# Week : 6

## Part One

**Debugging a learning algorithm:**

Suppose you have implemented regularized linear regression to predict housing prices.

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

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?

- Get more training examples
- Try smaller sets of features
- Try getting additional features
- Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, \text{etc.})$
- Try decreasing $\lambda$
- Try increasing $\lambda$

Andrew Ng

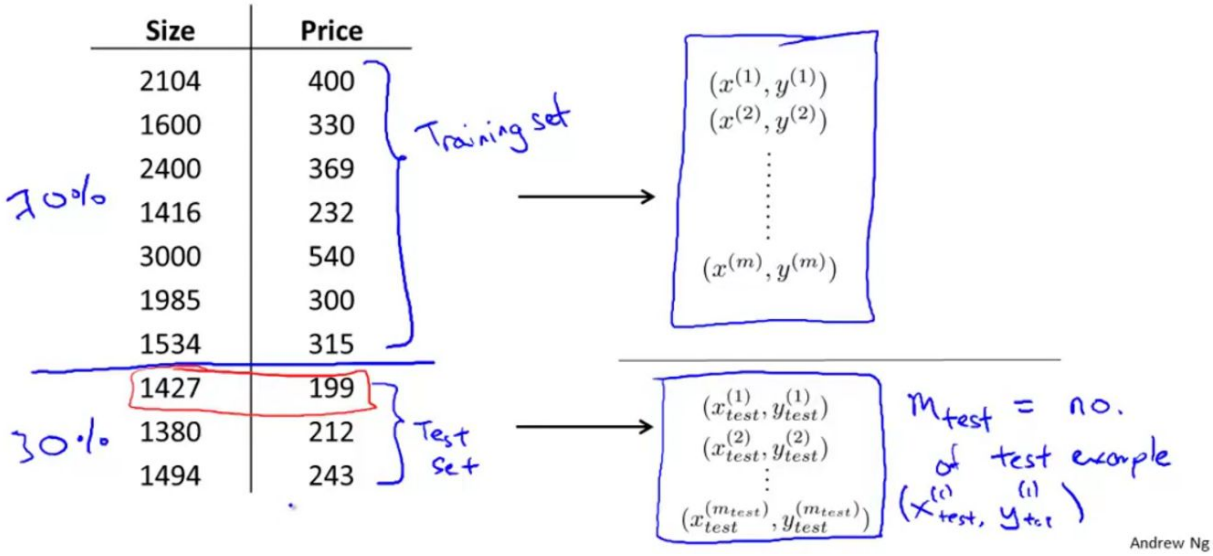
A hypothesis may have a low error for the training examples but still be inaccurate (because of overfitting). Thus, to evaluate a hypothesis, given a dataset of training examples, we can split up the data into two sets: a **training set** and a **test set**. Typically, the training set consists of 70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

The new procedure using these two sets is then:

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

## Evaluating your hypothesis

Dataset:




Andrew Ng

## Training/testing procedure for logistic regression

- Learn parameter $\theta$ from training data
- Compute test set error:

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

- Misclassification error (0/1 misclassification error):

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

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

Andrew Ng

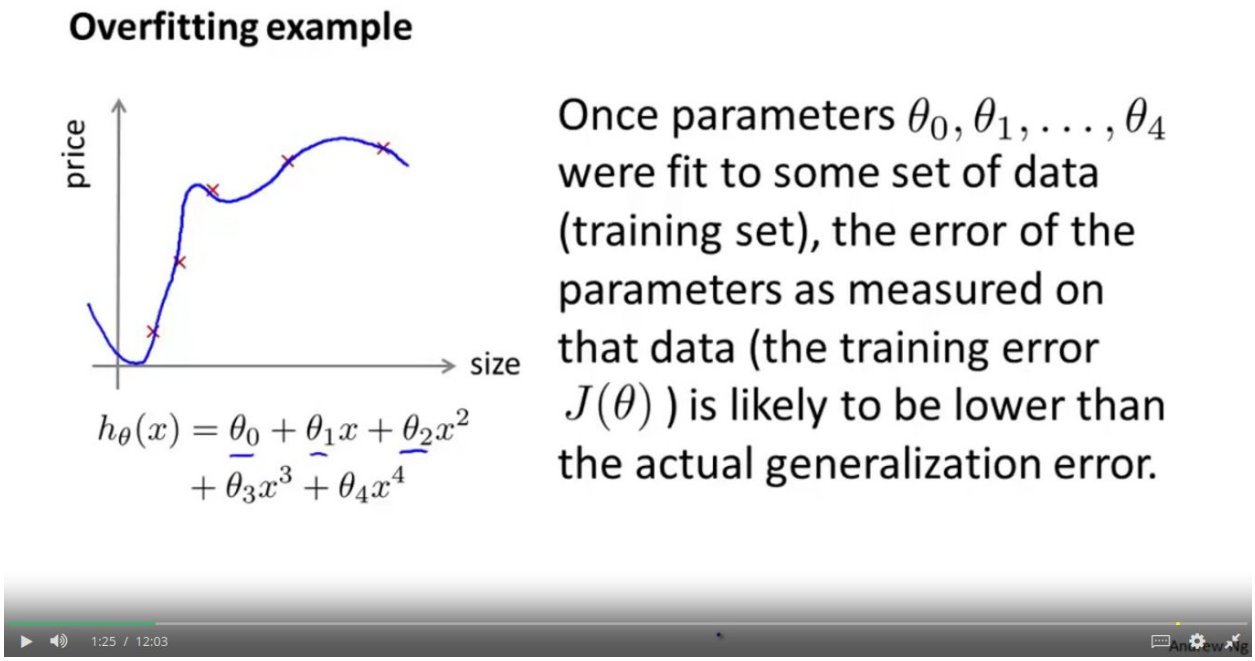
This gives us a binary 0 or 1 error result based on a misclassification. The average test error for the test set is:

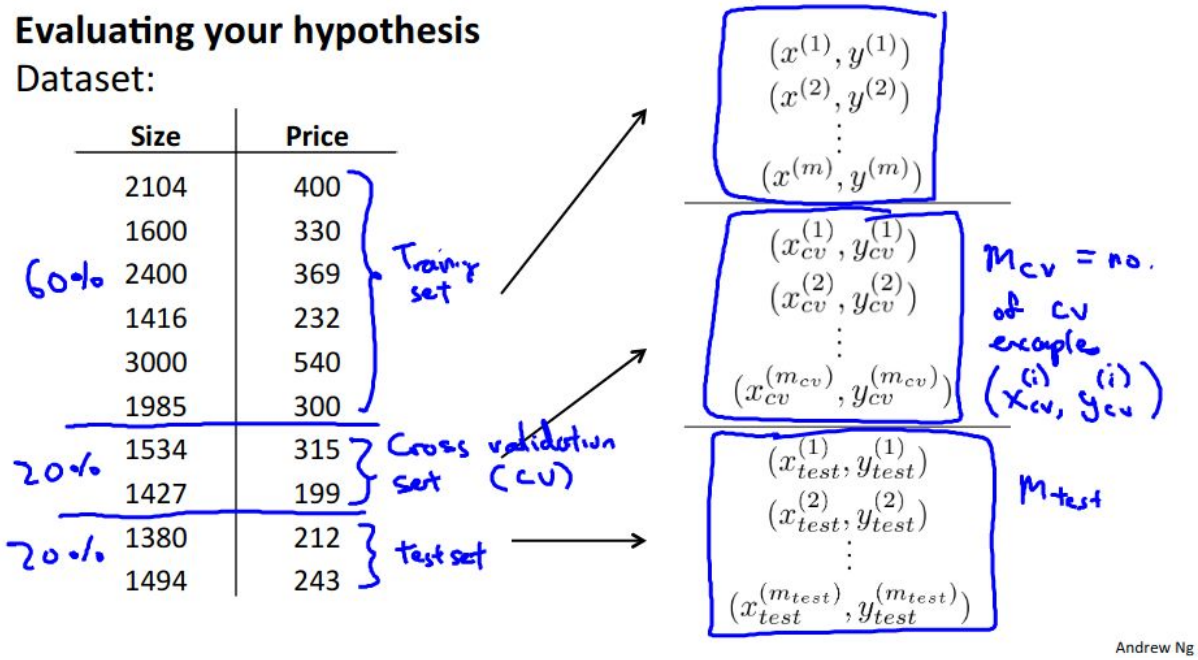
$$\text{Test Error} = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h_\Theta(x_{test}^{(i)}), y_{test}^{(i)})$$

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

# What degree of polynomial should we choose?

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.



First we need to divide our data training set into three portions : Training set, cross validation set and test set.

- 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.

**Train/validation/test error**

Training error:

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

Cross Validation error:

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

Test error:

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

Andrew Ng

**Model selection**

1. $h_\theta(x) = \theta_0 + \theta_1 x$ → $\min_\theta J(\theta)$ → $\theta^{(1)}$ → $J_{cv}(\theta^{(1)})$
2. $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$ → $\theta^{(2)}$ → $J_{cv}(\theta^{(2)})$
3. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_3 x^3$ → $\theta^{(3)}$ → $J_{cv}(\theta^{(3)})$

⋮

10. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10}$ → $\theta^{(10)}$ → $J_{cv}(\theta^{(10)})$

$d = 4$

Pick $\theta_0 + \theta_1 x_1 + \cdots + \theta_4 x^4$ ←

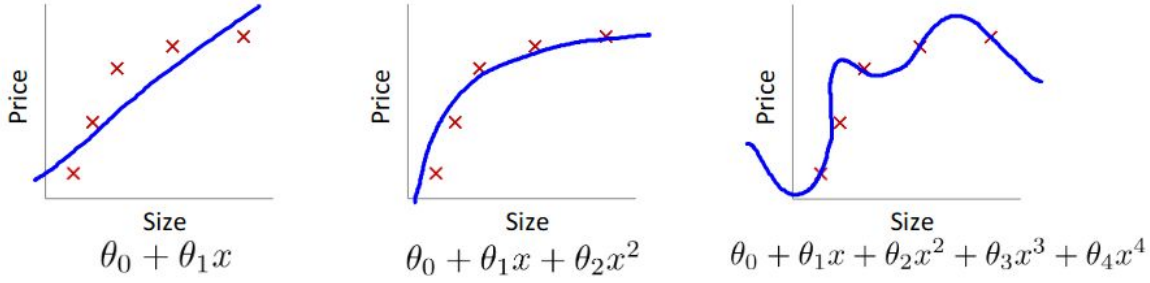
Estimate generalization error for test set $J_{test}(\theta^{(4)})$ ←

Andrew Ng

# Diagnosis bias vs variance :

We need to distinguish whether **bias** or **variance** is the problem contributing to bad predictions. High bias is underfitting and high variance is overfitting.

**Bias/variance**




High bias (underfit) $d = 1$

"Just right" $d = 2$

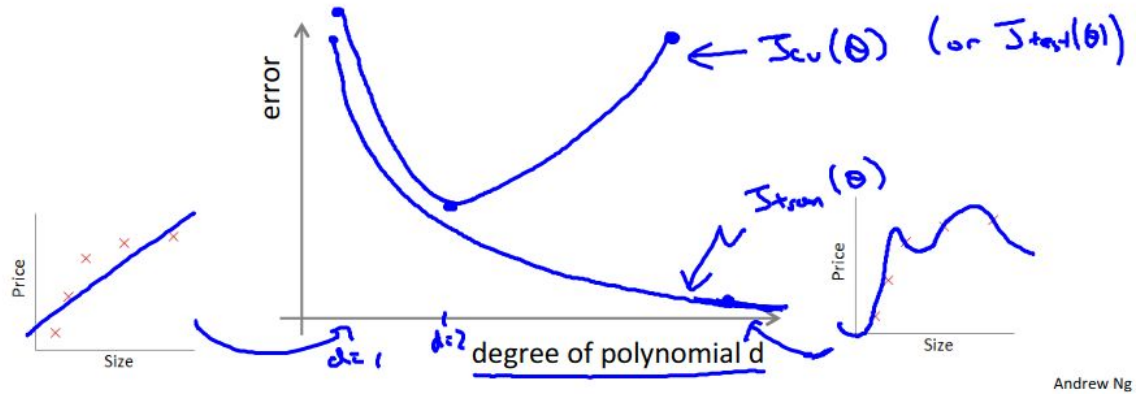
High variance (overfit) $d = 4$

Andrew Ng

## Bias/variance

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 $J_{test}(\theta)$)

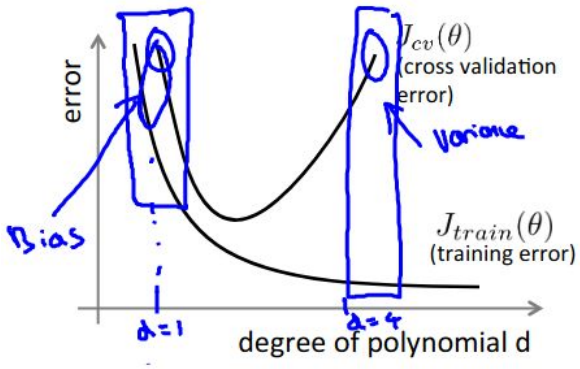



Andrew Ng

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.

## Diagnosing bias vs. 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
$J_{cv}(\theta) \approx J_{train}(\theta)$

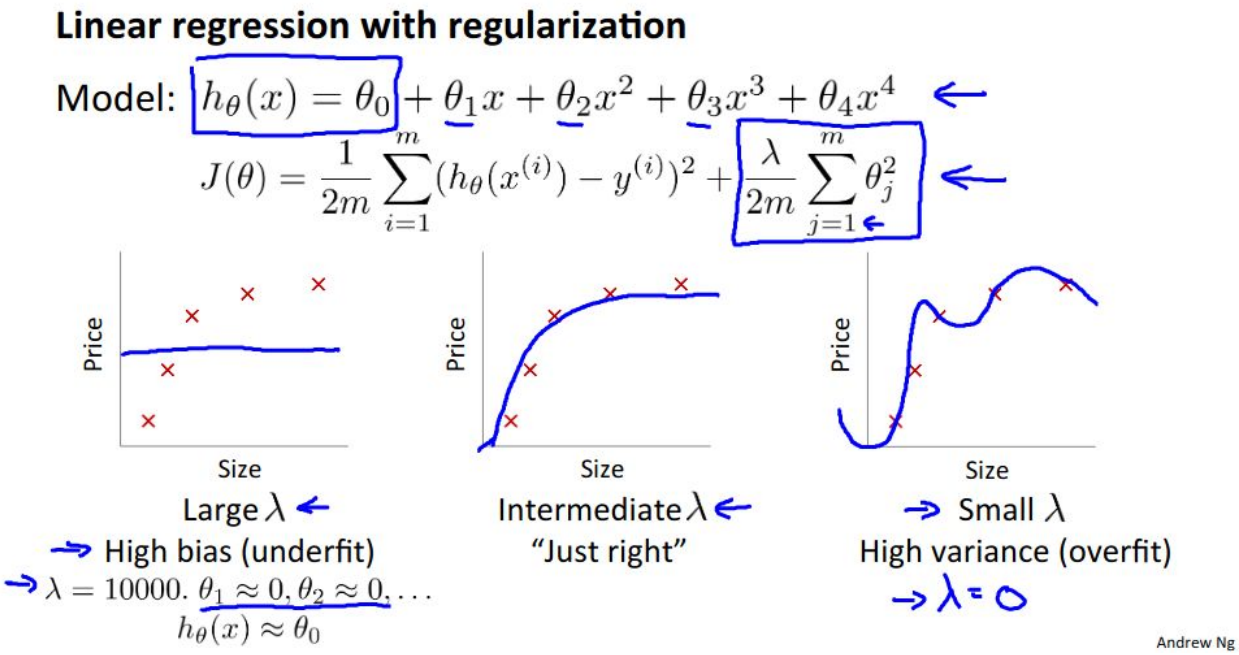
Variance (overfit):
→ $J_{train}(\theta)$ will be low
$J_{cv}(\theta) >> J_{train}(\theta)$

Andrew Ng

In an underfitting case, the equation will be very simple and the error difference of the training set and cross validation set will be nearly equal. But in an overfitting case, the

difference will be big. In overfitting case error of the training set will be low on the contrary for cross validation it will be high again.

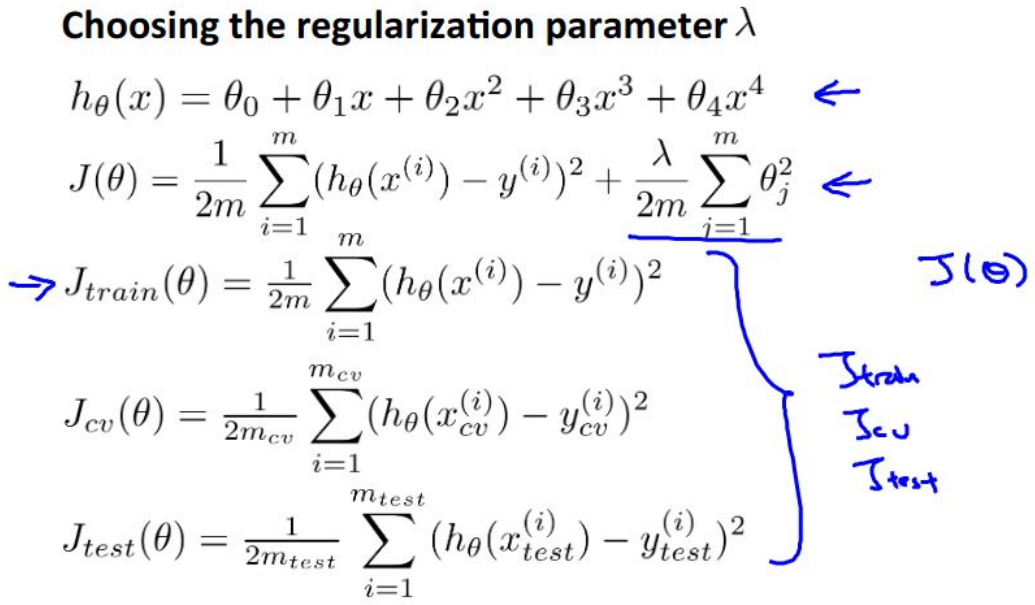
# Regularization and Bias/Variance

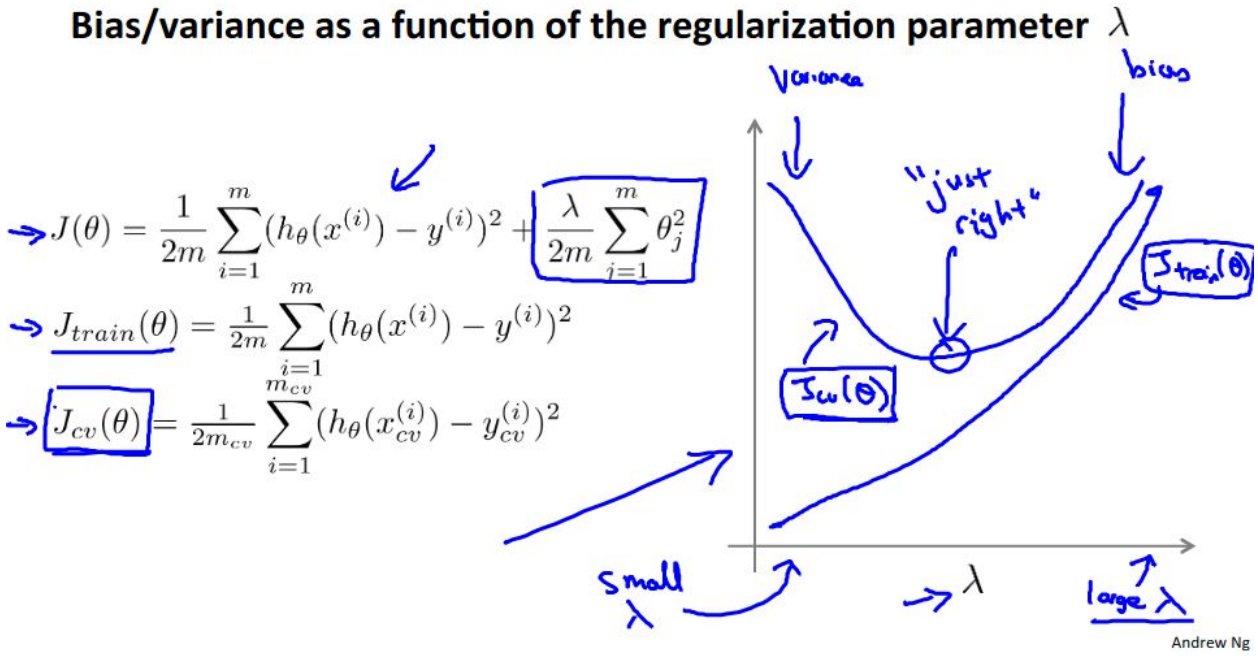



In the figure above, we see that as λ increases, our fit becomes more rigid. On the other hand, as λ approaches 0, we tend to over overfit the data. So how do we choose our parameter λ to get it 'just right' ? In order to choose the model and the regularization term λ, we need to:

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 λand for each λ\lambdaλ go through all the models to learn some Θ.
4. Compute the cross validation error using the learned Θ (computed with λ) on the( $J_{cv}$ (Θ)**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



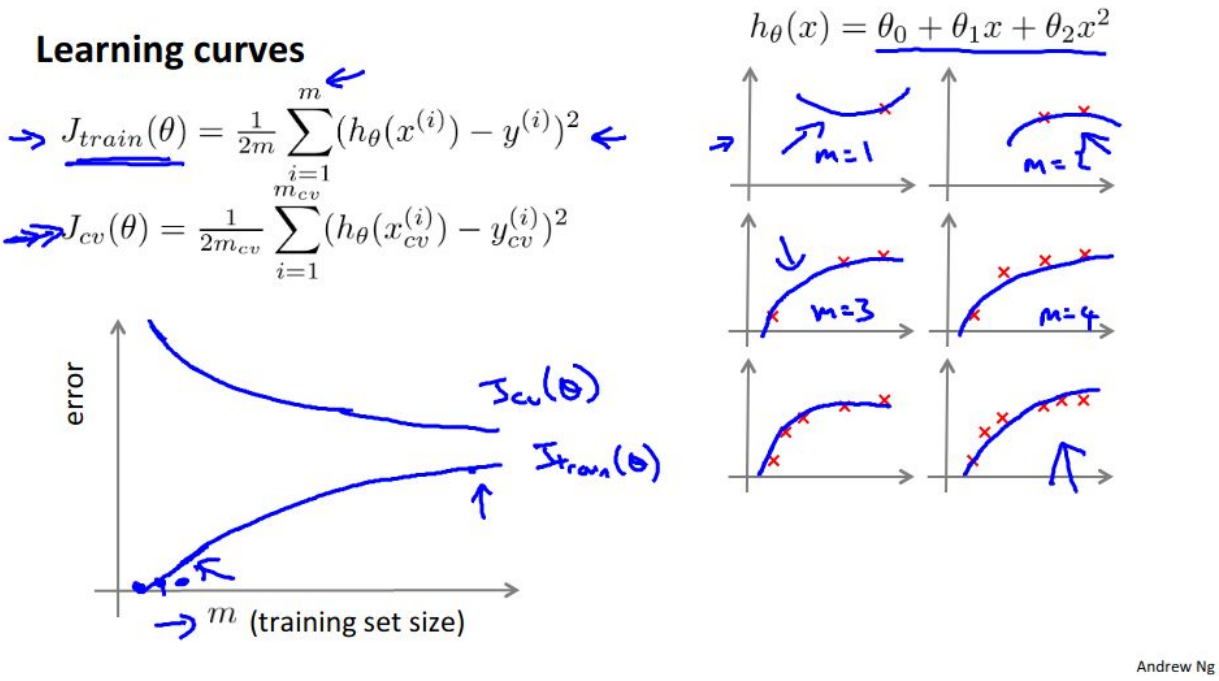

At first I was confused to see the graph as there is no lambda parameter in the train and cv function but they change with value lambda. It is because we get the min theta from the first equation and then apply it on the next two equations. As a result when the value

of lambda is low it's impact on the first equation is not so big and our train set will fit well but cross validation set will not. Similarly when lambda is too large, it's contribution to the first equation is too big. As a result the next two equations will show a large error without the lambda part.

# Learning curves

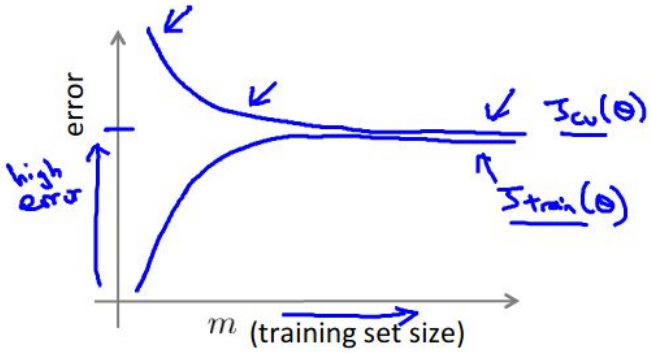
Training an algorithm on a very few number of data points (such as 1, 2 or 3) will easily have 0 errors because we can always find a quadratic curve that touches exactly those number of points. Hence:

- 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.

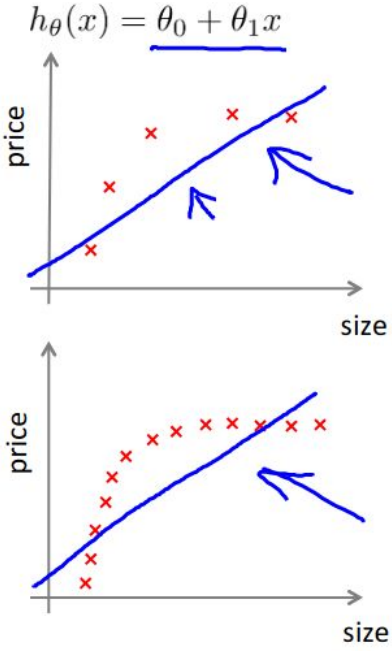



Andrew Ng

## High bias

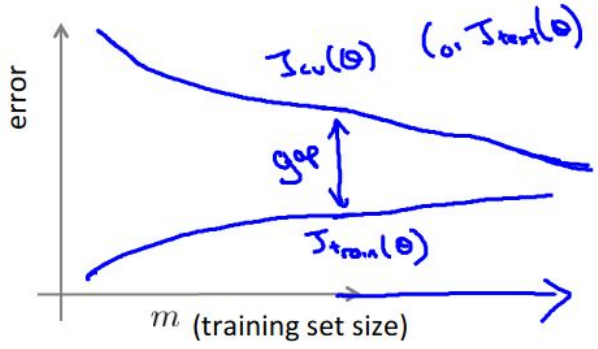



If a learning algorithm is suffering from high bias, getting more training data will not (by itself) help much.

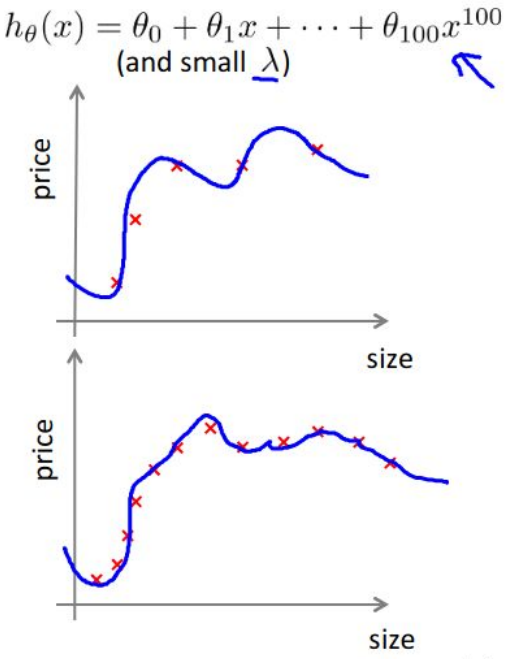



Andrew Ng

## High variance




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




Andrew Ng

# Summary of Part One:

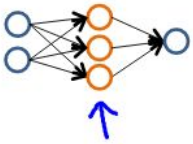
**Debugging a learning algorithm:**
Suppose you have implemented regularized linear regression to predict housing prices. However, when you test your hypothesis in a new set of houses, you find that it makes unacceptably large errors in its prediction. What should you try next?

- 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 $(x_1^2, x_2^2, x_1x_2, \text{etc})$ → fixes high bias.
- Try decreasing $\lambda$ → fixes high bias
- Try increasing $\lambda$ → fixes high variance

Andrew Ng

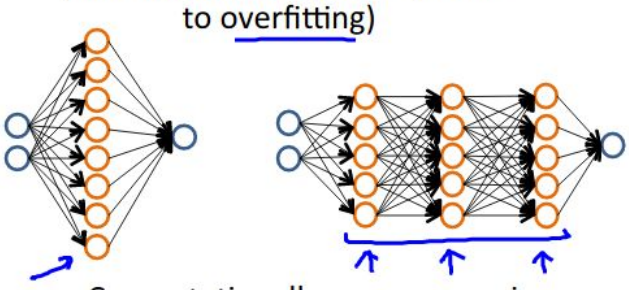
## Neural networks and overfitting

→ "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 ($\lambda$) to address overfitting.

$J_{cv}(\theta)$

Andrew Ng

## 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.

# Part Two : ML System Design

## Decide an email is spam or not:

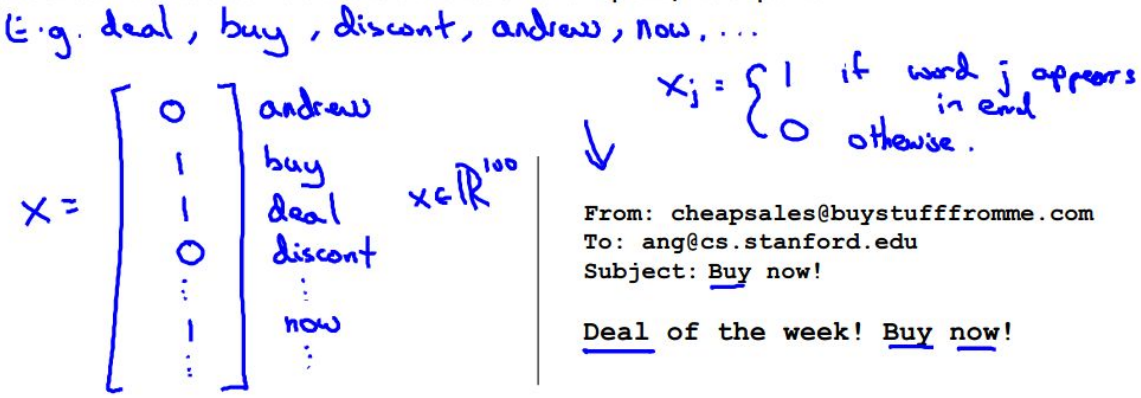
To decide if an email is spam or not, we have some way. One way is to collect so many emails and select which ones of those are spam and which are not. From the spam set we will collect a set of words and will use them to differentiate new email.

---

**Building a spam classifier**

Supervised learning. $x =$ features of email. $y =$ spam (1) or not spam (0).

Features $x$: Choose 100 words indicative of spam/not spam.




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

Andrew Ng

Besides this, we have some other way to do this :

**Building a spam classifier**

How to spend your time to make it have low error?

- Collect lots of data
  - E.g. “honeypot” project.
- Develop sophisticated features based on email routing information (from email header).
- Develop sophisticated features for message body, e.g. should “discount” and “discounts” be treated as the same word? How about “deal” and “Dealer”? Features about punctuation?
- Develop sophisticated algorithm to detect misspellings (e.g. m0rtgage, med1cine, w4tches.)

# Error analysis :

**Recommended approach**

- Start with a simple algorithm that you can implement quickly. Implement it and test it on your cross-validation data.
- Plot learning curves to decide if more data, more features, etc. are likely to help.
- Error analysis: Manually examine the examples (in cross validation set) that your algorithm made errors on. See if you spot any systematic trend in what type of examples it is making errors on.

**Error Analysis**

$m_{CV}$ = 500 examples in cross validation set
Algorithm misclassifies 100 emails.
Manually examine the 100 errors, and categorize them based on:
→ (i) What type of email it is — pharma, replica, steal passwords, ...
→ (ii) What cues (features) you think would have helped the algorithm classify them correctly.

Pharma: 12
Replica/fake: 4
→ Steal passwords: 53
Other: 31

→ Deliberate misspellings: 5
(m0rgage, med1cine, etc.)
→ Unusual email routing: 16
→ Unusual (spamming) punctuation: 32

Andrew Ng

It is very important to get error results as a single, numerical value. Otherwise it is difficult to assess your algorithm's performance. For example if we use stemming, which is the process of treating the same word with different forms (fail/failing/failed) as one word (fail), and get a 3% error rate instead of 5%, then we should definitely add it to our model. However, if we try to distinguish between uppercase and lowercase letters and end up getting a 3.2% error rate instead of 3%, then we should avoid using this new feature. Hence, we should try new things, get a numerical value for our error rate, and based on our result decide whether we want to keep the new feature or not.

**The importance of numerical evaluation**

Should discount/discounts/discounted/discounting be treated as the same word?
Can use "stemming" software (E.g. "Porter stemmer")
universe/university.
Error analysis may not be helpful for deciding if this is likely to improve performance. Only solution is to try it and see if it works.
Need numerical evaluation (e.g., cross validation error) of algorithm's performance with and without stemming.
Without stemming: 5% error With stemming: 3% error
Distinguish upper vs. lower case (Mom/mom): 3.2%

# Error matrix for skewed classes :

If we have some skewed or biased training sets then error rate can't help us to understand how much correct our algorithm is.

**Cancer classification example**

Train logistic regression model $h_\theta(x)$. ($y = 1$ if cancer, $y = 0$ otherwise)
Find that you got 1% error on test set.
(99% correct diagnoses)

Only 0.50% of patients have cancer.

→ skewed classes.

```
function y = predictCancer(x)
  -> y = 0; %ignore x!
return
```

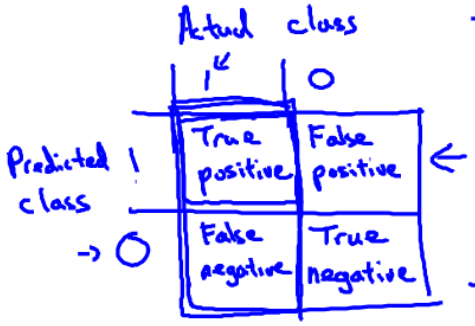
→ 0.5% error

→ 99.2% accuracy (0.8% error)
→ 99.5% accuracy (0.5% error)

To understand our algorithm's accuracy we have two formulas by which we can measure accuracy of our algorithm more correctly.

**Precision/Recall**

$y = 1$ in presence of rare class that we want to detect




→ **Precision**
(Of all patients where we predicted $y = 1$, what fraction actually has cancer?)

$$\frac{\text{True positives}}{\#\text{predicted positive}} = \frac{\text{True positive}}{\text{True pos + False pos}}$$

→ **Recall**
(Of all patients that actually have cancer, what fraction did we correctly detect as having cancer?)

$$\frac{\text{True positives}}{\#\text{actual positives}} = \frac{\text{True positives}}{\text{True pos + False neg}}$$

y = 0
recall = 0

# Trading of Precision and Recall :

Sometimes we need to tell our client if their test is negative or positive correctly. In that case we can get help from precision and recall. When we want to be confident then we will choose high precision and lower recall. Again when we want to avoid false negative result we will choose higher recall and lower precision.

**Trading off precision and recall**

→ Logistic regression: $0 \le h_\theta(x) \le 1$

Predict 1 if $h_\theta(x) \ge 0.5$ 0.7 0.9 0.3

Predict 0 if $h_\theta(x) < 0.5$ 0.7 0.9 0.3

→ Suppose we want to predict $y = 1$ (cancer) only if very confident.

→ Higher precision, lower recall

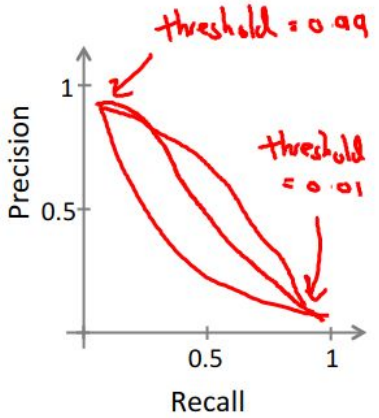
→ Suppose we want to avoid missing too many cases of cancer (avoid false negatives).

→ Higher recall, lower precision.

More generally: Predict 1 if $h_\theta(x) \ge$ threshold

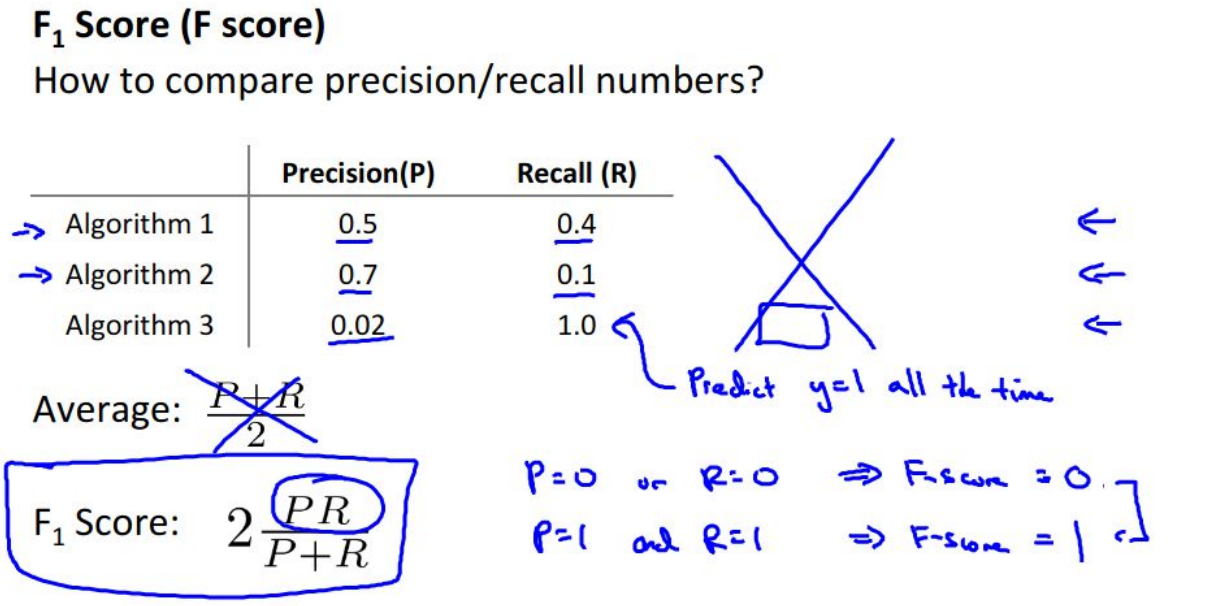
→ precision = $\frac{\text{true positives}}{\text{no. of predicted positive}}$

→ recall = $\frac{\text{true positives}}{\text{no. of actual positive}}$




Andrew Ng

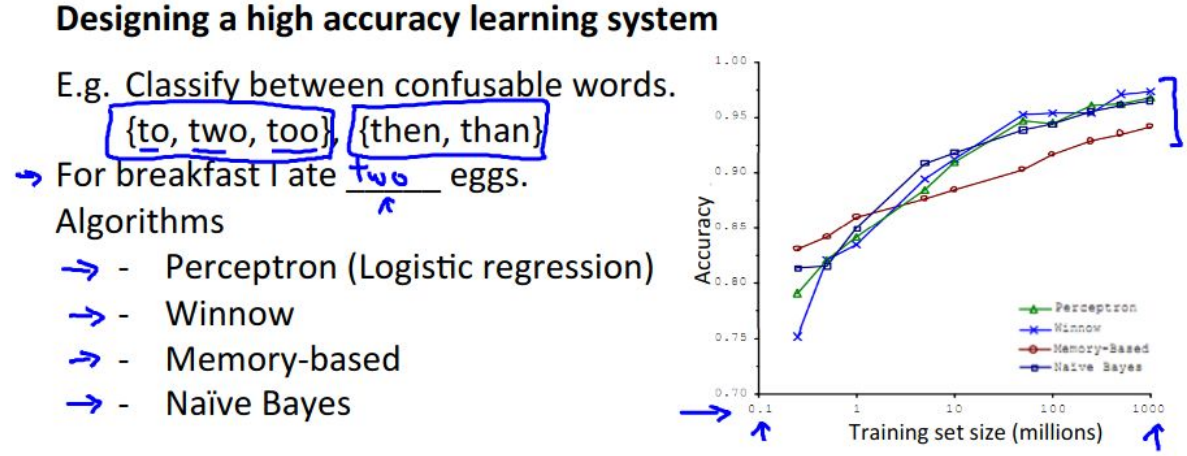
We need to choose one algorithm over others. In that case, F score will help us. We will apply the f score formula on cross validation sets and pick that algorithm with maximum f score




Andrew Ng

# Data for Machine learning :

With small training set different algorithms act significantly differently. But with large number of training set different algorithms can predict with nearly same training error.




**"It's not who has the best algorithm that wins.**
**It's who has the most data."**

[Banko and Brill, 2001]

**Large data rationale**

→ Use a learning algorithm with many parameters (e.g. logistic regression/linear regression with many features; neural network with many hidden units). low bias algorithms. ←

→ $J_{train}(\theta)$ will be small.

Use a very large training set (unlikely to overfit) low variance ←

→ $J_{train}(\theta) \approx J_{test}(\theta)$

$J_{test}(\theta)$ will be small