Week 6:
6/12/17

6. Evaluating Learning Algorithms

Now, we take a look at how to improve the learning algorithm. This includes how to tell when a learning algorithm is doing poorly, and describe the 'best practices' for how to 'debug' your learning algorithm and go about improving its performance.

We will also be covering machine learning system design. To optimize a machine learning algorithm, you'll need to first understand where the biggest improvements can be made. In these lessons, we discuss how to understand the performance of a machine learning system with multiple parts, and also how to deal with skewed data.

In developing a learning system, it's important to choose the most promising avenues to spend your time pursuing.

6.1. Debugging a learning algorithm.

To begin, let's look at an example:

Example 6.1 : *Debugging* Suppose you have implemented a regularized linear regression model to predict housing prices:

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

However, when you test your hypothesis on a new set of houses, you find that it makes unacceptably large errors in its predictions. *What should you try next?*

A few notable options are:

- Getting more training examples (*get more data*)
- Trying smaller sets of features
- Trying additional features
- Trying polynomial features
- Increasing/decreasing λ

Often is the case that someone randomly chooses one of these options, which takes several months. However, there are many ways to evaluate learning algorithms and determine which of the above is the best approach.

Definition 1. ***Machine Learning Diagnostics:*** A test that you can run to gain insight what is/isn't working with a learning algorithm, and gain guidance as to how best to improve its performance.

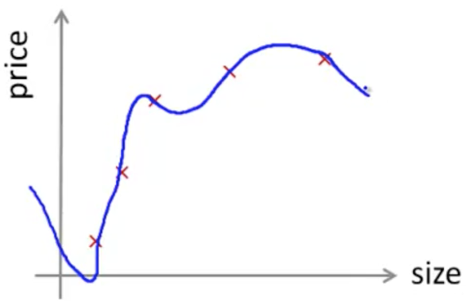
Diagnostics can take time to implement, but doing so can be a very good use of your time.

6.1.1. *Evaluating a Hypothesis.*

When we fit the parameters of our learning algorithm we think about choosing the parameters to minimize the training error. One might think that getting a really low value of training error might be a good thing, but we have already seen that just because a hypothesis has low training error, that doesn't mean it is necessarily a good hypothesis. As we've seen before,

We've already seen the example of how a hypothesis can overfit — and therefore fail to generalize the new examples not in the training set.

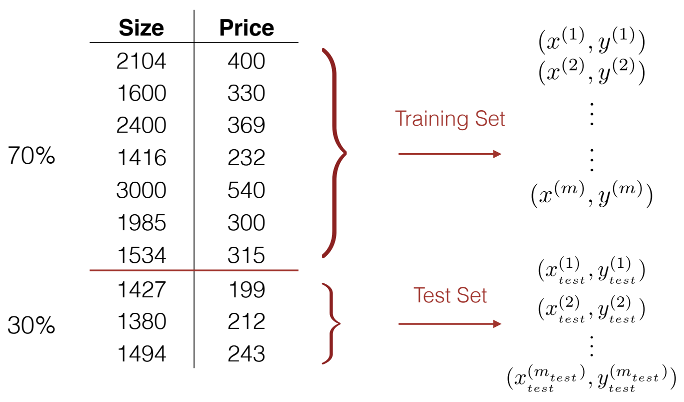
- ***Overfitting:*** Fails to generalize to new examples not in training set



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

So how do you tell if the hypothesis might be overfitting? In this simple example we could plot $h_\theta(x)$ and just see what was going on. But in general this is not feasible. So, the general way to approach this problem, is by evaluating the hypothesis function. In order to do this, we first divide the data into two sets:

- *(70%)* Training Set
- *(30%)* Test Set



	Size	Price		
70%	2104	400	Training Set	$(x^{(1)}, y^{(1)})$
	1600	330		$(x^{(2)}, y^{(2)})$
	2400	369		$\vdots$
	1416	232		$\vdots$
	3000	540		$(x^{(m)}, y^{(m)})$
	1985	300		
	1534	315		
30%	1427	199	Test Set	$(x^{(1)}_{test}, y^{(1)}_{test})$
	1380	212		$(x^{(2)}_{test}, y^{(2)}_{test})$
	1494	243		$\vdots$
				$(x^{(m_{test})}_{test}, y^{(m_{test})}_{test})$

In which m_{test} = *No. of test examples*, $(x^{(1)}_{test}), (y^{(1)}_{test})$. In general, if there is any sort of order to the data, it is best to ***randomly shuffle*** the data.

Now, we have a short procedure for testing & training *Linear Regression:*

- ***Learn*** parameter θ from training data *(minimizing training error $J(\theta)$.)*
- ***Compute*** test set error:

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

which is simply the averaged-squared-error of the hypothesis function, using the test examples. This can also be applied to *logistic regression: (Classification)*

- ***Learn*** parameter θ from training data *(minimizing training error $J(\theta)$.)*
- ***Compute*** test set error:

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

This is the same as before, but we've now used the m_{test} dataset. In addition, we also have:

– ***Misclassification*** error (0/1 misclassification error)

Here, we find the error of a prediction, given a particular label:

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

The first case corresponds to the case in which something was mislabeled. Then, we can define:

$$\textit{Test Error:} \ = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} \left(h_\theta(x_{test}^{(i)} - y_{test}^{(i)} \right)$$

6.1.2. *Model Selection.*

Suppose you're left to decide what degree of polynomial to fit to a data set. So that what features to include that gives you a learning algorithm. Or suppose you'd like to choose the regularization parameter longer for learning algorithm. How do you do that? These are basic *model selection* problems.

As shown above with overfitting, the training set error is not a great indicator. Just because a learning algorithm fits a training set well, that doesn't mean it's a good hypothesis. More generally, this is why the training set's error is not a good predictor for how well the hypothesis will do on new example.

Example 6.2 : *Model Selection*

Let's say that we have 10 different models to choose from *What degree of polynomial should we pick?* In addition, we also want an estimate as to how well our fitted hypothesis will generalize to new examples.

Now, looking at our 10 different models:

$$\begin{array}{c|llll} 1 & h_\theta(x) = \theta_0 + \theta_1 x & \to \ \theta^{(1)} & \to & J_{test}(\theta^{(1)}) \\ 2 & h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 & \to \ \theta^{(2)} & \to & J_{test}(\theta^{(2)}) \\ 3 & h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_3 x^3 & \to \ \theta^{(3)} & \to & J_{test}(\theta^{(3)}) \\ \vdots & \vdots & \quad \vdots & & \quad \vdots \\ 10 & h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10} & \to \ \theta^{(10)} & \to & J_{test}(\theta^{(10)}) \end{array}$$

- First, we minimize the training error, $\min_\theta J(\theta)$, for each hypothesis function to obtain a parameter vector θ
- Next, we measure the performance on the test set by computing the test set error J_{test}
- Then, we can choose the model with the lowest test set error, say $J(\theta^{(5)})$. This *might* give us an indication as to how well the model will generalize.

Problem: $J_{test}(\theta^{(5)})$ is likely to be a very optimistic estimate of the generalization error, since we have effectively chosen our model order (d=degree of polynomial) based on our test set.

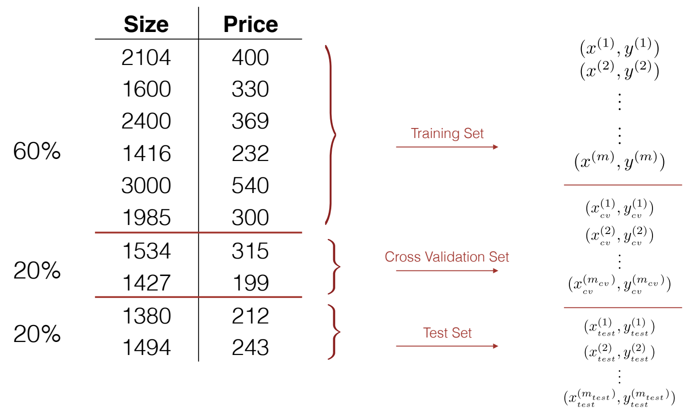
Hence, it is no longer fair to evaluate the hypothesis on this test set, because I fit my parameters to it.

Solution: Split our dataset into 3 parts:

– *(60%)* Training Set

– *(20%)* Cross Validation (CV) Set

– *(20%)* Test Set



Size	Price
2104	400
1600	330
2400	369
1416	232
3000	540
1985	300
1534	315
1427	199
1380	212
1494	243

60%
20%
20%
Training Set
Cross Validation Set
Test Set
$(x^{(1)}, y^{(1)})$
$(x^{(2)}, y^{(2)})$
$(x^{(m)}, y^{(m)})$
$(x_{cv}^{(1)}, y_{cv}^{(1)})$
$(x_{cv}^{(2)}, y_{cv}^{(2)})$
$(x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$
$(x_{test}^{(1)}, y_{test}^{(1)})$
$(x_{test}^{(2)}, y_{test}^{(2)})$
$(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

Now, we have ***three error types:***

Training Error:

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

Cross Validation Error:

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

Test Error:

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

Example 6.3 : *Model Selection 2*

Now, revisiting the model selection problem, we do the following

$$\begin{array}{l|llll}
1 & h_\theta(x) = \theta_0 + \theta_1 x & \to \ \theta^{(1)} & \to & J_{cv}(\theta^{(1)}) \\
2 & h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 & \to \ \theta^{(2)} & \to & J_{cv}(\theta^{(2)}) \\
3 & h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_3 x^3 & \to \ \theta^{(3)} & \to & J_{cv}(\theta^{(3)}) \\
\vdots & \vdots & \quad \vdots & & \quad \vdots \\
10 & h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10} & \to \ \theta^{(10)} & \to & J_{cv}(\theta^{(10)})
\end{array}$$

- First, we ***minimize the cost function***, $\min_\theta J(\theta)$, which provides some parameter vector, θ.
- Next, we measure the performance on the ***cross validation*** set, $J_{cv}(\theta)$
- Then we ***choose hypothesis*** function with the lowest cross validation error.

For the above example, let's say that $J_{cv}(\theta^{(4)})$, was the lowest (4th order model), so we pick: $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_4 x^4$.

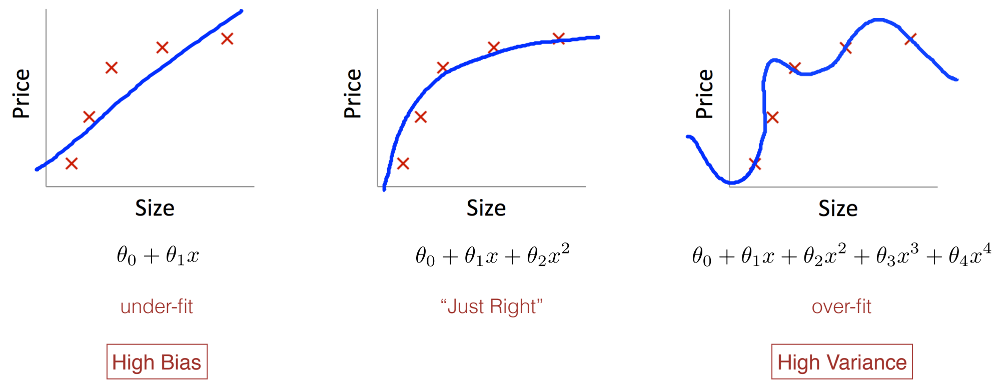
- Estimate generalization error, the ***test set error***, $J_{test}(\theta^{(4)})$.

Key takeaway: It is best to use a cross validation set to choose model.

6.2. Bias vs. Variance.

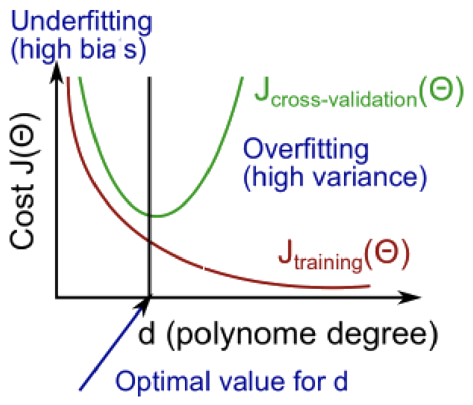
Often times, if the ML algorithm is performing poorly, this is due to an underfitting or overfitting problem.

Recall: 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.



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

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

$$q_1'' = q_2''$$
$$-k_1\left(\frac{dT}{dx}\right)_1 = -k_2\left(\frac{dT}{dx}\right)_2$$

6.2.1. *Regularization*.

You've seen how regularization can help prevent over-fitting. But how does it affect the bias and variances of a learning algorithm? We will now go deeper into this topic.

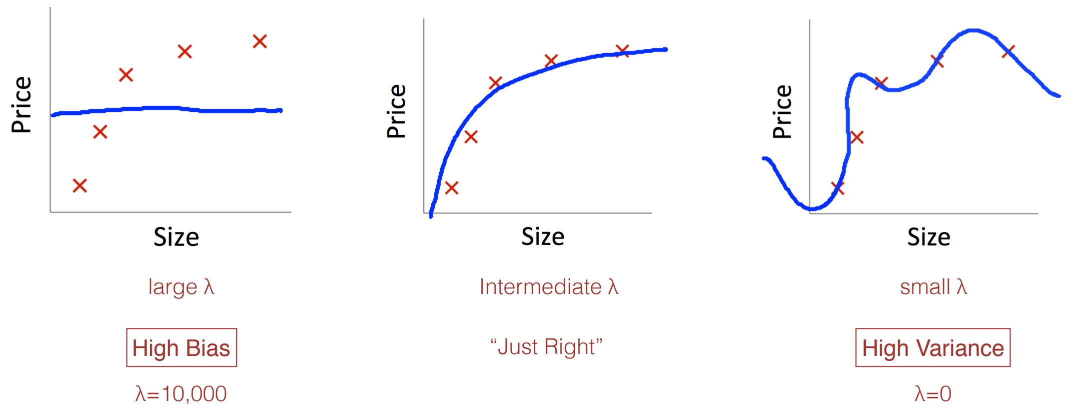
Example 6.4 : *Model Selection*

Suppose we're fitting a high-order polynomial, like that shown below, that uses regularization to prevent overfitting:

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$
$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)} - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

Note: As usual, the regularizations comes from $j = 1 \to m$, rather than $j = 0 \to m$.

Let's consider three λ-cases:



- ***Large*** λ ($\lambda \approx 10,000$): In this case, all of the parameters would be heavily penalized: $\theta_1, \theta_2, \ldots, \theta_n \to\approx 0$. Thus the hypothesis function, $h_\theta(x) \approx \theta_0$, or a constant value.
- ***Small*** λ ($\lambda \approx 0$): In this case, given that we're fitting a high order polynomial, this is a usual over-fitting setting, or *high variance.*
- ***Intermediate*** λ: It's only if we have some intermediate value of longer that is neither too large nor too small that we end up with parameters data that give us a reasonable fit to this data.

Question: How can we automatically choose our regularization parameter, λ?,

Using the same functions from above, we add the general training, cross validation, and test set cost functions discussed before - each of which *do not* have a regularization parameter.

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

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}\Big(h_\theta(x^{(i)}_{cv}) - y^{(i)}_{cv}\Big)^2$$

$$J_{test}(\theta) = \frac{1}{2m_{test}}\sum_{i=1}^{m_{test}}\Big(h_\theta(x^{(i)}_{test}) - y^{(i)}_{test}\Big)^2$$

Now, as with choosing the order of polynomial above, we try several cases of λ.

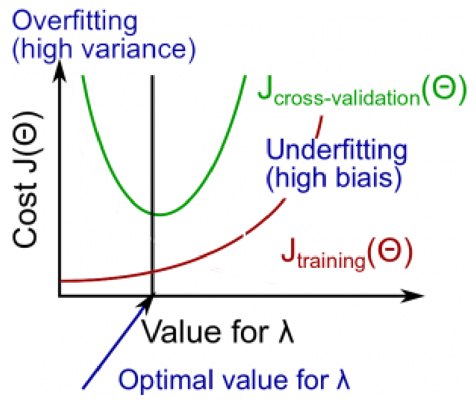
$$\begin{array}{c|llllllll}
1 & \lambda = 0.00 & \rightarrow & \min\limits_\theta J(\theta) & \rightarrow & \theta^{(1)} & \rightarrow & J_{cv}(\theta^{(1)}) \\
2 & \lambda = 0.01 & \rightarrow & \min\limits_\theta J(\theta) & \rightarrow & \theta^{(2)} & \rightarrow & J_{cv}(\theta^{(2)}) \\
3 & \lambda = 0.02 & \rightarrow & \min\limits_\theta J(\theta) & \rightarrow & \theta^{(3)} & \rightarrow & J_{cv}(\theta^{(3)}) \\
4 & \lambda = 0.04 & \rightarrow & \min\limits_\theta J(\theta) & \rightarrow & \theta^{(4)} & \rightarrow & J_{cv}(\theta^{(4)}) \\
5 & \lambda = 0.08 & \rightarrow & \min\limits_\theta J(\theta) & \rightarrow & \theta^{(5)} & \rightarrow & J_{cv}(\theta^{(5)}) \\
\vdots & \vdots & & \vdots & & \vdots & & \vdots \\
12 & \lambda = 10 & \rightarrow & \min\limits_\theta J(\theta) & \rightarrow & \theta^{(12)} & \rightarrow & J_{cv}(\theta^{(12)})
\end{array}$$

Then, we repeat the same procedure from before:

- First, we ***minimize the cost function***, $\min\limits_\theta J(\theta)$, which provides some parameter vector, θ.
- Now evaluate these parameters on the ***cross validation*** set, $J_{cv}(\theta)$
- Then we ***choose hypothesis*** function with the lowest cross validation error.

For the sake of this example, lets choose $J_{cv}(\theta^{(5)})$ as optimal. Then we calculate, and report, the test set error: $J_{test}(\theta^{(5)})$.

Now, lets explore how *cross validation* and *training error* vary as we vary the regularization parameter, λ. If we plot $J_{train}(\theta)$ and $J_{cv}(\theta)$, we find:



Similar to what we saw earlier, if we are assumed to have a *high-order polynomial*, then:

– ***Small*** λ: then we're not using much regularization, and run larger risk of overfitting, or a high variance - hence, $J_{cv}(\theta)$ will be high and $J_{train}(\theta)$ low, as fitting the data will be easy.

– ***Large*** λ: then we run the higher risk of having a biased problem. So $J_{train}(\theta)$ will tend to increase when λ increases, because a large value of λ corresponds to high bias where you might not even fit your trainings that well.

6.2.2. *Learning Curves*.

Now, we take a look at a useful tool in diagnosing ML issues: *Learning Curves*. These are often very useful to plot. If either you wanted to sanity check that your algorithm is working correctly, or if you want to improve the performance of the algorithm. More specifically, learning curves may be used to diagnose bias or variance in a problem.

Definition 2. ***Learning Curve*** Plot $J_{train}(\theta)$ and the average squared error of the training set, or J_{cv}, against the number of training examples, m.

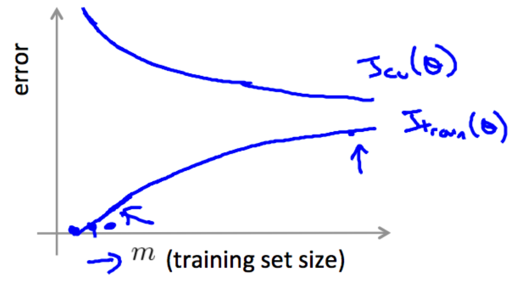
Example 6.5 : *Learning Curves*

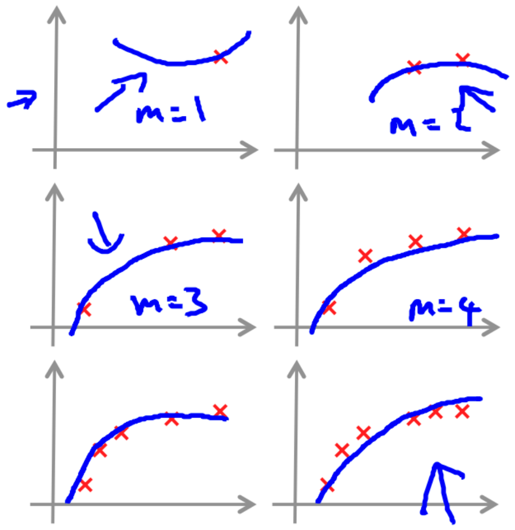
Provided with a training set and cross validation set of size m, we have cost functions:

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

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

Plotting the training error with increasing points, m (not using regularization):





In general, the training set fits nicely for a small number of data points, but as m increases the error rises. The opposite is true for the CV set: few points lead to high error, while more points will lead to an equilibrium.

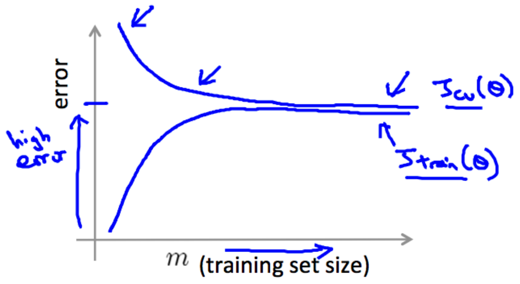
However, training an algorithm on a very few number of data points (such as 1, 2 or 3) will easily have 0 errors because we can always find a quadratic curve that touches exactly those number of points. Hence:

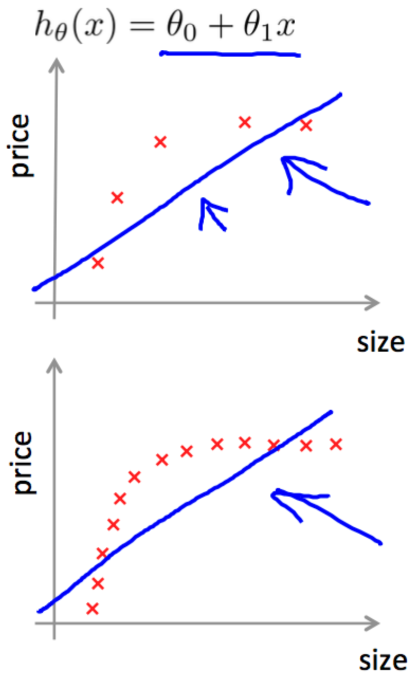
– As the training set gets larger, the error for a quadratic function increases.

– The error value will plateau out after a certain m, or training set size.

Now, if we take a look at a *high bias* case:

High Bias:



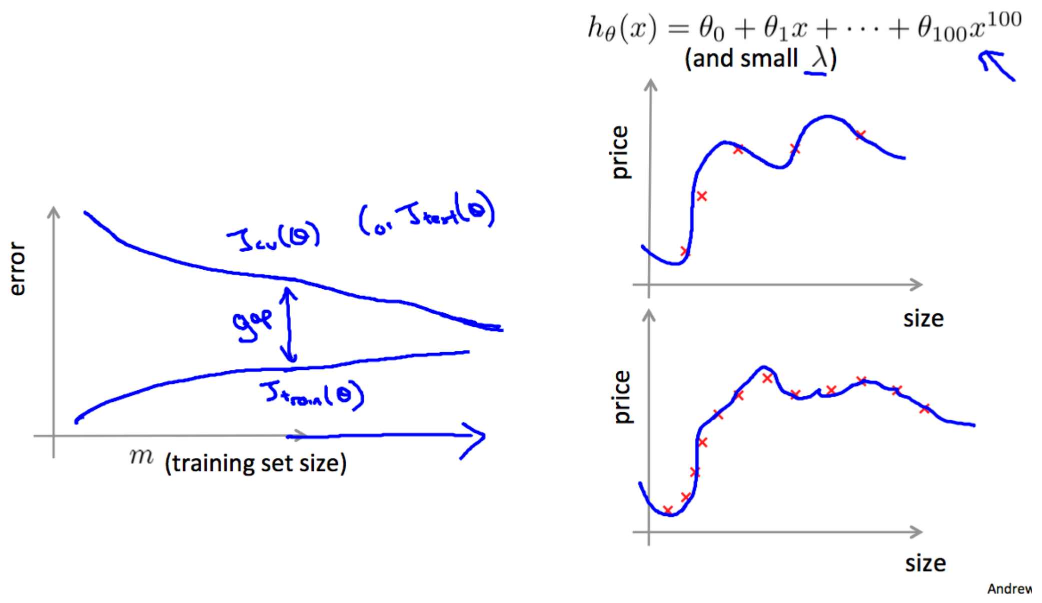


In this case, both perform quite poorly. Adding data will not help much.

– ***Low training set size:*** Causes $J_{train}(\theta)$ to be low and $J_{cv}(\theta)$ to be high.

- ***Large training set size:*** Causes $J_{train}(\theta)$ and $J_{cv}(\theta)$ to be high, with $J_{train}(\theta) \approx J_{cv}(\theta)$.

High Variance:



In this case, adding data will help.

- ***Low training set size:*** Causes $J_{train}(\theta)$ to be low and $J_{cv}(\theta)$ to be high.
- ***Large training set size:*** Causes $J_{train}(\theta)$ 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.

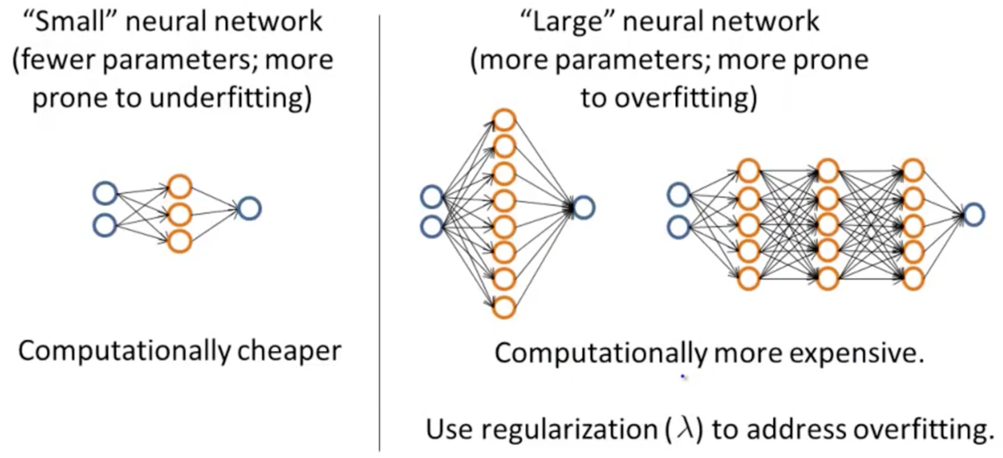
6.2.3. *What to do next? & Neural Networks.*

From the initial example, we had several options to improve a learning algorithm:

- Getting more training examples (*get more data*) $\rightarrow$ *fixes high variance*
- Trying smaller sets of features $\rightarrow$ *fixes high variance*
- Trying additional features $\rightarrow$ *fixes high bias*
- Trying polynomial features $\rightarrow$ *fixes high bias*
- Decreasing λ $\rightarrow$ *fixes high bias*
- Increasing λ $\rightarrow$ *fixes high variance*

Neural Networks and overfitting:

Returning to the neural networks, we have two basic scenarios:

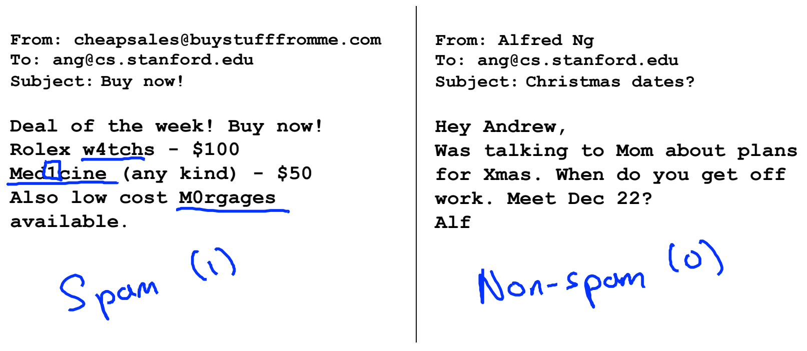

"Small" neural network
(fewer parameters; more prone to underfitting)
Computationally cheaper
"Large" neural network
(more parameters; more prone to overfitting)
Computationally more expensive.
Use regularization (λ) to address overfitting.

In most cases, the larger neural network is better, and then using regularization to address overfitting.

6.3. Machine Learning System Design.

Example 6.6 : *Spam Classification*

Lets say we want to build a spam classifier. Here are two typical examples of spam and not spam:



```
From: cheapsales@buystufffromme.com
To: ang@cs.stanford.edu
Subject: Buy now!

Deal of the week! Buy now!
Rolex w4tchs - $100
Med1cine (any kind) - $50
Also low cost M0rgages
available.
```

Spam (1)

```
From: Alfred Ng
To: ang@cs.stanford.edu
Subject: Christmas dates?

Hey Andrew,
Was talking to Mom about plans
for Xmas. When do you get off
work. Meet Dec 22?
Alf
```

Non-spam (0)

- Notice how spammers will deliberately misspell words, like Vincent with a 1, and mortgages with a zero.

Now, let's say that we have a labeled training set of some number of spam emails and some non-spam emails denoted with labels $y = 1$ or 0, *how do we build a classifier using supervised learning to distinguish between spam and non-spam?*

6.3.1. *Building a spam classifier.*

In order to apply *supervised learning*, the first decision we must make is *how do we want to represent x, that is, the features of the email?*

Given the features x and the labels y in our training set, we can then train a classifier, for example using *logistic regression.*

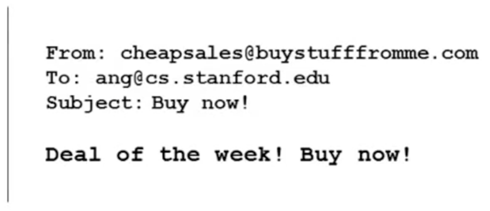
Here's one way to choose a set of features for our emails:

- ***Features:*** x - choose 100 words that are indicitive of spam / not spam. For example: *Deal, buy, discount, andrew, now, . . .*
- Next, ***sort the list*** of words alphabetically and check if any exist in the email sample.

Given an email, we can encode this into a *feature vector* as follows:

Feature Representation:

$$x = \begin{array}{c c} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 1 \\ \vdots \end{bmatrix} & \begin{array}{c} \textit{andrew} \\ \textit{buy} \\ \textit{deal} \\ \textit{discount} \\ \vdots \\ \textit{now} \\ \vdots \end{array} \end{array} \in \mathbb{R}^{100}$$



```
From: cheapsales@buystufffromme.com
To: ang@cs.stanford.edu
Subject: Buy now!

Deal of the week! Buy now!
```

More mathematically, x is defined by:

$$x_j = \begin{cases} 0 : \text{if word } j \text{ appears} \\ 1 : \text{otherwise} \end{cases}$$

Note: In practice, take most frequently occurring n words (10,000 to 50,000) in training set, rather than manually picking 100 words.

Now, the question becomes: *How to spend your time to make it have low error?*

- ***Collect lots of data*** *(this is a natural tendency).*
  + *E.g.* ‘Honeypot’ project.
- Develop sophisticated features based on ***email routing information*** (from e-mail header).
- Develop sophis3cated features for ***message body***, *e.g.* should “discount” and “discounts” be treated as the same word? How about “deal” and “Dealer”? Features about punctua3on?
- Develop sophis3cated algorithm to ***detect misspellings*** (ite.g. m0rtgage, med1cine, w4tches.)

HERE