i) Training set $\sim$ 60%

(ii) Cross-validation set $\sim$ 20%

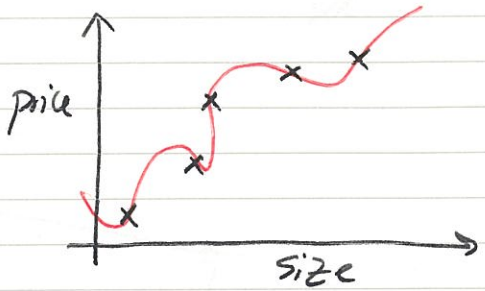
(iii) Test set $\sim$ 20%

The cross-validation set is also called the validation set. Going back to our problem, we would fit "d" to the cross-validation set by choosing the model with the smallest $J_{cv}(\theta^{(d)})$. Once we choose "d", we can report the generalization error as $J_{test}(\theta^{(d)})$.

Lecture 62: Diagnosing Bias vs. Variance



High bias (underfit)



High variance (overfit)

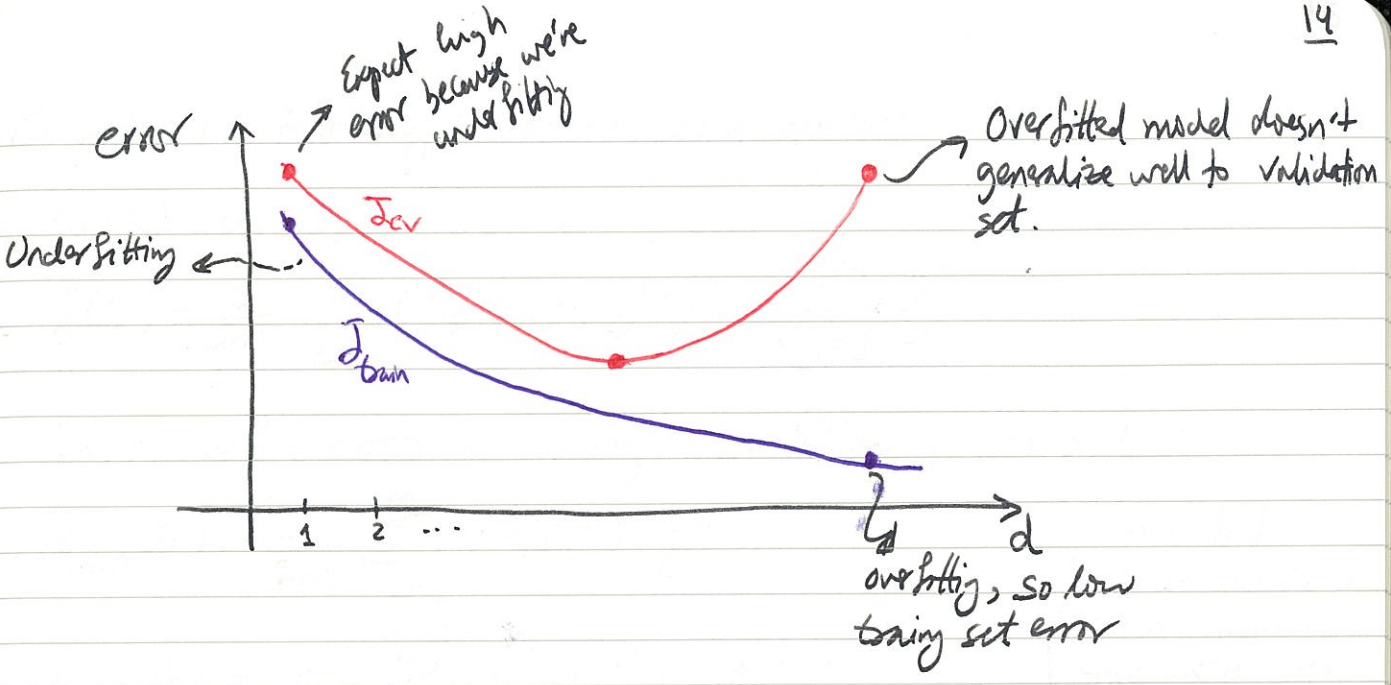
When a learning algorithm doesn't work as well as it should, it's almost always because we're either underfitting (high bias) or overfitting (high variance). How can we tell which is the case?

Consider the training and cross-validation error for the problem in the last lecture:

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

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

Remember what we did was train different polynomials with varying degrees $d = 1, 2, \ldots, 10$ on the training set, compute $J_{cv}(\theta)$ for each fit, and pick the model with the lowest J_{cv}. Let's think about how J_{train} & J_{cv} might behave as a function of d:



We can now see how we can distinguish between high bias vs. variance:

* High bias:
  - $J_{train}(\theta)$ is high
  - $J_{cv}(\theta) \approx J_{train}(\theta)$
* High variance:
  - $J_{train}(\theta)$ is low
  - $J_{cv}(\theta) \gg J_{train}(\theta)$

Lecture 63: Regularization and Bias/Variance

We've already seen that regularization can prevent overfitting. But how can we automatically choose an appropriate value for λ? If λ is too large, the model will have high bias. If it's too low, it won't help with overfitting at all. We need a more robust way of finding an appropriate value of λ than just guessing.

Consider the following linear regression model:

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \dots + \theta_{10} x^{10}$$

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

Let's define errors on training/cross-validation/test sets:

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

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

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

Note that we don't include the regularization term since we're just interested in the error of prediction.

We can select λ, just as we selected d in lecture 61:

(i) Fit model to training set for different values of λ:

1) $\lambda = 0$ — Regularized linear regression → $\theta^{(1)}$

2) $\lambda = 0.01$ — Regularized linear regression → $\theta^{(2)}$

3) $\lambda = 0.02$ — Regularized linear regression → $\theta^{(3)}$

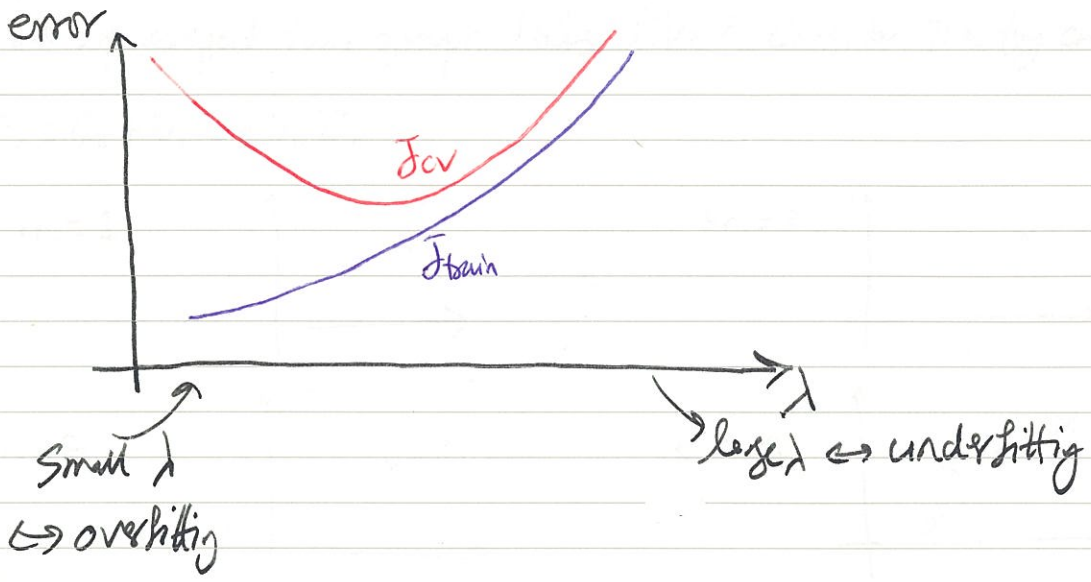
4) $\lambda = 0.04$ — Regularized linear regression → $\theta^{(4)}$

⋮

12) $\lambda = 10.24$ — Regularized linear regression → $\theta^{(12)}$

(ii) Pick value of λ with lowest $J_{cv}(\theta)$, i.e lowest cross-validation error. Let's say this was $\theta^{(4)}$.

(iii) Report $J_{test}(\theta^{(4)})$ as generalization error.



Lecture 64: Learning Curves

Learning curves are useful for diagnostics and reveal quite a lot of useful information about learning algorithms. Consider linear regression, for instance:

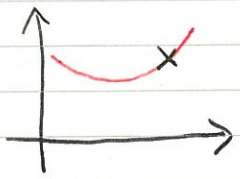
$$* J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

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

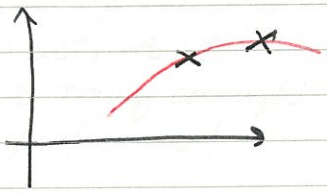
Learning curves are constructed by seeing how J_{train} and J_{cv} change with m, the # of training examples: (i) set m (say $m=10$), (ii) find optimal θ on training set which is now reduced to 10 samples, (iii) compute $J_{train}(\theta)$ on the reduced training set, (iv) compute $J_{cv}(\theta)$ on the entire cross-validation set. Repeat this for various values of m.

How do we expect this graph looks like? Consider the toy example $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$:

* $m = 1$:

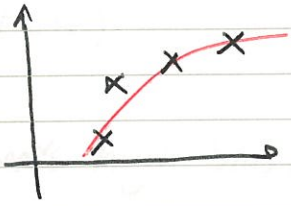


* $m = 2$:

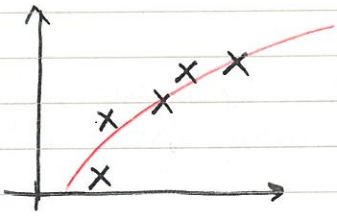


* $m = 3$:

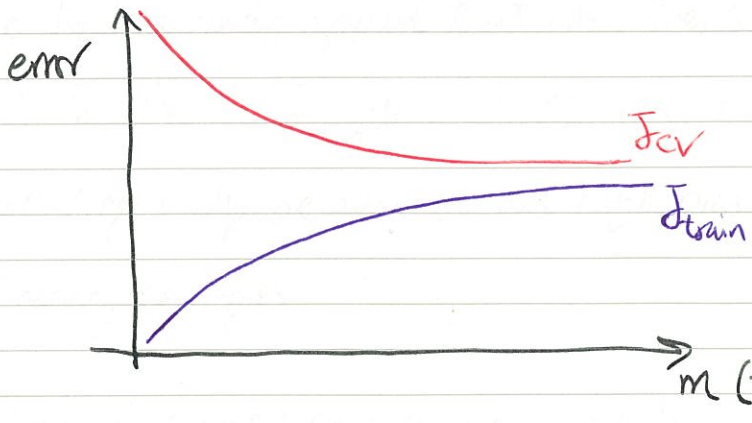
* $m = 4$:



* $m = 5$:



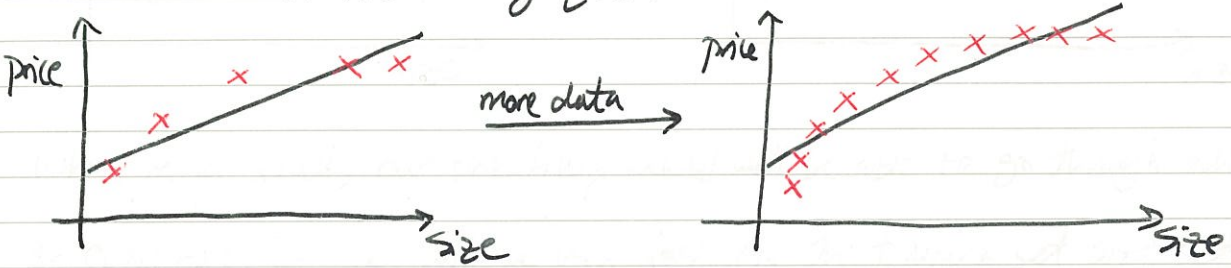
Of course, $J_{train}(\theta)$ for $m = 1, 2, 3$ would be zero, because we can fit a quadratic exactly through all data points. But when $m > 3$, we will start to see more noise and our model will not fit the training set perfectly. As a result, we would expect $J_{train}(\theta)$ to grow with m. The cross-validation error, however, will be the opposite: the more data we have in the training set, the more likely it is that our fit will also be better on the cross-validation set. We cannot fit the model to one data point, and expect it to be accurate on a unseen sample of size 100!



Note here the importance of having two separate data sets for training & cross-validation/testing. If we were measuring our generalization error on the training set, we would conclude that more training examples would hurt the performance!

What would the learning curves look like when there's high bias or high variance?

High Bias: Consider the model $h_\theta(x) = \theta_0 + \theta_1 x$ being fitted to a dataset that's clearly quadratic:

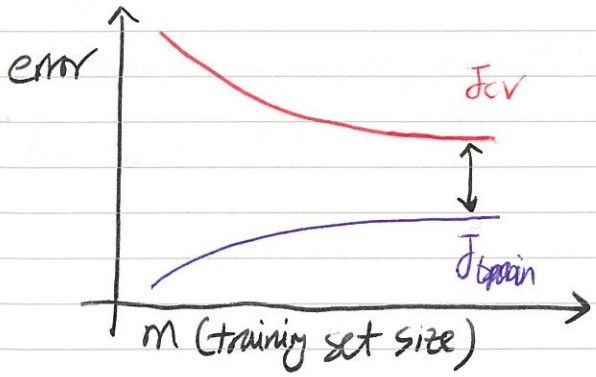


It should be clear that if we're underfitting, increasing the # of training examples is not likely to help. In other words, $J_{train}(\theta)$ should be high, and plateau relatively quickly as a function of the training size. We would then expect the learning curves to look as follows:

we had already argued that when there's high bias, $J_{train}(\theta)$ is high and $J_{cv}(\theta) \approx J_{train}(\theta)$. What's more, however, is that an algorithm that's behaving badly because it has high bias, will not benefit from more training examples.

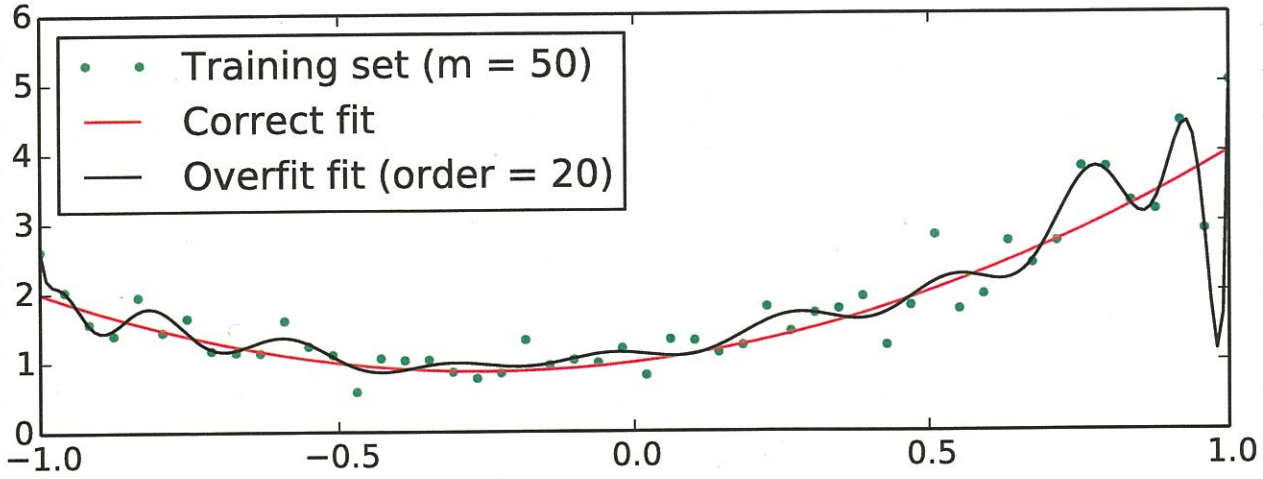
High Variance: consider the model $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_{100} x^{100}$ being fitted to a dataset which is quadratic but with some noise:

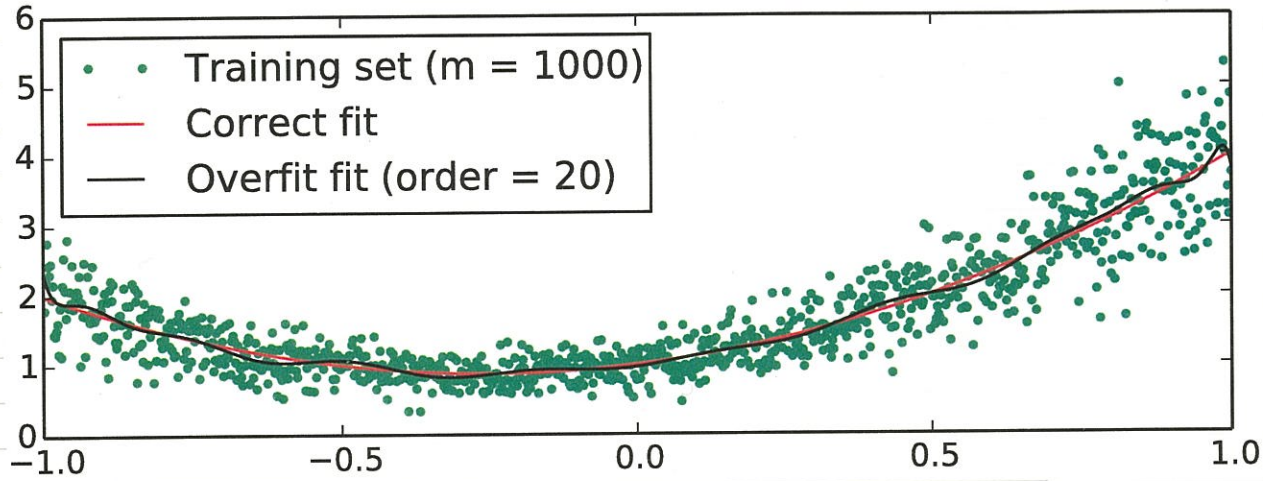
When m is small, our over-fitting model will be able to go through all the points & therefore $J_{train}(\theta)$ will be very low. As the training set size gets larger, it will still over-fit, but now it becomes harder to go through all points, which means $J_{train}(\theta)$ will get larger.



As we had argued before, we can detect high variance when $J_{train}(\theta)$ is small and $J_{cv}(\theta) \gg J_{train}(\theta)$, because our model doesn't generalize well.

Does having more training data help when there's high variance?





Yes! Having a lot more data has the effect of regularizing the overfitting away. As it can be seen from the above figure, when there's more data, the model can no longer go through most training examples and converges to the correct fit.

Lecture 65: Deciding What to Do Next Revisited

Let's go back to the example on page 7 and see what we can say about the choices we were contemplating for making our algorithm better:

* Get more training example
  → fixes high variance, but doesn't help with high bias.
* Try smaller set of features.
  → Helps with high variance. If we have high bias, it means we need more (complex) features, not less!
* Try getting additional features
  → Helps with high bias, since in that case our hypothesis might be too simple.
* Try adding polynomial features
  → Helps with high bias, since in that case our hypothesis is probably too simplistic.
* Try decreasing λ
  → Helps with high bias
* Try increasing λ
  → Helps with high variance.

A bit more on neural network architecture:

* Small neural network:
  - → Fewer parameters; more prone to underfitting.
  - → Computationally cheaper
* Large neural network:
  - → More parameters; more prone to overfitting.
  - → computationally more expensive. Andrew claims that most of the time, this is not a huge problem.

Often using a larger neural network with regularization is more effective than using a smaller one.

If we're unsure about having one hidden layer with many units vs. more hidden layers with fewer units in each, we can always train using different architectures, compute the error of each case on the cross-validation set, and pick the architecture with the lowest cross-validation error.