

Machine Learning

# Introduction

---

# Welcome

# Linear Algebra

Practice Quiz, 5 questions

**5/5 points (100%)**



**Congratulations! You passed!**

[Next Item](#)

# Linear Algebra

Practice Quiz, 5 questions

1 / 1  
point

5/5 points (100%)

1.

Let two matrices be

$$A = \begin{bmatrix} 4 & 3 \\ 6 & 9 \end{bmatrix}, \quad B = \begin{bmatrix} -2 & 9 \\ -5 & 2 \end{bmatrix}$$

What is  $A + B$ ?

$$\begin{bmatrix} 2 & 12 \\ 1 & 11 \end{bmatrix}$$

Correct

To add two matrices, add them element-wise.

$$\begin{bmatrix} 2 & 9 \\ 1 & 2 \end{bmatrix}$$

$$\begin{bmatrix} 6 & 12 \\ 11 & 11 \end{bmatrix}$$

$$\begin{bmatrix} 6 & -6 \\ 11 & 7 \end{bmatrix}$$

---

# Linear Algebra

Practice Quiz, 5 questions

1 / 1  
point

5/5 points (100%)

2.

Let  $x = \begin{bmatrix} 8 \\ 2 \\ 5 \\ 1 \end{bmatrix}$

What is  $2 * x$ ?

$\begin{bmatrix} 16 & 4 & 10 & 2 \end{bmatrix}$

$\begin{bmatrix} 4 & 1 & \frac{5}{2} & \frac{1}{2} \end{bmatrix}$

$\begin{bmatrix} 16 \\ 4 \\ 10 \\ 2 \end{bmatrix}$

Correct

To multiply the vector  $x$  by 2, take each element of  $x$  and multiply that element by 2.

$\begin{bmatrix} 4 \\ 1 \\ \frac{5}{2} \\ \frac{1}{2} \end{bmatrix}$

# Linear Algebra

Practice Quiz, 5 questions

1 / 1  
point

5/5 points (100%)

3.

Let  $u$  be a 3-dimensional vector, where specifically

$$u = \begin{bmatrix} 3 \\ 5 \\ 1 \end{bmatrix}$$

What is  $u^T$ ?

$\begin{bmatrix} 1 & 5 & 3 \end{bmatrix}$

$\begin{bmatrix} 3 & 5 & 1 \end{bmatrix}$

Correct

$\begin{bmatrix} 1 \\ 5 \\ 3 \end{bmatrix}$

$\begin{bmatrix} 3 \\ 5 \\ 1 \end{bmatrix}$



1 / 1  
point

4.

Let  $u$  and  $v$  be 3-dimensional vectors, where specifically  
**Linear Algebra**

Practice Quiz, 5 questions

**5/5 points (100%)**

$$u = \begin{bmatrix} 1 \\ 3 \\ -1 \end{bmatrix}$$

and

$$v = \begin{bmatrix} 2 \\ 2 \\ 4 \end{bmatrix}$$

What is  $u^T v$ ?

(Hint:  $u^T$  is a

1x3 dimensional matrix, and  $v$  can also be seen as a 3x1

matrix. The answer you want can be obtained by taking

the matrix product of  $u^T$  and  $v$ .) Do not add brackets to your answer.

4

**Correct Response**

# Linear Algebra

Practice Quiz, 5 questions

1 / 1  
point

5/5 points (100%)

5.

Let A and B be 3x3 (square) matrices. Which of the following must necessarily hold true? Check all that apply.



If  $v$  is a 3 dimensional vector, then  $A * B * v$  is a 3 dimensional vector.



**Correct**

Since A and B are both 3x3 matrices,  $A * B$  is 3x3 matrix. Thus,  $(A * B) * v$  is a 3x3 matrix times a  $3 \times 1$  matrix (since  $v$  is a 3 dimensional vector, and thus also a 3x1 matrix), and the result gives a 3x1 vector.



If  $C = A * B$ , then C is a 6x6 matrix.



**Un-selected is correct**



$A + B = B + A$



**Correct**

We add matrices element-wise. So, this must be true.



$A * B * A = B * A * B$



**Un-selected is correct**

---

# Linear Algebra

Practice Quiz, 5 questions

**5/5 points (100%)**





Apple - iPhoto - New full-Sc X

www.apple.com/ilife/iphoto/

Store Mac iPod iPhone iPad iTunes Support

iLife '11

iPhoto iMovie GarageBand Video Showcase Resources Upgrade Now

 iPhoto '11

From your Facebook Wall to your coffee table to your best friend's inbox (or mailbox). Do more with your photos than you ever thought possible. And do it all in one place. iPhoto.

 Watch the iPhoto video ▶



What's New in iPhoto What is iPhoto?

Andrew Ng



# Machine Learning

- Grew out of work in AI
- New capability for computers

## Examples:

- Database mining

Large datasets from growth of automation/web.

E.g., Web click data, medical records, biology, engineering

- Applications can't program by hand.

E.g., Autonomous helicopter, handwriting recognition, most of Natural Language Processing (NLP), Computer Vision.

# Machine Learning

- Grew out of work in AI

- Examples

- Examples



ig  
host of

# Machine Learning

- Grew out of work in AI
- New capability for computers

## Examples:

- Database mining

Large datasets from growth of automation/web.

E.g., Web click data, medical records, biology, engineering

- Applications can't program by hand.

E.g., Autonomous helicopter, handwriting recognition, most of Natural Language Processing (NLP), Computer Vision.

# Machine Learning

- Grew out of work in AI
- New capability for computers

## Examples:

- Database mining

Large datasets from growth of automation/web.

E.g., Web click data, medical records, biology, engineering

- Applications can't program by hand.

E.g., Autonomous helicopter, handwriting recognition, most of Natural Language Processing (NLP), Computer Vision.

- Self-customizing programs

E.g., Amazon, Netflix product recommendations

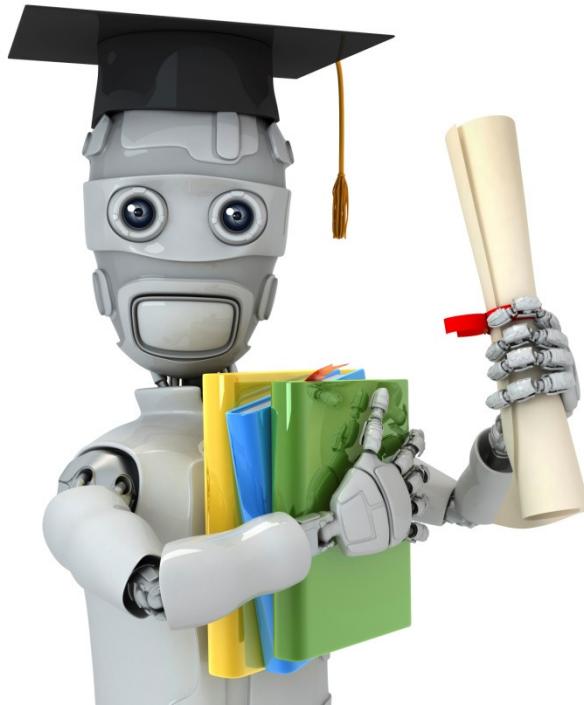
# Machine Learning

- Grew out of work in AI
- New capability for computers

## Examples:

- Database mining
  - Large datasets from growth of automation/web.  
E.g., Web click data, medical records, biology, engineering
- Applications can't program by hand.
  - E.g., Autonomous helicopter, handwriting recognition, most of Natural Language Processing (NLP), Computer Vision.
- Self-customizing programs
  - E.g., Amazon, Netflix product recommendations
- Understanding human learning (brain, real AI).





Machine Learning

# Introduction

---

# What is machine learning

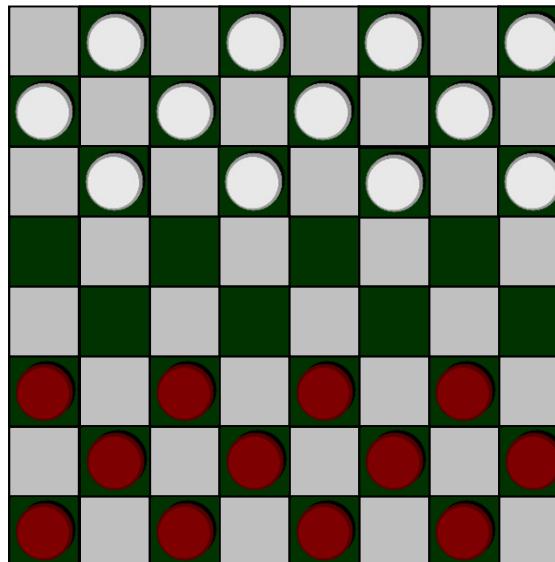
# Machine Learning definition

# Machine Learning definition

- Arthur Samuel (1959). Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.

# Machine Learning definition

- Arthur Samuel (1959). Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.



# Machine Learning definition

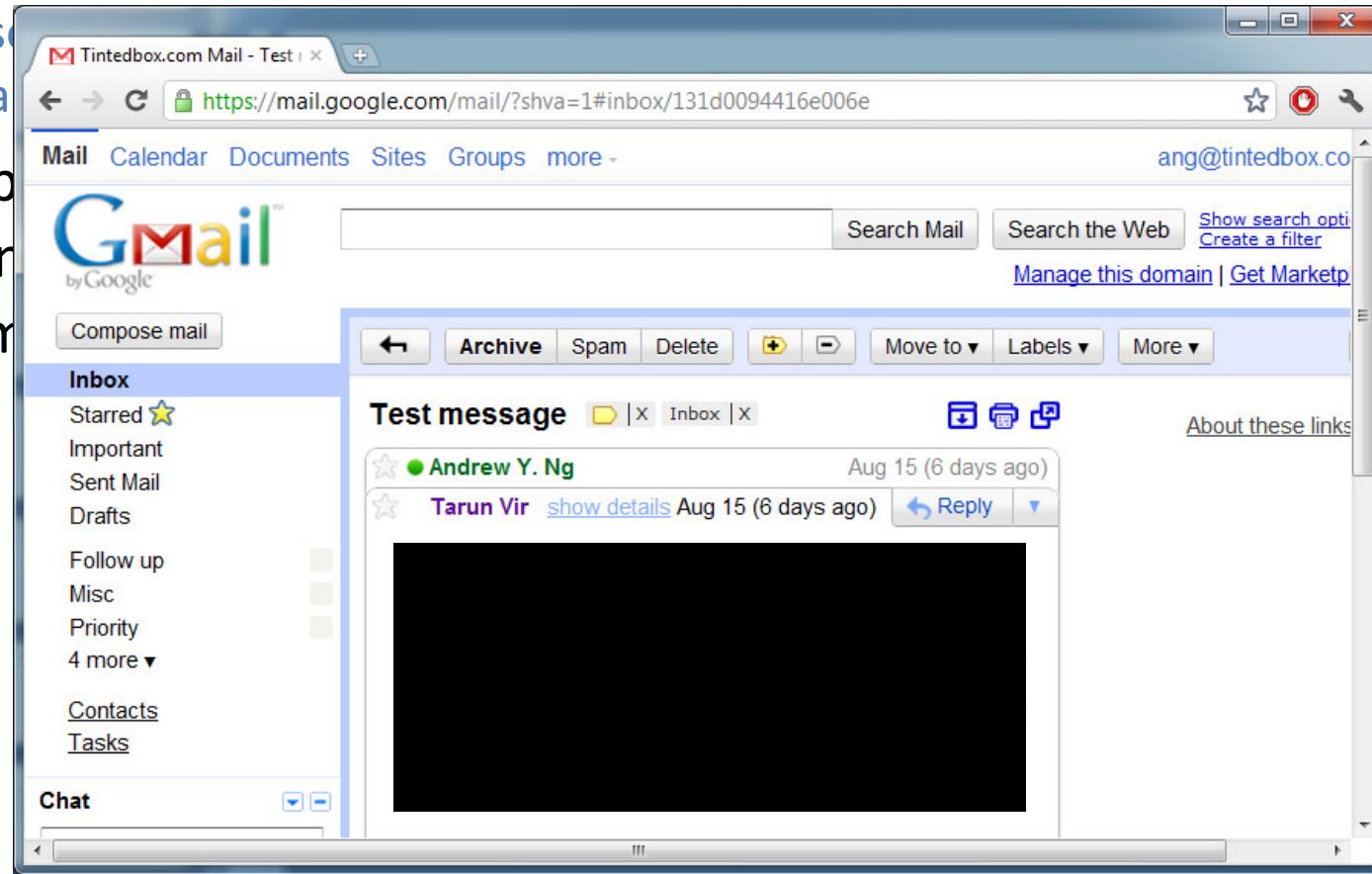
- Arthur Samuel (1959). Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.
- Tom Mitchell (1998) Well-posed Learning Problem: A computer program is said to *learn* from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.

“A computer program is said to *learn from* experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.”

Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam. What is the task T in this setting?

- Classifying emails as spam or not spam. T ↵
- Watching you label emails as spam or not spam. E ↵
- The number (or fraction) of emails correctly classified as spam/not spam. P ↵
- None of the above—this is not a machine learning problem.

“A computer program is said to *learn* from experience E with respect to



“A computer program is said to *learn from* experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E.”

Suppose your email program watches which emails you do or do not mark as spam, and based on that learns how to better filter spam. What is the task T in this setting?

- Classifying emails as spam or not spam. T ↵
- Watching you label emails as spam or not spam. E ↵
- The number (or fraction) of emails correctly classified as spam/not spam. P ↵
- None of the above—this is not a machine learning problem.

# Machine learning algorithms:

- Supervised learning
- Unsupervised learning

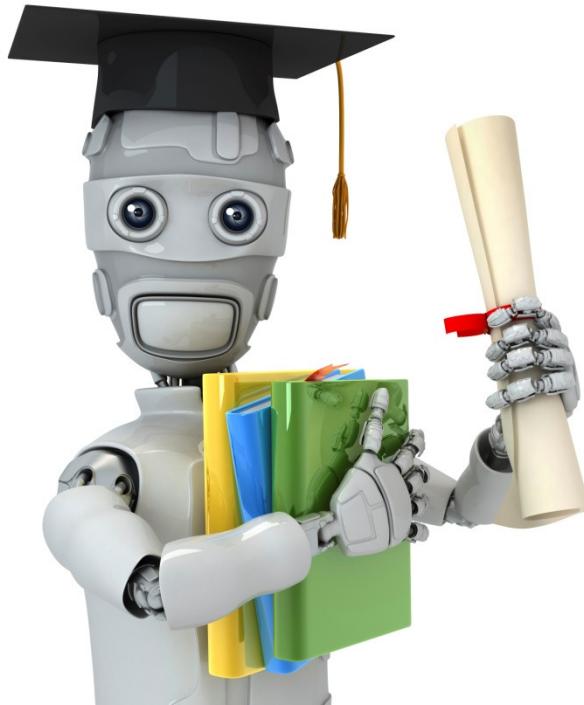


Others: Reinforcement learning, recommender systems.

Also talk about: Practical advice for applying learning algorithms.





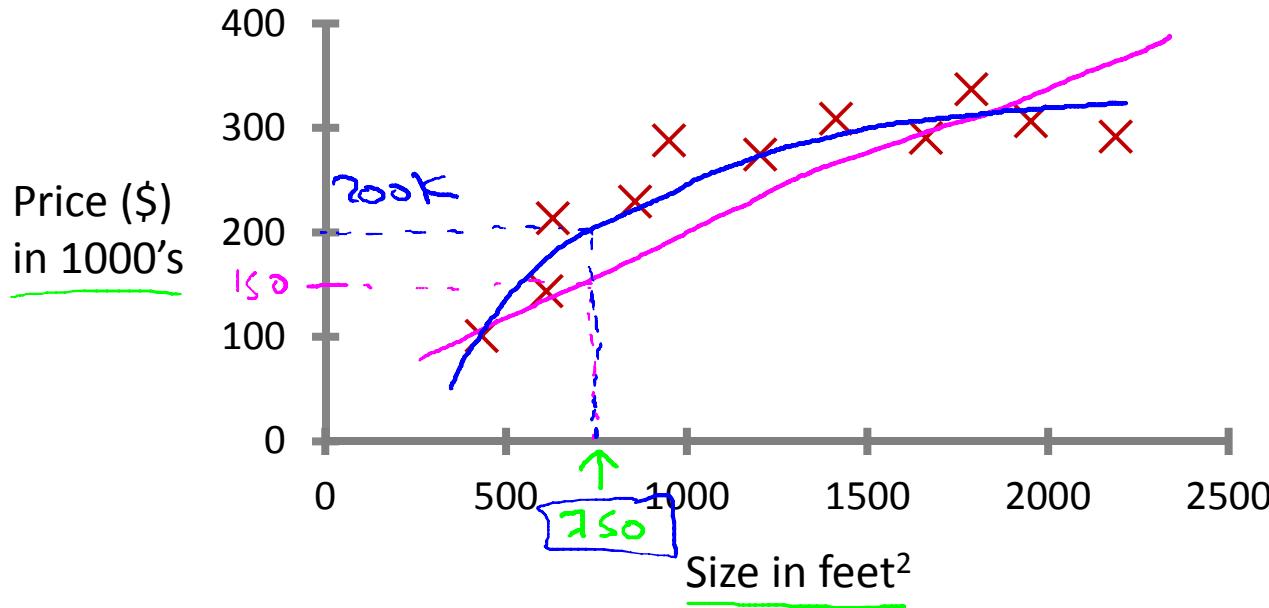


Machine Learning

# Introduction Supervised Learning

---

# Housing price prediction.

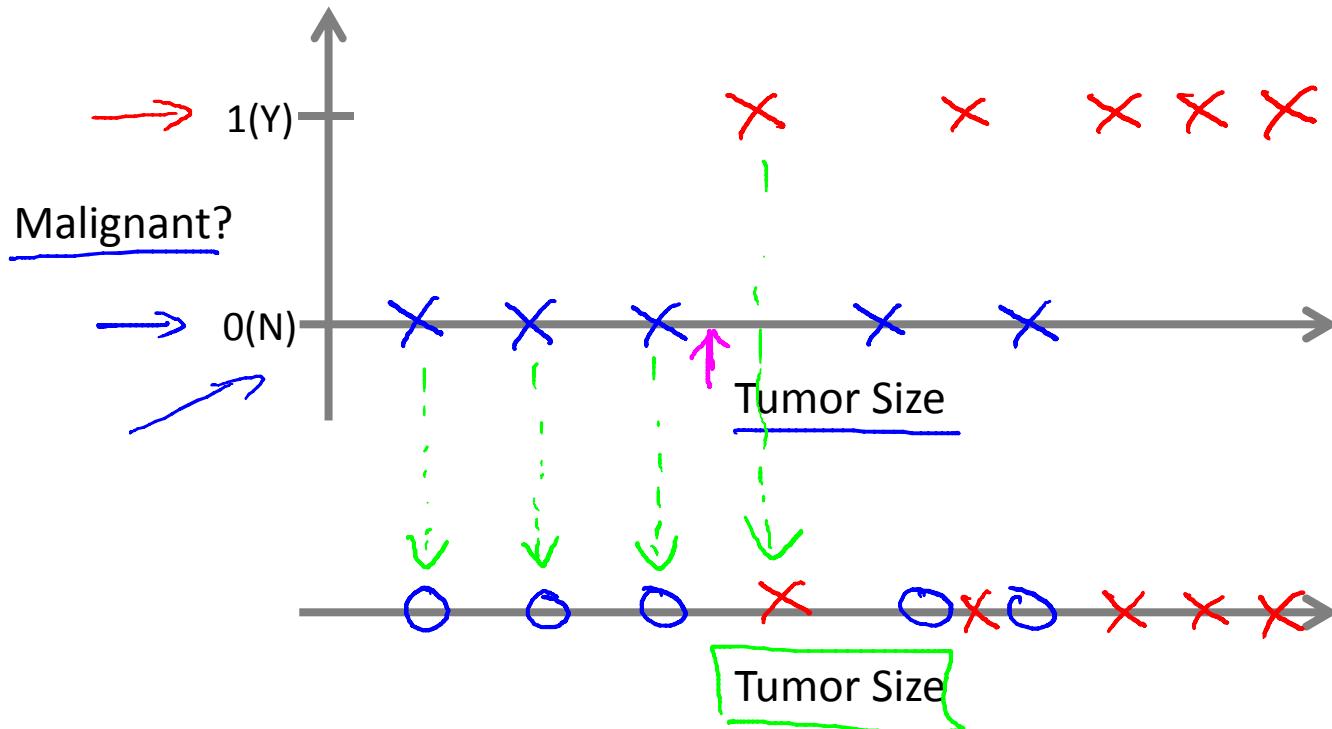


Supervised Learning

'right answers' given

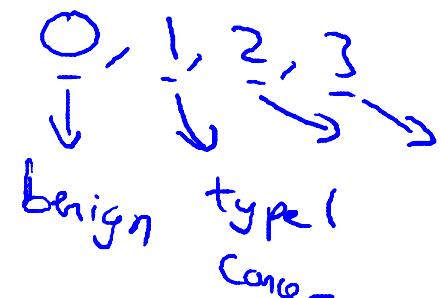
Regression: Predict continuous valued output (price)

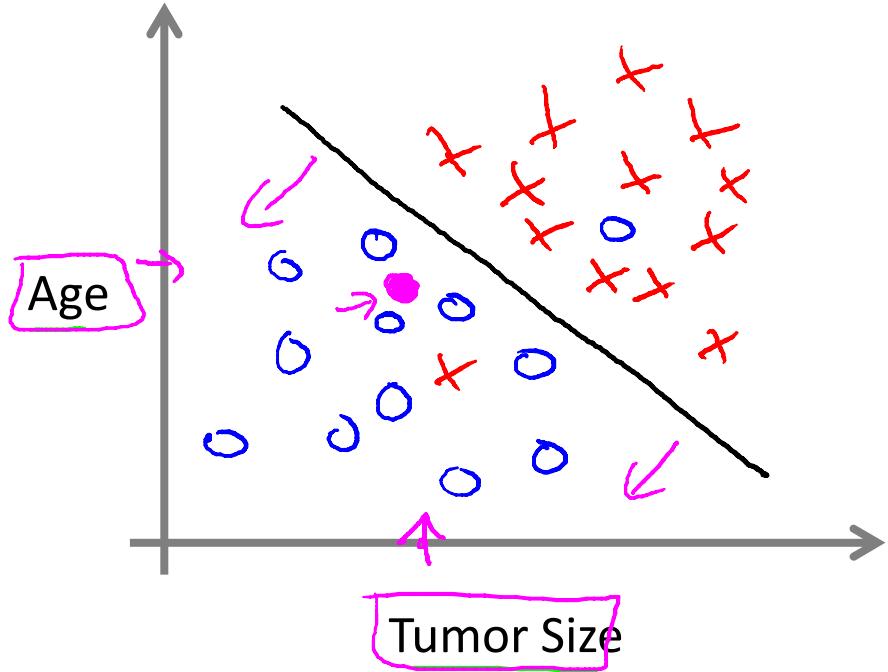
# Breast cancer (malignant, benign)



## Classification

Discrete valued output (0 or 1)





- Clump Thickness
- Uniformity of Cell Size
- Uniformity of Cell Shape
- ...

You're running a company, and you want to develop learning algorithms to address each of two problems.

1000's

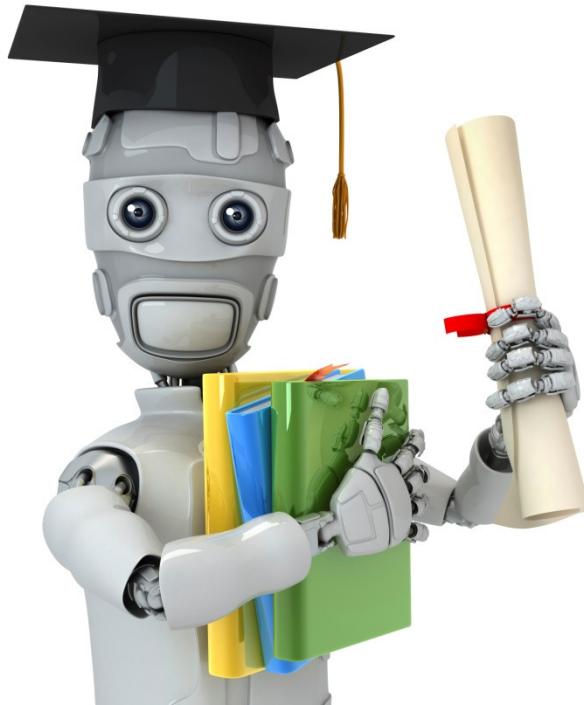
- Problem 1: You have a large inventory of identical items. You want to predict how many of these items will sell over the next 3 months.
- Problem 2: You'd like software to examine individual customer accounts, and for each account decide if it has been hacked/compromised.

→ 0 - not hacked  
→ 1 - hacked

Should you treat these as classification or as regression problems?

- Treat both as classification problems.
- Treat problem 1 as a classification problem, problem 2 as a regression problem.
- Treat problem 1 as a regression problem, problem 2 as a classification problem.
- Treat both as regression problems.





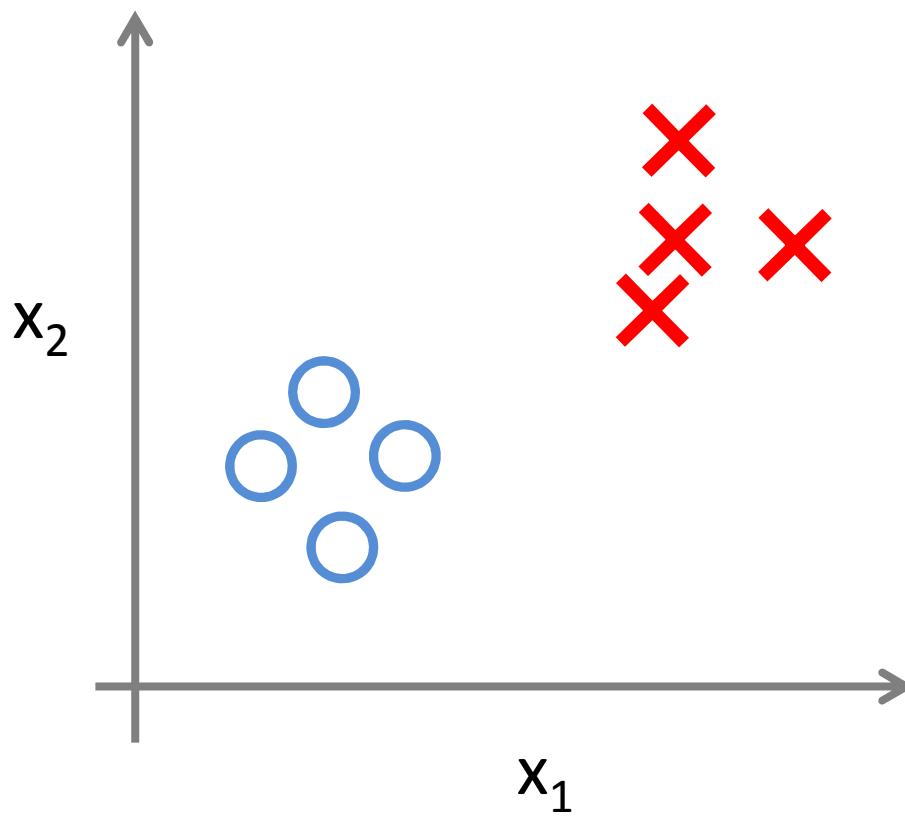
Machine Learning

# Introduction

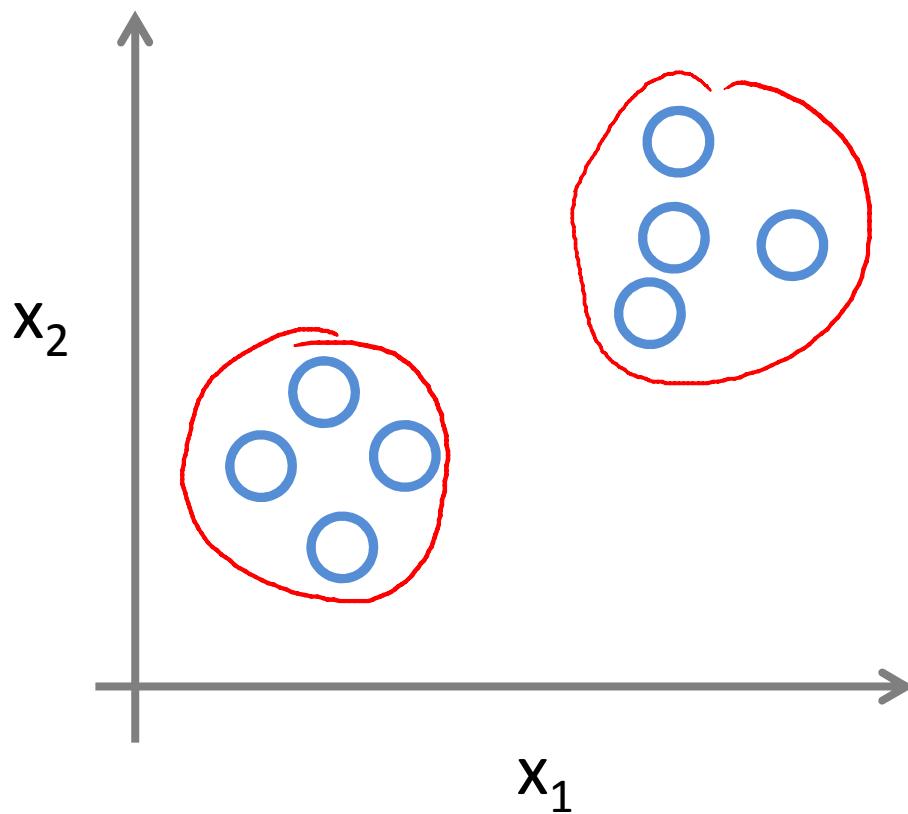
---

# Unsupervised Learning

# Supervised Learning



# Unsupervised Learning



Google News X

news.google.com X

Web Images Videos Maps News Shopping Gmail more ▾ andrewyantakng@gmail.com | Web History | Settings ▾ | Sign out

# Google news

Search News Search the Web Advanced news search U.S. edition ▾ Add a section »

**Top Stories**

- Deepwater Horizon
- Fed meeting
- Foreign exchange market
- Lindsay Lohan
- IBM
- Tom Brady
- Toronto International Film Festival
- Paris Hilton
- Iran
- Hurricane Igor

**Starred** ★

- San Francisco Bay Area
- World
- U.S.
- Business
- Sci/Tech
- More Top Stories
- Spotlight
- Health
- Sports
- Entertainment

All news Headlines Images

**Top Stories**

[Christine O'Donnell »](#)  
**White House official denies Tea Party-focused ad campaign**  
CNN International - Ed Henry - 1 hour ago  
Democratic sources say the White House is not considering an ad campaign tying Republicans to the Tea Party. Washington (CNN) -- A top White House official sharply denied a report that claims President Obama's political advisers are weighing a national ...  
[Tea Party is misplacing the blame, former President Bill Clinton claims](#)  
New York Daily News  
[GOP tea party backer defends Christine O'Donnell](#) The Associated Press  
Atlanta Journal Constitution - Politics Daily - MyFox Washington DC - Salon all 726 news articles »

**US Stocks Climb After Recession Called Over, Homebuilders Gain**  
MarketWatch - Kristina Peterson - 16 minutes ago  
NEW YORK (MarketWatch) -- US stocks climbed Monday, gaining speed after a key nonprofit organization officially called the recession over, giving investors a boost of confidence in the gradual economic recovery.  
[Longest recession since 1930s ended in June 2009, group says](#)  
Los Angeles Times  
[Downturn Was Longest in Decades, Panel Confirms](#) New York Times  
Wall Street Journal - AFP - CNN - USA Today all 276 news articles »

[Deepwater Horizon »](#)  
**BP Oil Well, Site of National Catastrophe, Dies at One**  
Vanity Fair - Juli Weiner - 22 minutes ago  
The BP oil well, site of the Deepwater Horizon explosion that led to the worst oil spill in US history, died today at one year old.  
Video: Blown-out BP Well Finally Killed in Gulf YouTube The Associated Press  
Weiss Doubts BP Would End Operations in Gulf of Mexico: Video Bloomberg  
CNN International - Wall Street Journal (blog) - The Guardian - New York Times all 2,292 news articles »

**Recent**

[Recession officially ended in June 2009](#)  
CNNMoney - Chris Isidore - 39 minutes ago

[Hurricane Igor lashes Bermuda](#)  
USA Today - Gerry Broome - 5 minutes ago

['Explain what you want from us,' reads front-page editorial](#)  
msnbc.com - Olivia Torres - 10 minutes ago

Crisis response: Pakistan floods

**San Francisco Bay Area - Edit**

[Clorox »](#)  
[Bay Biz Buzz: Clorox close to selling STP, Armor All](#)  
San Jose Mercury News - 48 minutes ago - all 24 articles »

[Google's official beekeeper keeps the company buzzing with excitement](#)  
San Jose Mercury News - Bruce Newman - 1 hour ago

[Jon Sylvia »](#)  
[Martinez man still unconscious as police investigate weekend shooting](#)  
San Jose Mercury News - Robert Salonga - 48 minutes ago - all 6 articles »

**Spotlight**

[Sarkozy rages at EU 'humiliation'](#)  
Financial Times - Peggy Hollinger - Sep 16, 2010

A screenshot of a CNN news article. The URL in the browser is edition.cnn.com/2010/09/20/gulf.oil.disaster/. The page header includes 'EDITION: INTERNATIONAL' and links for 'U.S.', 'MÉXICO', and 'ARABIC'. Below the header are navigation links for Home, Video, World, U.S., Africa, Asia, Europe, Latin America, Middle East, Business, and Weather. A large red arrow points from the left margin down to the headline of the story. The main headline reads 'Allen: Well is dead, but much Gulf Coast work remains'. Below it is a sub-headline 'By the CNN Wire Staff'. The date 'September 20, 2010 — Updated 1317 GMT (2117 HKT)' is also present. The main image shows an offshore oil rig in the Gulf of Mexico. A call-to-action button in the center of the image says 'Click to play' with a video camera icon.

**BP Kills Macondo, But Its Legacy Lives On**

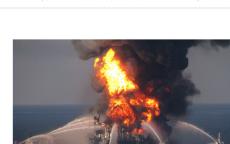
Article      Comments (2)

Email    Print    Permalink    + More

By James Herron

BP confirmed late Sunday that the Macondo well that leaked almost five million barrels of oil into the Gulf of Mexico has been permanently sealed, but the well will continue to affect BP and the wider oil industry for many years.

The most immediate worry for BP and its shareholders is how the authorities will apportion blame for the spill. BP's own investigation spread responsibility across

  
Associated Press  
Fire boat response crews battled the blazing remnants of the offshore oil platform Deepwater Horizon on April 21, 2010.

**About The Source**

The Source is WSJ.com's home for rapid-fire analysis of the day's big business and finance stories. It is edited by Lauren Mills, based in London.

**Most Recent**

Articles      Comments

- 1. Who Needs Plaza II Anyway
- 2. Will Banks Be Forced to Split Retail And Banking Arms?
- 3. Timing of Ratings Award Intriguing
- 4. BP Kills Macondo, But Its Legacy Lives On
- 5. We Already Need a Serval to Recall ISIS!!

spread responsibility across... [the Deepwater Horizon](#)

BP oil spill cost hits nearl... [BP oil spill](#)

Mobile site Sign in Register A Text larger smaller About us

---

**guardian.co.uk**

News | Sport | Comment | Culture | Business | Money | Life & style

Business > BP

# BP oil spill cost hits nearly \$10bn

BP has set up a \$20bn compensation fund after the Deepwater Horizon disaster, which has so far paid out 19,000 claims totalling more than \$240m

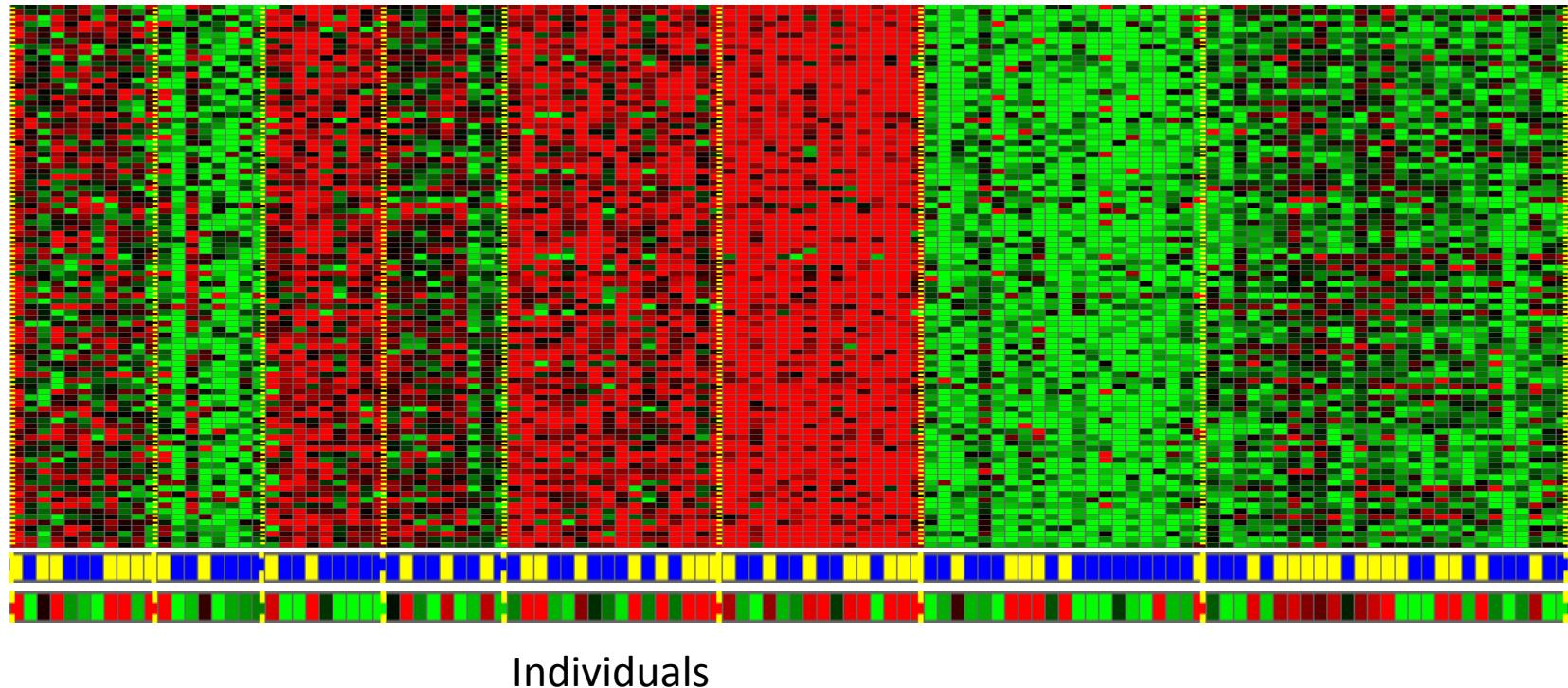
---

**Julia Kollewe**  
guardian.co.uk, Monday 20 September 2010 08.33 BST  
[Article history](#)

---



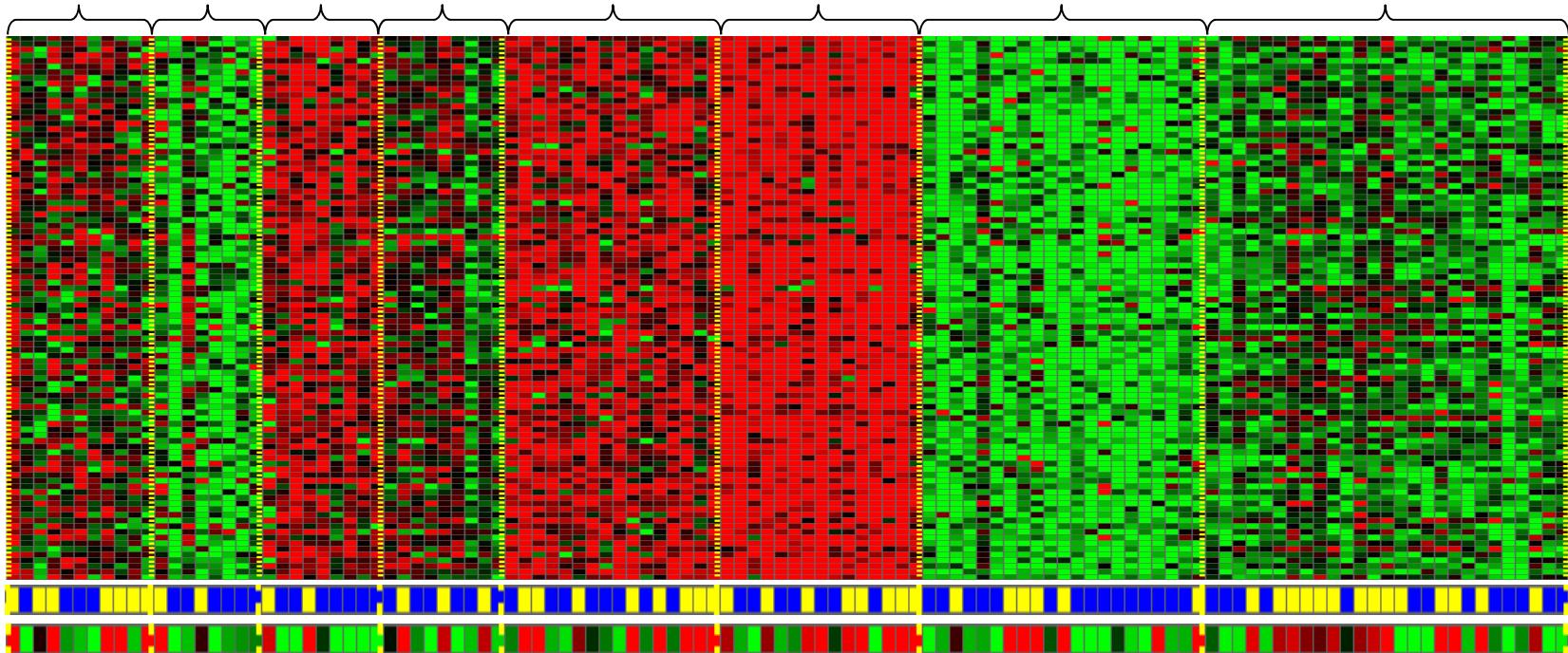
BP's costs for the Deepwater Horizon disaster have hit \$10bn. Photograph: Ho/Reuters



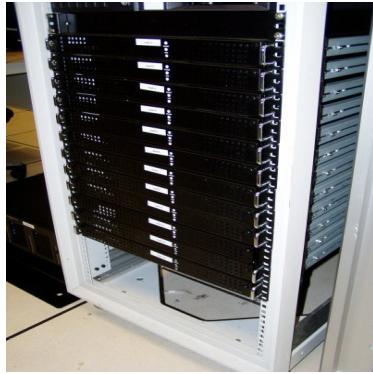
[Source: Daphne Koller]

Andrew Ng

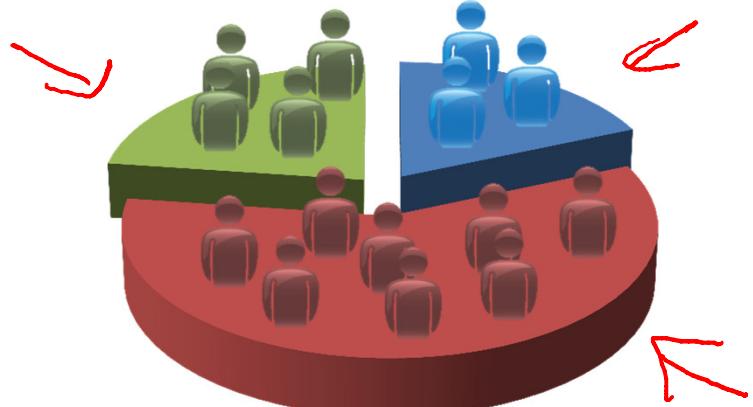
Genes



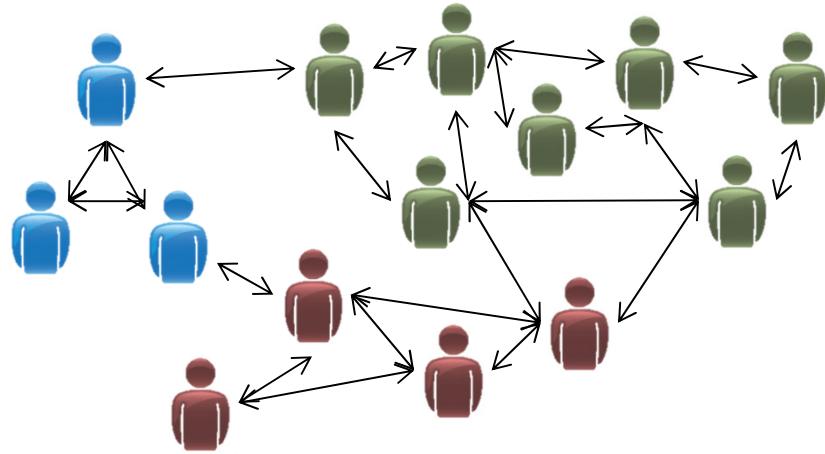
Individuals



Organize computing clusters



Market segmentation



Social network analysis

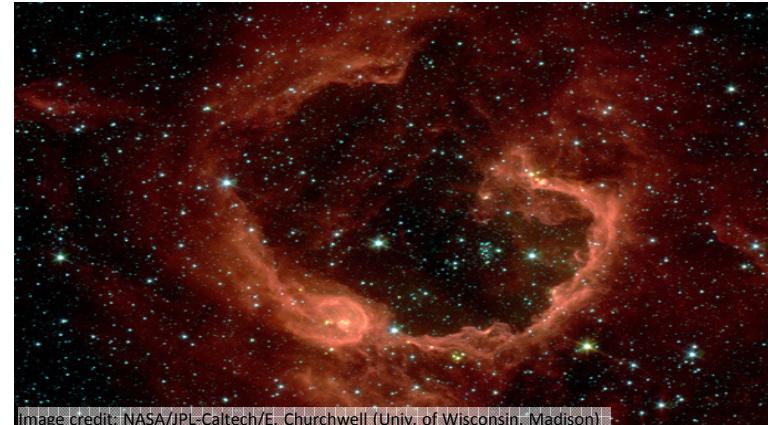
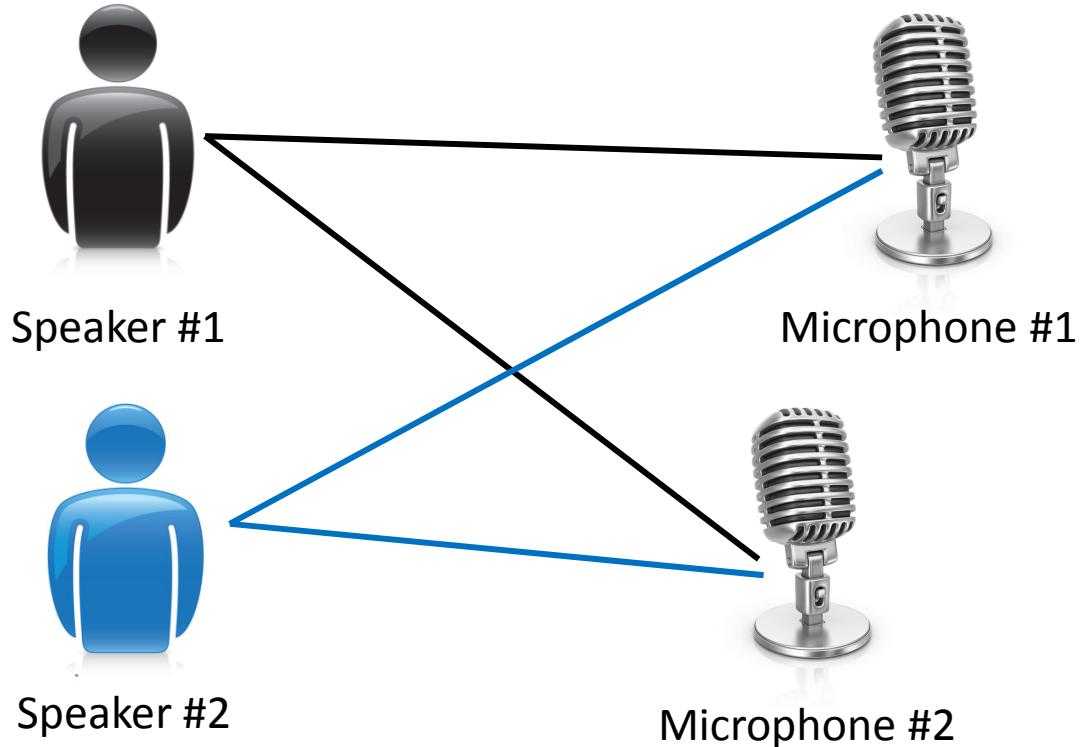


Image credit: NASA/JPL-Caltech/E. Churchwell (Univ. of Wisconsin, Madison)

Astronomical data analysis

Andrew Ng

# Cocktail party problem



Microphone #1: 

Output #1: 

Microphone #2: 

Output #2: 

---

Microphone #1: 

Output #1: 

Microphone #2: 

Output #2: 

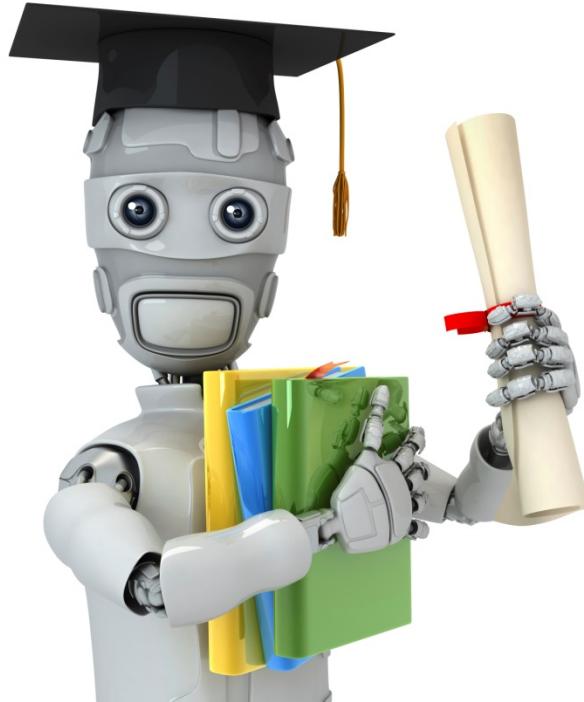
# Cocktail party problem algorithm

```
[W,s,v] = svd((repmat(sum(x.*x,1),size(x,1),1).*x)*x');
```

Of the following examples, which would you address using an unsupervised learning algorithm? (Check all that apply.)

- Given email labeled as spam/not spam, learn a spam filter.
- Given a set of news articles found on the web, group them into set of articles about the same story.
- Given a database of customer data, automatically discover market segments and group customers into different market segments.
- Given a dataset of patients diagnosed as either having diabetes or not, learn to classify new patients as having diabetes or not.





Machine Learning

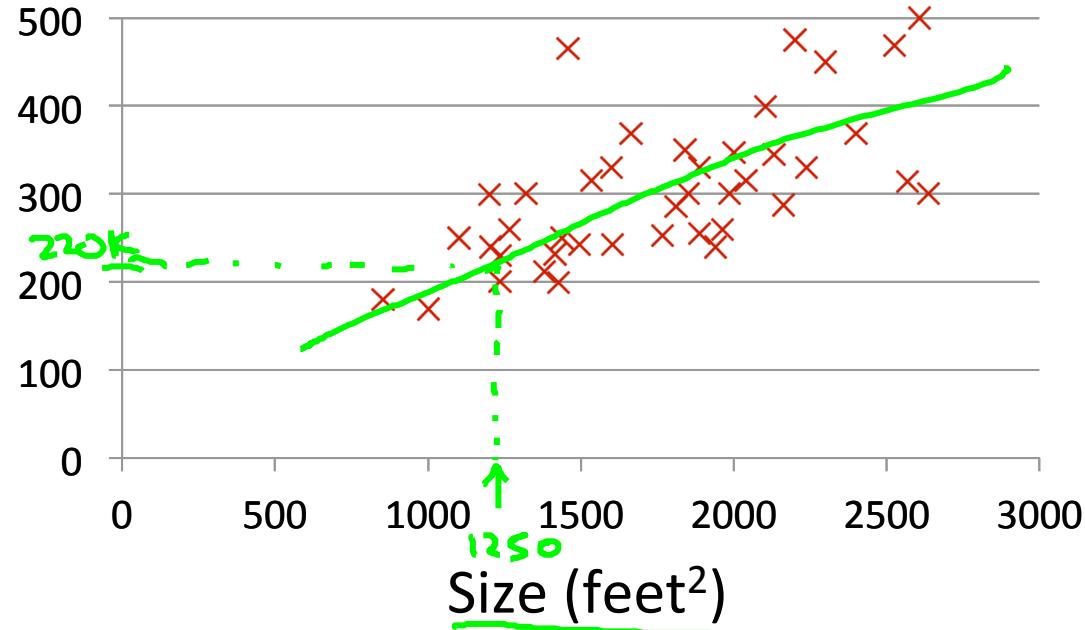
# Linear regression with one variable

---

## Model representation

# Housing Prices (Portland, OR)

Price  
(in 1000s  
of dollars)



## Supervised Learning

Given the "right answer" for each example in the data.

## Regression Problem

Predict real-valued output

Classification: Discrete-valued output

# Training set of housing prices (Portland, OR)

Size in feet <sup>2</sup> (x)	Price (\$) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

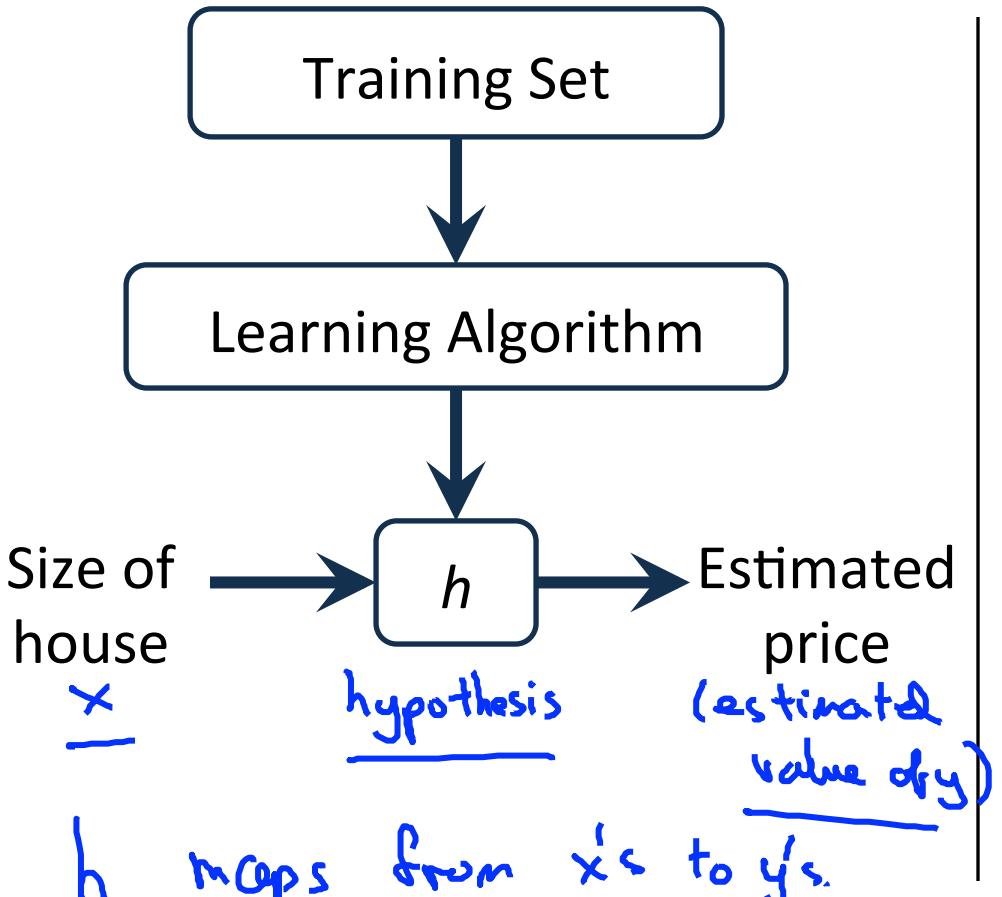
Notation:

- $m$  = Number of training examples
- $x$ 's = "input" variable / features
- $y$ 's = "output" variable / "target" variable

$(x, y)$  - one training example

$(x^{(i)}, y^{(i)})$  -  $i^{\text{th}}$  training example

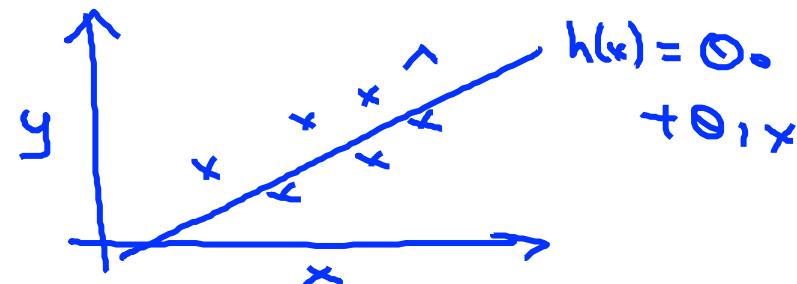
$$\left\{ \begin{array}{l} x^{(1)} = 2104 \\ x^{(2)} = 1416 \\ y^{(1)} = 460 \end{array} \right.$$



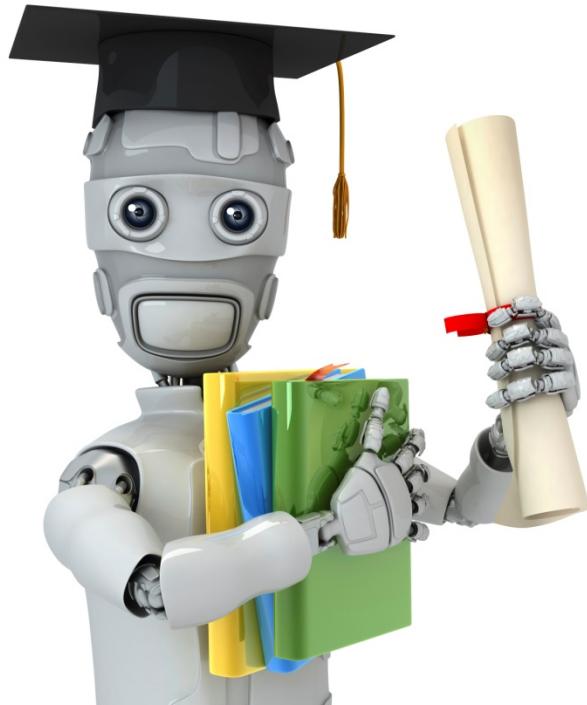
## How do we represent $h$ ?

$$h_{\Theta}(x) = \underline{\underline{\Theta_0 + \Theta_1 x}}$$

Shorthand:  $h(x)$



Linear regression with one variable.  
Univariate linear regression.  
One variable



Machine Learning

# Linear regression with one variable

---

## Cost function

# Training Set

Size in feet <sup>2</sup> (x)	Price (\$) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

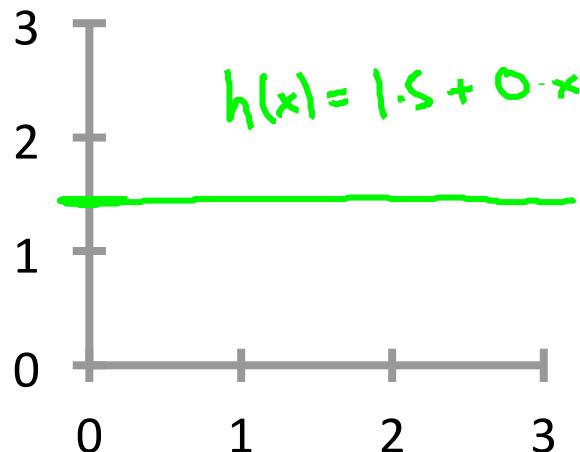
$m = 47$

Hypothesis: 
$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

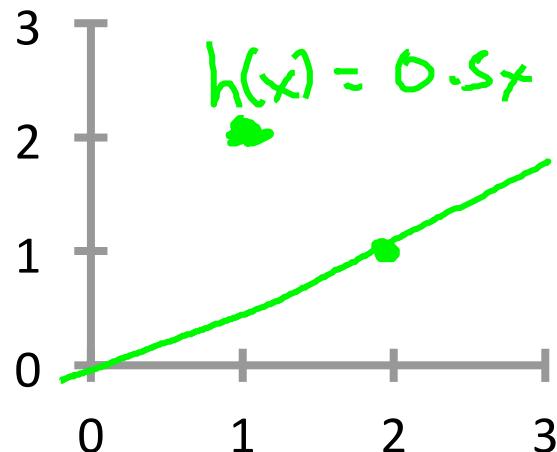
$\theta_i$ 's: Parameters

How to choose  $\theta_i$ 's ?

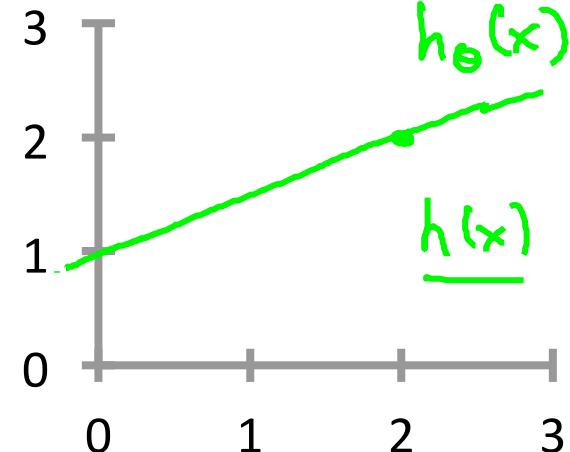
$$\underline{h_{\theta}(x) = \theta_0 + \theta_1 x}$$



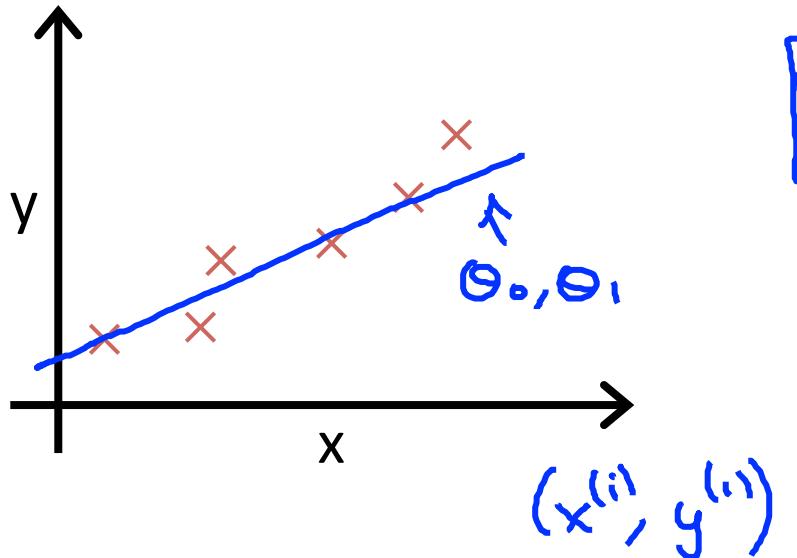
$$\begin{aligned}\rightarrow \theta_0 &= 1.5 \\ \rightarrow \theta_1 &= 0\end{aligned}$$



$$\begin{aligned}\rightarrow \theta_0 &= 0 \\ \rightarrow \theta_1 &= 0.5\end{aligned}$$



$$\begin{aligned}\rightarrow \theta_0 &= 1 \\ \rightarrow \theta_1 &= 0.5\end{aligned}$$



Idea: Choose  $\theta_0, \theta_1$  so that  $\underline{h_\theta(x)}$  is close to  $\underline{y}$  for our training examples  $(\underline{x}, \underline{y})$

$x, y$

minimize  $\theta_0, \theta_1$

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

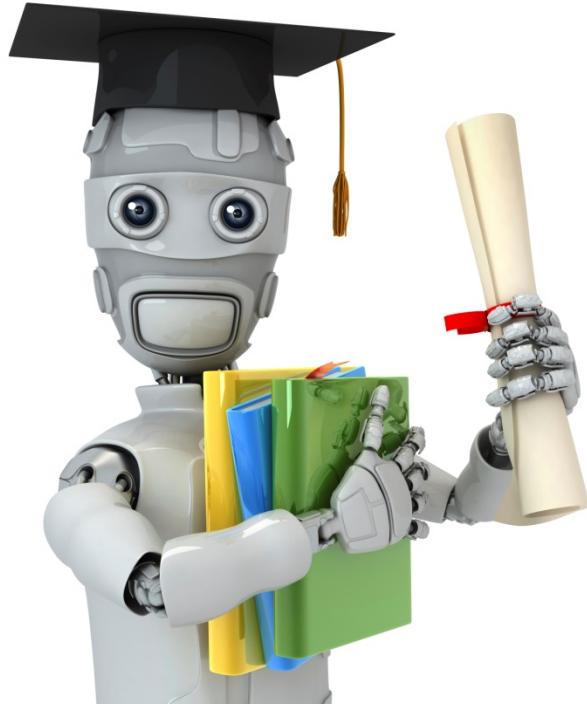
$h_\theta(\underline{x}^{(i)}) = \underline{\theta_0} + \underline{\theta_1 x^{(i)}}$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(\underline{x}^{(i)}) - \underline{y}^{(i)})^2$$

minimize  $\theta_0, \theta_1$   $J(\theta_0, \theta_1)$

Cost function

Squared error function



Machine Learning

Linear regression  
with one variable

---

Cost function  
intuition I

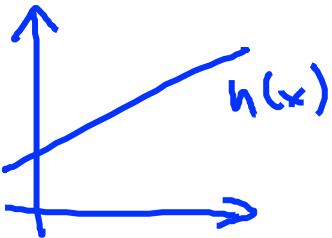
## Simplified

Hypothesis:

$$\underline{h_{\theta}(x) = \theta_0 + \theta_1 x}$$

Parameters:

$$\underline{\theta_0, \theta_1}$$



Cost Function:

$$\rightarrow J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

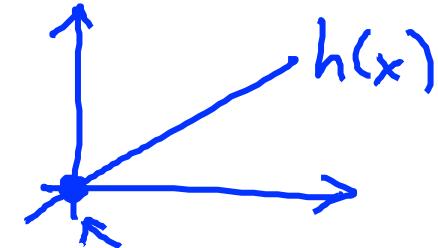
Goal: minimize  $J(\theta_0, \theta_1)$

$$\underline{\theta_0, \theta_1}$$

$$h_{\theta}(x) = \underline{\theta_1 x}$$

$$\underline{\theta_0 = 0}$$

$$\underline{\theta_1}$$



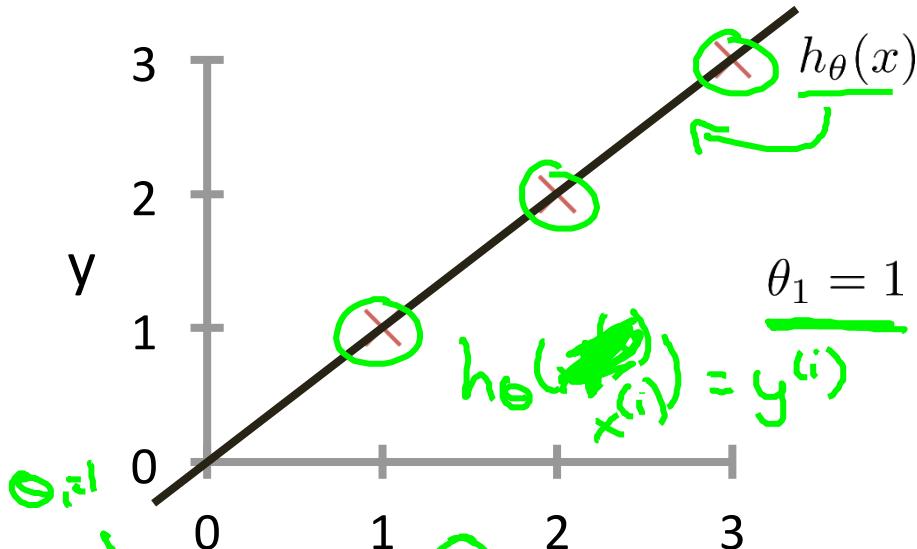
$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^m (\underline{h_{\theta}(x^{(i)}) - y^{(i)}})^2$$

$$\min_{\theta_1} \underline{J(\theta_1)}$$

$$\underline{\theta_0, x^{(i)}}$$

$\rightarrow \underline{h_\theta(x)}$

(for fixed  $\underline{\theta_1}$ , this is a function of x)

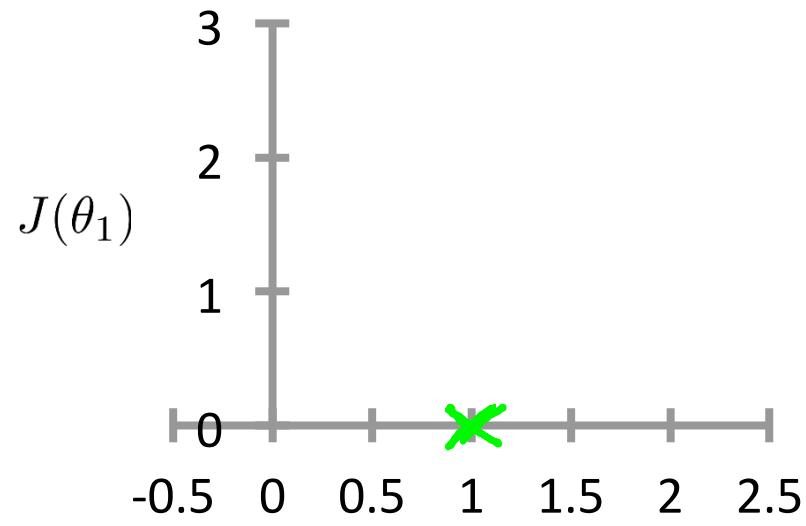


$$\underline{J(\theta_1)} = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

$$= \frac{1}{2m} \sum_{i=1}^m (\underline{\theta_1 x^{(i)}} - y^{(i)})^2 = \frac{1}{2m} (0^2 + 0^2 + 0^2) = 0^2$$

$\rightarrow \underline{J(\theta_1)}$

(function of the parameter  $\theta_1$ )



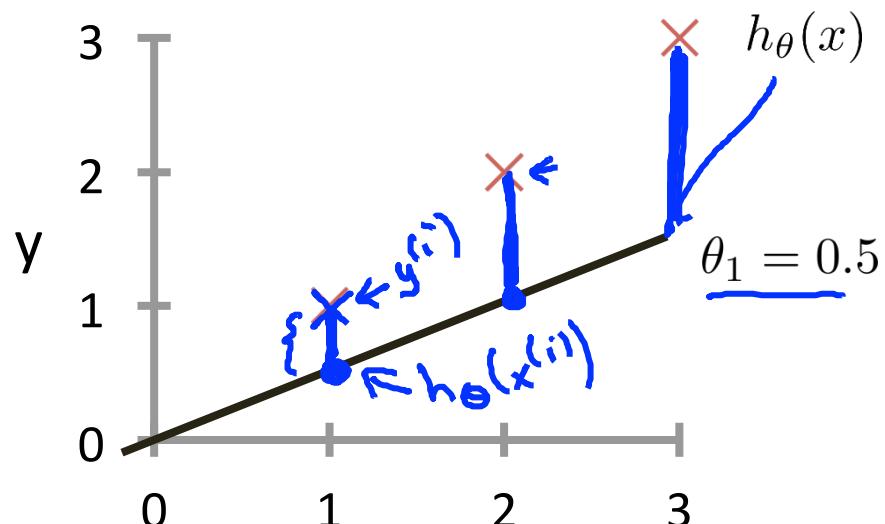
$$\theta_1 = 0.5?$$

$$\theta_1$$

$$\underline{J(1)} = 0$$

$$h_{\theta}(x)$$

(for fixed  $\theta_1$ , this is a function of  $x$ )

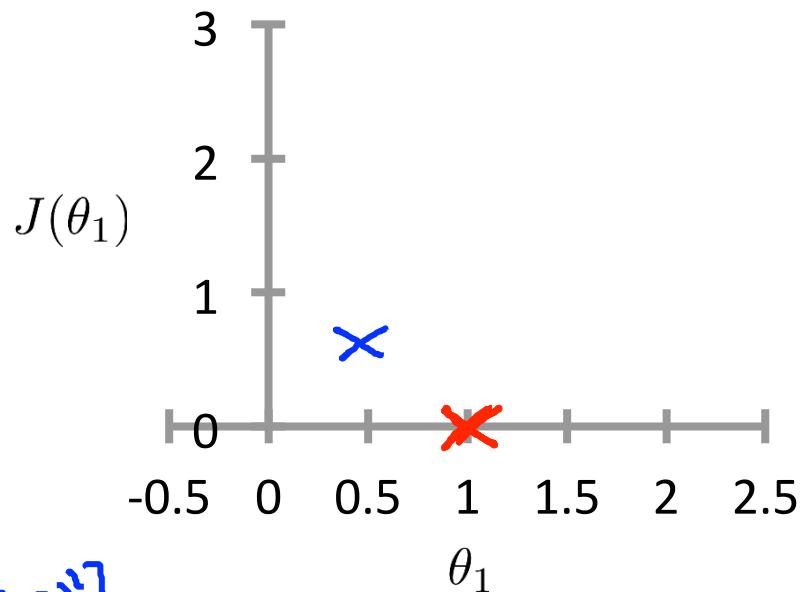


$$J(0.5) = \frac{1}{2m} [ (0.5 - 1)^2 + (1 - 2)^2 + (1.5 - 3)^2 ]$$

$$= \frac{1}{2 \times 3} (3.5) = \frac{3.5}{6} \approx \underline{0.58}$$

$$J(\theta_1)$$

(function of the parameter  $\theta_1$ )

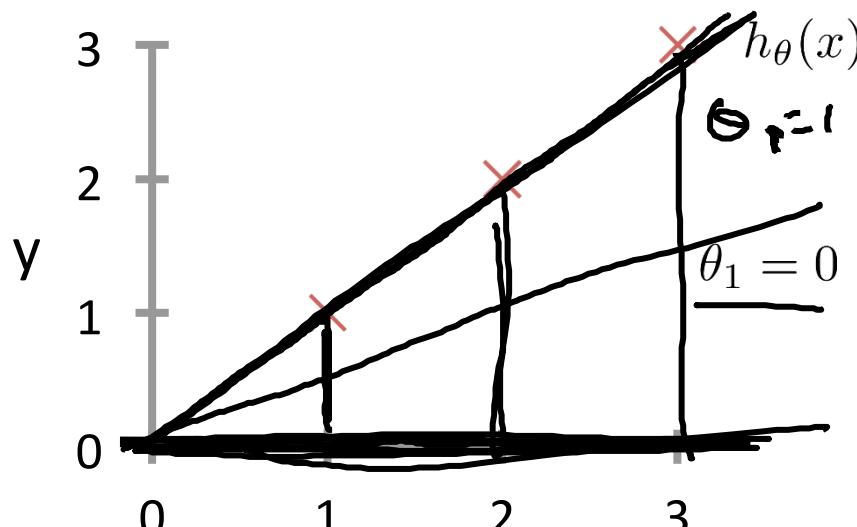


$$\Theta_1 = 0.58$$

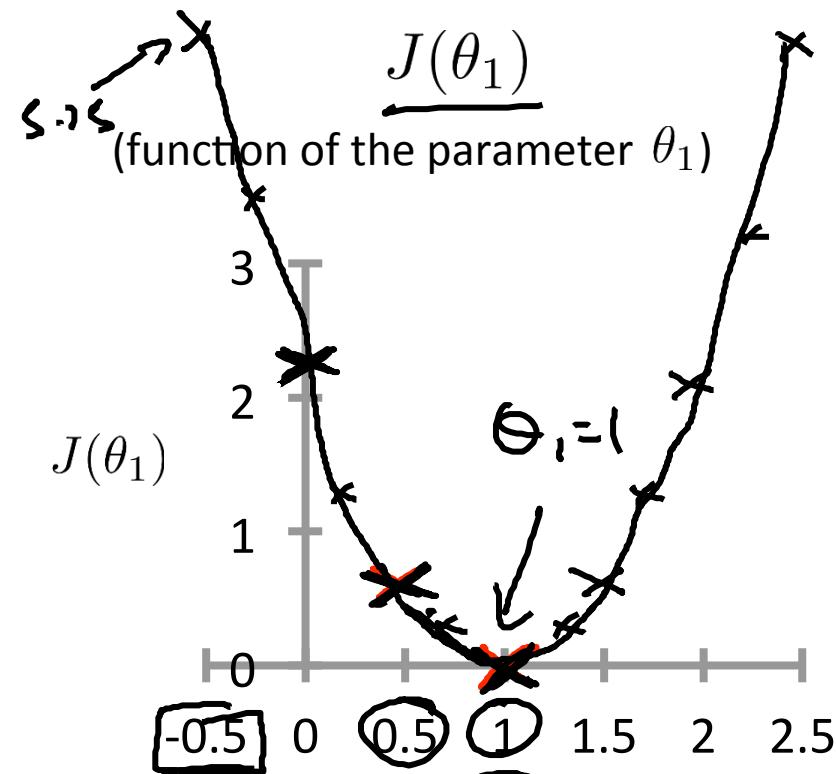
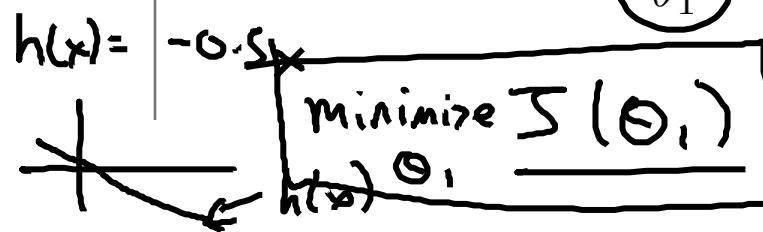
$$J(0) = ?$$

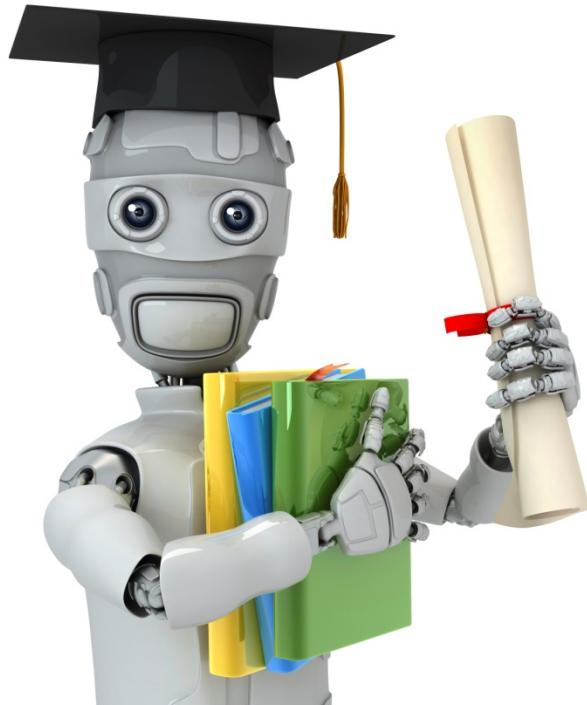
$$h_{\theta}(x)$$

(for fixed  $\theta_1$ , this is a function of  $x$ )



$$\begin{aligned} J(0) &= \frac{1}{2m} (1^2 + 2^2 + 3^2) \\ &= \frac{1}{6} \cdot 14 \approx 2.3 \end{aligned}$$





Machine Learning

Linear regression  
with one variable

---

Cost function  
intuition II

Hypothesis:  $h_{\theta}(x) = \theta_0 + \theta_1 x$

Parameters:  $\theta_0, \theta_1$

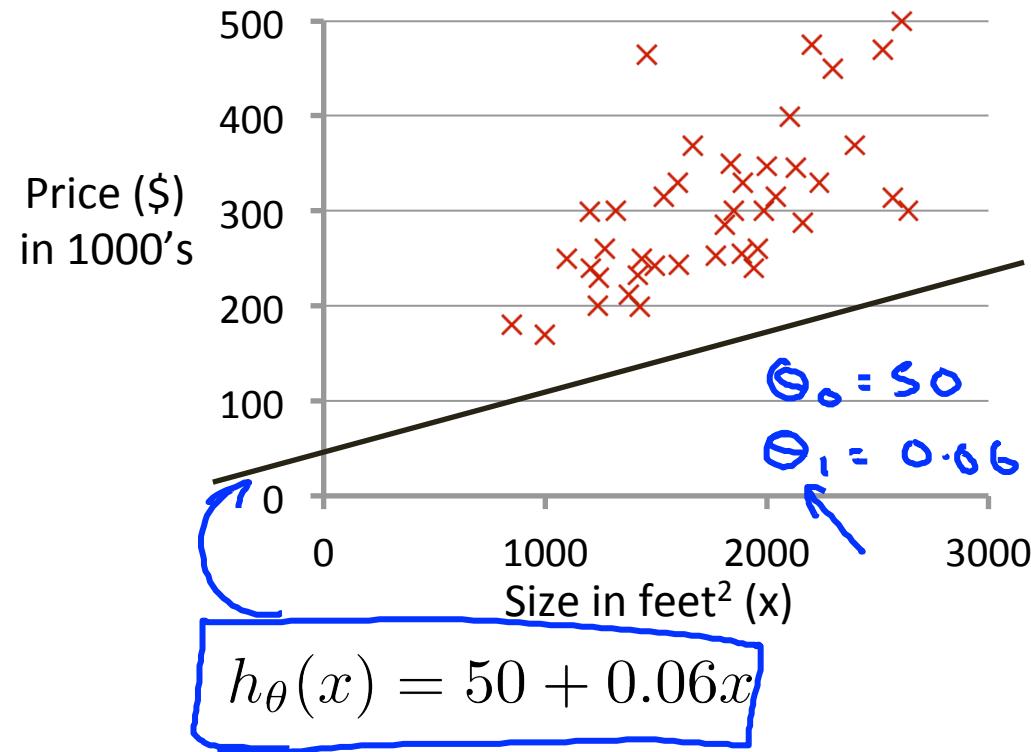
Cost Function:  $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$

Goal: minimize  $J(\theta_0, \theta_1)$   
 $\theta_0, \theta_1$

.

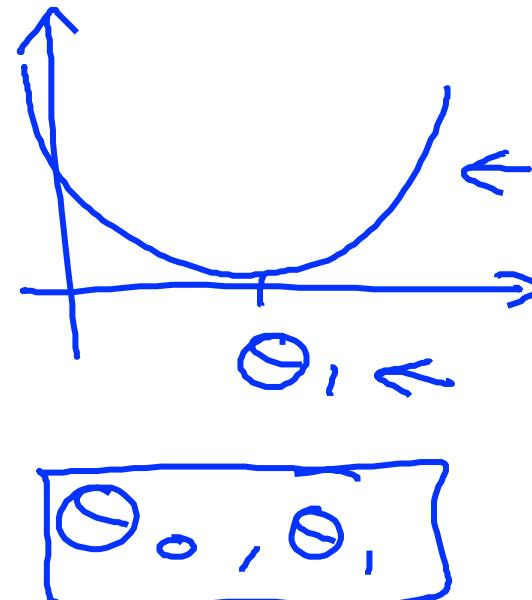
$$\underline{h_{\theta}(x)}$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )

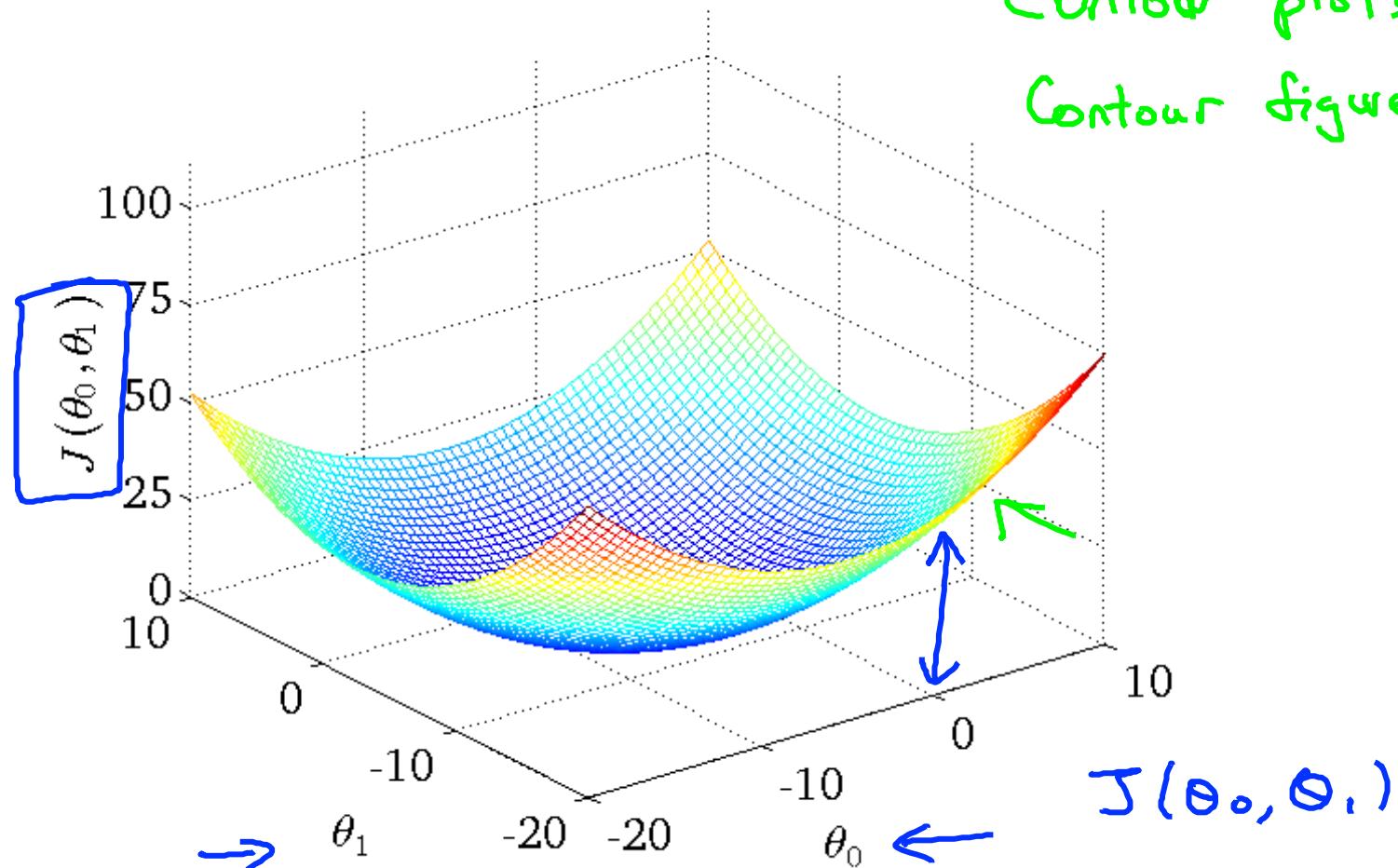


$$\underline{J(\theta_0, \theta_1)}$$

(function of the parameters  $\theta_0, \theta_1$ )

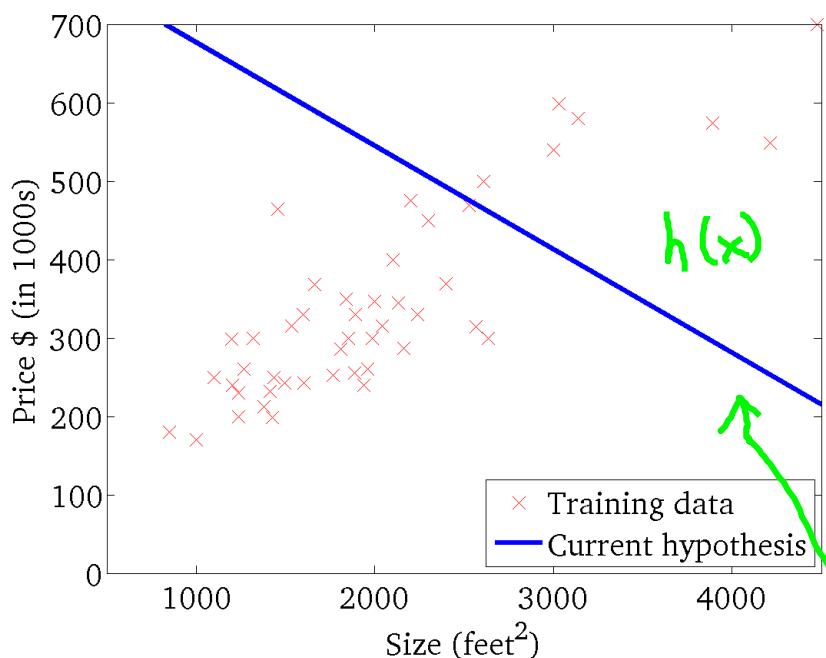


Contour plots  
Contour figures -



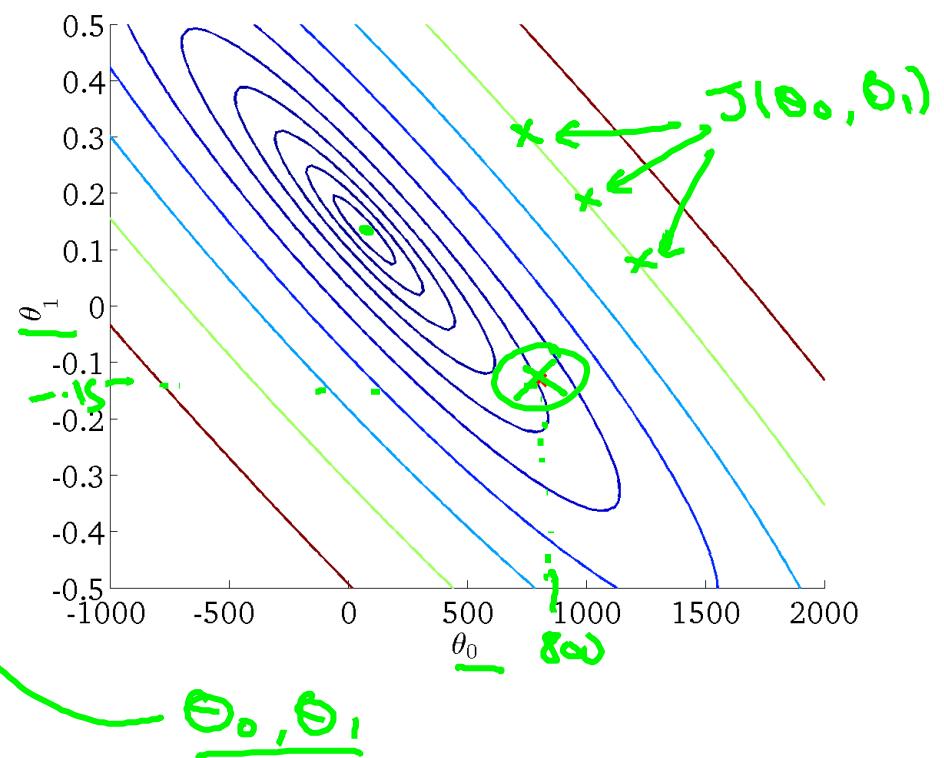
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



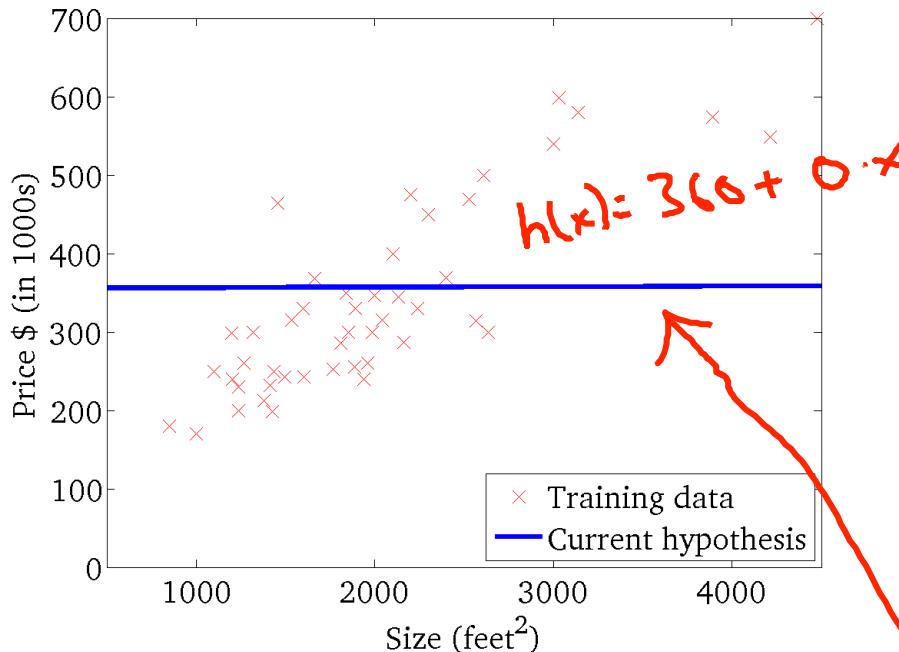
$$J(\theta_0, \theta_1)$$

(function of the parameters  $\theta_0, \theta_1$ )



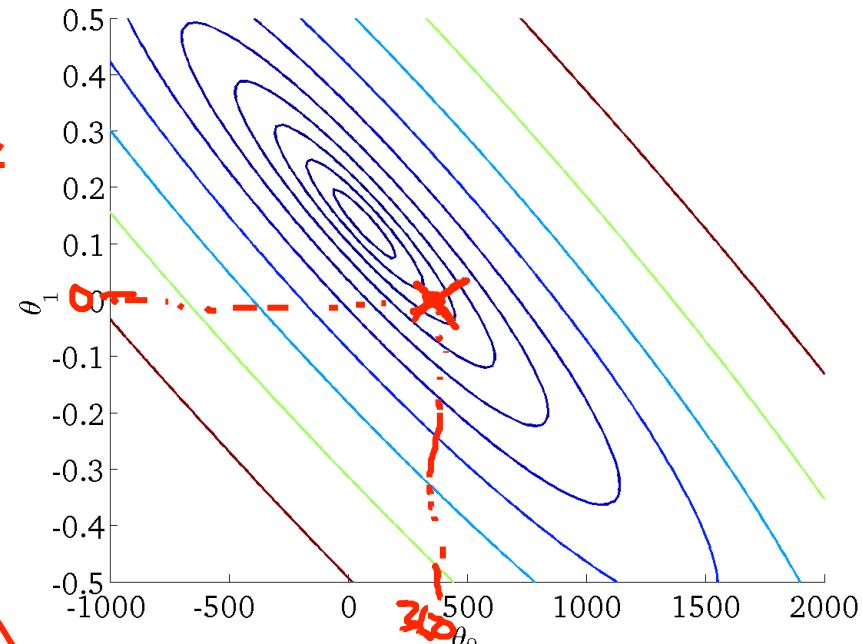
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



$$J(\theta_0, \theta_1)$$

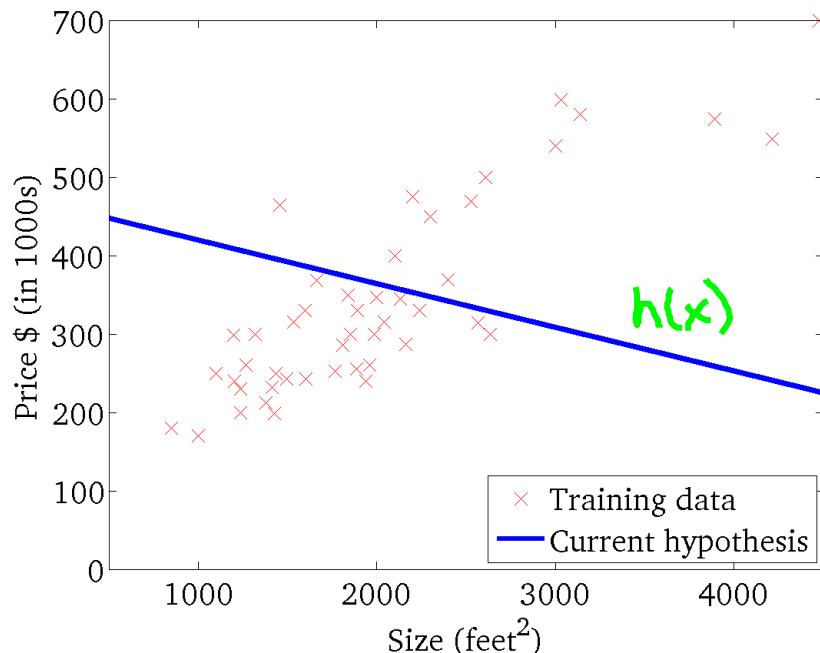
(function of the parameters  $\theta_0, \theta_1$ )



$$\begin{cases} \theta_0 = 360 \\ \theta_1 = 0 \end{cases}$$

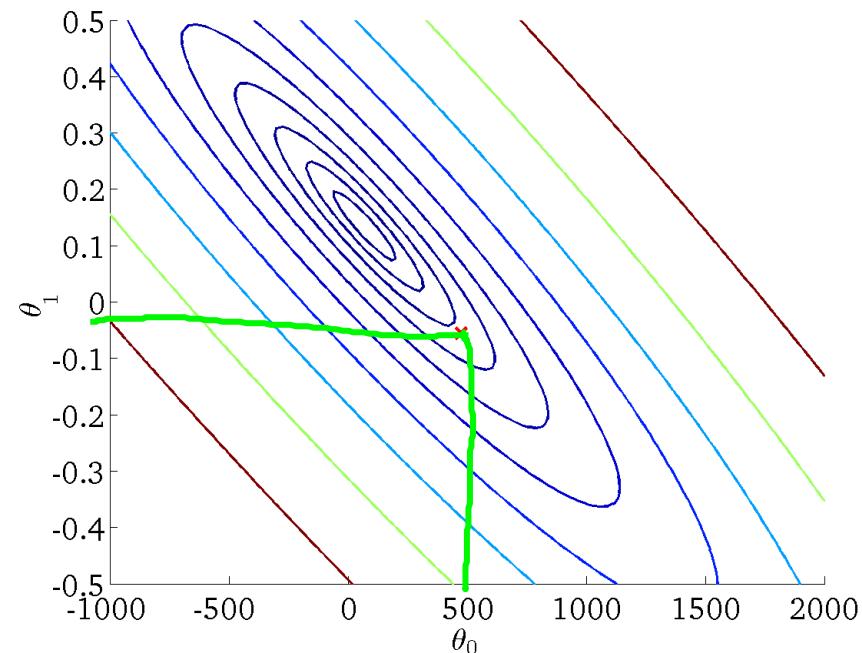
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



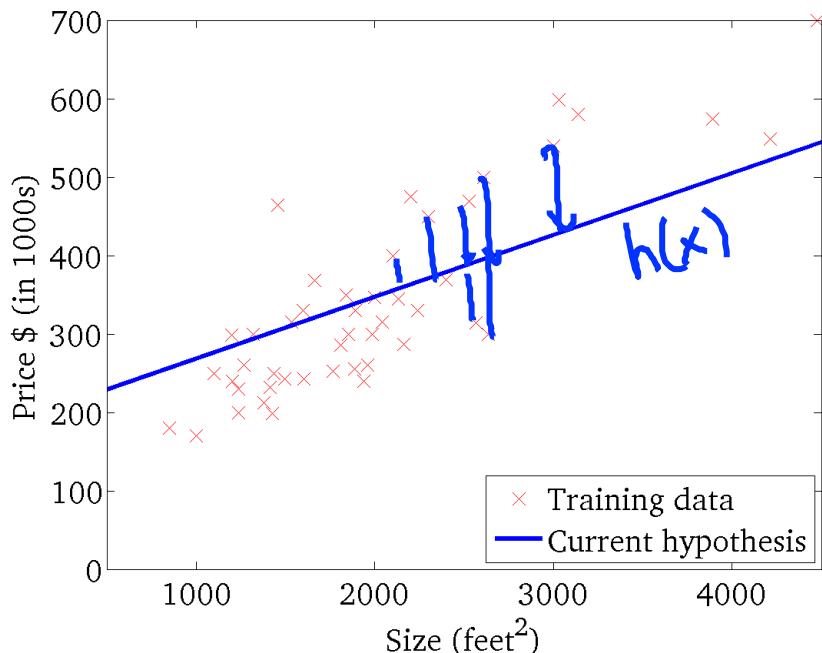
$$J(\theta_0, \theta_1)$$

(function of the parameters  $\theta_0, \theta_1$ )



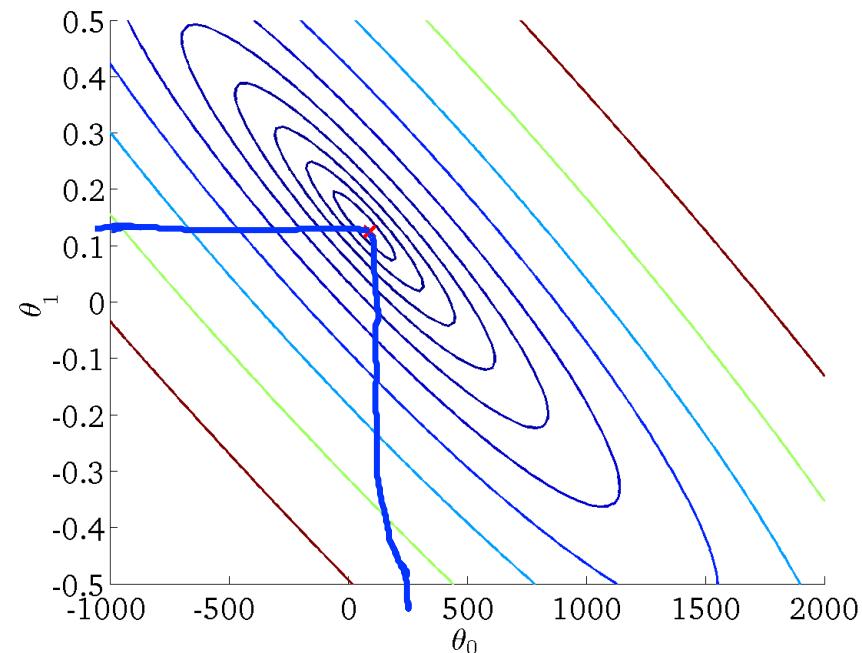
$$h_{\theta}(x)$$

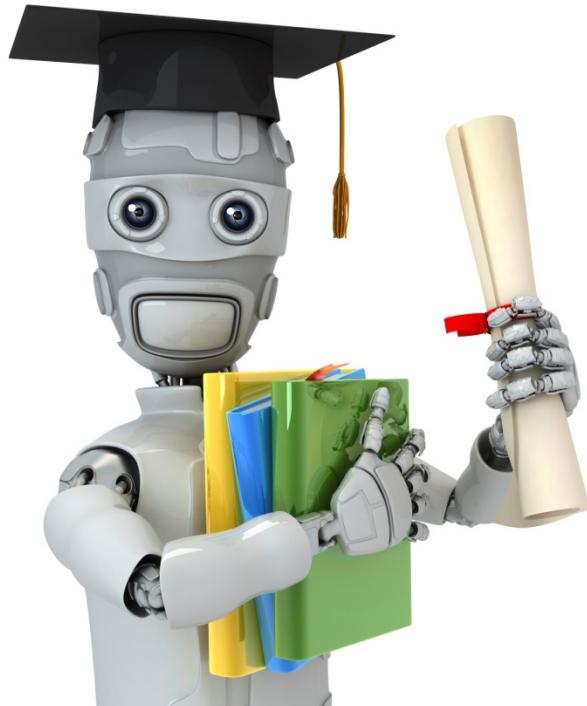
(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



$$J(\theta_0, \theta_1)$$

(function of the parameters  $\theta_0, \theta_1$ )





Machine Learning

Linear regression  
with one variable

---

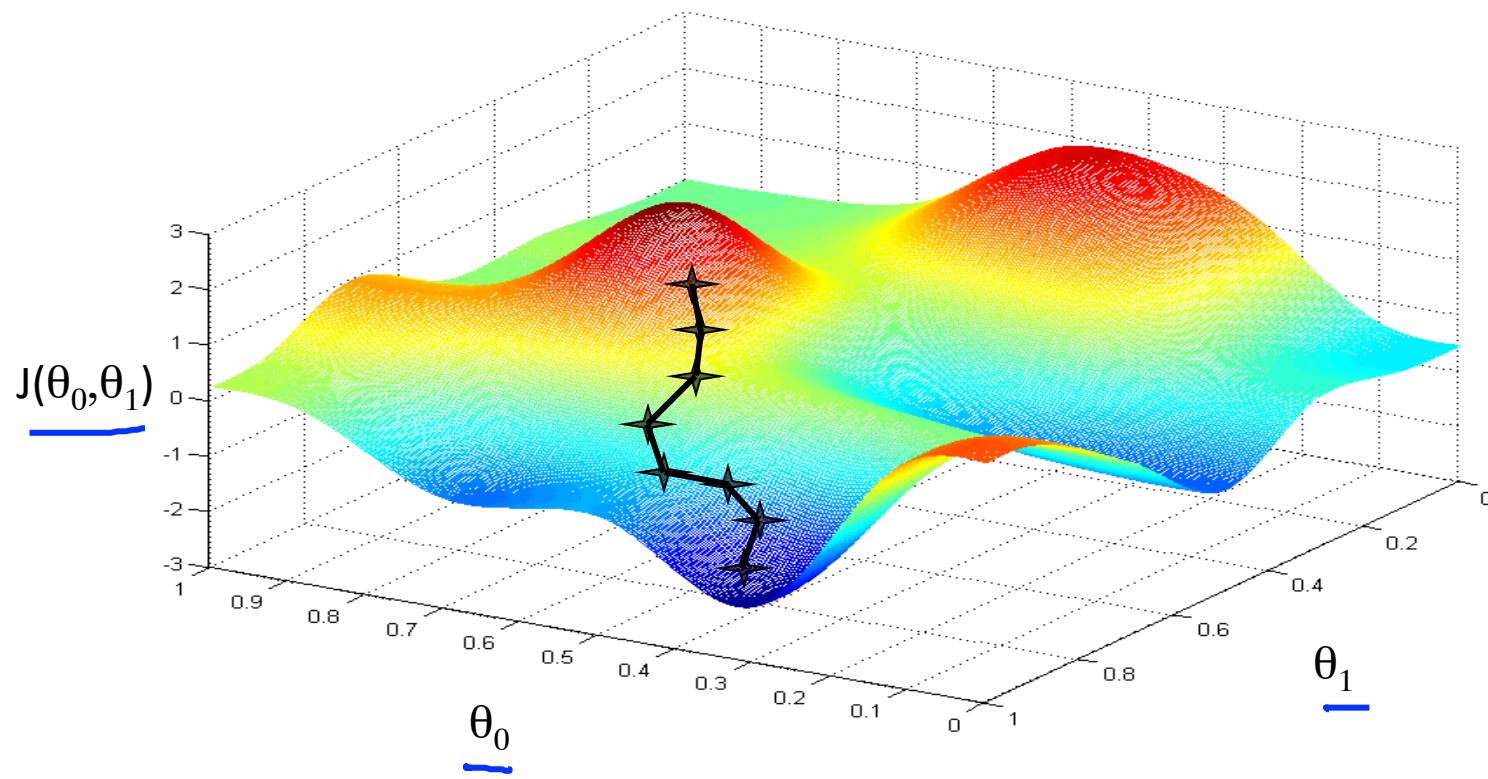
Gradient  
descent

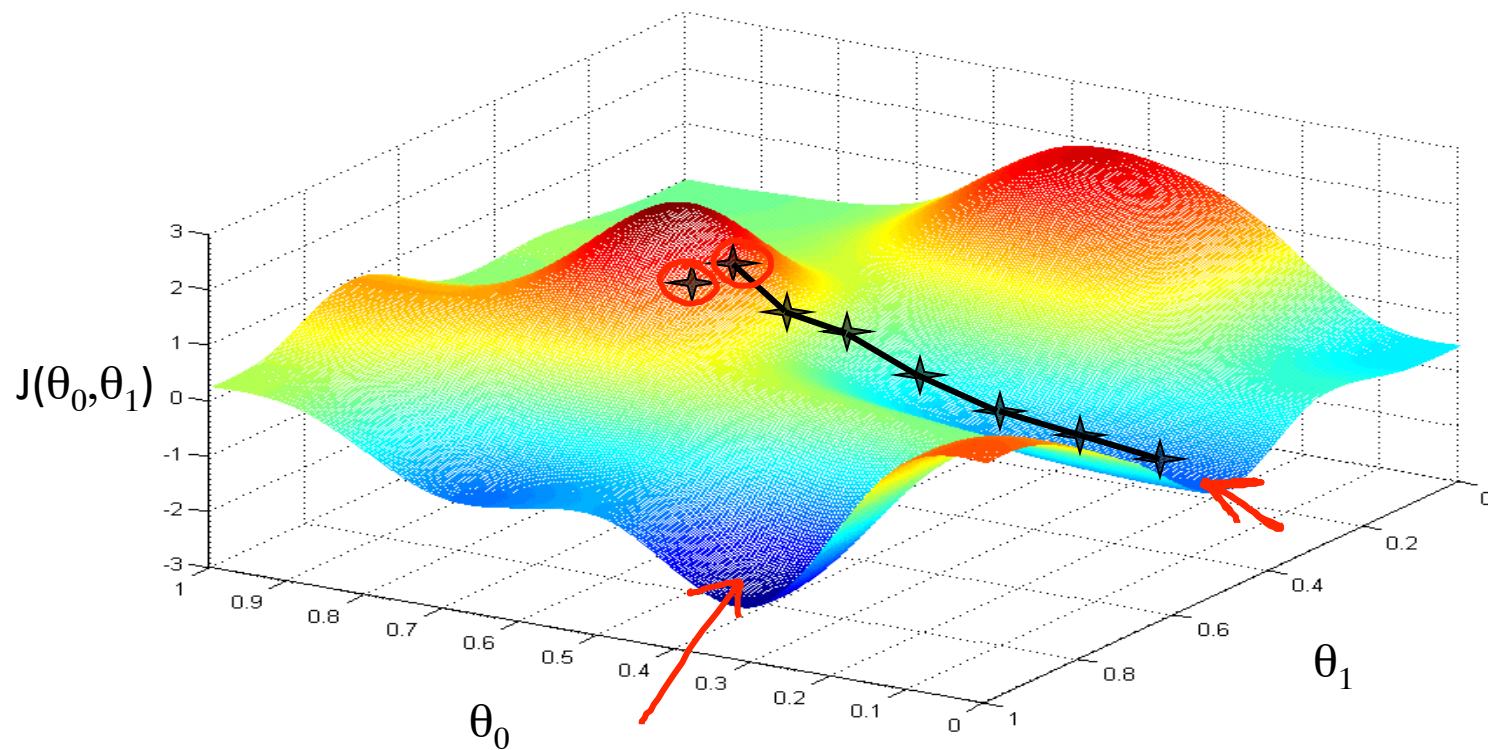
Have some function  $J(\theta_0, \theta_1)$   $\mathcal{J}(\theta_0, \theta_1, \theta_2, \dots, \theta_n)$

Want  $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$   $\min_{\theta_0, \dots, \theta_n} \mathcal{J}(\theta_0, \dots, \theta_n)$

## Outline:

- Start with some  $\theta_0, \theta_1$  (say  $\theta_0 = 0, \theta_1 = 0$ )
- Keep changing  $\theta_0, \theta_1$  to reduce  $J(\theta_0, \theta_1)$  until we hopefully end up at a minimum





# Gradient descent algorithm

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

learning rate

$\theta_0, \theta_1$

(for  $j = 0$  and  $j = 1$ )

Simultaneously update  
 $\theta_0$  and  $\theta_1$

Assignment

$$a := b$$

$$a := a + 1$$

Truth assertion

$$a = b$$

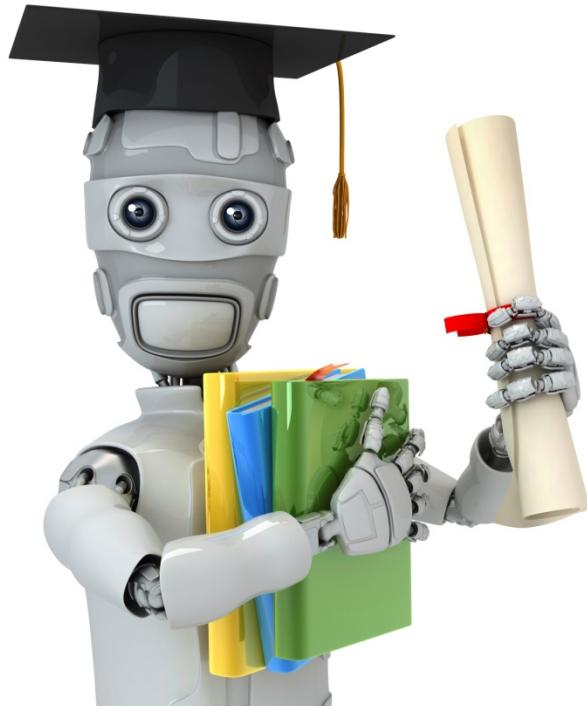
$$a = a + 1$$

Correct: Simultaneous update

- $\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$
- $\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$
- $\theta_0 := \text{temp0}$
- $\theta_1 := \text{temp1}$

Incorrect:

- $\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$
- $\theta_0 := \text{temp0}$
- $\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$
- $\theta_1 := \text{temp1}$



Machine Learning

# Linear regression with one variable

---

## Gradient descent intuition

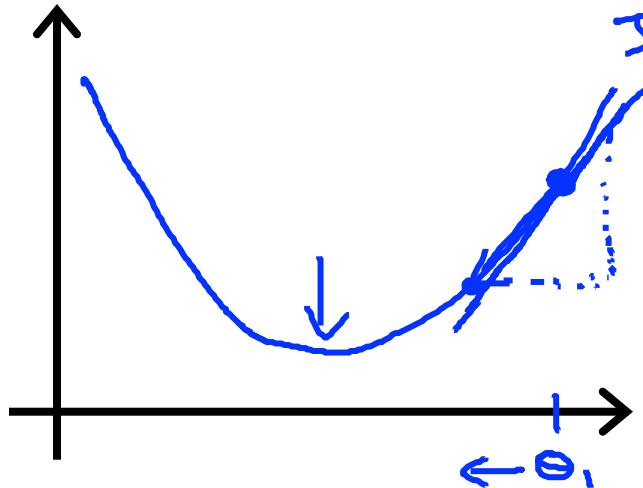
# Gradient descent algorithm

repeat until convergence {  
     $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$   
}

*learning rate*                    *derivative*

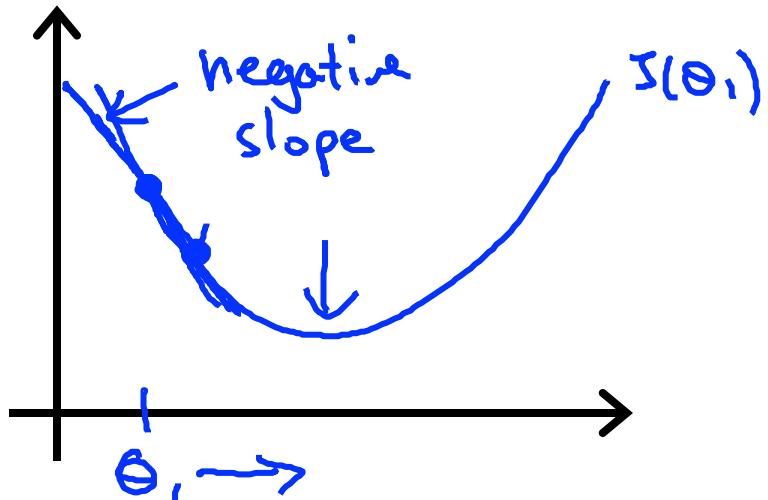
(simultaneously update  
 $j = 0$  and  $j = 1$ )

$$\min_{\theta_1} J(\theta_1) \quad \theta_1 \in \mathbb{R}$$



$J(\theta_1)$  ( $\theta_1 \in \mathbb{R}$ )

$$\theta_1 := \theta_1 - \frac{\alpha}{\frac{\partial}{\partial \theta_1} J(\theta_1)} \geq 0$$



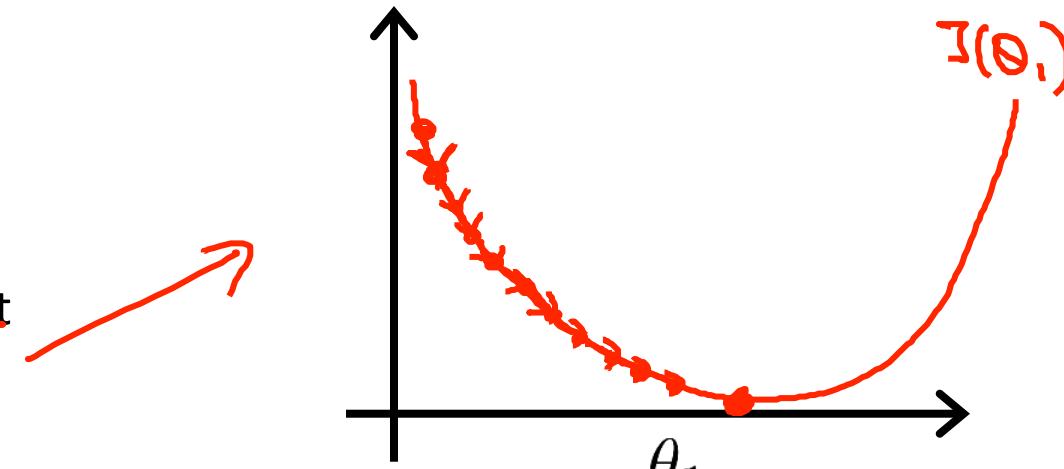
$\theta_1 := \theta_1 - \frac{\alpha}{\frac{\partial}{\partial \theta_1} J(\theta_1)}$  (positive number)

$$\frac{\frac{\partial}{\partial \theta_1} J(\theta_1)}{\leq 0}$$

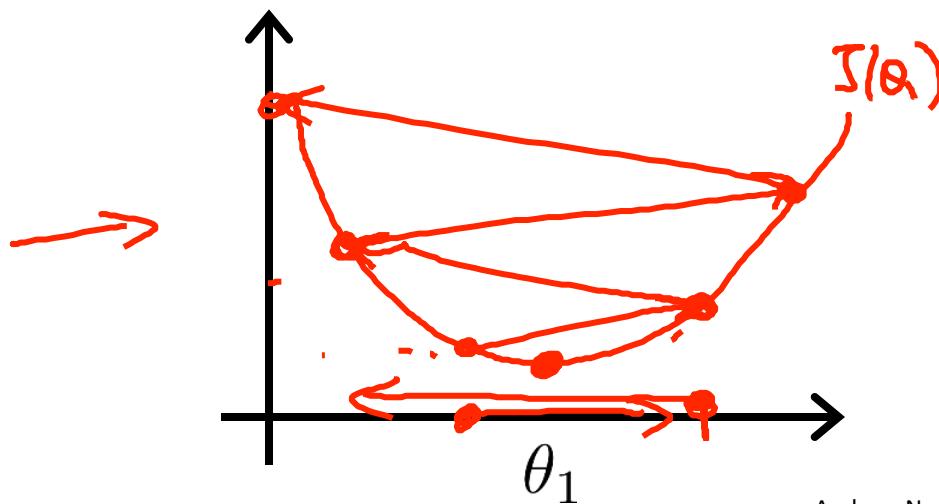
$\theta_1 := \theta_1 - \frac{\alpha}{\uparrow} \frac{\partial}{\partial \theta_1} J(\theta_1)$  (negative number)

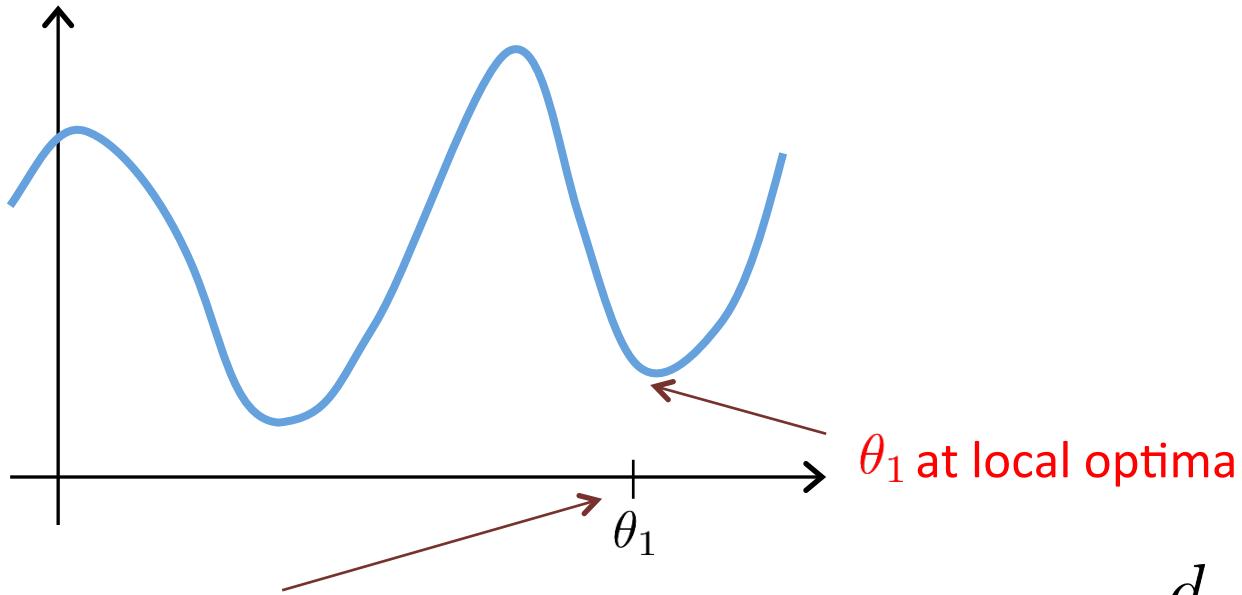
$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If  $\alpha$  is too small, gradient descent can be slow.



If  $\alpha$  is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.





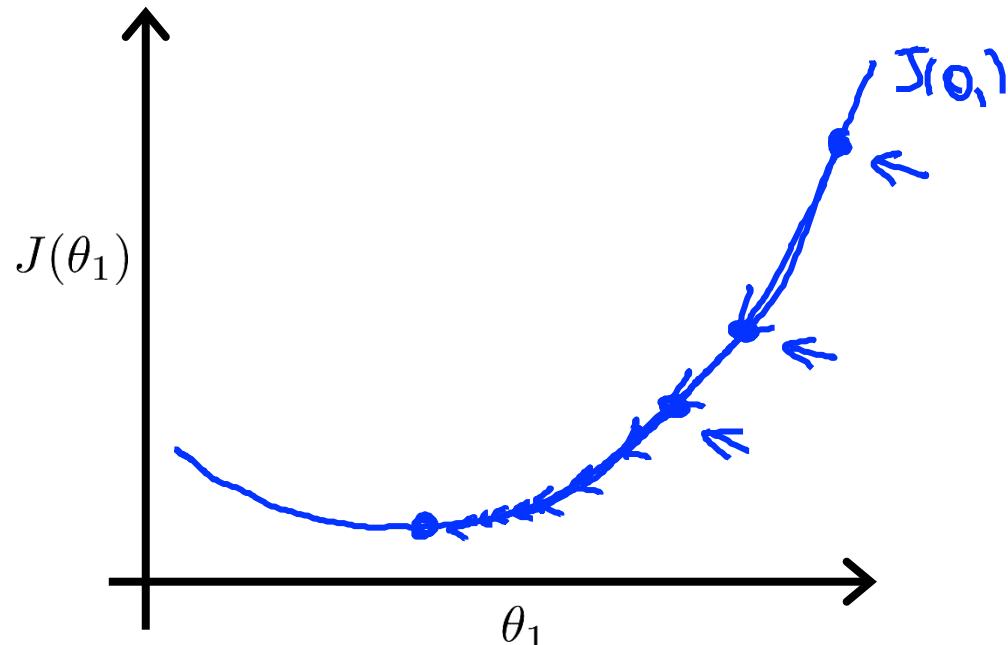
Current value of  $\theta_1$

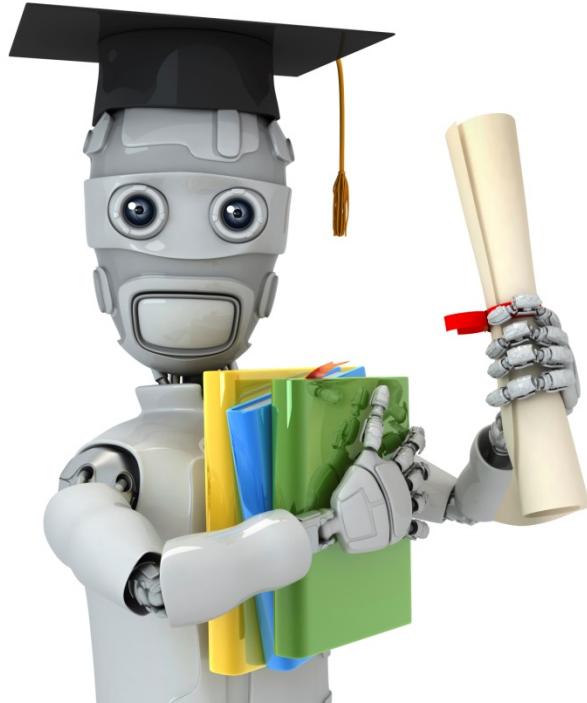
$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

Gradient descent can converge to a local minimum, even with the learning rate  $\alpha$  fixed.

$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease  $\alpha$  over time.





Machine Learning

# Linear regression with one variable

---

## Gradient descent for linear regression

## Gradient descent algorithm

```
repeat until convergence {  
     $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$   
    (for  $j = 1$  and  $j = 0$ )  
}
```

## Linear Regression Model

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{2}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

$$= \frac{2}{m} \sum_{i=1}^m (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2$$

$$j = 0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})$$

$$j = 1 : \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

## Gradient descent algorithm

repeat until convergence {

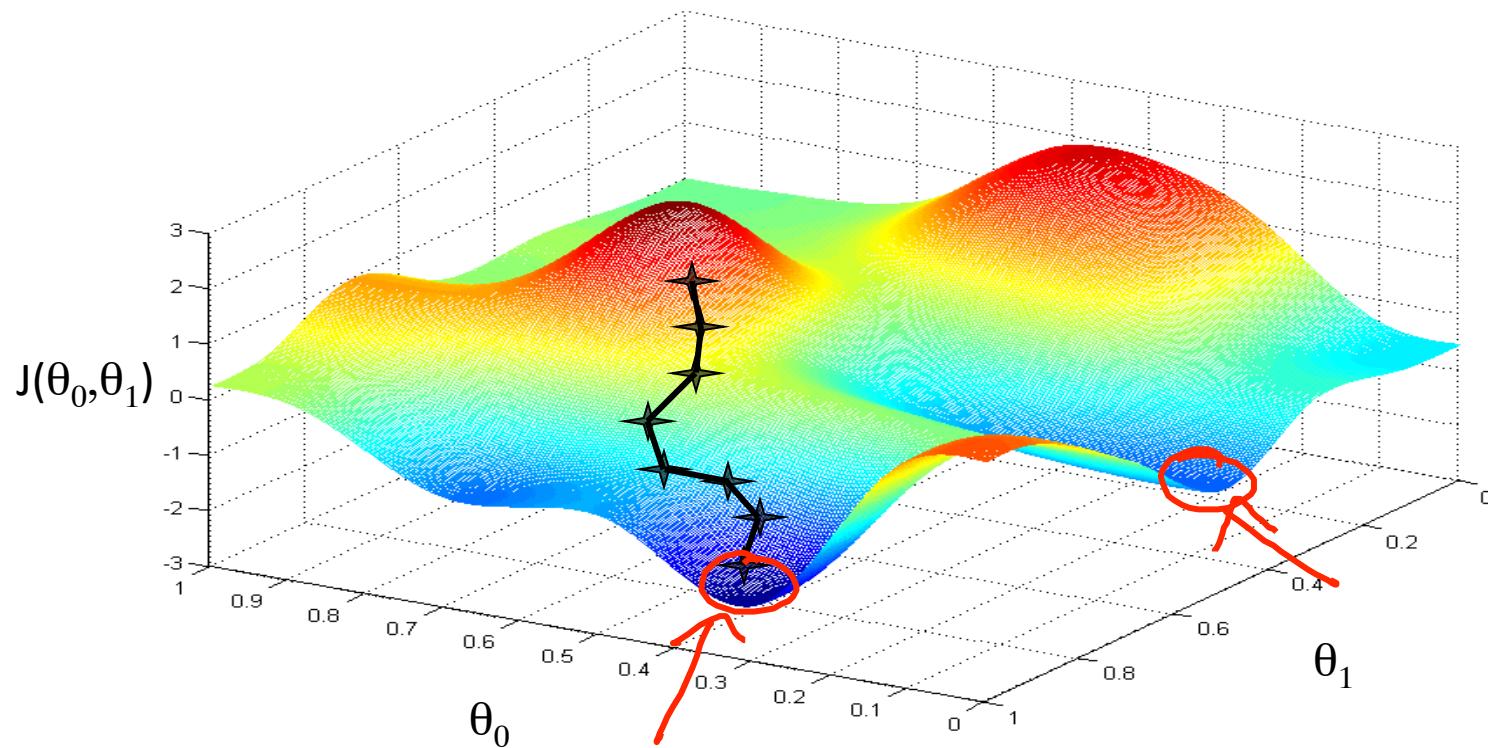
$$\theta_0 := \theta_0 - \alpha \left[ \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \right]$$
$$\theta_1 := \theta_1 - \alpha \left[ \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)} \right]$$

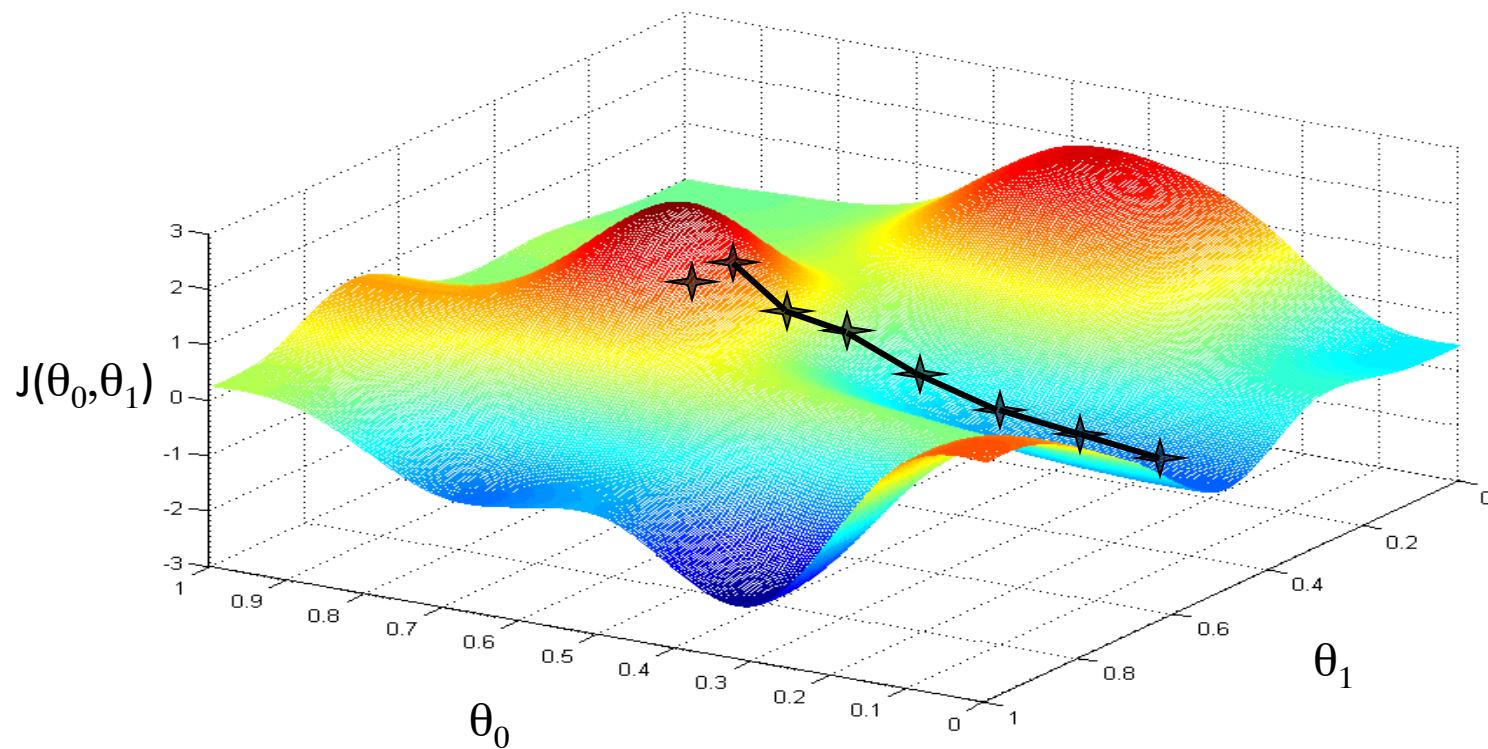
}

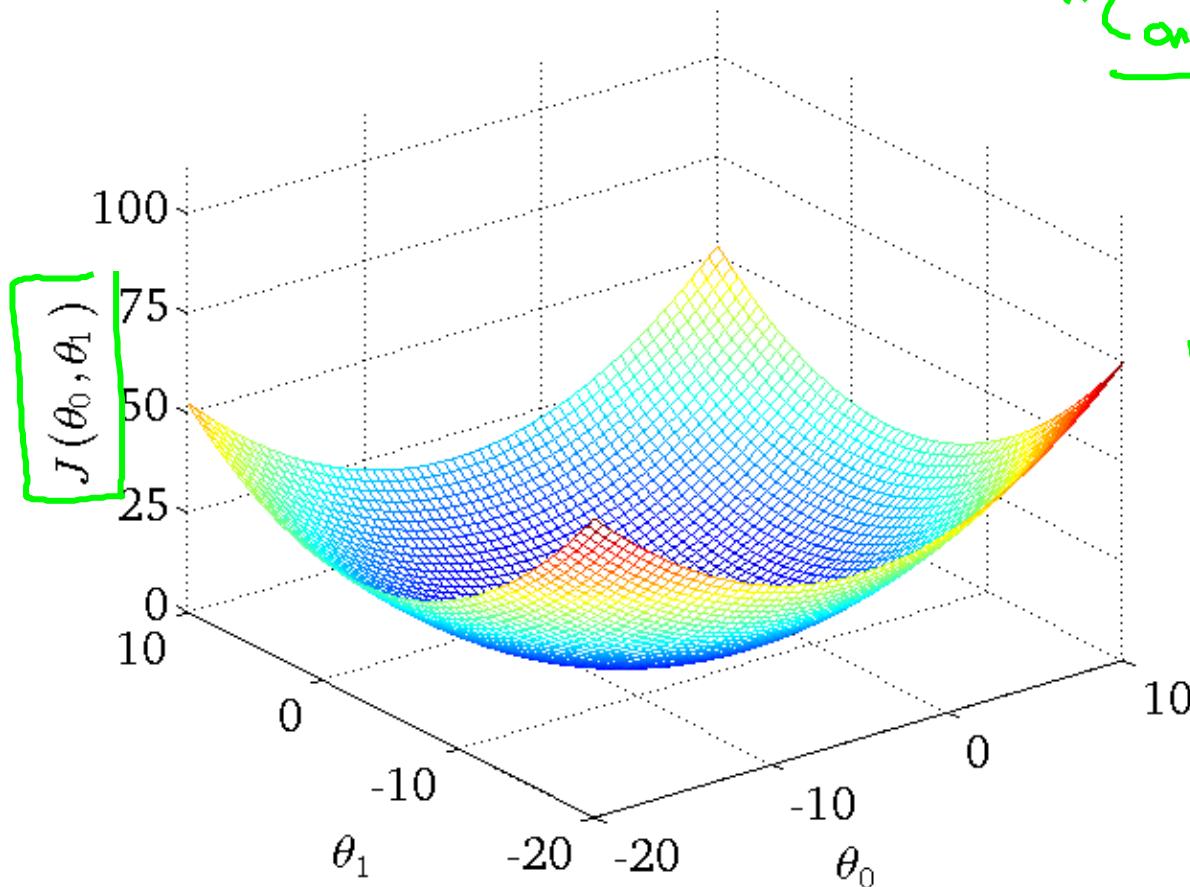
$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

update  
 $\theta_0$  and  $\theta_1$   
simultaneously

$$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

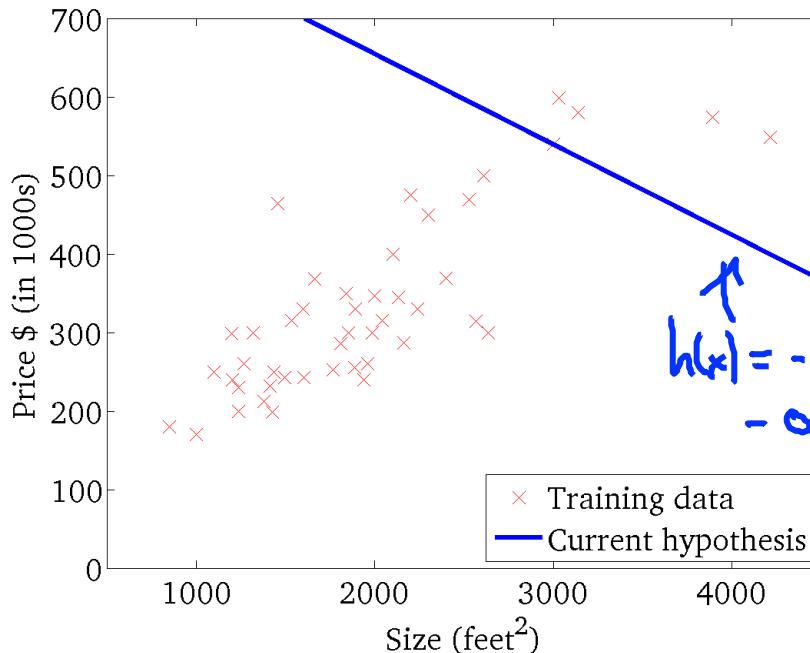






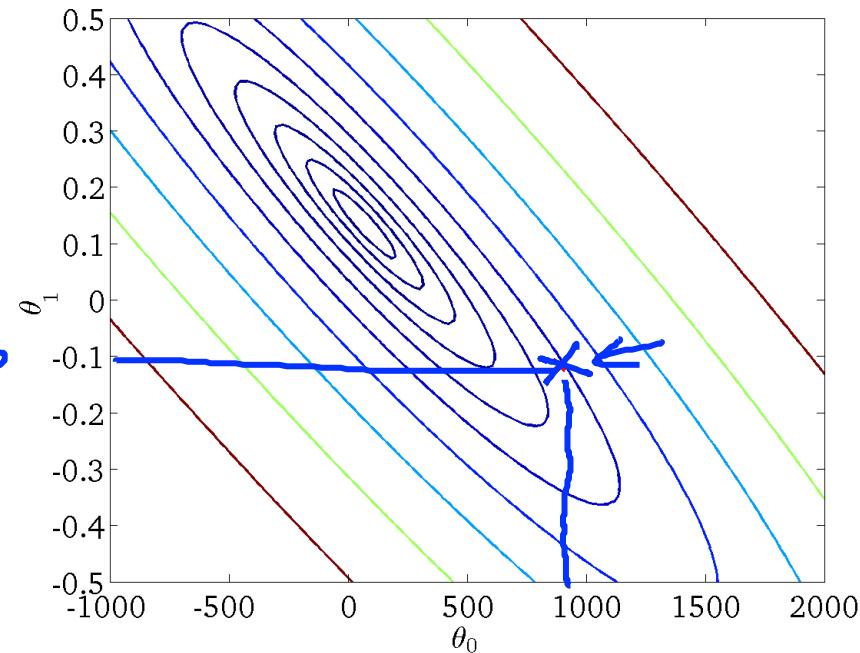
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



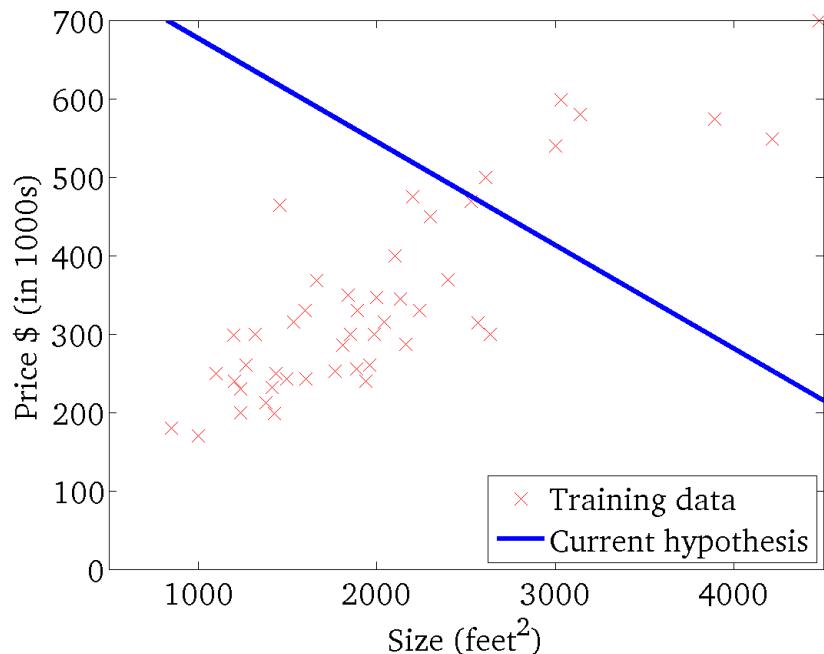
$$J(\theta_0, \theta_1)$$

(function of the parameters  $\theta_0, \theta_1$ )



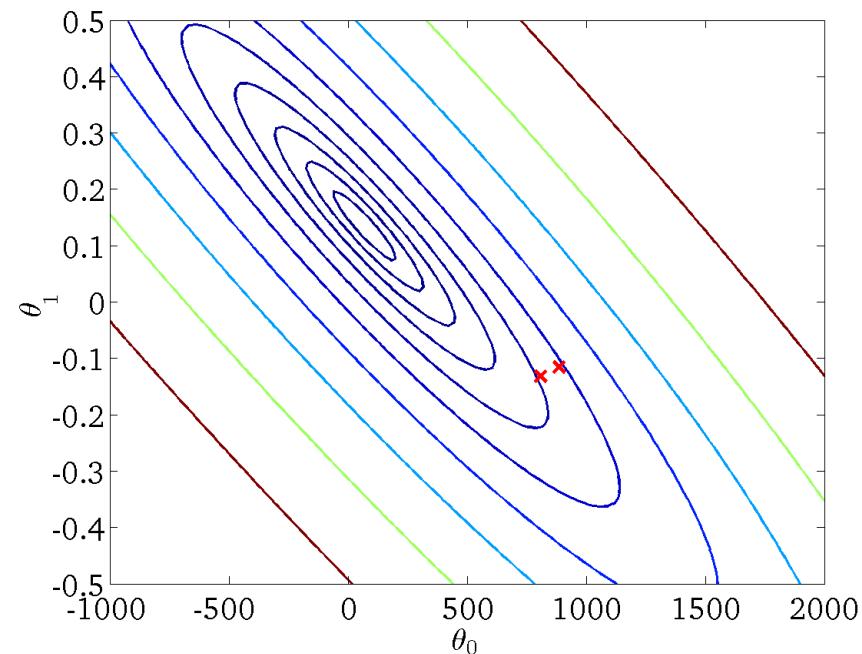
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



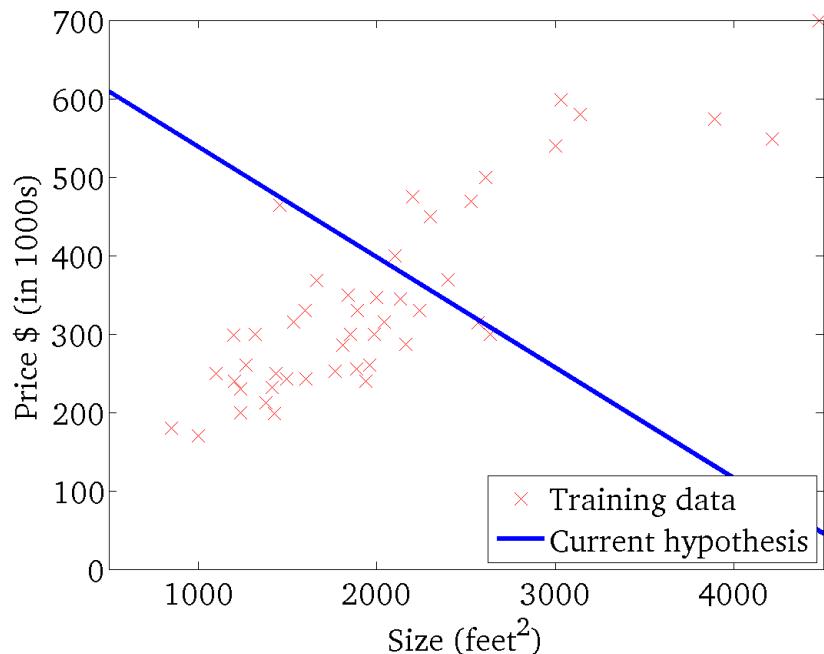
$$J(\theta_0, \theta_1)$$

(function of the parameters  $\theta_0, \theta_1$ )



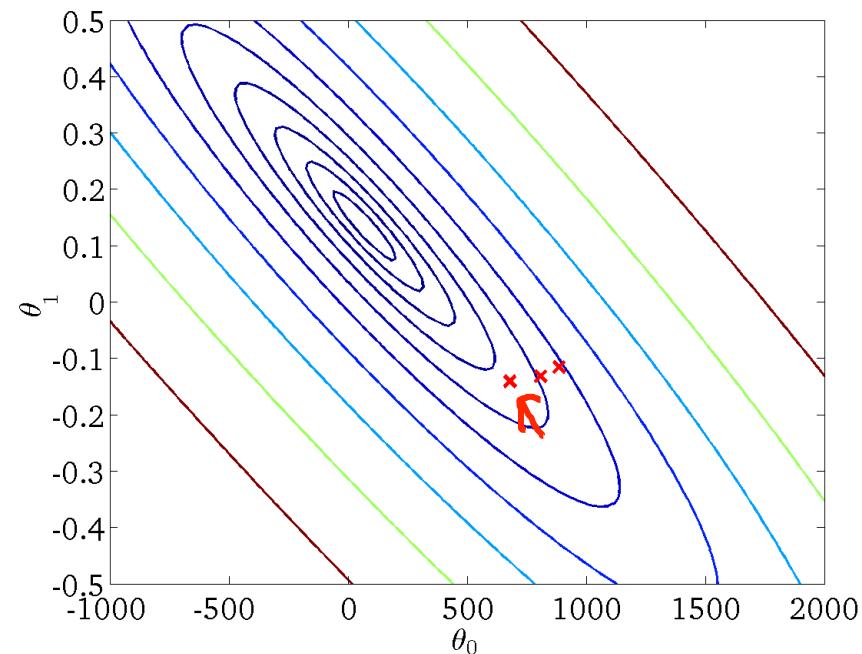
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



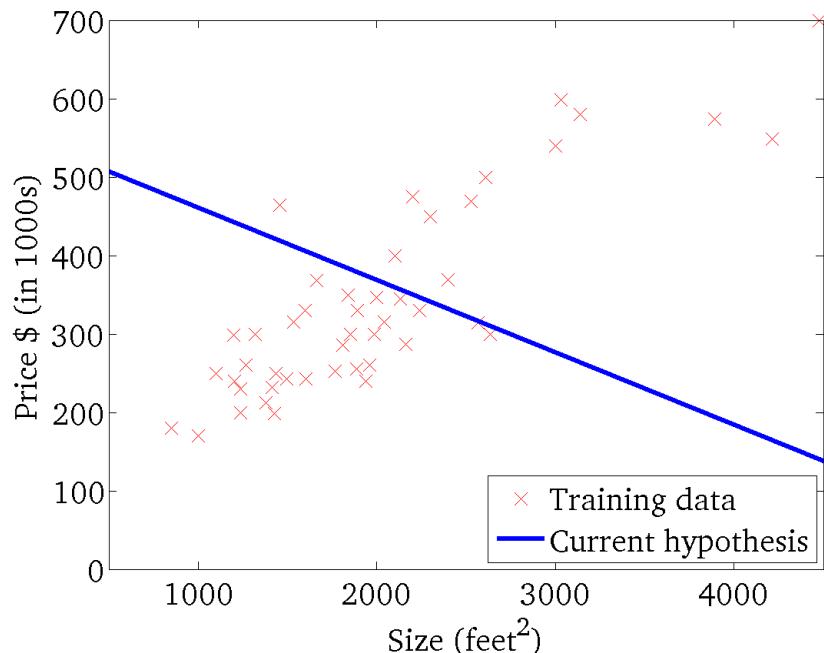
$$J(\theta_0, \theta_1)$$

(function of the parameters  $\theta_0, \theta_1$ )



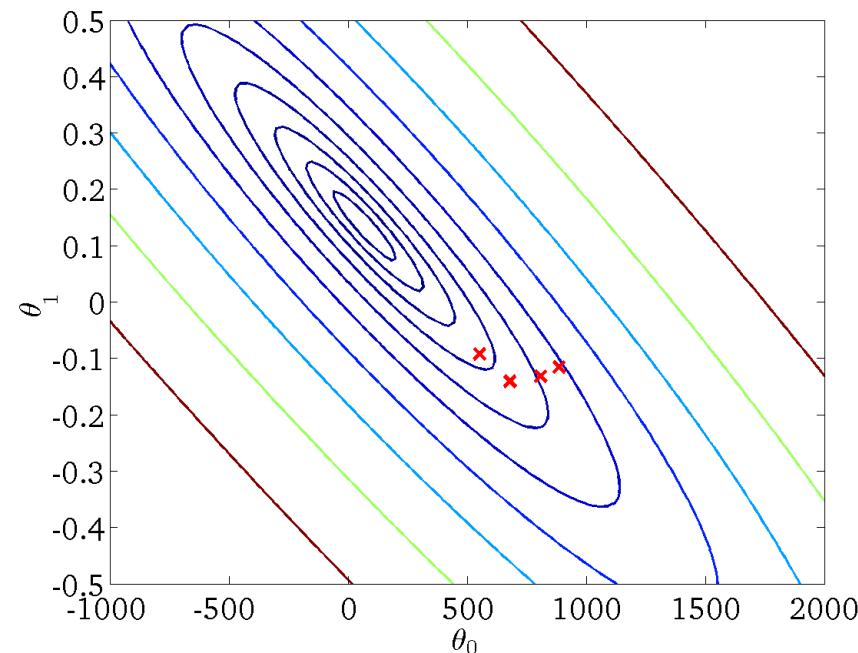
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



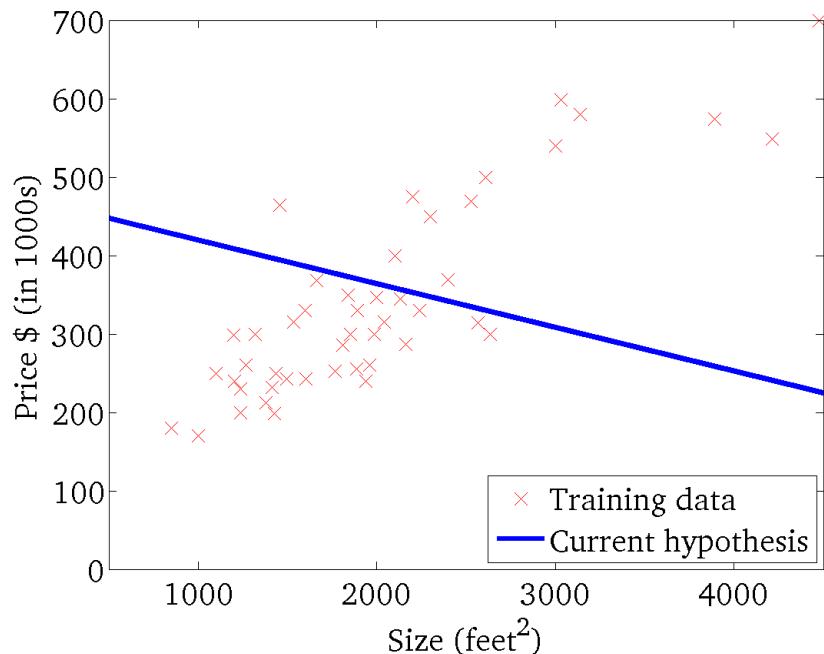
$$J(\theta_0, \theta_1)$$

(function of the parameters  $\theta_0, \theta_1$ )



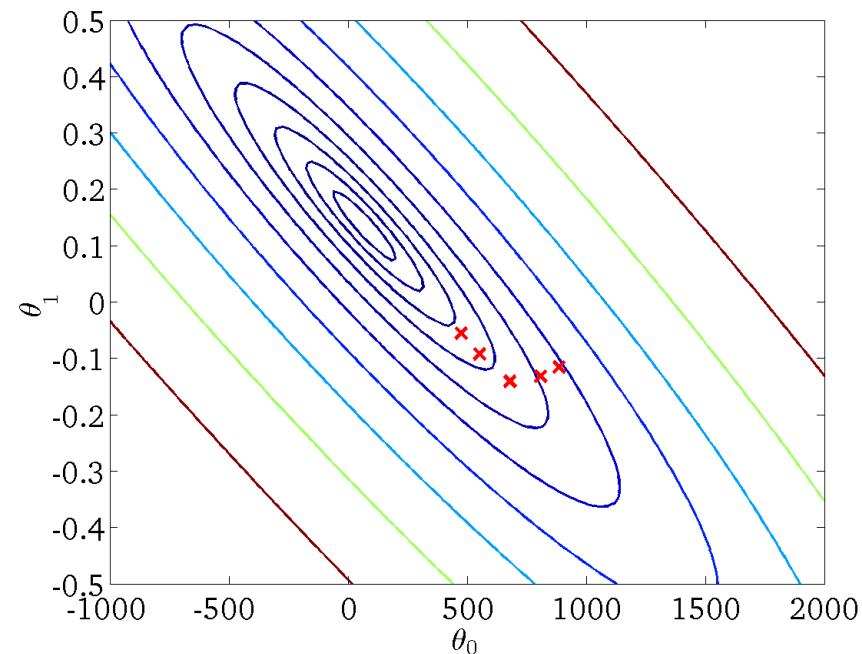
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



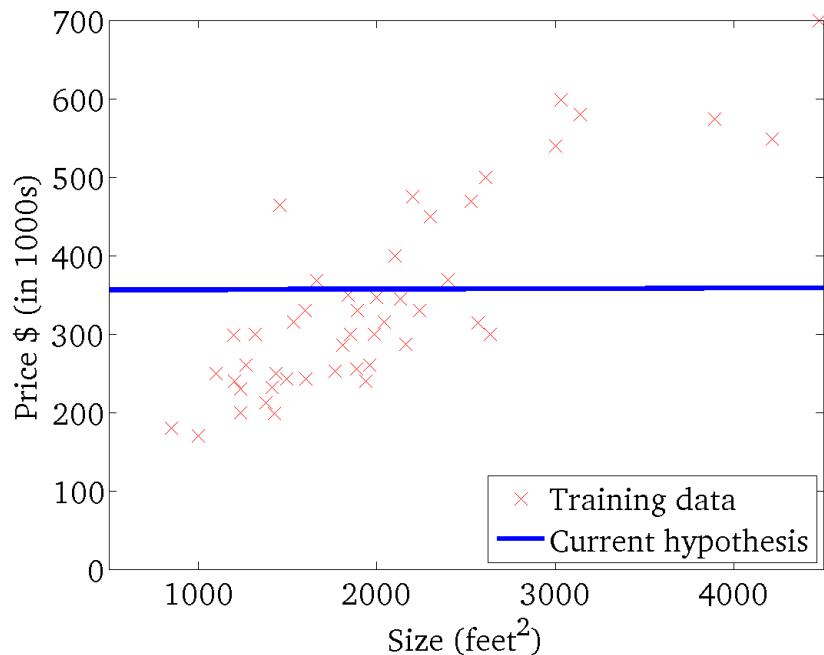
$$J(\theta_0, \theta_1)$$

(function of the parameters  $\theta_0, \theta_1$ )



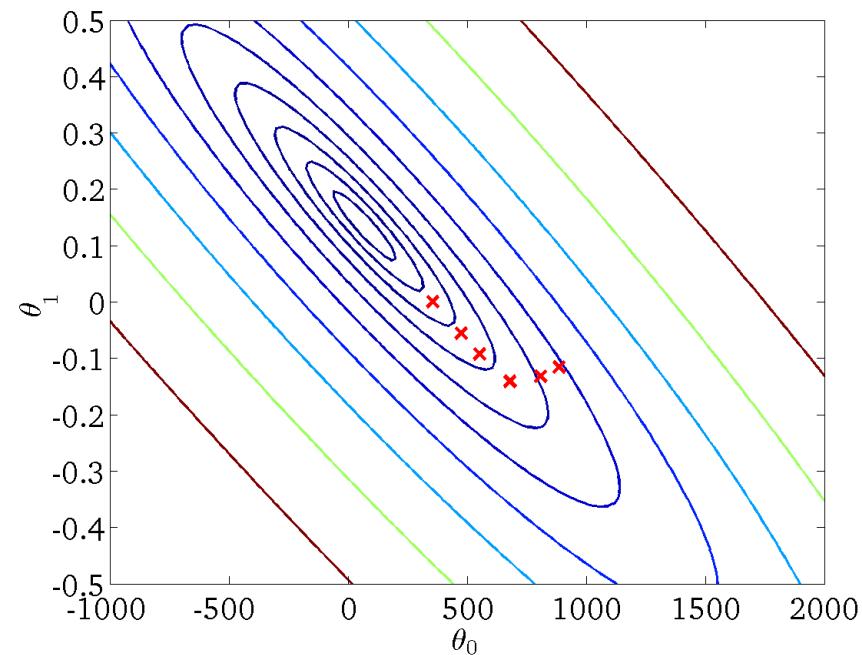
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



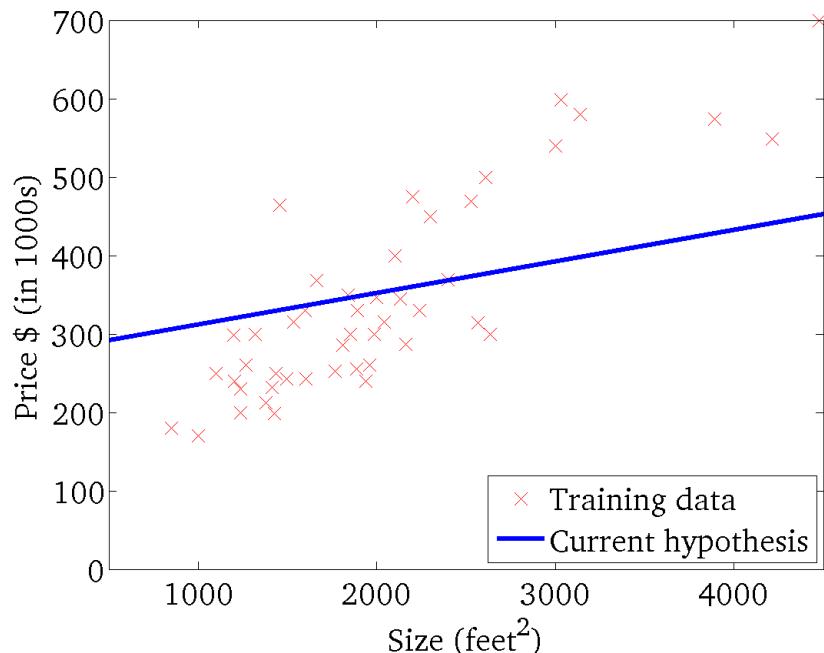
$$J(\theta_0, \theta_1)$$

(function of the parameters  $\theta_0, \theta_1$ )



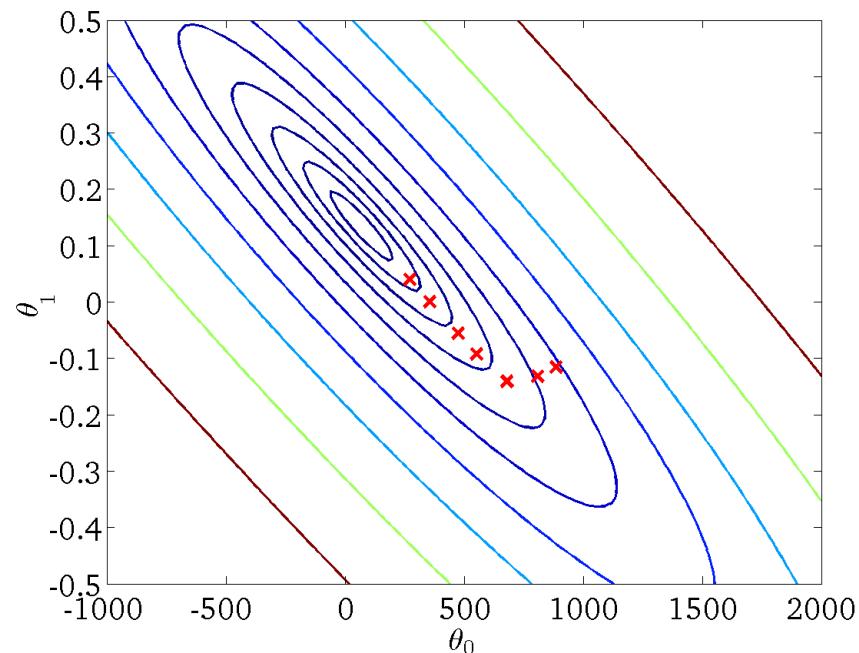
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



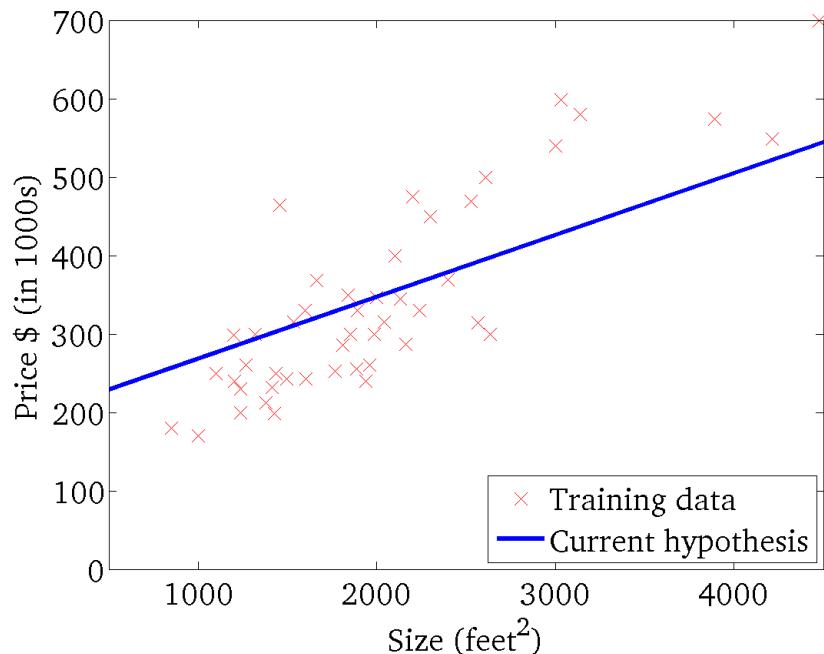
$$J(\theta_0, \theta_1)$$

(function of the parameters  $\theta_0, \theta_1$ )



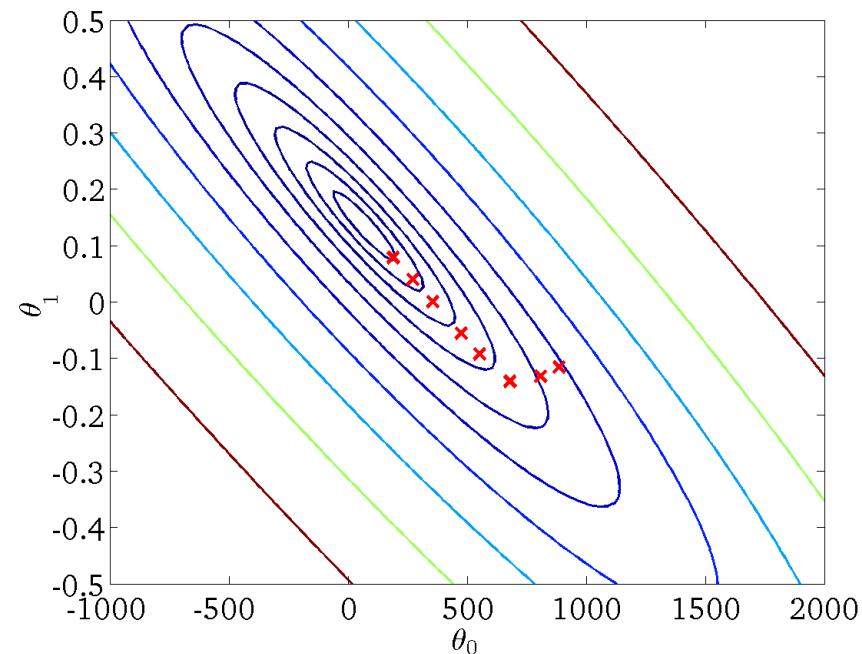
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



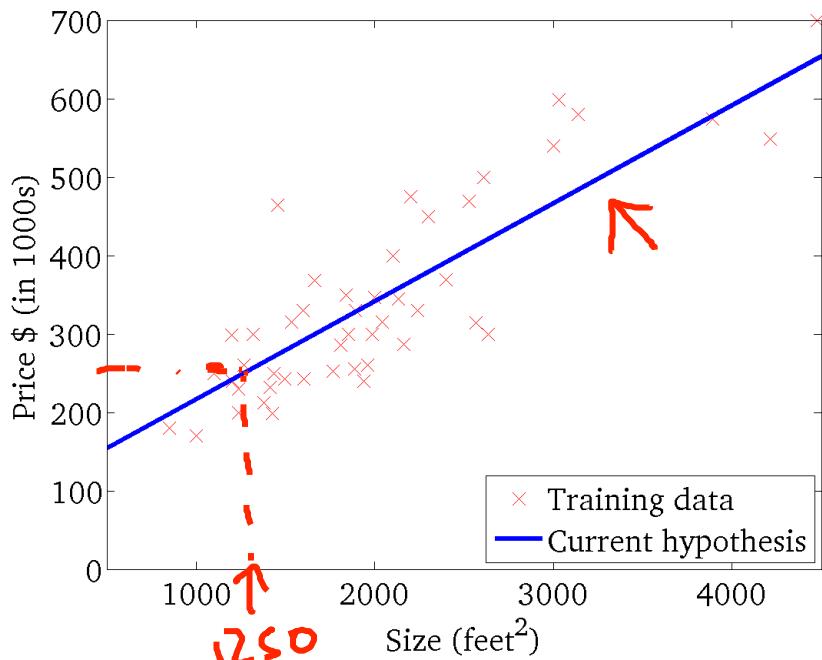
$$J(\theta_0, \theta_1)$$

(function of the parameters  $\theta_0, \theta_1$ )



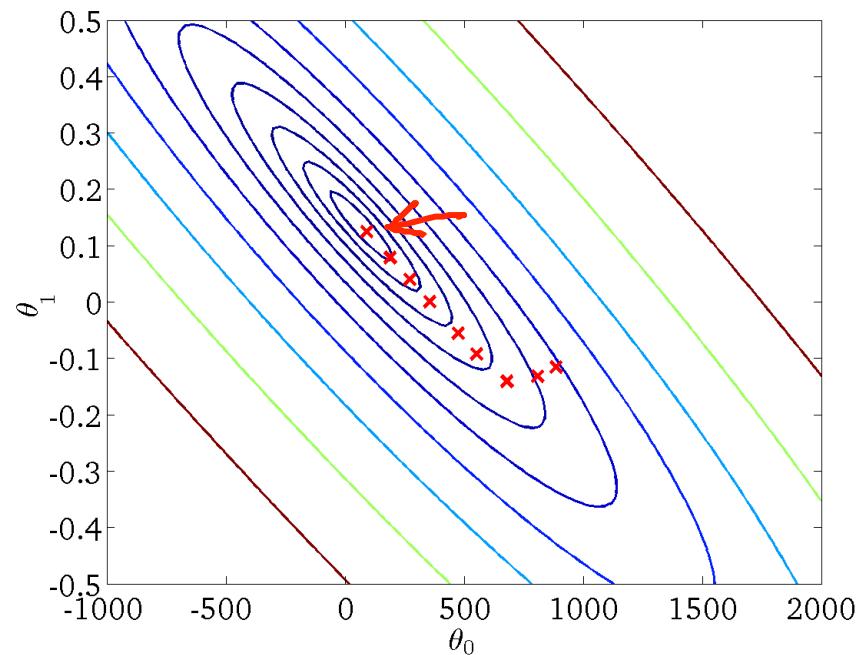
$$h_{\theta}(x)$$

(for fixed  $\theta_0, \theta_1$ , this is a function of  $x$ )



$$J(\theta_0, \theta_1)$$

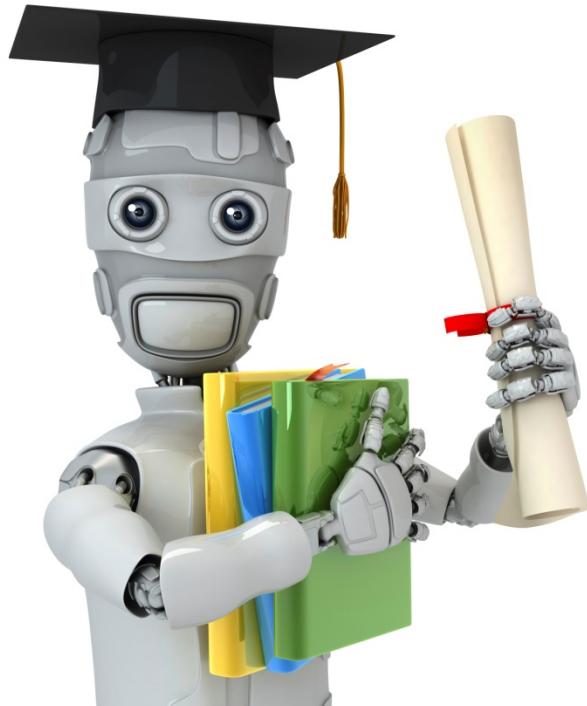
(function of the parameters  $\theta_0, \theta_1$ )



## “Batch” Gradient Descent

“Batch”: Each step of gradient descent uses all the training examples.

$$\xrightarrow{\text{all}} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})$$



Machine Learning

# Linear Algebra review (optional)

---

## Matrices and vectors

**Matrix:** Rectangular array of numbers:

$$\begin{array}{c} \rightarrow \\ \rightarrow \\ \rightarrow \\ \rightarrow \end{array} \left[ \begin{array}{cc} 1402 & 191 \\ 1371 & 821 \\ 949 & 1437 \\ 147 & 1448 \end{array} \right] \quad \begin{array}{c} \nearrow \\ \nearrow \\ \nearrow \\ \nearrow \end{array}$$

$4 \times 2$  matrix

$$\rightarrow [R^{4 \times 2}]$$

$$2 \rightarrow \left[ \begin{array}{ccc} 1 & 2 & 3 \\ 4 & 5 & 6 \end{array} \right] \quad \begin{array}{c} \uparrow \\ \uparrow \\ \uparrow \\ 3 \end{array} \quad \begin{array}{c} \uparrow \\ \uparrow \\ \uparrow \\ C \end{array}$$

$2 \times 3$  matrix

$$[R^{2 \times 3}]$$

Dimension of matrix: number of rows  $\times$  number of columns

## Matrix Elements (entries of matrix)

$$A = \begin{bmatrix} 1402 & 191 \\ 1371 & 821 \\ 949 & 1437 \\ 147 & 1448 \end{bmatrix}$$

$A_{ij}$  = “ $i, j$  entry” in the  $i^{th}$  row,  $j^{th}$  column.

$$A_{11} = 1402$$

$$A_{12} = 191$$

$$A_{32} = 1437$$

$$A_{41} = 147$$

$$\cancel{A_{33}} = \text{undefined (error)}$$

Vector: An  $n \times 1$  matrix.

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

$$n = 4$$

$\leftarrow$  4-dimensional vector

$$\mathbb{R}^{3 \times 2}$$

$$\underline{\mathbb{R}^4}$$

$y_i = i^{th}$  element

$$y_1 = 460$$

$$y_2 = 232$$

$$y_3 = 315$$

$\rightarrow [A, B, C, X]$

$a, b, x, y$

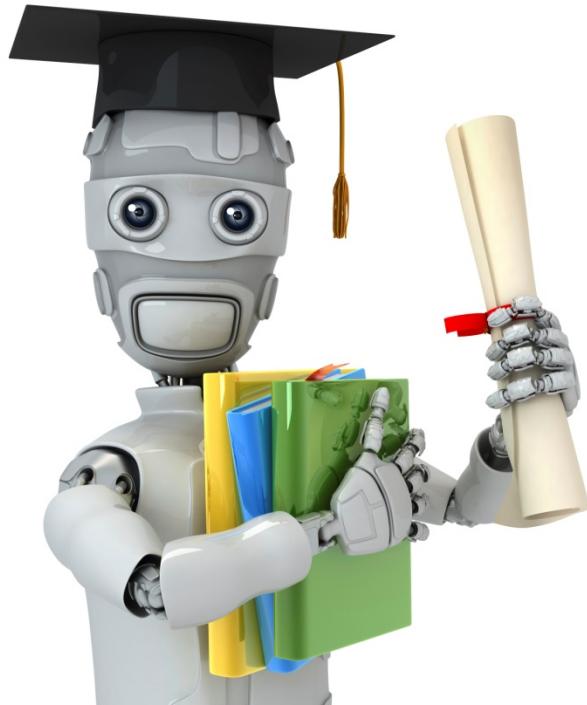
1-indexed vs 0-indexed:

$$y[1] \quad y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \quad \leftarrow$$

1-indexed

$$y[0] \quad y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \quad \leftarrow$$

0-indexed



Machine Learning

# Linear Algebra review (optional)

---

## Addition and scalar multiplication

# Matrix Addition

$$\begin{array}{c}
 \begin{array}{cc}
 \downarrow & \downarrow \\
 \boxed{1} & 0 \\
 \boxed{2} & 5 \\
 \boxed{3} & 1
 \end{array} & + & \begin{array}{cc}
 \boxed{4} & 0.5 \\
 \boxed{2} & 5 \\
 \boxed{0} & 1
 \end{array} & = & \begin{array}{cc}
 5 & 0.5 \\
 4 & 10 \\
 3 & 2
 \end{array}
 \end{array}$$

$3 \times 2$

A hand-drawn diagram illustrating matrix multiplication. It shows two 2x2 matrices being multiplied. The first matrix has columns [1, 2] and rows [3, 2]. The second matrix has columns [0, 4] and rows [5, 2]. The result is a 2x2 matrix with columns [1, 2] and rows [3, 2], which is the identity matrix  $I_2$ . The word "error" is written next to the result.

# Scalar Multiplication

real number

$$3 \times \begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} = \begin{array}{c} \text{Diagram of a 3x2 matrix with entries 3, 6, 9, 0, 15, 3, enclosed in a large bracket. An arrow points from the first column of the original matrix to this diagram.} \\ \boxed{\begin{bmatrix} 3 & 0 \\ 6 & 15 \\ 9 & 3 \end{bmatrix}} \end{array} = \begin{bmatrix} 1 & 0 \\ 2 & 5 \\ 3 & 1 \end{bmatrix} \times 3$$

$$\left[ \begin{array}{cc} 4 & 0 \\ 6 & 3 \end{array} \right] / 4 = \frac{1}{4} \left[ \begin{array}{cc} 4 & 0 \\ 6 & 3 \end{array} \right] = \left[ \begin{array}{cc} -\frac{1}{2} & 0 \\ \frac{3}{2} & \frac{3}{4} \end{array} \right]$$

# Combination of Operands

$$\begin{aligned} & 3 \times \begin{bmatrix} 1 \\ 4 \\ 2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix} - \begin{bmatrix} 3 \\ 0 \\ 2 \end{bmatrix} / 3 \\ & = \begin{bmatrix} 3 \\ 12 \\ 6 \end{bmatrix} + \begin{bmatrix} 6 \\ 0 \\ 5 \end{bmatrix} - \begin{bmatrix} 1 \\ 0 \\ \frac{2}{3} \end{bmatrix} \\ & = \begin{bmatrix} 2 \\ 12 \\ 10 \frac{1}{3} \end{bmatrix} \end{aligned}$$

Scalar multiplication

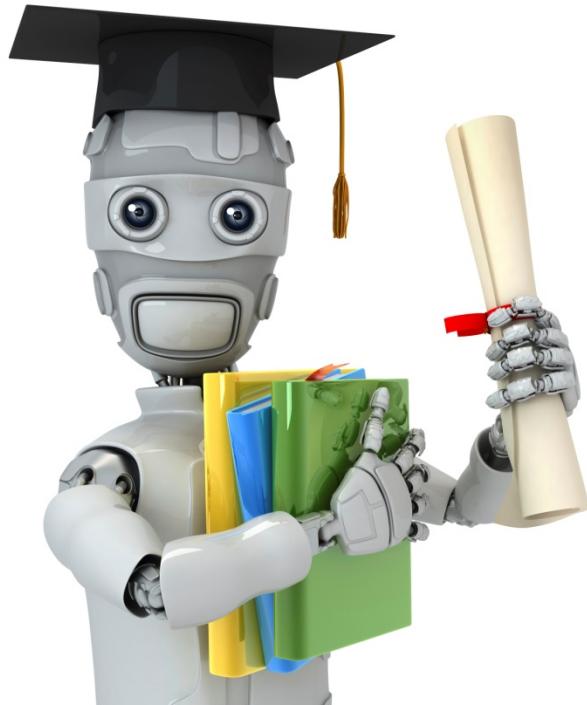
Scalar division

Matrix subtraction / Vector subtraction

Matrix addition / Vector addition

3x1 matrix

3-dimensional vector



Machine Learning

## Linear Algebra review (optional)

---

### Matrix-vector multiplication

# Example

$$\begin{matrix} & \begin{matrix} 1 & 3 \\ 4 & 0 \\ 2 & 1 \end{matrix} \\ \underbrace{\quad\quad}_{3 \times 2} & \end{matrix} \begin{matrix} 1 \\ 5 \end{matrix} = \begin{bmatrix} 16 \\ 4 \\ 7 \end{bmatrix}$$

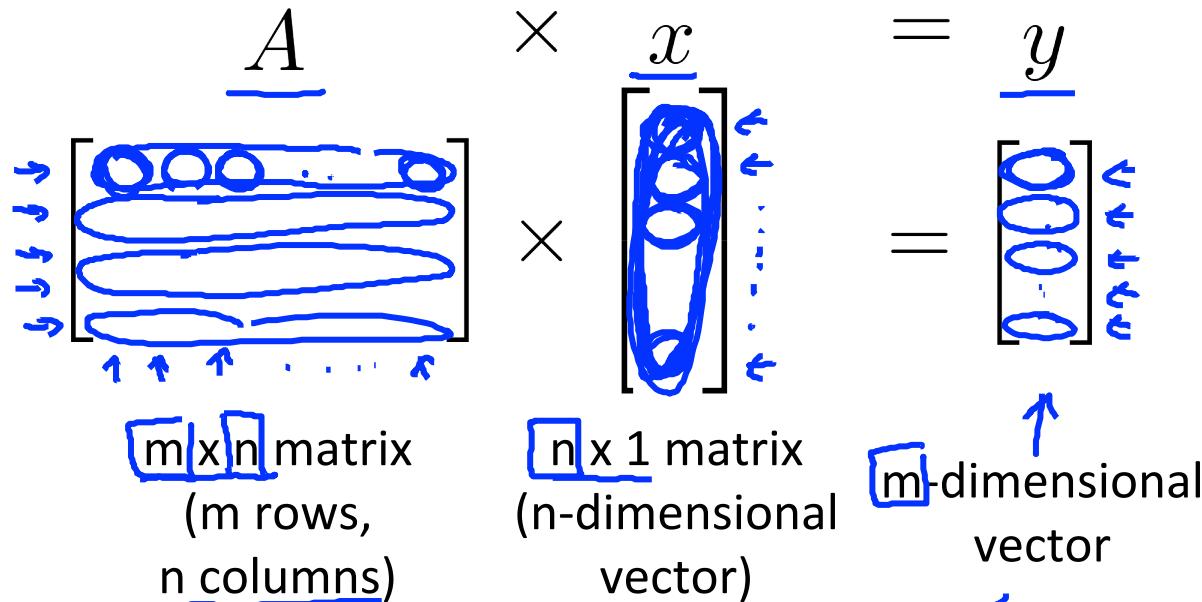
*3x1 matrix*

$$1 \times 1 + 3 \times 5 = 16$$

$$4 \times 1 + 0 \times 5 = 4$$

$$2 \times 1 + 1 \times 5 = 7$$

## Details:



To get  $y_i$ , multiply  $A$ 's  $i^{th}$  row with elements of vector  $x$ , and add them up.

# Example

$$\begin{bmatrix} 1 & 2 & 1 & 5 \\ 0 & 3 & 0 & 4 \\ -1 & -2 & 0 & 0 \end{bmatrix}$$

$3 \times 4$

$$\begin{array}{c} \downarrow \\ \begin{bmatrix} 1 \\ 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 14 \\ 13 \\ -7 \end{bmatrix} = \begin{bmatrix} 14 \\ 13 \\ -7 \end{bmatrix} \end{array}$$

$4 \times 1$        $3 \times 1$

$$1 \times 1 + 2 \times 3 + 1 \times 2 + 5 \times 1 = 14 ]$$

$$0 \times 1 + 3 \times 3 + 0 \times 2 + 4 \times 1 = 13 ]$$

$$-1 \times 1 + (-2) \times 3 + 0 \times 2 + 0 \times 1 = -7 ]$$

House sizes:

- 2104
- 1416
- 1534
- 852

Matrix  $x$

	$4 \times 2$
1	2104
1	1416
1	1534
1	852

$$h_{\theta}(x) = -40 + 0.25x$$

$$h_{\theta}(x)$$

$2 \times 1$

Vector

$$\begin{bmatrix} -40 \\ 0.25 \end{bmatrix}$$

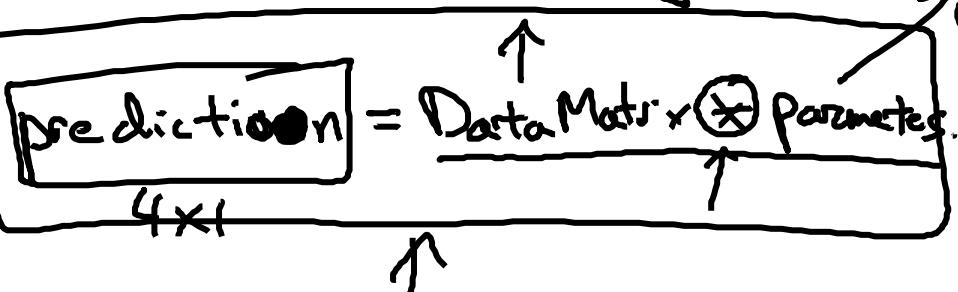
$\times$

$4 \times 1$  matrix

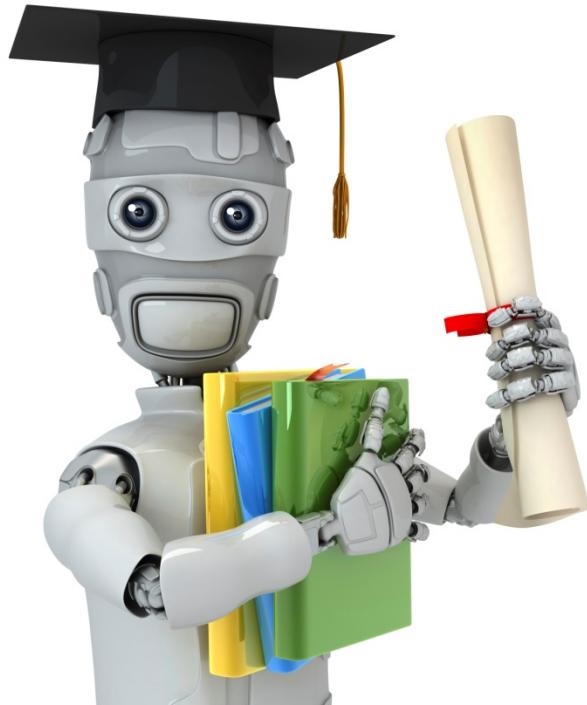
$$\begin{bmatrix} -40 \times 1 + 0.25 \times 2104 \\ -40 \times 1 + 0.25 \times 1416 \\ \vdots \\ -40 \times 1 + 0.25 \times 852 \end{bmatrix}$$

$$h_{\theta}(2104)$$

$$h_{\theta}(1416)$$



for  $i = 1: 1000$ ,  
 $\text{prediction}(i) := \dots$



Machine Learning

## Linear Algebra review (optional)

---

## Matrix-matrix multiplication

# Example

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \underbrace{\begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix} \times \begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix}}_{\textcircled{2} \times 3} = \begin{bmatrix} 11 & 10 & 14 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \times \underbrace{\begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix}}_{\textcircled{3} \times 1} = \begin{bmatrix} 11 \\ 9 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 & 2 \\ 4 & 0 & 1 \end{bmatrix} \times \underbrace{\begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix}}_{\textcircled{3} \times 1} = \begin{bmatrix} 10 \\ 14 \end{bmatrix}$$

## Details:

$$\begin{matrix} \underline{A} \\ \left[ \begin{array}{c} \end{array} \right] \end{matrix} \times \begin{matrix} \underline{B} \\ \left[ \begin{array}{c} \end{array} \right] \end{matrix} = \underline{C} = \begin{matrix} \underline{C} \\ \left[ \begin{array}{c} \end{array} \right] \end{matrix}$$

*A* is an  $m \times n$  matrix ( $m$  rows,  $n$  columns).  
*B* is an  $n \times o$  matrix ( $n$  rows,  $o$  columns).  
*C* is an  $m \times o$  matrix.

The  $i^{th}$  column of the matrix  $\underline{C}$  is obtained by multiplying  $\underline{A}$  with the  $i^{th}$  column of  $\underline{B}$ . (for  $i = 1, 2, \dots, o$ )

# Example

$$\begin{bmatrix} 1 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 3 & 2 \end{bmatrix} = \begin{bmatrix} 9 & 7 \\ 15 & 12 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 0 \\ 3 \end{bmatrix} = \begin{bmatrix} 1 \times 0 + 3 \times 3 \\ 2 \times 0 + 5 \times 3 \end{bmatrix} = \begin{bmatrix} 9 \\ 15 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 3 \\ 2 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 1 \times 1 + 3 \times 2 \\ 2 \times 1 + 5 \times 2 \end{bmatrix} = \begin{bmatrix} 7 \\ 12 \end{bmatrix}$$

House sizes:

$$\left\{ \begin{array}{r} 2104 \\ 1416 \\ 1534 \\ \hline 852 \end{array} \right.$$

Have 3 competing hypotheses:

$$1. h_{\theta}(x) = -40 + 0.25x$$

$$2. h_{\theta}(x) = 200 + 0.1x$$

$$3. h_{\theta}(x) = -150 + 0.4x$$

Matrix

$$\begin{bmatrix} 1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 1 & 852 \end{bmatrix}$$

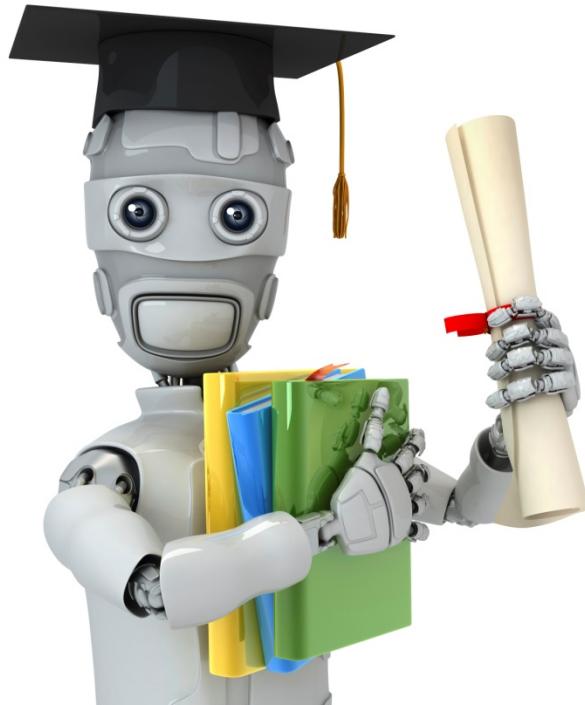
Matrix

$$\begin{bmatrix} -40 \\ 200 \\ -150 \\ 0.25 \\ 0.1 \\ 0.4 \end{bmatrix}$$

$$\begin{bmatrix} 486 \\ 314 \\ 344 \\ 173 \\ 410 \\ 342 \\ 353 \\ 285 \\ 692 \\ 416 \\ 464 \\ 191 \end{bmatrix}$$

Prediction  
of 1<sup>st</sup>  
 $h_{\theta}$

Predictions  
of 2<sup>nd</sup>  
 $h_{\theta}$



Machine Learning

# Linear Algebra review (optional)

---

## Matrix multiplication properties

$$\begin{matrix} 3 \times 5 \\ \text{---} \\ 5 \times 3 \end{matrix}$$

"Commutative"

Let  $A$  and  $B$  be matrices. Then in general,

$A \times B \neq B \times A$ . (not commutative.)

E.g.

$\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 0 \end{bmatrix}$ <p style="text-align: center;"><del><math>\neq</math></del></p>	$\begin{array}{c} A \times B \\ m \times n \quad n \times m \end{array}$ $\begin{array}{c} A \times B \quad \text{is} \quad m \times m \\ \hline B \times A \quad \text{is} \quad n \times n \end{array}$
--	--

$$\begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 2 & 2 \end{bmatrix}$$



$$\underline{3 \times 5 \times 2} \quad 3 \times (5+2) = (3+5) \times 2$$

$3 \times 10 = 30 = 15 \times 2$

"Associative"

$$A \times B \times C.$$

Let  $D = B \times C$ . Compute  $A \times D$ .

Let  $E = A \times B$ . Compute  $E \times C$ .

$$A \times (B \times C) \leftarrow$$

$(A \times B) \times C$  ←

$A \times (B \times C)$   
 $(A \times B) \times C$

Some answer.

1 is identity

## Identity Matrix

Denoted  $I$  (or  $I_{n \times n}$ ).

Examples of identity matrices:

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \underline{2 \times 2}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \underline{3 \times 3}$$

~~$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \underline{4 \times 4}$$~~

For any matrix  $A$ ,

$$A \cdot \boxed{I} = \boxed{I} \cdot A = A$$

$\uparrow \quad \uparrow \quad \uparrow \quad \uparrow$

$m \times n \quad n \times n \quad m \times m \quad m \times n \quad m \times n$

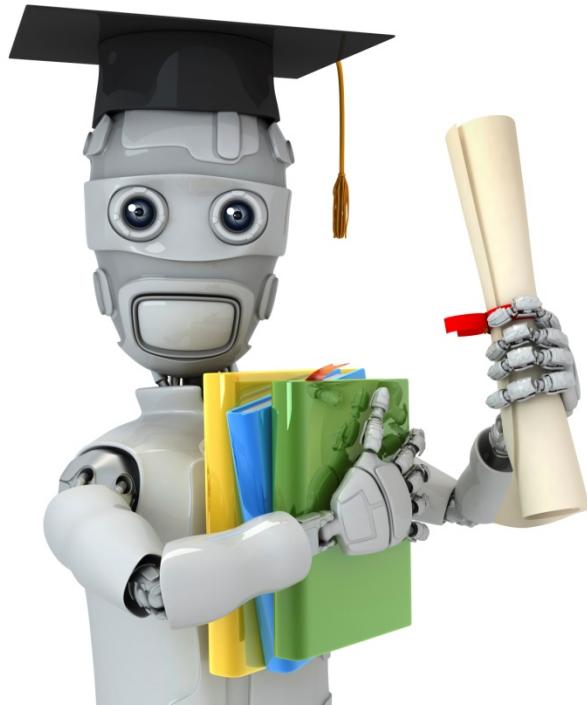
Note:  
 $AB \neq BA$  in general  
 $AI = IA$  ✓

$$\boxed{1 \times z = z \times 1 = z}$$

↑ for any  $z$

Informally:

$$\begin{bmatrix} 1 & 0 & \dots \\ 0 & 1 & \dots \\ \vdots & \vdots & \ddots \end{bmatrix} \quad \leftarrow$$



Machine Learning

# Linear Algebra review (optional)

---

## Inverse and transpose

$$1 = \text{"identity."}$$

$$3 \begin{matrix} (3^{-1}) \\ \frac{1}{3} \end{matrix} = 1$$

$$12 \begin{matrix} (12^{-1}) \\ \frac{1}{12} \end{matrix} = 1$$

$$0 \begin{matrix} (0^{-1}) \\ \underline{\hspace{2cm}} \end{matrix} \text{ undefined}$$

Not all numbers have an inverse.

**Matrix inverse:** If A is an  $m \times m$  matrix, and if it has an inverse,

$$\rightarrow A(A^{-1}) = A^{-1}A = I.$$

E.g.

$$A = \begin{bmatrix} 3 & 4 \\ 2 & 16 \end{bmatrix}$$

$$A^{-1} = \begin{bmatrix} 0.4 & -0.1 \\ -0.05 & 0.075 \end{bmatrix}$$

$$A^{-1}A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = I_{2 \times 2}$$

Matrices that don't have an inverse are “singular” or “degenerate”

## Matrix Transpose

Example:

$$\underline{A} = \begin{bmatrix} 1 & 2 & 0 \\ 3 & 5 & 9 \end{bmatrix}_{2 \times 3}$$

$$\underline{B} = \underline{A}^T = \begin{bmatrix} 1 & 3 \\ 2 & 5 \\ 0 & 9 \end{bmatrix}_{3 \times 2}$$

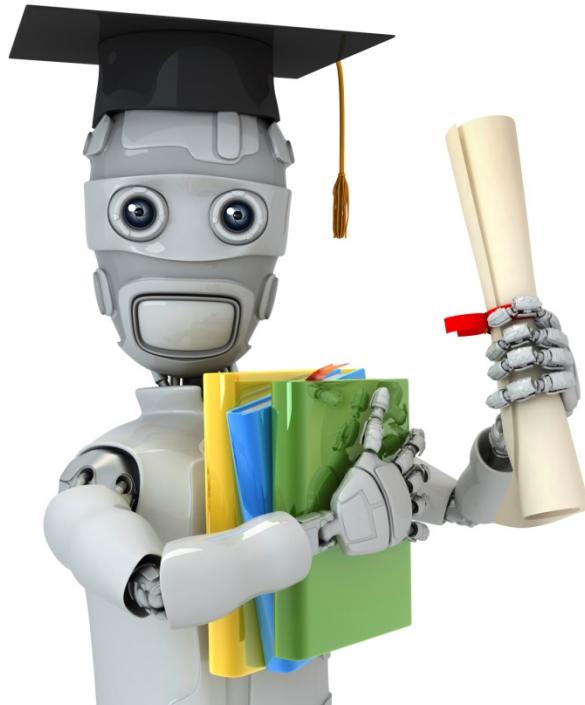
Let  $A$  be an  $m \times n$  matrix, and let  $B = A^T$ .

Then  $B$  is an  $n \times m$  matrix, and

$$\underline{B}_{ij} = \underline{A}_{ji}.$$

$$B_{12} = A_{21} = 2$$

$$B_{32} = 9 \quad A_{23} = 9.$$



Machine Learning

# Linear Regression with multiple variables

---

## Multiple features

## Multiple features (variables).

Size (feet <sup>2</sup> )	Price (\$1000)
$\rightarrow x$	$y \leftarrow$
2104	460
1416	232
1534	315
852	178
...	...

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$



# Multiple features (variables).

<u>Size (feet<sup>2</sup>)</u>	<u>Number of bedrooms</u>	<u>Number of floors</u>	<u>Age of home (years)</u>	Price (\$1000)
$x_1$	$x_2$	$x_3$	$x_4$	$y$
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

Notation:

- $n = 4$  = number of features
- $x^{(i)}$  = input (features) of  $i^{th}$  training example.
- $x_j^{(i)}$  = value of feature  $j$  in  $\underline{i^{th}}$  training example.

$x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}$

$x_3^{(2)} = 2$

Hypothesis:

Previously:

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

E.g.  $\underline{h_{\theta}(x)} = \underline{80} + \underline{0.1x_1} + \underline{0.01x_2} + \underline{3x_3} - \underline{2x_4}$

$$\rightarrow h_{\theta}(x) = \underline{\theta_0} + \underline{\theta_1}x_1 + \underline{\theta_2}x_2 + \cdots + \underline{\theta_n}x_n$$

For convenience of notation, define  $x_0 = 1.$  ( $x_0^{(i)} = 1$ )

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$\Theta = \begin{bmatrix} \Theta_0 \\ \Theta_1 \\ \Theta_2 \\ \vdots \\ \Theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

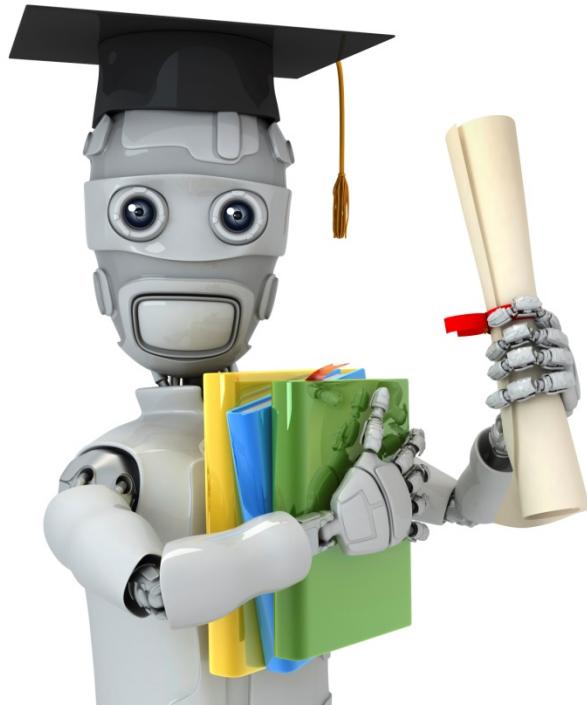
$$h_{\theta}(x) = \underline{\Theta_0x_0 + \Theta_1x_1 + \cdots + \Theta_nx_n}$$

$$= \boxed{\Theta^T x}$$

$$\Theta^T \underbrace{\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}}_x$$

$\Theta^T$   
 $(n+1) \times 1$   
matrix  
 $\Theta^T x$

Multivariate linear regression. 



Machine Learning

# Linear Regression with multiple variables

---

## Gradient descent for multiple variables

Hypothesis:  $\underline{h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n}$

Parameters:  $\underline{\theta_0, \theta_1, \dots, \theta_n}$   $\Theta$  n+1 - dimensional vector

Cost function:

$$\underline{J(\theta_0, \theta_1, \dots, \theta_n)} = \underline{\mathcal{J}(\Theta)} = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

Gradient descent:

Repeat {  
     $\rightarrow \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \dots, \theta_n)$   $\mathcal{J}(\Theta)$   
    }  
        ↑ simultaneously update for every  $j = 0, \dots, n$

# Gradient Descent

Previously (n=1):

Repeat {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})$$

$$\frac{\partial}{\partial \theta_0} J(\theta)$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x^{(i)}$$

(simultaneously update  $\theta_0, \theta_1$ )

}

New algorithm ( $n \geq 1$ ):

Repeat {

$$\frac{\partial}{\partial \theta_j} J(\theta)$$

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update  $\theta_j$  for  
 $j = 0, \dots, n$ )

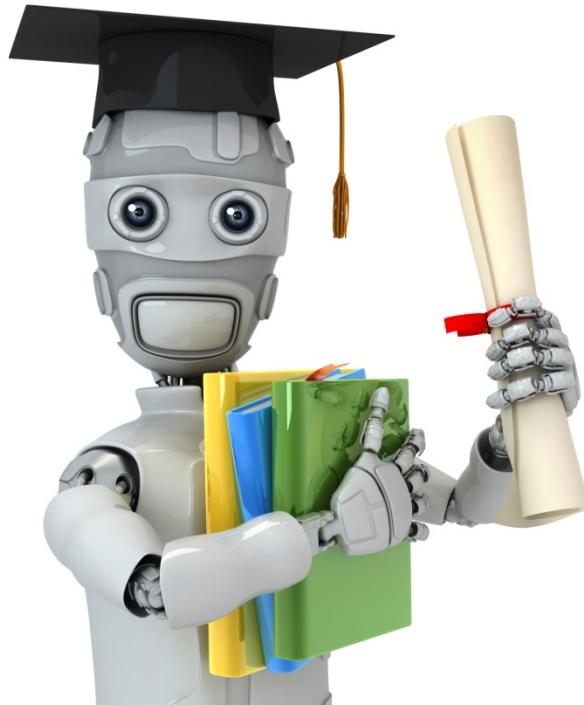
}

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

...



Machine Learning

# Linear Regression with multiple variables

---

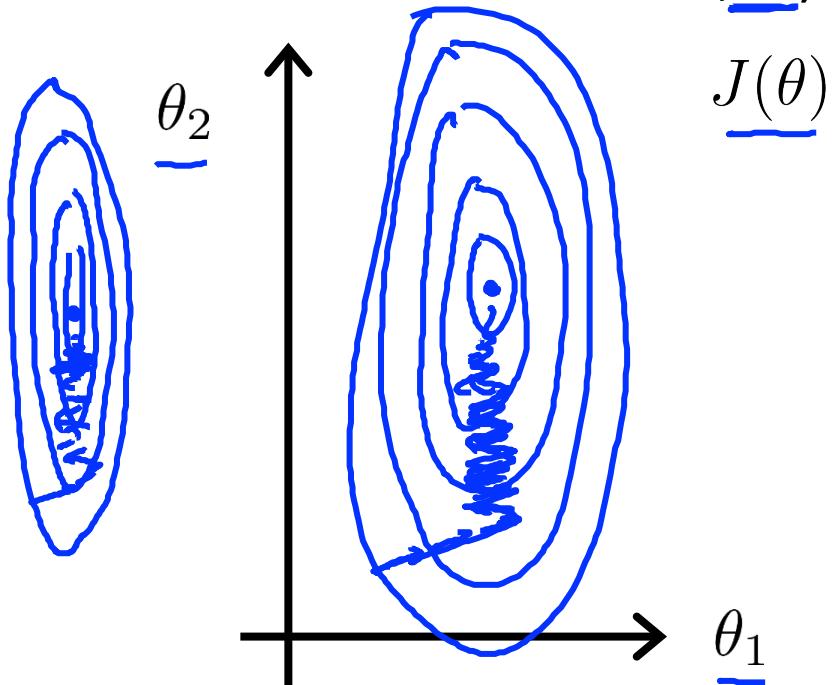
## Gradient descent in practice I: Feature Scaling

# Feature Scaling

Idea: Make sure features are on a similar scale.

E.g.  $x_1 = \text{size } (0\text{-}2000 \text{ feet}^2)$

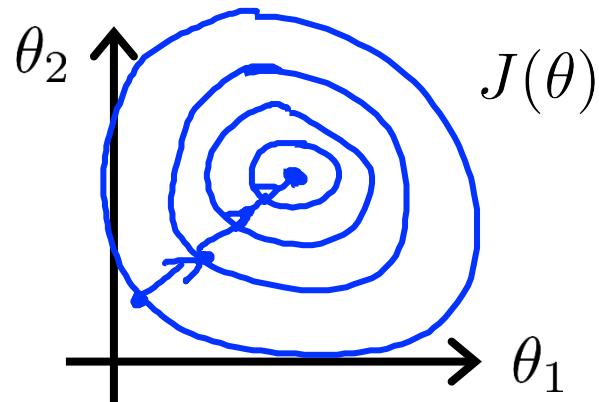
$x_2 = \text{number of bedrooms } (1\text{-}5)$



$$\rightarrow x_1 = \frac{\text{size (feet}^2)}{2000} \quad \swarrow$$

$$\rightarrow x_2 = \frac{\text{number of bedrooms}}{5} \quad \swarrow$$

$$0 \leq x_1 \leq 1 \quad 0 \leq x_2 \leq 1$$



## Feature Scaling

Get every feature into approximately a  $-1 \leq x_i \leq 1$  range.

$$x_0 = 1$$

$$6 \leq x_1 \leq 3 \quad \checkmark$$

$$-2 \leq x_2 \leq 0.5 \quad \checkmark$$

$$-100 \leq x_3 \leq 100 \quad \times$$

$$-0.0001 \leq x_4 \leq 0.0001 \quad \times$$

$$\boxed{-1 \leq x_i \leq 1}$$

$$-3 \text{ to } 3 \quad \checkmark$$

$$-\frac{1}{2} \text{ to } \frac{1}{2} \quad \checkmark$$

## Mean normalization

Replace  $x_i$  with  $\frac{x_i - \mu_i}{\sigma_i}$  to make features have approximately zero mean  
(Do not apply to  $x_0 = 1$ ).

E.g.  $x_1 = \frac{\text{size} - 1000}{2000}$

Average size  $\approx 100$

$$x_2 = \frac{\#\text{bedrooms} - 2}{5 - 4}$$

1-5 bedrooms

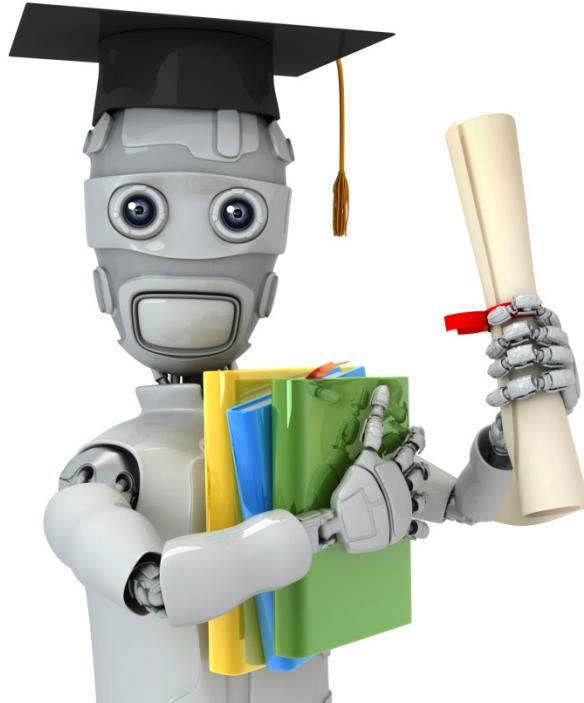
$$\rightarrow [-0.5 \leq x_1 \leq 0.5, -0.5 \leq x_2 \leq 0.5]$$

$$x_1 \leftarrow \frac{x_1 - \mu_1}{\sigma_1}$$

avg value of  $x_1$  in training set

range ( $\max - \min$ )  
(or standard deviation)

$$x_2 \leftarrow \frac{x_2 - \mu_2}{\sigma_2}$$



Machine Learning

# Linear Regression with multiple variables

---

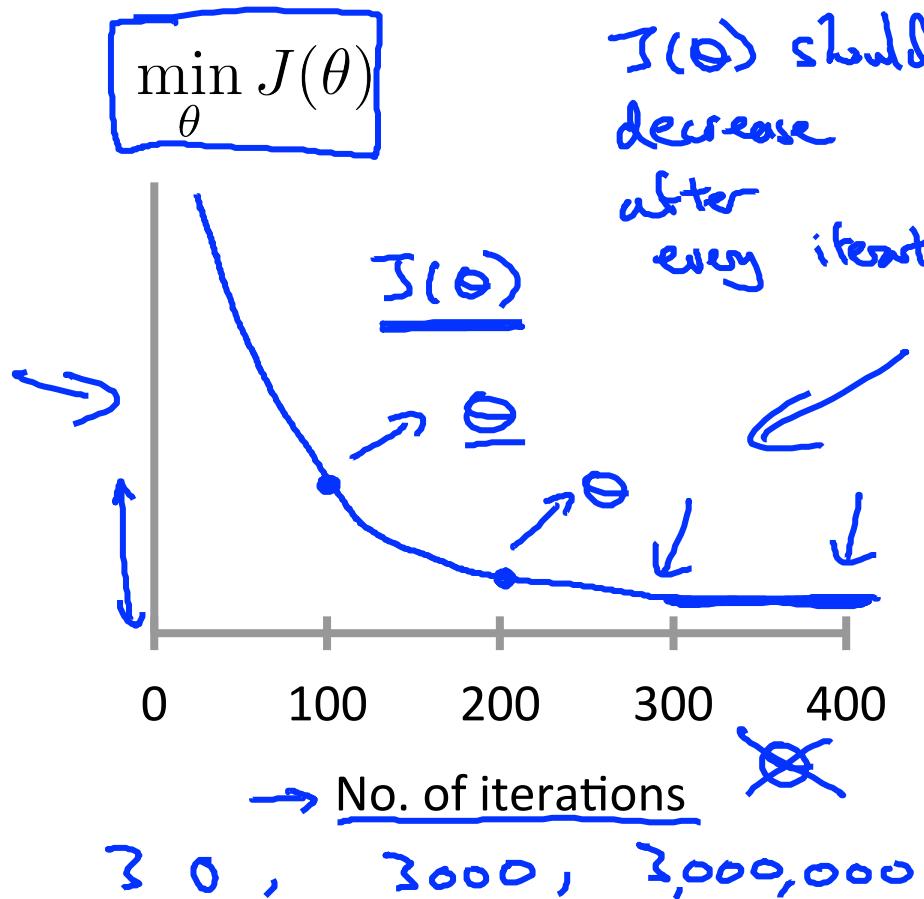
## Gradient descent in practice II: Learning rate

# Gradient descent

$$\Rightarrow \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- “Debugging”: How to make sure gradient descent is working correctly.
- How to choose learning rate  $\alpha$ .

# Making sure gradient descent is working correctly.



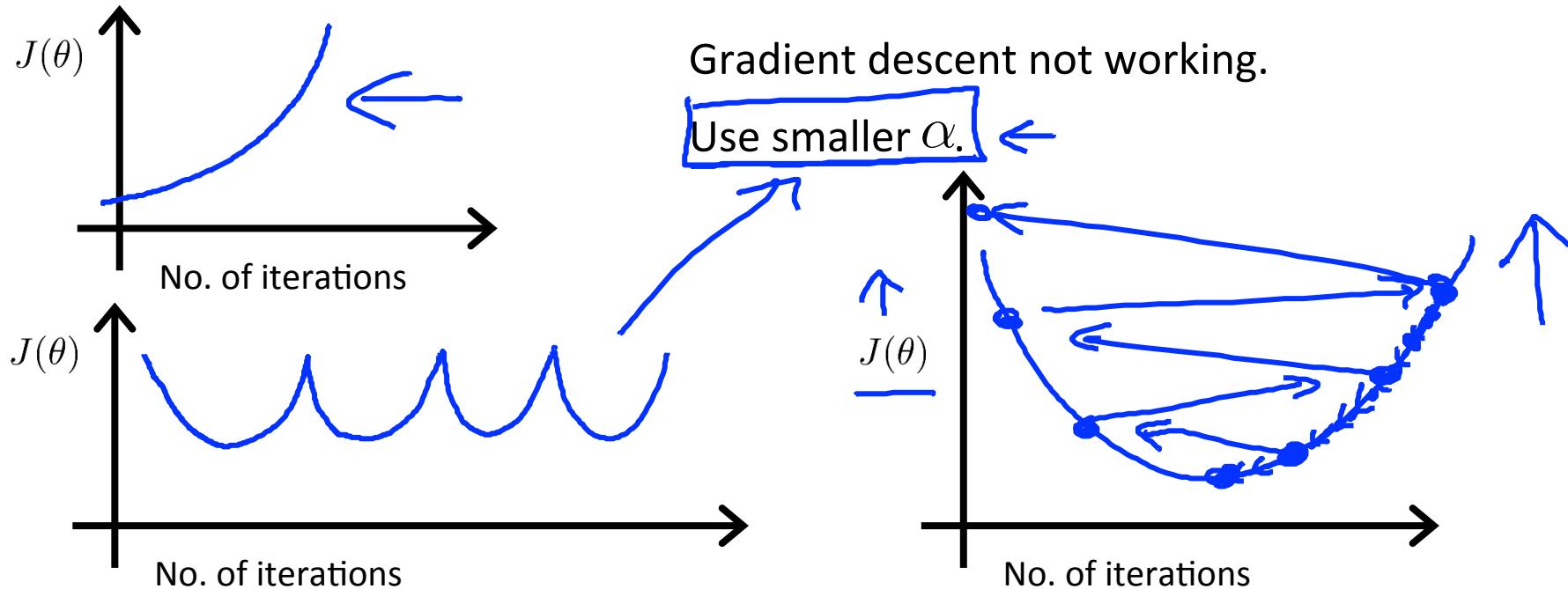
$J(\theta)$  should decrease after every iteration.

→ Example automatic convergence test:

→ Declare convergence if  $J(\theta)$  decreases by less than  $10^{-3}$  in one iteration.

$$10^{-3}$$

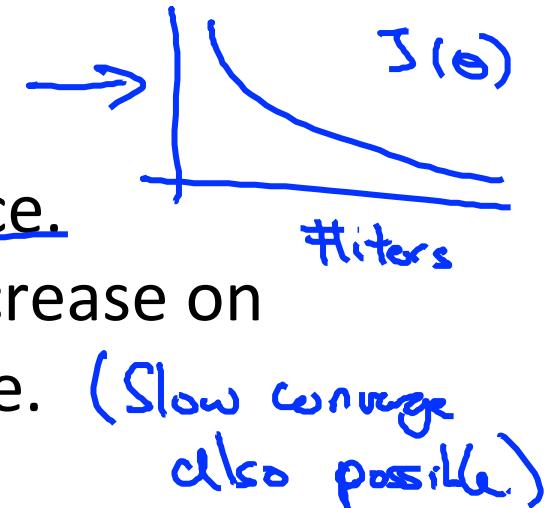
# Making sure gradient descent is working correctly.



- For sufficiently small  $\alpha$ ,  $J(\theta)$  should decrease on every iteration.
- But if  $\alpha$  is too small, gradient descent can be slow to converge.

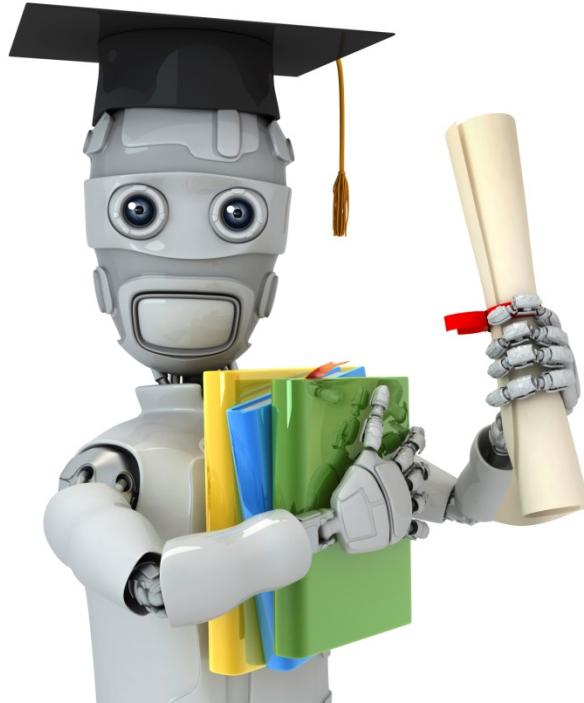
## Summary:

- If  $\alpha$  is too small: slow convergence.
- If  $\alpha$  is too large:  $J(\theta)$  may not decrease on every iteration; may not converge. (Slow converge also possible)



To choose  $\alpha$ , try

$$\dots, \underbrace{0.001}_{\uparrow}, \underbrace{0.003}_{\approx 3x}, \underbrace{0.01}_{\approx 3x}, \underbrace{0.03}_{3x}, \underbrace{0.1}_{\approx 3x}, \underbrace{0.3}_{3x}, \underbrace{1}_{\approx 3x}, \dots$$



Machine Learning

# Linear Regression with multiple variables

---

Features and  
polynomial regression

# Housing prices prediction

$$h_{\theta}(x) = \theta_0 + \theta_1 \times \boxed{\text{frontage}} + \theta_2 \times \boxed{\text{depth}}$$

$x_1$   
-



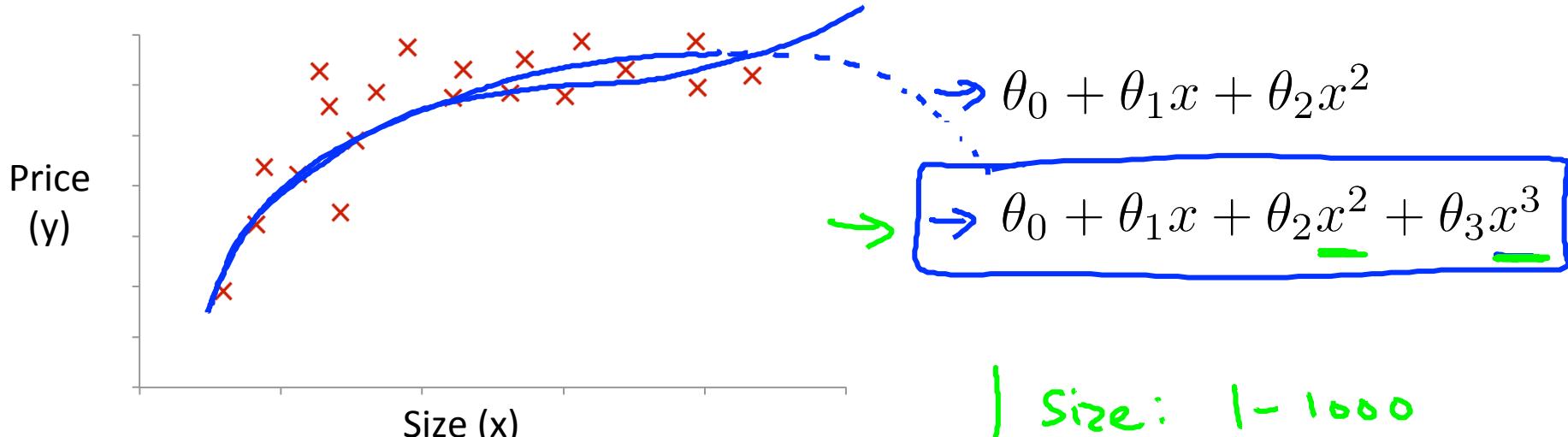
Area

$$\times = \underline{\text{frontage} \times \text{depth}}$$

$$h_{\theta}(x) = \theta_0 + \theta_1 x$$

~ land area

# Polynomial regression



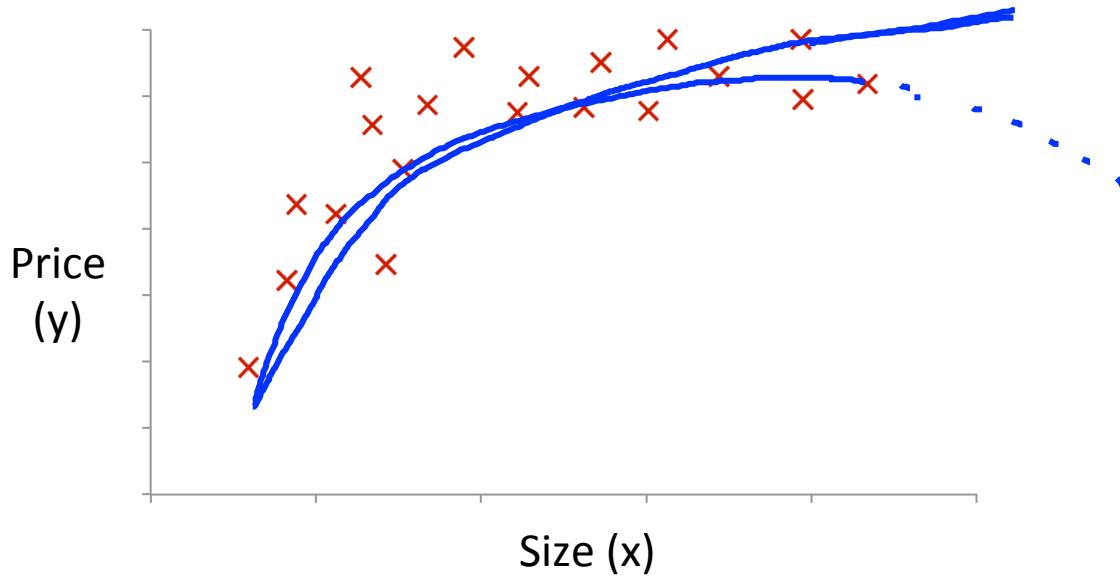
$$h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

$$= \theta_0 + \theta_1(\text{size}) + \theta_2(\text{size})^2 + \theta_3(\text{size})^3$$

$$\begin{aligned} \rightarrow x_1 &= (\text{size}) \\ \rightarrow x_2 &= (\text{size})^2 \\ \rightarrow x_3 &= (\text{size})^3 \end{aligned}$$

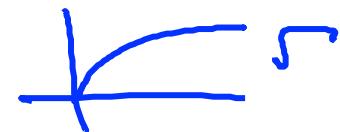
Size: 1 - 1000  
Size<sup>2</sup>: 1 - 1000, 000  
Size<sup>3</sup>: 1 - 10<sup>9</sup>

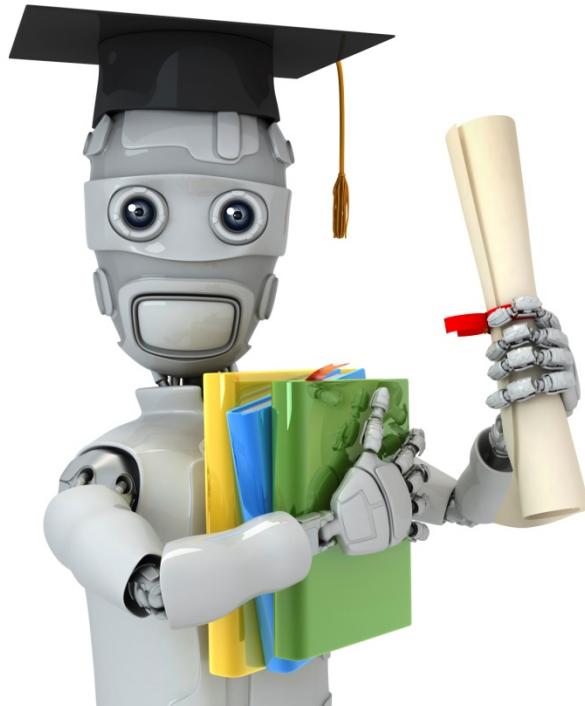
# Choice of features



$$h_{\theta}(x) = \theta_0 + \theta_1(\text{size}) + \theta_2(\text{size})^2$$

$$h_{\theta}(x) = \theta_0 + \theta_1(\text{size}) + \theta_2 \sqrt{(\text{size})}$$





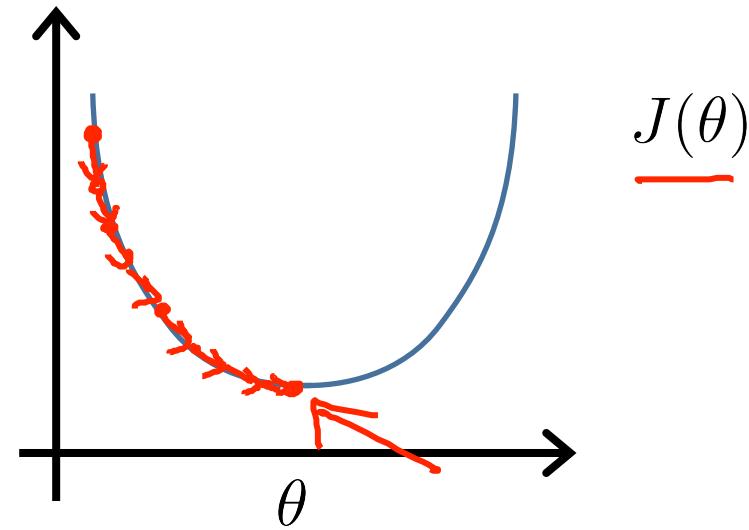
Machine Learning

# Linear Regression with multiple variables

---

## Normal equation

## Gradient Descent



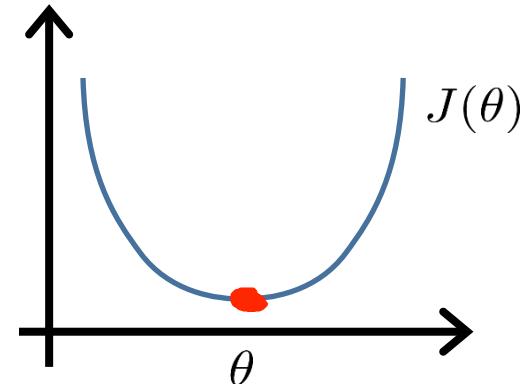
Normal equation: Method to solve for  $\underline{\theta}$  analytically.

Intuition: If 1D ( $\theta \in \mathbb{R}$ )

$$\rightarrow J(\theta) = a\theta^2 + b\theta + c$$

$$\frac{\partial}{\partial \theta} J(\theta) = \dots \stackrel{\text{set}}{=} 0$$

Solve for  $\theta$



$$\underline{\theta \in \mathbb{R}^{n+1}}$$

$$\underline{J(\theta_0, \theta_1, \dots, \theta_m)} = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

$$\underline{\frac{\partial}{\partial \theta_j} J(\theta) = \dots = 0} \quad (\text{for every } j)$$

Solve for  $\underline{\theta_0, \theta_1, \dots, \theta_n}$

Examples:  $m = 4$ .

	Size (feet <sup>2</sup> )	Number of bedrooms	Number of floors	Age of home (years)	Price (\$1000)
$x_0$	$x_1$	$x_2$	$x_3$	$x_4$	$y$
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

Diagram illustrating the data matrix  $X$  and the price vector  $y$ :

$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}$

$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$

$\theta = (X^T X)^{-1} X^T y$

$m \times (n+1)$

$m$ -dimensional vector

$m$  examples  $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})$ ;  $n$  features.

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1} \quad X = \begin{bmatrix} \cdots & (x^{(1)})^\top & \cdots \\ \cdots & (x^{(1)})^\top & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & (x^{(m)})^\top & \cdots \end{bmatrix}$$

(design matrix)

E.g. If  $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix}$

$$X = \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_1^{(2)} \\ \vdots & \vdots \\ 1 & x_1^{(m)} \end{bmatrix}_{m \times 2}$$

$$y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

$$\Theta = (X^T X)^{-1} X^T y$$

$$\theta = \boxed{(X^T X)^{-1} X^T y}$$

$(X^T X)^{-1}$  is inverse of matrix  $X^T X$ .

Set  $A := X^T X$

$$(X^T X)^{-1} = A^{-1}$$

Octave:  $\text{pinv}(X' * X) * X' * y$

$$\text{pinv}(X^T * X) * X^T * y$$

$$\theta = \boxed{(X^T X)^{-1} X^T y}$$

$$\min_{\theta} J(\theta)$$

$$\left| \begin{array}{l} X' \\ X^T \\ \hline \cancel{\text{Feature Scaling}} \\ 0 \leq x_1 \leq 1 \\ 0 \leq x_2 \leq 1000 \\ 0 \leq x_3 \leq 10^{-5} \end{array} \right| \checkmark$$

$m$  training examples,  $n$  features.

### Gradient Descent

- • Need to choose  $\alpha$ .
- • Needs many iterations.
- Works well even when  $n$  is large.

$$\underline{n = 10^6}$$

### Normal Equation

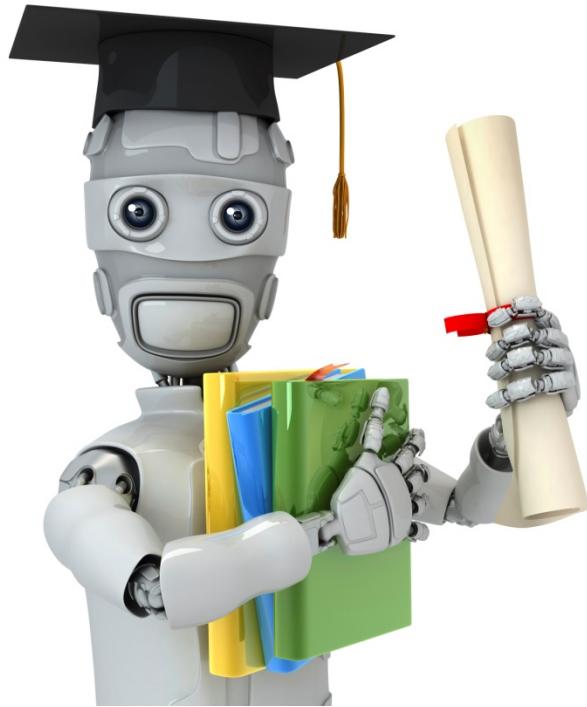
- • No need to choose  $\alpha$ .
- • Don't need to iterate.
- Need to compute  
$$(X^T X)^{-1}$$
  $n \times n$   $O(n^3)$
- Slow if  $n$  is very large.

$$n = 100$$

$$n = 1000$$

$$\underline{n = 10000}$$





Machine Learning

# Linear Regression with multiple variables

---

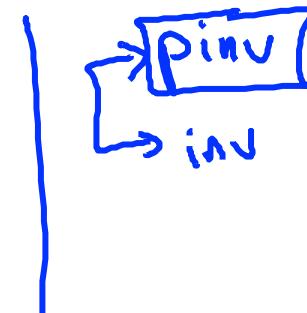
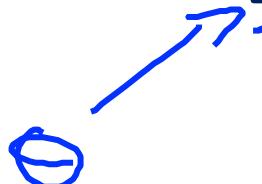
Normal equation  
and non-invertibility  
(optional)

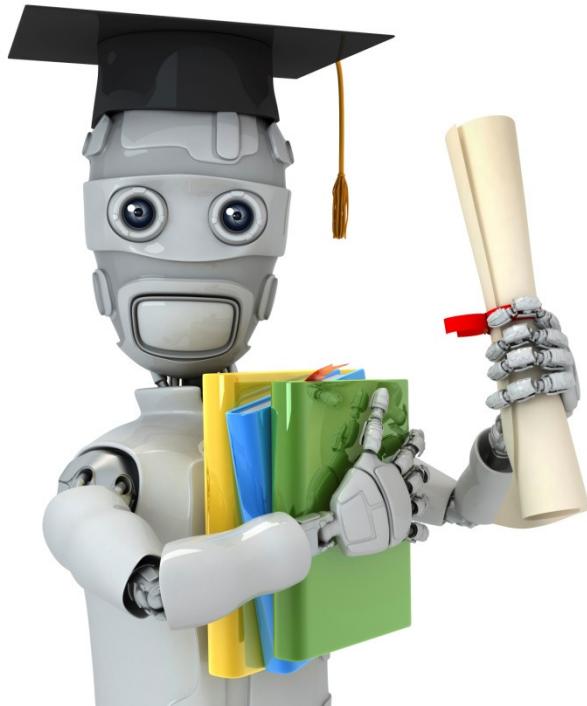
## Normal equation

$$\theta = \underline{(X^T X)^{-1} X^T y}$$

$X^T X$

- What if  $X^T X$  is non-invertible? (singular/degenerate)
- Octave: `pinv(X' * X) * X' * y`





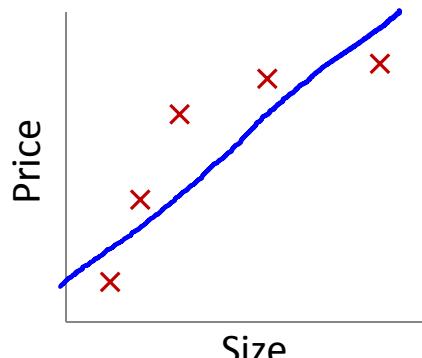
Machine Learning

# Regularization

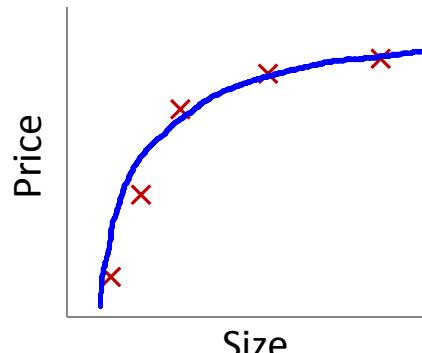
---

## The problem of overfitting

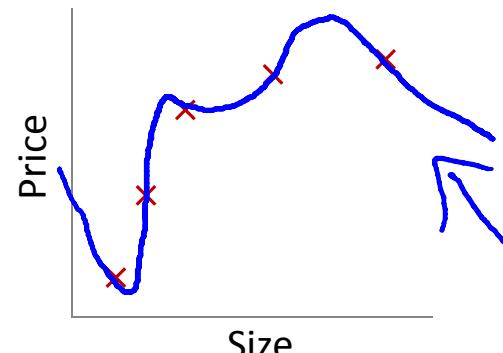
## Example: Linear regression (housing prices)



$\rightarrow \theta_0 + \theta_1 x$   
"Underfit" "High bias"



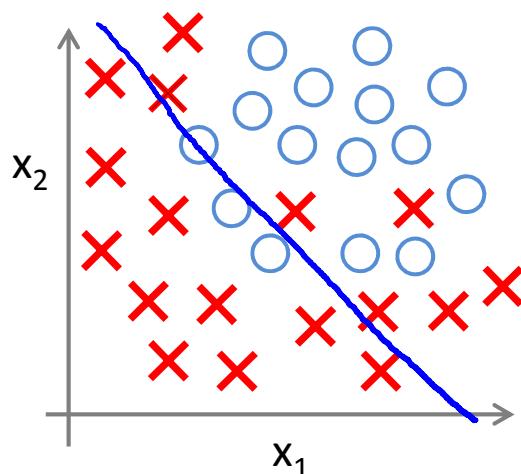
$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2$   
"Just right"



$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$   
"Overfit" "High variance"

**Overfitting:** If we have too many features, the learned hypothesis may fit the training set very well ( $J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$ ), but fail to generalize to new examples (predict prices on new examples).

## Example: Logistic regression

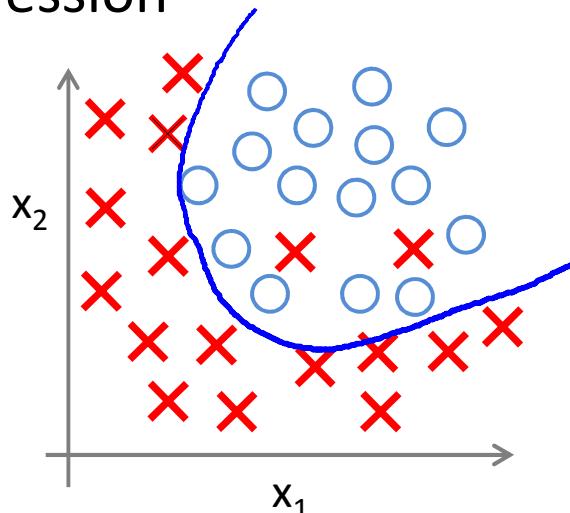


$$\rightarrow h_{\theta}(x) = g(\underline{\theta_0 + \theta_1 x_1 + \theta_2 x_2})$$

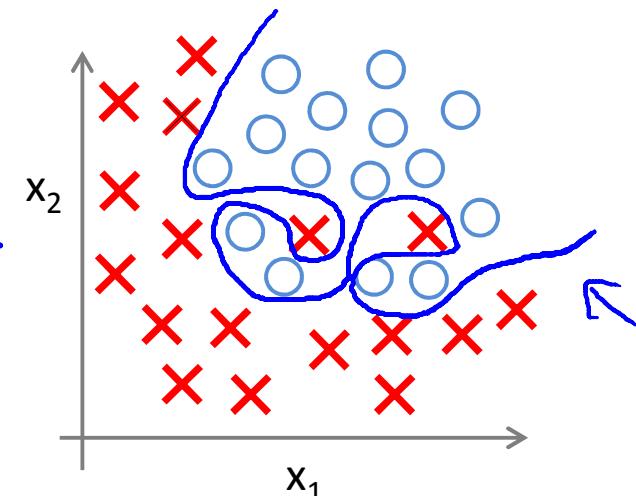
( $g$  = sigmoid function)



"Underfit"



$$g(\underline{\theta_0 + \theta_1 x_1 + \theta_2 x_2} \\ + \underline{\theta_3 x_1^2} + \underline{\theta_4 x_2^2} \\ + \underline{\theta_5 x_1 x_2})$$



$$g(\underline{\theta_0 + \theta_1 x_1 + \theta_2 x_1^2} \\ + \underline{\theta_3 x_1^2 x_2} + \underline{\theta_4 x_1^2 x_2^2} \\ + \underline{\theta_5 x_1^2 x_2^3} + \underline{\theta_6 x_1^3 x_2} + \dots)$$

"Overfit"

## Addressing overfitting:

$x_1$  = size of house

$x_2$  = no. of bedrooms

$x_3$  = no. of floors

$x_4$  = age of house

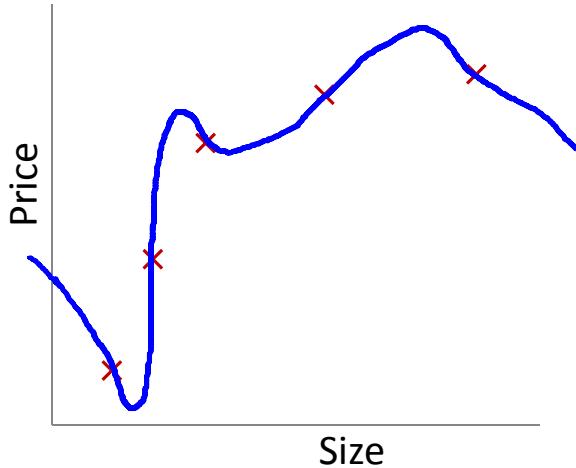
$x_5$  = average income in neighborhood

$x_6$  = kitchen size

:

:

$x_{100}$

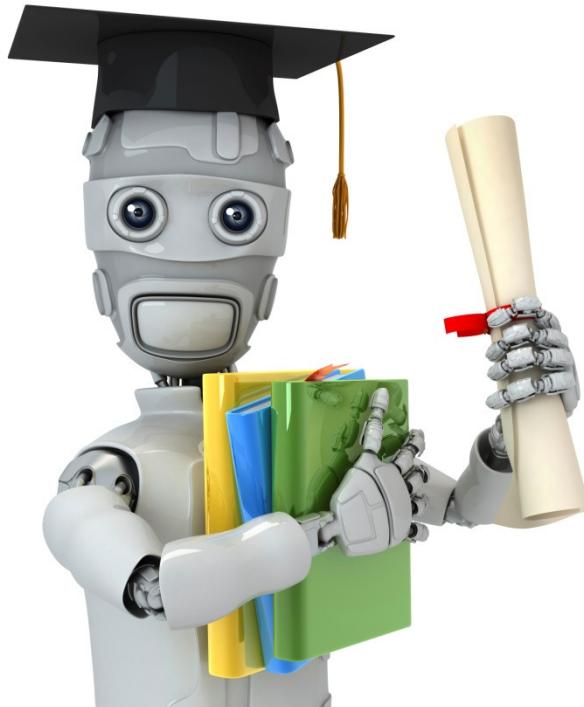


## Addressing overfitting:

Options:

1. Reduce number of features.
  - — Manually select which features to keep.
  - — Model selection algorithm (later in course).
2. Regularization.
  - — Keep all the features, but reduce magnitude/values of parameters  $\theta_j$
  - Works well when we have a lot of features, each of which contributes a bit to predicting  $y$ .





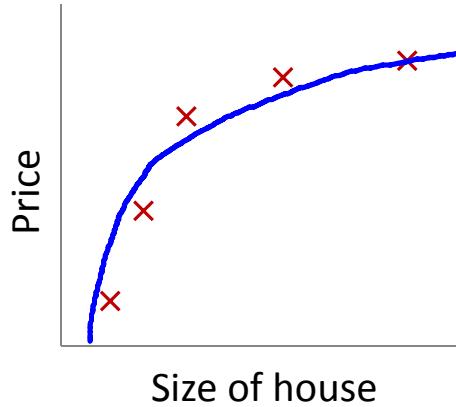
Machine Learning

# Regularization

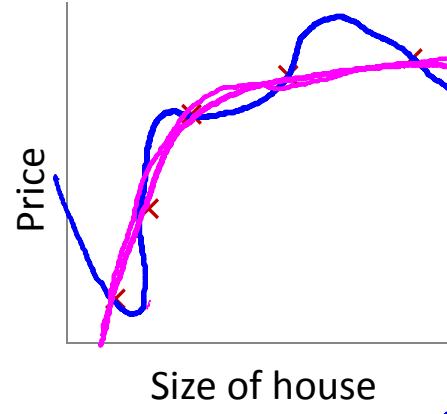
---

## Cost function

# Intuition



$$\theta_0 + \theta_1 x + \theta_2 x^2$$



$$\underline{\theta_0 + \theta_1 x + \theta_2 x^2} + \cancel{\theta_3 x^3} + \cancel{\theta_4 x^4}$$

Suppose we penalize and make  $\theta_3, \theta_4$  really small.

$$\rightarrow \min_{\theta} \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + 1000 \frac{\theta_3^2}{\underline{\theta_3}} + 1000 \frac{\theta_4^2}{\underline{\theta_4}}$$

$$\underline{\theta_3 \approx 0}$$

$$\underline{\theta_4 \approx 0}$$

# Regularization.

Small values for parameters

$$\theta_0, \theta_1, \dots, \theta_n$$

- “Simpler” hypothesis
- Less prone to overfitting

$$\theta_3, \theta_4$$

$\uparrow \approx 0$

Housing:

- Features:  $x_1, x_2, \dots, x_{100}$
- Parameters:  $\theta_0, \theta_1, \theta_2, \dots, \theta_{100}$

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

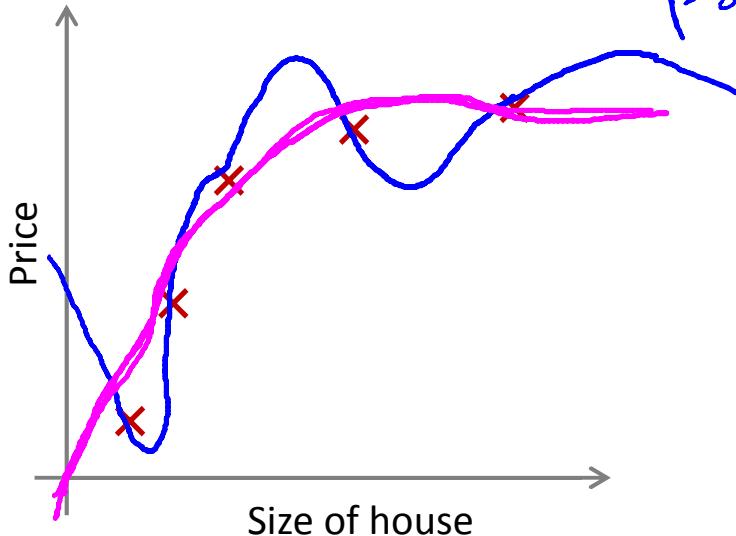
~~$\theta_1, \theta_2, \theta_3, \dots, \theta_{100}$~~

## Regularization.

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

$\min_{\theta} J(\theta)$

regularization parameter



In regularized linear regression, we choose  $\theta$  to minimize

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

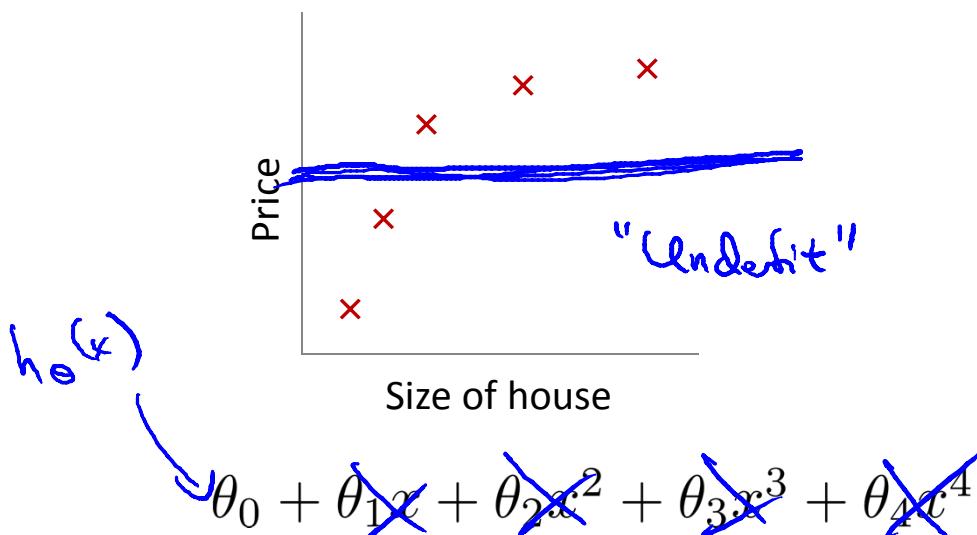
What if  $\lambda$  is set to an extremely large value (perhaps for too large for our problem, say  $\lambda = 10^{10}$ )?

- Algorithm works fine; setting  $\lambda$  to be very large can't hurt it
- Algorithm fails to eliminate overfitting.
- Algorithm results in underfitting. (Fails to fit even training data well).
- Gradient descent will fail to converge.

In regularized linear regression, we choose  $\theta$  to minimize

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

What if  $\lambda$  is set to an extremely large value (perhaps for too large for our problem, say  $\lambda = 10^{10}$ )?



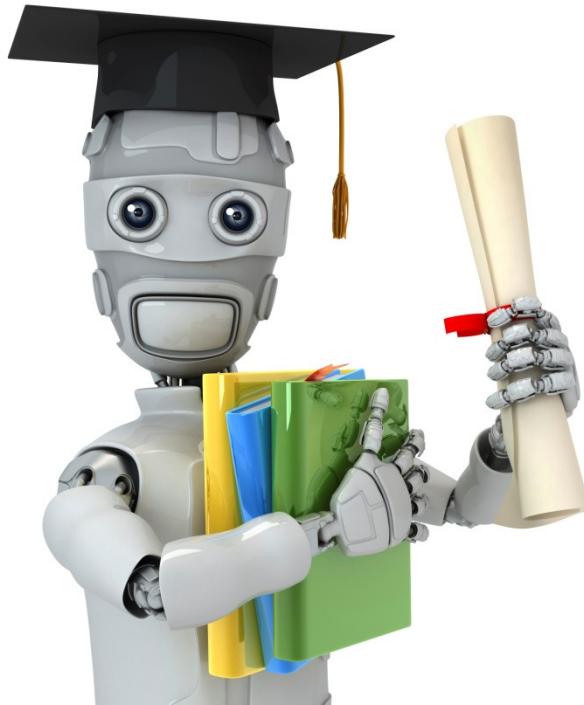
$$\underline{\theta}_1, \underline{\theta}_2, \underline{\theta}_3, \underline{\theta}_4$$

$$\theta_0 \approx 0, \theta_2 \approx 0$$

$$\theta_3 \approx 0, \theta_4 \approx 0$$

$$h_\theta(x) = \theta_0$$





Machine Learning

# Regularization

---

## Regularized linear regression

# Regularized linear regression

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

$$\min_{\theta} J(\theta)$$

## Gradient descent

Repeat {

$$\rightarrow \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\frac{\partial}{\partial \theta_0} J(\theta)$$

$$\begin{aligned} \theta_j &:= \theta_j - \boxed{\alpha} \left[ \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right] \\ &\quad (j = \cancel{0}, 1, 2, 3, \dots, n) \end{aligned}$$

}

$$\theta_j := \boxed{\theta_j (1 - \alpha \frac{\lambda}{m})} - \boxed{\alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}} \rightarrow J(\theta)$$

$$1 - \alpha \frac{\lambda}{m} < 1$$

0.99       $\theta_j \times 0.99$

$$\boxed{\theta_j^2}$$

# Normal equation

$$\underline{X} = \begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix} \leftarrow \text{m} \times (n+1)$$

$$y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \uparrow \mathbb{R}^m$$

$$\rightarrow \min_{\theta} J(\theta) \quad \underline{\text{---}}$$

$$\Rightarrow \underline{\theta} = (X^T X + \lambda \underbrace{\begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 1 & 0 & 1 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}}_{(n+1) \times (n+1)})^{-1} X^T y$$

E.g. n=2  $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$   $(n+1) \times (n+1)$

## Non-invertibility (optional/advanced).

Suppose  $m \leq n$ ,  $\leftarrow$

(#examples) (#features)

$$\theta = (X^T X)^{-1} X^T y$$

non-invertible / singular

pinv

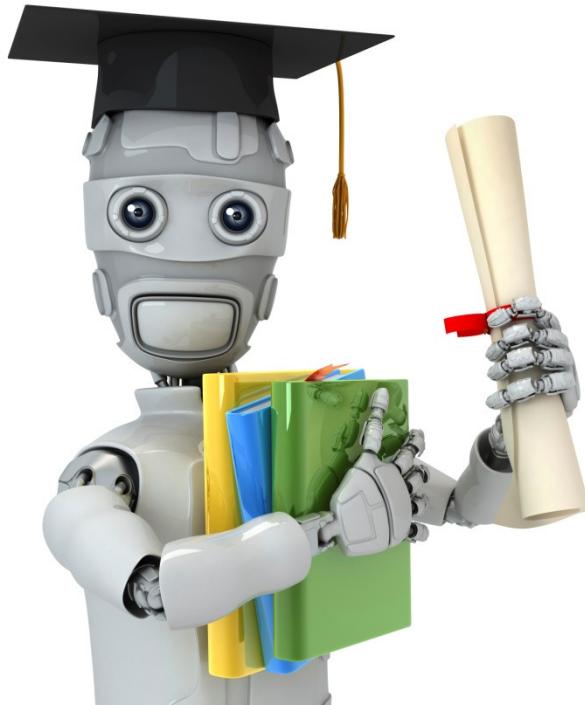
inv  
R

If  $\lambda > 0$ ,

$$\theta = \left( X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & 1 & \\ & & & \ddots \\ & & & & 1 \end{bmatrix} \right)^{-1} X^T y$$

invertible .





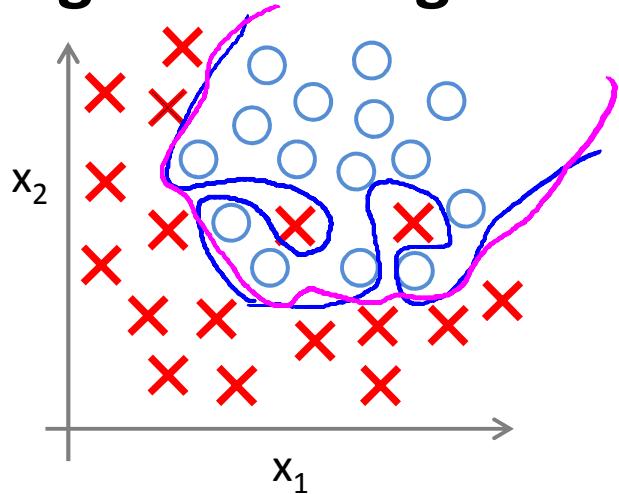
Machine Learning

# Regularization

---

## Regularized logistic regression

# Regularized logistic regression.



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^2 x_2 + \theta_4 x_1^2 x_2^2 + \theta_5 x_1^2 x_2^3 + \dots)$$

Cost function:

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

$\theta_1, \theta_2, \dots, \theta_n$

# Gradient descent

Repeat {

$$\rightarrow \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\rightarrow \theta_j := \theta_j - \alpha \left[ \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right] \leftarrow$$

$\underbrace{(j = \cancel{X}, 1, 2, 3, \dots, n)}_{\theta_1, \dots, \theta_n}$

}

$$\frac{\partial}{\partial \theta_j} J(\theta)$$

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^\top x}}$$

## Advanced optimization

f minunc (Q costFunction)  
 $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$  theta(1) ←  
theta(2) ←  
theta(n+1) ←

→ function [jVal, gradient] = costFunction(theta)

jVal = [ code to compute  $J(\theta)$  ] ;

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

→ gradient (1) = [ code to compute  $\frac{\partial}{\partial \theta_0} J(\theta)$  ] ;

$$\frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

→ gradient (2) = [ code to compute  $\frac{\partial}{\partial \theta_1} J(\theta)$  ] ;

$$\left( \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)} \right) - \frac{\lambda}{m} \theta_1$$

$J(\theta)$

→ gradient (3) = [ code to compute  $\frac{\partial}{\partial \theta_2} J(\theta)$  ] ;

$$\vdots \quad \left( \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)} \right) - \frac{\lambda}{m} \theta_2$$

gradient (n+1) = [ code to compute  $\frac{\partial}{\partial \theta_n} J(\theta)$  ] ;



What if  $X^T X$  is non-invertible?



- Redundant features (linearly dependent).

E.g.

$$\begin{aligned}x_1 &= \text{size in feet}^2 \\x_2 &= \text{size in m}^2 \\x_1 &= (3.28)^2 x_2\end{aligned}$$

$$1_m = 3.28 \text{ feet}$$

$$\rightarrow \underline{n = 10} \leftarrow$$

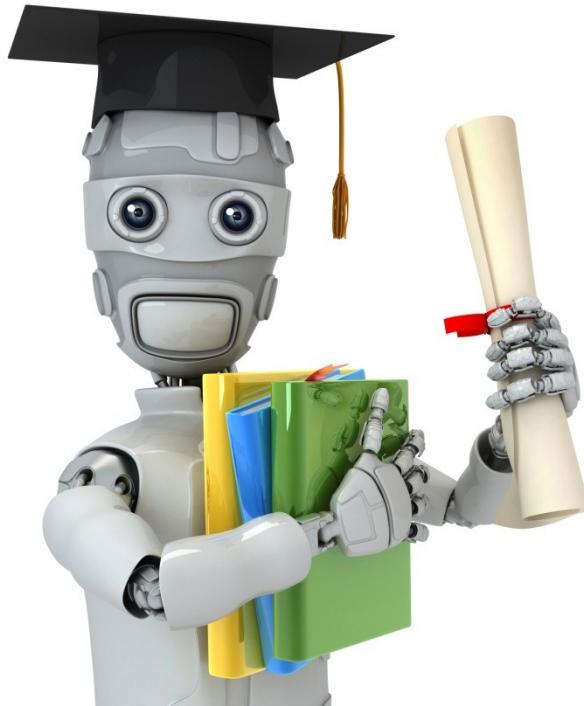
$$\rightarrow \underline{n = 100} \leftarrow$$

$$\Theta \in \mathbb{R}^{101}$$

- Too many features (e.g.  $m \leq n$ ).

- Delete some features, or use regularization.

↓ later



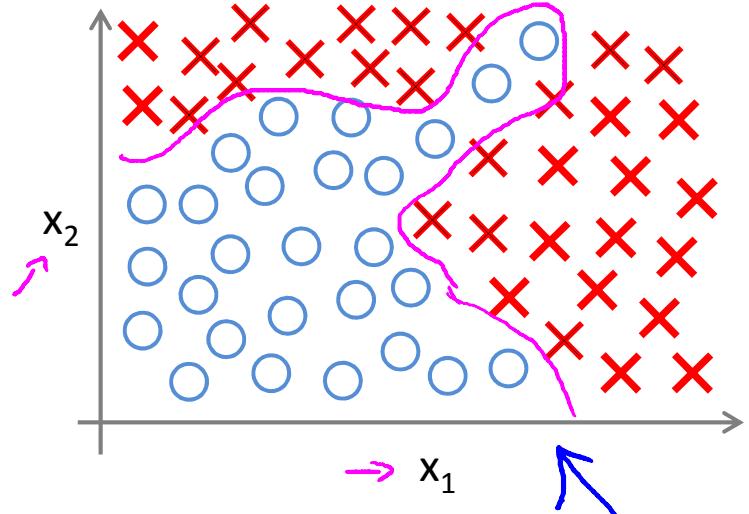
Machine Learning

# Neural Networks: Representation

---

## Non-linear hypotheses

# Non-linear Classification



$\rightarrow \underline{x_1}$  = size  
 $\underline{x_2}$  = # bedrooms  
 $\underline{x_3}$  = # floors  
 $x_4$  = age  
 $\dots$   
 $x_{100}$  -

$\left. \right\} h=100$

$$\begin{aligned}
 & \downarrow \\
 & g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 \\
 & + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 \\
 & + \theta_5 x_1^3 x_2 + \underline{\theta_6 x_1 x_2^2} + \dots)
 \end{aligned}$$

$$\begin{aligned}
 & \rightarrow \underline{x_1^2}, \underline{x_1 x_2}, \underline{x_1 x_3}, \underline{x_1 x_4} \dots \underline{x_1 x_{100}} \\
 & \underline{x_2^2}, \underline{x_2 x_3} \dots
 \end{aligned}$$

$\approx 5000$  feature

$$\begin{aligned}
 & \mathcal{O}(n^2) \\
 & \frac{n^2}{2} \\
 & \cancel{10}
 \end{aligned}$$

$$\rightarrow \underline{x_1^2}, \underline{x_2^2}, \underline{x_3^2}, \dots, \underline{x_{100}^2}$$

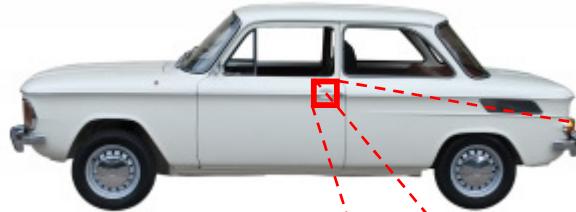
$$\rightarrow \underline{x_1 x_2 x_3}, \underline{x_1^2 x_2}, \underline{x_{10} x_{11} x_{12}}, \dots$$

$\mathcal{O}(n^3)$

170,000

# What is this?

You see this:



But the camera sees this:

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50



# Computer Vision: Car detection



Cars

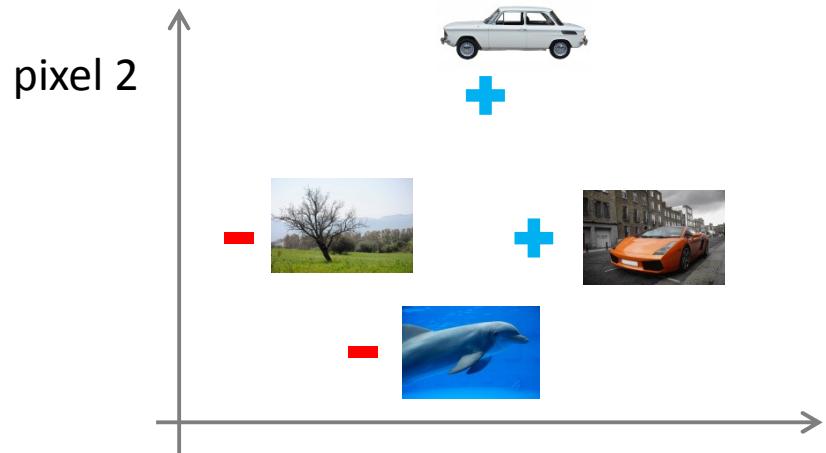
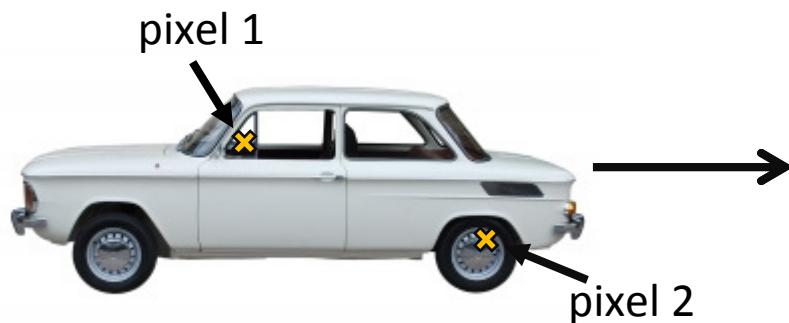


Not a car

Testing:

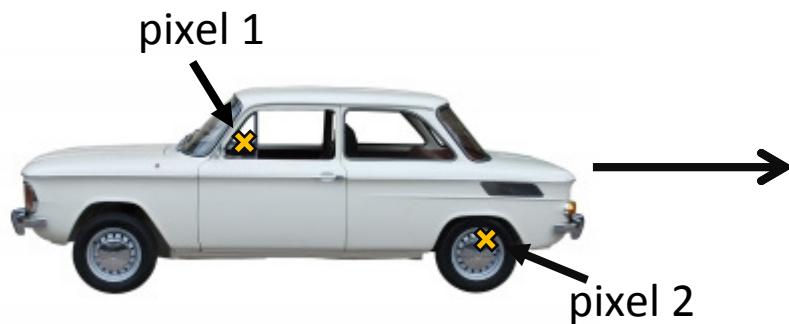


What is this?

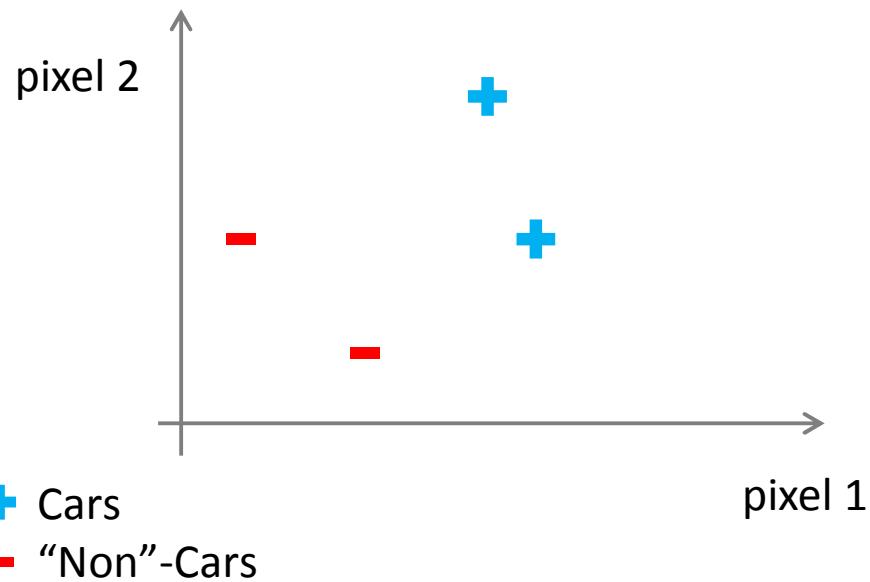


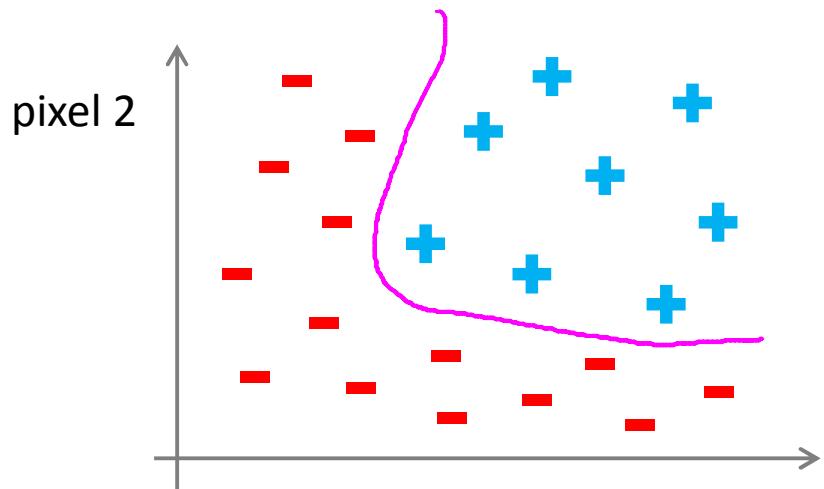
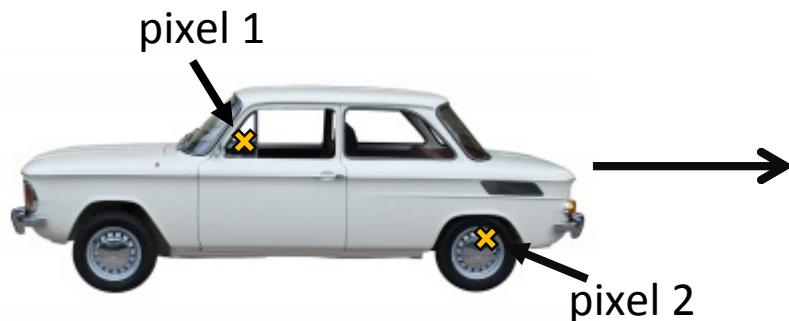
+ Cars

- "Non"-Cars



Learning  
Algorithm





$50 \times 50$  pixel images  $\rightarrow$  2500 pixels  
 $n = 2500$  (7500 if RGB)

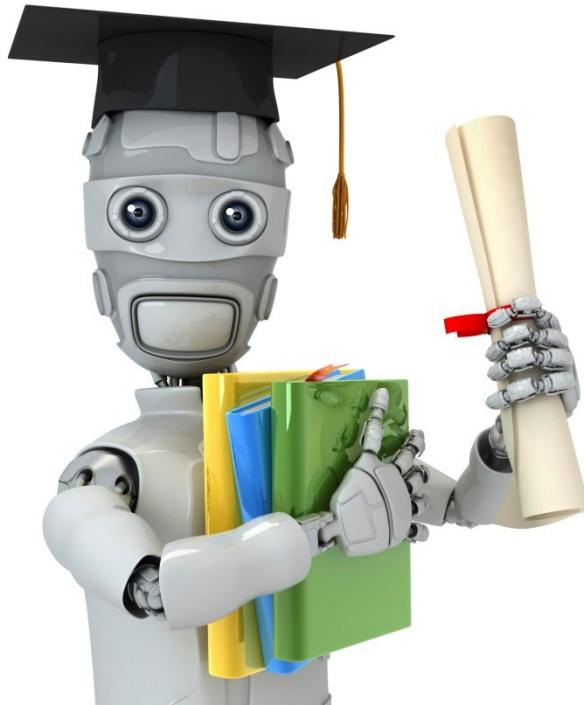
$$x = \begin{bmatrix} \text{pixel 1 intensity} \\ \text{pixel 2 intensity} \\ \vdots \\ \text{pixel 2500 intensity} \end{bmatrix}$$

$0 - 255$

+ Cars  
- "Non"-Cars

Quadratic features ( $x_i \times x_j$ ):  $\approx 3$  million features





Machine Learning

# Neural Networks: Representation

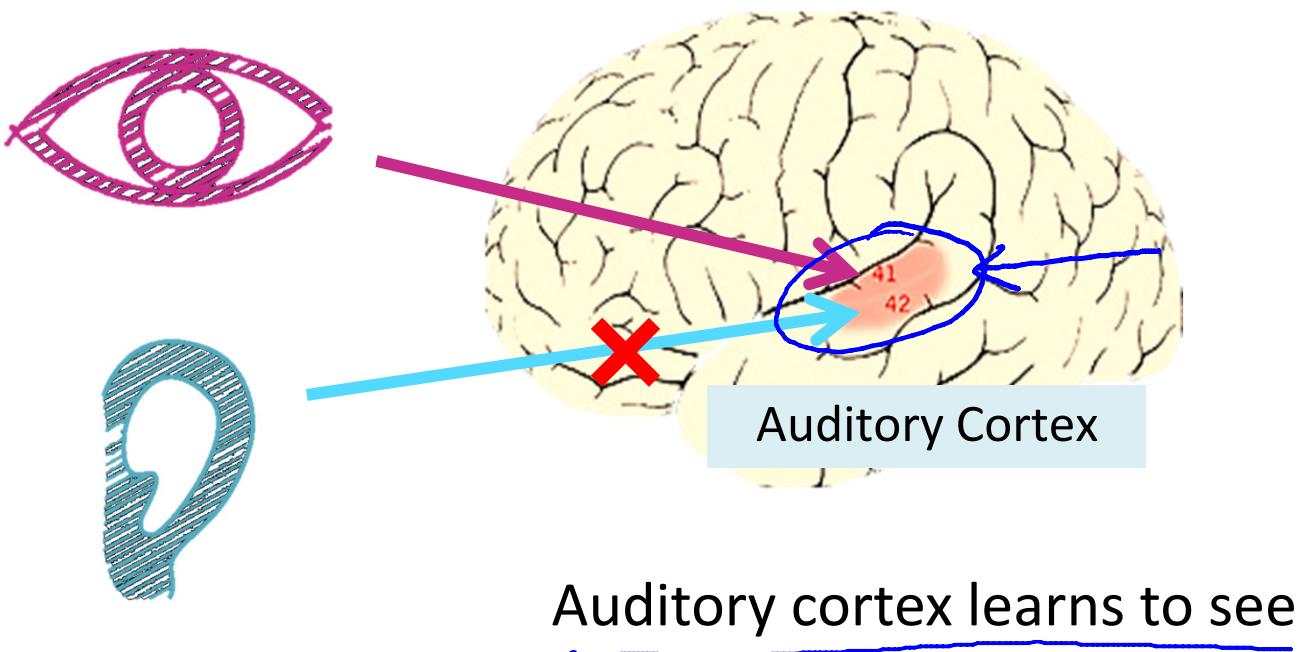
---

## Neurons and the brain

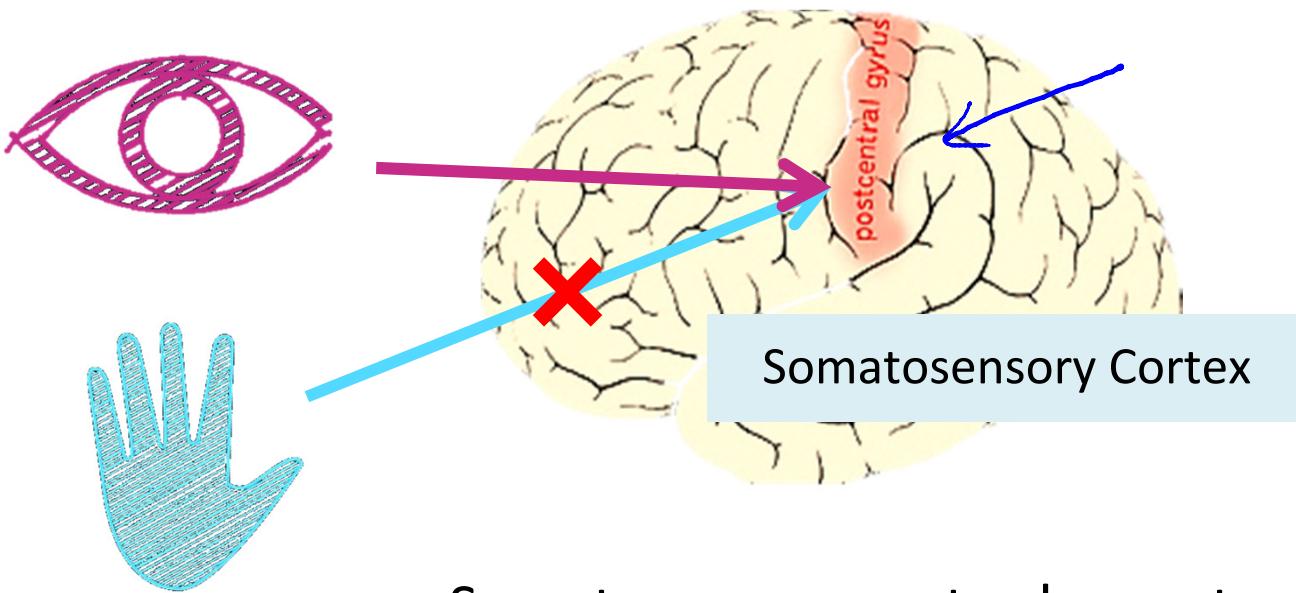
# Neural Networks

- Origins: Algorithms that try to mimic the brain.
- Was very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications

# The “one learning algorithm” hypothesis



# The “one learning algorithm” hypothesis



Somatosensory cortex learns to see

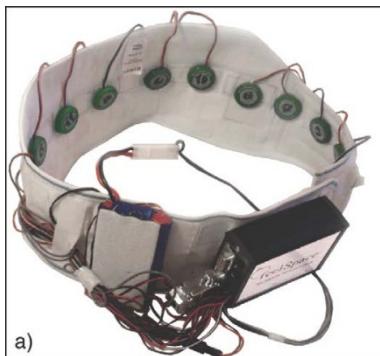
# Sensor representations in the brain



Seeing with your tongue



Human echolocation (sonar)

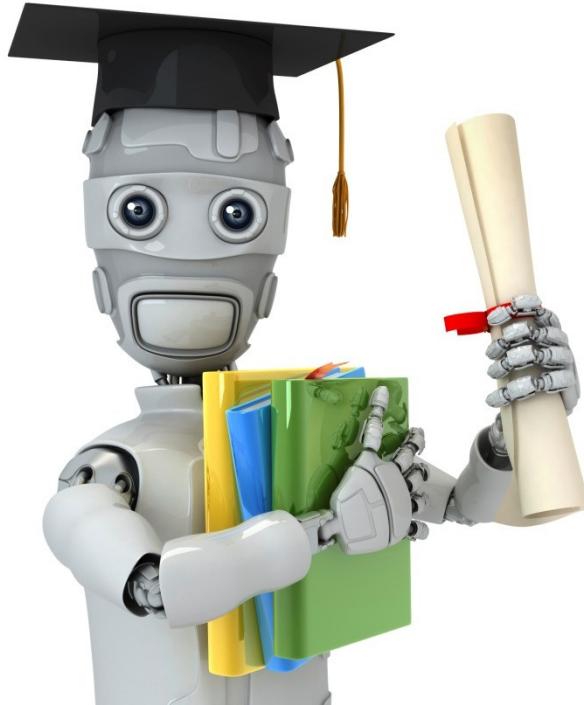


Haptic belt: Direction sense



Implanting a 3<sup>rd</sup> eye





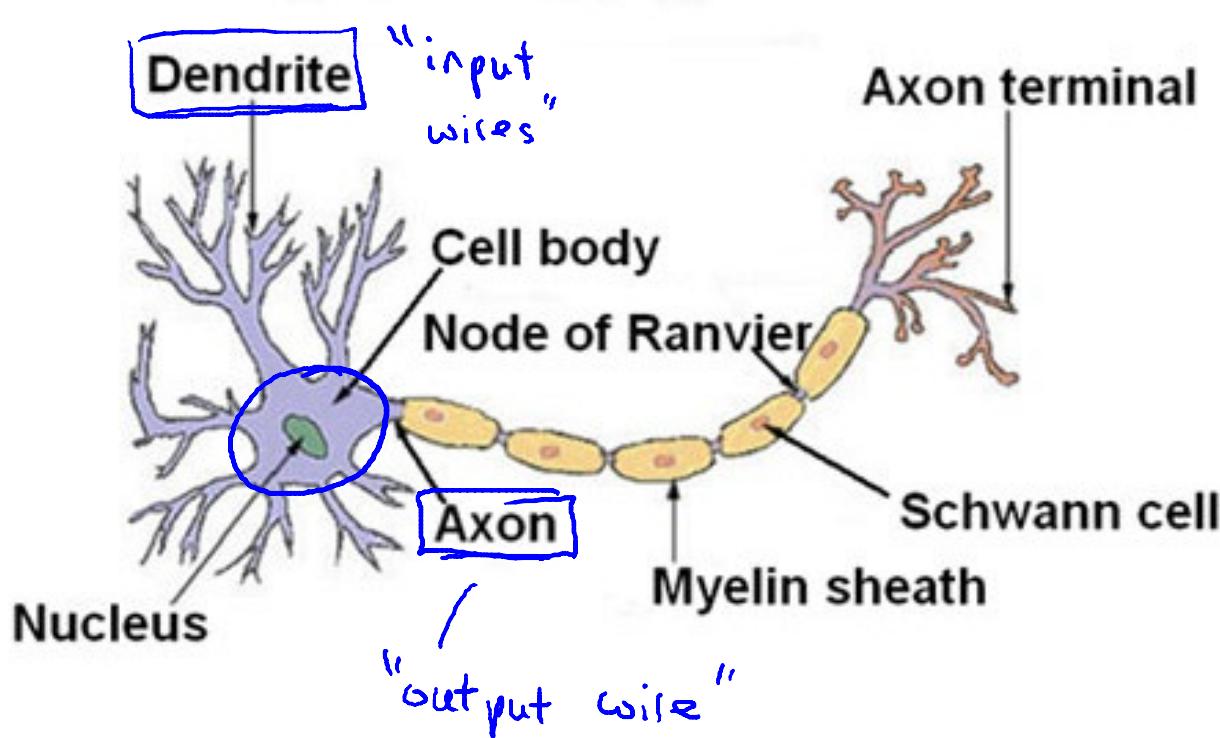
Machine Learning

# Neural Networks: Representation

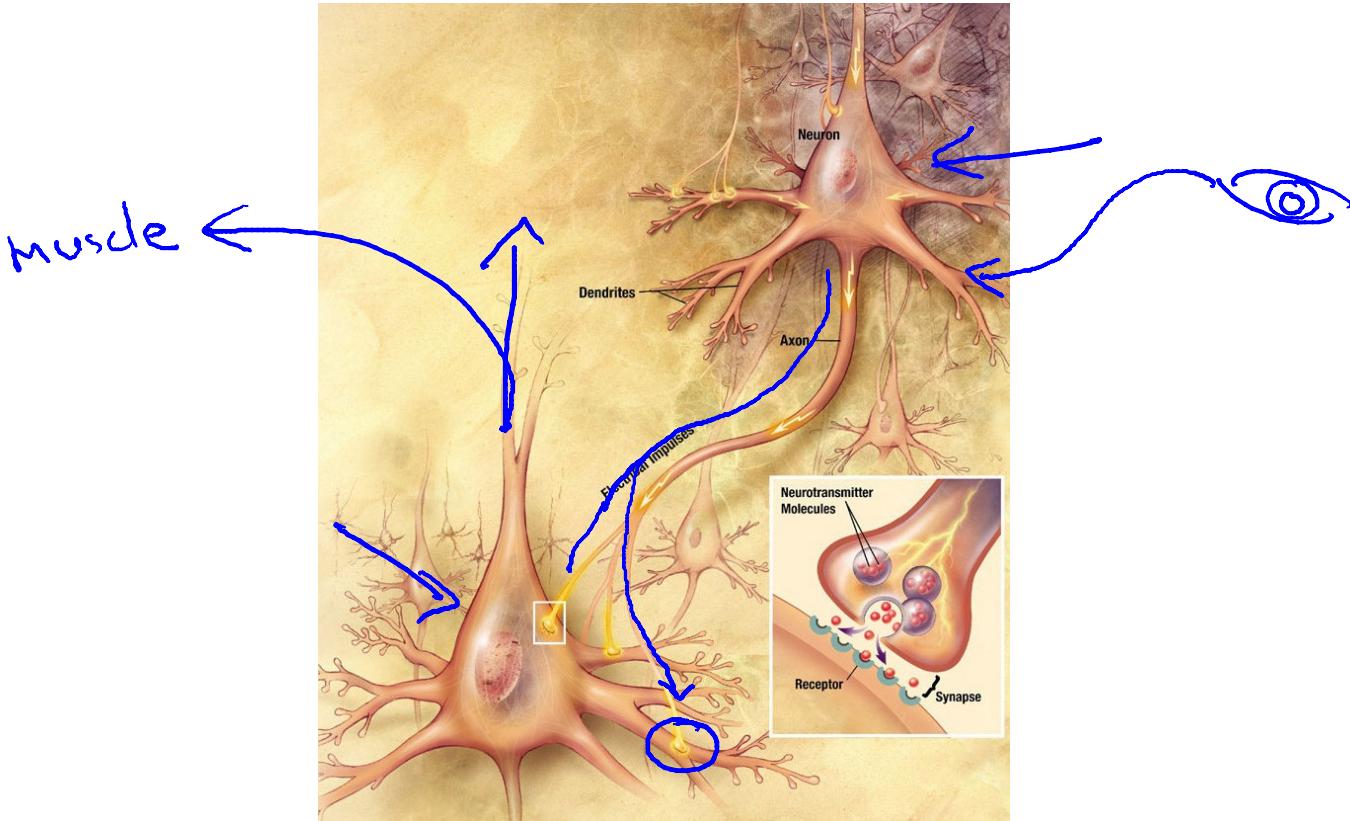
---

## Model representation I

# Neuron in the brain



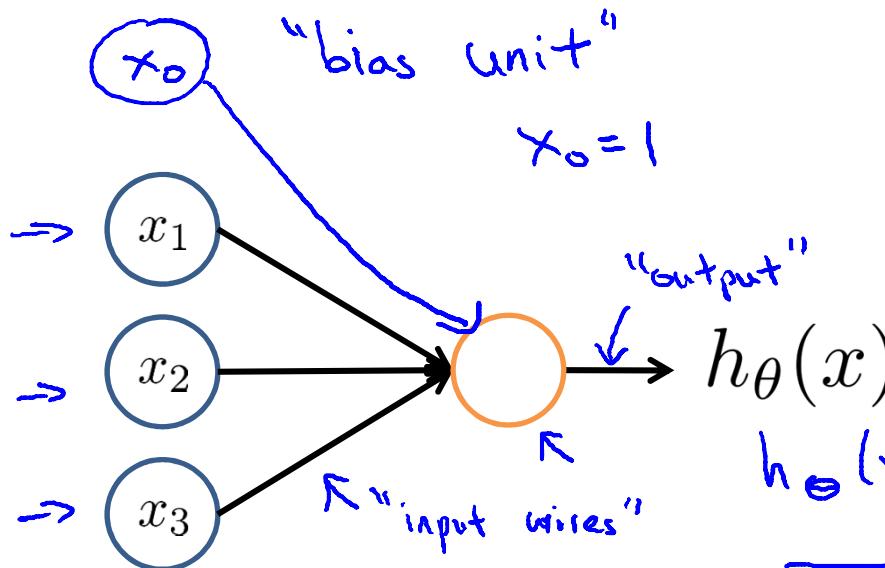
# Neurons in the brain



[Credit: US National Institutes of Health, National Institute on Aging]

Andrew Ng

## Neuron model: Logistic unit



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

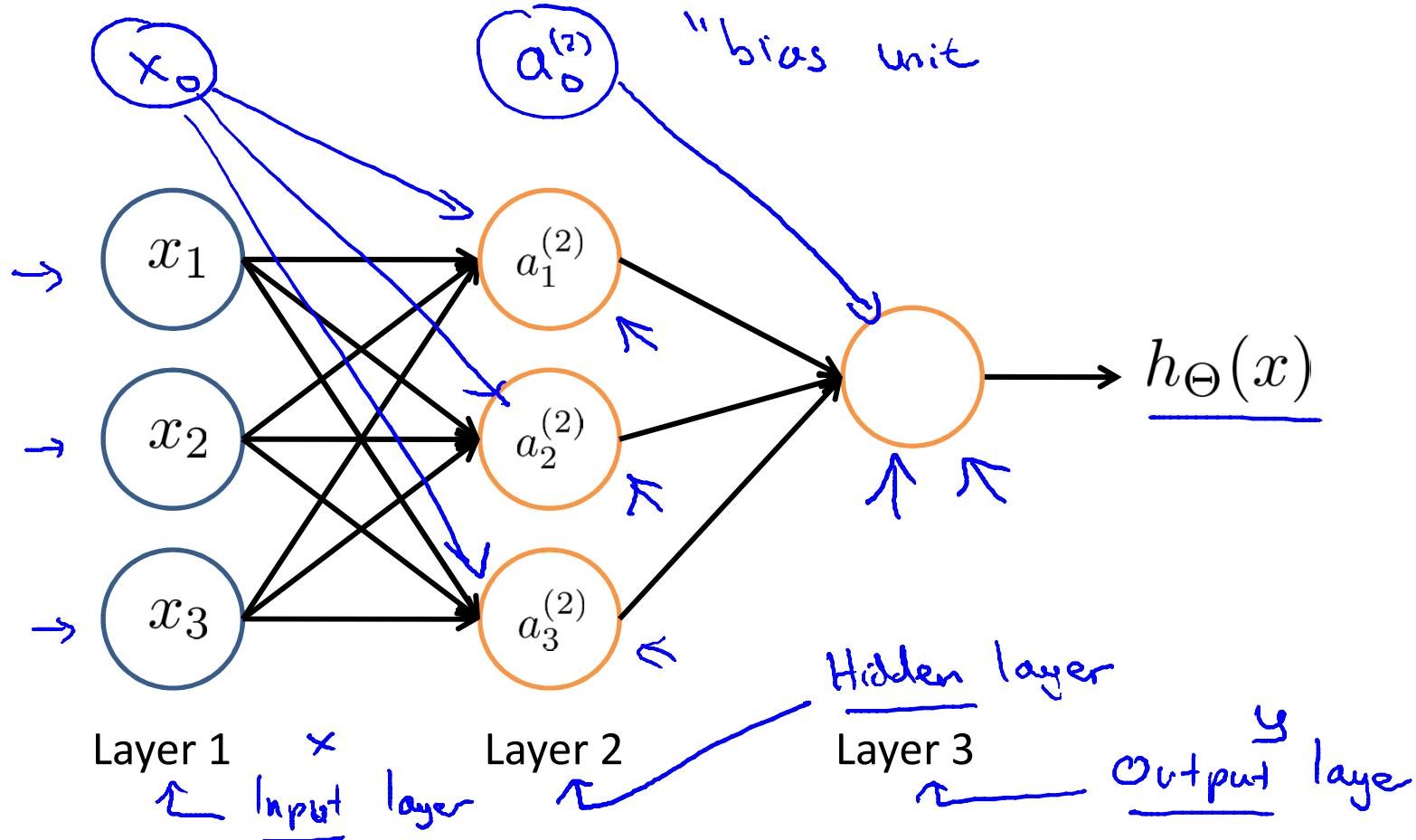
↑  
"weights" ←  
(parameters ←)

$$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$$

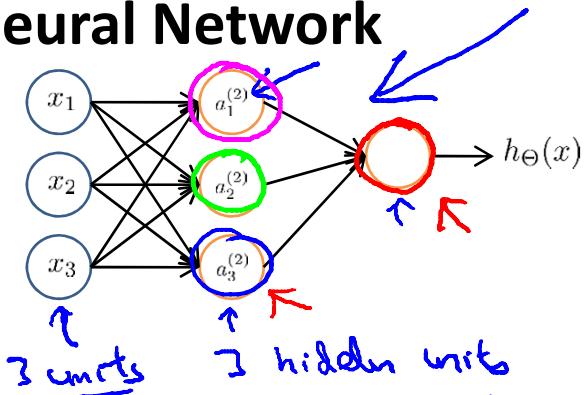
Sigmoid (logistic) activation function.

$$g(z) = \frac{1}{1+e^{-z}}$$

# Neural Network



# Neural Network



→  $a_i^{(j)}$  = “activation” of unit  $i$  in layer  $j$

→  $\Theta^{(j)}$  = matrix of weights controlling function mapping from layer  $j$  to layer  $j + 1$

$$\Theta^{(j)} \in \mathbb{R}^{3 \times 4}$$

$$h_{\Theta}(x)$$

$$\rightarrow a_1^{(2)} = g(\underline{\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3})$$

$$\rightarrow a_2^{(2)} = g(\underline{\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3})$$

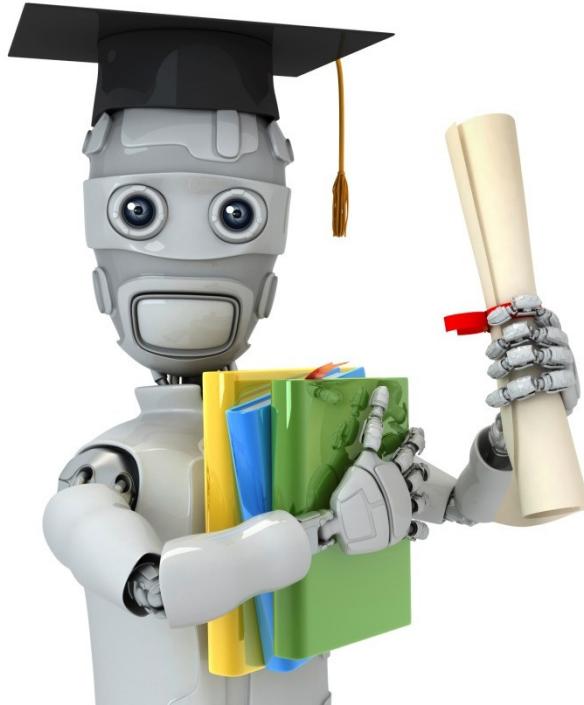
$$\rightarrow a_3^{(2)} = g(\underline{\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3})$$

$$\rightarrow h_{\Theta}(x) = \underline{a_1^{(3)}} = g(\underline{\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)}})$$

- If network has  $s_j$  units in layer  $j$ ,  $s_{j+1}$  units in layer  $j + 1$ , then  $\underline{\Theta^{(j)}}$  will be of dimension  $\underline{s_{j+1} \times (s_j + 1)}$ .

$$s_{j+1} \times (s_j + 1)$$

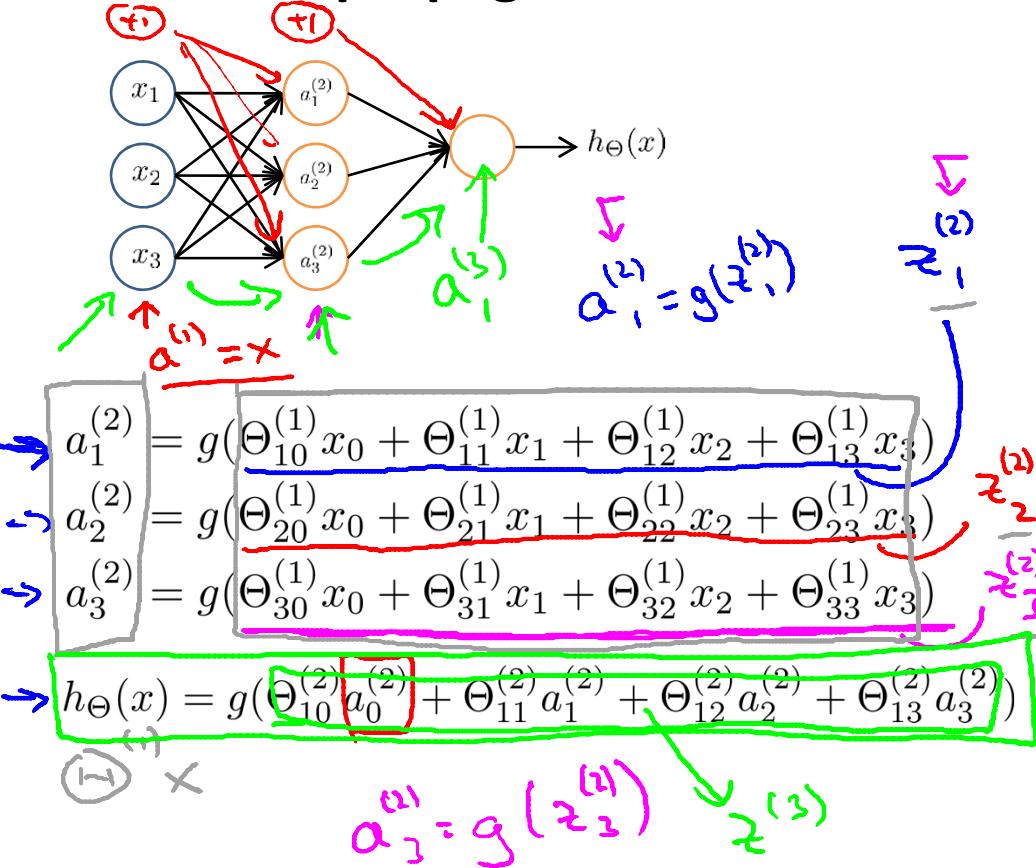




Machine Learning

# Neural Networks: Representation --- Model representation II

## Forward propagation: Vectorized implementation



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$\begin{aligned} z^{(2)} &= \Theta^{(1)} x \\ a^{(2)} &= g(z^{(2)}) \end{aligned}$$

$\Theta^{(1)}$  is crossed out with a red X.

$a^{(2)}$  is circled in red.

$z^{(2)}$  is circled in red.

$\Theta^{(1)}$  is circled in red.

$a^{(2)}$  is circled in red.

$z^{(2)}$  is circled in red.

Add  $a_0^{(2)} = 1$ .  $\rightarrow a^{(2)}$

$a^{(2)}$  is circled in red.

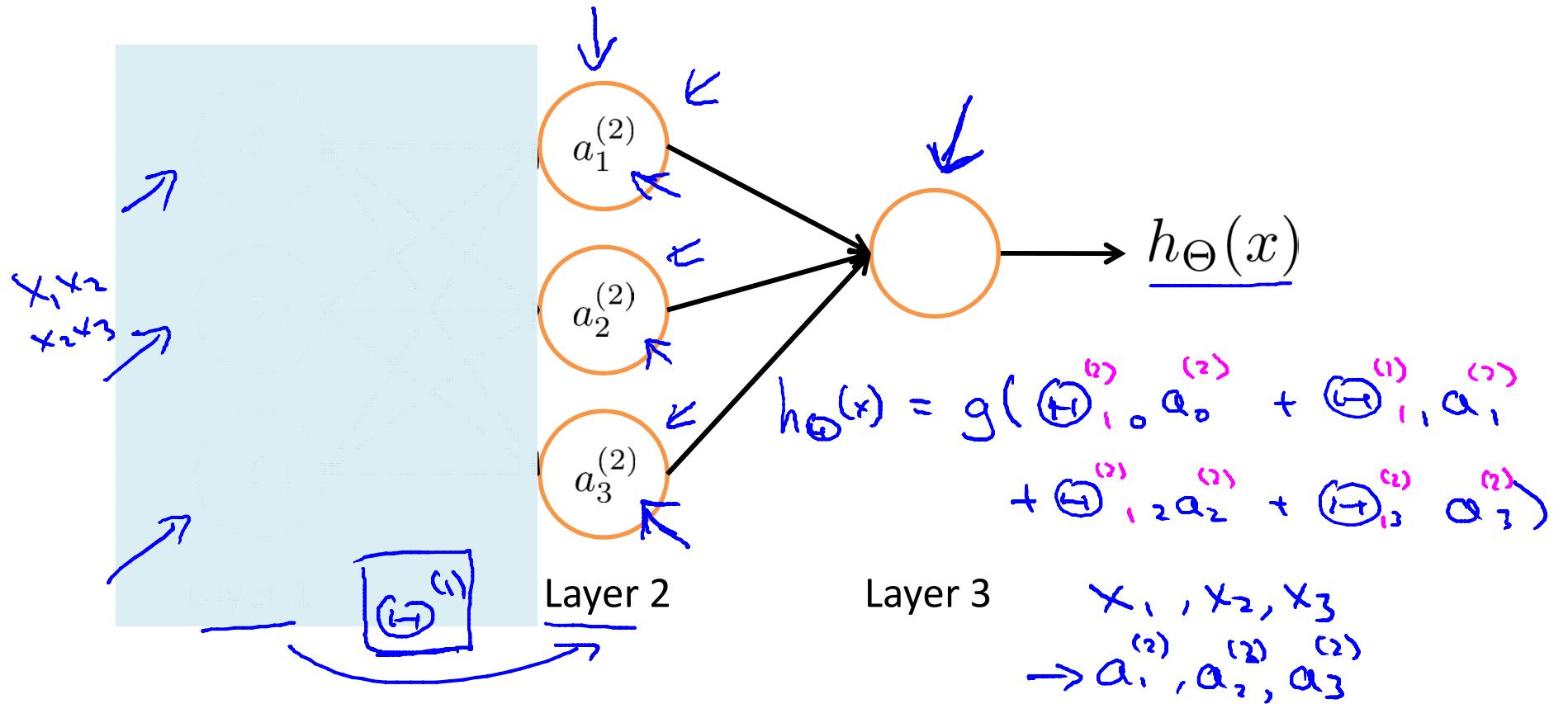
$z^{(3)} = \Theta^{(2)} a^{(2)}$

$h_{\Theta}(x) = a^{(3)} = g(z^{(3)})$

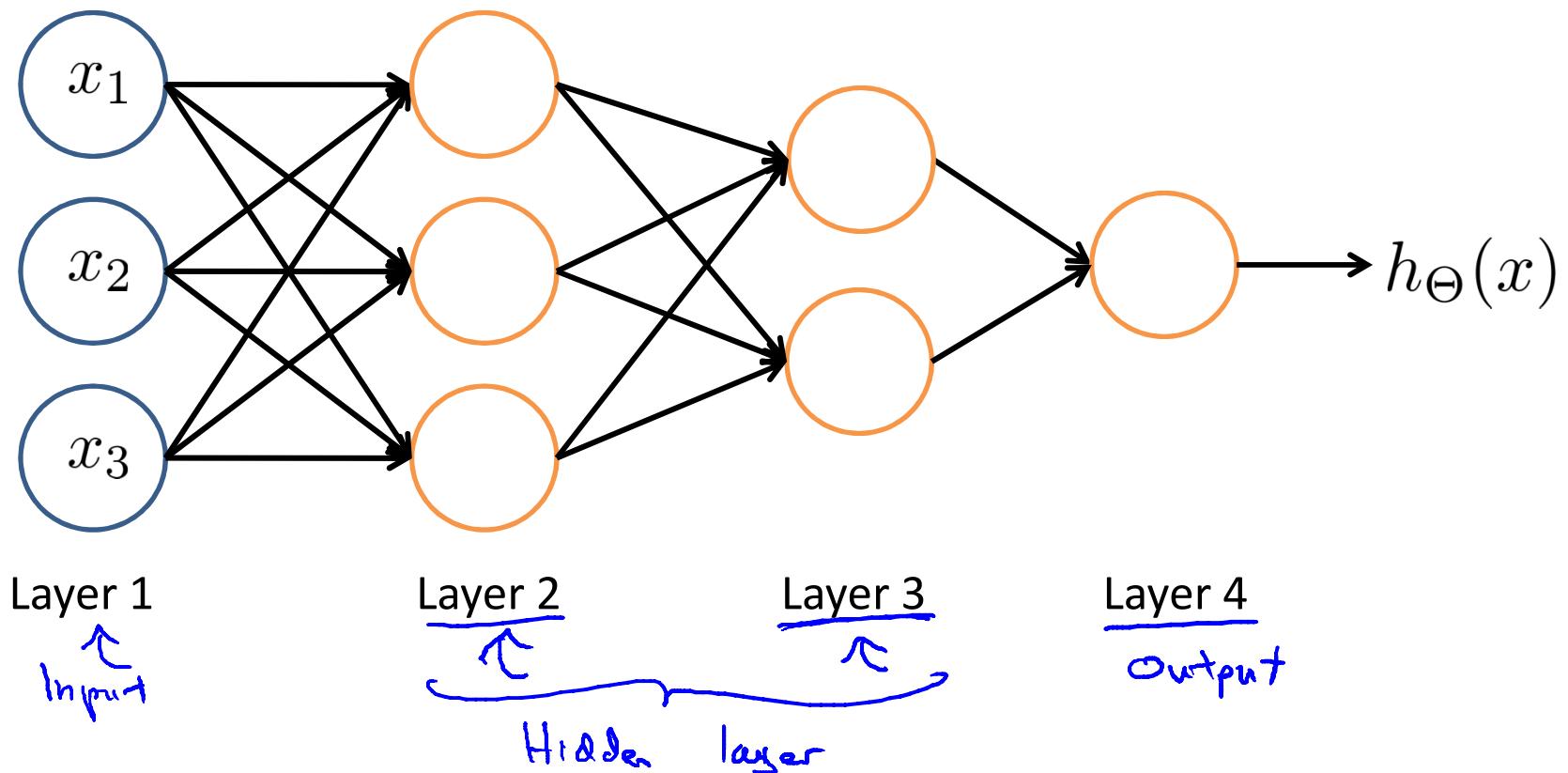
$a^{(3)}$  is circled in red.

$z^{(3)}$  is circled in red.

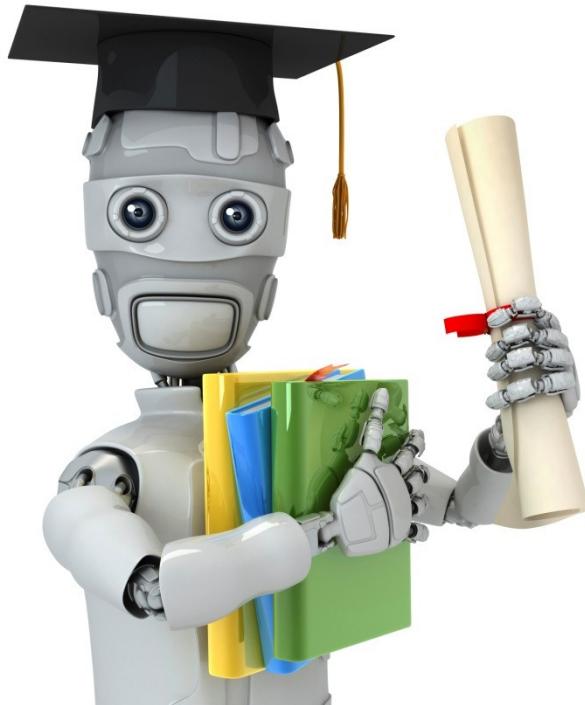
# Neural Network learning its own features



## Other network architectures







Machine Learning

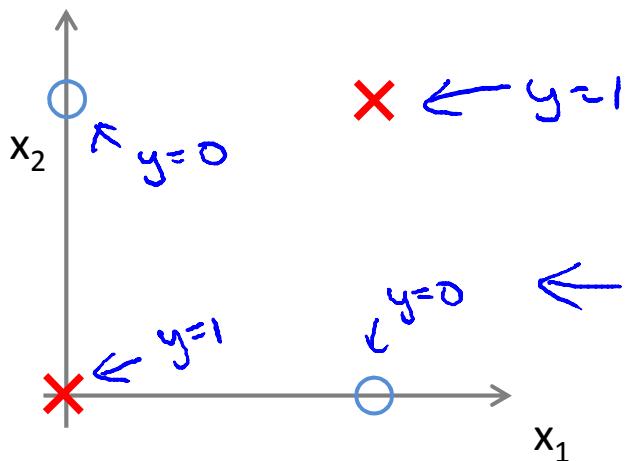
# Neural Networks: Representation

---

## Examples and intuitions I

## Non-linear classification example: XOR/XNOR

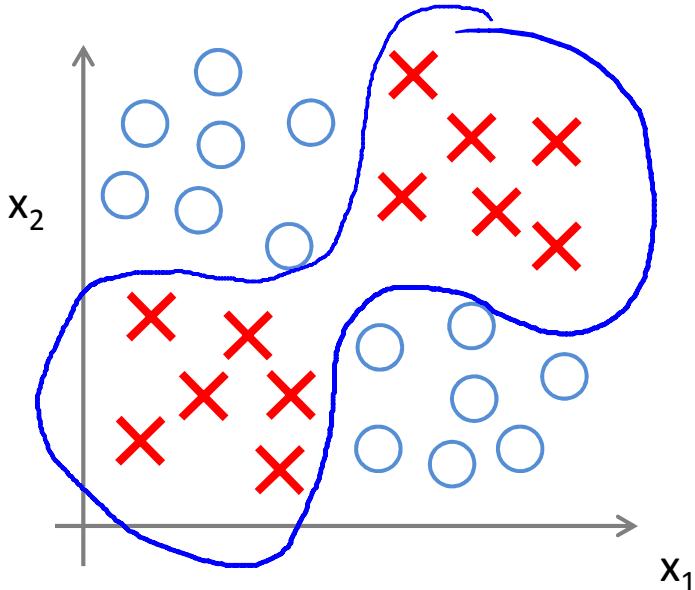
→  $x_1, x_2$  are binary (0 or 1).



$$y = \underline{x_1 \text{ XOR } x_2}$$

→  $\underline{x_1 \text{ XNOR } x_2}$

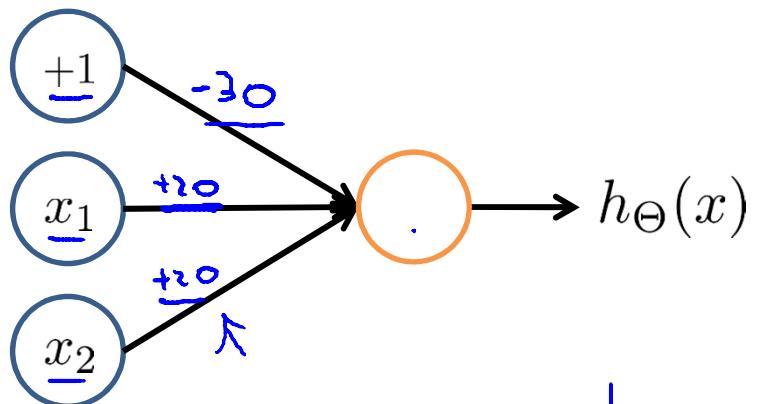
→  $\underline{\text{NOT} (x_1 \text{ XOR } x_2)}$



## Simple example: AND

$$\rightarrow x_1, x_2 \in \{0, 1\}$$

$$\rightarrow y = x_1 \text{ AND } x_2$$



$$\rightarrow h_{\Theta}(x) = g\left(\frac{-30}{\pi} + \frac{20}{\pi}x_1 + \frac{20}{\pi}x_2\right)$$

$\uparrow$                      $\downarrow$   
 $\Theta_{1,0}^{(1)}$        $\Theta_{1,1}^{(1)}$        $\Theta_{1,2}^{(1)}$

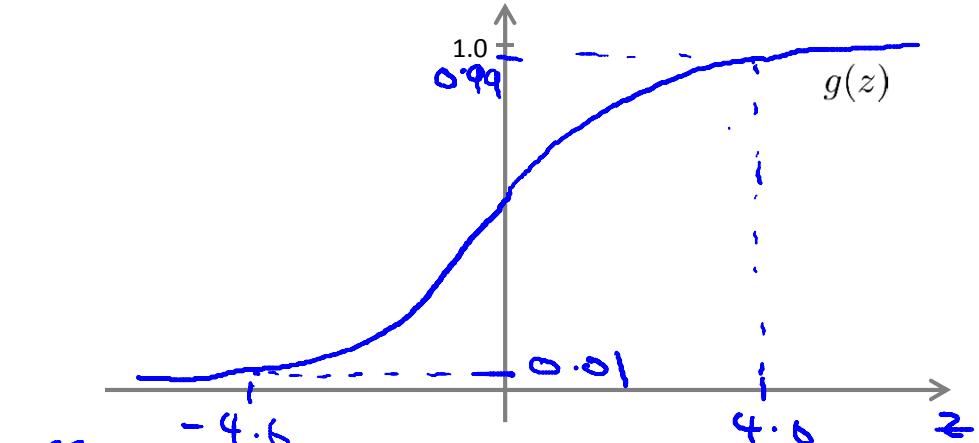
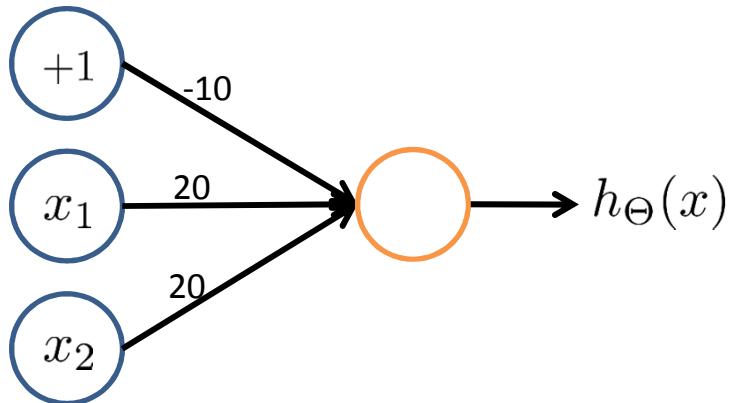


Table showing the output of the hypothesis function  $h_{\Theta}(x)$  for different input combinations:

$x_1$	$x_2$	$h_{\Theta}(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

$h_{\Theta}(x) \approx x_1 \text{ AND } x_2$

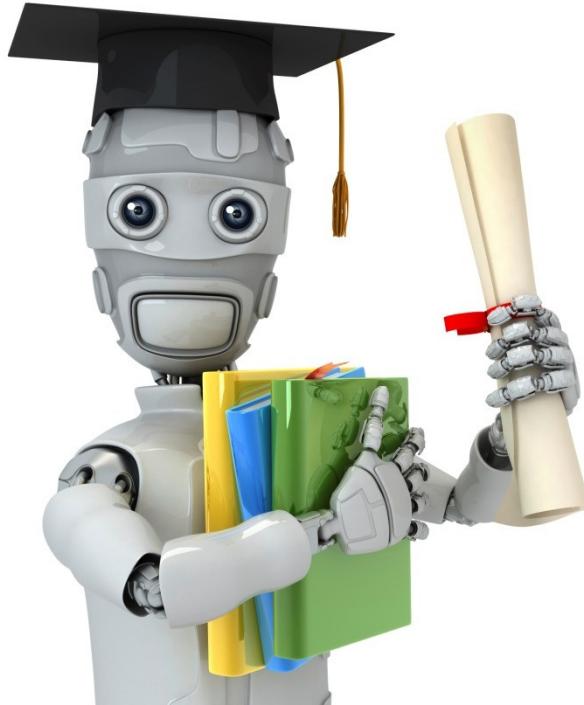
## Example: OR function



$$g(-10 + 20x_1 + 20x_2)$$

$x_1$	$x_2$	$h_{\Theta}(x)$
0	0	$g(-10) \approx 0$
0	1	$g(10) \approx 1$
1	0	$\approx 1$
1	1	$\approx 1$





Machine Learning

# Neural Networks: Representation

---

## Examples and intuitions II

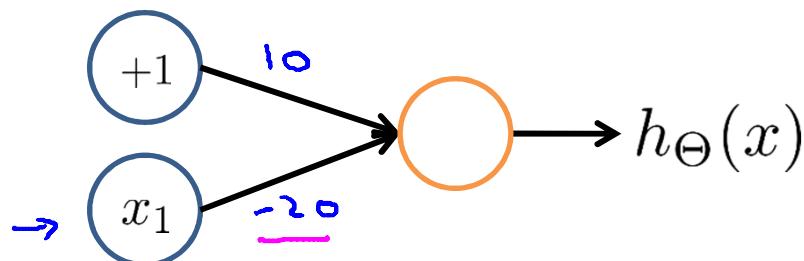
$\rightarrow x_1 \text{ AND } x_2$

$\rightarrow x_1 \text{ OR } x_2$

$\{0, 1\}$ .

**Negation:**

NOT  $x_1$



$x_1$	$h_{\Theta}(x)$
0	$g(10) \approx 1$
1	$g(-20) \approx 0$

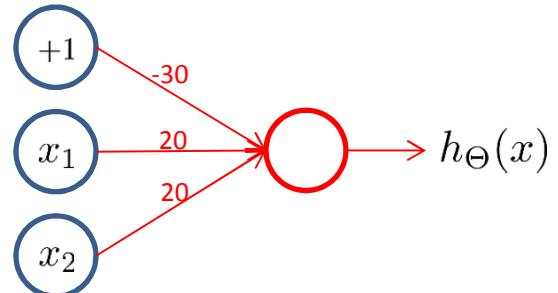
$$h_{\Theta}(x) = g(10 - 20x_1)$$

$\rightarrow (\text{NOT } x_1) \text{ AND } (\text{NOT } x_2)$

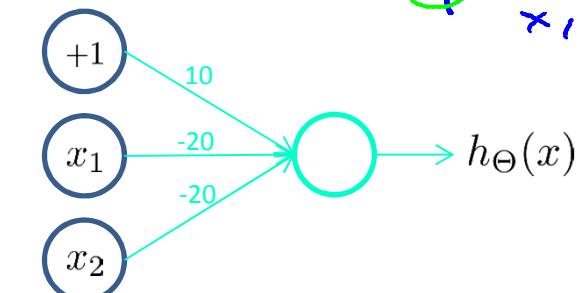
$\begin{cases} = 1 & \text{if and only if} \\ = 0 & \end{cases}$

$\rightarrow x_1 = x_2 = 0$

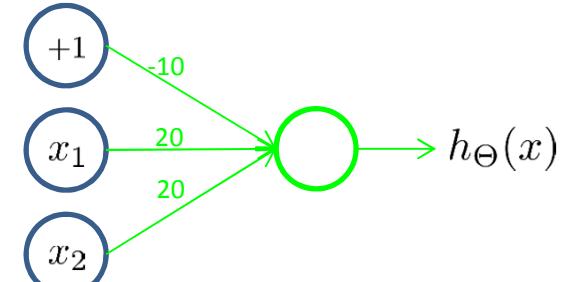
## Putting it together: $x_1$ XNOR $x_2$



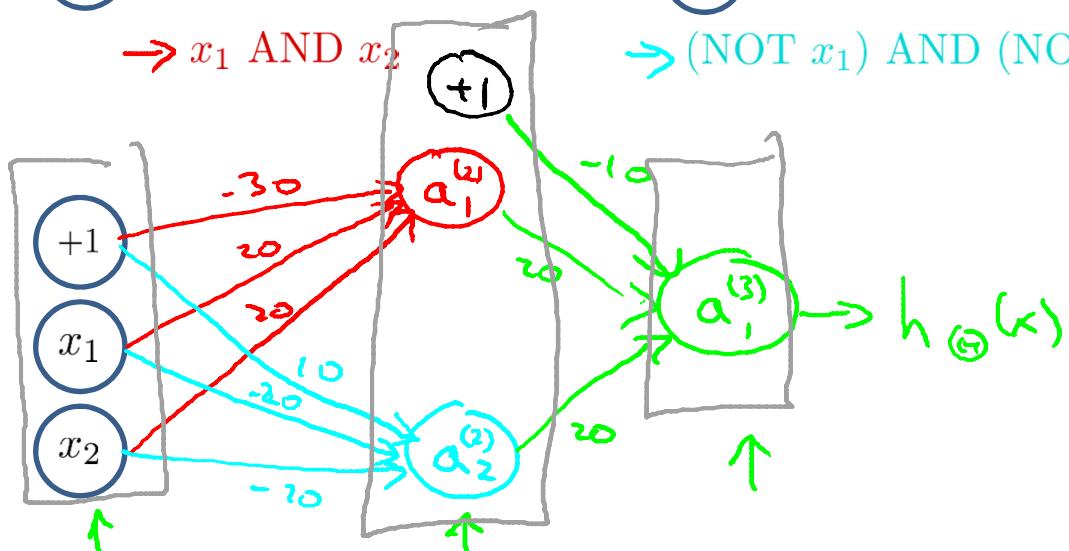
$\rightarrow x_1$  AND  $x_2$



$\rightarrow (\text{NOT } x_1) \text{ AND } (\text{NOT } x_2)$

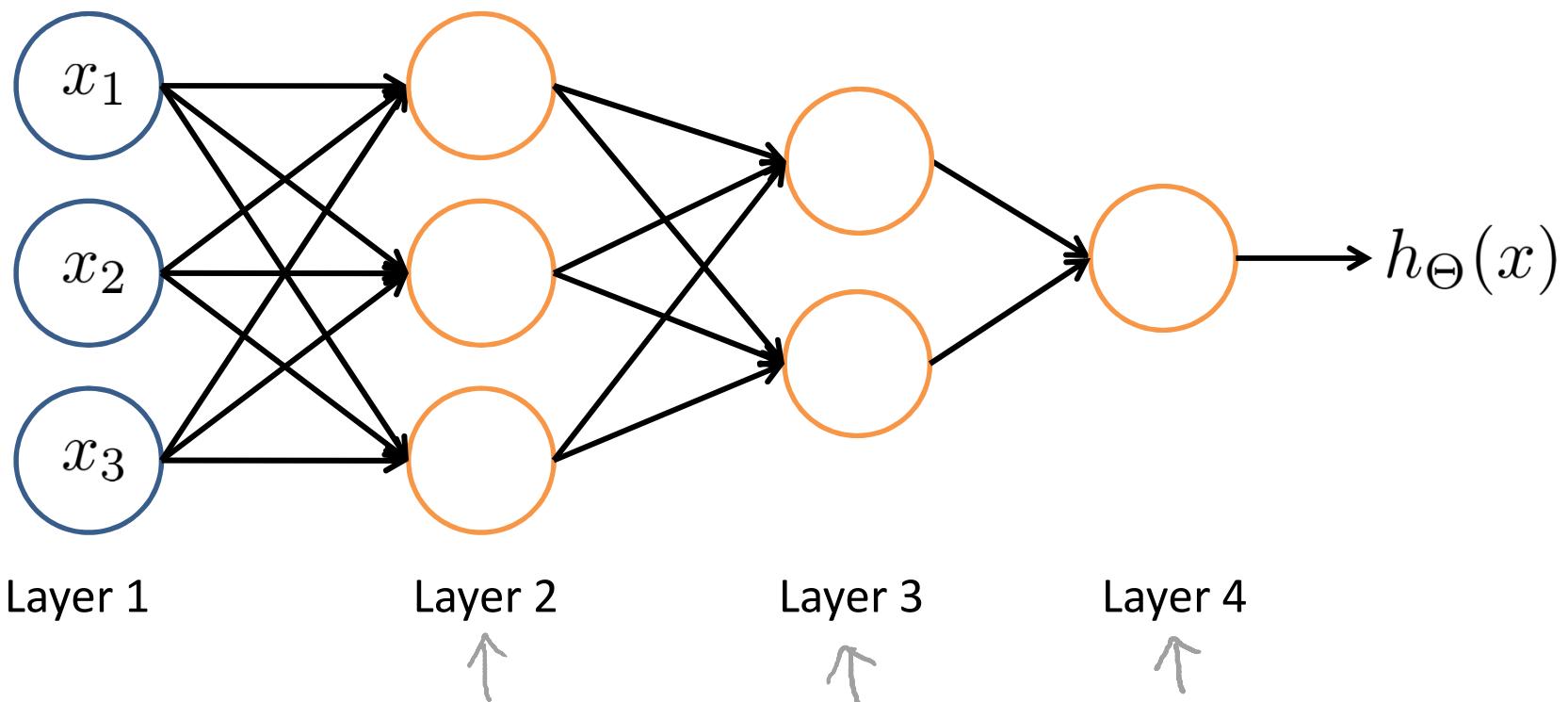


$\rightarrow x_1$  OR  $x_2$

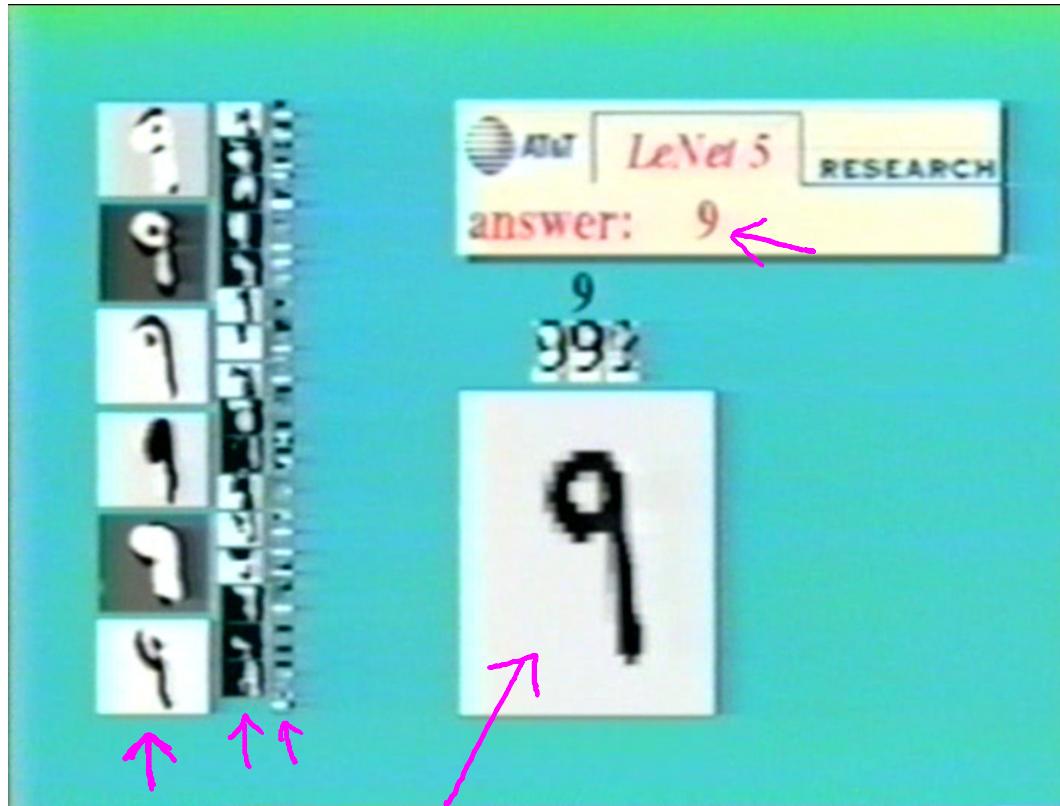


$x_1$	$x_2$	$a_1^{(2)}$	$a_2^{(2)}$	$h_{\Theta}(x)$
0	0	0	1	1 ↘
0	1	0	0	0 ↘
1	0	0	0	0 ↘
1	1	1	0	1 ↘

# Neural Network intuition



# Handwritten digit classification

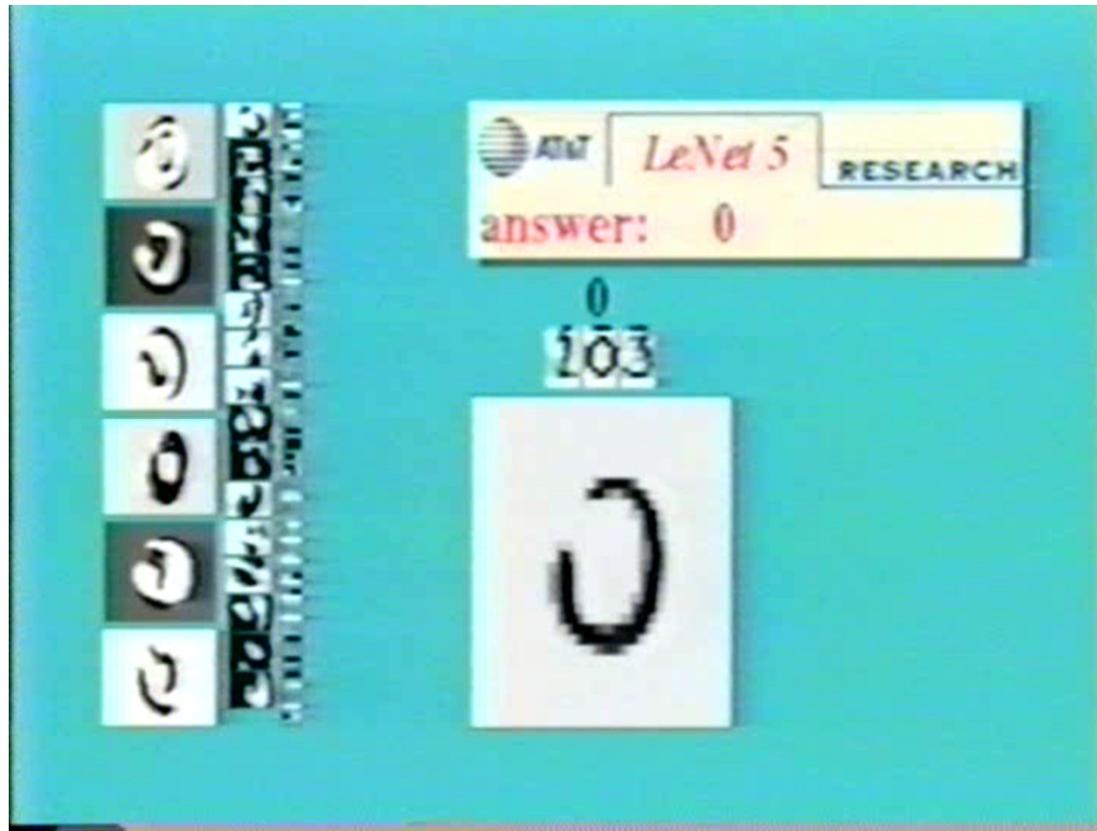


[Courtesy of Yann LeCun]

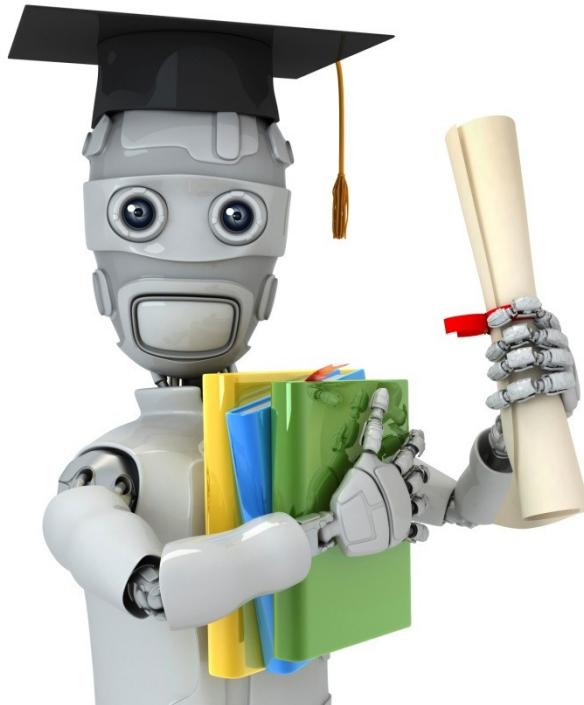


Andrew Ng

# Handwritten digit classification







Machine Learning

# Neural Networks: Representation

---

## Multi-class classification

# Multiple output units: One-vs-all.



Pedestrian



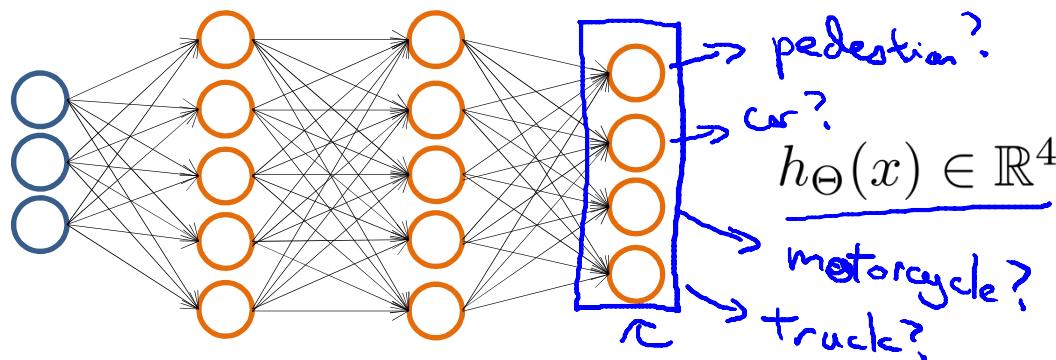
Car



Motorcycle



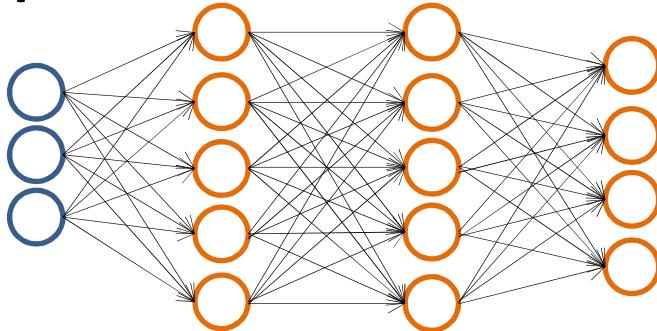
Truck



Want  $h_{\Theta}(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ ,     $h_{\Theta}(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ ,     $h_{\Theta}(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ , etc.

when pedestrian                          when car                          when motorcycle

## Multiple output units: One-vs-all.



$$h_{\Theta}(x) \in \mathbb{R}^4$$

Want  $h_{\Theta}(x) \approx \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ ,  $h_{\Theta}(x) \approx \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$ ,  $h_{\Theta}(x) \approx \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$ , etc.  
when pedestrian      when car      when motorcycle

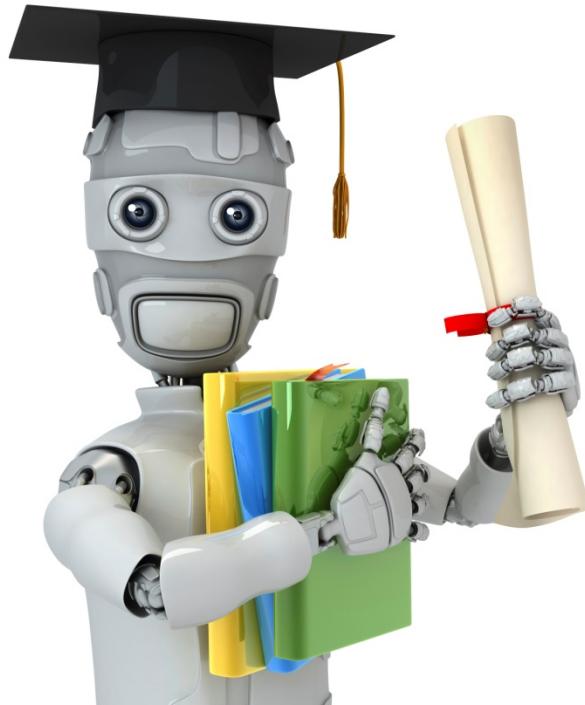
Training set:  $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$

→  $y^{(i)}$  one of  
pedestrian      car      motorcycle      truck

$$\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \quad \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

~~Previously~~  
 $y \in \{1, 2, 3, 4\}$   
 $\underline{h_{\Theta}(x^{(i)}) \approx y^{(i)}} \in \mathbb{R}^4$





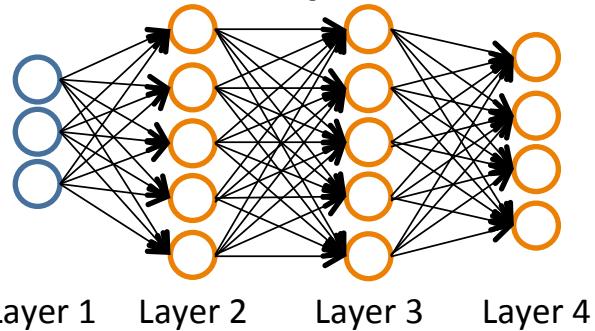
Machine Learning

# Neural Networks: Learning

---

## Cost function

# Neural Network (Classification)



$$\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$$

$L$  = total no. of layers in network

$s_l$  = no. of units (not counting bias unit) in layer  $l$

## Binary classification

$$y = 0 \text{ or } 1$$

1 output unit

## Multi-class classification ( $K$ classes)

$$y \in \mathbb{R}^K \quad \text{E.g. } \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

pedestrian   car   motorcycle   truck

$K$  output units

# Cost function

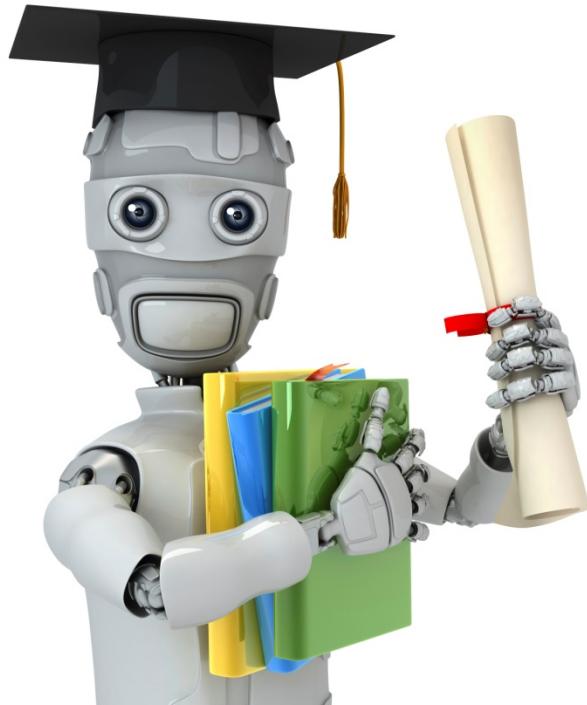
Logistic regression:

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

Neural network:

$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$\begin{aligned} J(\Theta) &= -\frac{1}{m} \left[ \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k) \right] \\ &\quad + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2 \end{aligned}$$



Machine Learning

# Neural Networks: Learning

---

## Backpropagation algorithm

## Gradient computation

$$\Rightarrow \underline{J(\Theta)} = -\frac{1}{m} \left[ \sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log h_\theta(x^{(i)})_k + (1 - y_k^{(i)}) \log(1 - h_\theta(x^{(i)})_k) \right] \\ + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_j^{(l)})^2$$

$$\Rightarrow \min_{\Theta} J(\Theta)$$

Need code to compute:

$$\rightarrow - \underline{J(\Theta)}$$
$$\rightarrow - \underline{\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)} \quad \leftarrow$$

$$\Theta_{ij}^{(l)} \in \mathbb{R}$$

# Gradient computation

Given one training example  $(\underline{x}, \underline{y})$ :

Forward propagation:

$$\underline{a}^{(1)} = \underline{x}$$

$$\rightarrow \underline{z}^{(2)} = \Theta^{(1)} \underline{a}^{(1)}$$

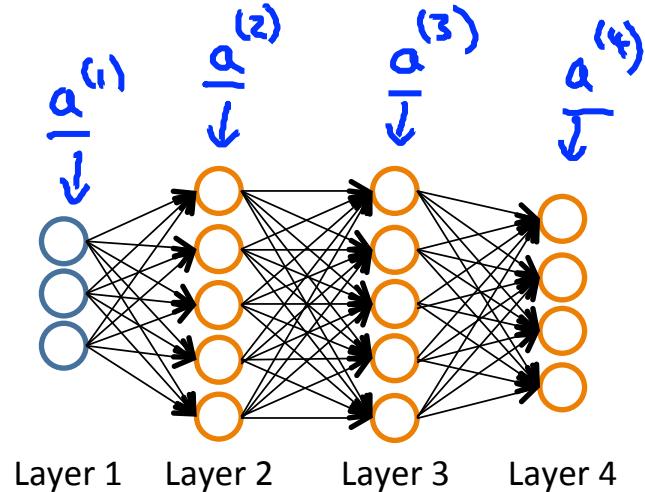
$$\rightarrow \underline{a}^{(2)} = g(\underline{z}^{(2)}) \quad (\text{add } \underline{a}_0^{(2)})$$

$$\rightarrow \underline{z}^{(3)} = \Theta^{(2)} \underline{a}^{(2)}$$

$$\rightarrow \underline{a}^{(3)} = g(\underline{z}^{(3)}) \quad (\text{add } \underline{a}_0^{(3)})$$

$$\rightarrow \underline{z}^{(4)} = \Theta^{(3)} \underline{a}^{(3)}$$

$$\rightarrow \underline{a}^{(4)} = \underline{h}_{\Theta}(\underline{x}) = g(\underline{z}^{(4)})$$

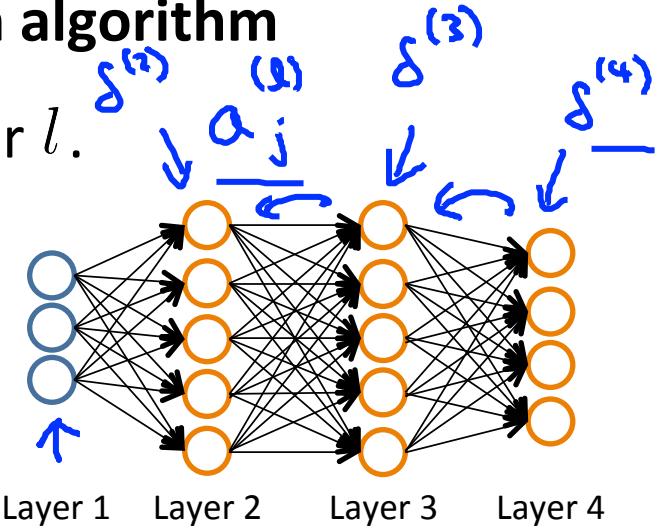


## Gradient computation: Backpropagation algorithm

Intuition:  $\underline{\delta_j^{(l)}}$  = “error” of node  $j$  in layer  $l$ .

For each output unit (layer  $L = 4$ )

$$\underline{\delta_j^{(4)}} = \underline{a_j^{(4)}} - \underline{y_j} \quad (\underline{h_{\Theta}(x)})_j \quad \underline{\delta^{(4)}} = \underline{a^{(4)}} - \underline{y}$$



$$\delta^{(3)} = (\underline{\Theta^{(3)}})^T \underline{\delta^{(4)}} * g'(z^{(3)})$$

$$\delta^{(2)} = (\underline{\Theta^{(2)}})^T \underline{\delta^{(3)}} * g'(z^{(2)})$$

(No  $\delta^{(1)}$ )

$$\frac{\partial}{\partial \Theta^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{a^{(3)}}{a^{(2)}} * \frac{(1-a^{(3)})}{a^{(2)} * (1-a^{(2)})}$$

(ignoring  $\lambda$ ; if  
 $\lambda = 0$ )

## Backpropagation algorithm

→ Training set  $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$

Set  $\Delta_{ij}^{(l)} = 0$  (for all  $l, i, j$ ).

(use to compute  $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$ )

For  $i = 1$  to  $m$  ←

$(\underline{x^{(i)}}, \underline{y^{(i)}})$

Set  $\underline{a^{(1)}} = \underline{x^{(i)}}$

→ Perform forward propagation to compute  $\underline{a^{(l)}}$  for  $l = 2, 3, \dots, L$

→ Using  $\underline{y^{(i)}}$ , compute  $\delta^{(L)} = \underline{a^{(L)}} - \underline{y^{(i)}}$

→ Compute  $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$

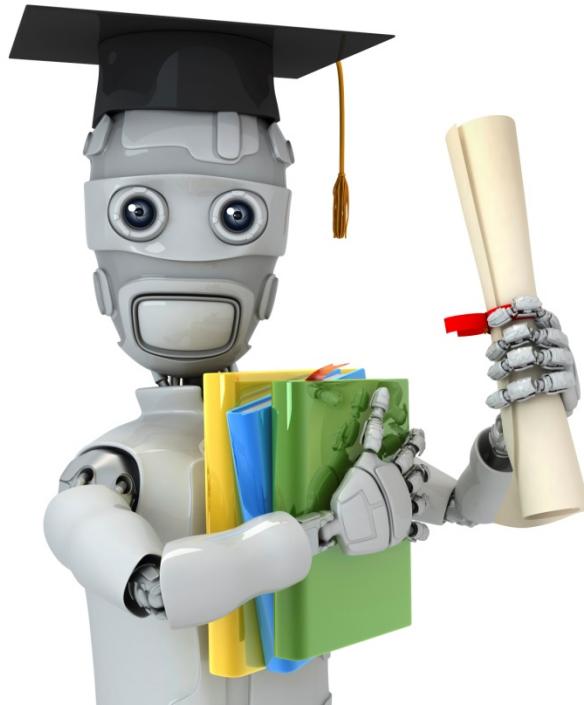
$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + \delta^{(l+1)} (a^{(l)})^T$ .

$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}$  if  $j \neq 0$

$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$  if  $j = 0$

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$$

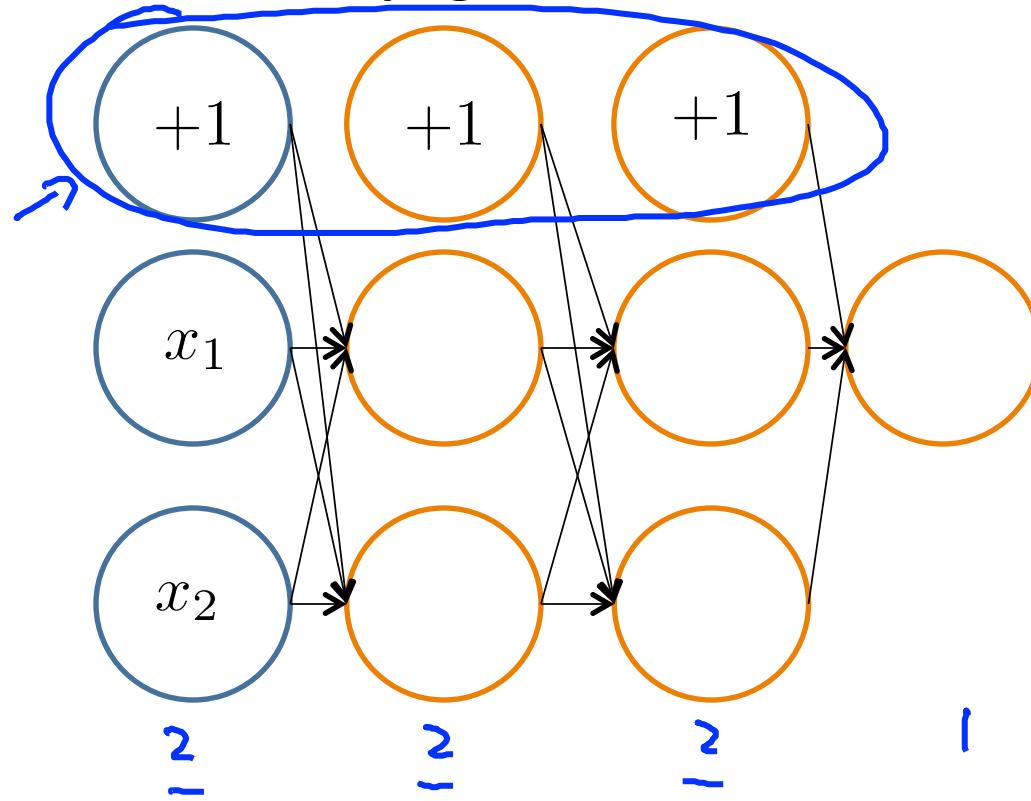


Machine Learning

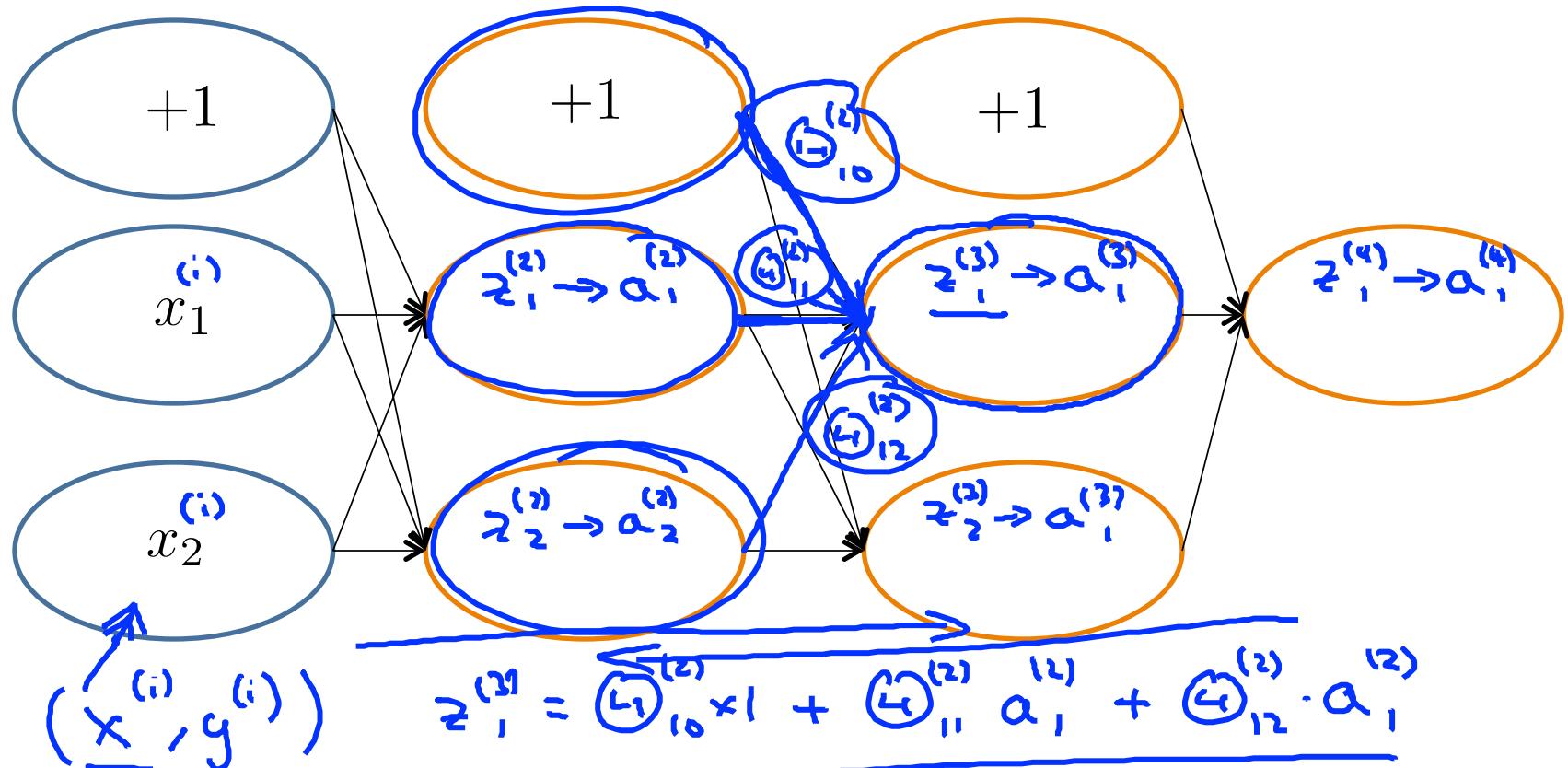
# Neural Networks: Learning

## Backpropagation intuition

## Forward Propagation



# Forward Propagation



# What is backpropagation doing?

$$J(\Theta) = -\frac{1}{m} \left[ \sum_{i=1}^m y^{(i)} \log(h_\Theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\Theta(x^{(i)})) \right] \\ + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

$(x^{(i)}, y^{(i)})$

Focusing on a single example  $x^{(i)}$ ,  $y^{(i)}$ , the case of 1 output unit, and ignoring regularization ( $\lambda = 0$ ),

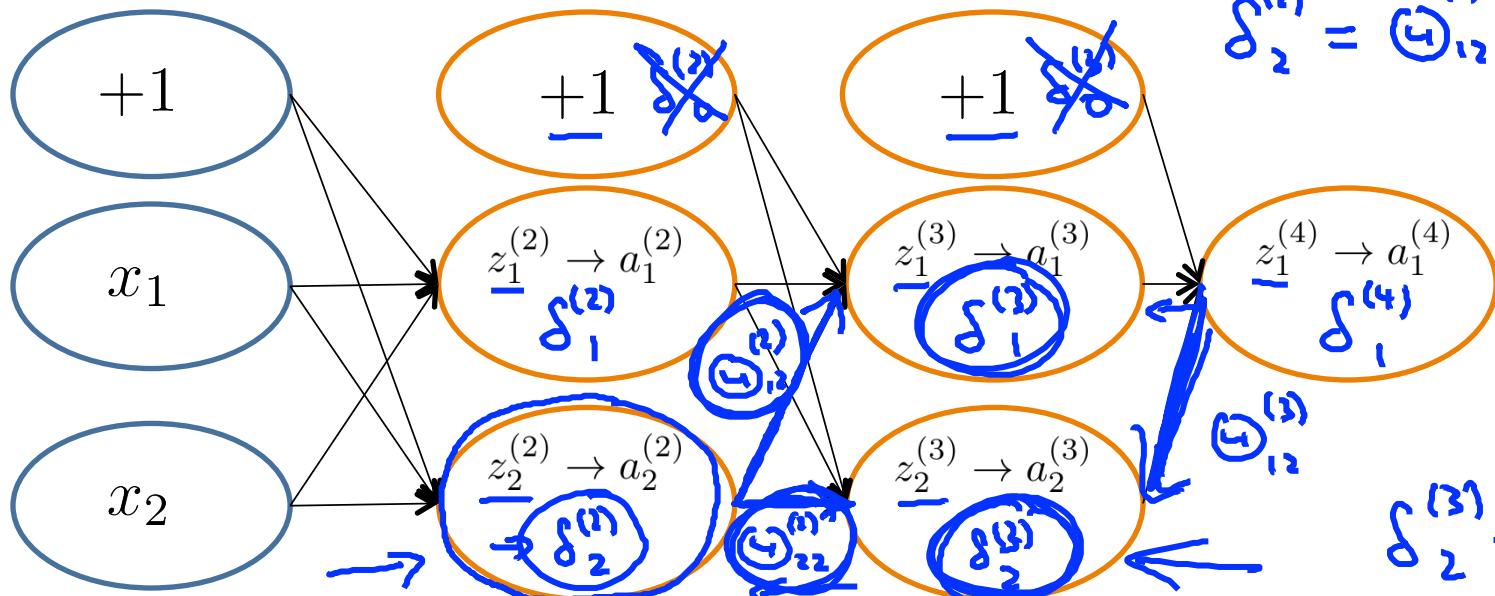
Note: Mistake on lecture, it is supposed to be  $1 - h(x)$

$$\text{cost}(i) = y^{(i)} \log h_\Theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\Theta(x^{(i)}))$$

(Think of  $\text{cost}(i) \approx (h_\Theta(x^{(i)}) - y^{(i)})^2$ )

I.e. how well is the network doing on example i?

## Forward Propagation



$\rightarrow \delta_j^{(l)}$  = “error” of cost for  $\underline{a}_j^{(l)}$  (unit  $j$  in layer  $l$ ).

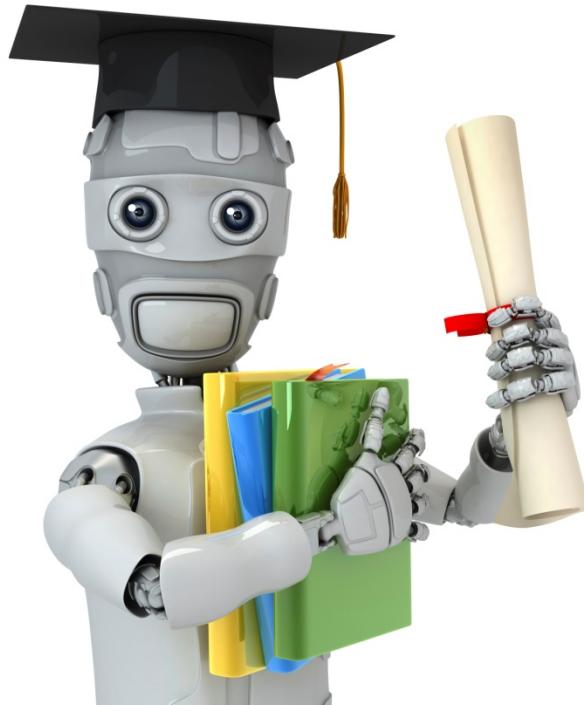
Formally,  $\underline{\delta}_j^{(l)} = \frac{\partial \text{cost}(i)}{\partial z_j^{(l)}}$  (for  $j \geq 0$ ), where

$$\text{cost}(i) = \underline{y}^{(i)} \log h_{\Theta}(x^{(i)}) + (1 - \underline{y}^{(i)}) \log \underline{h}_{\Theta}(x^{(i)})$$

$$\delta_1^{(4)} = \underline{y}^{(i)} - \underline{a}_1^{(4)}$$

$$\delta_2^{(2)} = \underline{a}_{12}^{(2)} \delta_1^{(3)} + \underline{a}_{22}^{(2)} \delta_2^{(3)}$$

$$\delta_2^{(3)} = \underline{a}_{12}^{(3)} \cdot \delta_1^{(4)}$$



Machine Learning

# Neural Networks: Learning

---

## Implementation note: Unrolling parameters

## Advanced optimization

```
function [jVal, gradient] = costFunction(theta)
    ...
optTheta = fminunc(@costFunction, initialTheta, options)
```

Diagram annotations:

- An arrow points from the gradient parameter to the text  $\mathbb{R}^{n+1}$ .
- An arrow points from the theta parameter to the text  $\mathbb{R}^{n+1}$  (vectors).
- An arrow points from the initialTheta parameter to the text "Neural Network ( $L=4$ ):".

Neural Network ( $L=4$ ):

→  $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$  - matrices (Theta1, Theta2, Theta3)

→  $D^{(1)}, D^{(2)}, D^{(3)}$  - matrices (D1, D2, D3)

"Unroll" into vectors

## Example

$$s_1 = 10, s_2 = 10, s_3 = 1$$

$\Theta^{(1)} \in \mathbb{R}^{10 \times 11}$ ,  $\Theta^{(2)} \in \mathbb{R}^{10 \times 11}$ ,  $\Theta^{(3)} \in \mathbb{R}^{1 \times 11}$

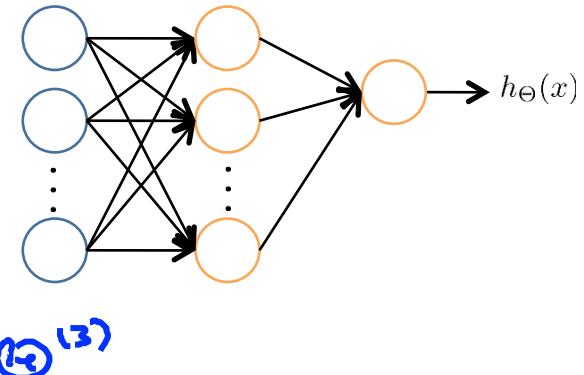
$D^{(1)} \in \mathbb{R}^{10 \times 11}$ ,  $D^{(2)} \in \mathbb{R}^{10 \times 11}$ ,  $D^{(3)} \in \mathbb{R}^{1 \times 11}$

```
→ thetaVec = [ Theta1(:); Theta2(:); Theta3(:) ];  
→ DVec = [D1(:); D2(:); D3(:)];
```

```
Theta1 = reshape(thetaVec(1:110), 10, 11);
```

```
→ Theta2 = reshape(thetaVec(111:220), 10, 11);
```

```
→ Theta3 = reshape(thetaVec(221:231), 1, 11);
```



## Learning Algorithm

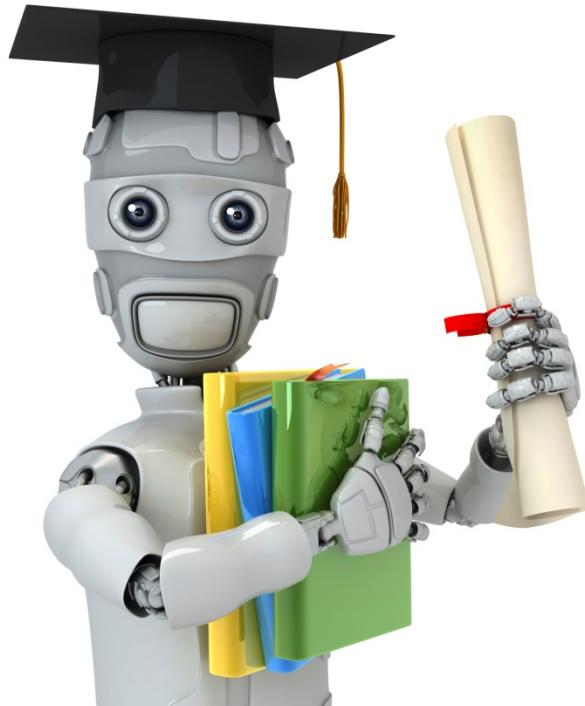
- Have initial parameters  $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ .
- Unroll to get `initialTheta` to pass to
- `fminunc(@costFunction, initialTheta, options)`

```
function [jval, gradientVec] = costFunction(thetaVec)
```

→ From thetaVec, get  $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ . reshape

→ Use forward prop/back prop to compute  $D^{(1)}, D^{(2)}, D^{(3)}$  and  $D_{\cdot}^{(1)}, D^{(2)}, D^{(3)}$   $J(\Theta)$

Unroll  $D_{\cdot}^{(1)}, D^{(2)}, D^{(3)}$  to get gradientVec.



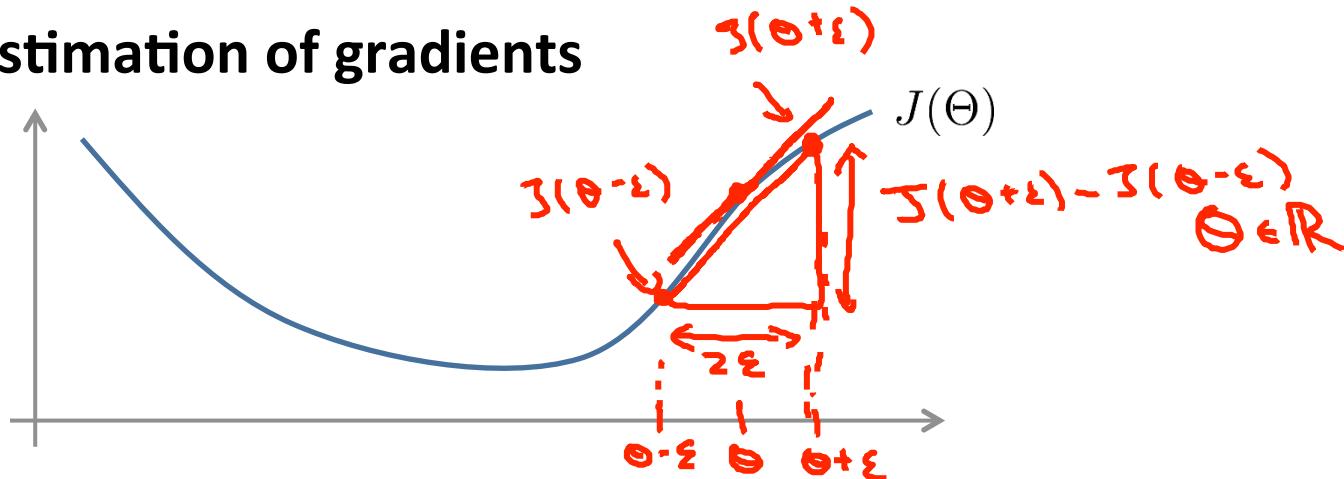
Machine Learning

# Neural Networks: Learning

---

## Gradient checking

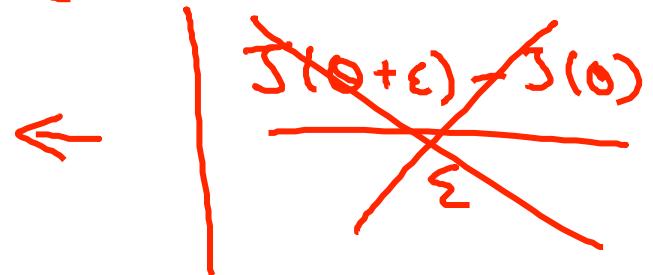
## Numerical estimation of gradients



$$\frac{\partial}{\partial \theta} J(\theta) \approx$$

$$\frac{J(\theta + \epsilon) - J(\theta - \epsilon)}{2\epsilon}$$

$$\epsilon = 10^{-4}$$



Implement: gradApprox =  $(J(\text{theta} + \text{EPSILON}) - J(\text{theta} - \text{EPSILON})) / (2 * \text{EPSILON})$

## Parameter vector $\theta$

- $\theta \in \mathbb{R}^n$  (E.g.  $\theta$  is “unrolled” version of  $\underline{\Theta^{(1)}}, \underline{\Theta^{(2)}}, \underline{\Theta^{(3)}}$ )
- $\theta = [\theta_1, \theta_2, \theta_3, \dots, \theta_n]$
- $\frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1 + \epsilon, \theta_2, \theta_3, \dots, \theta_n) - J(\theta_1 - \epsilon, \theta_2, \theta_3, \dots, \theta_n)}{2\epsilon}$
- $\frac{\partial}{\partial \theta_2} J(\theta) \approx \frac{J(\theta_1, \theta_2 + \epsilon, \theta_3, \dots, \theta_n) - J(\theta_1, \theta_2 - \epsilon, \theta_3, \dots, \theta_n)}{2\epsilon}$
- ⋮
- $\frac{\partial}{\partial \theta_n} J(\theta) \approx \frac{J(\theta_1, \theta_2, \theta_3, \dots, \theta_n + \epsilon) - J(\theta_1, \theta_2, \theta_3, \dots, \theta_n - \epsilon)}{2\epsilon}$

```

for i = 1:n, ←
  thetaPlus = theta;
  thetaPlus(i) = thetaPlus(i) + EPSILON;
  thetaMinus = theta;
  thetaMinus(i) = thetaMinus(i) - EPSILON;
  gradApprox(i) = (J(thetaPlus) - J(thetaMinus))
                  / (2*EPSILON);
end;

```

$\frac{\partial}{\partial \theta_j} J(\theta)$ .

Check that gradApprox  $\approx$  DVec ←

From back prop.

$$\begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_i + \epsilon \\ \vdots \\ \theta_n \end{bmatrix} \rightarrow \theta_0 \dots \theta_n$$

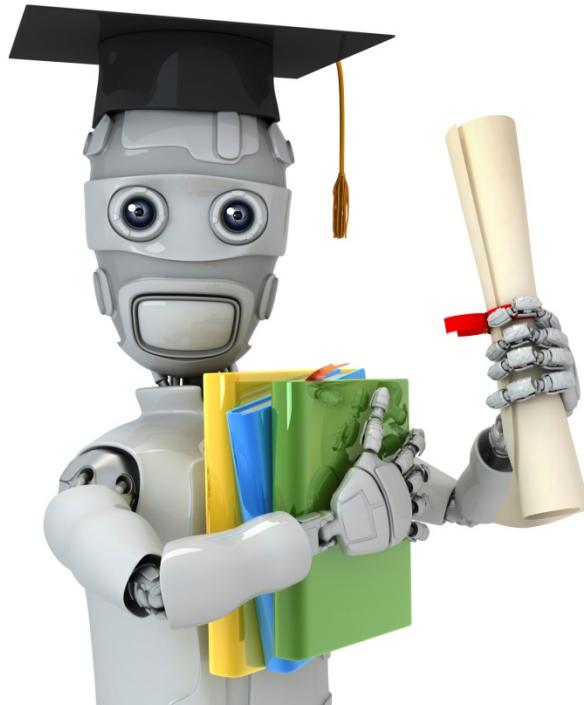
## Implementation Note:

- - Implement backprop to compute DVec (unrolled  $D^{(1)}, D^{(2)}, D^{(3)}$ ).
- - Implement numerical gradient check to compute gradApprox.
- - Make sure they give similar values.
- - Turn off gradient checking. Using backprop code for learning.

## Important:

- - Be sure to disable your gradient checking code before training your classifier. If you run numerical gradient computation on every iteration of gradient descent (or in the inner loop of `costFunction(...)`) your code will be very slow.

DVec  
 $\delta^{(1)}, \delta^{(2)}, \delta^{(3)}$



Machine Learning

# Neural Networks: Learning

---

## Random initialization

## Initial value of $\Theta$

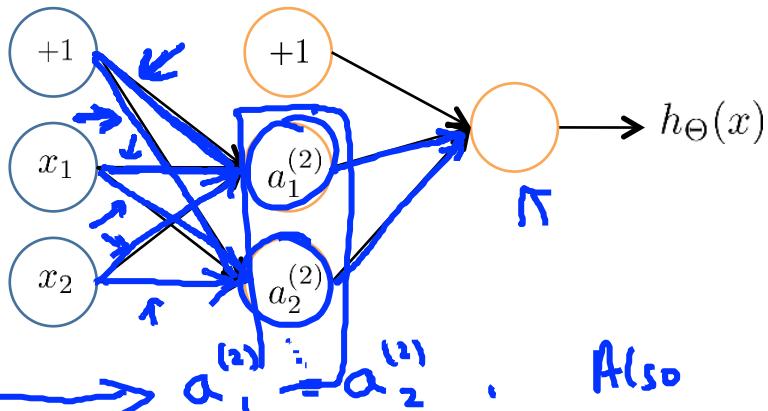
For gradient descent and advanced optimization method, need initial value for  $\Theta$ .

```
optTheta = fminunc(@costFunction,  
                    initialTheta, options)
```

Consider gradient descent

Set initialTheta = zeros(n,1) ?

## Zero initialization



$$\Rightarrow \Theta_{ij}^{(l)} = 0 \text{ for all } i, j, l.$$

Also  $\delta_i^{(l)} = \delta_j^{(l)}$ .

$$\frac{\partial}{\partial \Theta_{01}^{(l)}} J(\Theta) = \frac{\partial}{\partial \Theta_{02}^{(l)}} J(\Theta)$$

$$\underline{\Theta_{01}^{(l)}} = \underline{\Theta_{02}^{(l)}}$$

After each update, parameters corresponding to inputs going into each of two hidden units are identical.

$$\underline{\underline{\Theta_{01}^{(l)}}} = \underline{\underline{\Theta_{02}^{(l)}}}$$

## Random initialization: Symmetry breaking

→ Initialize each  $\Theta_{ij}^{(l)}$  to a random value in  $[-\epsilon, \epsilon]$   
(i.e.  $-\epsilon \leq \Theta_{ij}^{(l)} \leq \epsilon$ )

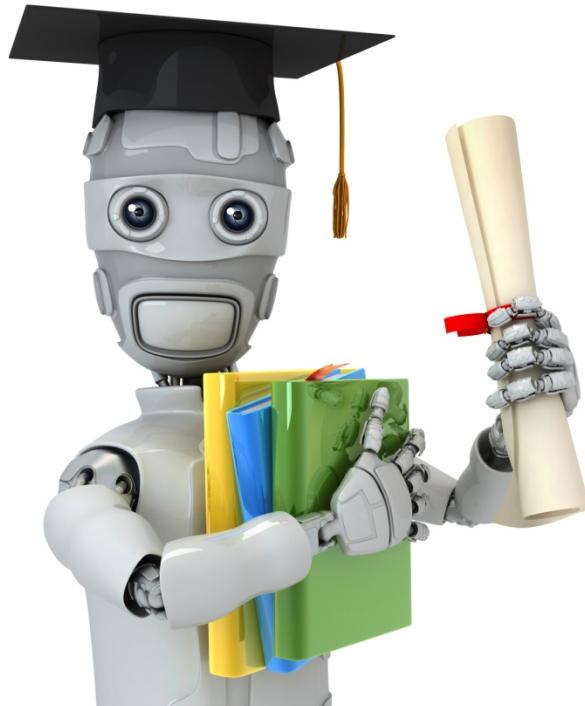
E.g.

Random  $10 \times 11$  matrix (betw. 0 and 1)

→ Theta1 = rand(10,11) \* (2\*INIT\_EPSILON)  
- INIT\_EPSILON;

$[-\epsilon, \epsilon]$

→ Theta2 = rand(1,11) \* (2\*INIT\_EPSILON)  
- INIT\_EPSILON;



Machine Learning

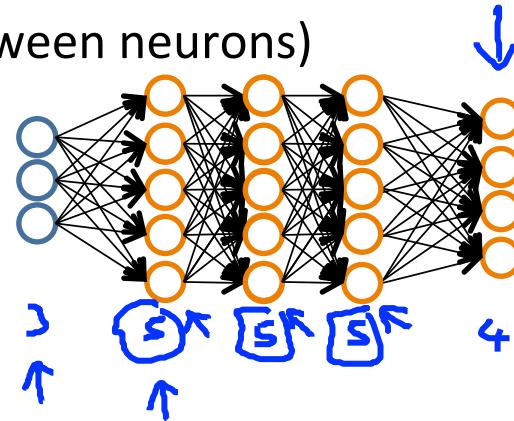
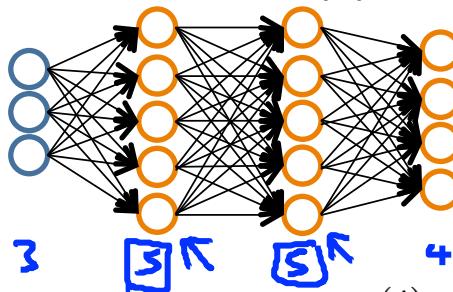
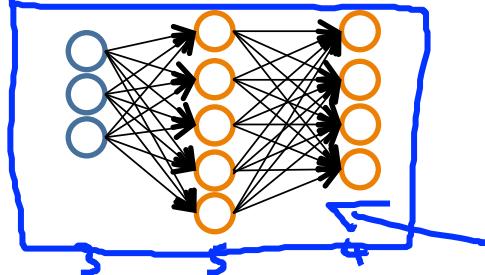
# Neural Networks: Learning

---

# Putting it together

## Training a neural network

Pick a network architecture (connectivity pattern between neurons)



→ No. of input units: Dimension of features  $x^{(i)}$

→ No. output units: Number of classes

[Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)]

$$y \in \{1, 2, 3, \dots, 10\}$$

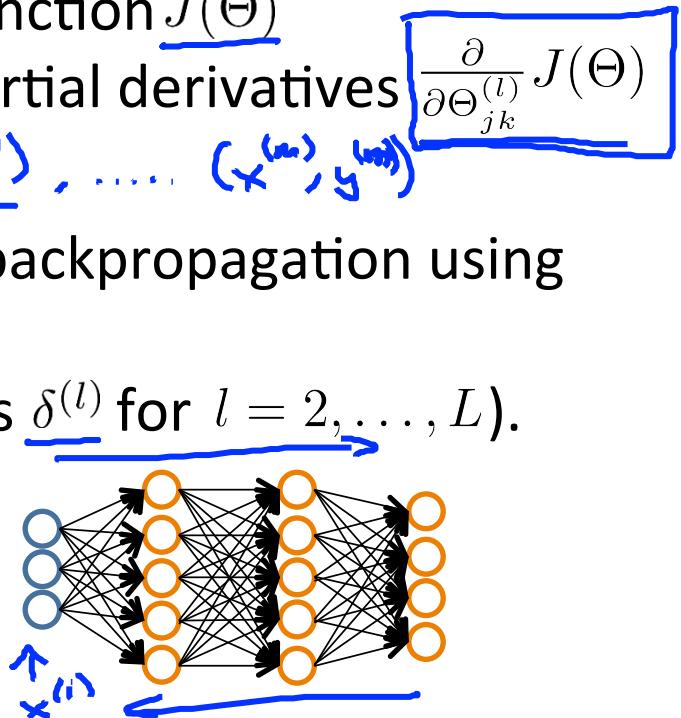
~~$y \in S$~~

$$y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \text{ or } \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

$\leftarrow$

# Training a neural network

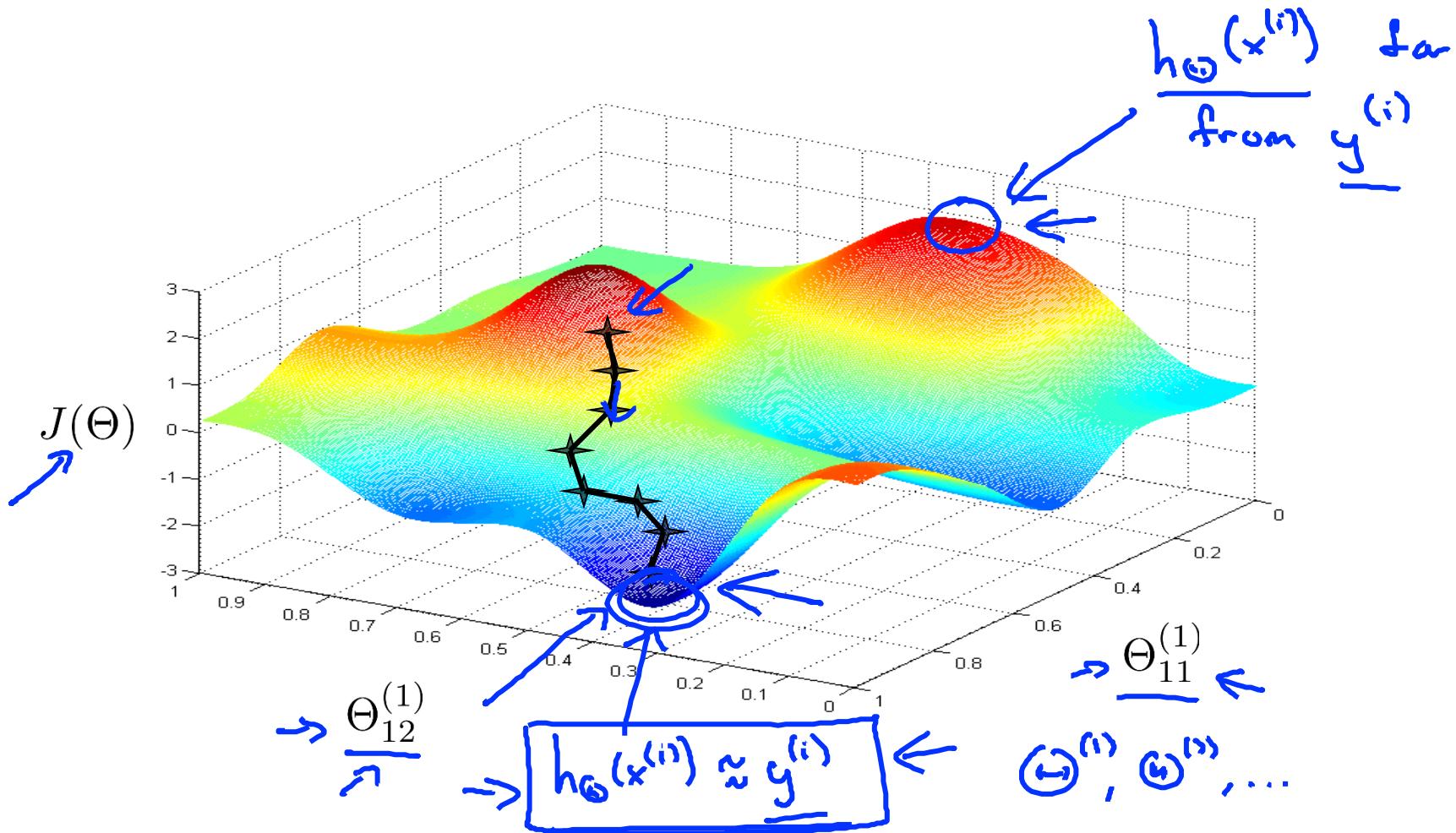
- 1. Randomly initialize weights
- 2. Implement forward propagation to get  $h_{\Theta}(x^{(i)})$  for any  $\underline{x}^{(i)}$
- 3. Implement code to compute cost function  $J(\Theta)$
- 4. Implement backprop to compute partial derivatives  $\frac{\partial}{\partial \Theta_j^{(l)}} J(\Theta)$
- for  $i = 1:m$  {  $(\underline{x}^{(1)}, y^{(1)})$      $(\underline{x}^{(2)}, y^{(2)})$  , ....  $(\underline{x}^{(m)}, y^{(m)})$  }
  - Perform forward propagation and backpropagation using example  $(x^{(i)}, y^{(i)})$
  - (Get activations  $a^{(l)}$  and delta terms  $\delta^{(l)}$  for  $l = 2, \dots, L$ .)
  - $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l)} (a^{(l)})^T$
  - ...  
}
  - Compute  $\frac{\Delta \Theta^{(l)}}{m} J(\Theta)$ .

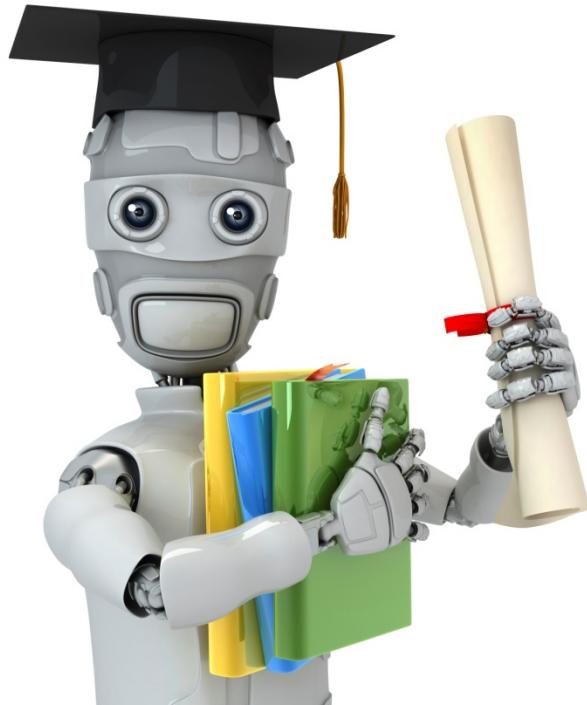


## Training a neural network

- 5. Use gradient checking to compare  $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$  computed using backpropagation vs. using numerical estimate of gradient of  $J(\Theta)$ .
  - Then disable gradient checking code.
- 6. Use gradient descent or advanced optimization method with backpropagation to try to minimize  $J(\Theta)$  as a function of parameters  $\Theta$

$$\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta) \quad \text{non-convex.}$$



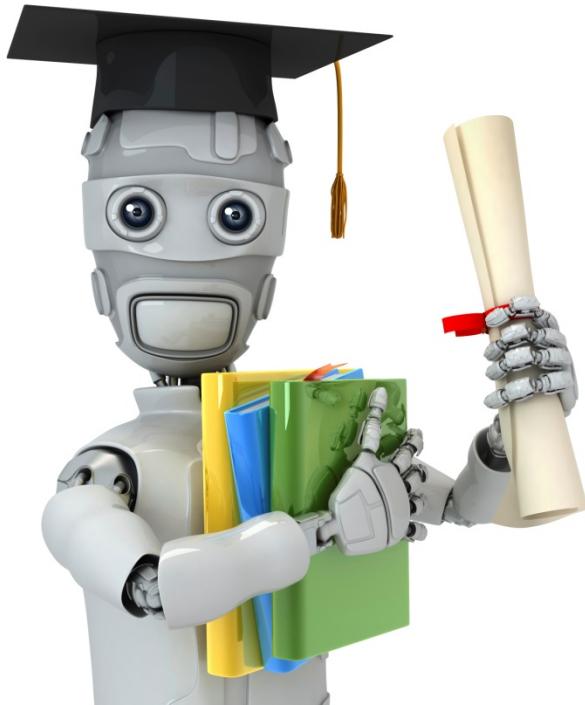


Machine Learning

# Neural Networks: Learning

---

Backpropagation  
example: Autonomous  
driving (optional)



Machine Learning

Advice for applying  
machine learning

---

Deciding what  
to try next

## Debugging a learning algorithm:

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

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

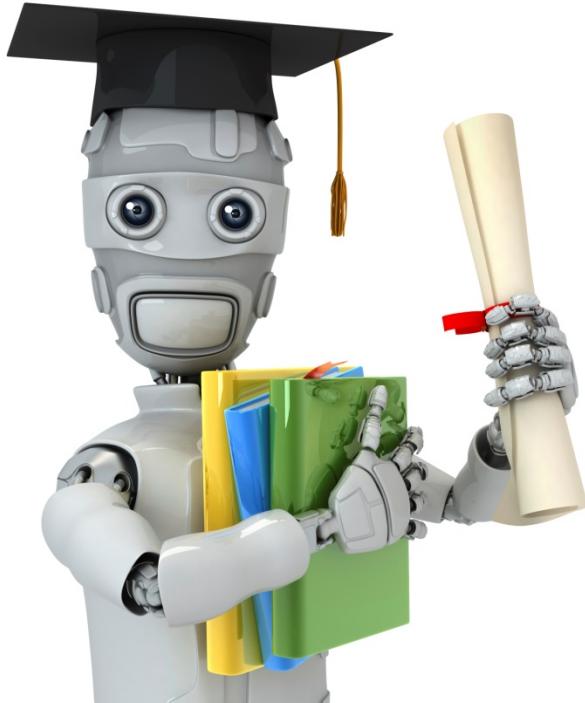
However, when you test your hypothesis on a new set of houses, you find that it makes unacceptably large errors in its predictions. What should you try next?

- - Get more training examples
- Try smaller sets of features  $x_1, x_2, x_3, \dots, x_{100}$
- - Try getting additional features
- Try adding polynomial features  $(x_1^2, x_2^2, x_1 x_2, \text{etc.})$
- Try decreasing  $\lambda$
- Try increasing  $\lambda$

## Machine learning diagnostic:

Diagnostic: 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.



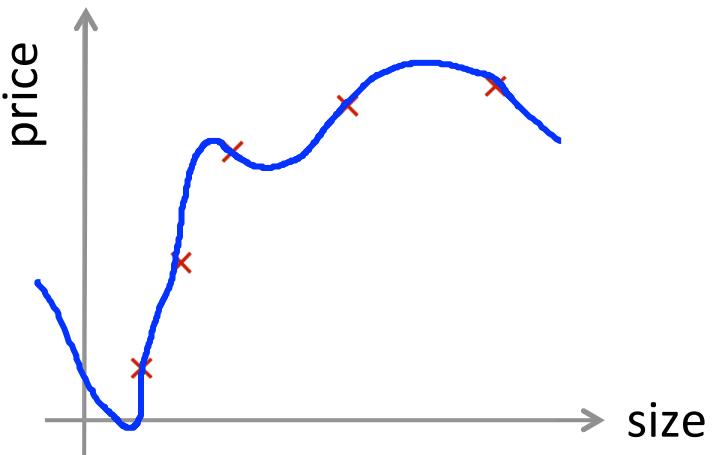
Machine Learning

Advice for applying  
machine learning

---

Evaluating a  
hypothesis

# Evaluating your hypothesis



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

Fails to generalize to new examples not in training set.

- $x_1$  = size of house
- $x_2$  = no. of bedrooms
- $x_3$  = no. of floors
- $x_4$  = age of house
- $x_5$  = average income in neighborhood
- $x_6$  = kitchen size
- :
- :
- $x_{100}$

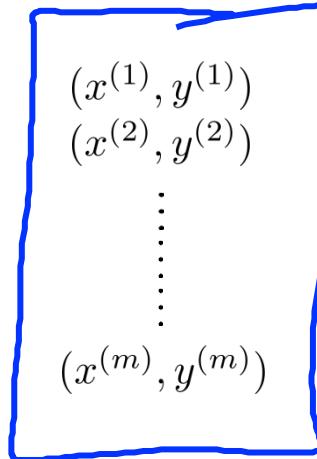
# Evaluating your hypothesis

Dataset:

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

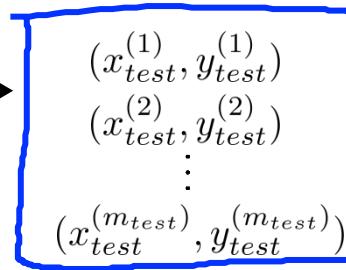
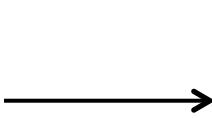
70%

Training set



30%

Test set



$m_{test}$  = no. of test example  
 $(x_{test}^{(1)}, y_{test}^{(1)})$

# Training/testing procedure for linear regression

- - Learn parameter  $\underline{\theta}$  from training data (minimizing training error  $J(\theta)$ ) 70%
- Compute test set error:

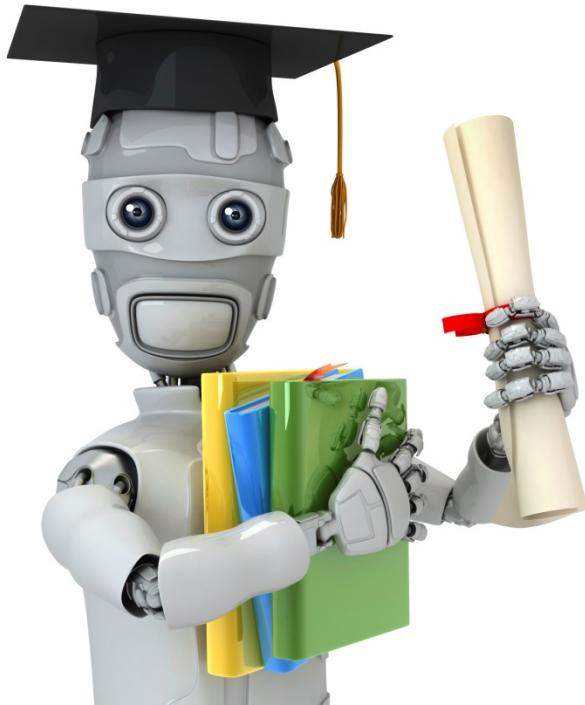
$$J_{\text{test}}(\theta) = \frac{1}{2m_{\text{test}}} \sum_{i=1}^{m_{\text{test}}} \left( h_{\theta}(x_{\text{test}}^{(i)}) - y_{\text{test}}^{(i)} \right)^2$$

## Training/testing procedure for logistic regression

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

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

- Misclassification error (0/1 misclassification error):



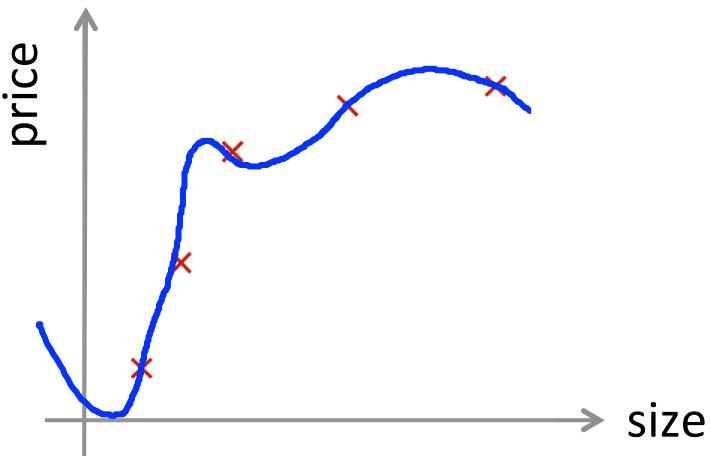
Machine Learning

# Advice for applying machine learning

---

Model selection and  
training/validation/test  
sets

## Overfitting example



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

Once parameters  $\theta_0, \theta_1, \dots, \theta_4$  were fit to some set of data (training set), the error of the parameters as measured on that data (the training error  $J(\theta)$ ) is likely to be lower than the actual generalization error.

## Model selection

$$d=1 \quad 1. \quad h_{\theta}(x) = \theta_0 + \theta_1 x$$

$$\rightarrow \Theta^{(1)} \rightarrow J_{test}(\Theta^{(1)})$$

$$d=2 \quad 2. \quad h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

$$\rightarrow \Theta^{(2)} \rightarrow J_{test}(\Theta^{(2)})$$

$$d=3 \quad 3. \quad h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_3 x^3$$

$$\rightarrow \Theta^{(3)} \rightarrow J_{test}(\Theta^{(3)})$$

⋮

⋮

$$d=10 \quad 10. \quad h_{\theta}(x) = \theta_0 + \theta_1 x + \dots + \theta_{10} x^{10}$$

$$\rightarrow \Theta^{(10)} \rightarrow J_{test}(\Theta^{(10)})$$

Choose  $\boxed{\theta_0 + \dots + \theta_5 x^5}$  ↙

How well does the model generalize? Report test set error  $\underline{J_{test}(\theta^{(5)})}$ . ↗

$\Theta^{(5)}$

$\boxed{\theta_0, \theta_1, \dots}$  ↗

Problem:  $J_{test}(\theta^{(5)})$  is likely to be an optimistic estimate of generalization error. I.e. our extra parameter (d = degree of polynomial) is fit to test set.

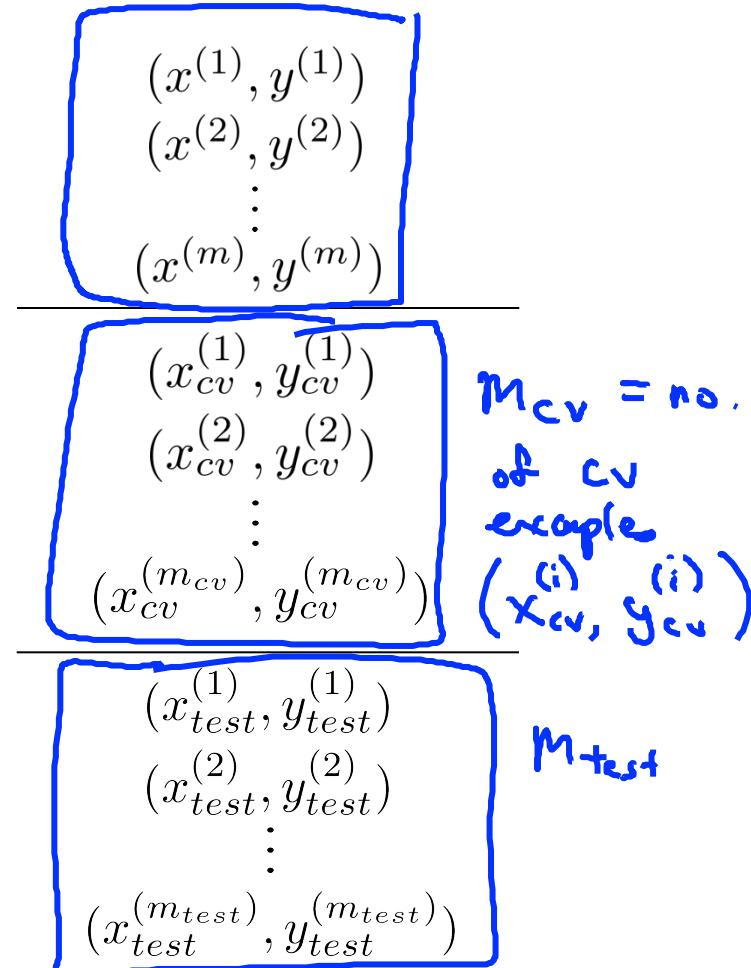
# Evaluating your hypothesis

Dataset:

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

Annotations:

- A blue curly brace groups the first six rows (2104, 1600, 2400, 1416, 3000, 1985) and is labeled "Training set".
- A blue curly brace groups the next two rows (1534, 1427) and is labeled "Cross validation (CV)".
- A blue curly brace groups the last two rows (1380, 1494) and is labeled "test set".
- A large black arrow points from the "Training set" to the top box containing  $(x^{(1)}, y^{(1)})$  through  $(x^{(m)}, y^{(m)})$ .
- A large black arrow points from the "Cross validation (CV)" to the middle box containing  $(x_{cv}^{(1)}, y_{cv}^{(1)})$  through  $(x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$ .
- A large black arrow points from the "test set" to the bottom box containing  $(x_{test}^{(1)}, y_{test}^{(1)})$  through  $(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$ .



# Train/validation/test error

Training error:

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

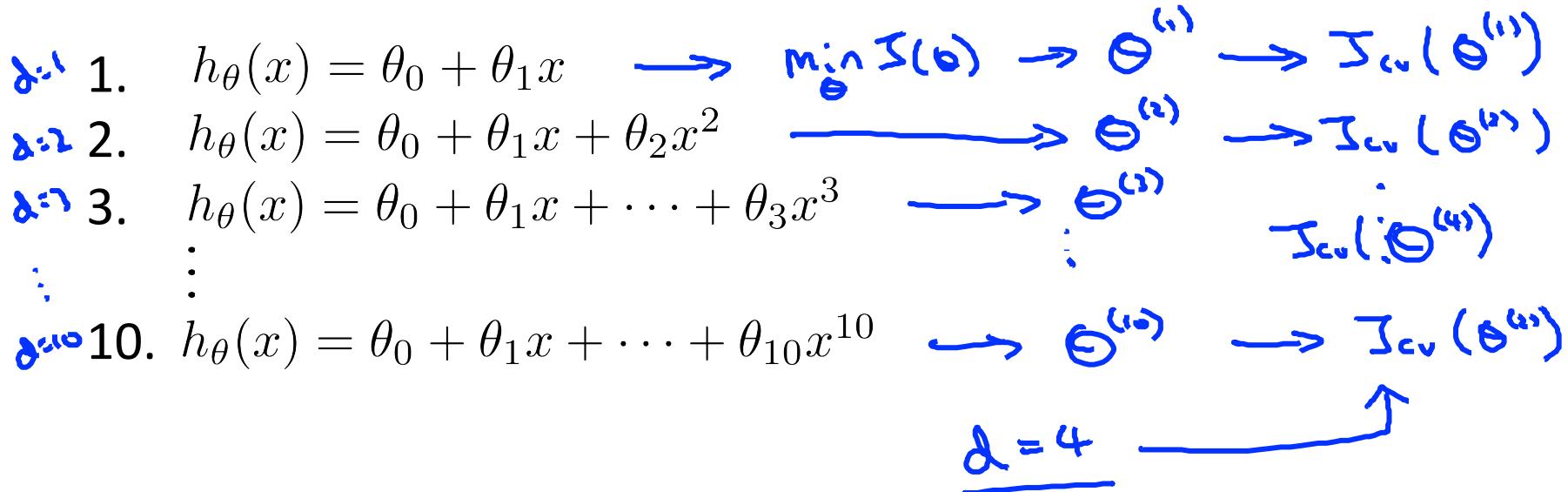
Cross Validation error:

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

Test error:

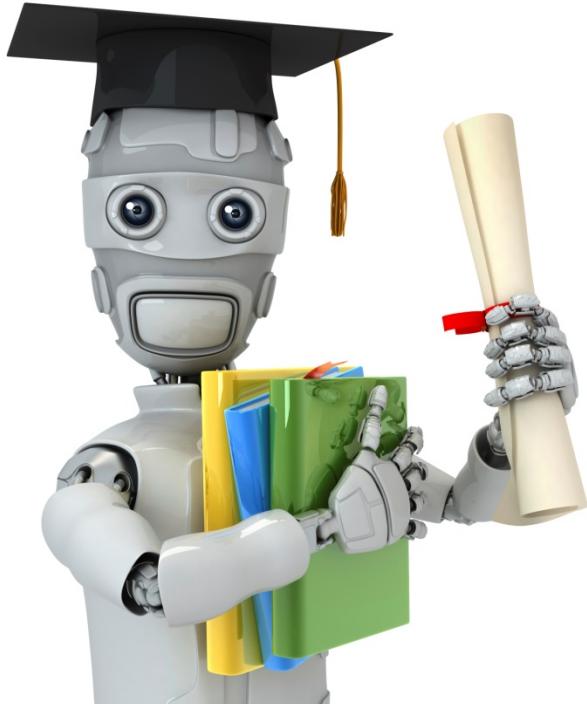
$$\rightarrow J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

## Model selection



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

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



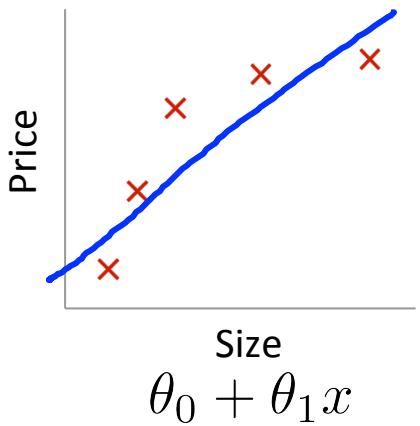
Machine Learning

# Advice for applying machine learning

---

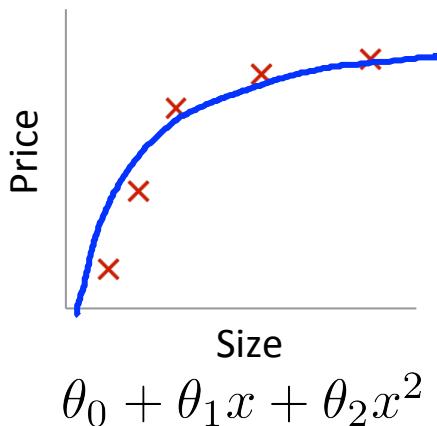
## Diagnosing bias vs. variance

# Bias/variance

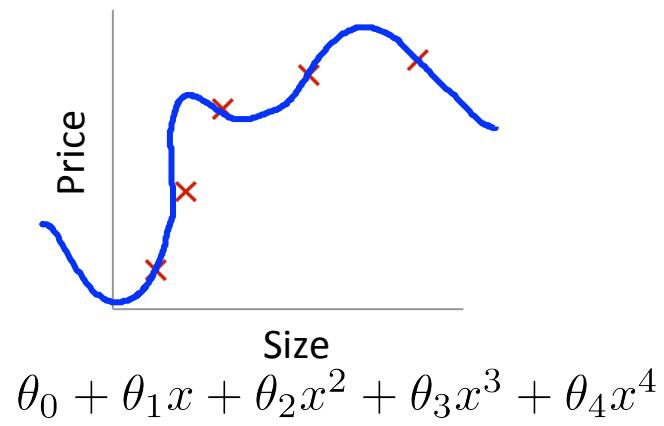


High bias  
(underfit)

$$d=1$$



“Just right”  
 $d=2$



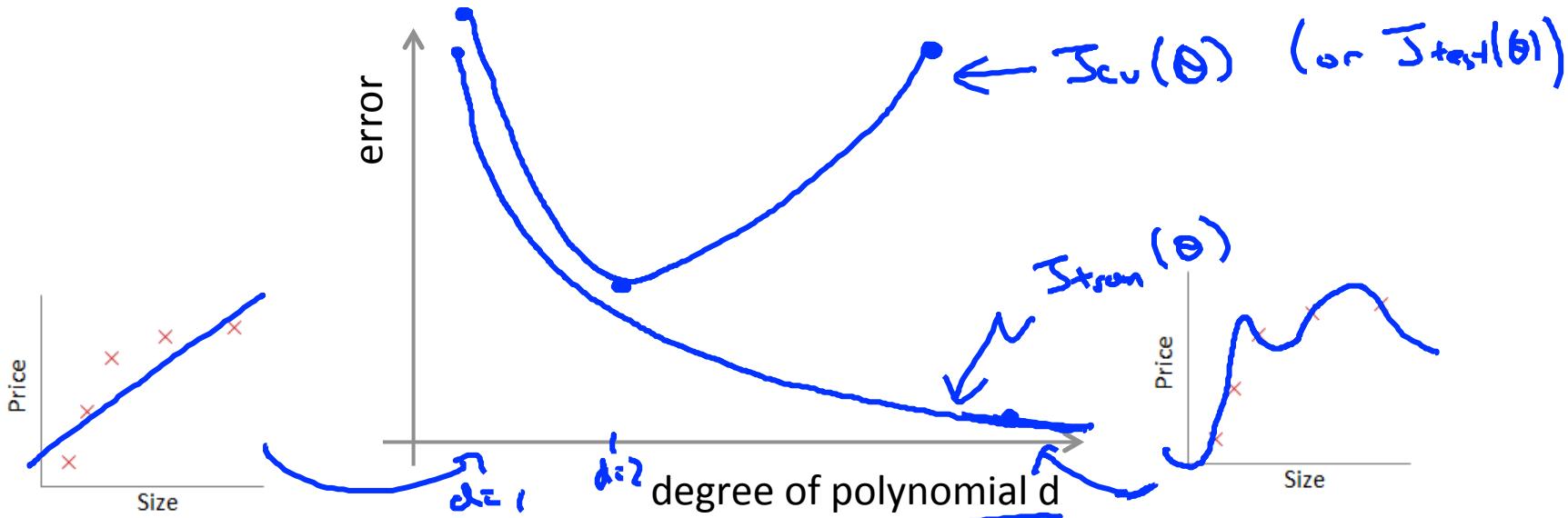
High variance  
(overfit)

$$d=4$$

# Bias/variance

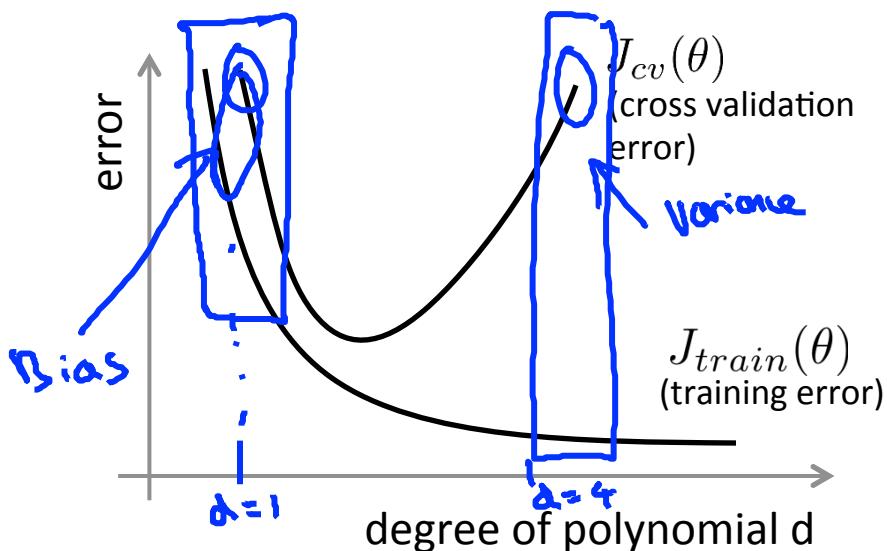
Training error:  $\underline{J_{train}(\theta)} = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$

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



## 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):

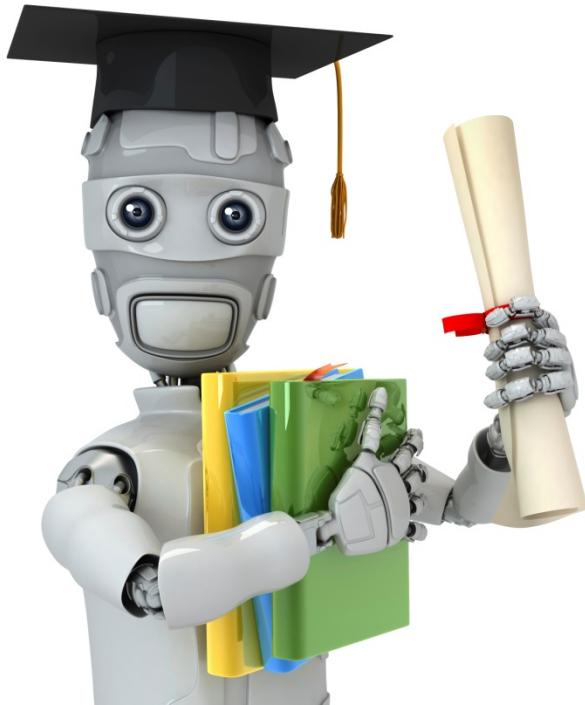
$\rightarrow J_{train}(\theta)$  will be high }  
 $J_{cv}(\theta) \approx J_{train}(\theta)$

Variance (overfit):

$\rightarrow J_{train}(\theta)$  will be low }

$J_{cv}(\theta) \gg J_{train}(\theta)$

>>



Machine Learning

# Advice for applying machine learning

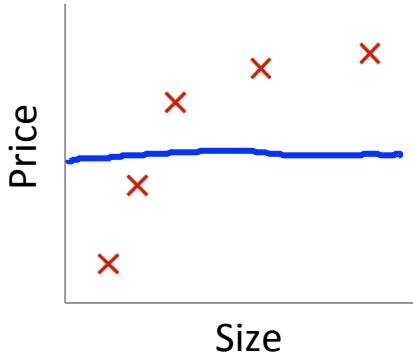
---

## Regularization and bias/variance

# Linear regression with regularization

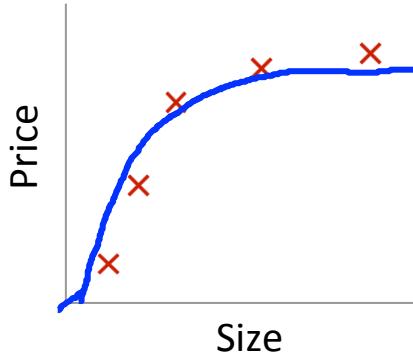
Model: 
$$h_{\theta}(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

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

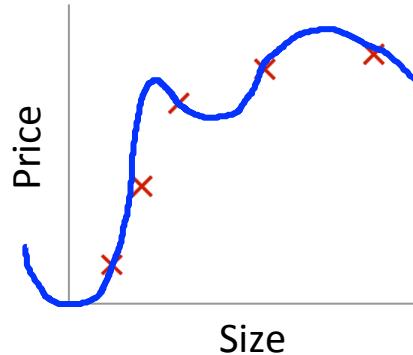


Large  $\lambda$  ←

→ High bias (underfit)  
→  $\lambda = 10000$ .  $\theta_1 \approx 0, \theta_2 \approx 0, \dots$   
 $h_{\theta}(x) \approx \theta_0$



Intermediate  $\lambda$  ←  
“Just right”



→ Small  $\lambda$   
High variance (overfit)  
→  $\lambda = 0$

## Choosing the regularization parameter $\lambda$

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

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

$$\rightarrow J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 \quad \underbrace{\qquad\qquad\qquad}_{J(\theta)}$$
$$J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_{\theta}(x_{cv}^{(i)}) - y_{cv}^{(i)})^2 \quad \begin{matrix} J_{train} \\ J_{cv} \\ J_{test} \end{matrix}$$
$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_{\theta}(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

## Choosing the regularization parameter $\lambda$

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

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

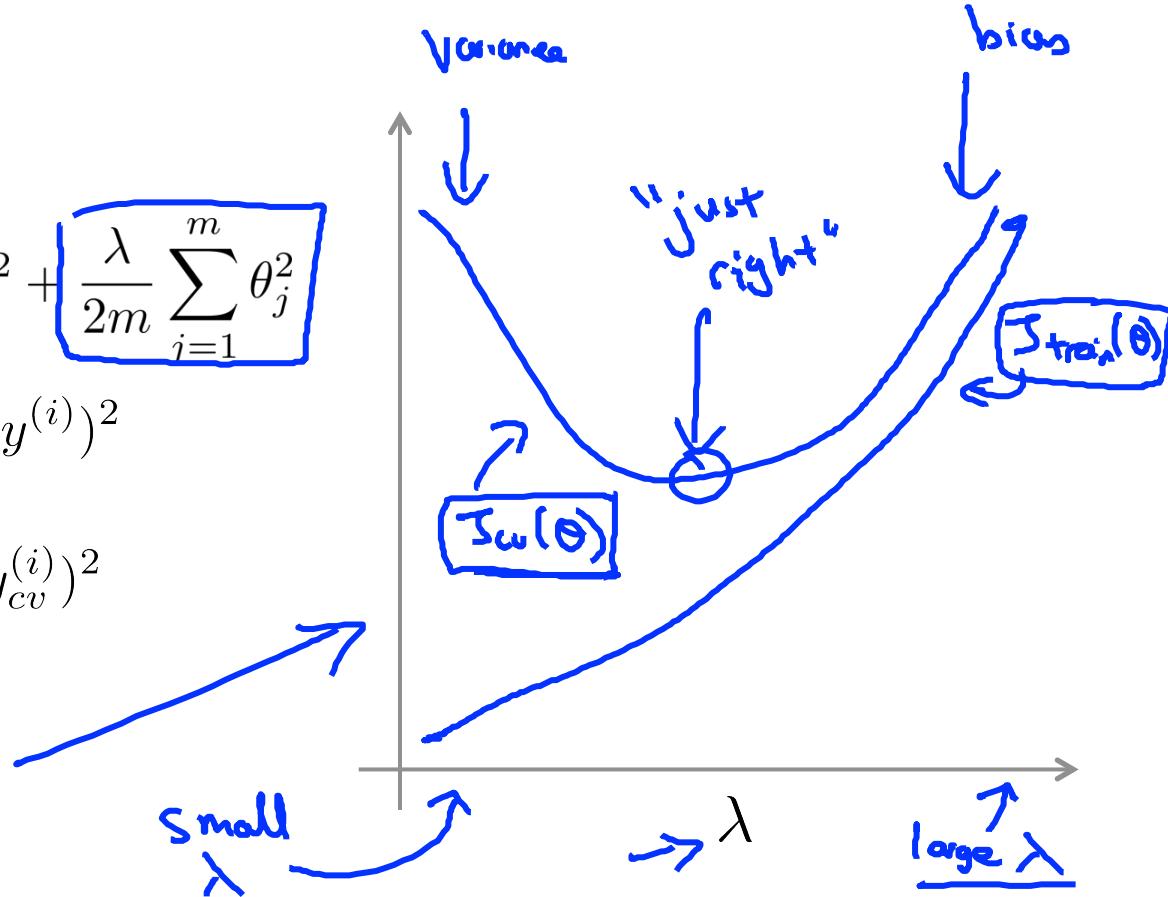
1. Try  $\lambda = 0$    $\rightarrow \min_{\theta} J(\theta) \rightarrow \theta^{(0)} \rightarrow J_{cv}(\theta^{(0)})$
  2. Try  $\lambda = 0.01$    $\rightarrow \min_{\theta} J(\theta) \rightarrow \theta^{(1)} \rightarrow J_{cv}(\theta^{(1)})$
  3. Try  $\lambda = 0.02$    $\rightarrow \theta^{(2)} \rightarrow J_{cv}(\theta^{(2)})$
  4. Try  $\lambda = 0.04$  
  5. Try  $\lambda = 0.08$    $\vdots \rightarrow \theta^{(5)} \rightarrow J_{cv}(\theta^{(5)})$
  - ⋮
  12. Try  $\lambda = 10$    $\rightarrow \theta^{(12)} \rightarrow J_{cv}(\theta^{(12)})$
- Pick (say)  $\theta^{(5)}$ . Test error:  $J_{test}(\theta^{(5)})$

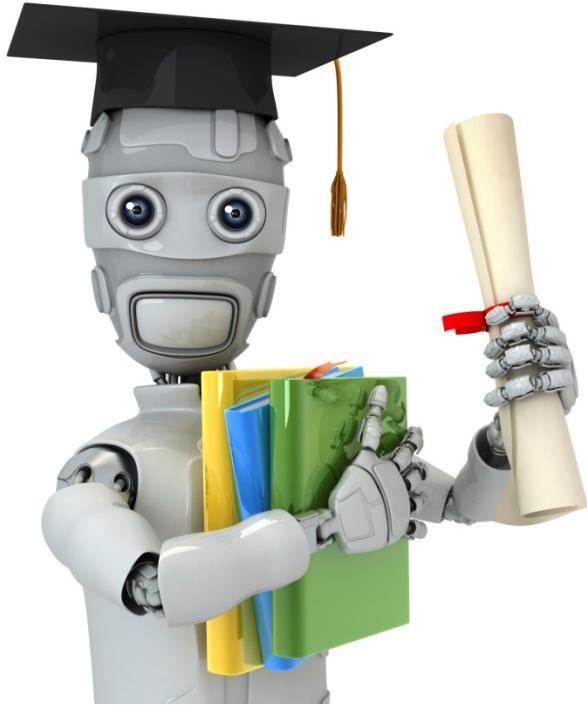
# Bias/variance as a function of the regularization parameter $\lambda$

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

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

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





Machine Learning

# Advice for applying machine learning

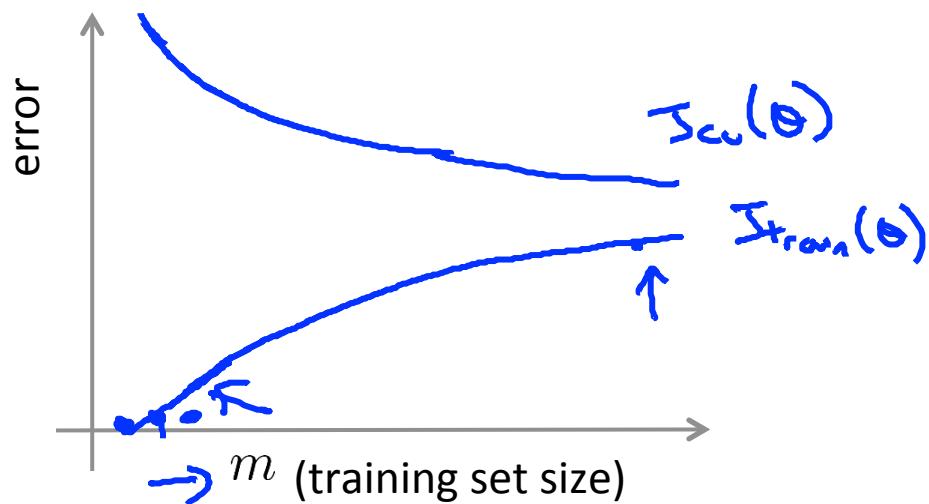
---

## Learning curves

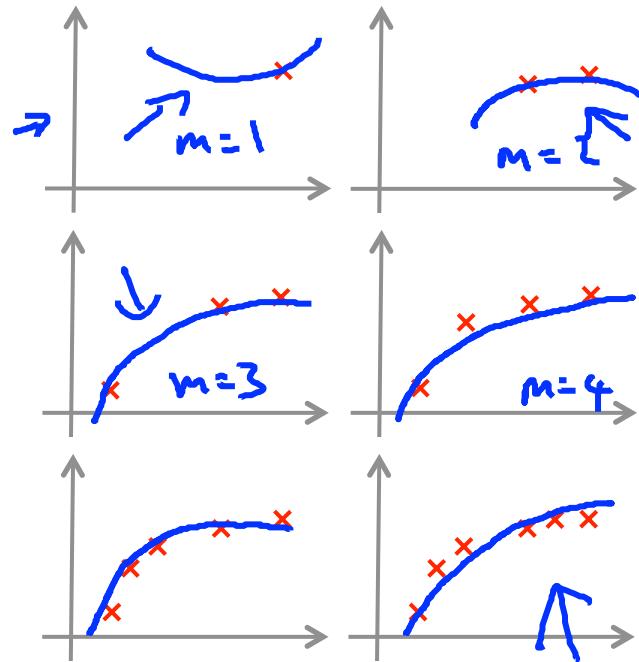
## Learning curves

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

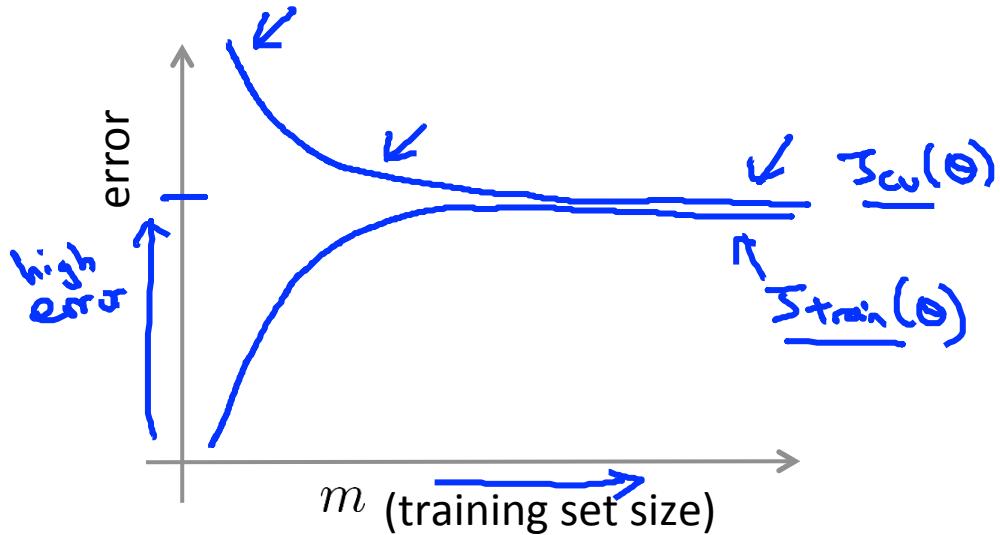
$$\rightarrow J_{cv}(\theta) = \frac{1}{2m_{cv}} \sum_{i=1}^{m_{cv}} (h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$



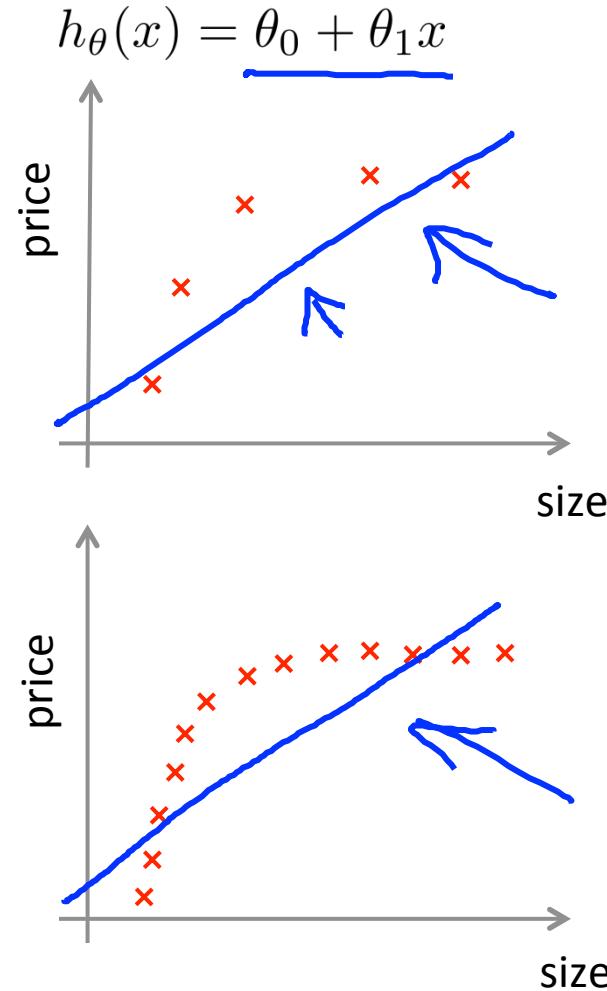
$$h_\theta(x) = \underline{\theta_0 + \theta_1 x + \theta_2 x^2}$$



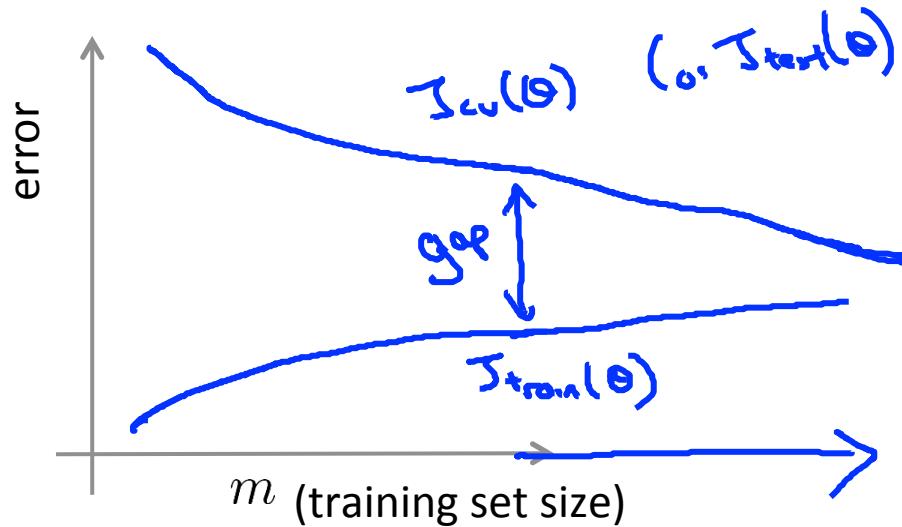
## High bias



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



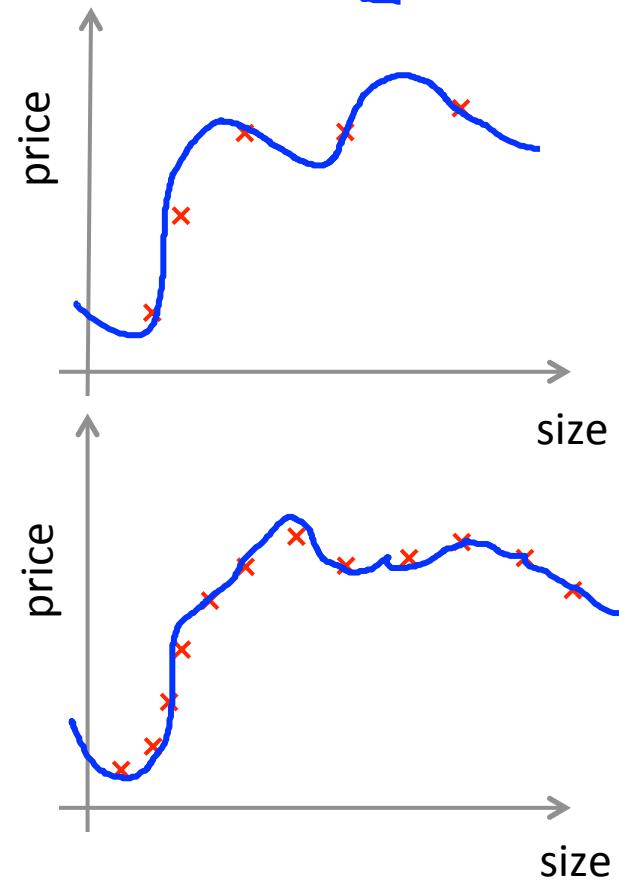
## High variance

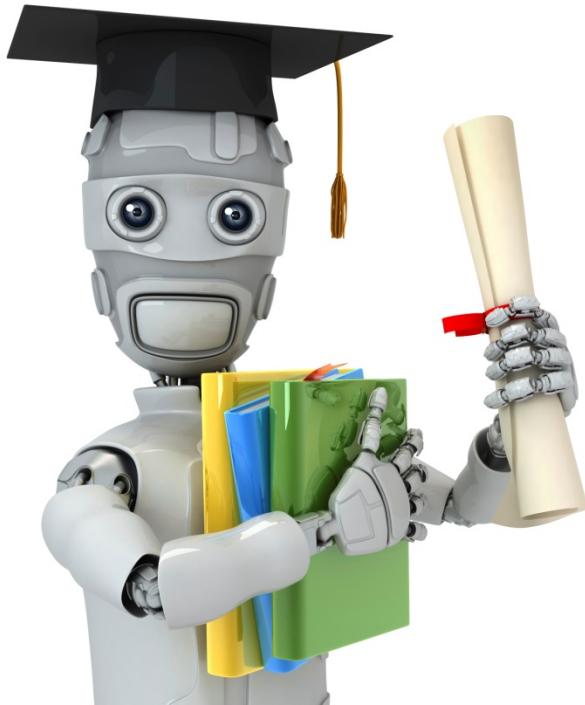


If a learning algorithm is suffering from high variance, getting more training data is likely to help. ↫

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

(and small  $\lambda$ ) ↗





Machine Learning

## Advice for applying machine learning

---

### Deciding what to try next (revisited)

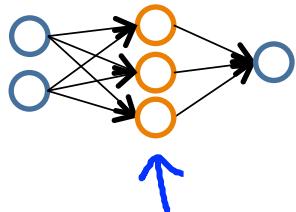
## Debugging a learning algorithm:

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

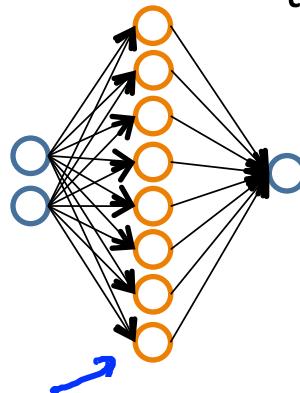
# Neural networks and overfitting

→ “Small” neural network  
(fewer parameters; more  
prone to underfitting)



Computationally cheaper

→ “Large” neural network  
(more parameters; more prone  
to overfitting)

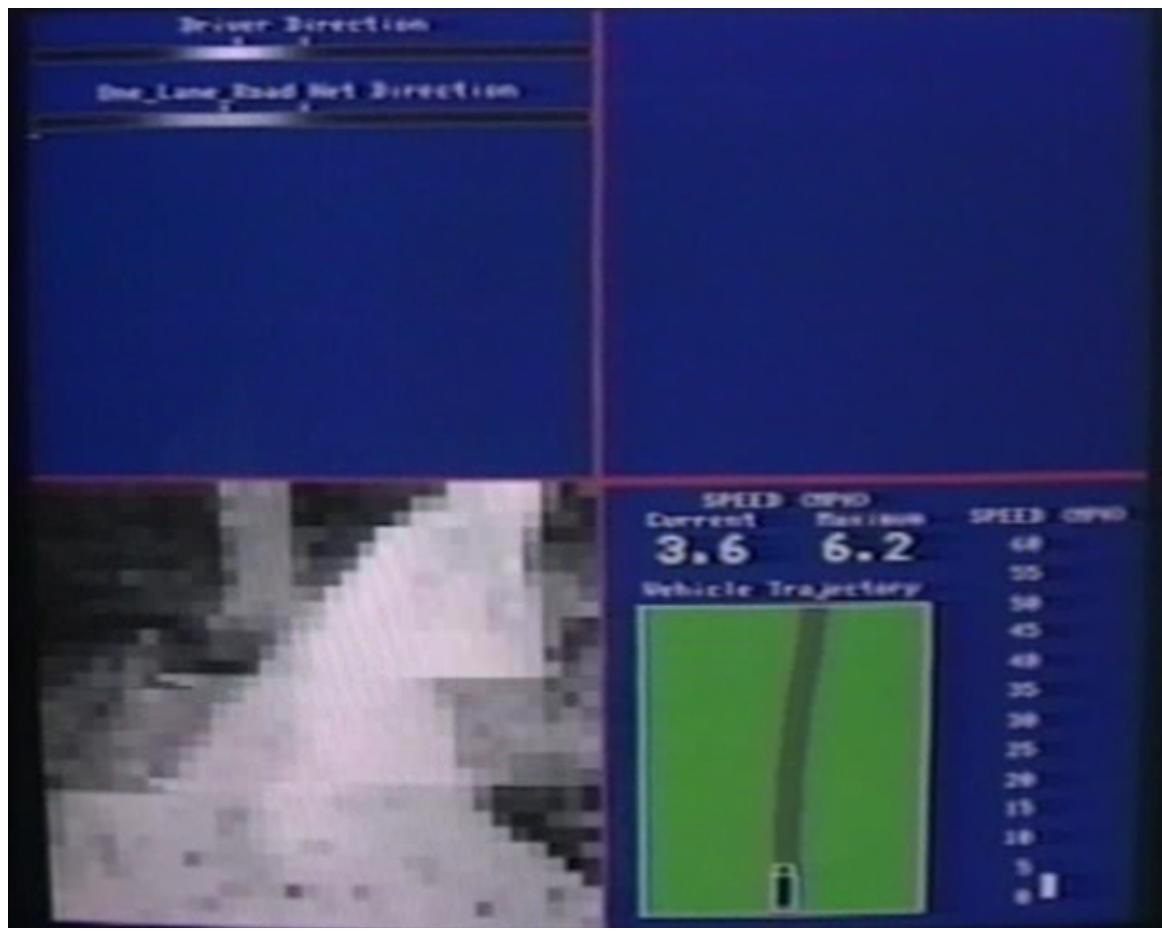


Computationally more expensive.

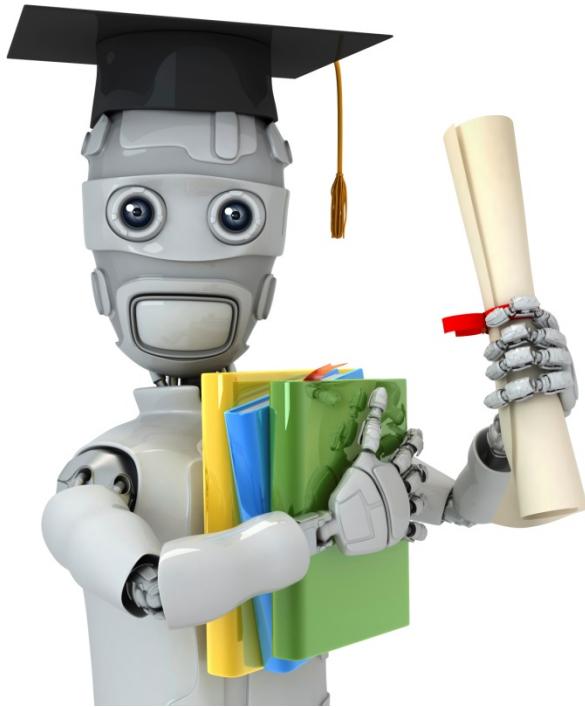
Use regularization ( $\lambda$ ) to address overfitting.

$$\mathcal{J}_{\text{reg}}(\Theta)$$





[Courtesy of Dean Pomerleau]



Machine Learning

# Machine learning system design

---

Prioritizing what to  
work on: Spam  
classification example

# Building a spam classifier

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

Deal of the week! Buy now!  
Rolex w4tchs - \$100  
Medcine (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)

## 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.

E.g. deal, buy, discount, andrew, now, ...

$$x = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 1 \end{bmatrix} \quad \begin{array}{l} \text{andrew} \\ \text{buy} \\ \text{deal} \\ \text{discount} \\ \vdots \\ \text{now} \end{array} \quad x \in \mathbb{R}^{100}$$

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

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

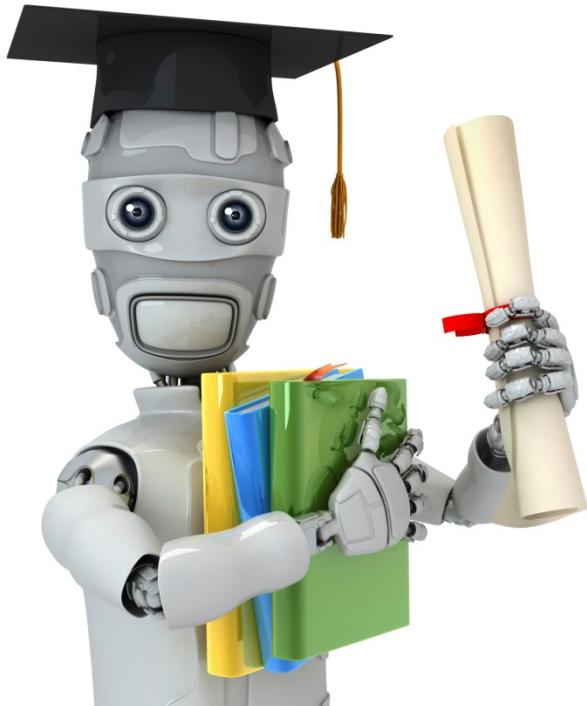
Deal of the week! Buy now!

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

## 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.)



Machine Learning

Machine learning  
system design

---

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

→ Deliberate misspellings: 5

Replica/fake: 4

→ (m0rgage, med1cine, etc.)

→ Steal passwords: 53

→ Unusual email routing: 16

Other: 31

→ Unusual (spamming) punctuation: 32

## The importance of numerical evaluation

Should discount/discounts/discounted/discounting be treated as the same word?

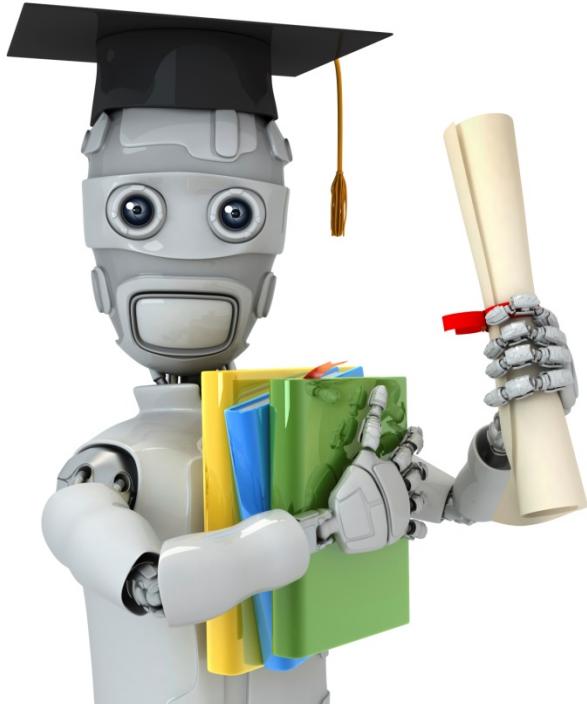
Can use “stemming” software (E.g. “Porter stemmer”)  
universe/university.

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%



Machine Learning

# Machine learning system design

---

## Error metrics for skewed classes

## 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% acc way (0.8% error)

→ 99.5% acc way (0.5% error)

# Precision/Recall

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

Actual class		
	1	
1	True positive	False positive
0	False negative	True negative

$$y=0 \\ \text{recall} = 0$$

→ Precision

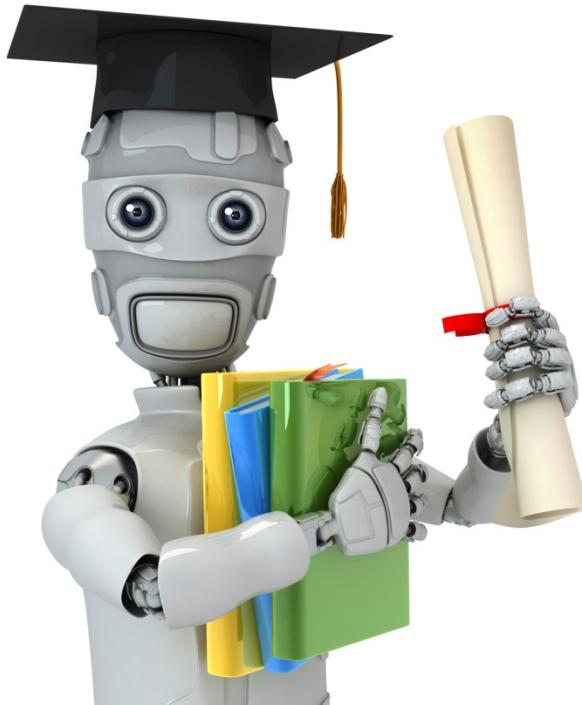
(Of all patients where we predicted  $y = 1$ , what fraction actually has cancer?)

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

→ Recall

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

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



Machine Learning

# Machine learning system design

---

Trading off precision  
and recall

## Trading off precision and recall

→ Logistic regression:  $0 \leq h_{\theta}(x) \leq 1$

Predict 1 if  $h_{\theta}(x) \geq 0.5$  ~~0.7~~ ~~0.9~~ ~~0.3~~  $\leftarrow$

Predict 0 if  $h_{\theta}(x) < 0.5$  ~~0.7~~ ~~0.9~~  $\leftarrow$  0.3

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

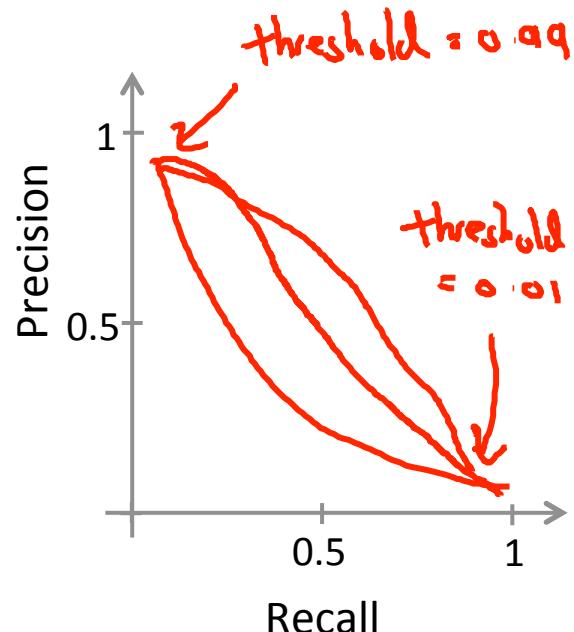
→ Higher precision, lower recall

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

→ Higher recall, lower precision.

More generally: Predict 1 if  $h_{\theta}(x) \geq \text{threshold}$   $\leftarrow$

$$\rightarrow \text{precision} = \frac{\text{true positives}}{\text{no. of predicted positive}}$$
$$\rightarrow \text{recall} = \frac{\text{true positives}}{\text{no. of actual positive}}$$



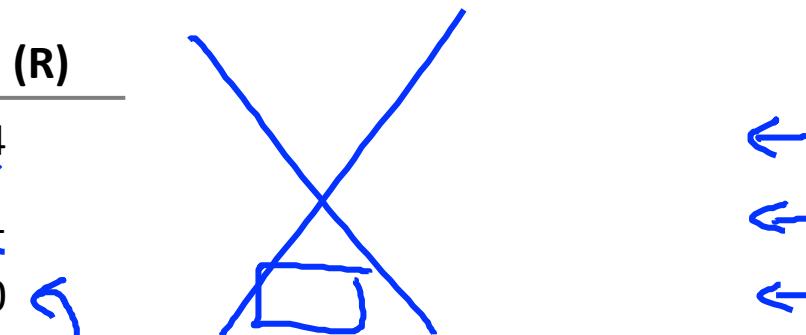
# $F_1$ Score (F score)

How to compare precision/recall numbers?

	Precision(P)	Recall (R)
Algorithm 1	0.5	0.4
Algorithm 2	0.7	0.1
Algorithm 3	0.02	1.0

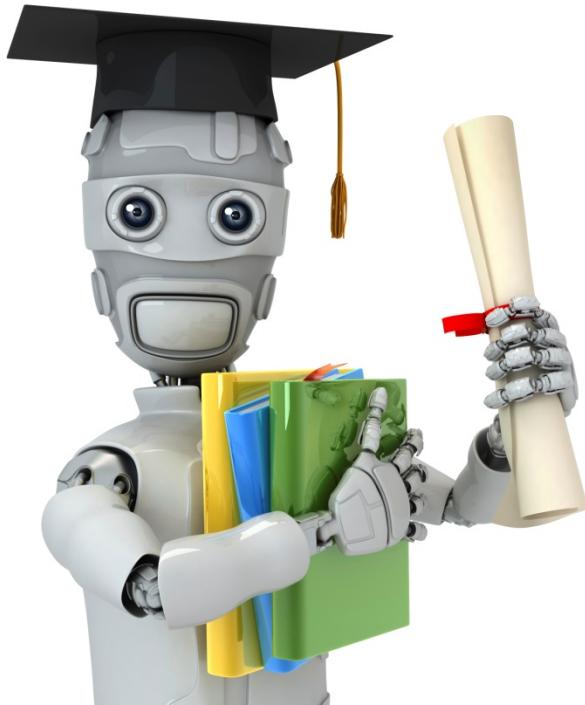
Average:  ~~$\frac{P+R}{2}$~~

$F_1$  Score:  $2 \frac{PR}{P+R}$



Predict  $y=1$  all the time

$$\begin{aligned} P=0 \quad \text{or} \quad R=0 &\Rightarrow F\text{-Score} = 0 \\ P=1 \quad \text{and} \quad R=1 &\Rightarrow F\text{-Score} = 1 \end{aligned}$$



Machine Learning

# Machine learning system design

---

## Data for machine learning

# Designing a high accuracy learning system

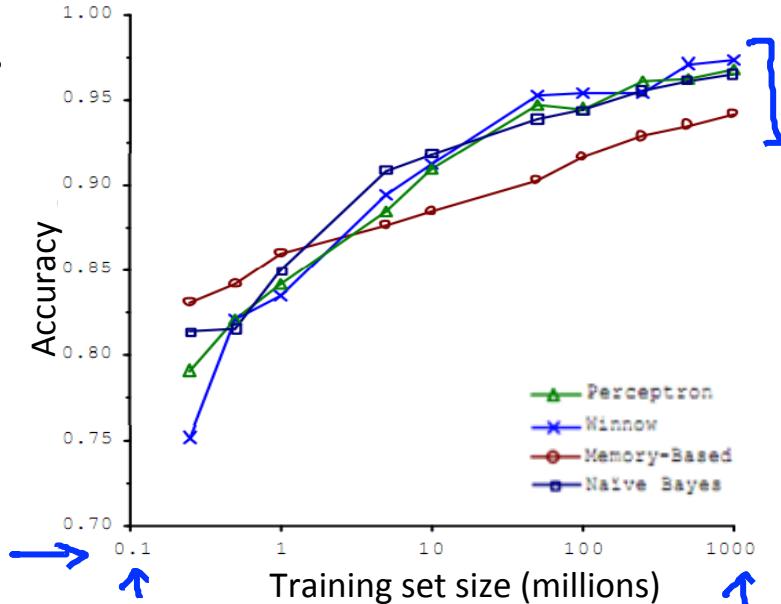
E.g. Classify between confusable words.

{to, two, too}, {then, than}

→ For breakfast I ate two eggs.

Algorithms

- - Perceptron (Logistic regression)
- - Winnow
- - Memory-based
- - Naïve Bayes



“It’s not who has the best algorithm that wins.

It’s who has the most data.”



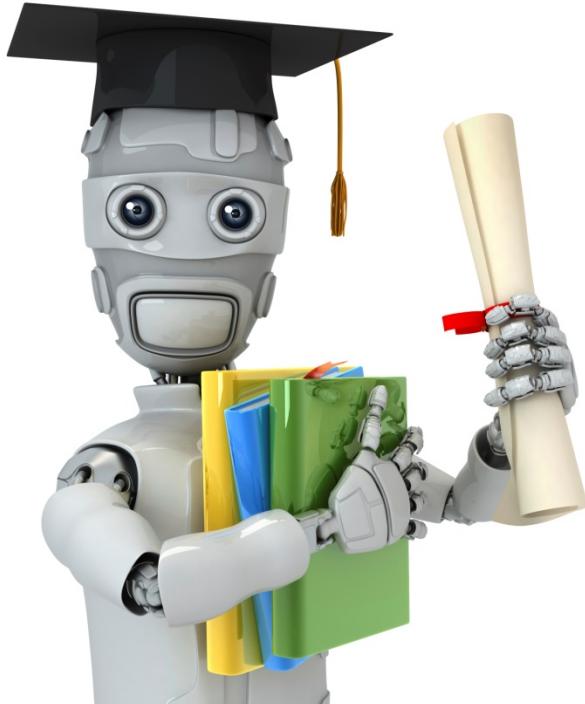
## Large data rationale

→ Assume feature  $x \in \mathbb{R}^{n+1}$  has sufficient information to predict  $y$  accurately.

Example: For breakfast I ate two eggs.

Counterexample: Predict housing price from only size (feet<sup>2</sup>) and no other features.

Useful test: Given the input  $x$ , can a human expert confidently predict  $y$ ?



Machine Learning

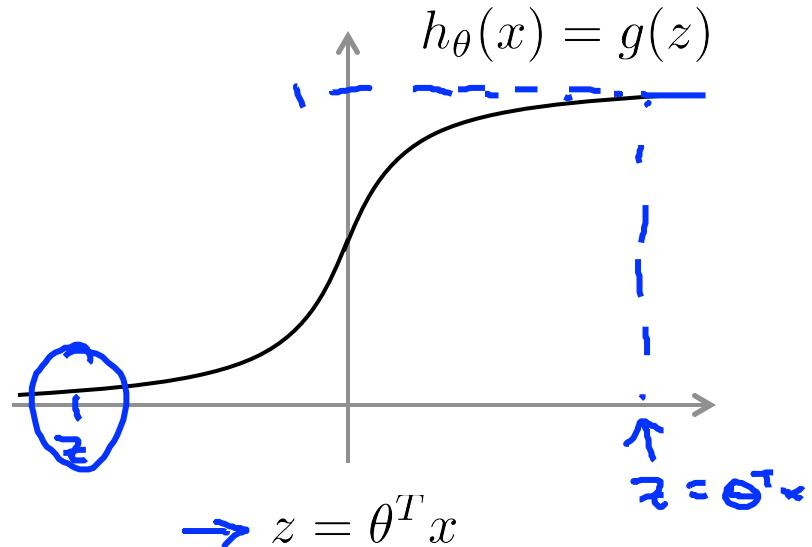
# Support Vector Machines

---

## Optimization objective

# Alternative view of logistic regression

$$\rightarrow h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$



If  $y = 1$ , we want  $h_{\theta}(x) \approx 1$      $\underline{\theta^T x \gg 0}$

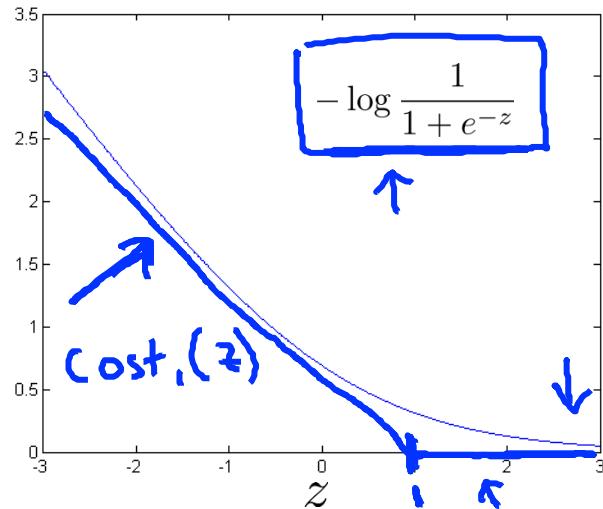
If  $y = 0$ , we want  $h_{\theta}(x) \approx 0$      $\underline{\theta^T x \ll 0}$

## Alternative view of logistic regression

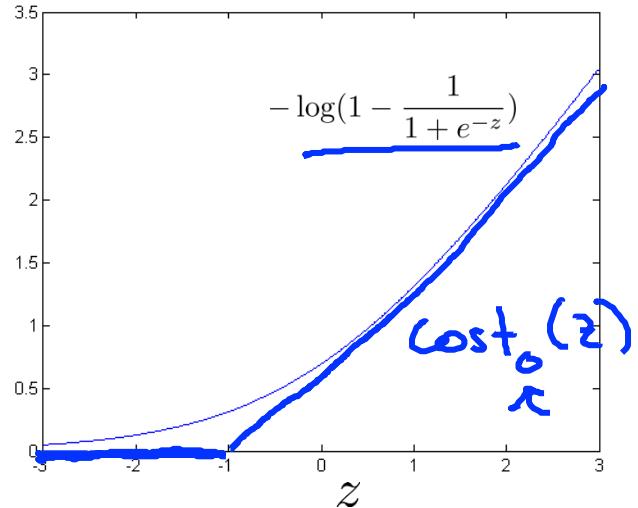
$$\text{Cost of example: } -(y \log h_{\theta}(x) + (1 - y) \log(1 - h_{\theta}(x))) \leftarrow$$

$$= -y \log \frac{1}{1 + e^{-\theta^T x}} - (1 - y) \log(1 - \frac{1}{1 + e^{-\theta^T x}}) \leftarrow$$

If  $y = 1$  (want  $\theta^T x \gg 0$ ):  
 $z = \theta^T x$



If  $y = 0$  (want  $\theta^T x \ll 0$ ):



# Support vector machine

Logistic regression:

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

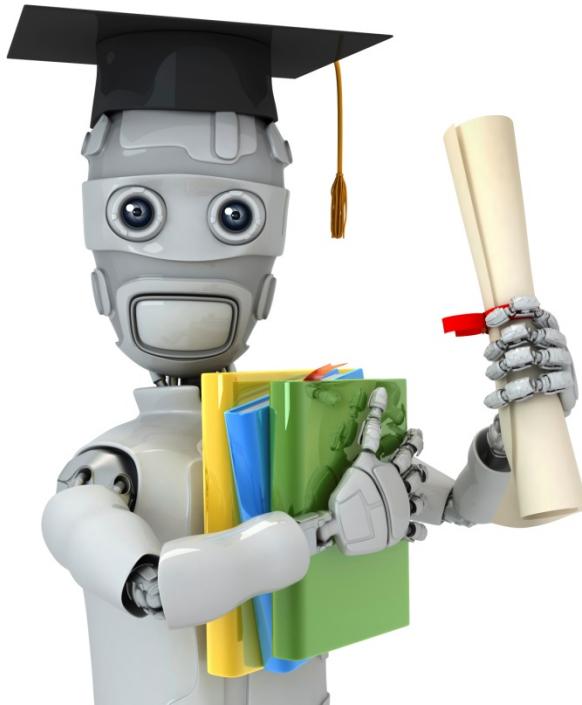
Support vector machine:

$$\min_{\theta} C \sum_{i=1}^m \left[ y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^n \theta_j^2$$

## SVM hypothesis

$$\min_{\theta} C \sum_{i=1}^m \left[ y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

Hypothesis:



Machine Learning

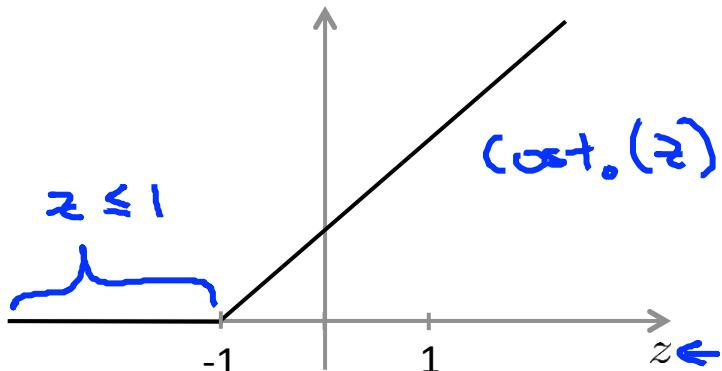
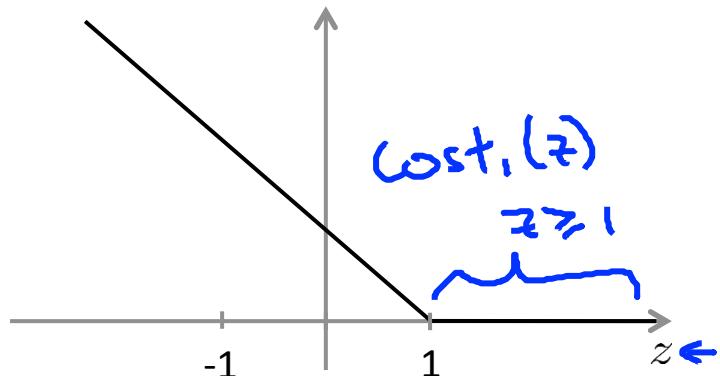
# Support Vector Machines

---

## Large Margin Intuition

# Support Vector Machine

$$\rightarrow \min_{\theta} C \sum_{i=1}^m \left[ y^{(i)} \underbrace{\text{cost}_1(\theta^T x^{(i)})}_{z \geq 1} + (1 - y^{(i)}) \underbrace{\text{cost}_0(\theta^T x^{(i)})}_{z \leq -1} \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$



→ If  $y = 1$ , we want  $\underline{\theta^T x \geq 1}$  (not just  $\geq 0$ )

$$\Theta^T x \geq \alpha \ 1$$

→ If  $y = 0$ , we want  $\underline{\theta^T x \leq -1}$  (not just  $< 0$ )

$$\Theta^T x \leq \alpha \ -1$$

$$C = 100,000$$

## SVM Decision Boundary

$$\min_{\theta} C \sum_{i=1}^m \left[ y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

Whenever  $y^{(i)} = 1$ :

$$\theta^T x^{(i)} \geq 1$$

Whenever  $y^{(i)} = 0$ :

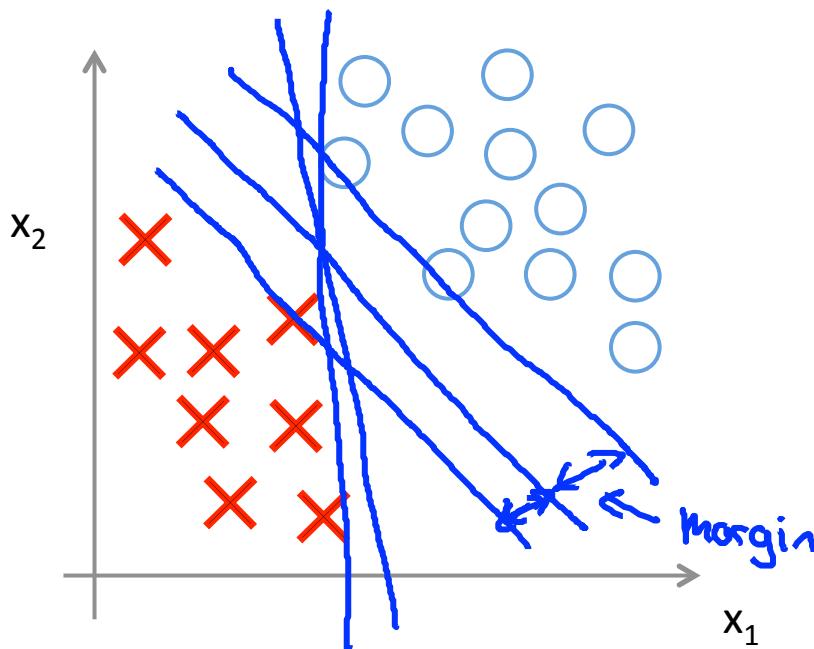
$$\theta^T x^{(i)} \leq -1$$

$$\min_{\theta} C \sum_{i=1}^m \text{cost}_1(\theta^T x^{(i)}) + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

$$\text{s.t. } \theta^T x^{(i)} \geq 1 \quad \text{if } y^{(i)} = 1$$

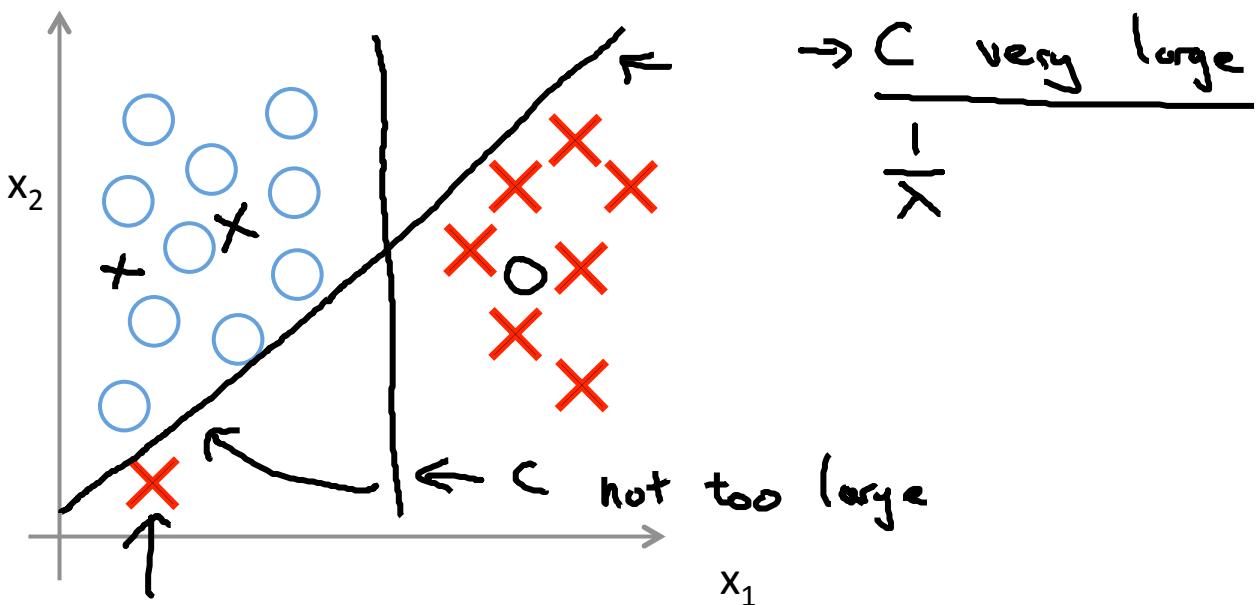
$$\theta^T x^{(i)} \leq -1 \quad \text{if } y^{(i)} = 0$$

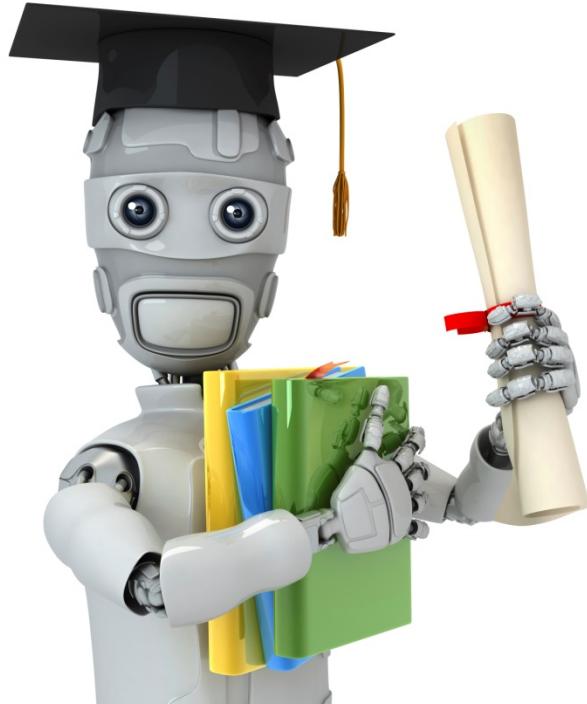
## SVM Decision Boundary: Linearly separable case



Large margin classifier

# Large margin classifier in presence of outliers





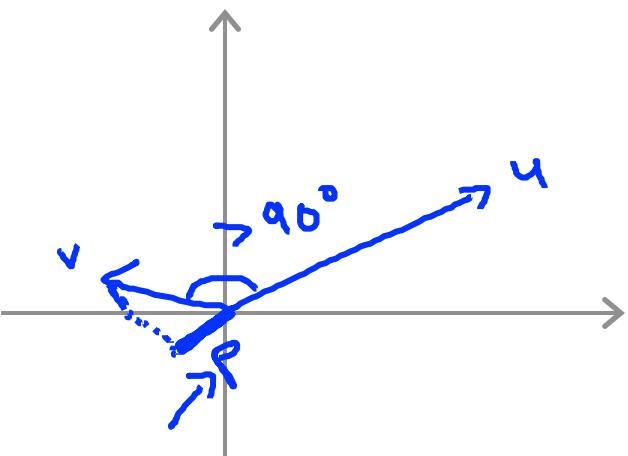
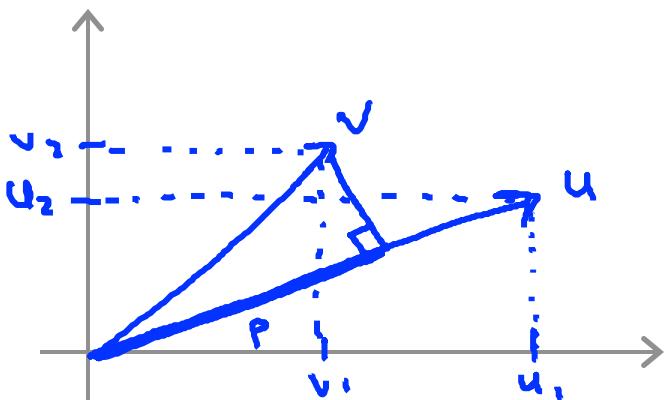
Machine Learning

# Support Vector Machines

---

The mathematics  
behind large margin  
classification (optional)

## Vector Inner Product



$$\rightarrow u = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \rightarrow v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$u^T v = ? \quad [u_1 \ u_2] \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$$\|u\| = \text{length of vector } u \\ = \sqrt{u_1^2 + u_2^2} \in \mathbb{R}$$

$p = \text{length of projection of } v \text{ onto } u.$

$$u^T v = \underline{p \cdot \|u\|} \leftarrow = v^T u$$

Signed

$$= u_1 v_1 + u_2 v_2 \leftarrow p \in \mathbb{R}$$

$$u^T v = p \cdot \|u\|$$

$$p < 0$$

$$\omega = (\sqrt{\omega})^2$$

## SVM Decision Boundary

$$\min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2 = \frac{1}{2} (\theta_1^2 + \theta_2^2) = \frac{1}{2} (\boxed{\theta_1^2 + \theta_2^2})^2 = \frac{1}{2} \|\theta\|^2$$

s.t.  $\boxed{\theta^T x^{(i)} \geq 1}$  if  $y^{(i)} = 1$

$$\rightarrow \theta^T x^{(i)} \leq -1 \quad \text{if } y^{(i)} = 0$$

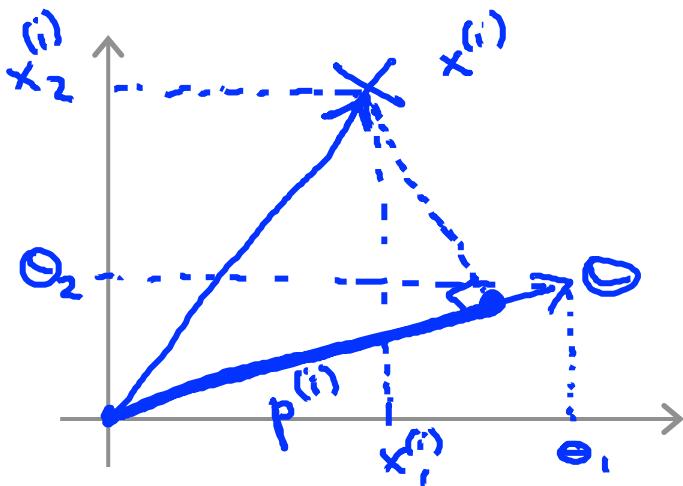
Simplification:  $\underline{\theta_0 = 0}$ .  $\underline{n=2}$

$$= \|\theta\|$$

$$\begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}, \theta_0 = 0$$

$$\underline{\theta^T x^{(i)}} = ?$$

$$\begin{array}{c} \uparrow \\ \theta^T x^{(i)} \\ \uparrow \\ u^T v \end{array}$$



$$\underline{\theta^T x^{(i)}} = \boxed{p^{(i)} \cdot \|\theta\|} \leftarrow$$

$$= \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} \leftarrow$$

## SVM Decision Boundary

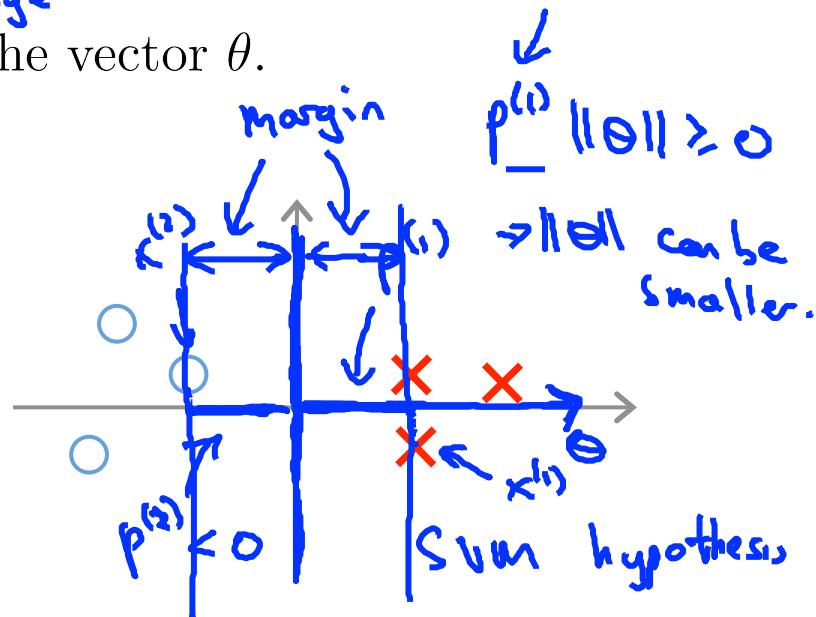
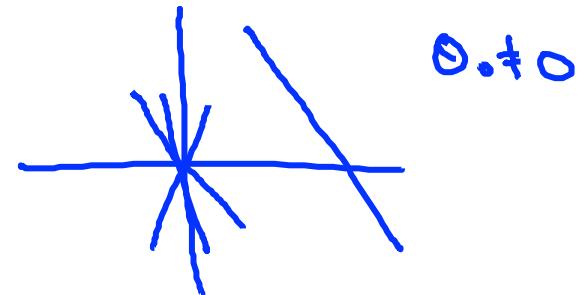
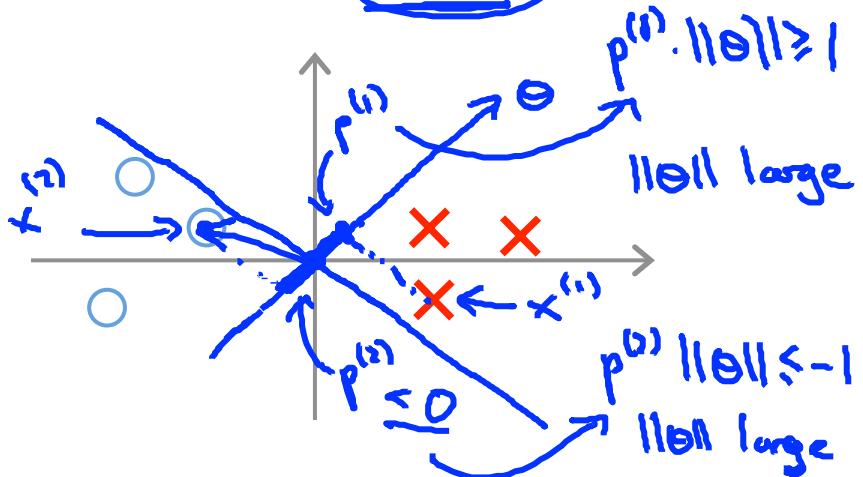
$$\rightarrow \min_{\theta} \frac{1}{2} \sum_{j=1}^n \theta_j^2 = \frac{1}{2} \|\theta\|^2 \leftarrow$$

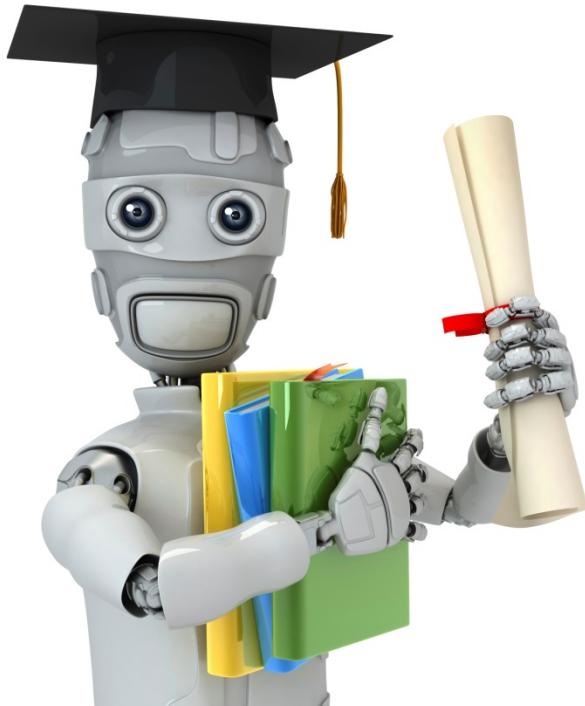
s.t.  $\boxed{p^{(i)} \cdot \|\theta\| \geq 1}$  if  $y^{(i)} = 1$

$\underline{p^{(i)} \cdot \|\theta\| \leq -1}$  if  $y^{(i)} = -1$

where  $p^{(i)}$  is the projection of  $x^{(i)}$  onto the vector  $\theta$ .

Simplification:  $\theta_0 = 0$





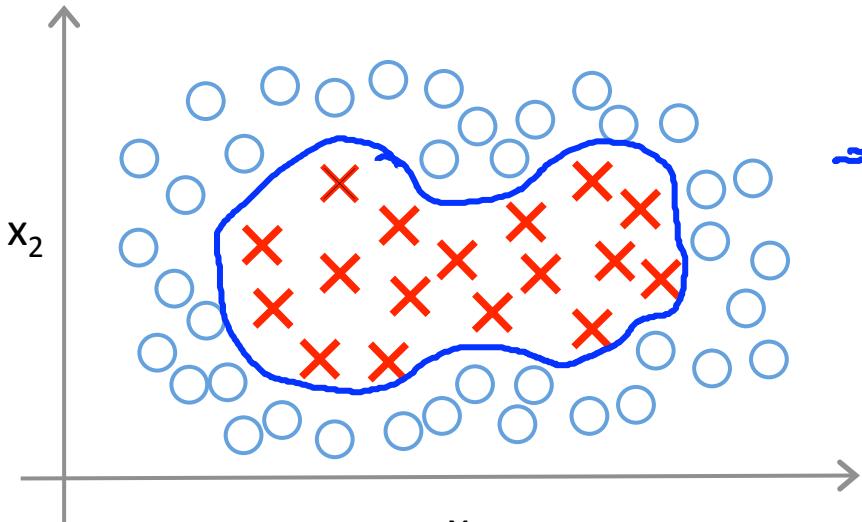
Machine Learning

# Support Vector Machines

---

## Kernels I

## Non-linear Decision Boundary



Predict  $y = 1$  if

$$\theta_0 + \theta_1 \underline{x_1} + \theta_2 \underline{x_2} + \theta_3 \underline{x_1 x_2} \\ + \theta_4 \underline{x_1^2} + \theta_5 \underline{x_2^2} + \dots \geq 0$$

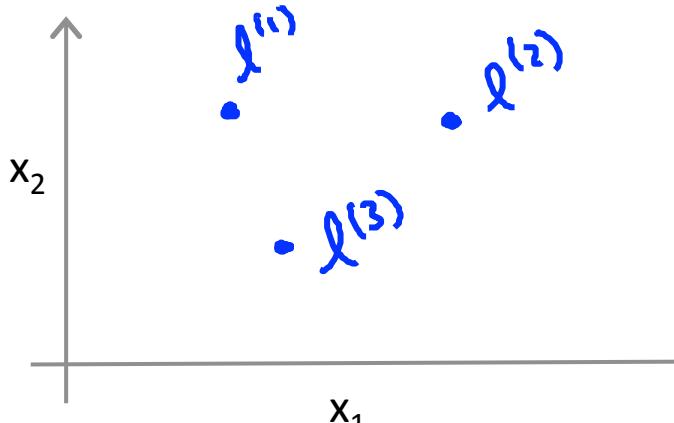
$$h_{\theta}(x) = \begin{cases} 1 & \text{if } \theta_0 + \theta_1 x_1 + \dots \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

$$\rightarrow \theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \dots$$

$$f_1 = x_1, \quad f_2 = x_2, \quad f_3 = x_1 x_2, \quad f_4 = x_1^2, \quad f_5 = x_2^2, \dots$$

Is there a different / better choice of the features  $f_1, f_2, f_3, \dots$ ?

## Kernel



Given  $x$ , compute new feature depending on proximity to landmarks  $l^{(1)}, l^{(2)}, l^{(3)}$

Given  $x$ :

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

$$f_2 = \text{similarity}(x, l^{(2)}) = \exp\left(-\frac{\|x - l^{(2)}\|^2}{2\sigma^2}\right)$$

$$f_3 = \text{similarity}(x, l^{(3)}) = \exp(\dots)$$

↑ Kernel (Gaussian kernels)

$$k(x, l^{(1)})$$

## Kernels and Similarity

$$f_1 = \text{similarity}(x, \underline{l}^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

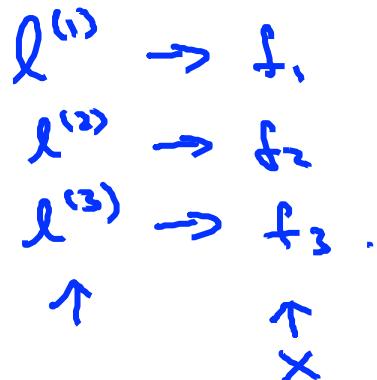


If  $x \approx l^{(1)}$  :

$$f_1 \underset{\uparrow}{\approx} \exp\left(-\frac{0^2}{2\sigma^2}\right) \underset{\downarrow}{\approx} 1$$

If  $x$  if far from  $l^{(1)}$  :

$$f_1 = \exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \underset{\uparrow}{\approx} 0.$$



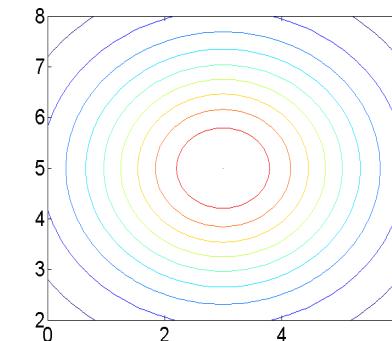
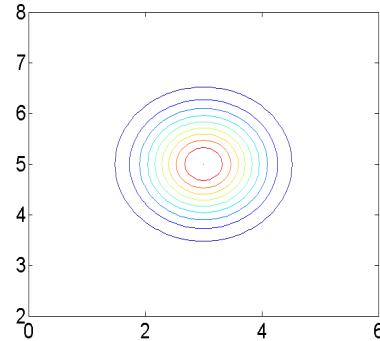
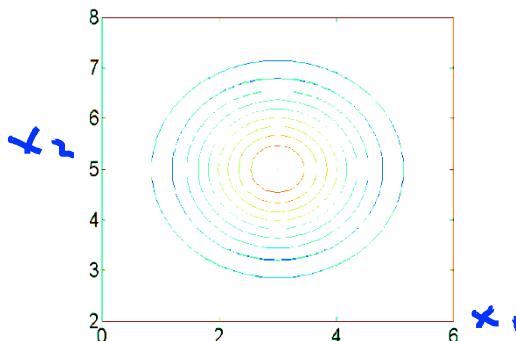
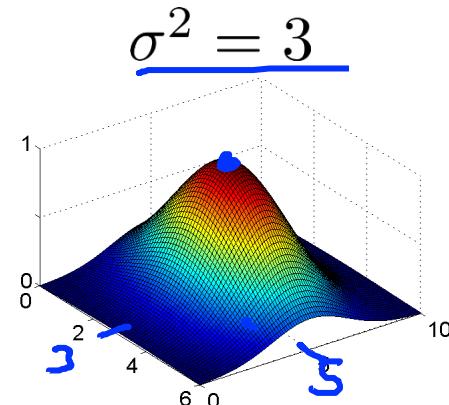
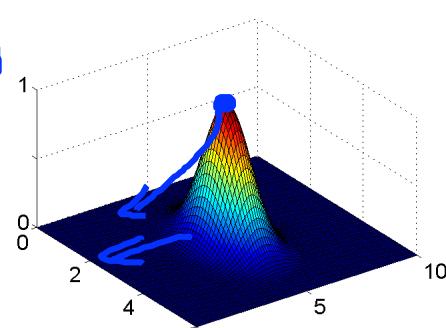
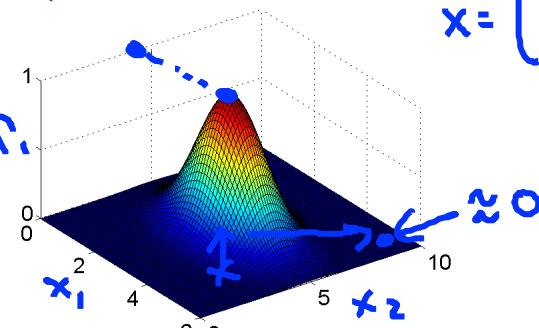
## Example:

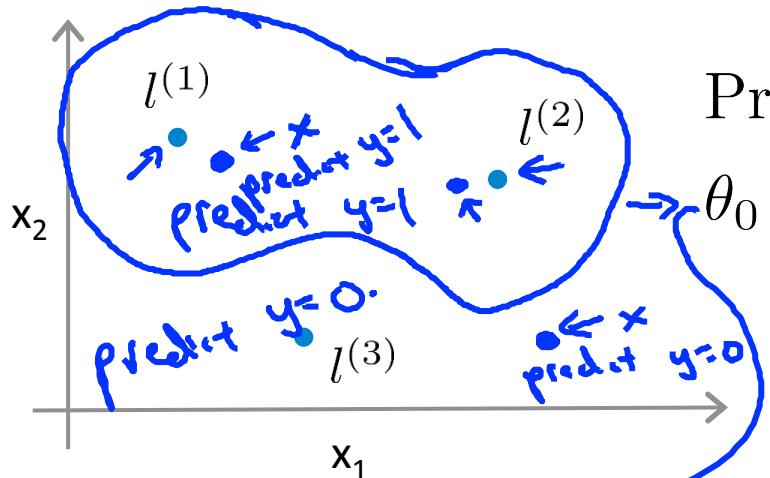
$$\rightarrow l^{(1)} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \quad f_1 = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

$$\rightarrow \sigma^2 = 1$$

$$x = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$

$$\sigma^2 = 0.5$$





Predict "1" when

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$$



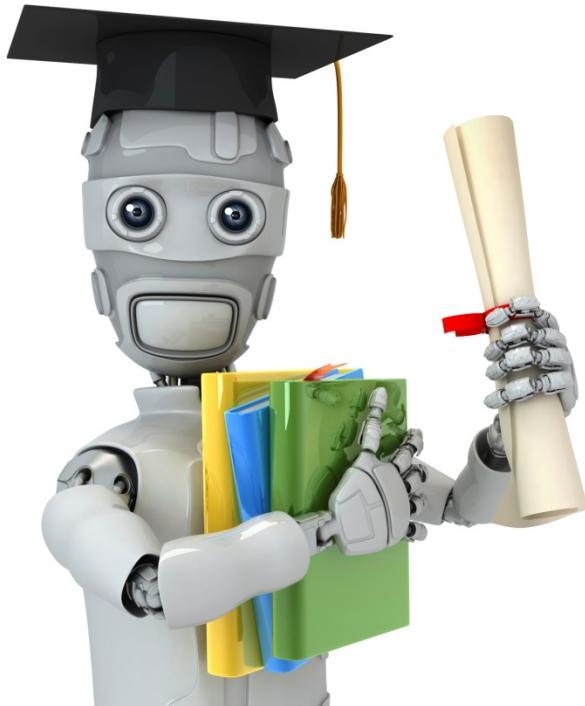
$$\underline{\theta_0 = -0.5, \theta_1 = 1, \theta_2 = 1, \theta_3 = 0}$$

$$f_1 \approx 1, f_2 \approx 0, f_3 \approx 0.$$

$$\begin{aligned} \rightarrow \theta_0 + \theta_1 \cdot 1 + \theta_2 \cdot 0 + \theta_3 \cdot 0 \\ = -0.5 + 1 = 0.5 \geq 0 \end{aligned}$$

$$f_1, f_2, f_3 \approx 0$$

$$\rightarrow \underline{\theta_0 + \theta_1 f_1 + \dots} \approx -0.5 < 0$$



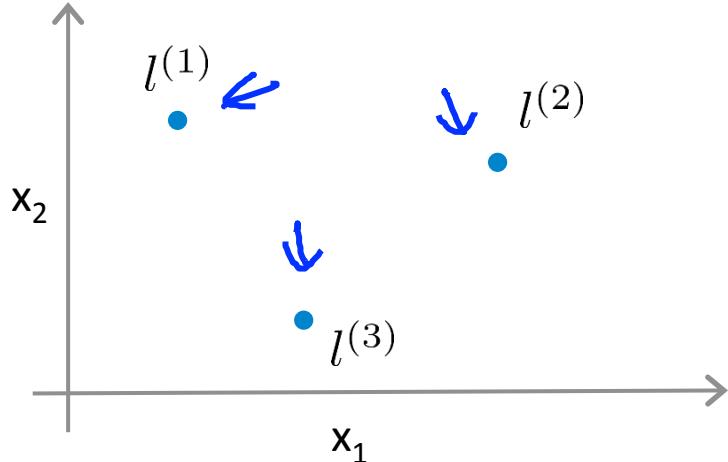
Machine Learning

# Support Vector Machines

---

## Kernels II

## Choosing the landmarks

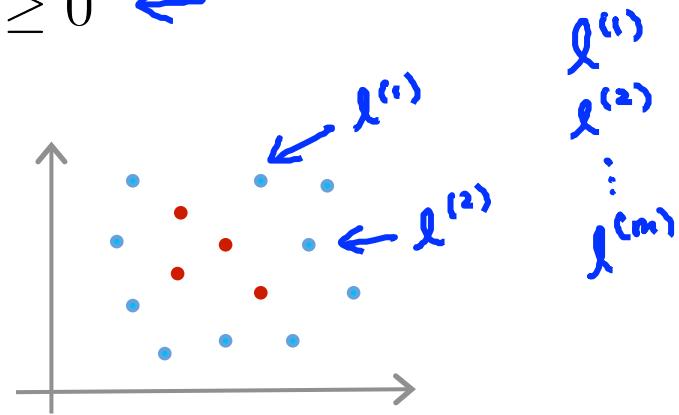
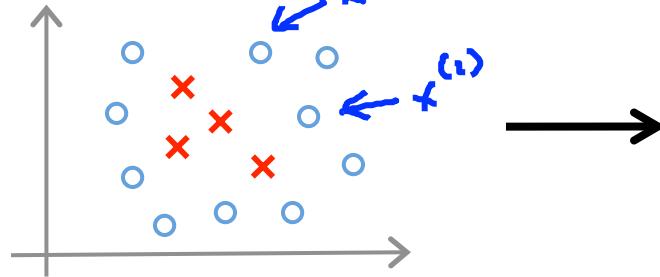


Given  $x$ :

$$\begin{aligned} \rightarrow f_i &= \text{similarity}(x, l^{(i)}) \\ &= \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right) \end{aligned}$$

Predict  $y = 1$  if  $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

Where to get  $l^{(1)}, l^{(2)}, l^{(3)}, \dots$ ?



## SVM with Kernels

- Given  $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})$ ,
- choose  $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \dots, l^{(m)} = x^{(m)}$ .

Given example  $\underline{x}$ :

$$\begin{aligned} \rightarrow f_1 &= \text{similarity}(x, l^{(1)}) & \downarrow x^{(1)} \\ \rightarrow f_2 &= \text{similarity}(x, l^{(2)}) \\ &\vdots \\ \rightarrow f_m &= \text{similarity}(x, l^{(m)}) \end{aligned}$$

$$f = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix} \quad f_0 = 1$$

For training example  $(x^{(i)}, y^{(i)})$ :

$$\begin{aligned} \underline{x^{(i)}} \rightarrow f_1^{(i)} &= \overline{\text{sim}}(x^{(i)}, l^{(1)}) & \downarrow x^{(i)} \\ f_2^{(i)} &= \overline{\text{sim}}(x^{(i)}, l^{(2)}) \\ &\vdots \\ f_m^{(i)} &= \overline{\text{sim}}(x^{(i)}, l^{(m)}) = \exp(-\frac{\alpha}{\gamma_{\text{sim}}}) = 1 \end{aligned}$$

$$\begin{aligned} \underline{x^{(i)}} \in \mathbb{R}^{n+1} & \quad (\text{or } \mathbb{R}^n) \\ f^{(i)} = & \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ f_2^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix} \\ f_0^{(i)} &= 1 \end{aligned}$$

## SVM with Kernels

Hypothesis: Given  $\underline{x}$ , compute features  $\underline{f} \in \mathbb{R}^{m+1}$

→ Predict "y=1" if  $\theta^T \underline{f} \geq 0$

$$\sqrt{\theta_0 + \theta_1 + \dots + \theta_m}$$

$$\theta \in \mathbb{R}^{m+1}$$

$$\theta_0 f_0 + \theta_1 f_1 + \dots + \theta_m f_m$$

Training:

$$\min_{\theta} C \sum_{i=1}^m y^{(i)} \text{cost}_1(\theta^T f^{(i)}) + (1 - y^{(i)}) \text{cost}_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^n \theta_j^2$$

$$\begin{array}{c} n = m \\ \cancel{\theta_0} = m \\ \frac{1}{2} \sum_{j=1}^m \theta_j^2 \\ \rightarrow \theta_0 \end{array}$$

$$\begin{aligned} - \sum_j \theta_j^2 &= \theta^T \theta \quad \leftarrow \theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_m \end{bmatrix} \\ &\rightarrow \underline{\theta^T M \theta} \quad \leftarrow \| \theta \|^2 \end{aligned}$$

(ignoring  $\theta_0$ )  
 $M = 10,000$

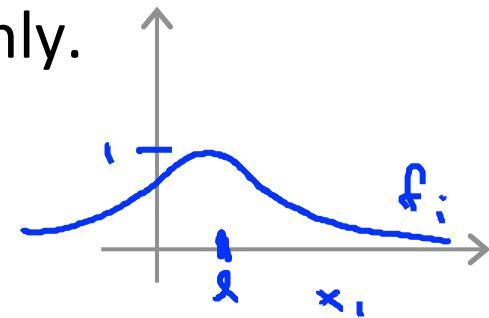
## SVM parameters:

$C \left( = \frac{1}{\lambda} \right)$ .  $\rightarrow$  Large C: Lower bias, high variance. λ (small λ)  
 $\rightarrow$  Small C: Higher bias, low variance. λ (large λ)

$\sigma^2$  Large  $\sigma^2$ : Features  $f_i$  vary more smoothly.

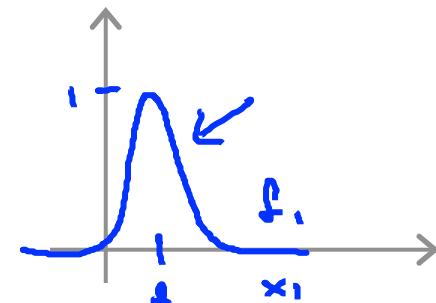
$\rightarrow$  Higher bias, lower variance.

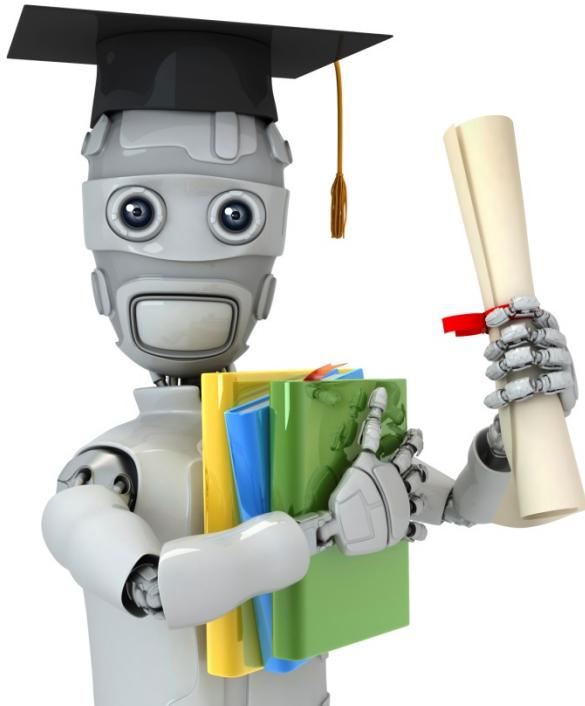
$$\exp \left( - \frac{\|x - f_i\|^2}{2\sigma^2} \right)$$



Small  $\sigma^2$ : Features  $f_i$  vary less smoothly.

Lower bias, higher variance.





Machine Learning

# Support Vector Machines

---

## Using an SVM

Use SVM software package (e.g. liblinear, libsvm, ...) to solve for parameters  $\theta$ .



Need to specify:

→ Choice of parameter C.

Choice of kernel (similarity function):

E.g. No kernel ("linear kernel")

Predict "y = 1" if  $\underline{\theta^T x} \geq 0$

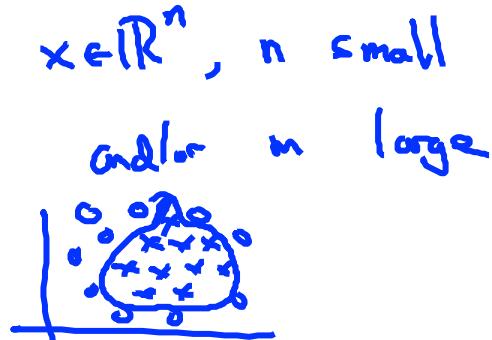
$$\theta_0 + \theta_1 x_1 + \dots + \theta_n x_n \geq 0$$

→ n large, m small  $x \in \mathbb{R}^{n+1}$

→ Gaussian kernel:

$$f_i = \exp\left(-\frac{\|x - l^{(i)}\|^2}{2\sigma^2}\right), \text{ where } l^{(i)} = x^{(i)}$$

Need to choose  $\underline{\sigma^2}$ .



## Kernel (similarity) functions:

function  $f = \text{kernel}(x_1, x_2)$

$$f = \exp\left(-\frac{\|x_1 - x_2\|^2}{2\sigma^2}\right)$$

return

$$\begin{aligned} x &\rightarrow \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{aligned}$$

→ Note: Do perform feature scaling before using the Gaussian kernel.

$$\begin{aligned} &\Rightarrow \|x - l\|^2 \quad x \in \mathbb{R}^{n+1} \\ &V = x - l \\ &\|v\|^2 = v_1^2 + v_2^2 + \dots + v_n^2 \\ &= (x_1 - l_1)^2 + (x_2 - l_2)^2 + \dots + (x_n - l_n)^2 \\ &\quad \underbrace{\quad}_{1000 \text{ feet}^2} \quad \underbrace{\quad}_{1-5 \text{ bedrooms}} \end{aligned}$$

## Other choices of kernel

Note: Not all similarity functions  $\text{similarity}(x, l)$  make valid kernels.

→ (Need to satisfy technical condition called “Mercer’s Theorem” to make sure SVM packages’ optimizations run correctly, and do not diverge).

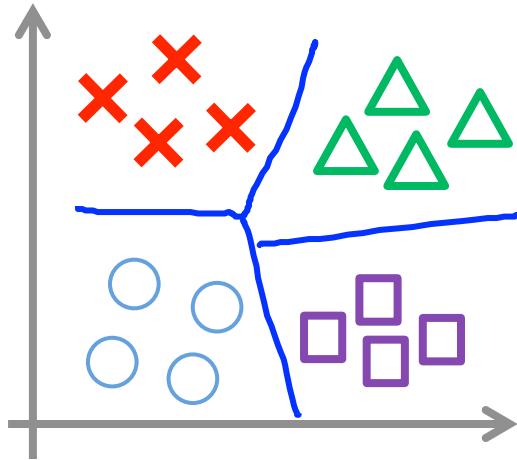
Many off-the-shelf kernels available:

- Polynomial kernel:  $k(x, l) =$

$$(x^T l)^2, \quad (x^T l + \text{constant})^{\text{degree}}$$
$$(x^T l)^3, \quad (x^T l + 1)^3, \quad (x^T l + 5)^4$$

- More esoteric: String kernel, chi-square kernel, histogram intersection kernel, ...  
 $\text{sim}(x, l)$

## Multi-class classification



$$y \in \{1, 2, 3, \dots, K\}$$

Many SVM packages already have built-in multi-class classification functionality.

- Otherwise, use one-vs.-all method. (Train  $K$  SVMs, one to distinguish  $y = i$  from the rest, for  $i = 1, 2, \dots, K$ ), get  $\theta^{(1)}, \theta^{(2)}, \dots, \underline{\theta^{(K)}}$
- Pick class  $i$  with largest  $(\theta^{(i)})^T x$

$\overset{\uparrow}{y=1} \quad \overset{\uparrow}{y=2} \quad \cdots \quad \overset{\uparrow}{\theta^{(K)}}$

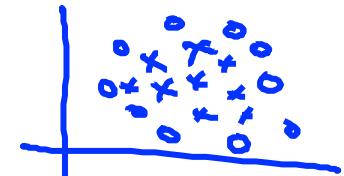
## Logistic regression vs. SVMs

$n$  = number of features ( $x \in \mathbb{R}^{n+1}$ ),  $m$  = number of training examples

- If  $n$  is large (relative to  $m$ ): (e.g.  $n \geq m$ ,  $n = \underline{10,000}$ ,  $m = \underline{10} \dots \underline{1000}$ )
- Use logistic regression, or SVM without a kernel ("linear kernel")

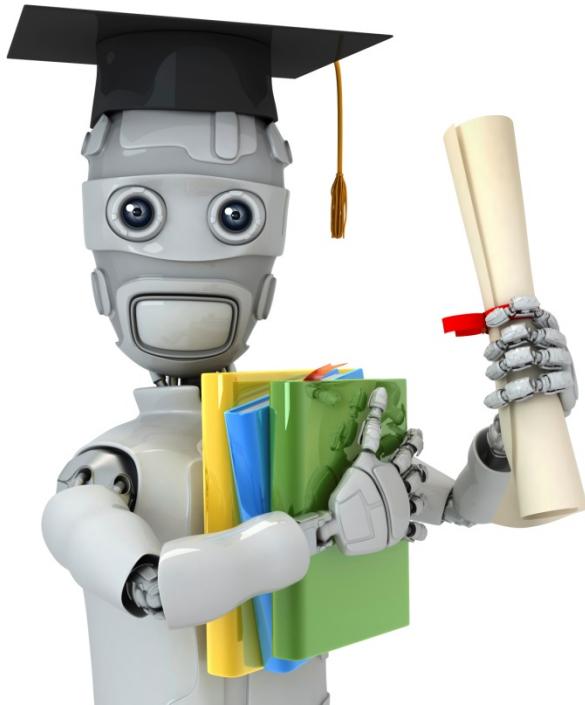
- If  $n$  is small,  $m$  is intermediate: ( $n = \underline{1-1000}$ ,  $m = \underline{10 - 10,000}$ )
  - Use SVM with Gaussian kernel

If  $n$  is small,  $m$  is large: ( $n = \underline{1-1000}$ ,  $m = \underline{50,000+}$ )



- Create/add more features, then use logistic regression or SVM without a kernel

- Neural network likely to work well for most of these settings, but may be slower to train.



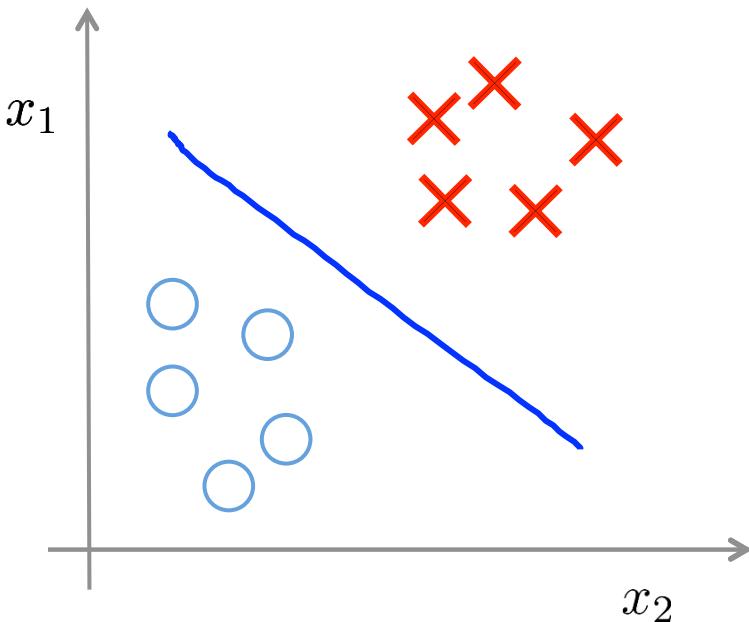
Machine Learning

# Clustering

---

## Unsupervised learning introduction

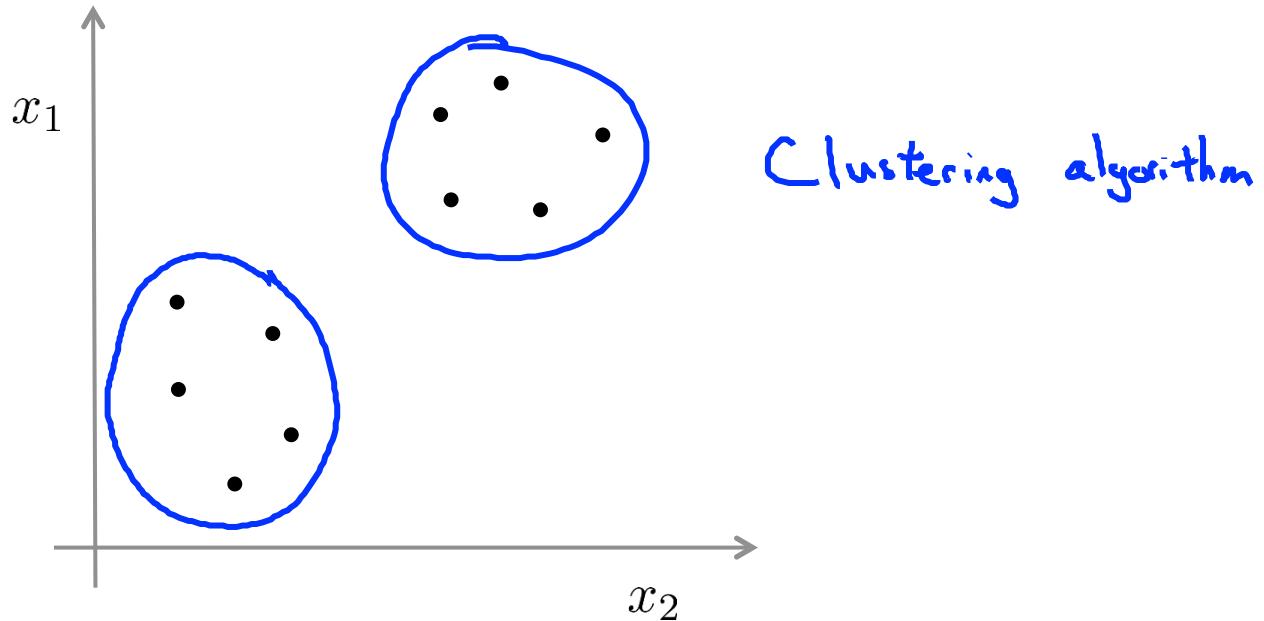
# Supervised learning



Training set:  $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), (x^{(3)}, y^{(3)}), \dots, (x^{(m)}, y^{(m)})\}$

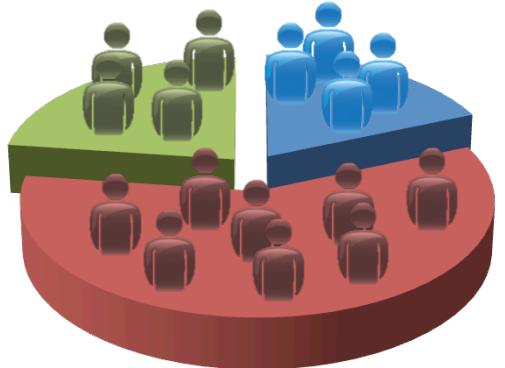


# Unsupervised learning

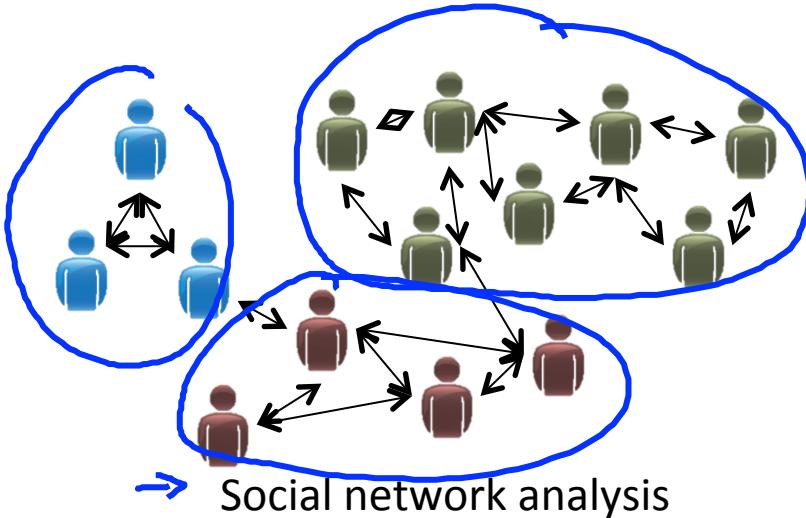


Training set:  $\{\underline{x}^{(1)}, \underline{x}^{(2)}, \underline{x}^{(3)}, \dots, \underline{x}^{(m)}\}$   $\leftarrow$

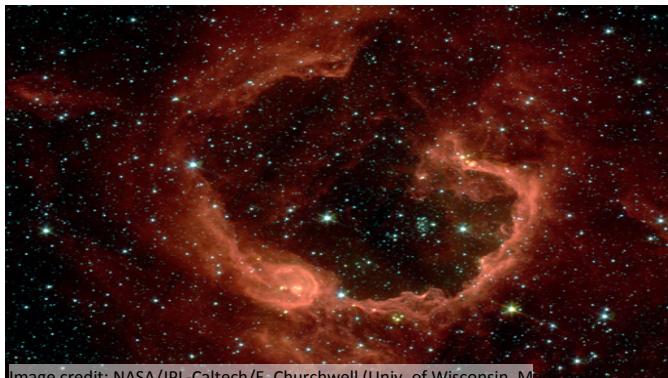
# Applications of clustering



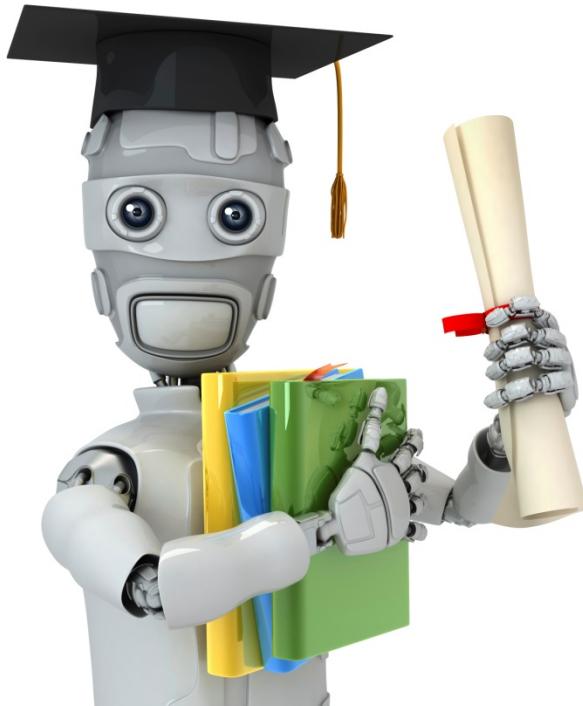
→ Market segmentation



Organize computing clusters



→ Astronomical data analysis

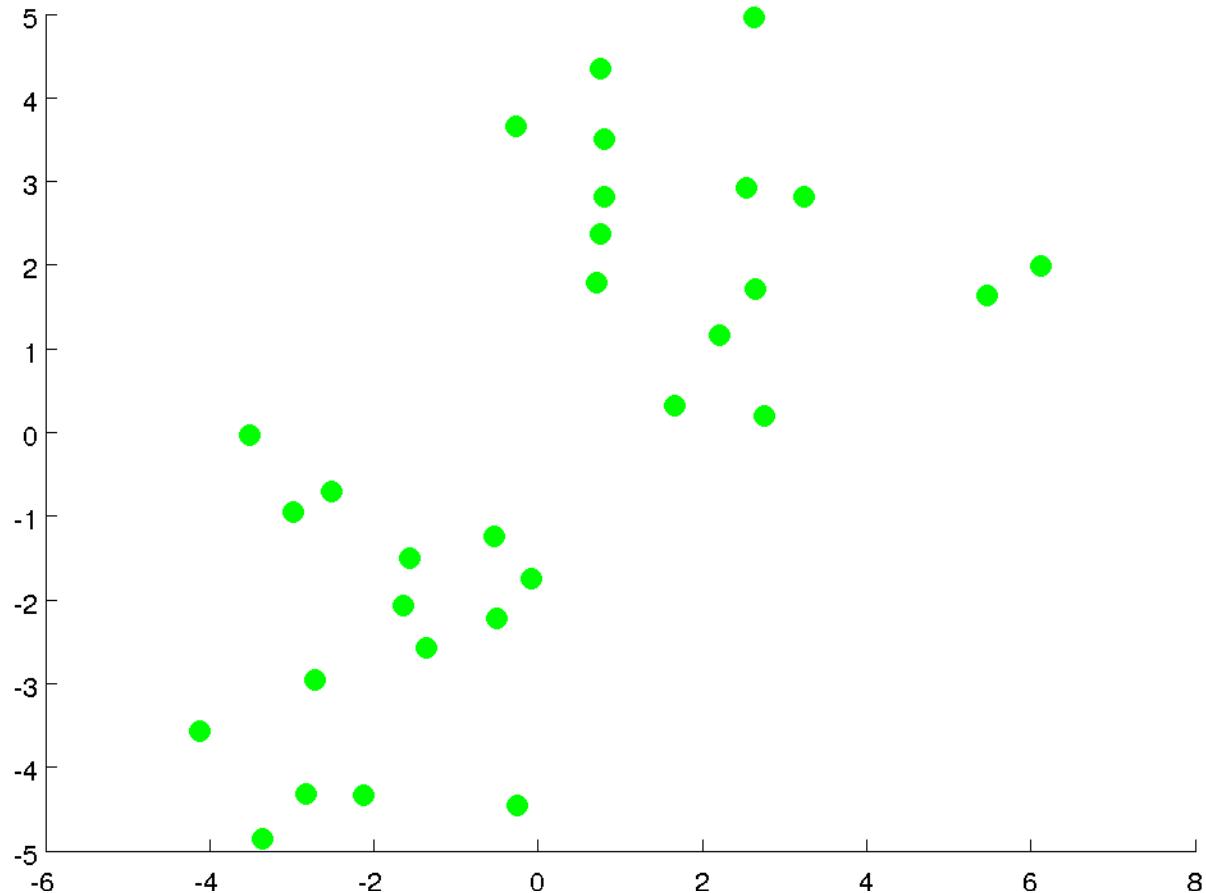


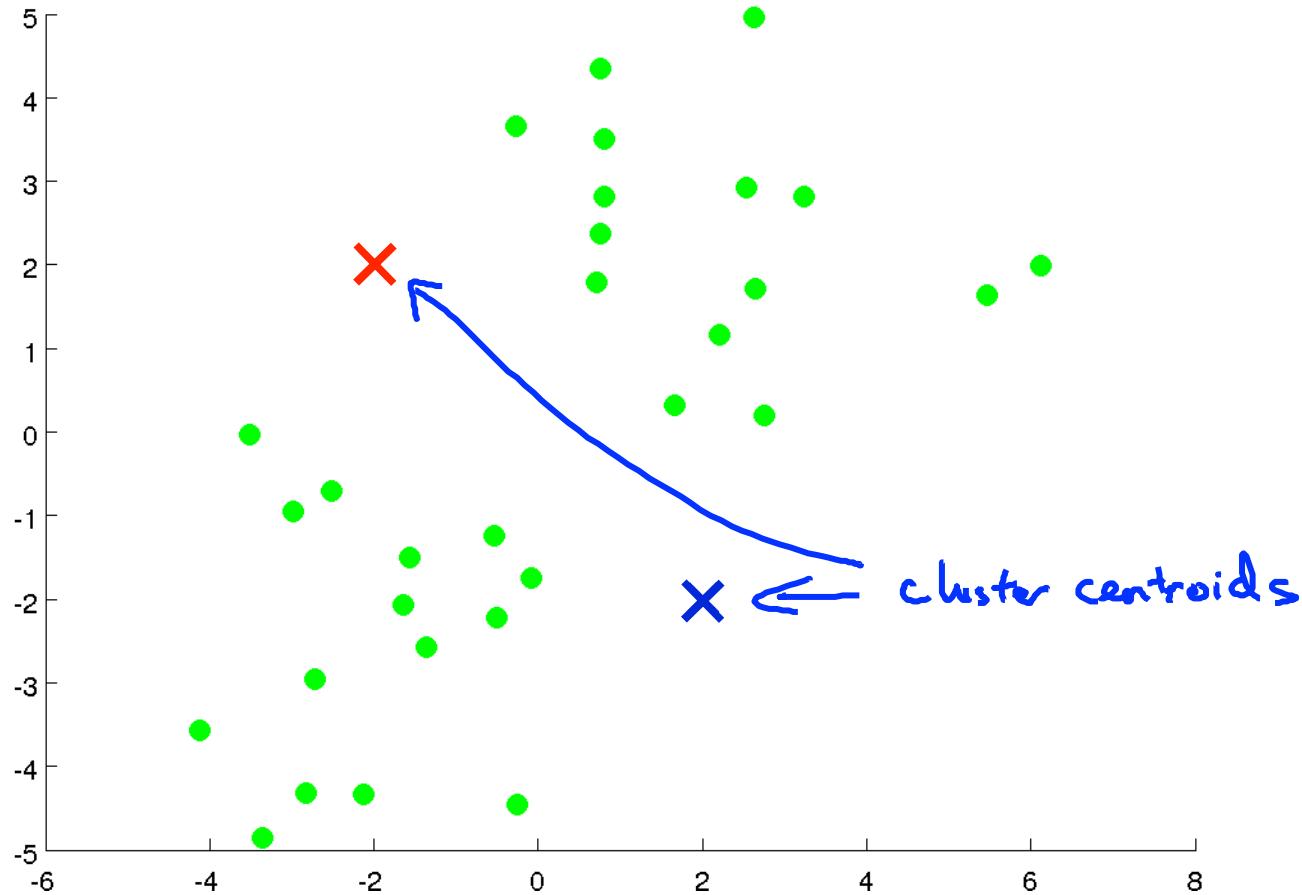
Machine Learning

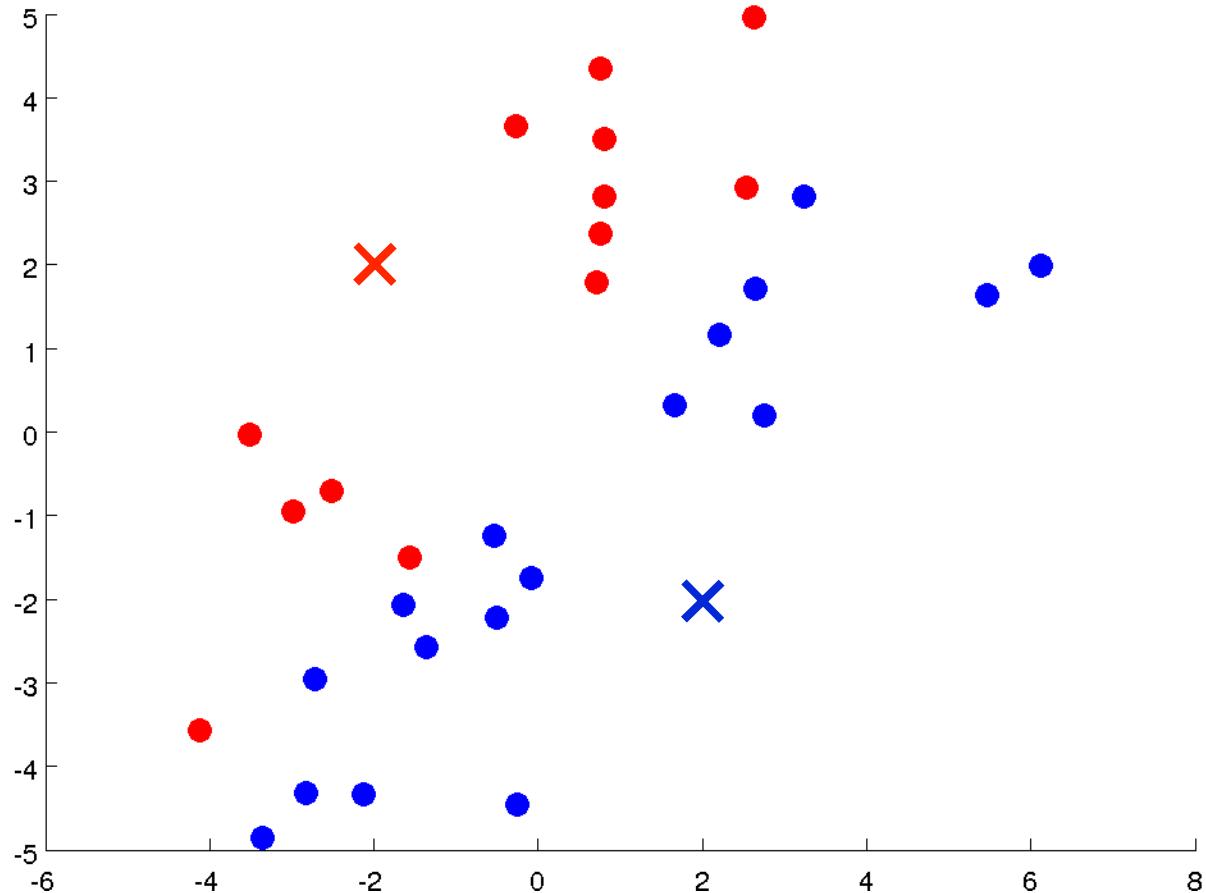
# Clustering

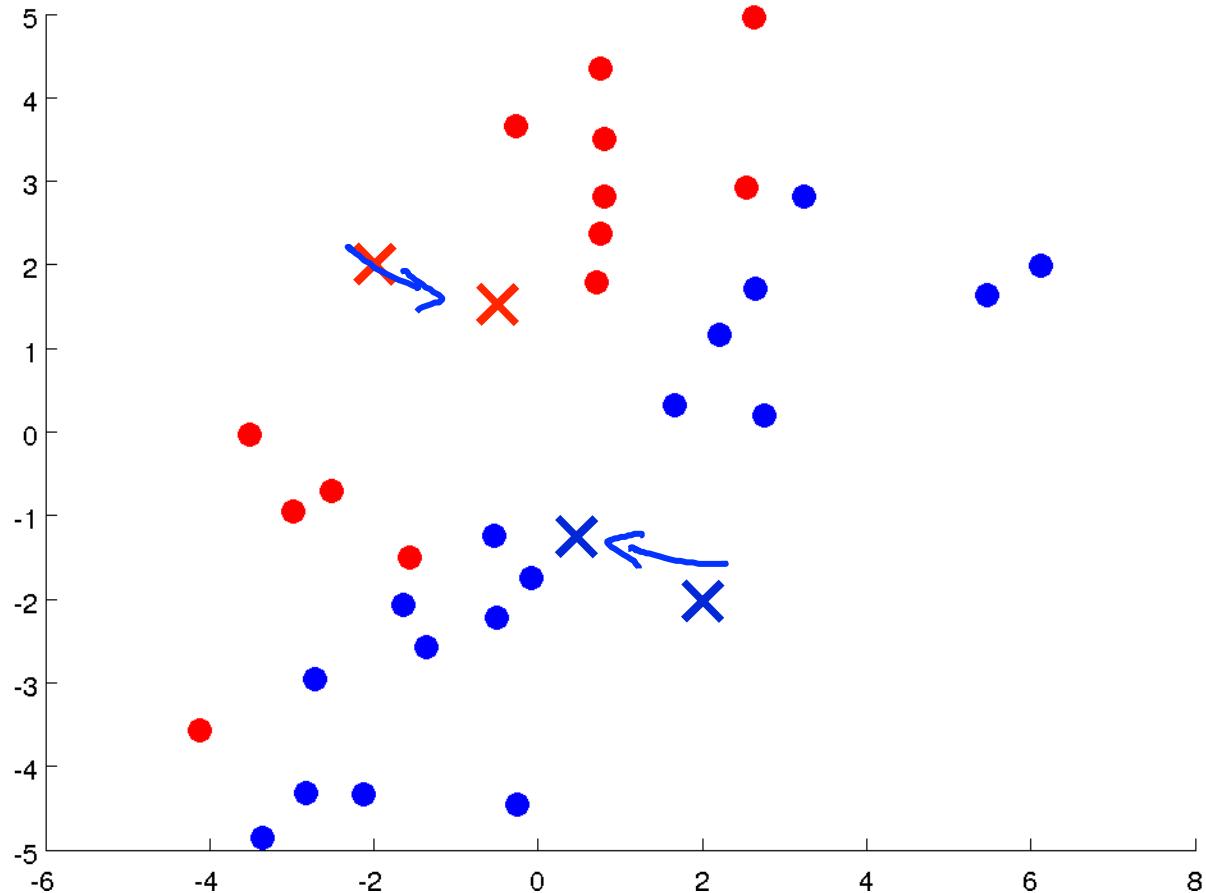
---

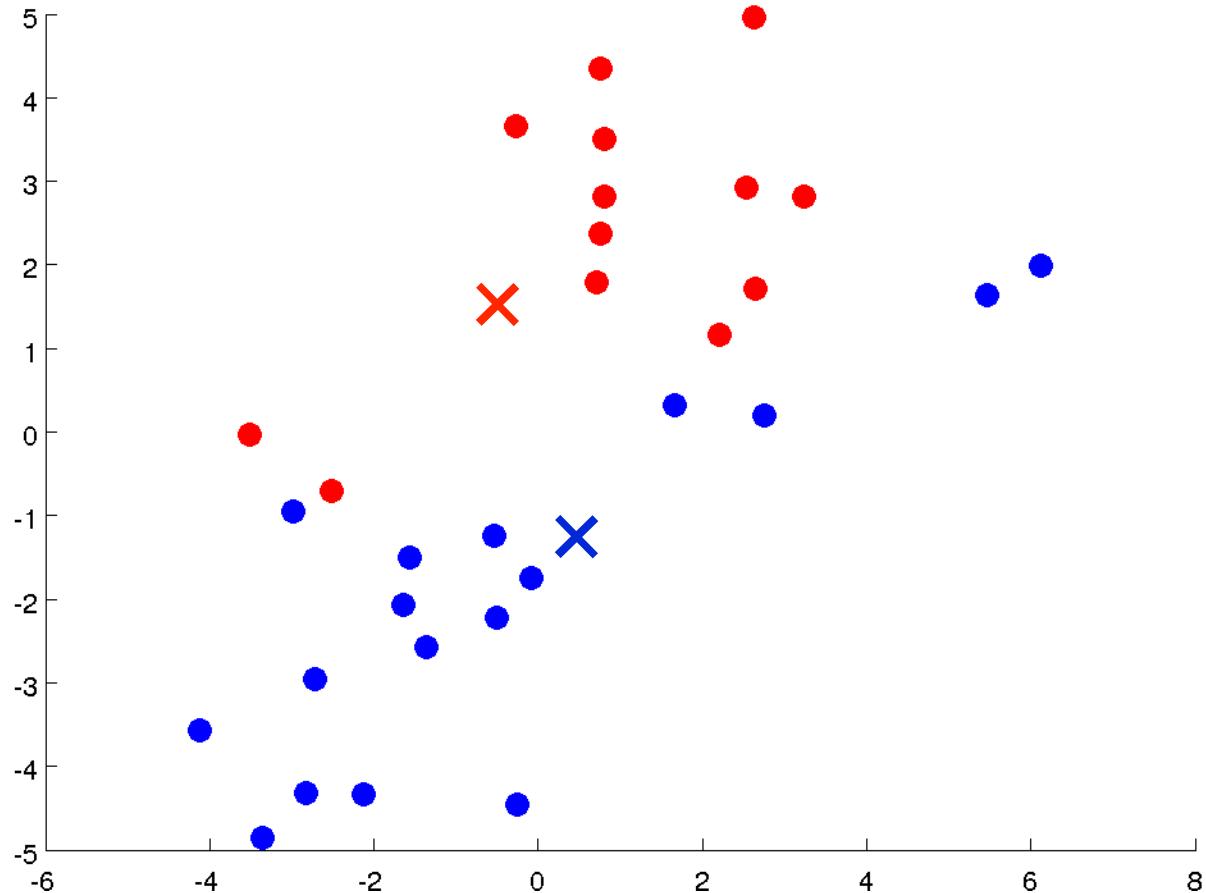
K-means  
algorithm

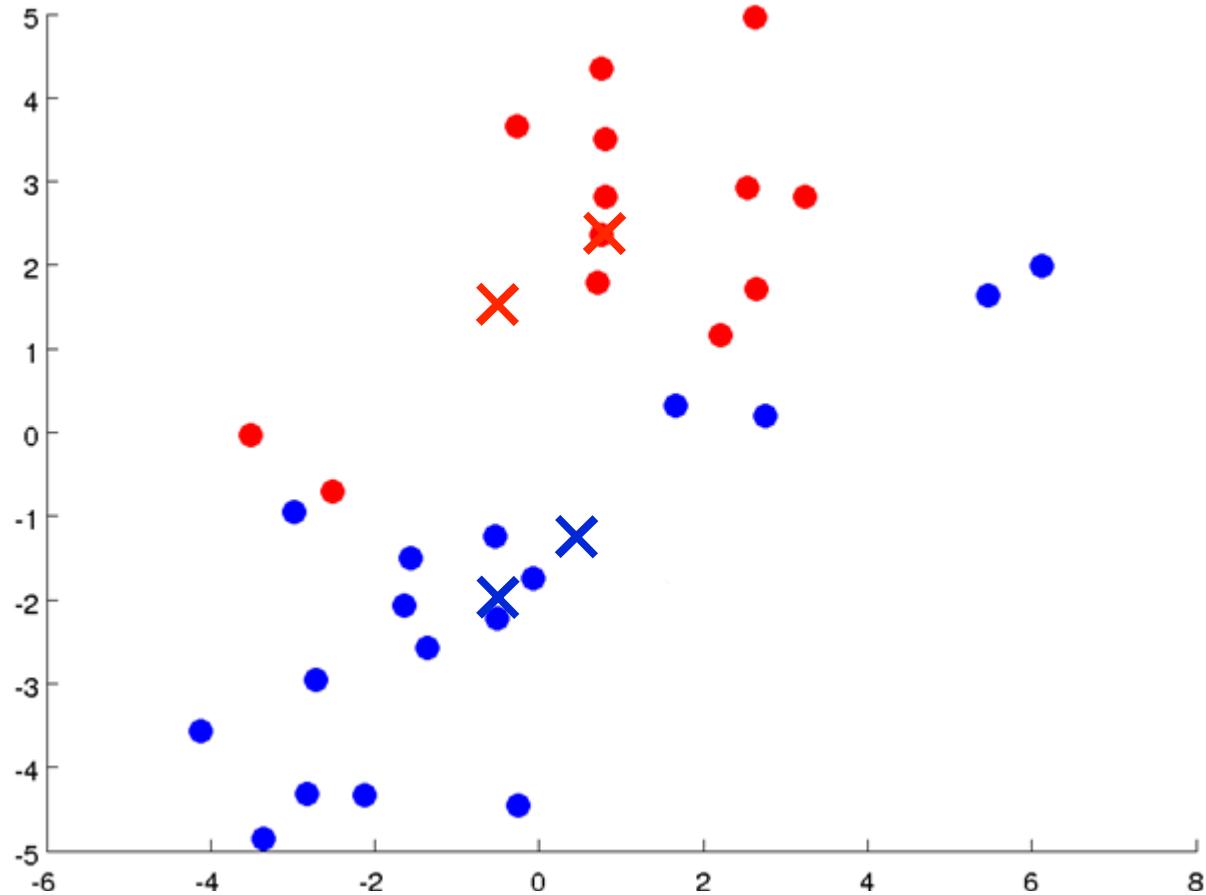


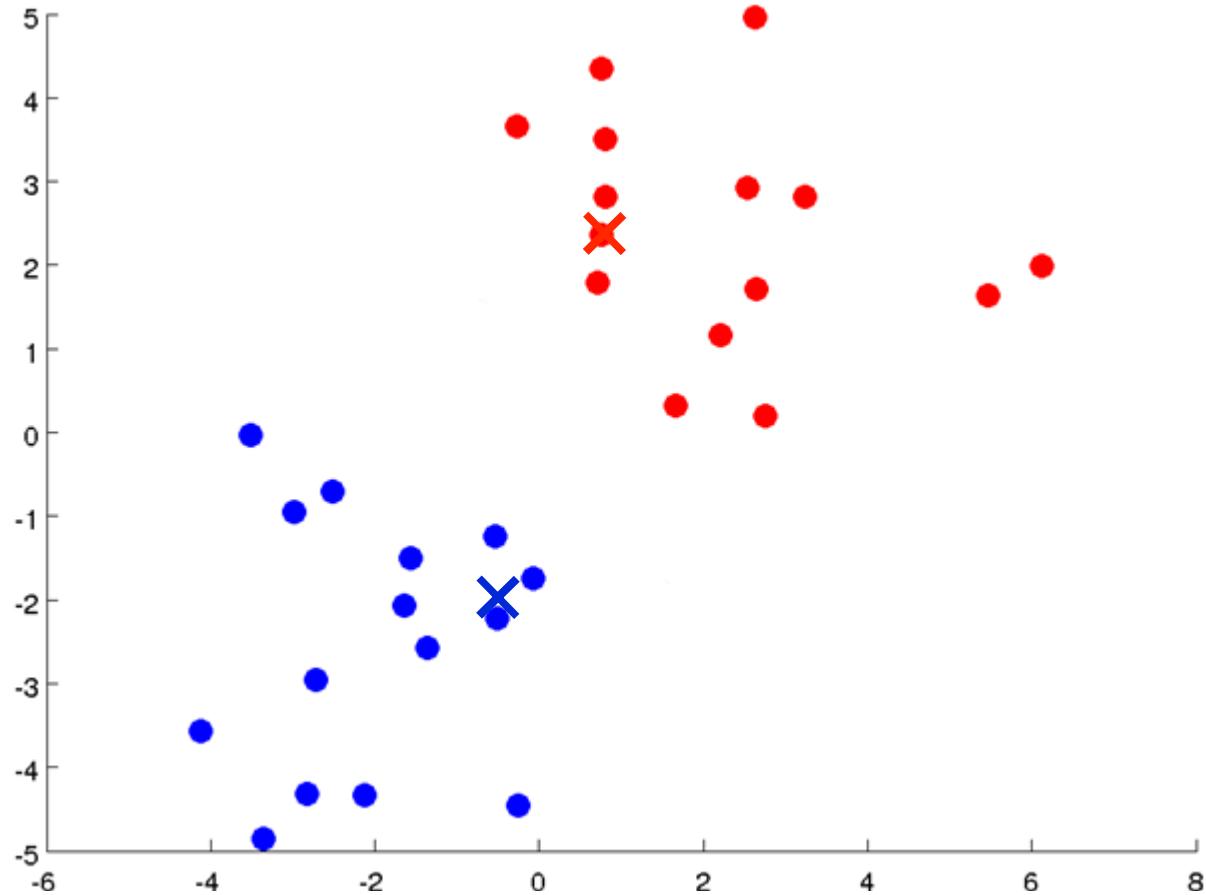


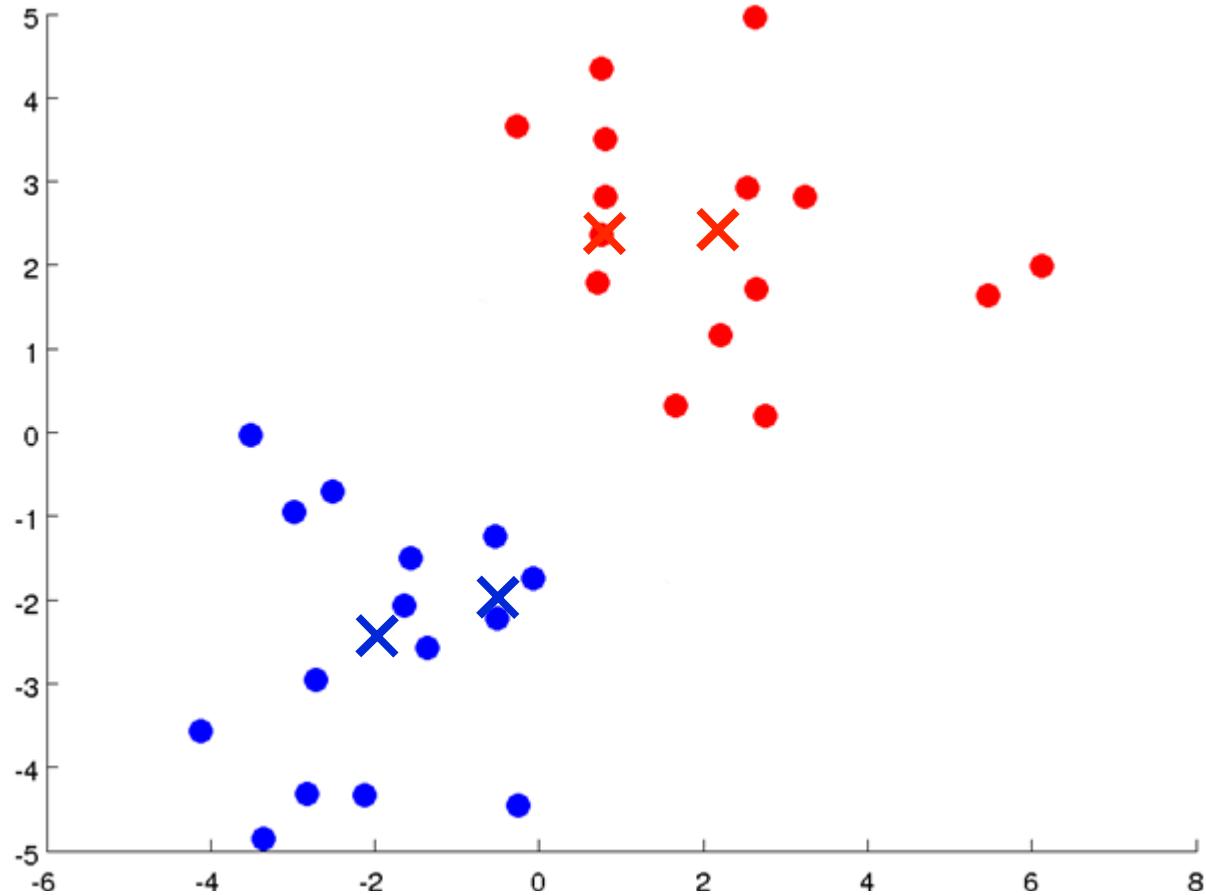


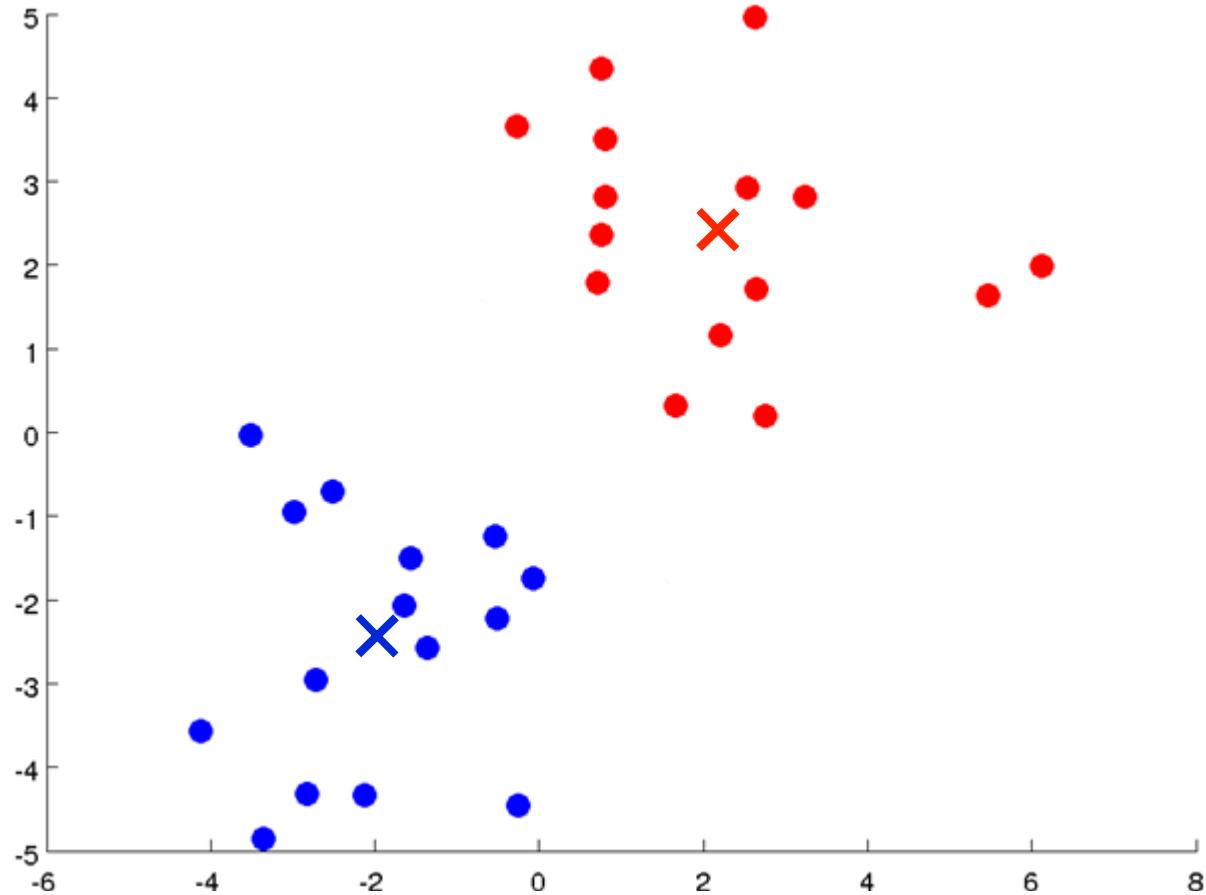












# K-means algorithm

Input:

- $K$  (number of clusters) 
- Training set  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$  

$x^{(i)} \in \mathbb{R}^n$  (drop  $x_0 = 1$  convention)

## K-means algorithm

$$\mu_1 \quad \mu_2$$

Randomly initialize  $K$  cluster centroids  $\underline{\mu}_1, \underline{\mu}_2, \dots, \underline{\mu}_K \in \mathbb{R}^n$

Repeat {

Cluster  
assignment  
step

for  $i = 1$  to  $m$

$\underline{c}^{(i)}$  := index (from 1 to  $K$ ) of cluster centroid  
closest to  $x^{(i)}$

$$\min_k \|\underline{x}^{(i)} - \underline{\mu}_k\|^2$$

for  $k = 1$  to  $K$

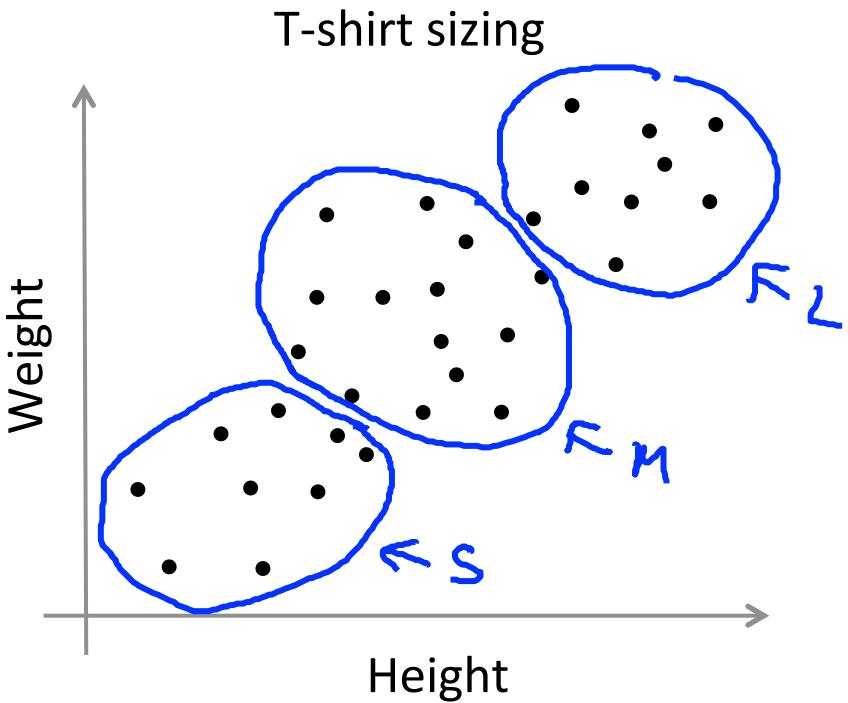
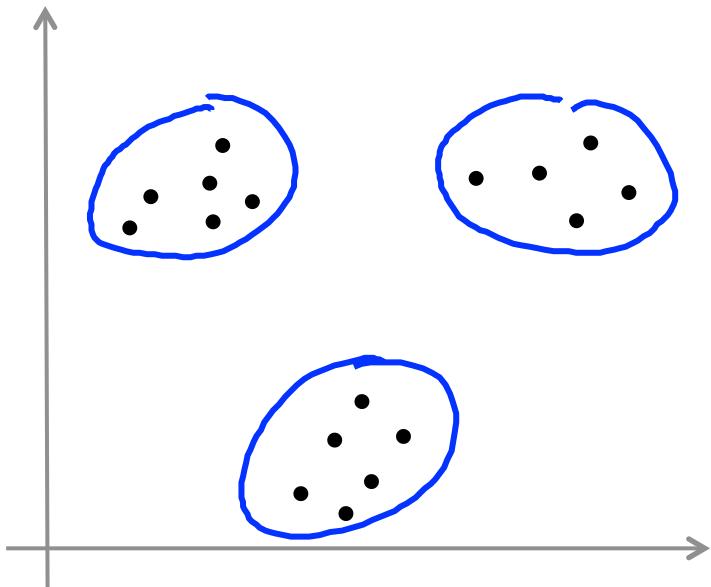
$\rightarrow \underline{\mu}_k$  := average (mean) of points assigned to cluster  $k$   
 $\underline{x}^{(1)}, \underline{x}^{(2)}, \underline{x}^{(3)}, \underline{x}^{(4)}$   $\rightarrow \underline{c}^{(1)}=2, \underline{c}^{(2)}=2, \underline{c}^{(3)}=2, \underline{c}^{(4)}=2$

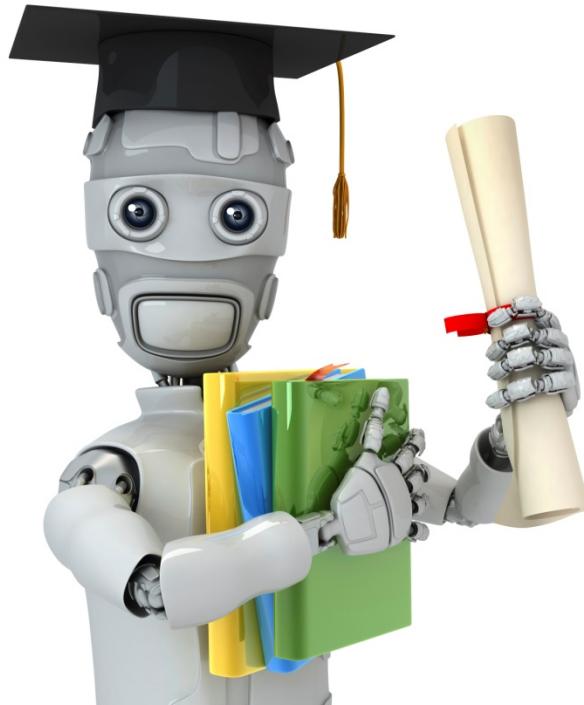
}

$$\underline{\mu}_2 = \frac{1}{4} \left[ \underline{x}^{(1)} + \underline{x}^{(2)} + \underline{x}^{(3)} + \underline{x}^{(4)} \right] \in \mathbb{R}^n$$

## K-means for non-separated clusters

S, M, L





Machine Learning

# Clustering Optimization objective

---

## K-means optimization objective

- $c^{(i)}$  = index of cluster ( $1, 2, \dots, K$ ) to which example  $x^{(i)}$  is currently assigned
- $\mu_k$  = cluster centroid  $k$  ( $\mu_k \in \mathbb{R}^n$ )  $K$   
 $k \in \{1, 2, \dots, K\}$
- $\mu_{c^{(i)}}$  = cluster centroid of cluster to which example  $x^{(i)}$  has been assigned  $x^{(i)} \rightarrow S$   
 $c^{(i)} = s$   
 $\mu_{c^{(i)}} = \mu_s$

Optimization objective:

$$\rightarrow J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

min  $c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K$       Distortion

# K-means algorithm

Randomly initialize  $K$  cluster centroids  $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$

Repeat {      [Cluster assignment step]  
                Minimize  $J(\dots)$  wrt  $[c^{(1)}, c^{(2)}, \dots, c^{(n)}] \leftarrow$   
                (holding  $\mu_1, \dots, \mu_K$  fixed)

for  $i = 1$  to  $m$

$c^{(i)} :=$  index (from 1 to  $K$ ) of cluster centroid  
closest to  $x^{(i)}$

move  
centroid

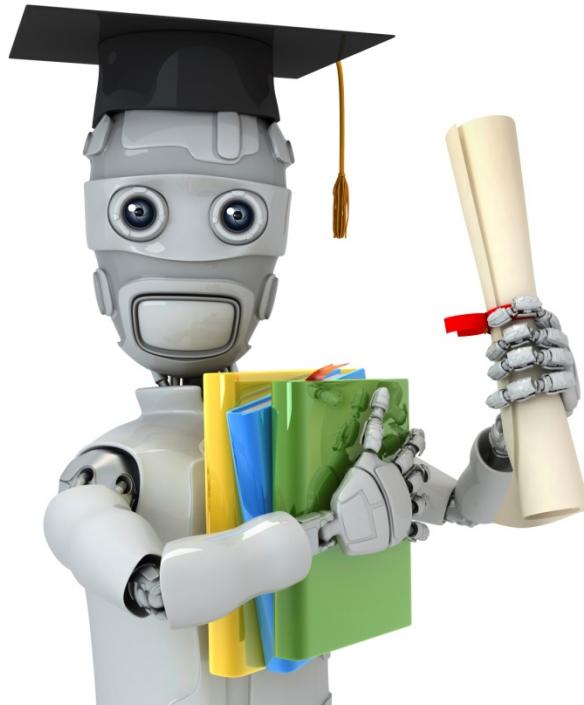
for  $k = 1$  to  $K$

$\mu_k :=$  average (mean) of points assigned to cluster  $k$

}

minimize  $J(\dots)$  wrt

$[\mu_1, \dots, \mu_K]$



Machine Learning

# Clustering

---

## Random initialization

## K-means algorithm

Randomly initialize  $K$  cluster centroids  $\mu_1, \mu_2, \dots, \mu_K \in \mathbb{R}^n$

Repeat {

    for  $i = 1$  to  $m$

$c^{(i)} :=$  index (from 1 to  $K$ ) of cluster centroid  
        closest to  $x^{(i)}$

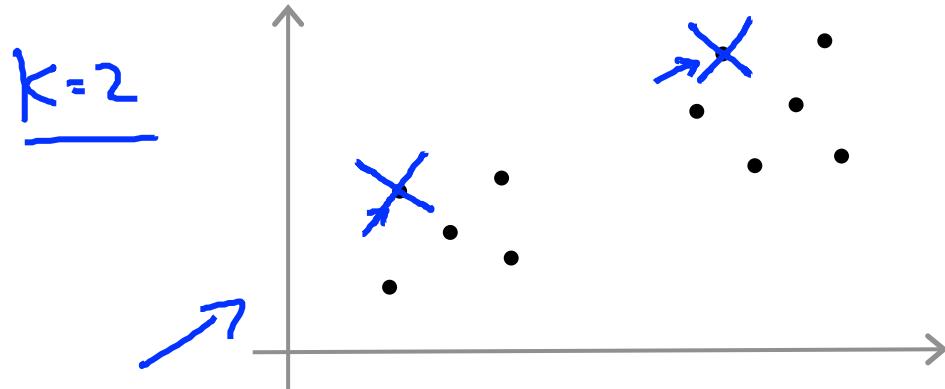
    for  $k = 1$  to  $K$

$\mu_k :=$  average (mean) of points assigned to cluster  $k$

}

## Random initialization

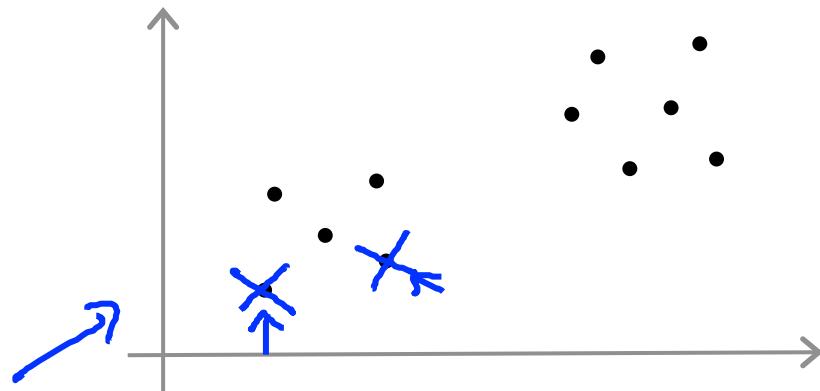
Should have  $K < m$



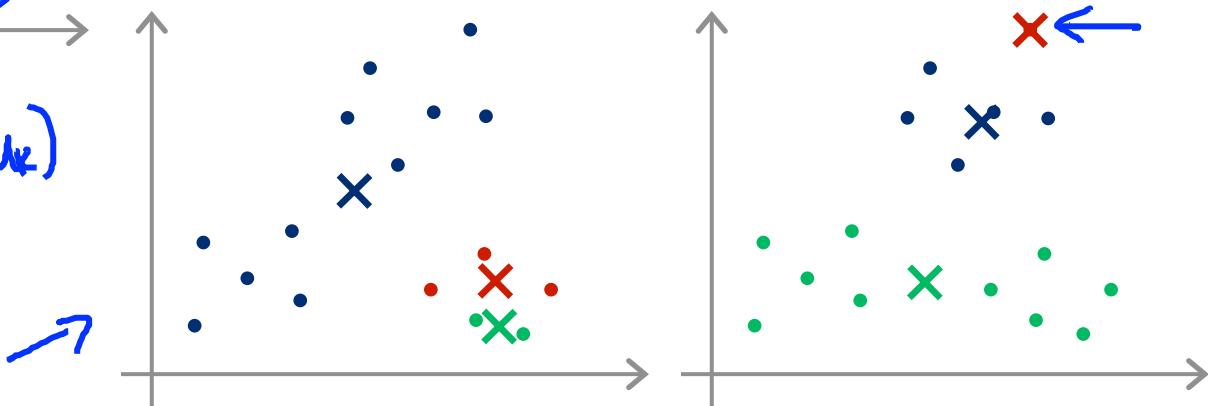
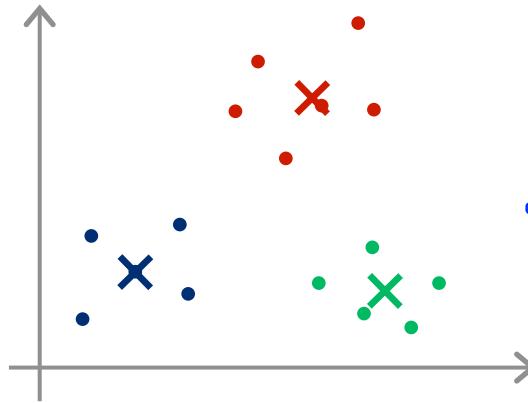
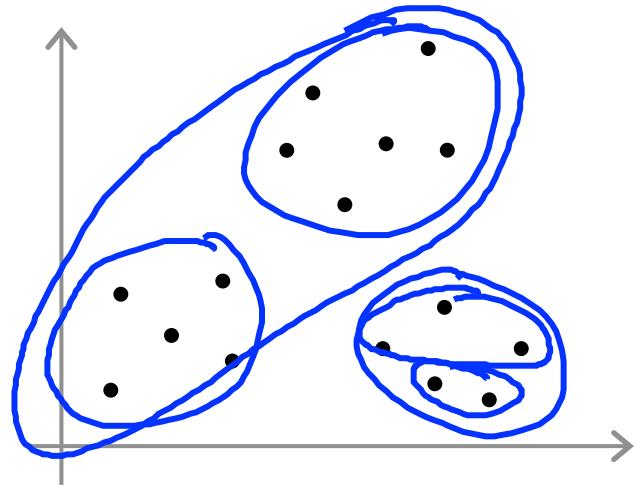
Randomly pick  $K$  training examples.

Set  $\mu_1, \dots, \mu_K$  equal to these  $K$  examples.

$$\begin{aligned}\mu_1 &= x^{(1)} \\ \mu_2 &= x^{(2)} \\ &\vdots\end{aligned}$$



## Local optima



## Random initialization

For i = 1 to 100 {

    Randomly initialize K-means.

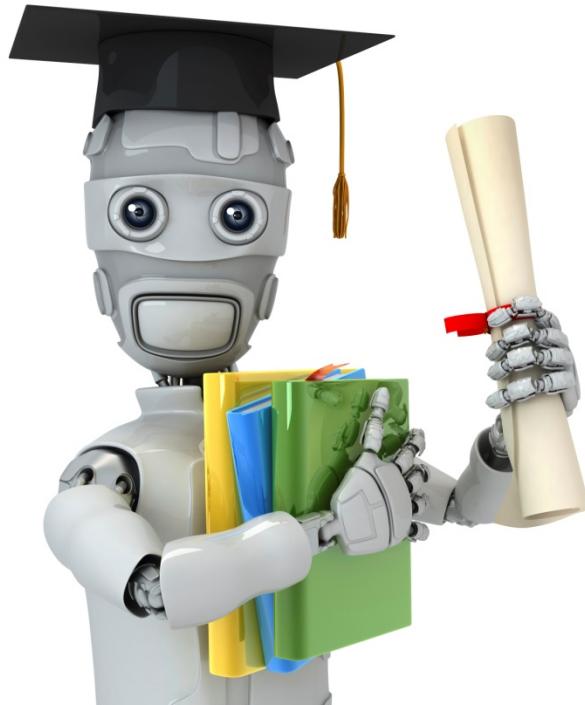
    Run K-means. Get  $c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K$ .

    Compute cost function (distortion)

$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$$

}

Pick clustering that gave lowest cost  $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$

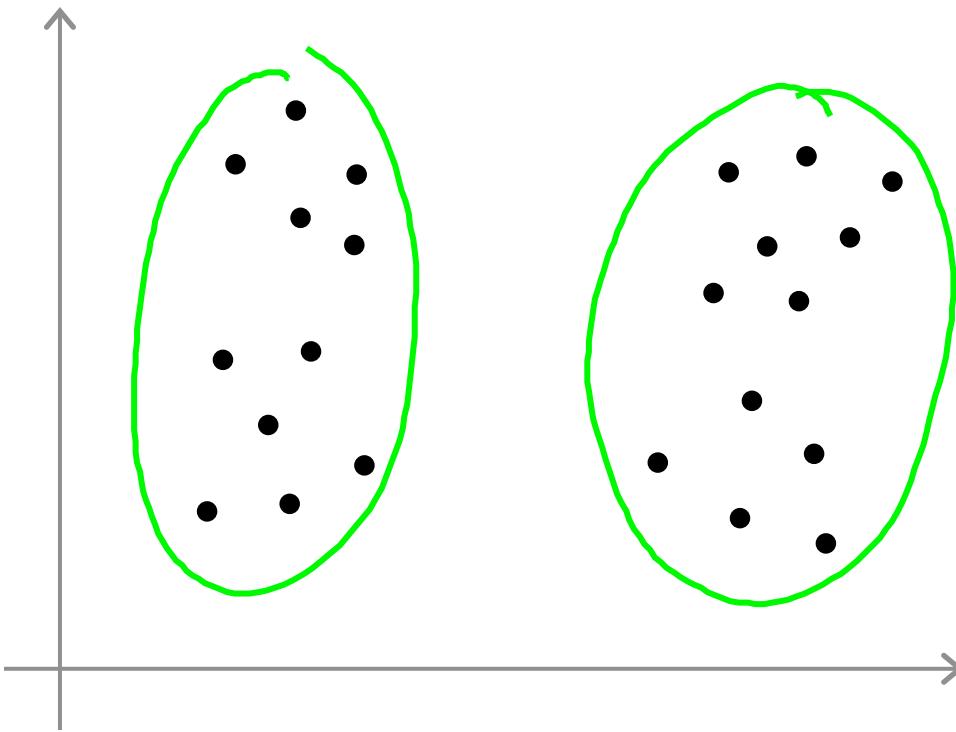


Machine Learning

# Clustering

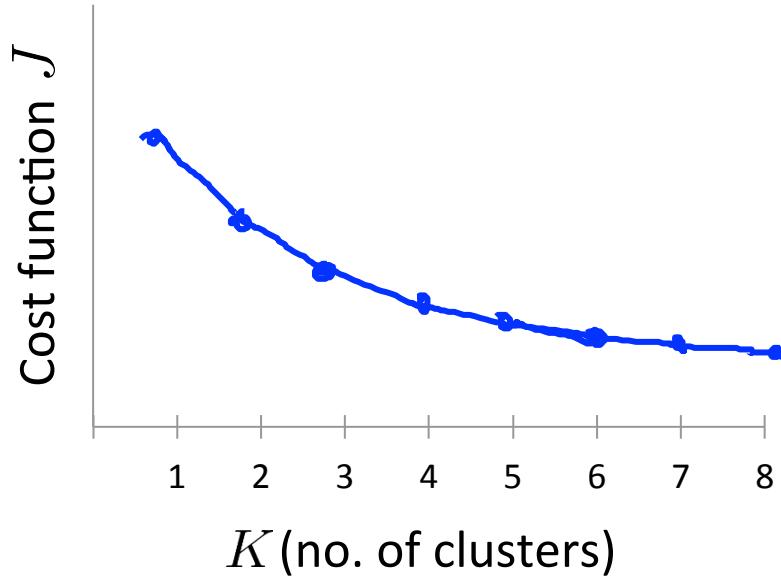
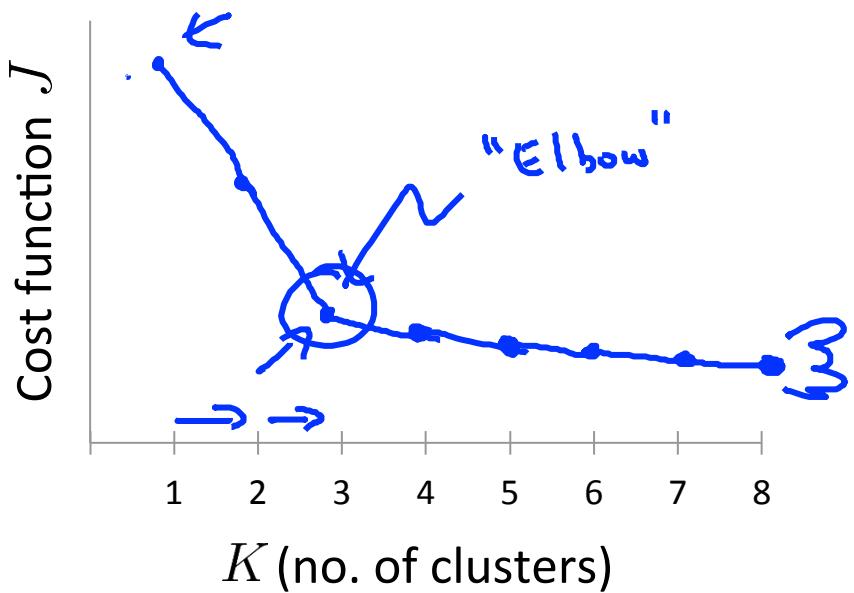
## Choosing the number of clusters

# What is the right value of K?



# Choosing the value of K

Elbow method:

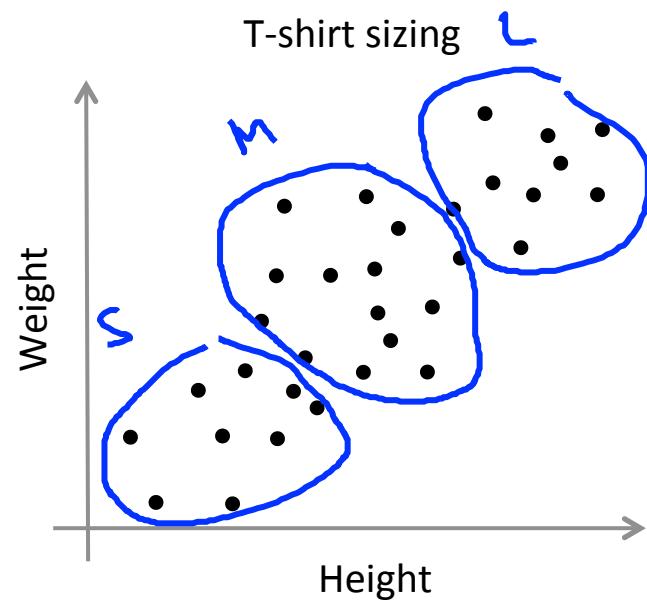


## Choosing the value of K

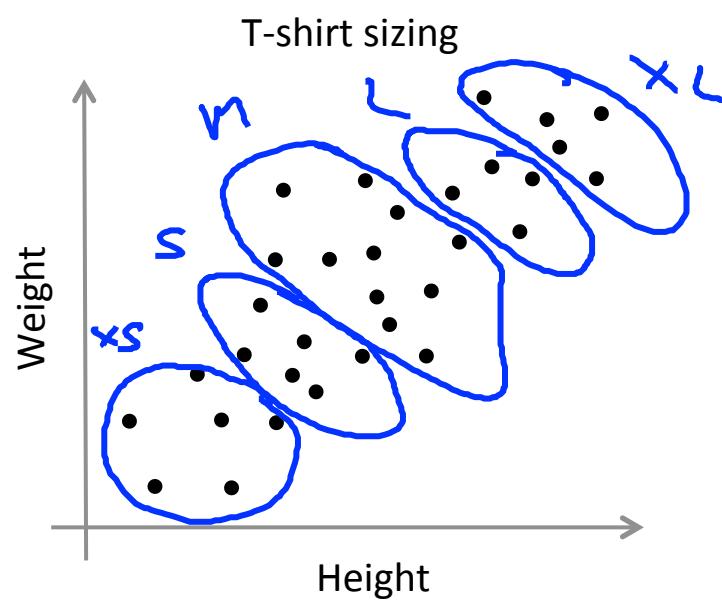
Sometimes, you're running K-means to get clusters to use for some later/downstream purpose. Evaluate K-means based on a metric for how well it performs for that later purpose.

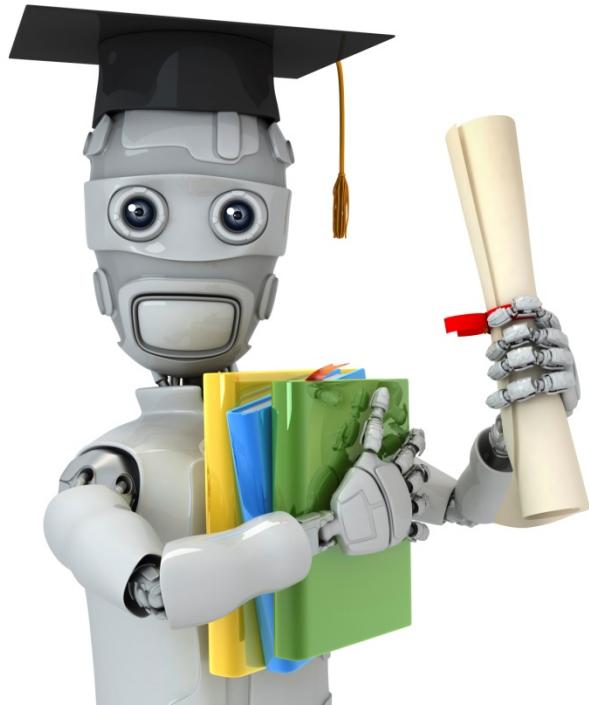
$K=3$       S, M, L

E.g.



$K=5$       XS, S, M, L, XL





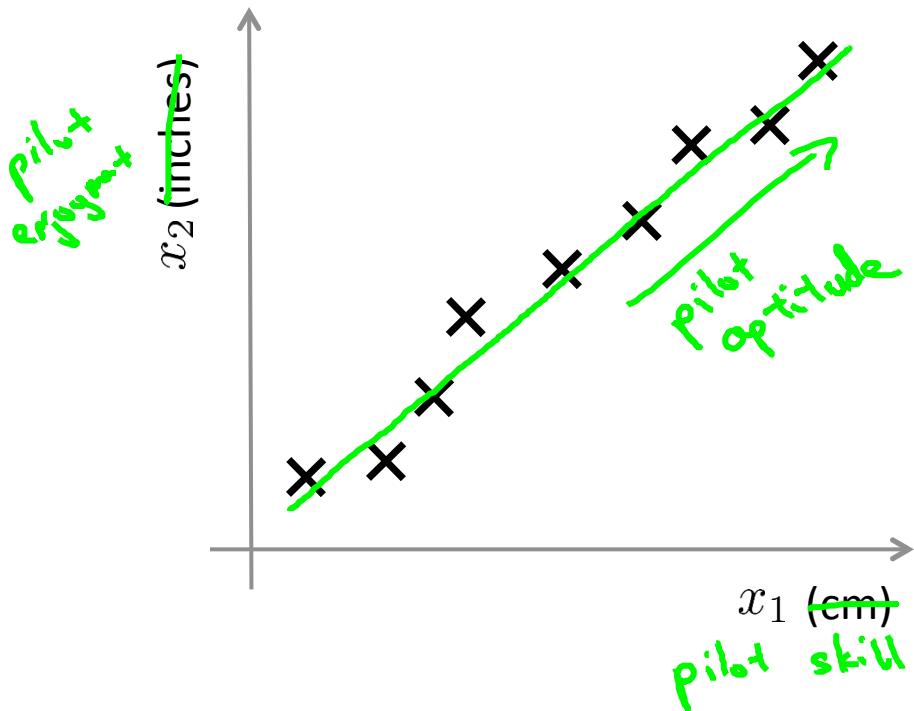
Machine Learning

# Dimensionality Reduction

---

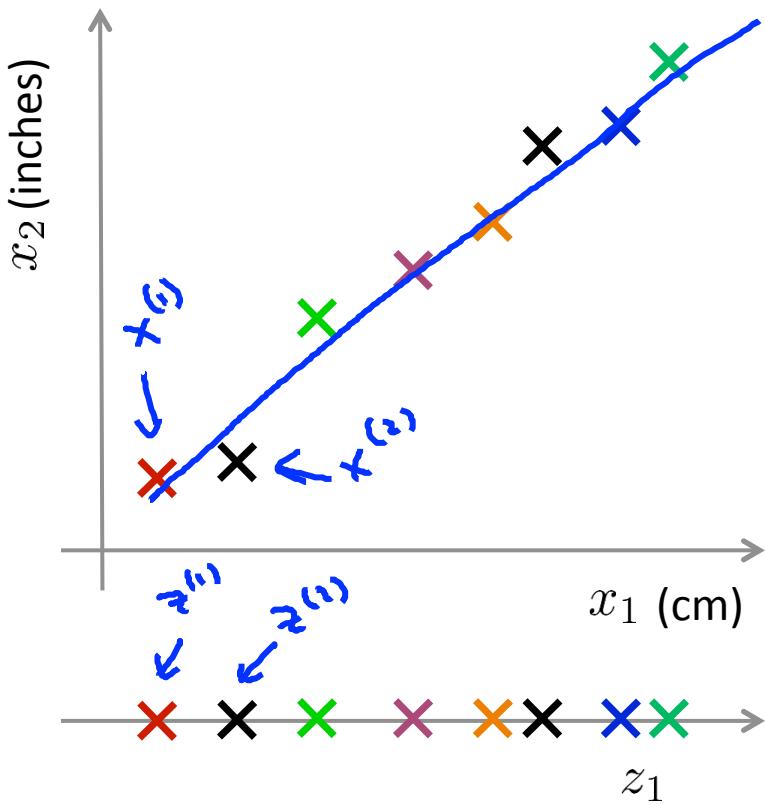
## Motivation I: Data Compression

# Data Compression



Reduce data from  
2D to 1D

# Data Compression



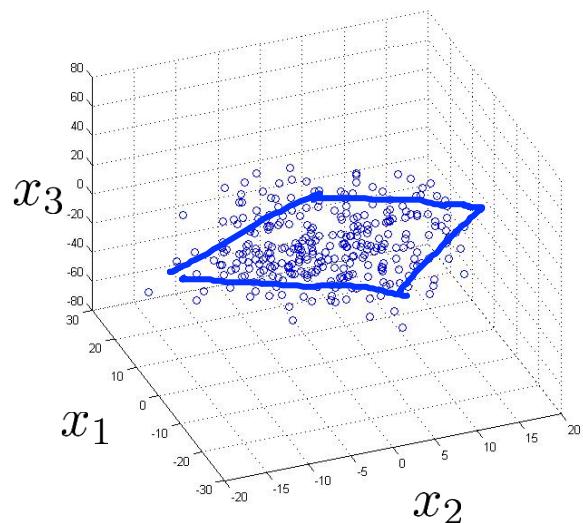
Reduce data from  
2D to 1D

$$\begin{aligned}x^{(1)} \in \mathbb{R}^2 &\rightarrow z^{(1)} \in \mathbb{R} \\x^{(2)} \in \mathbb{R}^2 &\rightarrow z^{(2)} \in \mathbb{R} \\&\vdots \\x^{(m)} \in \mathbb{R}^2 &\rightarrow z^{(m)} \in \mathbb{R}\end{aligned}$$

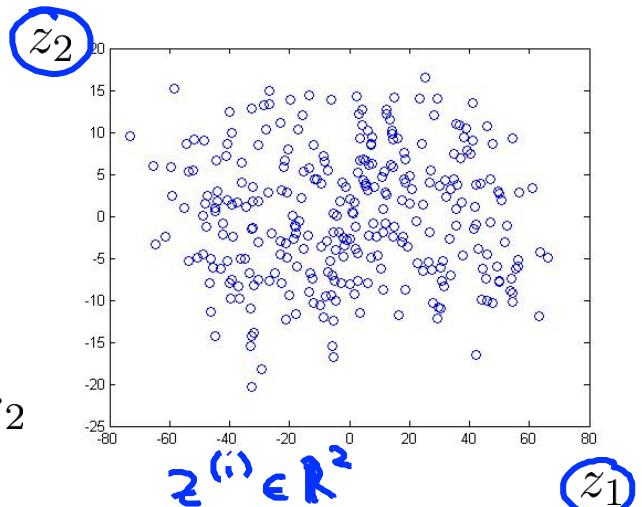
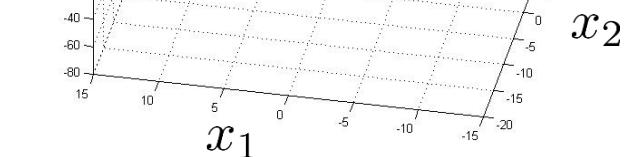
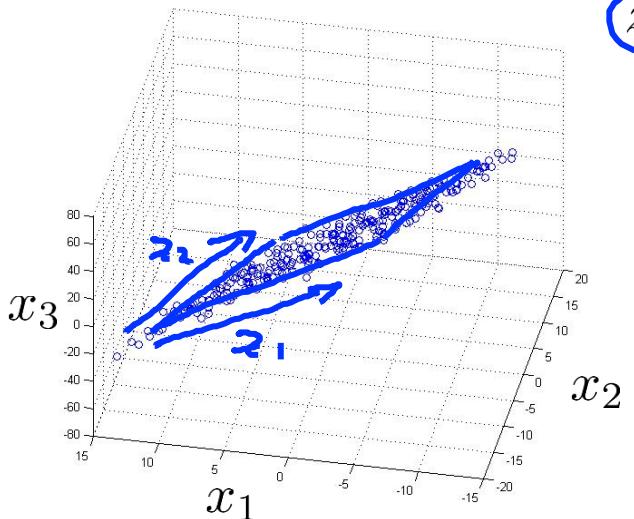
# Data Compression

10000  $\rightarrow$  1000

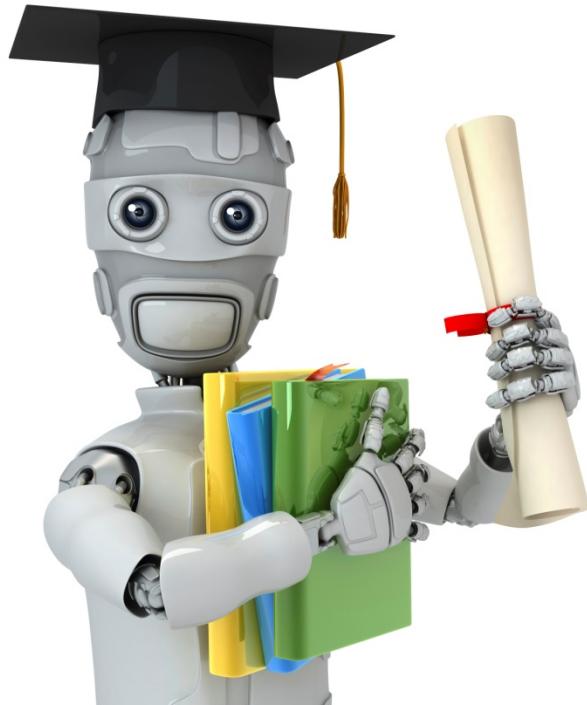
Reduce data from 3D to 2D



$$x^{(1)} \in \mathbb{R}^3$$



$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \quad z^{(1)} = \begin{bmatrix} z_1^{(1)} \\ z_2^{(1)} \end{bmatrix}$$



Machine Learning

# Dimensionality Reduction

---

## Motivation II: Data Visualization

# Data Visualization

$$x \in \mathbb{R}^{50}$$

$$x^{(i)} \in \mathbb{R}^{50}$$

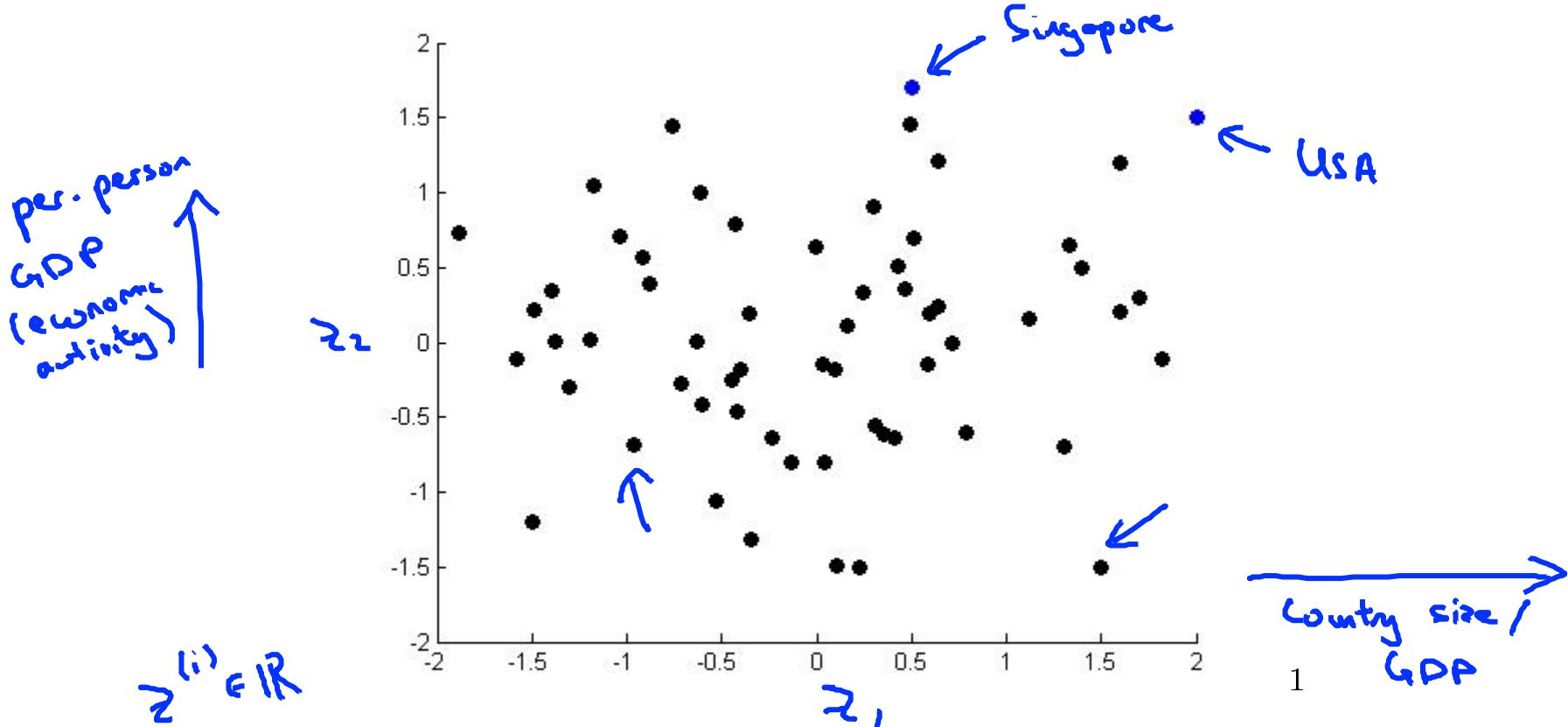
$x_6$

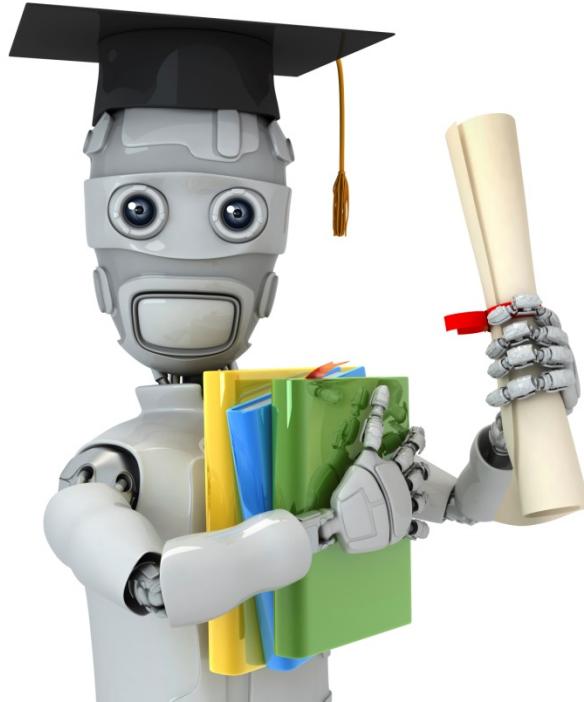
Country	$x_1$ GDP (trillions of US\$)	$x_2$ Per capita GDP (thousands of intl. \$)	$x_3$ Human Develop- ment Index	$x_4$ Life expectancy	$x_5$ Poverty Index (Gini as percentage)	Mean household income (thousands of US\$)	...
Canada	1.577	39.17	0.908	80.7	32.6	67.293	...
China	5.878	7.54	0.687	73	46.9	10.22	...
India	1.632	3.41	0.547	64.7	36.8	0.735	...
Russia	1.48	19.84	0.755	65.5	39.9	0.72	...
Singapore	0.223	56.69	0.866	80	42.5	67.1	...
USA	14.527	46.86	0.91	78.3	40.8	84.3	...
...	...	...	...	...	...	...	...

# Data Visualization

Country	$z_1$	$z_2$	$z^{(i)} \in \mathbb{R}^2$
Canada	1.6	1.2	
China	1.7	0.3	Reduce data
India	1.6	0.2	from 500
Russia	1.4	0.5	to 2D
Singapore	0.5	1.7	
USA	2	1.5	
...	...	...	

# Data Visualization





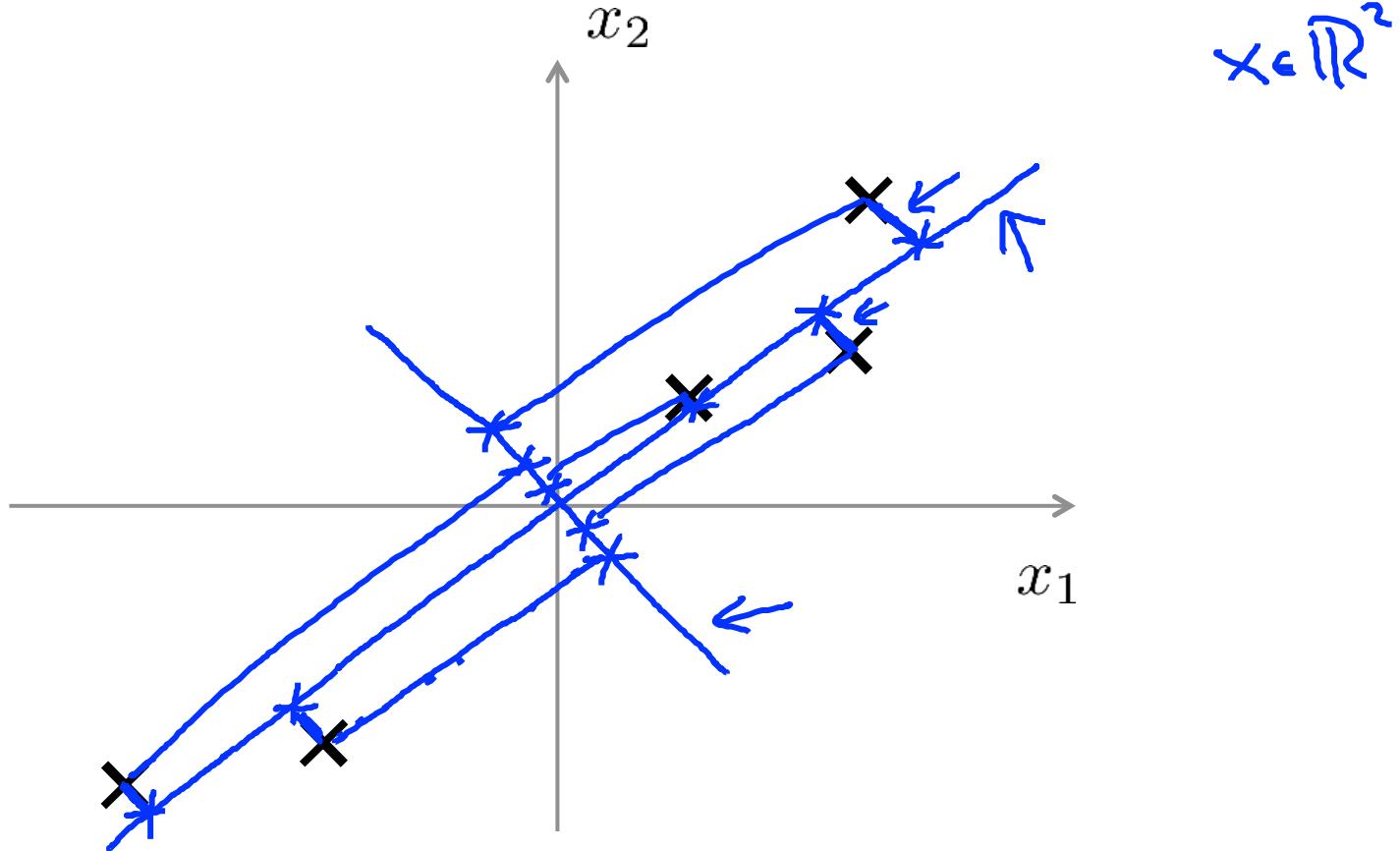
Machine Learning

# Dimensionality Reduction

---

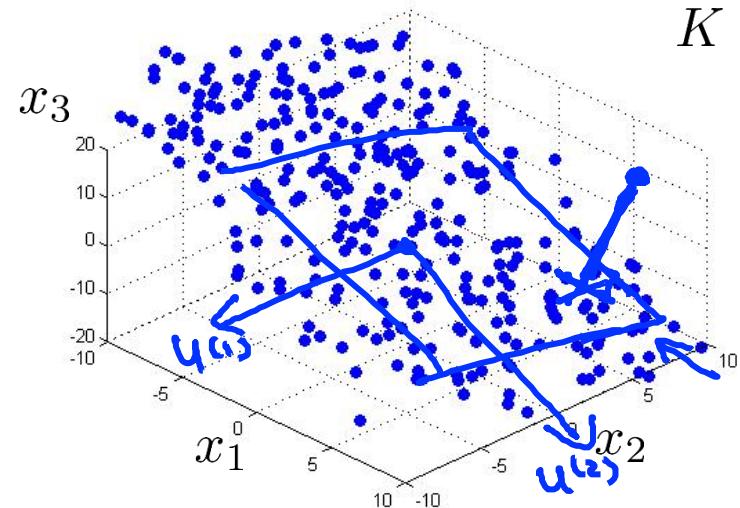
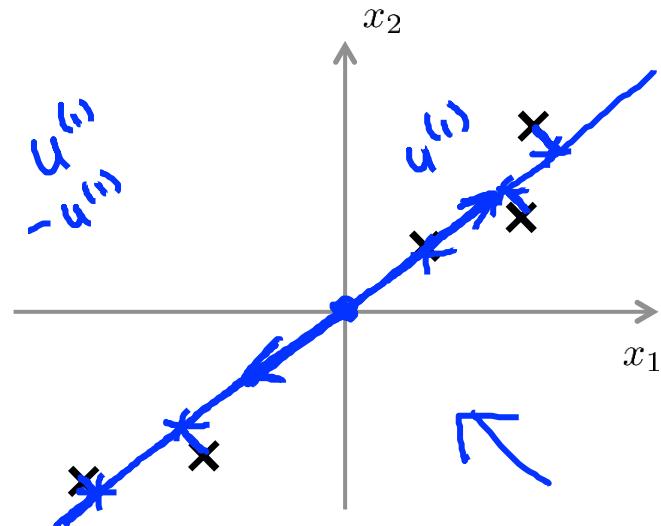
Principal Component  
Analysis problem  
formulation

# Principal Component Analysis (PCA) problem formulation



## Principal Component Analysis (PCA) problem formulation

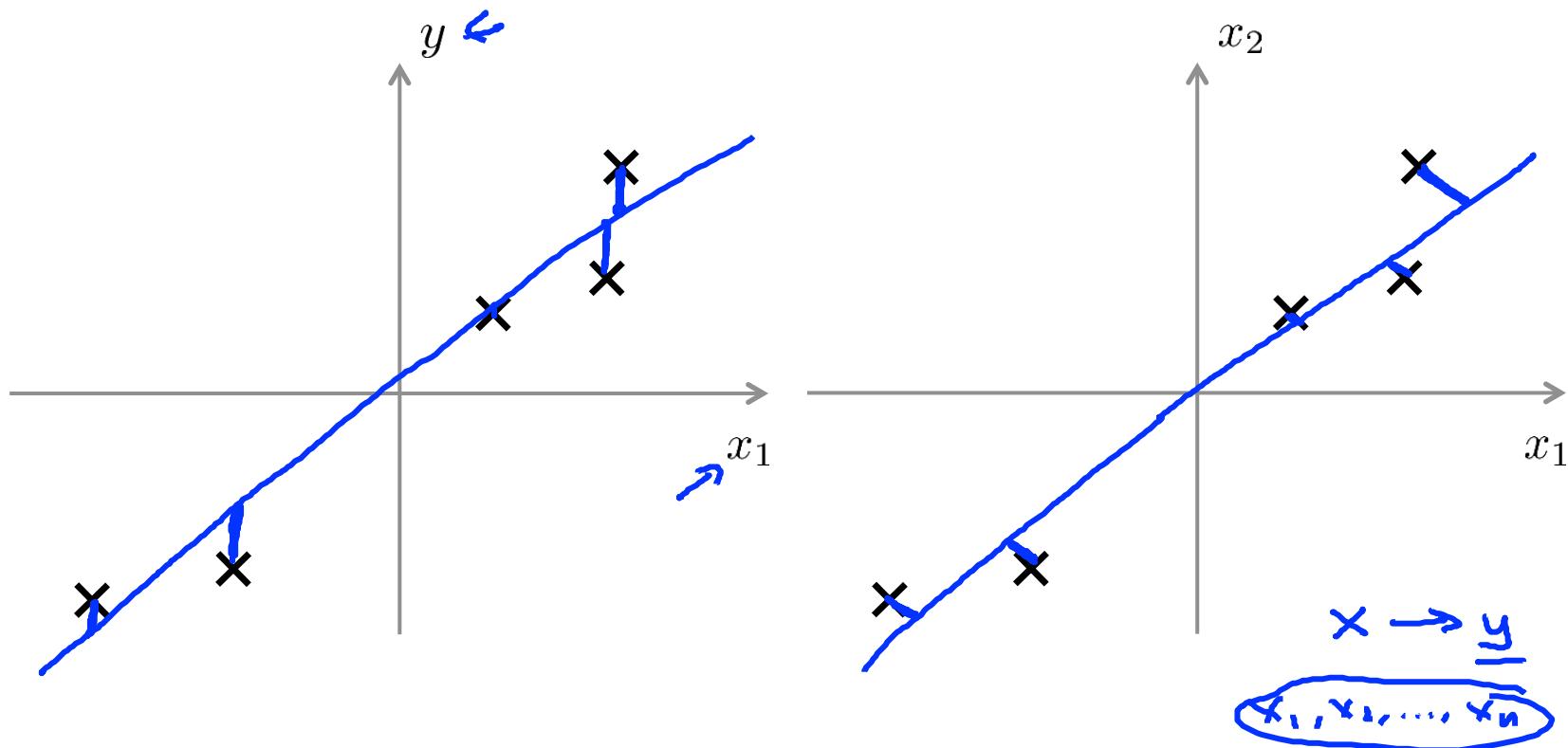
$$3D \rightarrow 2D \\ K = 2$$



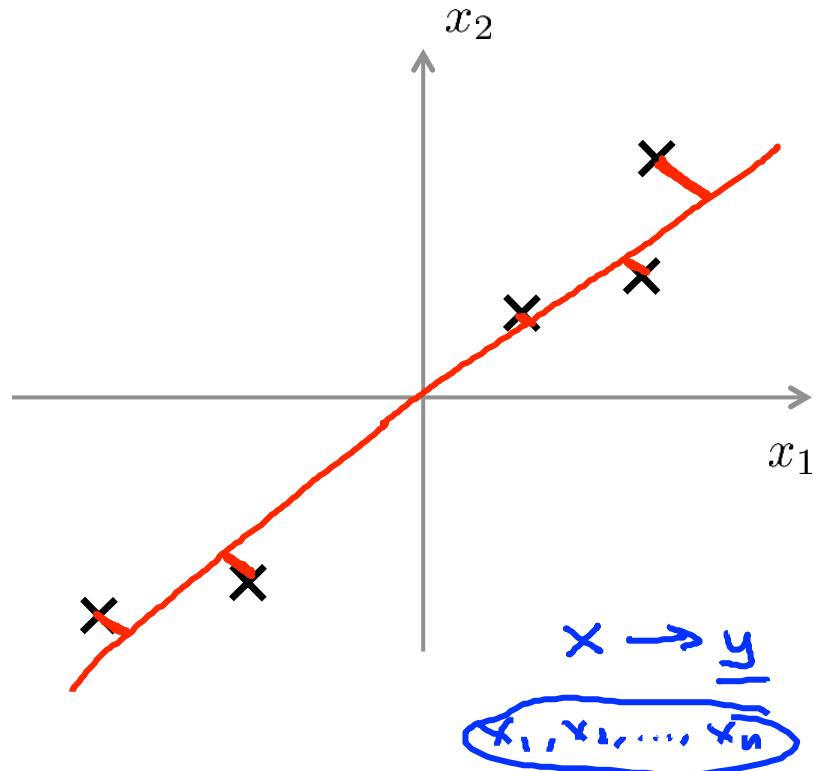
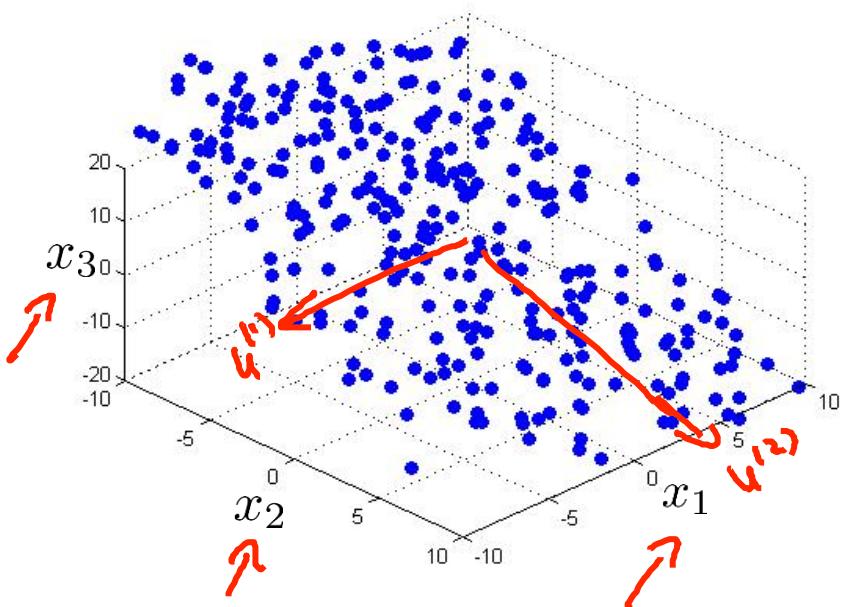
Reduce from 2-dimension to 1-dimension: Find a direction (a vector  $\underline{u^{(1)} \in \mathbb{R}^n}$ ) onto which to project the data so as to minimize the projection error.

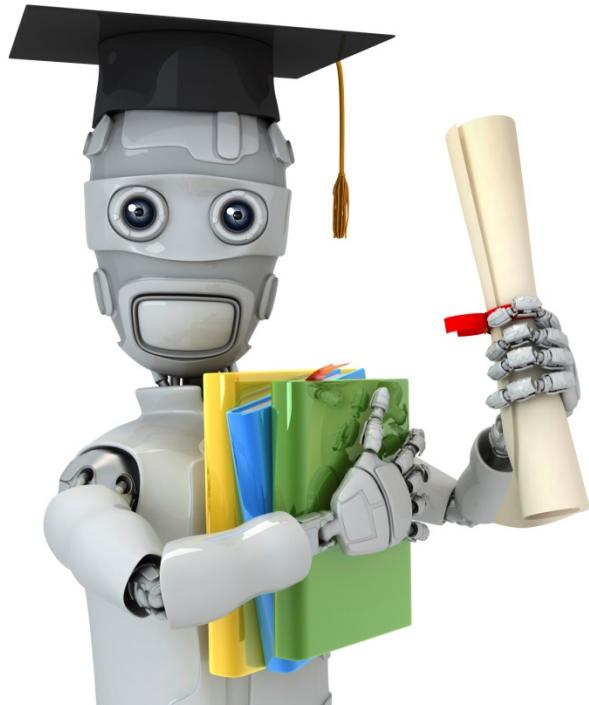
Reduce from  $n$ -dimension to  $k$ -dimension: Find  $k$  vectors  $\underline{u^{(1)}, u^{(2)}, \dots, u^{(k)}}$  onto which to project the data, so as to minimize the projection error.

# PCA is not linear regression



# PCA is not linear regression





Machine Learning

# Dimensionality Reduction

---

Principal Component  
Analysis algorithm

## Data preprocessing

Training set:  $x^{(1)}, x^{(2)}, \dots, x^{(m)}$  

Preprocessing (feature scaling/mean normalization):

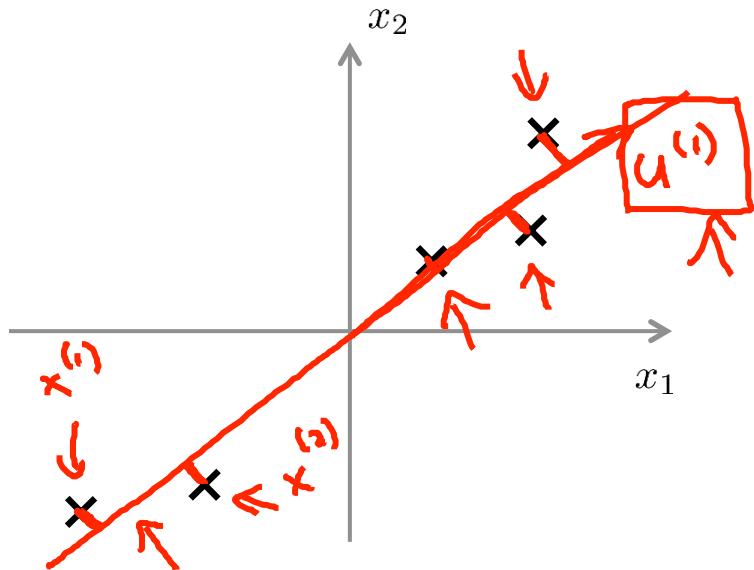
$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}$$

Replace each  $x_j^{(i)}$  with  $\underline{x_j - \mu_j}$ .

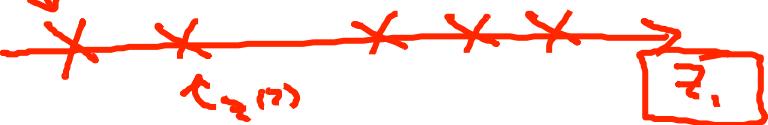
If different features on different scales (e.g.,  $x_1$  = size of house,  $x_2$  = number of bedrooms), scale features to have comparable range of values.

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$$

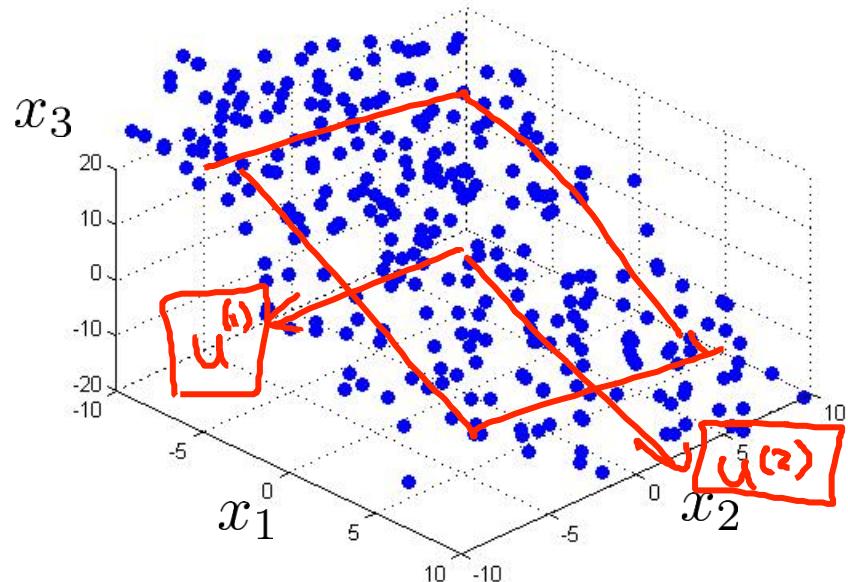
# Principal Component Analysis (PCA) algorithm



Reduce data from 2D to 1D

$$x^{(i)} \in \mathbb{R}^2 \rightarrow z^{(i)} \in \underline{\mathbb{R}}$$


A horizontal line representing the 1D principal component  $z^{(i)}$ . Five red 'x' marks, representing the projections of the original data points  $x^{(i)}$ , are placed along this line.



Reduce data from 3D to 2D

$$x^{(i)} \in \mathbb{R}^3 \rightarrow z^{(i)} \in \mathbb{R}^2$$
$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

# Principal Component Analysis (PCA) algorithm

Reduce data from  $n$ -dimensions to  $\underline{k}$ -dimensions

Compute "covariance matrix":

$$\Sigma = \frac{1}{m} \sum_{i=1}^n (x^{(i)}) (x^{(i)})^T$$

$n \times 1$        $1 \times n$

n × n

Sigma

Compute "eigenvectors" of matrix  $\Sigma$ :

$$\rightarrow [U, S, V] = \underline{\text{svd}}(\text{Sigma}) ;$$

→ Singular value decomposition  
eig(Sigma)

n × n matrix

$$U = \begin{bmatrix} | & | & | & | \\ u^{(1)} & u^{(2)} & u^{(3)} & \dots & u^{(m)} \\ | & | & | & & | \end{bmatrix}$$

k

$U \in \mathbb{R}^{n \times n}$

$u^{(1)}, \dots, u^{(k)}$

# Principal Component Analysis (PCA) algorithm

From  $[U, S, V] = \text{svd}(\Sigma)$ , we get:

$$\rightarrow U = \begin{bmatrix} u^{(1)} & u^{(2)} & \dots & u^{(n)} \end{bmatrix} \in \mathbb{R}^{n \times n}$$

$\underbrace{\phantom{u^{(1)} \quad u^{(2)} \quad \dots \quad u^{(n)}}_k}$

$$x \in \mathbb{R}^n \rightarrow z \in \mathbb{R}^k$$

$$z \in \mathbb{R}^k \quad z^{(i)} = \begin{bmatrix} u^{(1)} & u^{(2)} & \dots & u^{(k)} \end{bmatrix}^T$$

$n \times k$

U<sub>reduce</sub>

$$x^{(i)} = \begin{bmatrix} (u^{(1)})^T \\ \vdots \\ (u^{(k)})^T \end{bmatrix} \quad \begin{matrix} x^{(i)} \\ n \times 1 \end{matrix}$$

$k \times n$

$k \times 1$

# Principal Component Analysis (PCA) algorithm summary

- After mean normalization (ensure every feature has zero mean) and optionally feature scaling:

$$\text{Sigma} = \frac{1}{m} \sum_{i=1}^m (x^{(i)})(x^{(i)})^T$$

$$X = \begin{bmatrix} \vdots & & \vdots \\ x^{(1)\top} & \cdots & x^{(m)\top} \end{bmatrix}$$
$$\text{Sigma} = (1/m) * X' * X;$$

$$\rightarrow [U, S, V] = \text{svd}(\text{Sigma});$$

$$\rightarrow U_{\text{reduce}} = U(:, 1:k);$$

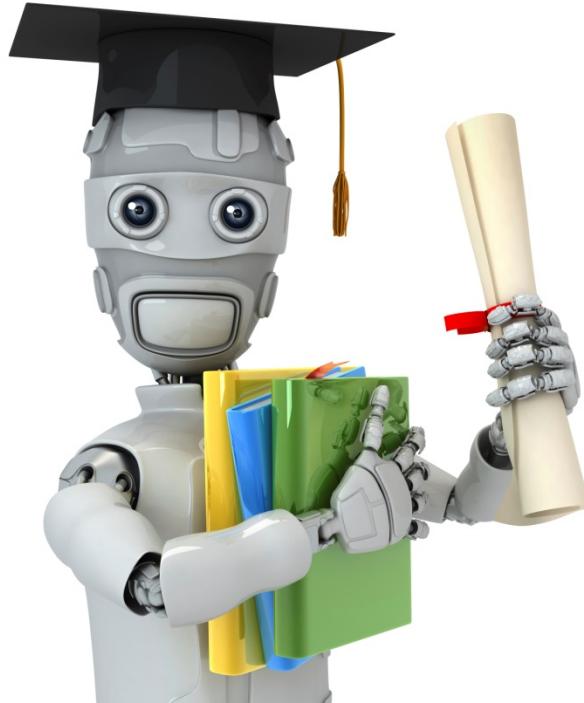
$$\rightarrow z = U_{\text{reduce}}' * x;$$

↑

↑

$$x \in \mathbb{R}^n$$

$$x \neq 1$$



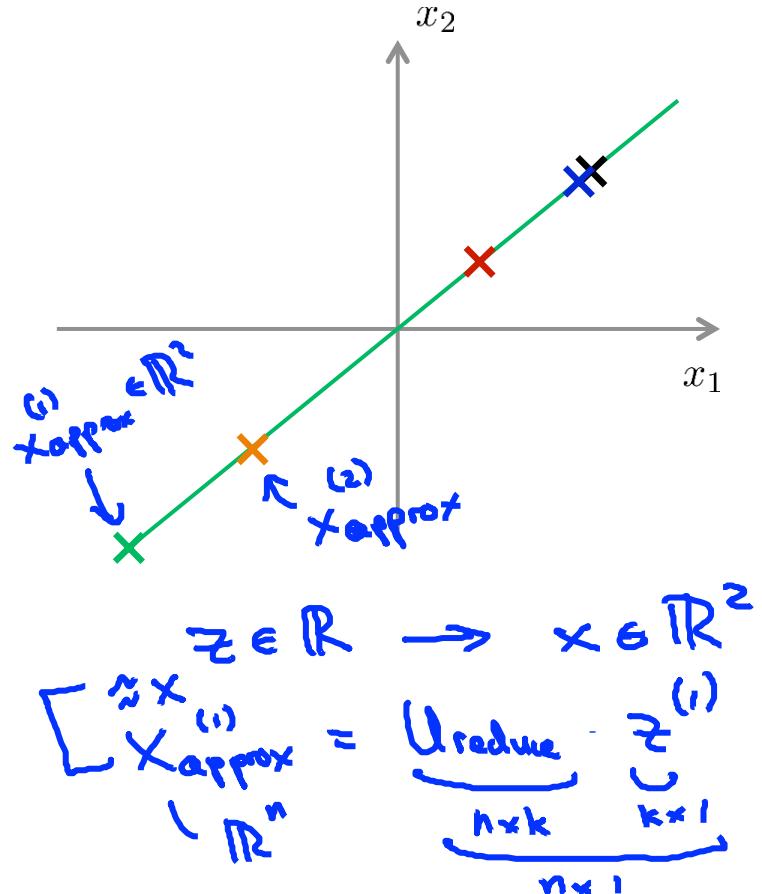
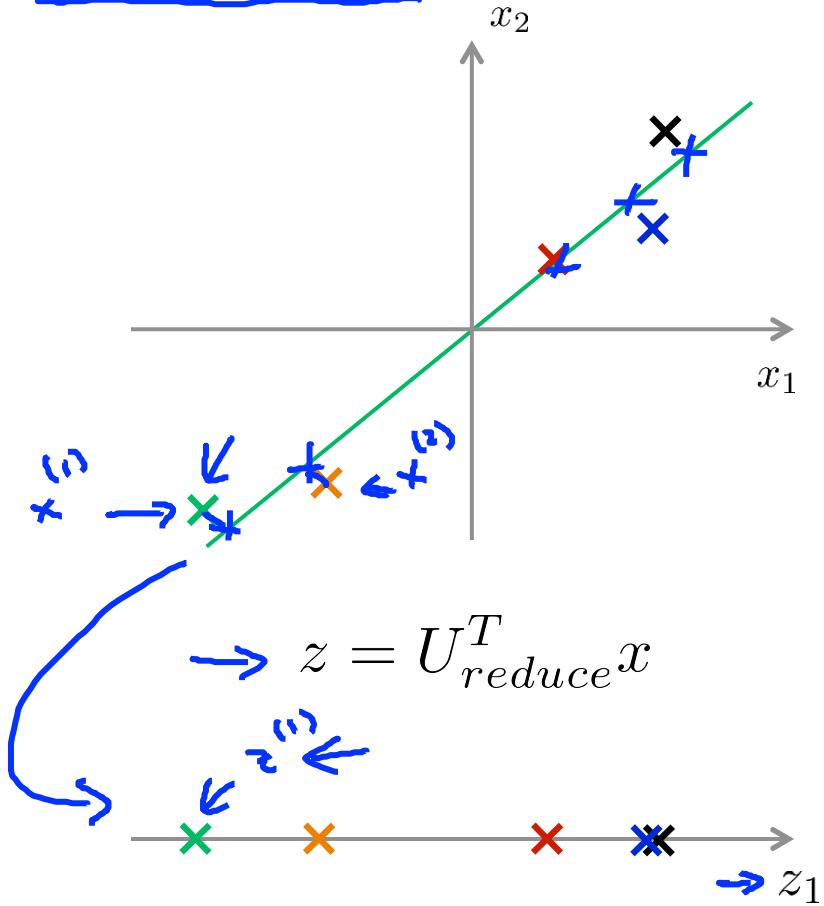
Machine Learning

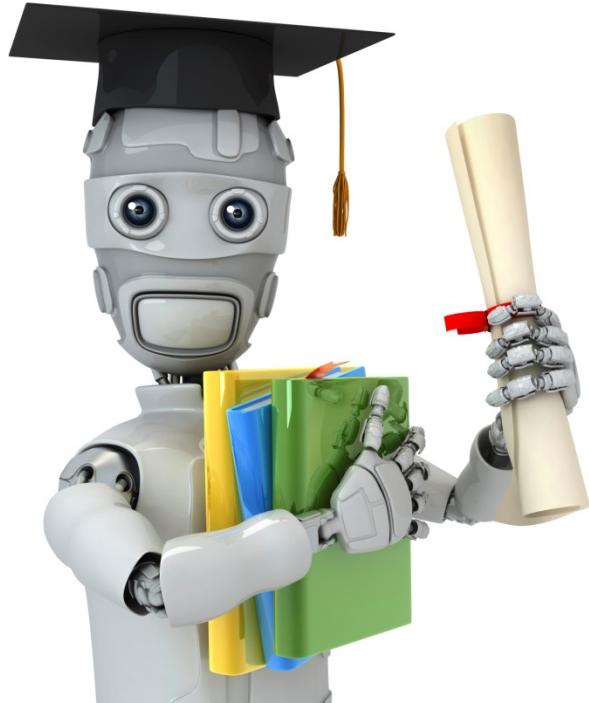
# Dimensionality Reduction

---

Reconstruction from  
compressed  
representation

## Reconstruction from compressed representation





Machine Learning

# Dimensionality Reduction

---

Choosing the number of principal components

## Choosing $k$ (number of principal components)

Average squared projection error:  $\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2$

Total variation in the data:  $\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2$

Typically, choose  $k$  to be smallest value so that

$$\rightarrow \frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} \leq 0.01$$

$$\frac{(1\%)}{\frac{0.05}{5\%}} \quad (10\%)$$

$\rightarrow$  "99% of variance is retained"  
~~95%~~ 90%

# Choosing $k$ (number of principal components)

Algorithm:

Try PCA with  $k = 1$   $\xrightarrow{k=2}$   $\xrightarrow{k=3}$   $\xrightarrow{k=4}$  ...

Compute  $U_{reduce}, z^{(1)}, z^{(2)}, \dots, z^{(m)}, x_{approx}^{(1)}, \dots, x_{approx}^{(m)}$

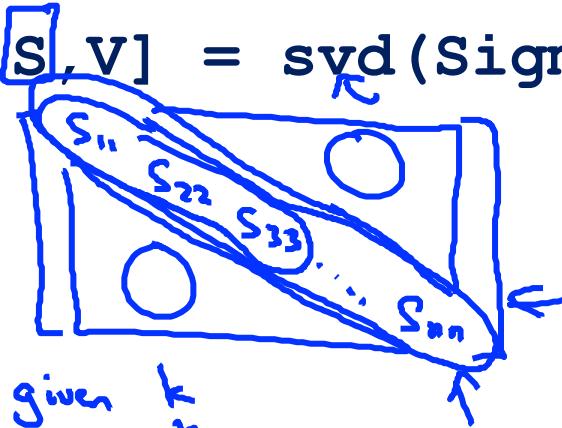
Check if

$$\frac{\frac{1}{m} \sum_{i=1}^m \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m} \sum_{i=1}^m \|x^{(i)}\|^2} \leq 0.01?$$

$k = 17$

$$\rightarrow [U, S, V] = svd(\Sigma)$$

$$\rightarrow \Sigma =$$



For given  $k$

$$1 - \frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^m S_{ii}} \leq 0.01$$

$$\rightarrow \frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^m S_{ii}} \geq 0.99$$

# Choosing $k$ (number of principal components)

→  $[U, S, V] = \text{svd}(\Sigma)$

Pick smallest value of  $k$  for which

$$\frac{\sum_{i=1}^k S_{ii}}{\sum_{i=1}^m S_{ii}} \geq 0.99$$

$k=100$

(99% of variance retained)

## Large data rationale

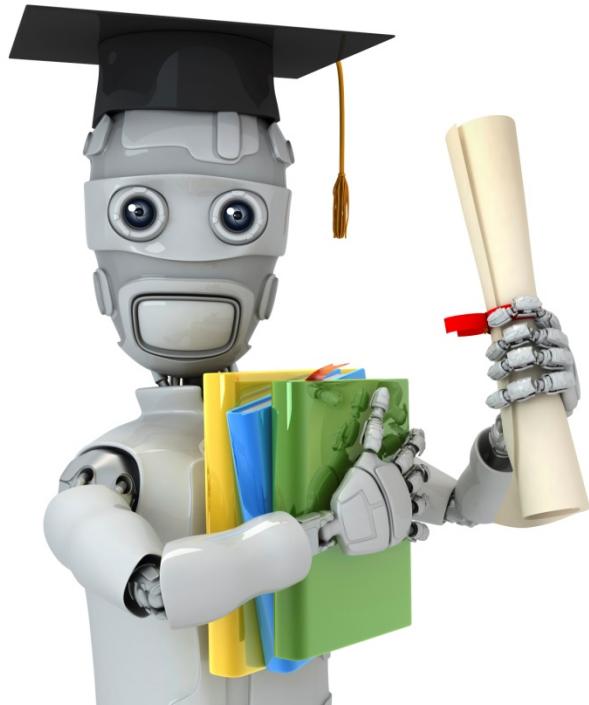
→ Use a learning algorithm with many parameters (e.g. logistic regression/linear regression with many features; neural network with many hidden units). low bias algorithms. ←

→  $J_{\text{train}}(\theta)$  will be small.

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

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

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



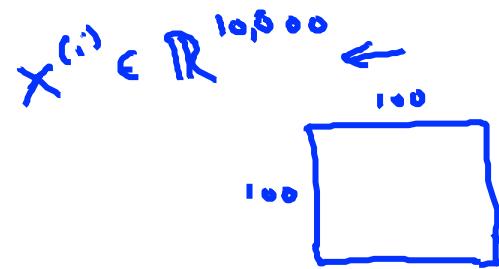
Machine Learning

# Dimensionality Reduction

---

## Advice for applying PCA

## Supervised learning speedup



→  $(\underline{x}^{(1)}, y^{(1)}), (\underline{x}^{(2)}, y^{(2)}), \dots, (\underline{x}^{(m)}, y^{(m)})$

Extract inputs:

Unlabeled dataset:  $\underline{x}^{(1)}, \underline{x}^{(2)}, \dots, \underline{x}^{(m)} \in \mathbb{R}^{10000}$

$\downarrow PCA$

$\underline{z}^{(1)}, \underline{z}^{(2)}, \dots, \underline{z}^{(m)} \in \mathbb{R}^{1000}$

$x \downarrow z$

New training set:

$(\underline{z}^{(1)}, y^{(1)}), (\underline{z}^{(2)}, y^{(2)}), \dots, (\underline{z}^{(m)}, y^{(m)})$

$$h_{\Theta}(z) = \frac{1}{1 + e^{-\Theta^T z}}$$

Note: Mapping  $x^{(i)} \rightarrow z^{(i)}$  should be defined by running PCA

only on the training set. This mapping can be applied as well to the examples  $x_{cv}^{(i)}$  and  $x_{test}^{(i)}$  in the cross validation and test sets.

# Application of PCA

- Compression
  - Reduce memory/disk needed to store data
  - Speed up learning algorithm ←

Choose  $k$  by % of variance retain

- Visualization

$k=2$  or  $k=3$

## Bad use of PCA: To prevent overfitting

→ Use  $z^{(i)}$  instead of  $x^{(i)}$  to reduce the number of features to  $k < n$ .

Thus, fewer features, less likely to overfit.

Bad!

This might work OK, but isn't a good way to address overfitting. Use regularization instead.

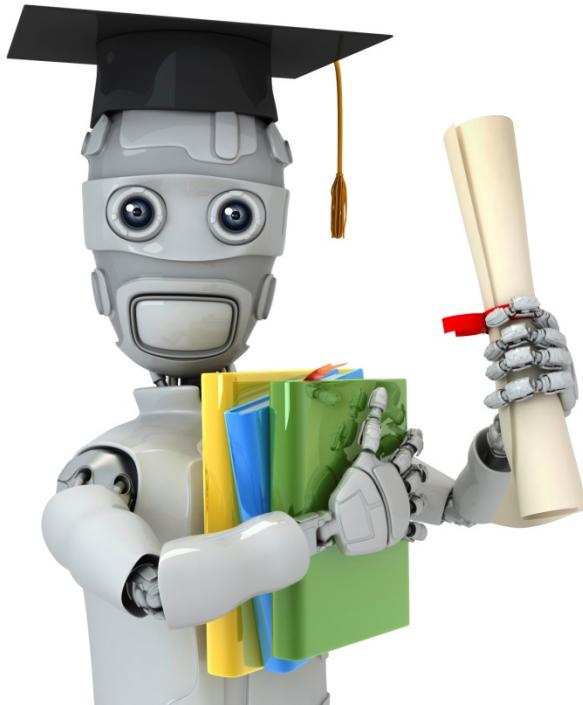
$$\rightarrow \min_{\theta} \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2 + \boxed{\frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2}$$

## PCA is sometimes used where it shouldn't be

Design of ML system:

- - Get training set  $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$
- - ~~Run PCA to reduce  $x^{(i)}$  in dimension to get  $z^{(i)}$~~
- - Train logistic regression on  $\{(z^{(1)}, y^{(1)}), \dots, (z^{(m)}, y^{(m)})\}$
- - Test on test set: Map  $x_{test}^{(i)}$  to  $z_{test}^{(i)}$ . Run  $h_\theta(z)$  on  $\{(z_{test}^{(1)}, y_{test}^{(1)}), \dots, (z_{test}^{(m)}, y_{test}^{(m)})\}$

- How about doing the whole thing without using PCA?
- Before implementing PCA, first try running whatever you want to do with the original/raw data  $x^{(i)}$ . Only if that doesn't do what you want, then implement PCA and consider using  $\underline{z^{(i)}}$ .



Machine Learning

# Anomaly detection

---

Problem  
motivation

## Anomaly detection example

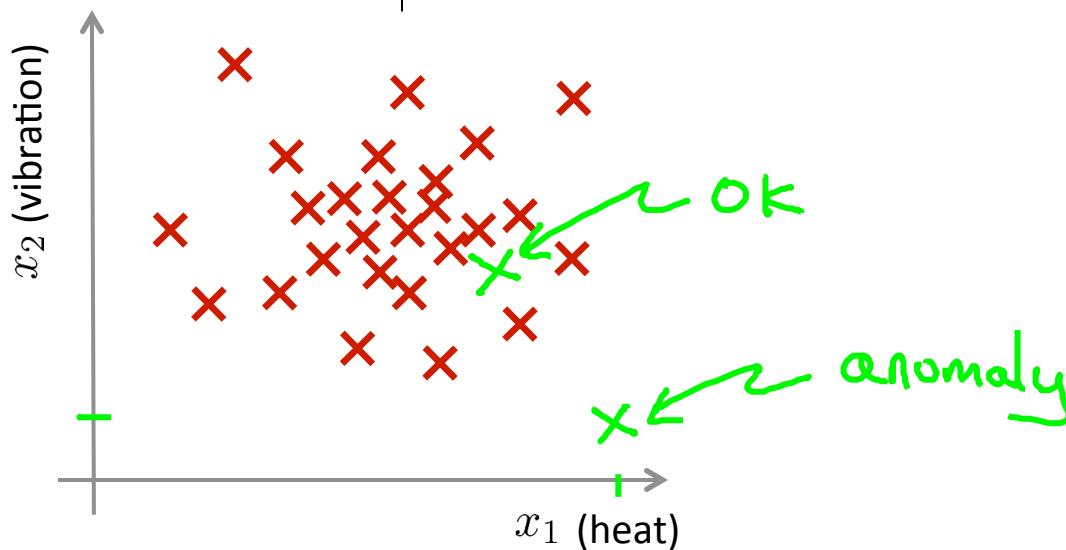
Aircraft engine features:

- $x_1$  = heat generated
- $x_2$  = vibration intensity

...

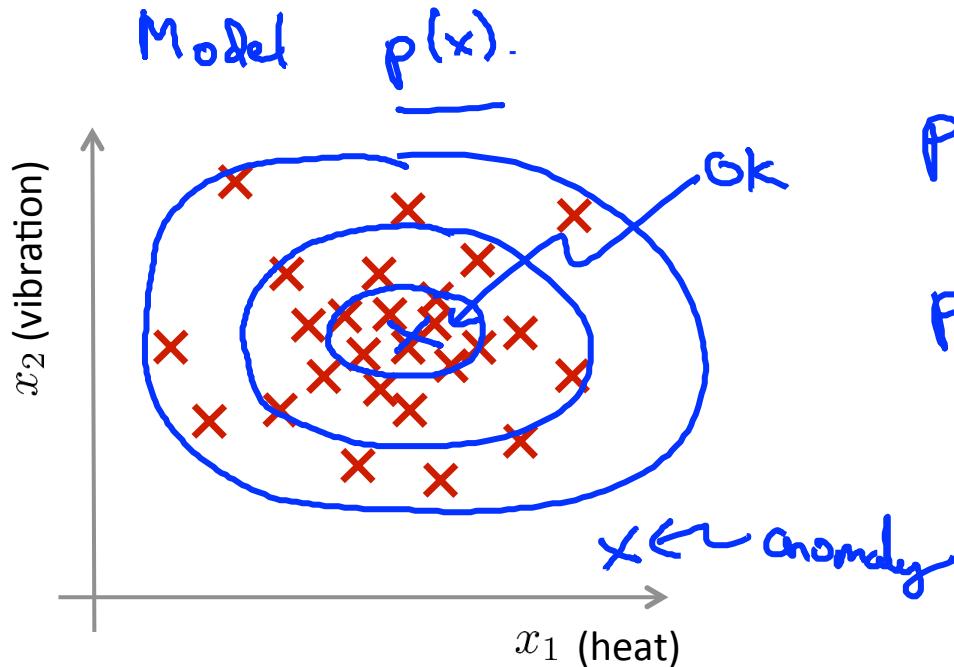
Dataset:  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

New engine:  $x_{test}$



# Density estimation

- Dataset:  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$
- Is  $x_{test}$  anomalous?



$p(x_{test}) < \varepsilon \rightarrow \text{flag anomaly}$

$p(x_{test}) \geq \varepsilon \rightarrow \text{OK}$

## Anomaly detection example

→ Fraud detection:

→  $x^{(i)}$  = features of user  $i$ 's activities

→ Model  $p(x)$  from data.

→ Identify unusual users by checking which have  $p(x) < \varepsilon$

$$\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \end{array} \quad p(x)$$

→ Manufacturing

→ Monitoring computers in a data center.

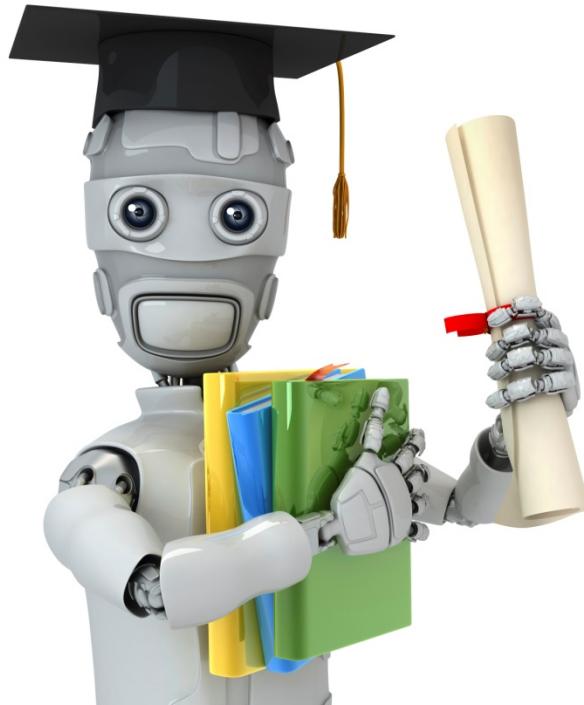
→  $x^{(i)}$  = features of machine  $i$

$x_1$  = memory use,  $x_2$  = number of disk accesses/sec,

$x_3$  = CPU load,  $x_4$  = CPU load/network traffic.

...

$$p(x) < \varepsilon$$



Machine Learning

# Anomaly detection

---

## Gaussian distribution

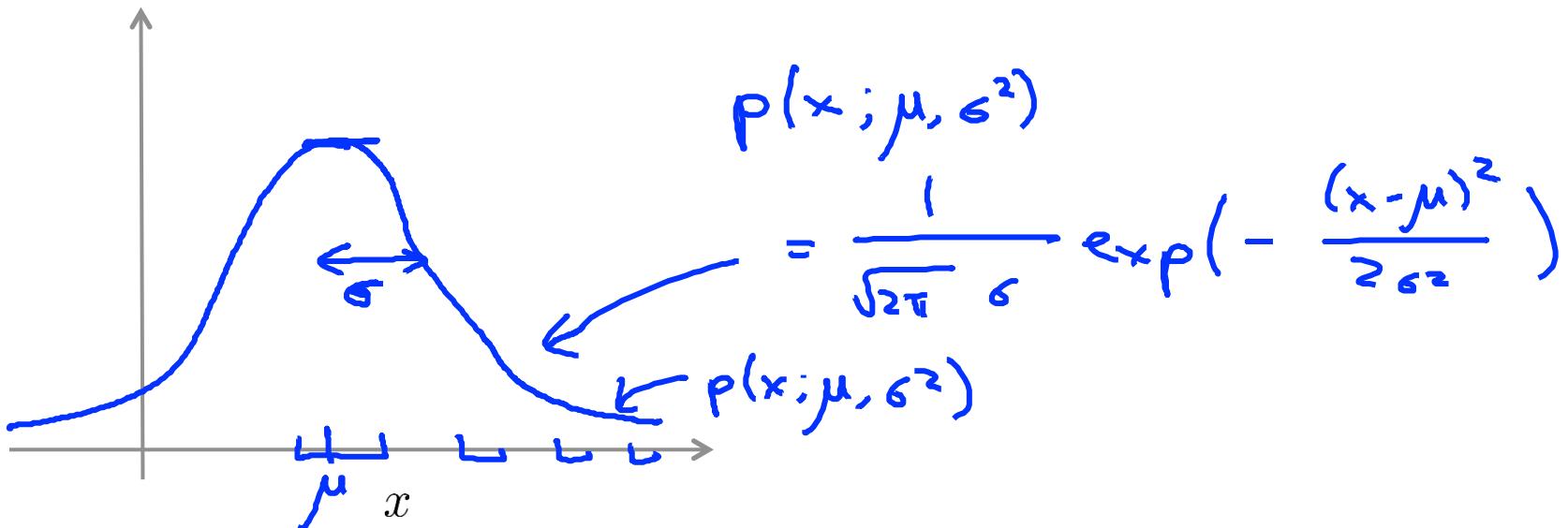
## Gaussian (Normal) distribution

Say  $x \in \mathbb{R}$ . If  $x$  is a distributed Gaussian with mean  $\mu$ , variance  $\sigma^2$ .

$$x \sim \mathcal{N}(\mu, \sigma^2)$$

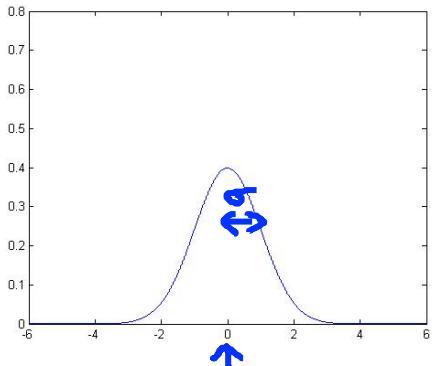
↖ "distributed as"

$\sigma$  standard deviation

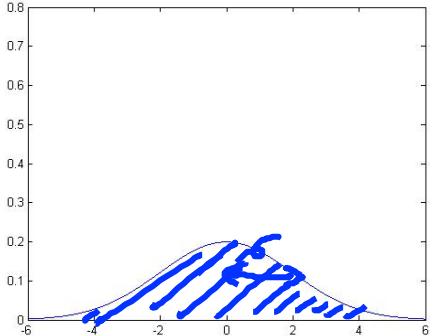


# Gaussian distribution example

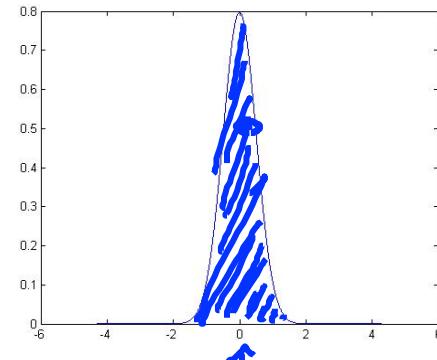
$$\rightarrow \mu = 0, \sigma = 1$$



$$\rightarrow \mu = 0, \sigma = 2$$

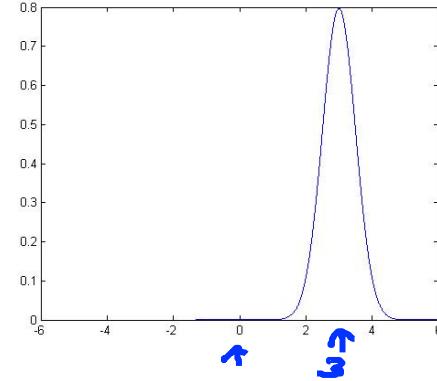


$$\rightarrow \mu = 0, \sigma = 0.5$$



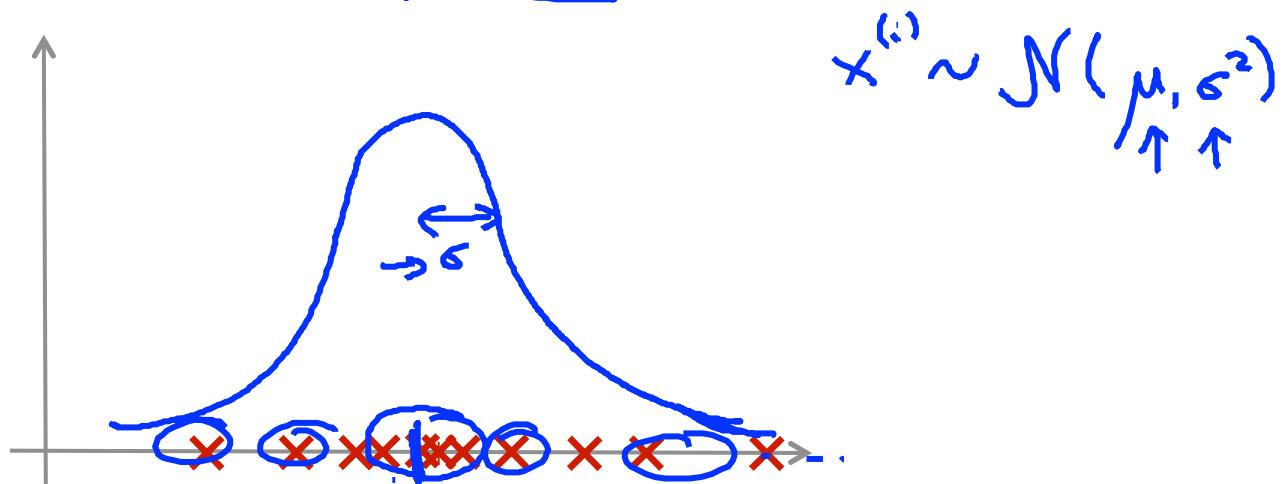
$$\zeta^2 = 0.25$$

$$\rightarrow \mu = 3, \sigma = 0.5$$



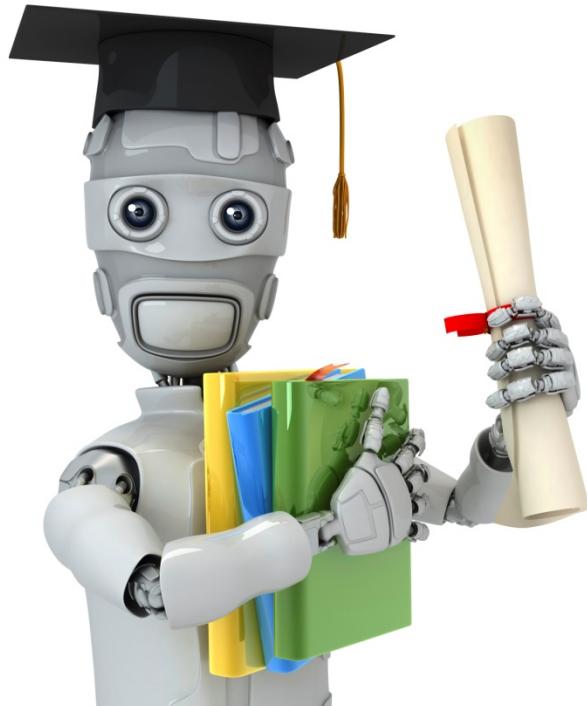
## Parameter estimation

→ Dataset:  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$   $x^{(i)} \in \mathbb{R}$



$$\Rightarrow \hat{\mu} = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$

$$\Rightarrow \hat{\sigma}^2 = \frac{1}{m-1} \sum_{i=1}^m (x^{(i)} - \hat{\mu})^2$$



Machine Learning

# Anomaly detection

---

## Algorithm

## → Density estimation

→ Training set:  $\{x^{(1)}, \dots, x^{(m)}\}$

Each example is  $x \in \mathbb{R}^n$

→  $p(x)$

$$= \boxed{p(x_1; \mu_1, \sigma_1^2) p(x_2; \mu_2, \sigma_2^2) p(x_3; \mu_3, \sigma_3^2) \dots p(x_n; \mu_n, \sigma_n^2)}$$

$$= \boxed{\prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2)}$$

$$x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$$

$$x_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$$

$$x_3 \sim \mathcal{N}(\mu_3, \sigma_3^2)$$

$$\sum_{i=1}^n i = 1+2+3+\dots+n$$

$$\prod_{i=1}^n i = 1 \times 2 \times 3 \times \dots \times n$$

# Anomaly detection algorithm

- 1. Choose features  $x_i$  that you think might be indicative of anomalous examples.

$$\{x^{(1)}, \dots, x^{(m)}\}$$

- 2. Fit parameters  $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$

$$\rightarrow \boxed{\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)}}$$

$$p(x_j; \mu_j, \sigma_j^2)$$

$$\rightarrow \boxed{\sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2}$$

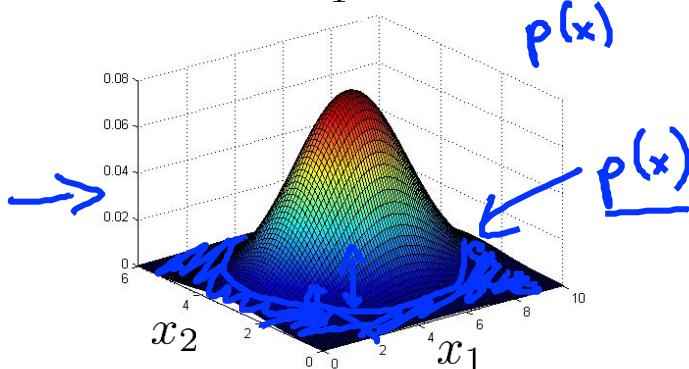
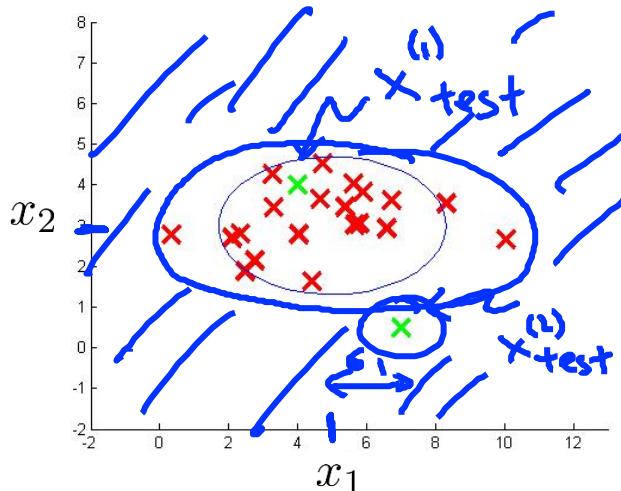
$$\mu_1, \mu_2, \dots, \mu_n$$
$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix} = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$

- 3. Given new example  $x$ , compute  $p(x)$ :

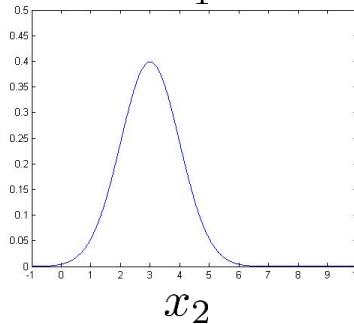
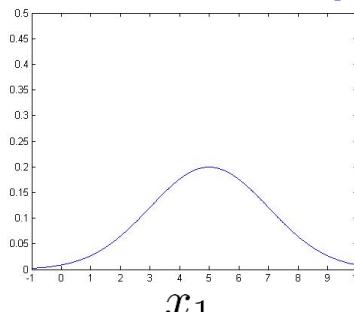
$$p(x) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^n \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

Anomaly if  $\underline{p(x) < \varepsilon}$

# Anomaly detection example



$$\rightarrow p(x) = p(x_1; \mu_1, \sigma_1^2) \times p(x_2; \mu_2, \sigma_2^2)$$



$$p(x_1; \mu_1, \sigma_1^2)$$

$$p(x_1; \mu_1, \sigma_1^2)$$

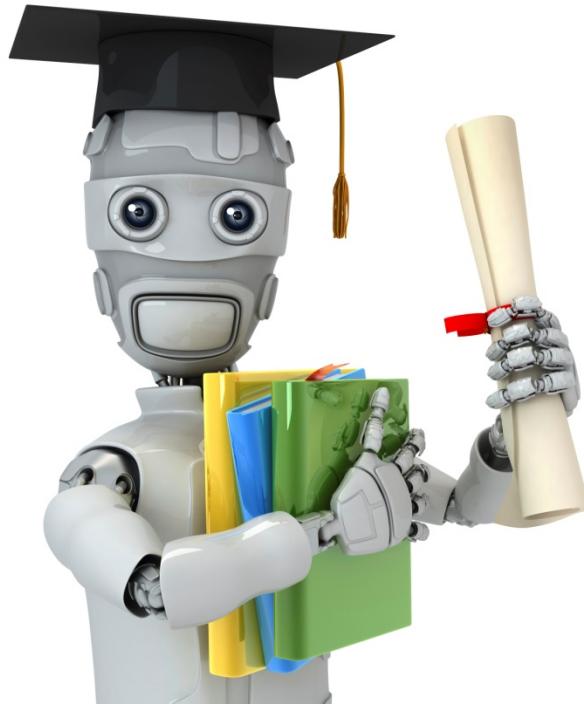


$$p(x_2; \mu_2, \sigma_2^2)$$

$$\varepsilon = 0.02$$

$$p(x_{test}^{(1)}) = 0.0426 \geq \varepsilon$$

$$p(x_{test}^{(2)}) = 0.0021 < \varepsilon$$



Machine Learning

# Anomaly detection

---

Developing and  
evaluating an anomaly  
detection system

## The importance of real-number evaluation

When developing a learning algorithm (choosing features, etc.), making decisions is much easier if we have a way of evaluating our learning algorithm.

- Assume we have some labeled data, of anomalous and non-anomalous examples. ( $y = 0$  if normal,  $y = 1$  if anomalous).
- Training set:  $x^{(1)}, x^{(2)}, \dots, x^{(m)}$  (assume normal examples/not anomalous)
- Cross validation set:  $(x_{cv}^{(1)}, y_{cv}^{(1)}), \dots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$
- Test set:  $(x_{test}^{(1)}, y_{test}^{(1)}), \dots, (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

$$y=1$$

## Aircraft engines motivating example

- 10000 good (normal) engines
- 20 flawed engines (anomalous) 2 - 50 y = 1
- Training set: 6000 good engines ( $y = 0$ )  $p(x) = p(x_1; \mu_1, \sigma^2_1) \dots p(x_n; \mu_n, \sigma^2_n)$
- CV: 2000 good engines ( $y = 0$ ), 10 anomalous ( $y = 1$ )
- Test: 2000 good engines ( $y = 0$ ), 10 anomalous ( $y = 1$ )

Alternative:

Training set: 6000 good engines

→ CV: 4000 good engines ( $y = 0$ ), 10 anomalous ( $y = 1$ )

→ Test: 4000 good engines ( $y = 0$ ), 10 anomalous ( $y = 1$ )

## Algorithm evaluation

- Fit model  $p(x)$  on training set  $\{x^{(1)}, \dots, x^{(m)}\}$
- On a cross validation/test example  $x$ , predict

$(x_{\text{test}}^{(i)}, y_{\text{test}}^{(i)})$



$$y = \begin{cases} 1 & \text{if } p(x) < \varepsilon \text{ (anomaly)} \\ 0 & \text{if } p(x) \geq \varepsilon \text{ (normal)} \end{cases}$$

$y=0$

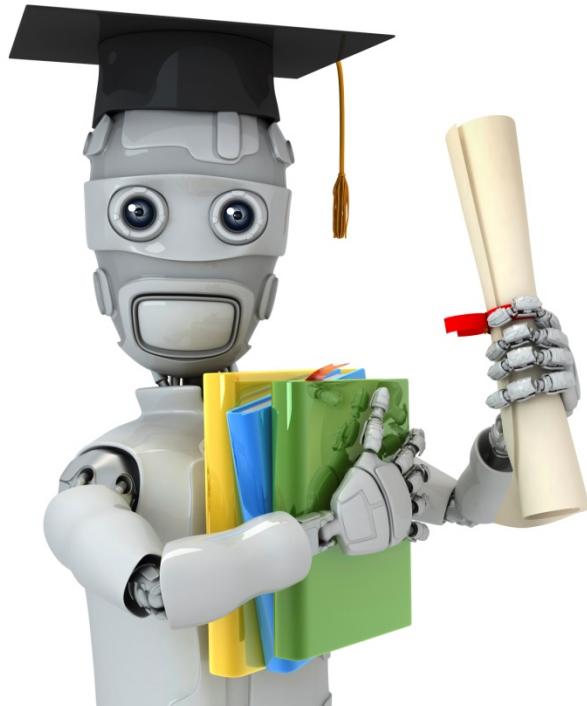
Possible evaluation metrics:

- - True positive, false positive, false negative, true negative
- - Precision/Recall
- -  $F_1$ -score

CV

Test set

Can also use cross validation set to choose parameter  $\varepsilon$



Machine Learning

# Anomaly detection

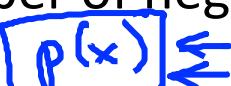
---

Anomaly detection  
vs. supervised  
learning

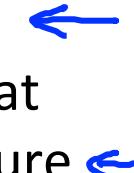
## Anomaly detection

vs.

## Supervised learning

- Very small number of positive examples ( $y = 1$ ). (0-20 is common).
- Large number of negative ( $y = 0$ ) examples. 
- Many different “types” of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like;
- future anomalies may look nothing like any of the anomalous examples we've seen so far.

Large number of positive and negative examples. 

Enough positive examples for algorithm to get a sense of what positive examples are like, future positive examples likely to be similar to ones in training set. 

Spam 

## Anomaly detection

- • Fraud detection  $y=1$
- • Manufacturing (e.g. aircraft engines)
- • Monitoring machines in a data center

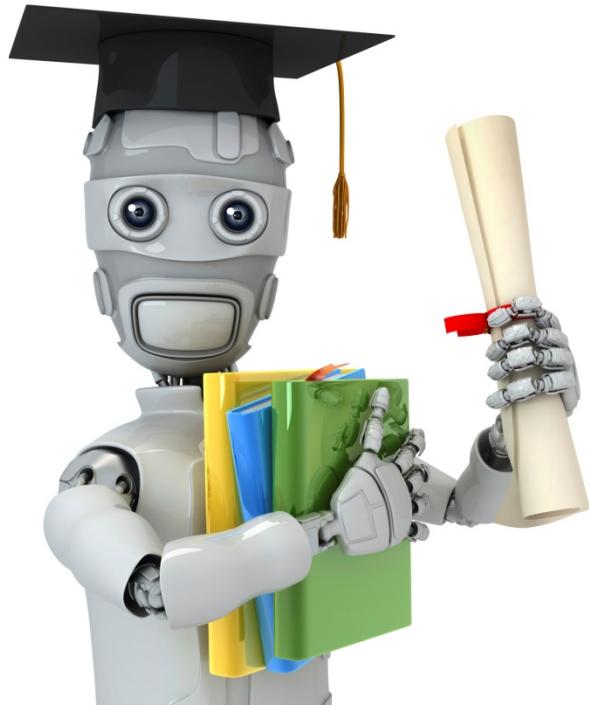
vs.

## Supervised learning

- Email spam classification ←
- Weather prediction (sunny/rainy/etc).
- Cancer classification ←

:

:



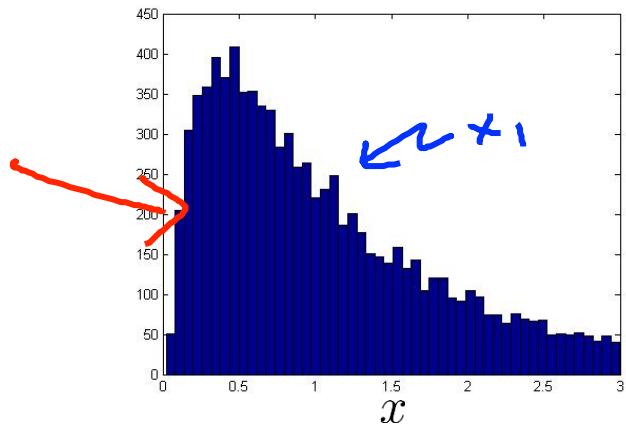
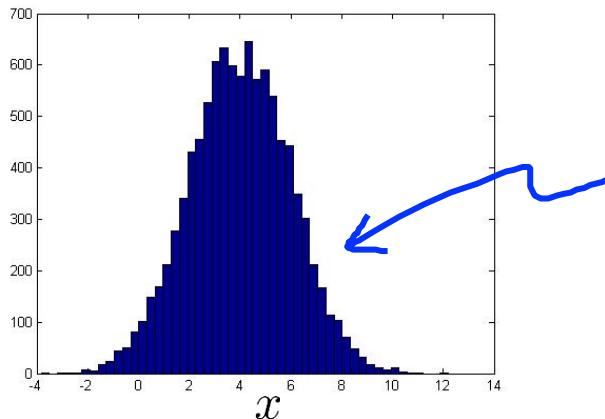
Machine Learning

# Anomaly detection

---

Choosing what features to use

# Non-gaussian features



$p(x_i; \mu_i, \sigma_i^2)$

hist

$x_1 \leftarrow \log(x_1)$

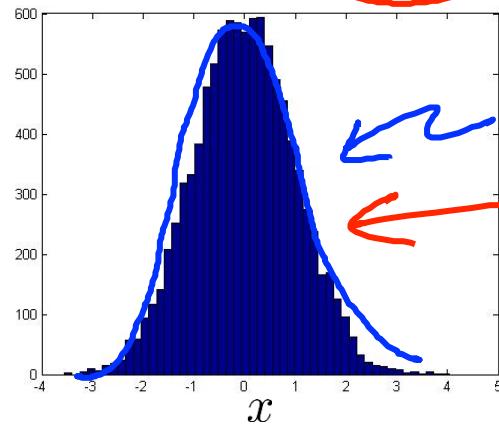
$x_2 \leftarrow \log(x_2 + 1)$

$x_3 \leftarrow \sqrt{x_3} = x_3^{1/2}$

$x_4 \leftarrow \frac{x_4}{x_4 + 1} =$

$\log(x_2 + 1)$

Diagram illustrating feature transformation. A red oval encloses four steps: 1.  $x_1 \leftarrow \log(x_1)$ , 2.  $x_2 \leftarrow \log(x_2 + 1)$ , 3.  $x_3 \leftarrow \sqrt{x_3} = x_3^{1/2}$ , and 4.  $x_4 \leftarrow \frac{x_4}{x_4 + 1} =$ . Blue arrows point from the original histograms to these transformed features.

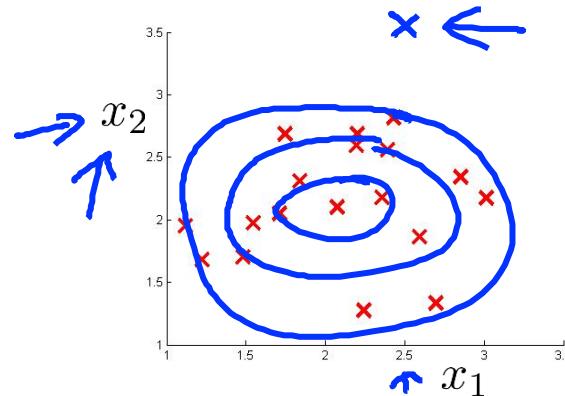
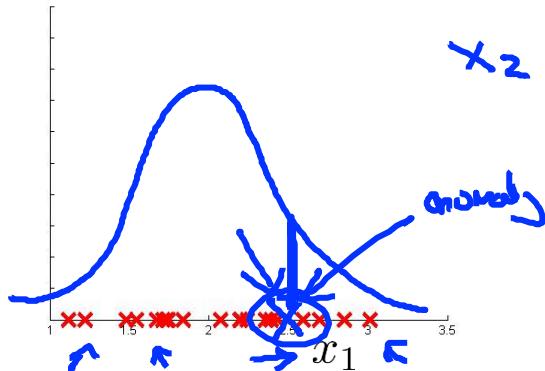


## → Error analysis for anomaly detection

Want  $p(x)$  large for normal examples  $x$ .  
 $p(x)$  small for anomalous examples  $x$ .

Most common problem:

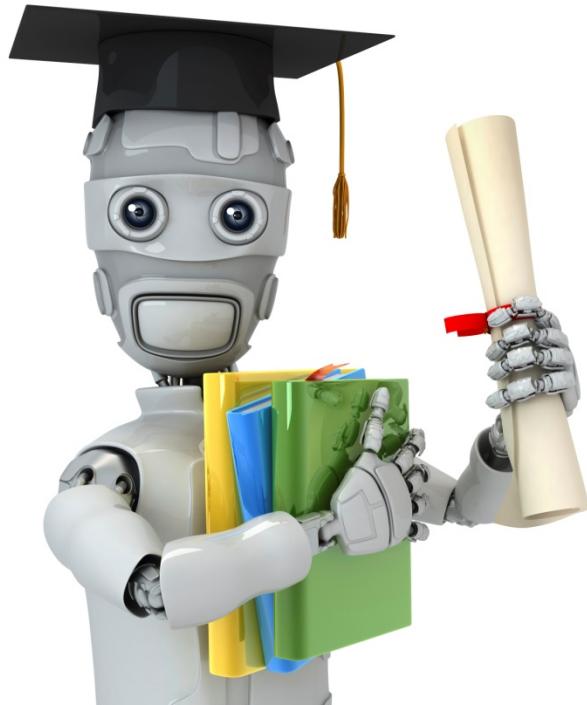
$p(x)$  is comparable (say, both large) for normal and anomalous examples



- Monitoring computers in a data center
- Choose features that might take on unusually large or small values in the event of an anomaly.
  - $x_1$  = memory use of computer
  - $x_2$  = number of disk accesses/sec
  - $x_3$  = CPU load ←
  - $x_4$  = network traffic ←

$$x_5 = \frac{\text{CPU load}}{\text{network traffic}}$$

$$x_6 = \frac{(\text{CPU load})^2}{\text{network traffic}}$$



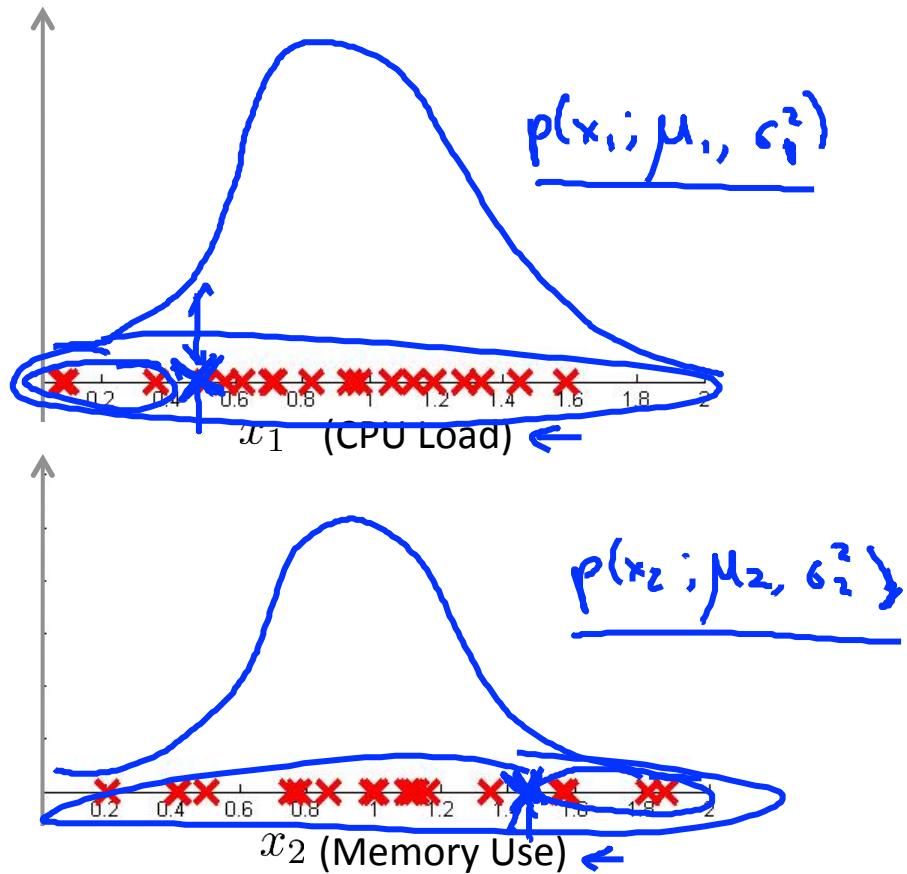
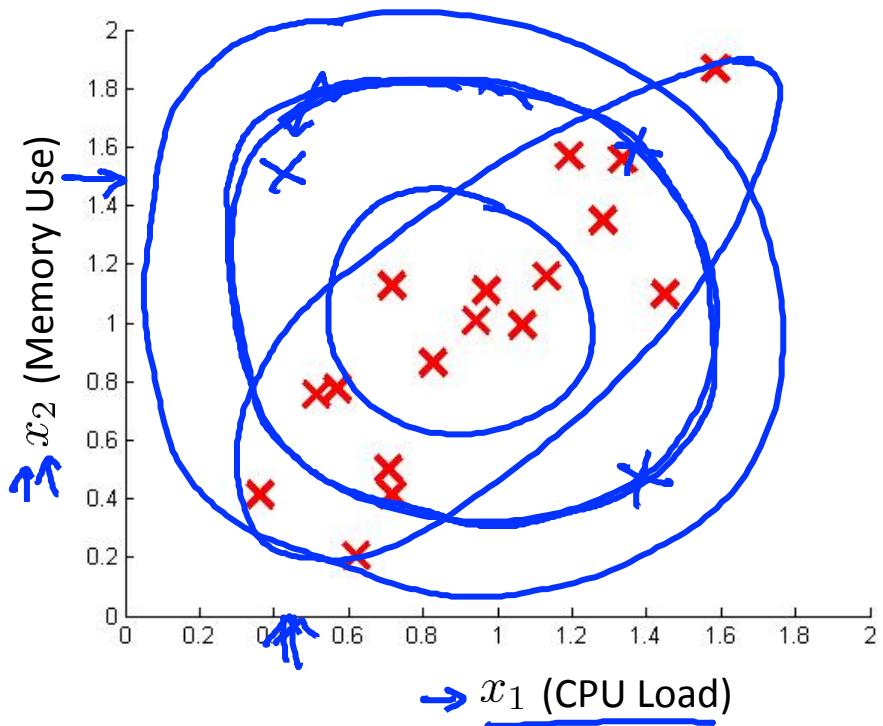
Machine Learning

# Anomaly detection

---

Multivariate  
Gaussian distribution

# Motivating example: Monitoring machines in a data center



# Multivariate Gaussian (Normal) distribution

→  $x \in \mathbb{R}^n$ . Don't model  $p(x_1), p(x_2), \dots$ , etc. separately.  
Model  $p(x)$  all in one go.  
Parameters:  $\mu \in \mathbb{R}^n$ ,  $\Sigma \in \mathbb{R}^{n \times n}$  (covariance matrix)

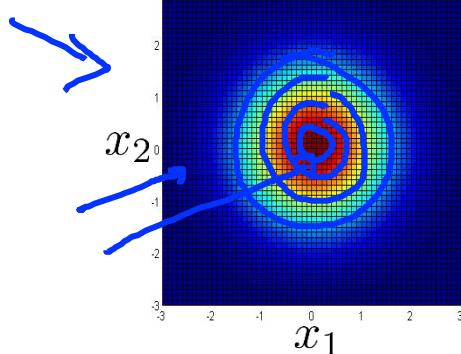
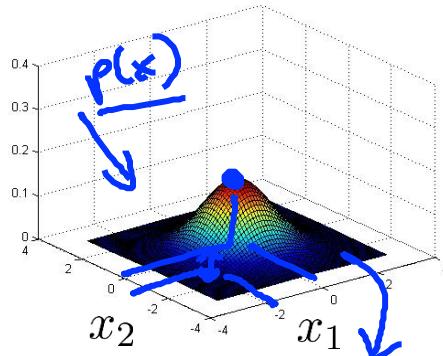
$$p(x; \mu, \Sigma) =$$

$$\frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2} (x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

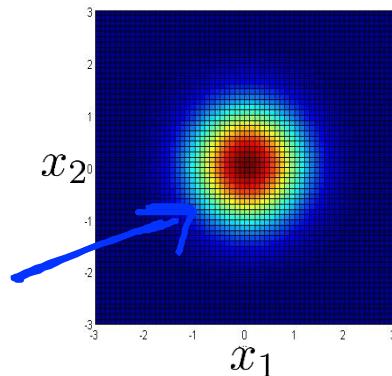
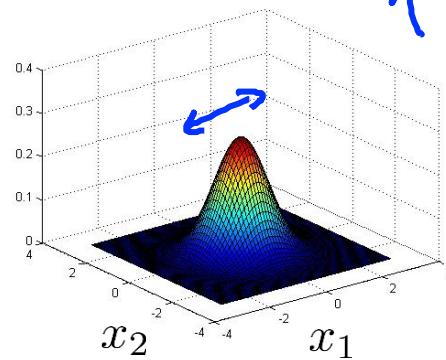
$|\Sigma| = \text{determinant of } \Sigma \quad | \det(\text{Sigma})$

# Multivariate Gaussian (Normal) examples

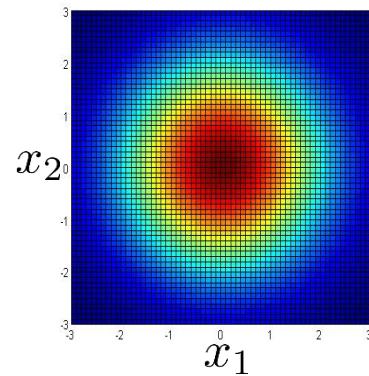
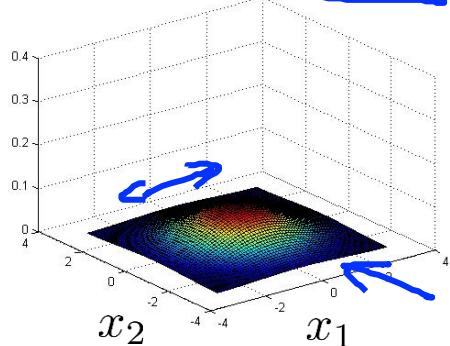
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix}$$

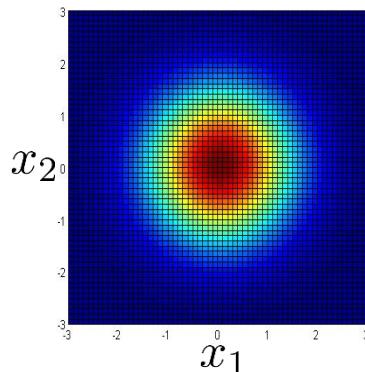
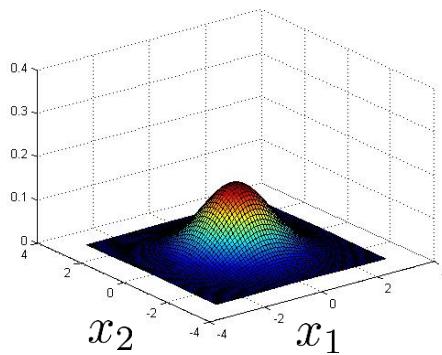


$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$$

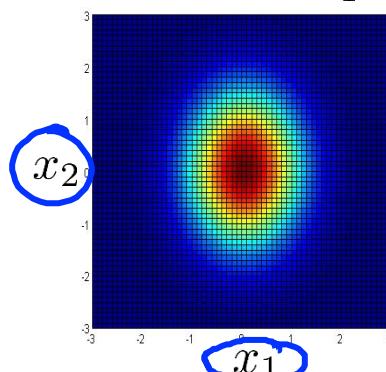
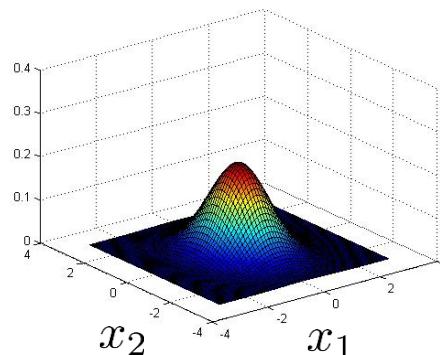


# Multivariate Gaussian (Normal) examples

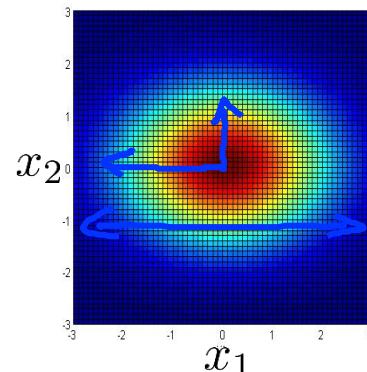
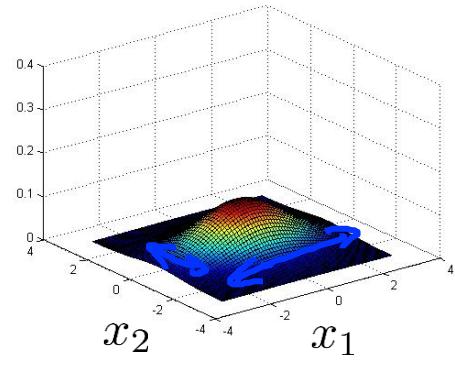
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 1 \end{bmatrix}$$

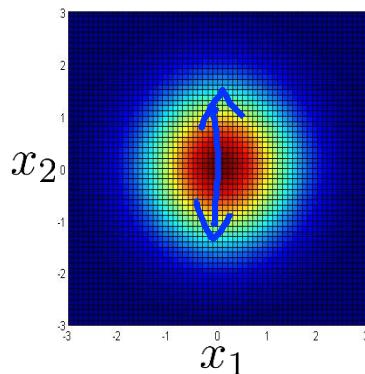
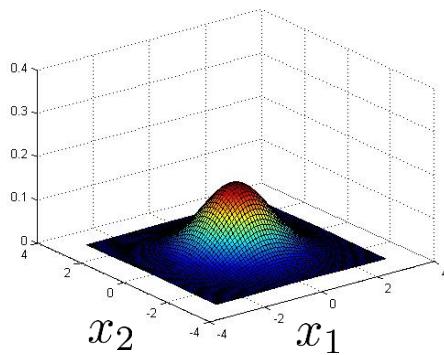


$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$

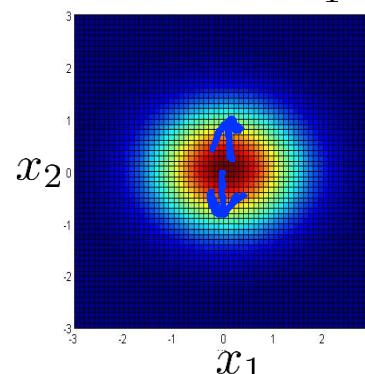
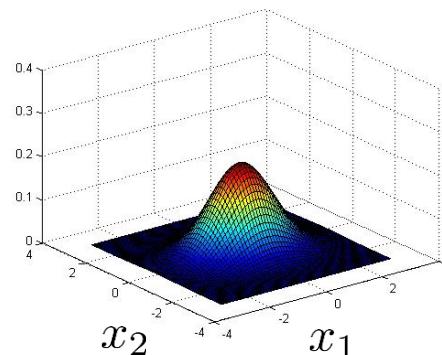


# Multivariate Gaussian (Normal) examples

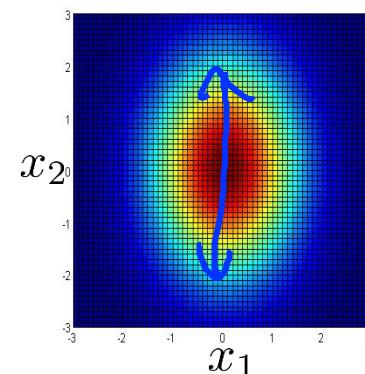
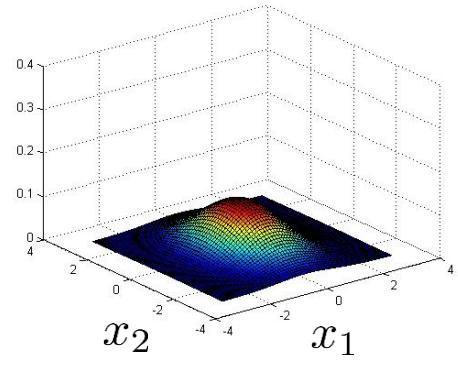
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 0.6 \end{bmatrix}$$

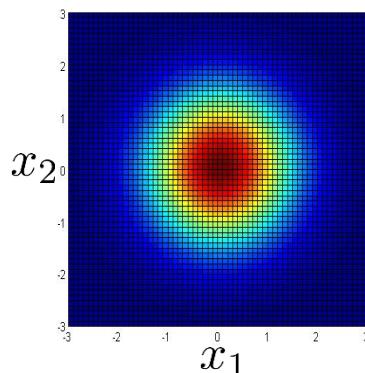
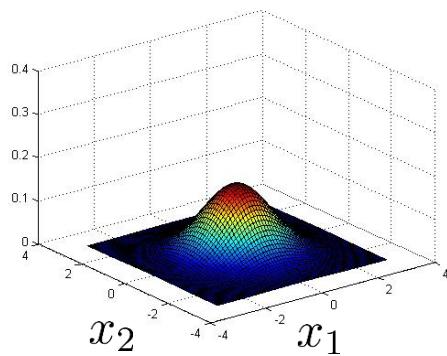


$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 2 \end{bmatrix}$$

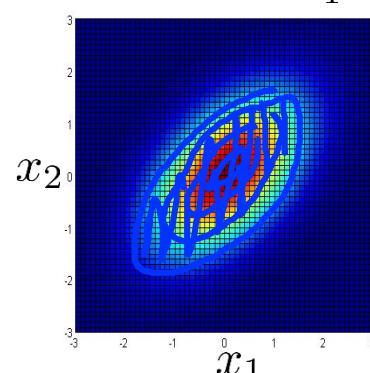
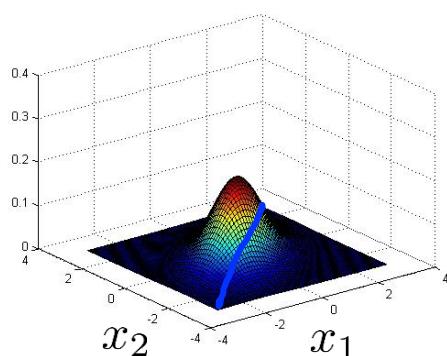


# Multivariate Gaussian (Normal) examples

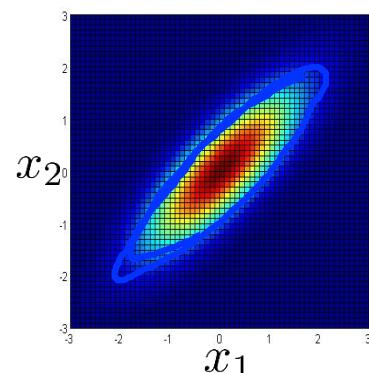
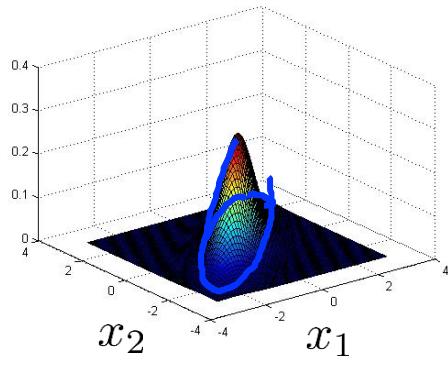
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

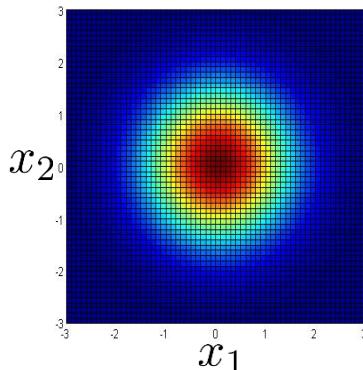
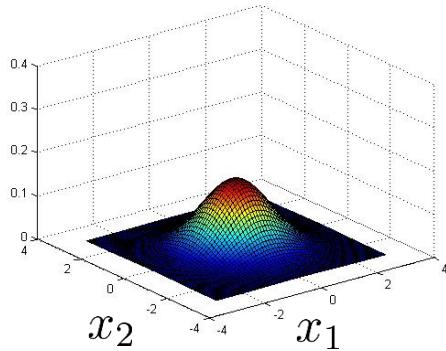


$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$

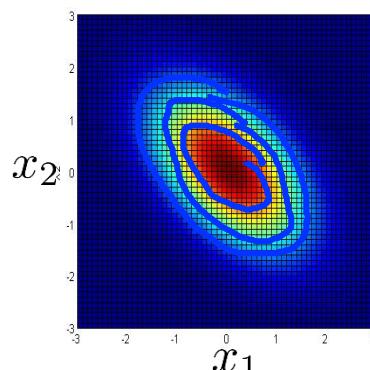
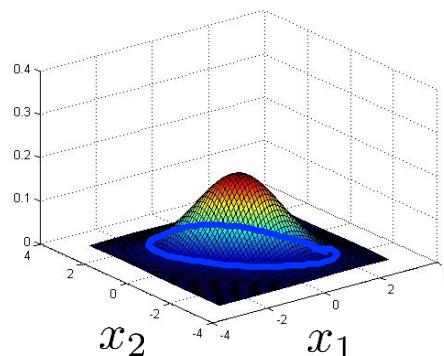


# Multivariate Gaussian (Normal) examples

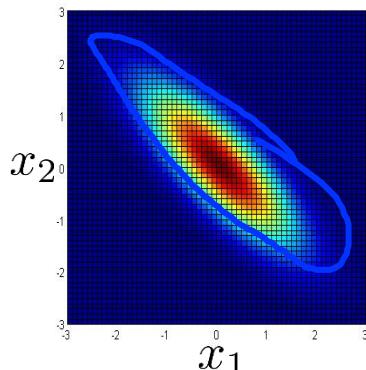
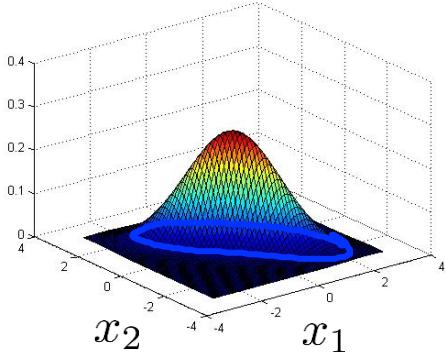
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$$

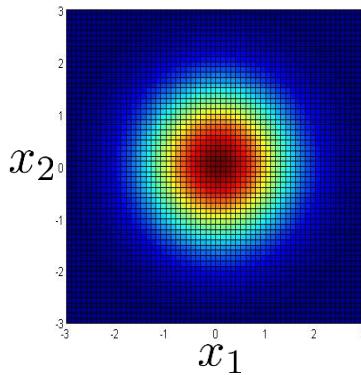
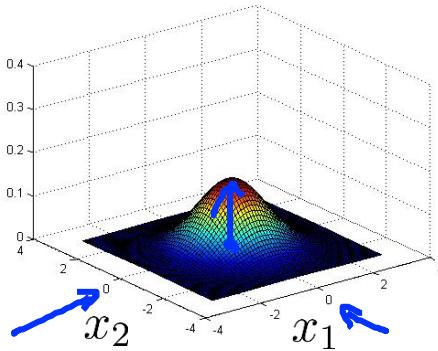


$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}$$

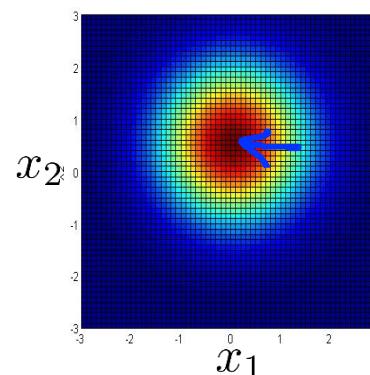
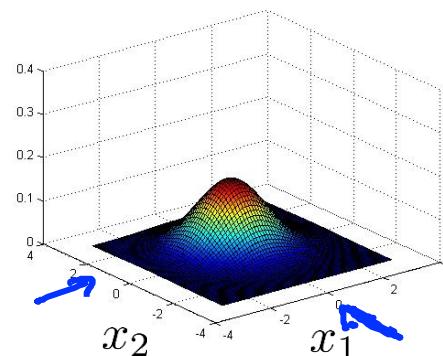


# Multivariate Gaussian (Normal) examples

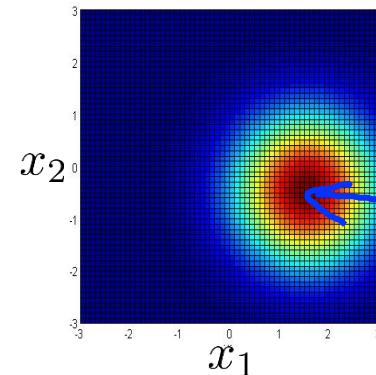
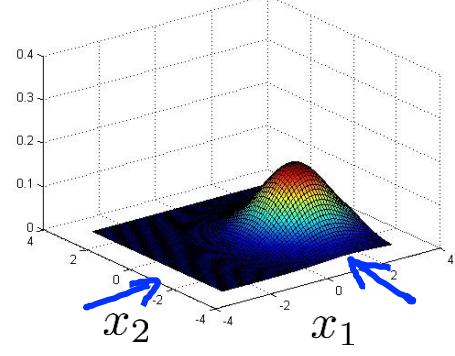
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

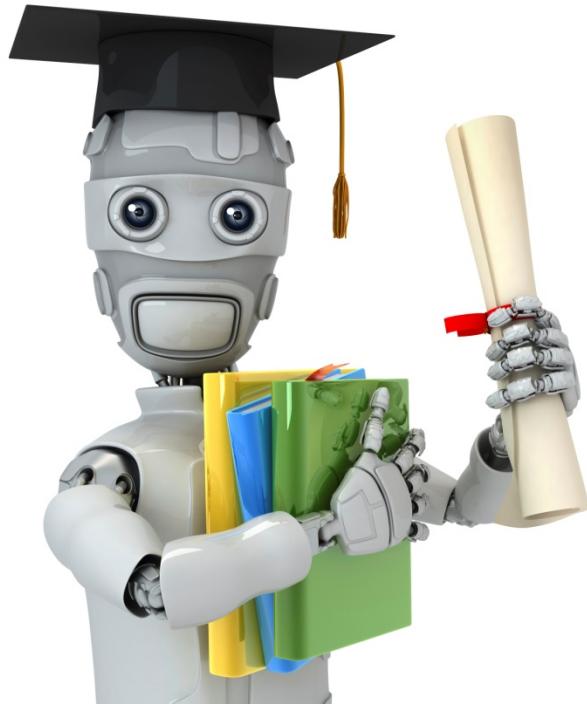


$$\mu = \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 1.5 \\ -0.5 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$





Machine Learning

# Anomaly detection

---

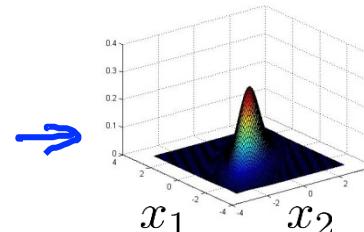
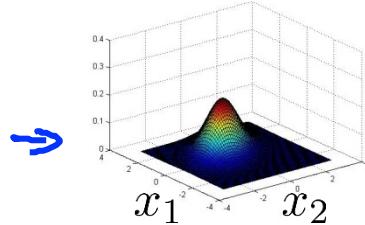
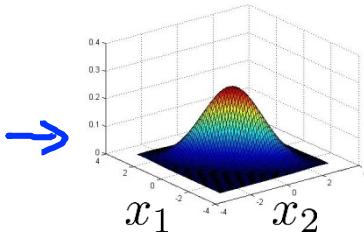
Anomaly detection using  
the multivariate  
Gaussian distribution

# Multivariate Gaussian (Normal) distribution

Parameters  $\mu, \Sigma$

$$\mu \in \mathbb{R}^n \quad \Sigma \in \mathbb{R}^{n \times n}$$

$$\rightarrow p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp \left( -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right)$$



Parameter fitting:

Given training set  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

$$x \in \mathbb{R}^n$$

$$\rightarrow \boxed{\mu} = \frac{1}{m} \sum_{i=1}^m x^{(i)} \quad \rightarrow \boxed{\Sigma} = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

# Anomaly detection with the multivariate Gaussian

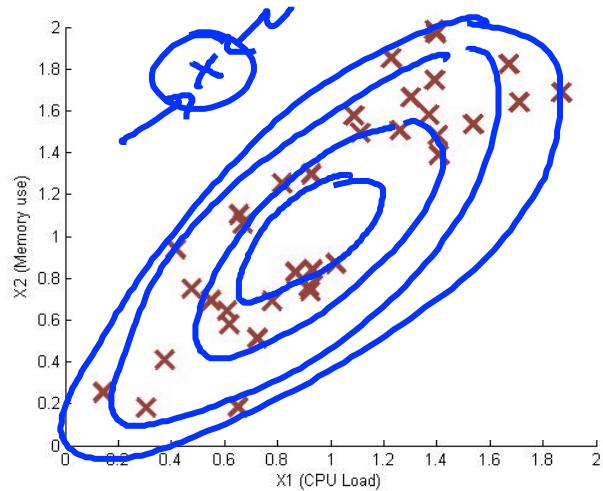
1. Fit model  $p(x)$  by setting

$$\left[ \begin{array}{l} \mu = \frac{1}{m} \sum_{i=1}^m x^{(i)} \\ \Sigma = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)(x^{(i)} - \mu)^T \end{array} \right]$$

2. Given a new example  $x$ , compute

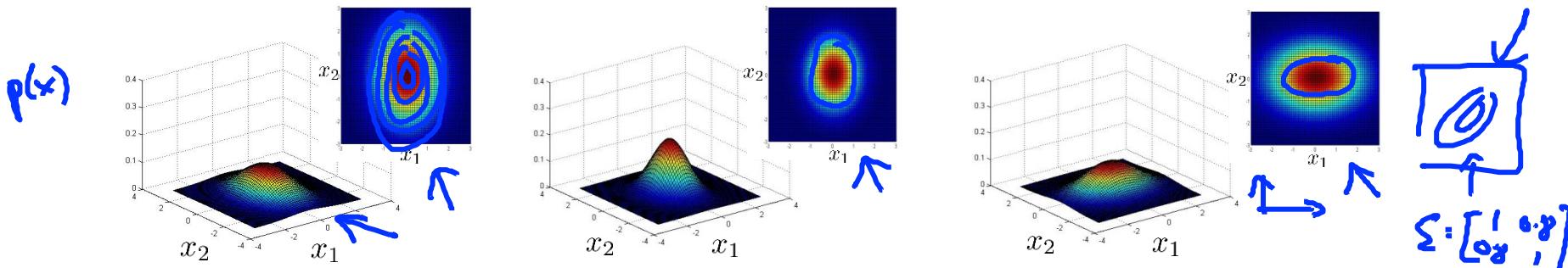
$$p(x) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp \left( -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right)$$

Flag an anomaly if  $\underline{p(x) < \varepsilon}$



## Relationship to original model

Original model:  $p(x) = p(x_1; \mu_1, \sigma_1^2) \times p(x_2; \mu_2, \sigma_2^2) \times \cdots \times p(x_n; \mu_n, \sigma_n^2)$



Corresponds to multivariate Gaussian

$$\rightarrow p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp \left( -\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right)$$

where

$$\Sigma = \begin{bmatrix} \dots & & \\ & \dots & \\ \vdots & & \end{bmatrix}$$

## → Original model

$$p(x_1; \mu_1, \sigma_1^2) \times \cdots \times p(x_n; \mu_n, \sigma_n^2)$$

Manually create features to capture anomalies where  $x_1, x_2$  take unusual combinations of values.

$$\rightarrow x_3 = \frac{x_1}{x_2} = \frac{\text{CPU load}}{\text{memory}}$$

- Computationally cheaper (alternatively, scales better to large  $n=10,000, n=100,000$ )
  - OK even if  $m$  (training set size) is small

## vs. → Multivariate Gaussian

$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}} |\Sigma|^{\frac{1}{2}}} \exp \left( -\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu) \right)$$

→ Automatically captures correlations between features

$$\Sigma \in \mathbb{R}^{n \times n}$$

$$\underline{\Sigma^{-1}}$$

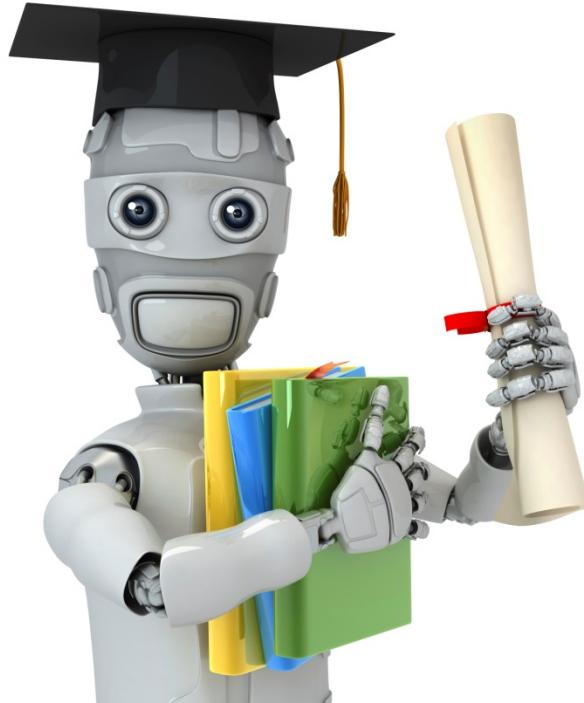
Computationally more expensive

$$\rightarrow \Sigma \sim \frac{n^2}{2}$$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 + x_5 \\ x_5 \end{bmatrix}$$

Must have  $m > n$  or else  $\Sigma$  is non-invertible.

$$\underline{m \geq n}$$



Machine Learning

# Recommender Systems

---

## Problem formulation

## Example: Predicting movie ratings

→ User rates movies using ~~one to five stars~~  
~~zero~~

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)
Love at last	5	5	0	6
Romance forever	5	?	0	0
Cute puppies of love	?	4	0	?
Nonstop car chases	5	0	5	4
Swords vs. karate	5	0	?	?

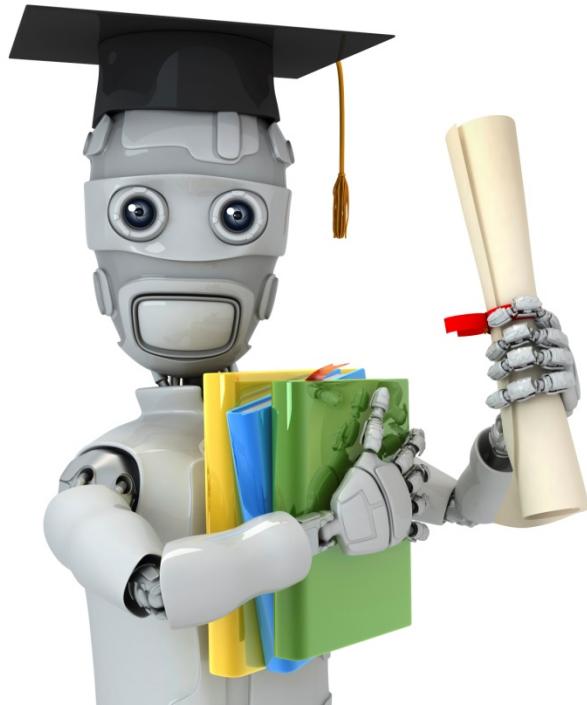
$$n_u = 4$$

$$n_m = 5$$



→  $n_u$  = no. users  
→  $n_m$  = no. movies  
 $r(i, j) = 1$  if user  $j$  has rated movie  $i$   
 $y^{(i,j)}$  = rating given by user  $j$  to movie  $i$   
(defined only if  $r(i, j) = 1$ )

$$0, \dots, 5$$



Machine Learning

# Recommender Systems

---

Content-based  
recommendations

# Content-based recommender systems

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	$n_u = 4, n_m = 5$	$x_0 = 1$	$x^{(1)} = \begin{bmatrix} 1 \\ 0.9 \\ 0 \end{bmatrix}$
Love at last	1	5	5	0	0		
Romance forever	2	5	?	?	0		
Cute puppies of love	3	?	4	0	?		
Nonstop car chases	4	0	0	5	4		
Swords vs. karate	5	0	0	5	?		

For each user  $j$ , learn a parameter  $\theta^{(j)} \in \mathbb{R}^3$ . Predict user  $j$  as rating movie  $(\theta^{(j)})^T x^{(i)}$  stars.

$$x^{(3)} = \begin{bmatrix} 1 \\ 0.99 \\ 0 \end{bmatrix} \leftrightarrow \theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}$$

$$(\theta^{(1)})^T x^{(3)} = 5 \times 0.99 = 4.95$$

## Problem formulation

- $r(i, j) = 1$  if user  $j$  has rated movie  $i$  (0 otherwise)
- $y^{(i,j)}$  = rating by user  $j$  on movie  $i$  (if defined)
- $\theta^{(j)}$  = parameter vector for user  $j$
- $x^{(i)}$  = feature vector for movie  $i$
- For user  $j$ , movie  $i$ , predicted rating:  $(\theta^{(j)})^T(x^{(i)})$
- $m^{(j)}$  = no. of movies rated by user  $j$

To learn  $\theta^{(j)}$ :

$$\min_{\theta^{(j)}} \frac{1}{2} \sum_{i : r(i,j)=1} \left( (\theta^{(j)})^T (x^{(i)}) - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{k=1}^n (\theta_k^{(j)})^2$$

$$\theta^{(j)} \in \mathbb{R}^{n+1}$$

## Optimization objective:

To learn  $\theta^{(j)}$  (parameter for user  $j$ ):

$$\rightarrow \min_{\theta^{(j)}} \frac{1}{2} \sum_{i:r(i,j)=1} \left( (\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{k=1}^n (\theta_k^{(j)})^2$$

To learn  $\theta^{(1)}$ ,  $\theta^{(2)}$ , ...,  $\theta^{(n_u)}$ :

$$\min_{\theta^{(1)}, \dots, \theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left( (\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^{(j)})^2$$

$\Theta^{(1)}, \dots, \Theta^{(n_u)}$

## Optimization algorithm:

$$\min_{\theta^{(1)}, \dots, \theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} \left( (\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^{(j)})^2$$

$\mathcal{J}(\theta^{(1)}, \dots, \theta^{(n_u)})$

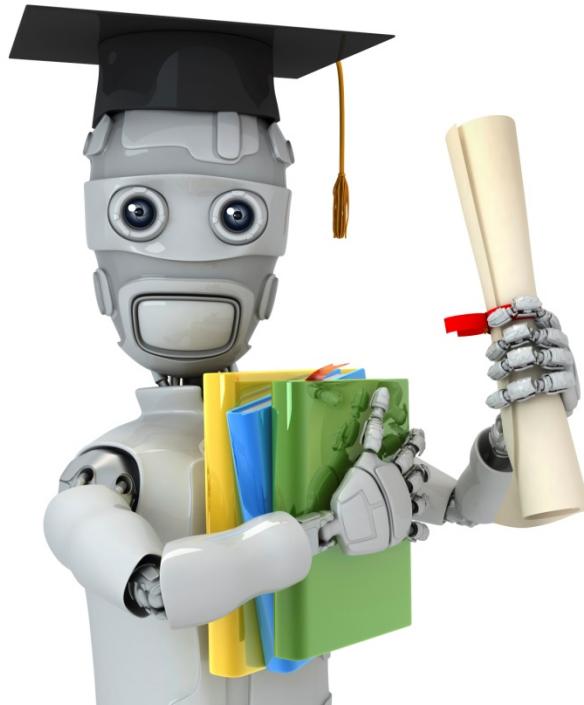
Gradient descent update:

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} \quad (\text{for } k = 0)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left( \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} + \lambda \theta_k^{(j)} \right) \quad (\text{for } k \neq 0)$$

~~$m^{(j)}$~~

$$\frac{\partial}{\partial \theta_k^{(j)}} \mathcal{J}(\theta^{(1)}, \dots, \theta^{(n_u)})$$



Machine Learning

# Recommender Systems

---

## Collaborative filtering

# Problem motivation

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	$x_1$ (romance)	$x_2$ (action)
Love at last	5	5	0	0	0.9	0
Romance forever	5	?	?	0	1.0	0.01
Cute puppies of love	?	4	0	?	0.99	0
Nonstop car chases	0	0	5	4	0.1	1.0
Swords vs. karate	0	0	5	?	0	0.9



# Problem motivation

$x^{(1)}$

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	$x_1$ (romance)	$x_2$ (action)	$x_o = 1$
<del>Love at last</del>	5	5	0	0	1.0	0.0	
Romance forever	5	?	?	0	?	?	
Cute puppies of love	?	4	0	?	?	?	
Nonstop car chases	0	0	5	4	?	?	
Swords vs. karate	0	0	5	?	?	?	

$x^{(2)}$

$\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}, \theta^{(2)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}, \theta^{(3)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}, \theta^{(4)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$

$\theta^{(j)}$

$(\theta^{(1)})^T x^{(1)} \approx 5$

$(\theta^{(2)})^T x^{(1)} \approx 5$

$(\theta^{(3)})^T x^{(1)} \approx 0$

$(\theta^{(4)})^T x^{(1)} \approx 0$

$x^N = \begin{bmatrix} 1 \\ 1.0 \\ ? \\ ? \\ ? \\ ? \end{bmatrix}$

$x^{(1)}$

Andrew Ng

# Optimization algorithm

Given  $\theta^{(1)}, \dots, \theta^{(n_u)}$ , to learn  $x^{(i)}$ :

$$\min_{x^{(i)}} \frac{1}{2} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^n (x_k^{(i)})^2$$

Given  $\theta^{(1)}, \dots, \theta^{(n_u)}$ , to learn  $x^{(1)}, \dots, x^{(n_m)}$ :

$$\min_{x^{(1)}, \dots, x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

# Collaborative filtering

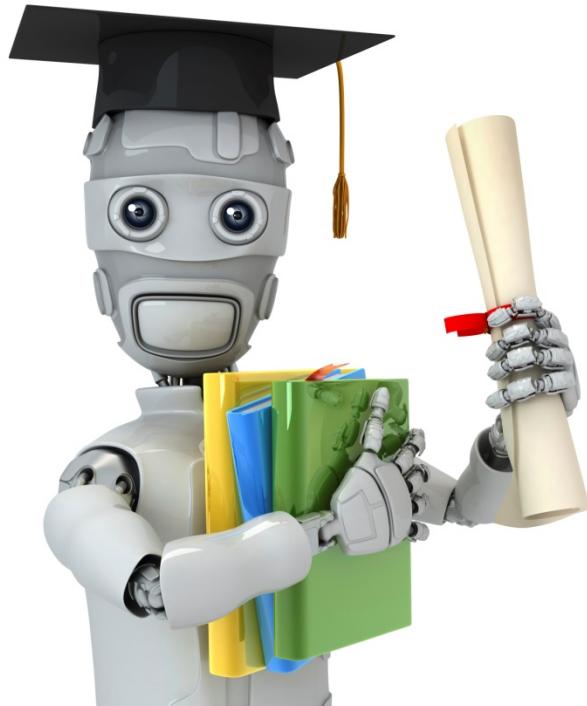
Given  $x^{(1)}, \dots, x^{(n_m)}$  (and movie ratings),  
can estimate  $\theta^{(1)}, \dots, \theta^{(n_u)}$

$$\begin{matrix} r^{(i,j)} \\ y^{(i,j)} \end{matrix}$$



Given  $\theta^{(1)}, \dots, \theta^{(n_u)}$ ,  
can estimate  $x^{(1)}, \dots, x^{(n_m)}$

Guess  $\Theta \rightarrow x \rightarrow \Theta \rightarrow x \rightarrow \Theta \rightarrow x \rightarrow \dots$



Machine Learning

# Recommender Systems

---

Collaborative  
filtering algorithm

## Collaborative filtering optimization objective

Given  $x^{(1)}, \dots, x^{(n_m)}$ , estimate  $\theta^{(1)}, \dots, \theta^{(n_u)}$ :

$$\min_{\theta^{(1)}, \dots, \theta^{(n_u)}} \frac{1}{2} \sum_{j=1}^{n_u} \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^{(j)})^2$$

$$(i,j) : r(i,j) = 1$$

$$x \in \mathbb{R}^n$$

$$\theta \in \mathbb{R}^n$$

$$x_1 = 1$$

Given  $\theta^{(1)}, \dots, \theta^{(n_u)}$ , estimate  $x^{(1)}, \dots, x^{(n_m)}$ :

$$\min_{x^{(1)}, \dots, x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2$$

Minimizing  $x^{(1)}, \dots, x^{(n_m)}$  and  $\theta^{(1)}, \dots, \theta^{(n_u)}$  simultaneously:

$$J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}) = \frac{1}{2} \sum_{(i,j):r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^{(j)})^2$$

$$\min_{x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}} J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$$



## Collaborative filtering algorithm

~~$x \in \mathbb{R}^n$ ,  $\theta \in \mathbb{R}^n$~~

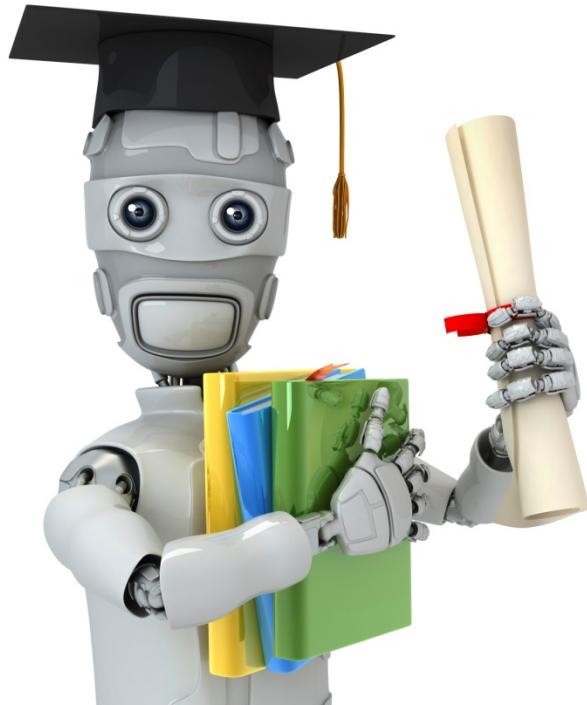
- 1. Initialize  $x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)}$  to small random values.
- 2. Minimize  $J(x^{(1)}, \dots, x^{(n_m)}, \theta^{(1)}, \dots, \theta^{(n_u)})$  using gradient descent (or an advanced optimization algorithm). E.g. for every  $j = 1, \dots, n_u, i = 1, \dots, n_m$  :

$$x_k^{(i)} := x_k^{(i)} - \alpha \left( \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) \theta_k^{(j)} + \lambda x_k^{(i)} \right)$$
$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha \left( \sum_{i:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)}) x_k^{(i)} + \lambda \theta_k^{(j)} \right)$$

$\frac{\partial}{\partial x_k^{(i)}} J(\dots)$

- 3. For a user with parameters  $\underline{\theta}$  and a movie with (learned) features  $\underline{x}$ , predict a star rating of  $\underline{\theta}^T \underline{x}$ .

$$(\underline{\theta}^{(i)})^T (\underline{x}^{(i)})$$



Machine Learning

# Recommender Systems

---

Vectorization:  
Low rank matrix  
factorization

# Collaborative filtering

$$n_m = 5$$

$$n_u = 4$$

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)
Love at last	5	5	0	0
Romance forever	5	?	?	0
Cute puppies of love	?	4	0	?
Nonstop car chases	0	0	5	4
Swords vs. karate	0	0	5	?

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix}$$



$y^{(i,j)}$

## Collaborative filtering

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 \\ 5 & ? & ? & 0 \\ ? & 4 & 0 & ? \\ 0 & 0 & 5 & 4 \\ 0 & 0 & 5 & 0 \end{bmatrix}$$

$$\mathbf{x} \Theta^T \leftarrow (\Theta^{(1)})^T (x^{(1)})$$

Predicted ratings:

$$\begin{bmatrix} (\theta^{(1)})^T (x^{(1)}) \\ (\theta^{(1)})^T (x^{(2)}) \\ \vdots \\ (\theta^{(1)})^T (x^{(n_m)}) \end{bmatrix} \quad \begin{bmatrix} (\theta^{(2)})^T (x^{(1)}) \\ (\theta^{(2)})^T (x^{(2)}) \\ \vdots \\ (\theta^{(2)})^T (x^{(n_m)}) \end{bmatrix} \quad \dots \quad \begin{bmatrix} (\theta^{(n_u)})^T (x^{(1)}) \\ (\theta^{(n_u)})^T (x^{(2)}) \\ \vdots \\ (\theta^{(n_u)})^T (x^{(n_m)}) \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} -(x^{(1)})^T \\ -(x^{(2)})^T \\ \vdots \\ -(x^{(n_m)})^T \end{bmatrix} \quad \Theta = \begin{bmatrix} -(\Theta^{(1)})^T \\ -(\Theta^{(2)})^T \\ \vdots \\ -(\Theta^{(n_u)})^T \end{bmatrix}$$

→ Low rank matrix factorization

## Finding related movies

For each product  $i$ , we learn a feature vector  $\underline{x}^{(i)} \in \mathbb{R}^n$ .

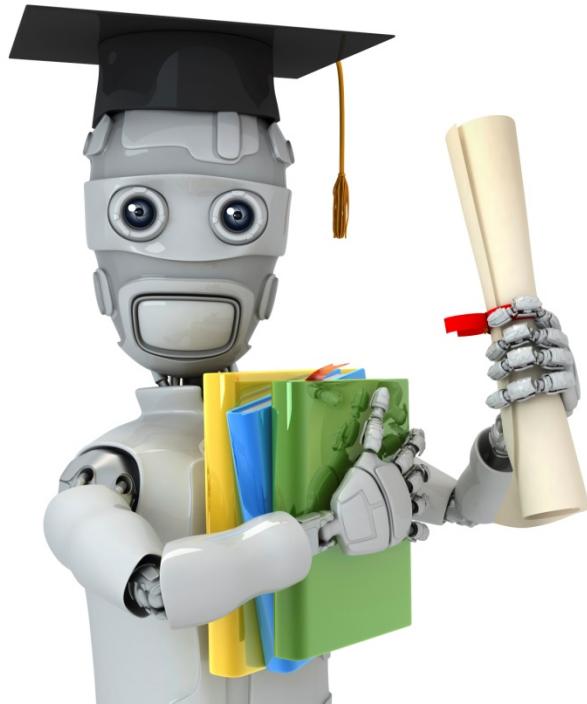
$\rightarrow x_1 = \text{romance}$ ,  $x_2 = \text{action}$ ,  $x_3 = \text{comedy}$ ,  $x_4 = \dots$

How to find movies  $j$  related to movie  $i$ ?

small  $\|x^{(i)} - x^{(j)}\|$   $\rightarrow$  movie  $j$  and  $i$  are "similar"

5 most similar movies to movie  $i$ :

Find the 5 movies  $j$  with the smallest  $\|x^{(i)} - x^{(j)}\|$ .



Machine Learning

# Recommender Systems

---

Implementational  
detail: Mean  
normalization

# Users who have not rated any movies

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	Eve (5)
Love at last	5	5	0	0	?
Romance forever	5	?	?	0	?
Cute puppies of love	?	4	0	?	?
Nonstop car chases	0	0	5	4	?
Swords vs. karate	0	0	5	?	?

↓

$$Y = \begin{bmatrix} 5 & 5 & 0 & 0 & ? \\ 5 & ? & ? & 0 & ? \\ ? & 4 & 0 & ? & ? \\ 0 & 0 & 5 & 4 & ? \\ 0 & 0 & 5 & 0 & ? \end{bmatrix}$$

$$\min_{\substack{x^{(1)}, \dots, x^{(n_m)} \\ \theta^{(1)}, \dots, \theta^{(n_u)}}} \frac{1}{2} \sum_{(i,j): r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^n (x_k^{(i)})^2 + \frac{\lambda}{2} \sum_{j=1}^{n_u} \sum_{k=1}^n (\theta_k^{(j)})^2$$

$n=2$

$$\underline{\Theta}^{(s)} \in \mathbb{R}^2$$

$$\underline{\Theta}^{(s)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$\frac{\lambda}{2} [(\underline{\Theta}_1^{(s)})^2 + (\underline{\Theta}_2^{(s)})^2] \leftarrow$$

$$(\underline{\Theta}^{(s)})^T \underline{x}^{(i)} = 0$$

## Mean Normalization:

$$Y = \begin{bmatrix} \rightarrow & 5 & 5 & 0 & 0 & ? & -2.5 \\ \rightarrow & 5 & ? & ? & 0 & ? & -2.5 \\ Y = & ? & 4 & 0 & ? & ? & -2 \\ \rightarrow & 0 & 0 & 5 & 4 & ? & \vdots \\ \rightarrow & 0 & 0 & 5 & 0 & ? & \vdots \end{bmatrix}$$

$$\mu = \begin{bmatrix} \rightarrow & 2.5 \\ \rightarrow & 2.5 \\ \rightarrow & 2 \\ \rightarrow & 2.25 \\ \rightarrow & 1.25 \end{bmatrix} \rightarrow \underline{Y} =$$

$$\begin{bmatrix} \circled{2.5} & \circled{2.5} & \circled{-2.5} & \circled{-2.5} & ? \\ 2.5 & ? & ? & -2.5 & ? \\ ? & 2 & -2 & ? & ? \\ -2.25 & -2.25 & 2.75 & 1.75 & ? \\ -1.25 & -1.25 & 3.75 & -1.25 & ? \end{bmatrix}$$

For user  $j$ , on movie  $i$  predict:

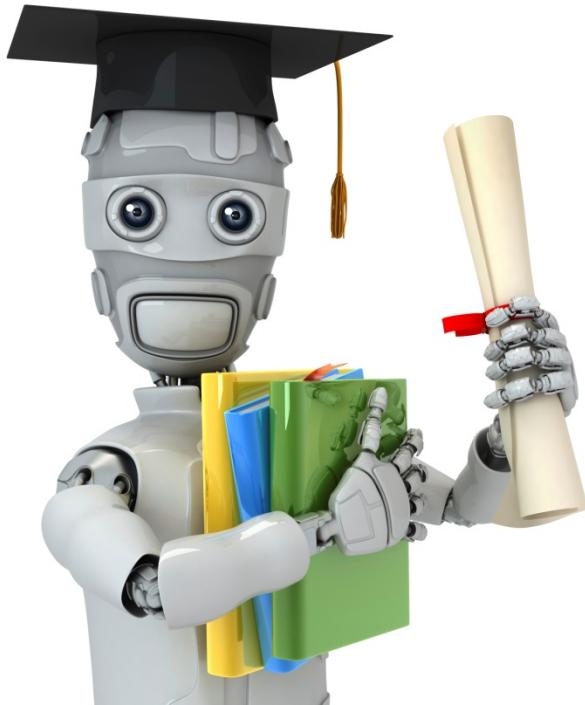
$$\rightarrow (\underline{\theta}^{(s)})^T (\underline{x}^{(i)}) + \underline{\mu_i}$$

$\downarrow$   
learn  $\underline{\theta}^{(s)}, \underline{x}^{(i)}$

User 5 (Eve):

$$\underline{\theta}^{(s)} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

$$\underbrace{(\underline{\theta}^{(s)})^T (\underline{x}^{(i)})}_{\sim 0} + \boxed{\underline{\mu_i}}$$



Machine Learning

Large scale  
machine learning

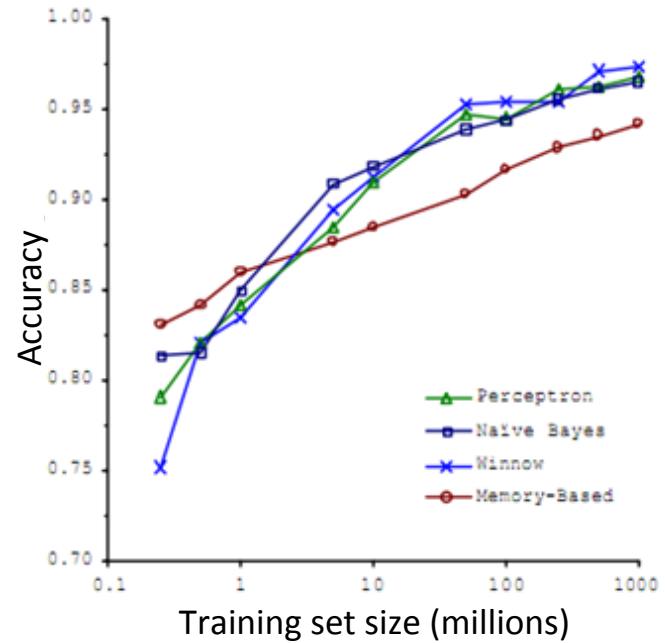
---

Learning with  
large datasets

# Machine learning and data

Classify between confusable words.  
E.g., {to, two, too}, {then, than}.

For breakfast I ate two eggs.



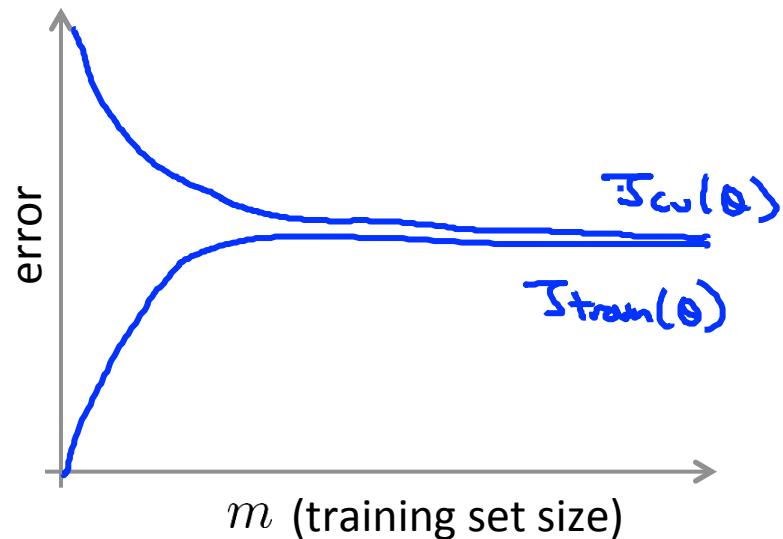
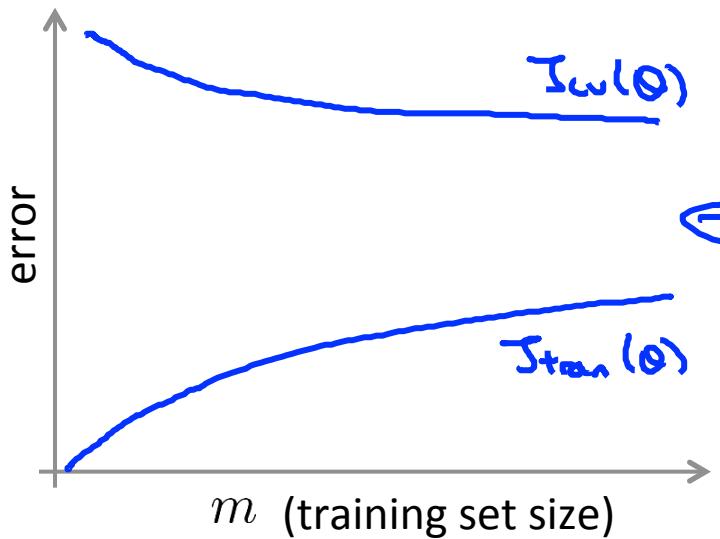
“It’s not who has the best algorithm that wins.  
It’s who has the most data.”

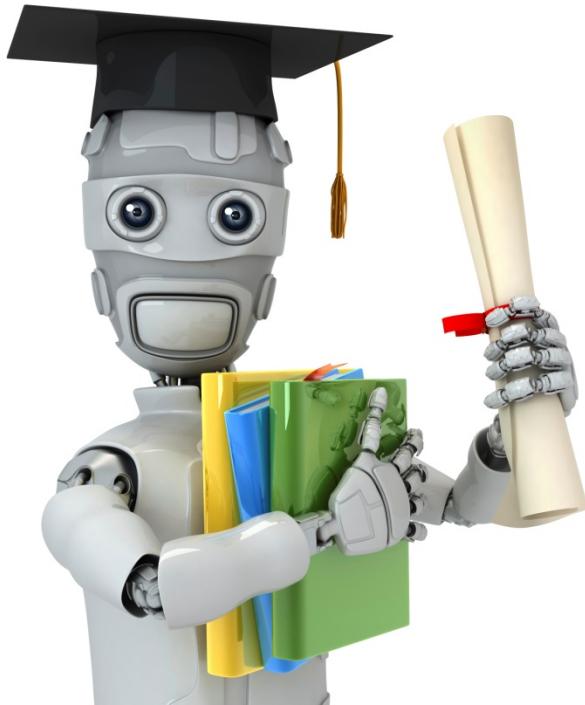
# Learning with large datasets

$m = 100,000,000$

$m = 1,000?$

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$





Machine Learning

Large scale  
machine learning

---

Stochastic  
gradient descent

# Linear regression with gradient descent

$$\rightarrow h_{\theta}(x) = \sum_{j=0}^n \theta_j x_j$$

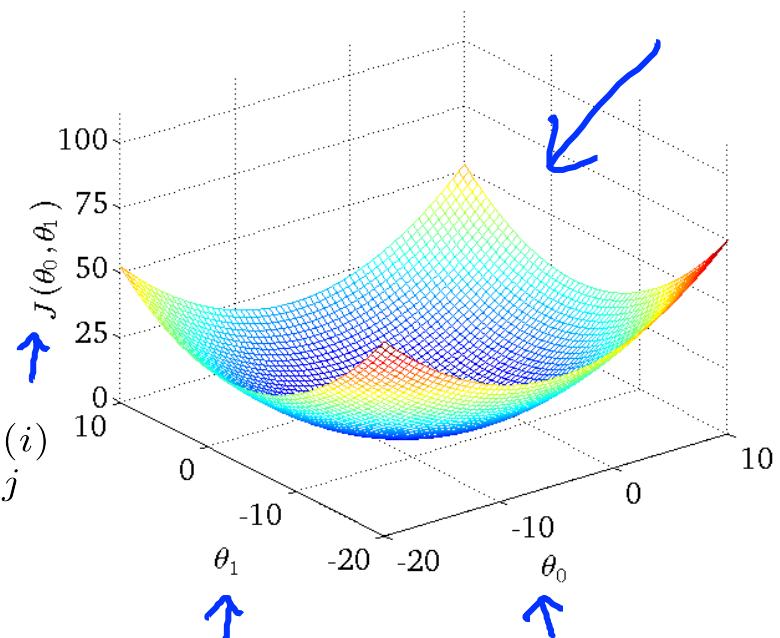
$$\rightarrow J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

Repeat {

$$\rightarrow \theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(for every  $j = 0, \dots, n$ )

}



# Linear regression with gradient descent

$$h_{\theta}(x) = \sum_{j=0}^n \theta_j x_j$$

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

Repeat {

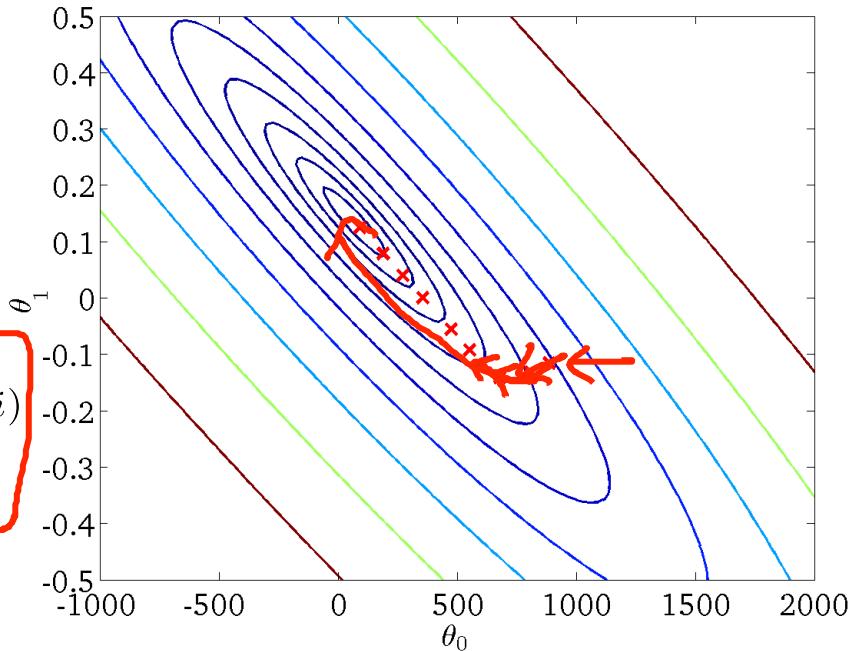
$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(for every  $j = 0, \dots, n$ )

}

$$M = \underline{300,000,000}$$

Batch gradient descent



## Batch gradient descent

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

Repeat {

$$\rightarrow \theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

$$\frac{\partial}{\partial \theta_j} J_{train}(\theta)$$

(for every  $j = 0, \dots, n$ )

}

$m = 300,000,000$

## Stochastic gradient descent

$$\rightarrow \underline{cost}(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2} (h_\theta(x^{(i)}) - y^{(i)})^2$$

$$\rightarrow J_{train}(\theta) = \frac{1}{m} \sum_{i=1}^m cost(\theta, (x^{(i)}, y^{(i)}))$$

1. Randomly shuffle dataset. ←

2. Repeat {

for  $i=1, \dots, m$  {

$$\theta_j := \theta_j - \alpha \boxed{(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}}$$

} (for  $j=0, \dots, n$ )

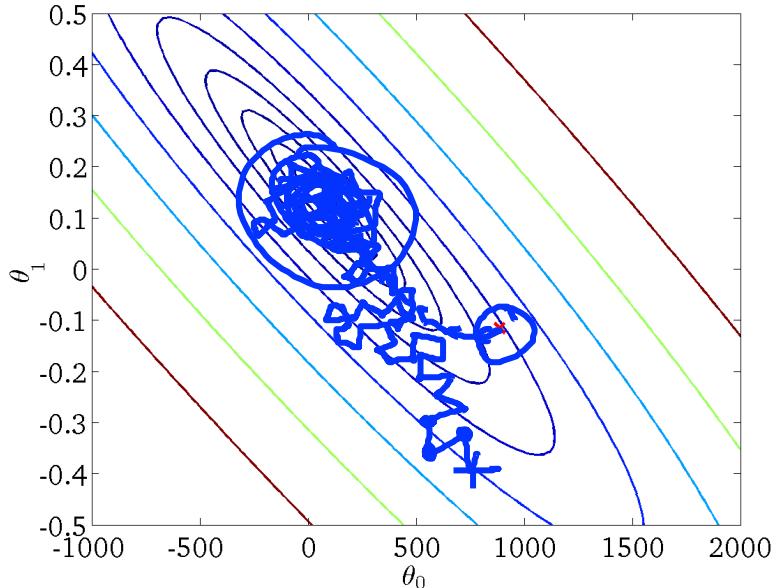
$$\rightarrow \frac{\partial}{\partial \theta_j} \underline{cost}(\theta, (x^{(i)}, y^{(i)}))$$

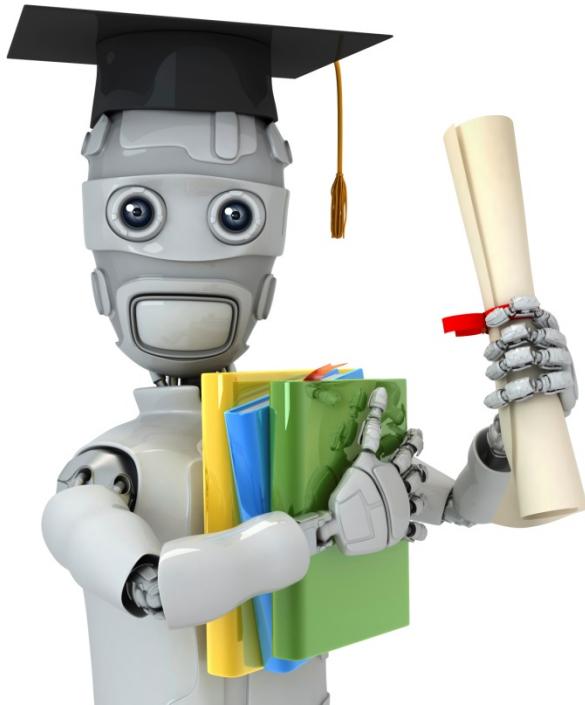
$\rightarrow \underline{(x^{(1)}, y^{(1)})}, \underline{(x^{(2)}, y^{(2)})}, \underline{(x^{(3)}, y^{(3)})}, \dots$

# Stochastic gradient descent

→ 1. Randomly shuffle (reorder) training examples

→ 2. Repeat { 1 - 10x  
    for  $i := 1, \dots, m$  {  
         $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$   
        (for  $j = 0, \dots, n$ )  
    every } }  
    )  
    →  $m = 300, 000, 000$





Machine Learning

Large scale  
machine learning

---

Mini-batch  
gradient descent

## Mini-batch gradient descent

- Batch gradient descent: Use all<sup>m</sup> examples in each iteration
- Stochastic gradient descent: Use 1 example in each iteration

Mini-batch gradient descent: Use b examples in each iteration

$b = \text{mini-batch size}$ .       $b = 10$ .       $\frac{2-100}{2-100}$

Get  $b = 10$  examples  $(x^{(i)}, y^{(i)}) \dots (x^{(i+9)}, y^{(i+9)})$

$$\nabla_{\theta_j} := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_{\theta}(x^{(k)}) - y^{(k)}) \cdot x_j^{(k)}$$

$$i := i + 10$$

# Mini-batch gradient descent

Say  $b = 10$ ,  $m = 1000$ .

Repeat {

→ for  $i = 1, 11, 21, 31, \dots, 991$  {

→  $\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$

(for every  $j = 0, \dots, n$ )

}

}

$$\underline{m = 300, 600, 000}$$

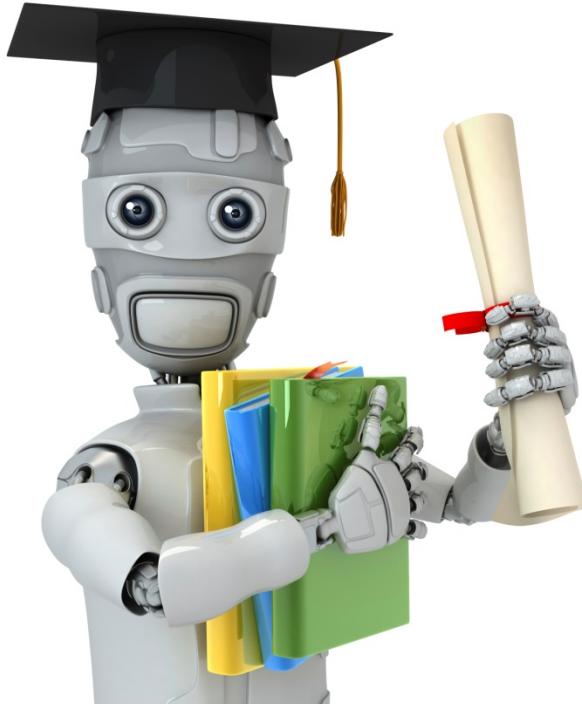


→  $b$  examples

→ 1 example

Vectorization

$$b = \underline{10}$$



Machine Learning

# Large scale machine learning

---

## Stochastic gradient descent convergence

## Checking for convergence

→ Batch gradient descent:

→ Plot  $J_{train}(\theta)$  as a function of the number of iterations of gradient descent.

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

$m = 300, 500, 1000$

→ Stochastic gradient descent:

$$\rightarrow cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2} (h_\theta(x^{(i)}) - y^{(i)})^2$$

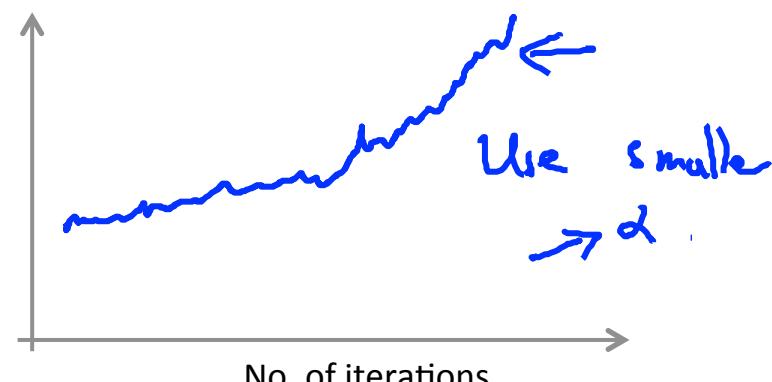
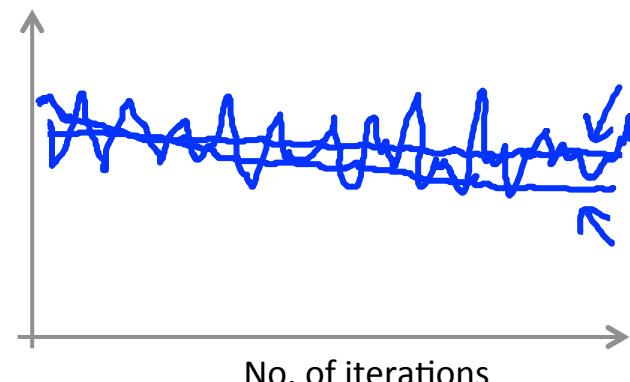
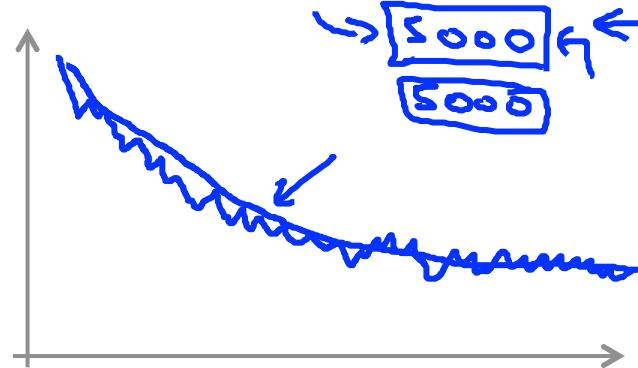
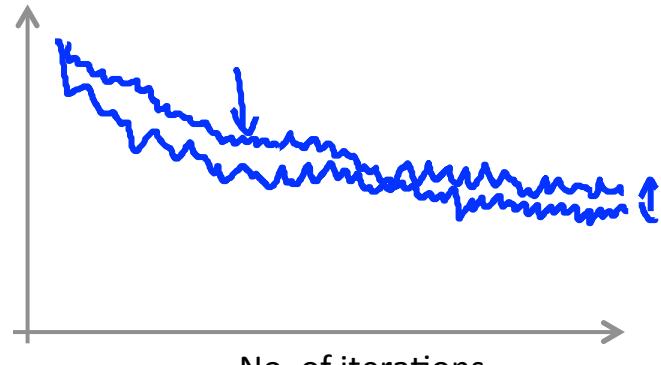
→ During learning, compute  $\underset{\uparrow}{cost}(\theta, \underset{\uparrow}{(x^{(i)}, y^{(i)})})$  before updating  $\theta$  using  $(x^{(i)}, y^{(i)})$ .

$$\Rightarrow \underline{(x^{(i)}, y^{(i)})}, \underline{(x^{(i+1)}, y^{(i+1)})}, \dots$$

→ Every 1000 iterations (say), plot  $cost(\theta, (x^{(i)}, y^{(i)})$  averaged over the last 1000 examples processed by algorithm.

## Checking for convergence

Plot  $\text{cost}(\theta, (x^{(i)}, y^{(i)}))$ , averaged over the last 1000 (say) examples



Use smaller  
 $\alpha$ .

# Stochastic gradient descent

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

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

1. Randomly shuffle dataset.

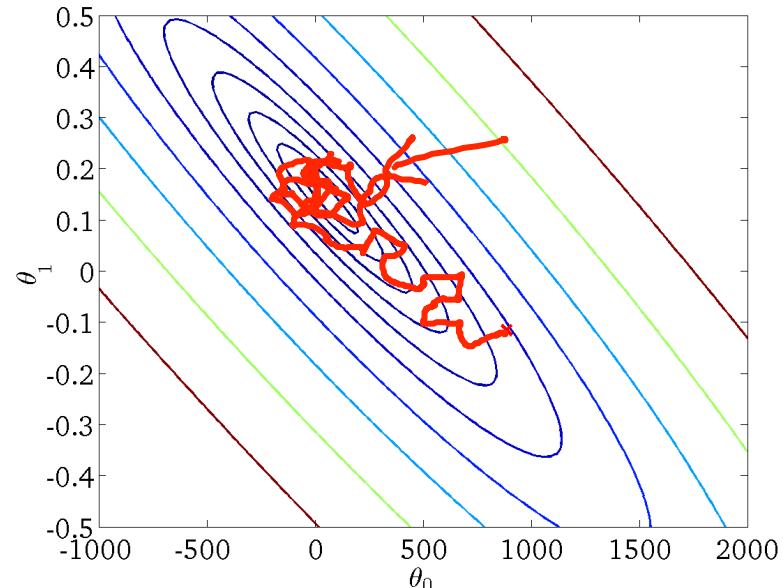
2. Repeat {

```
    for i := 1, ..., m      {
        theta_j := theta_j - alpha(h_theta(x^{(i)}) - y^{(i)}) * x_j^{(i)}
        (for j = 0, ..., n)
```

}

}

Learning rate  $\alpha$  is typically held constant. Can slowly decrease  $\alpha$  over time if we want  $\theta$  to converge. (E.g.  $\alpha = \frac{\text{const1}}{\text{iterationNumber} + \text{const2}}$ )

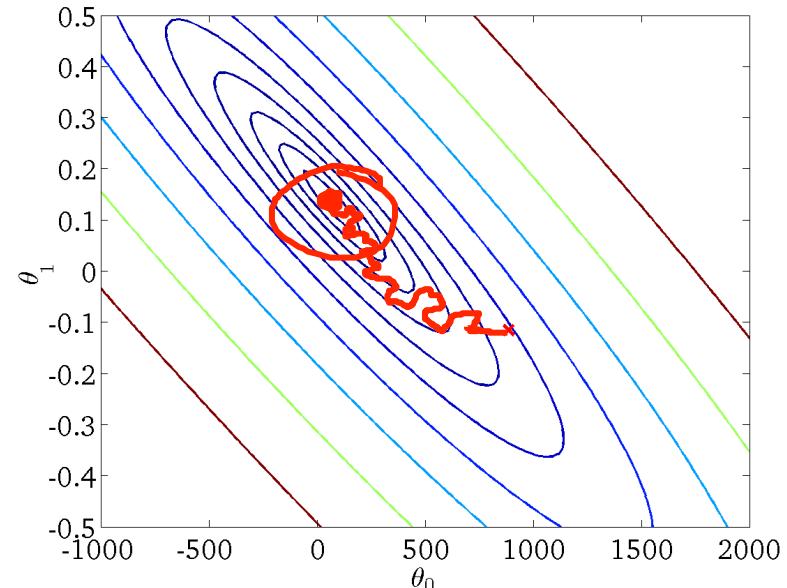


# Stochastic gradient descent

$$cost(\theta, (x^{(i)}, y^{(i)})) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

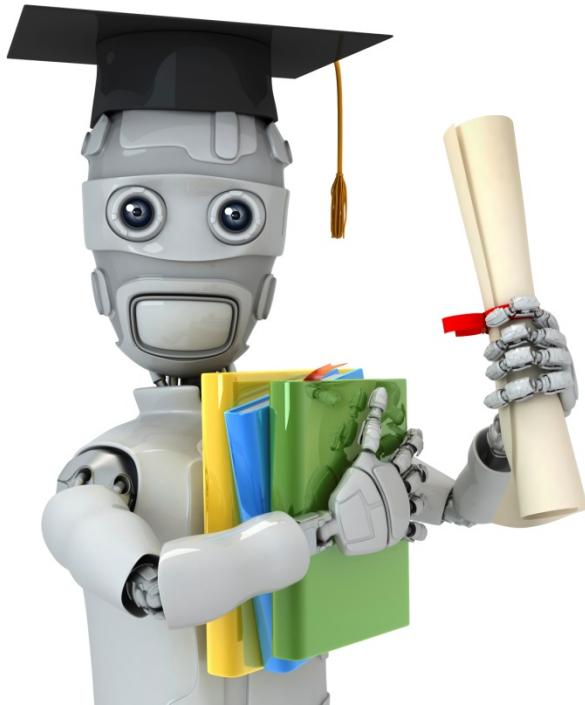
$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^m cost(\theta, (x^{(i)}, y^{(i)}))$$

1. Randomly shuffle dataset.
2. Repeat {  
     $\text{for } i := 1, \dots, m \quad \{$   
         $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$   
         $(\text{for } j = 0, \dots, n)$   
    }  
}



Learning rate  $\alpha$  is typically held constant. Can slowly decrease  $\alpha$  over time if we want  $\theta$  to converge. (E.g.  $\alpha = \frac{\text{const1}}{\text{iterationNumber} + \text{const2}}$ )  $\xrightarrow{\text{d}\rightarrow 0}$

$$\alpha = \frac{\text{const1}}{\text{iterationNumber} + \text{const2}}$$



Machine Learning

Large scale  
machine learning

---

Online learning

## Online learning

Shipping service website where user comes, specifies origin and destination, you offer to ship their package for some asking price, and users sometimes choose to use your shipping service ( $y = 1$ ), sometimes not ( $y = 0$ ).

Features  $x$  capture properties of user, of origin/destination and asking price. We want to learn  $p(y = 1|x; \theta)$  to optimize price.

Repeat forever {

Get  $(x, y)$  corresponding to user.

Update  $\theta$  using  $(x, y)$ :  $\cancel{(x, y)}$

$\rightarrow \theta_j := \theta_j - \alpha (h_\theta(x) - y) \cdot x_j \quad (j=0, \dots, n)$

$\rightarrow$  Can adapt to changing user preference.

price      logistic regression

## Other online learning example:

### Product search (learning to search)

User searches for “Android phone 1080p camera” 

Have 100 phones in store. Will return 10 results.

→  $x = \text{features of phone, how many words in user query match name of phone, how many words in query match description of phone, etc.}$

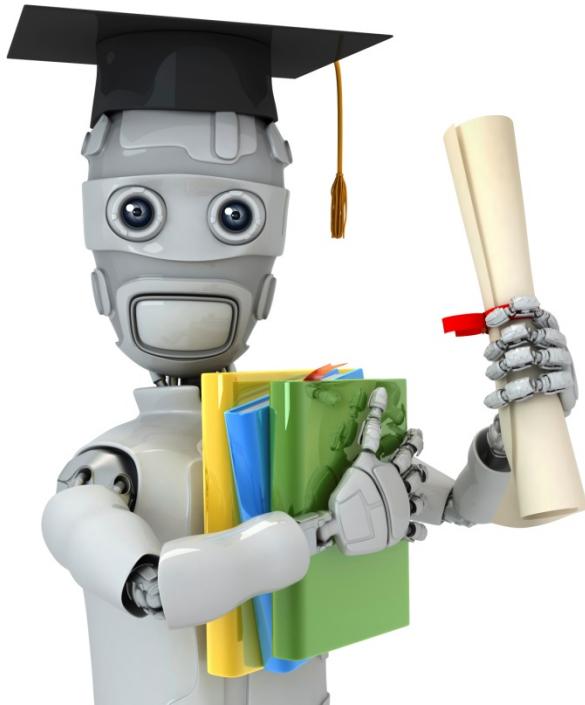
→  $y = 1$  if user clicks on link.  $y = 0$

$(x, y)$    
otherwise 

→ Learn  $p(y = 1|x; \theta)$ .  predicted CTR

→ Use to show user the 10 phones they’re most likely to click on.

Other examples: Choosing special offers to show user; customized selection of news articles; product recommendation; ...



Machine Learning

# Large scale machine learning

---

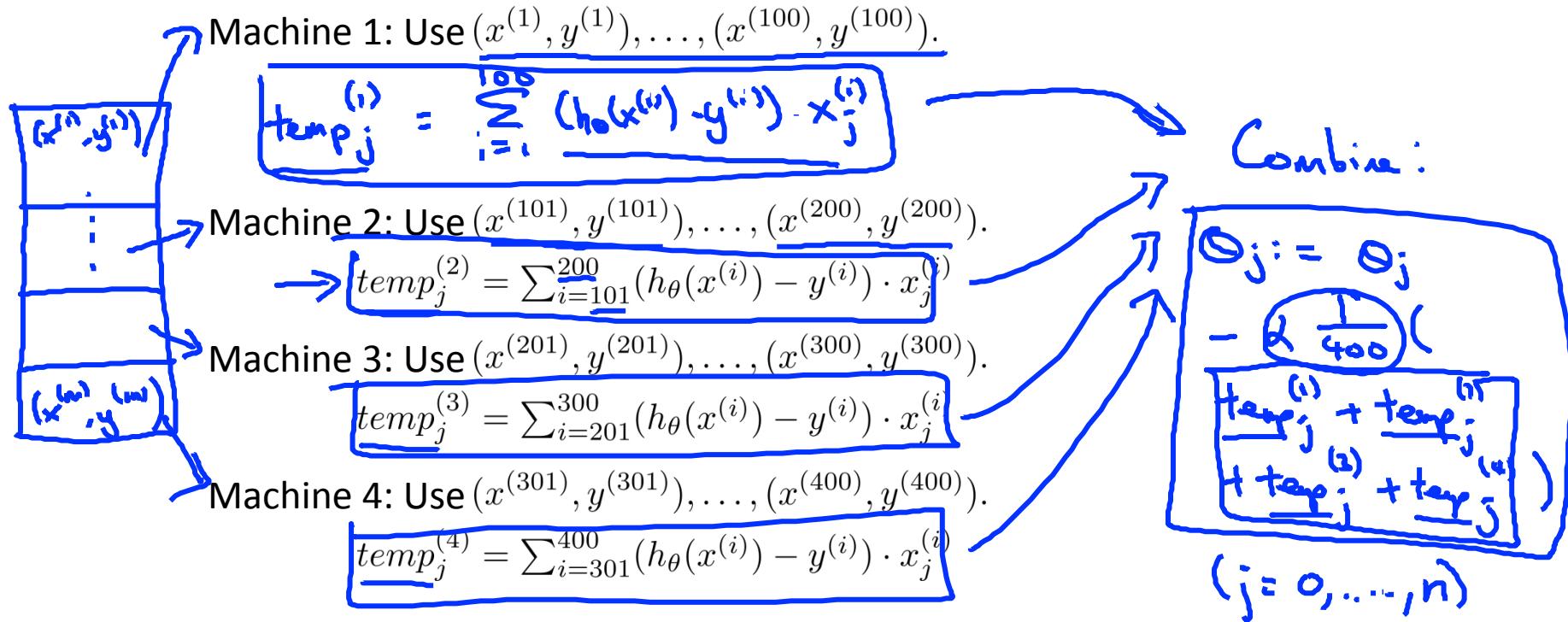
## Map-reduce and data parallelism

## Map-reduce

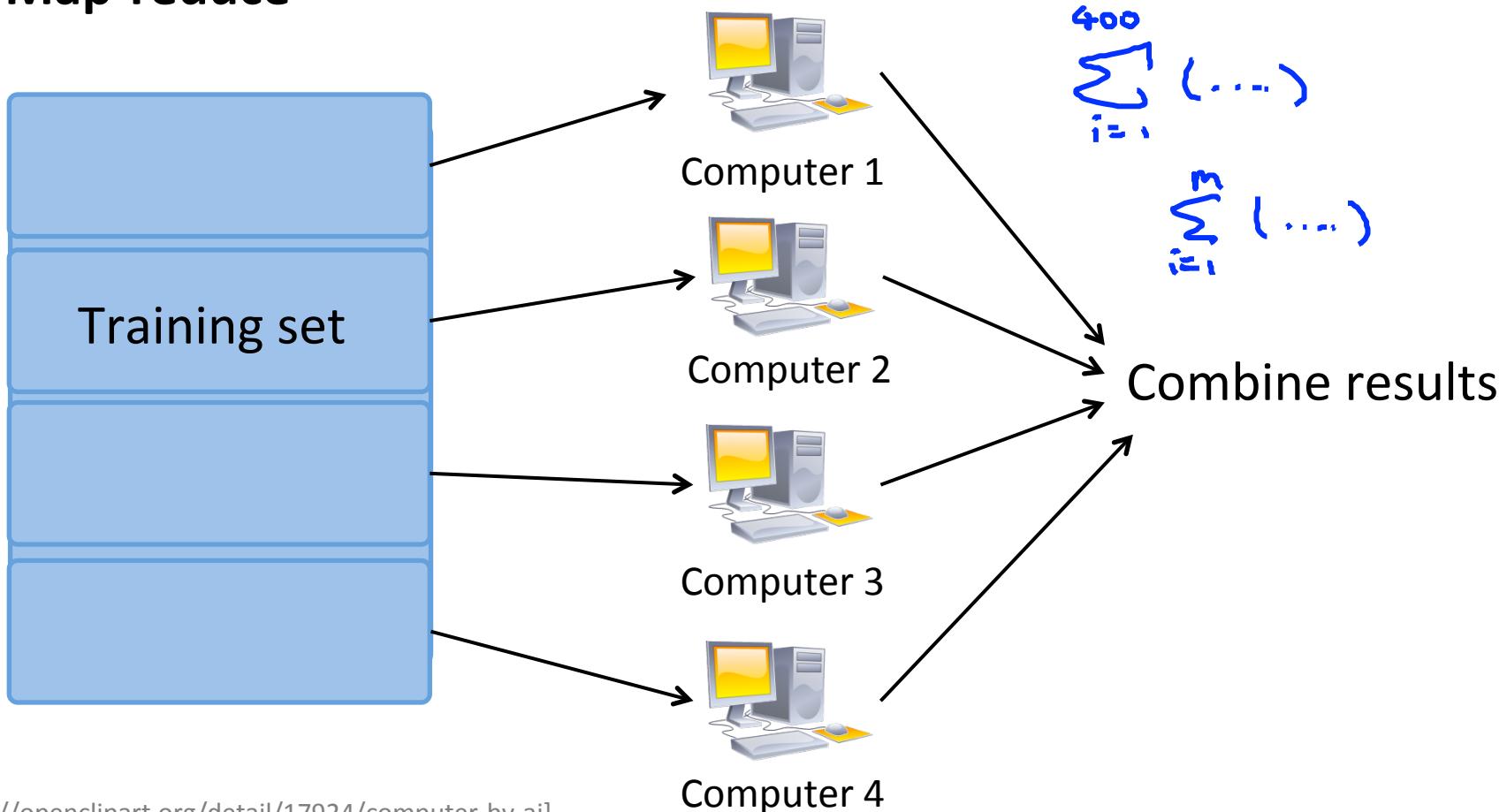
Batch gradient descent:

$$m = 400 \leftarrow m = 400,000,000$$

$$\theta_j := \theta_j - \alpha \frac{1}{400} \sum_{i=1}^{400} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$



# Map-reduce



## Map-reduce and summation over the training set

Many learning algorithms can be expressed as computing sums of functions over the training set.

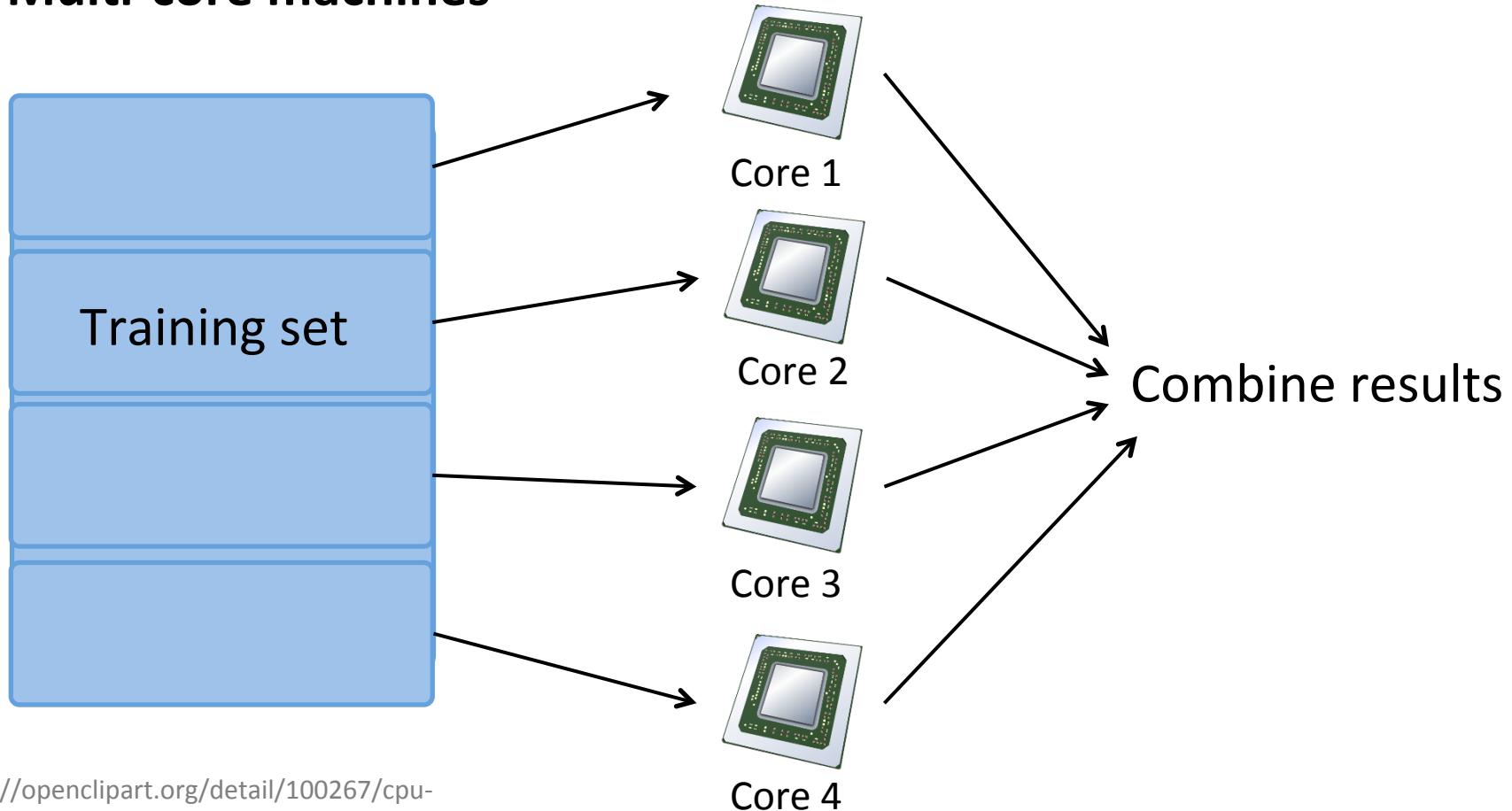
E.g. for advanced optimization, with logistic regression, need:

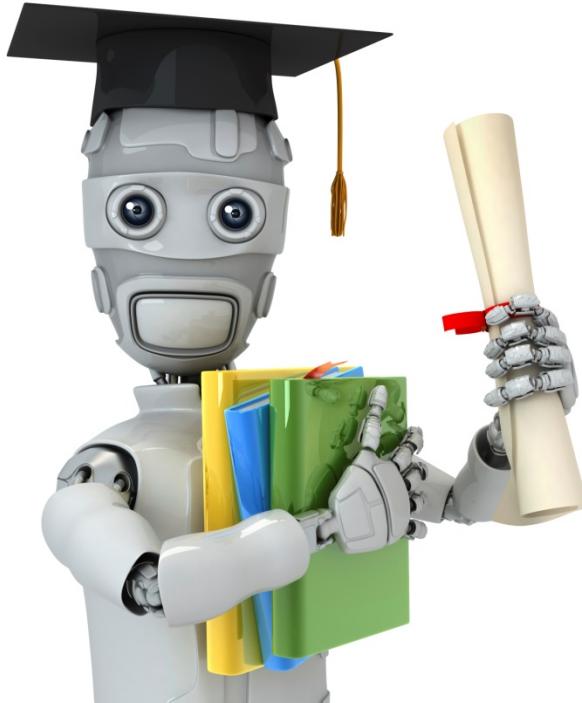
$$\rightarrow \underline{J_{train}(\theta)} = -\frac{1}{m} \sum_{i=1}^m \underbrace{y^{(i)} \log h_\theta(x^{(i)}) - (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))}_{\textcircled{C}}$$

$$\rightarrow \underline{\frac{\partial}{\partial \theta_j} J_{train}(\theta)} = \frac{1}{m} \sum_{i=1}^m \underbrace{(h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}}_{\textcircled{C}}$$

$$+ \text{temp}^{(i)} \quad \text{temp}_j^{(i)} \leftarrow$$

# Multi-core machines





Machine Learning

## Application example: Photo OCR

---

Problem description  
and pipeline

# The Photo OCR problem



# Photo OCR pipeline

- 1. Text detection



- 2. Character segmentation

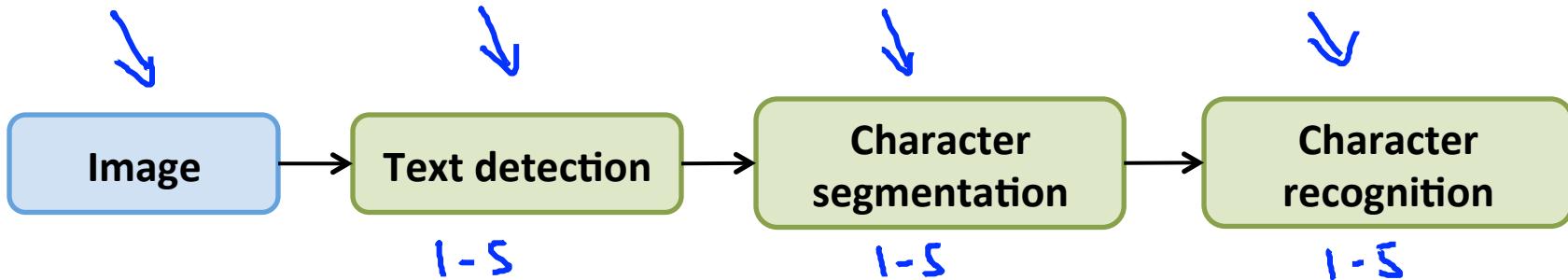


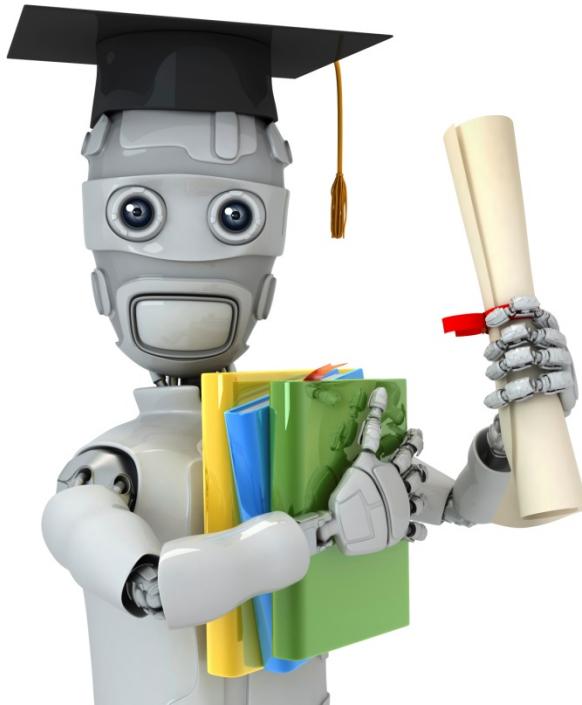
- 3. Character classification



Cleaning → Cleaning

# Photo OCR pipeline





Machine Learning

Application example:  
Photo OCR

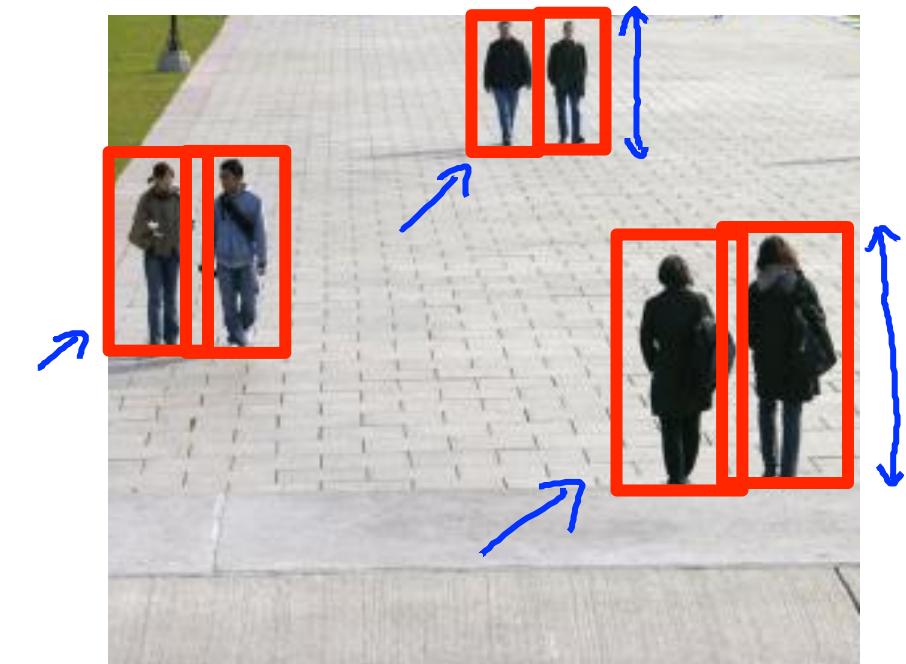
---

Sliding windows

## Text detection



## Pedestrian detection



# Supervised learning for pedestrian detection

$x = \text{pixels in } 82 \times 36 \text{ image patches}$

1,000  
10,000  
...



Positive examples ( $y = 1$ )



Negative examples ( $y = 0$ )

# Sliding window detection



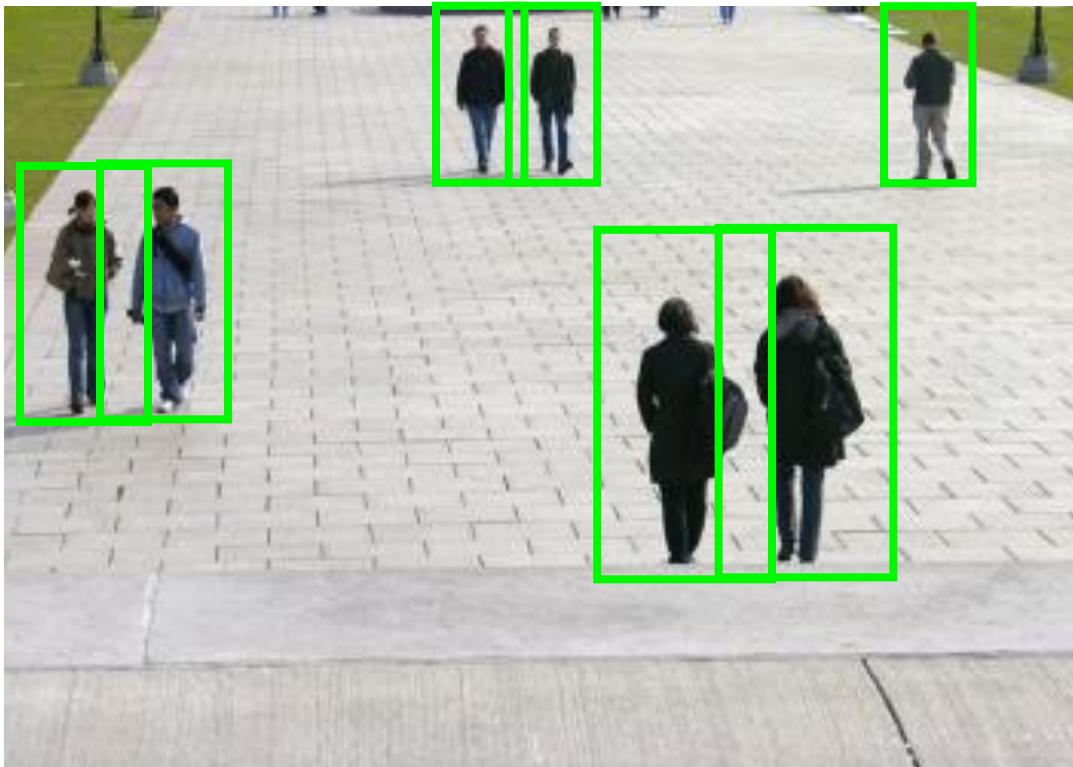
# Sliding window detection



# Sliding window detection



# Sliding window detection



# Text detection



# Text detection



Positive examples ( $y = 1$ )

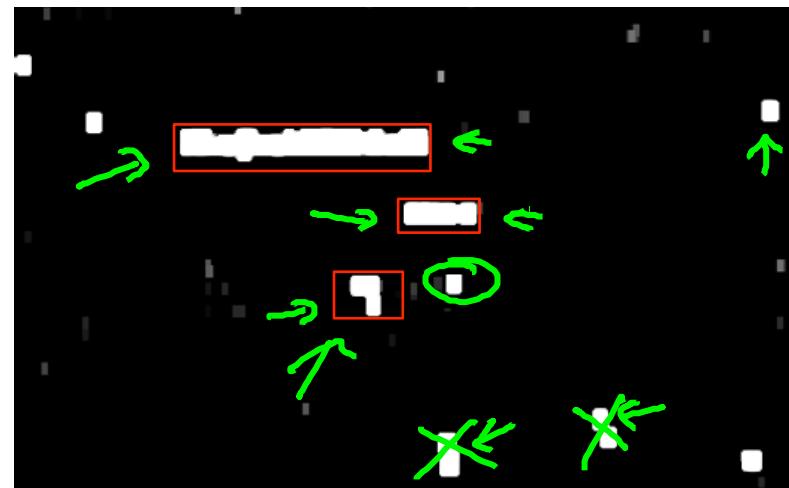


Negative examples ( $y = 0$ )

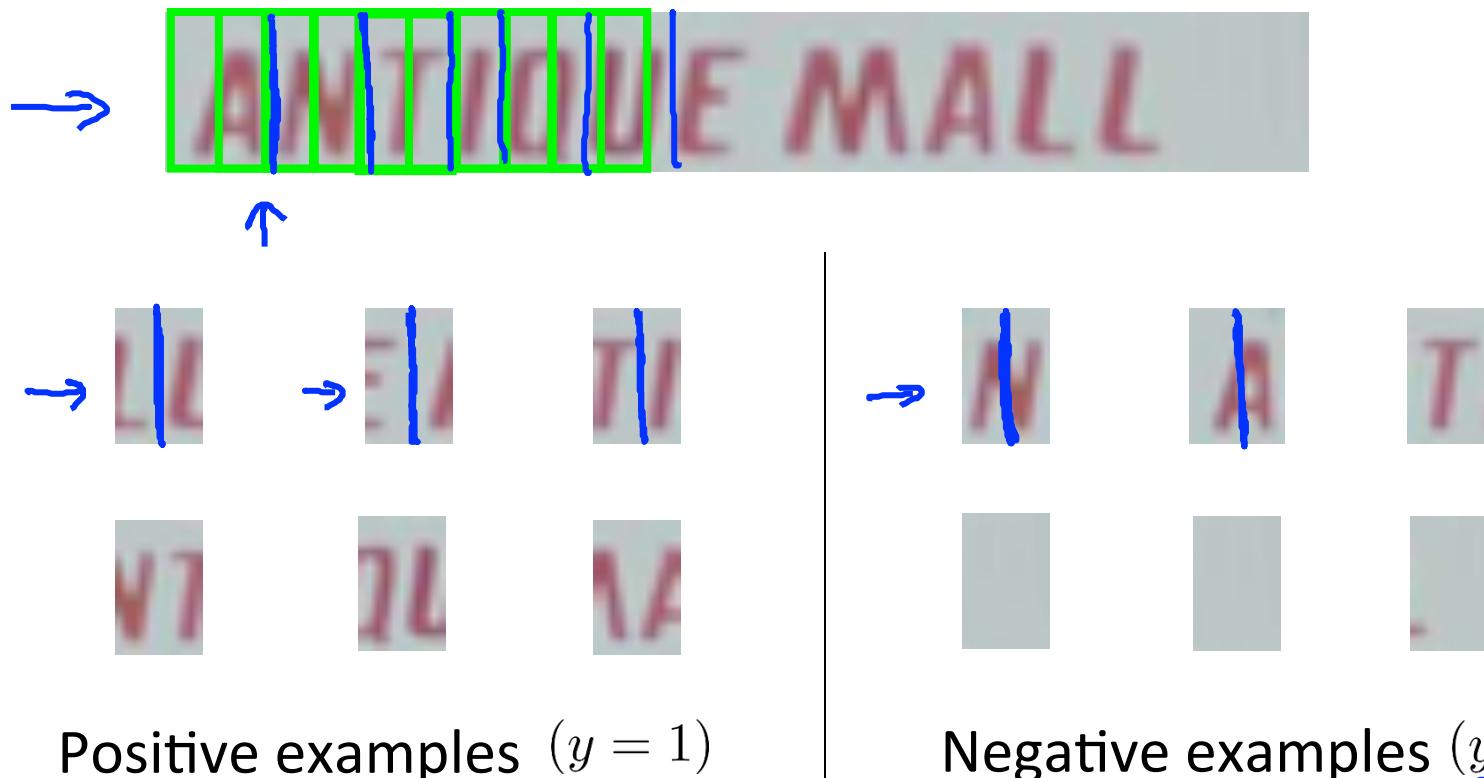
# Text detection



"Expansion"



## 1D Sliding window for character segmentation



# Photo OCR pipeline

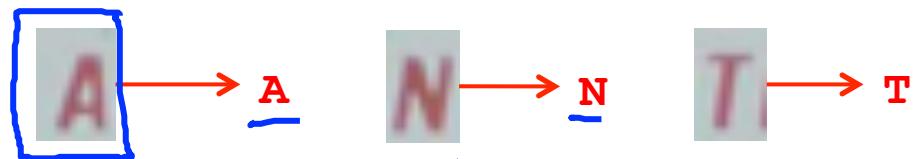
- 1. Text detection

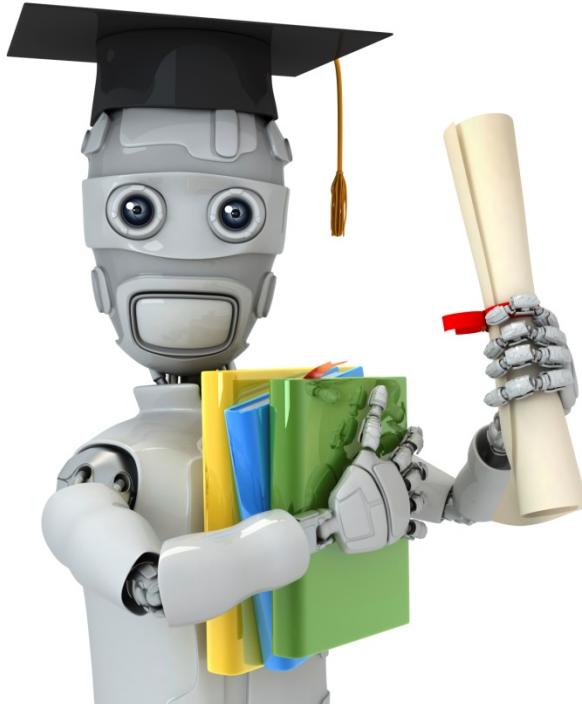


- 2. Character segmentation



- 3. Character classification





Machine Learning

## Application example: Photo OCR

---

Getting lots of  
data: Artificial  
data synthesis

# Character recognition



→ A



→ N



→ T



→ I

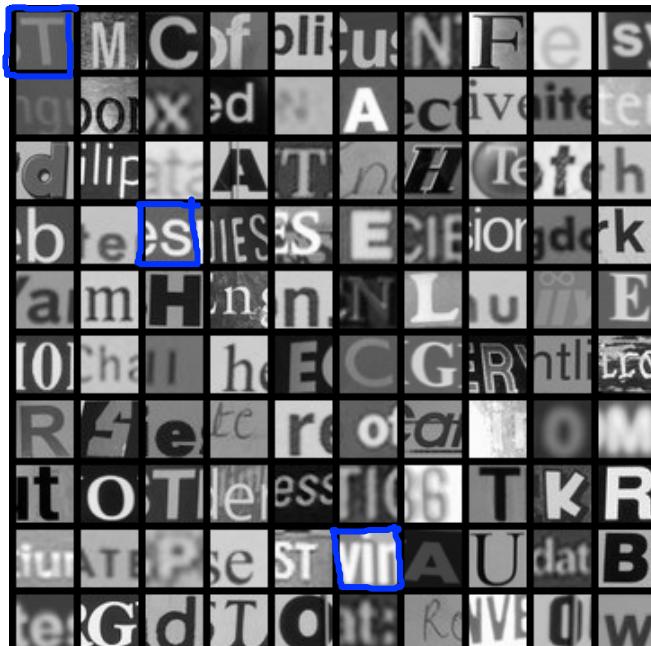


→ Q



→ A

# Artificial data synthesis for photo OCR



Real data

Abcdefg  
Abc<sup>C</sup>defg  
Abcdefg  
Abcdefg  
Abcdefg  
Abcdefg

# Artificial data synthesis for photo OCR

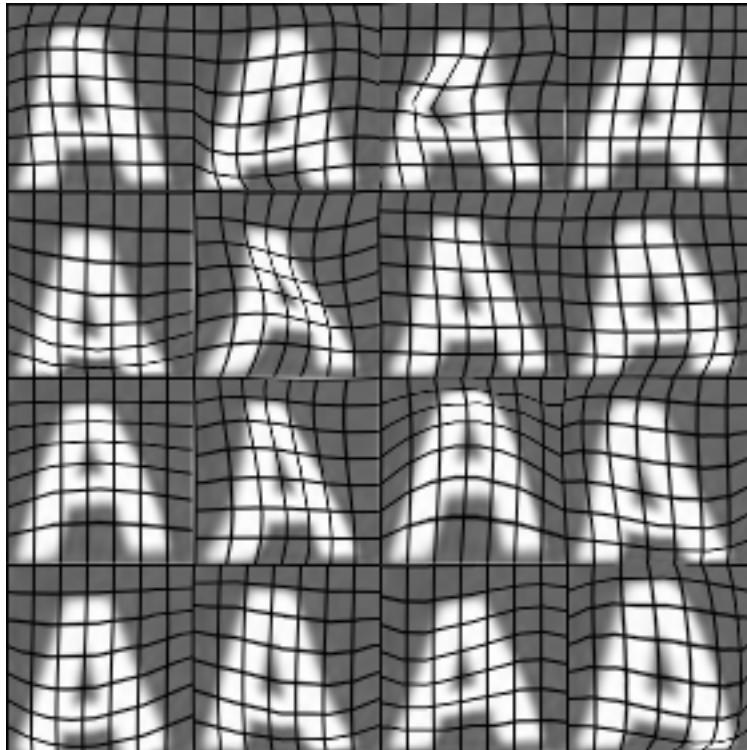
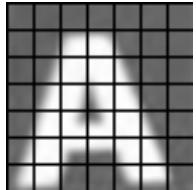


Real data



Synthetic data

# Synthesizing data by introducing distortions



# Synthesizing data by introducing distortions: Speech recognition



Original audio: 



Audio on bad cellphone connection



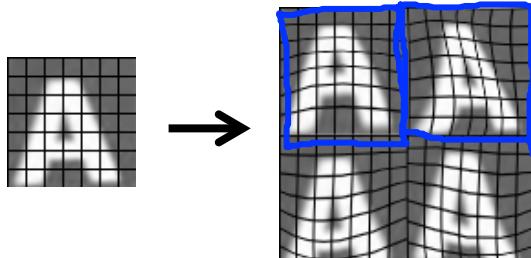
Noisy background: Crowd



Noisy background: Machinery

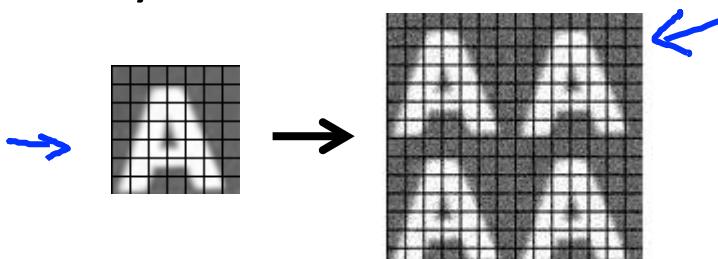
## Synthesizing data by introducing distortions

- Distortion introduced should be representation of the type of noise/distortions in the test set.



→ Audio:  
Background noise,  
bad cellphone connection

- Usually does not help to add purely random/meaningless noise to your data.



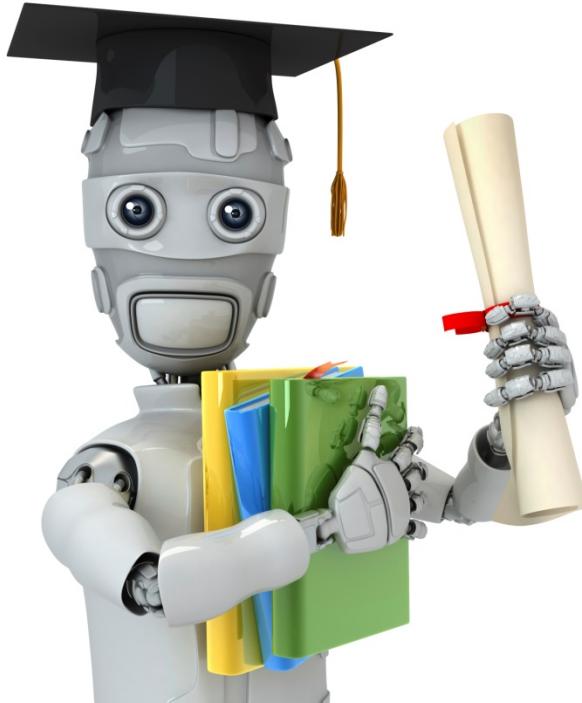
→  $x_i$  = intensity (brightness) of pixel  $i$   
→  $x_i \leftarrow x_i + \text{random noise}$

## Discussion on getting more data

1. Make sure you have a low bias classifier before expending the effort. (Plot learning curves). E.g. keep increasing the number of features/number of hidden units in neural network until you have a low bias classifier.
  2. “How much work would it be to get 10x as much data as we currently have?”
    - Artificial data synthesis
    - Collect/label it yourself
    - “Crowd source” (E.g. Amazon Mechanical Turk)
- #hours?
- $m = 1,000$
- $\rightarrow 10 \text{ secs/example}$
- $m = 10,000$

## Discussion on getting more data

1. Make sure you have a low bias classifier before expending the effort. (Plot learning curves). E.g. keep increasing the number of features/number of hidden units in neural network until you have a low bias classifier.
2. “How much work would it be to get 10x as much data as we currently have?”
  - Artificial data synthesis
  - Collect/label it yourself
  - “Crowd source” (E.g. Amazon Mechanical Turk)



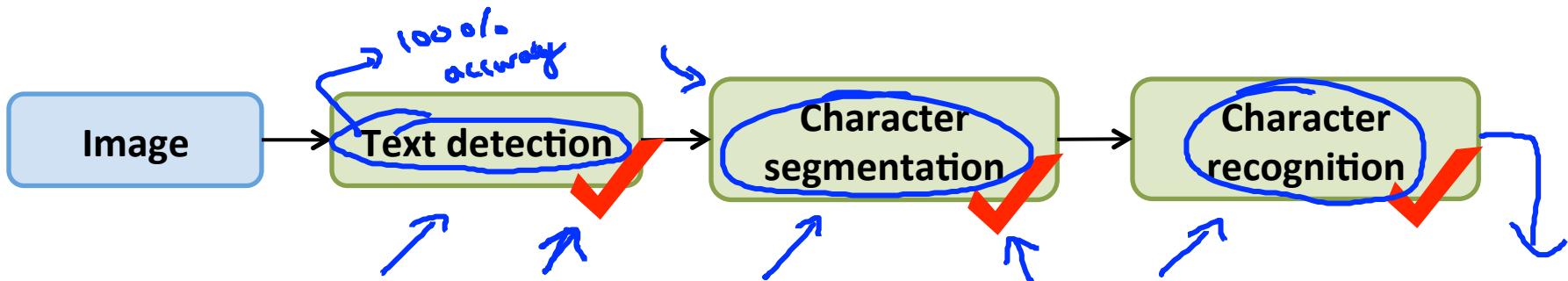
Machine Learning

## Application example: Photo OCR

---

Ceiling analysis: What  
part of the pipeline to  
work on next

## Estimating the errors due to each component (ceiling analysis)



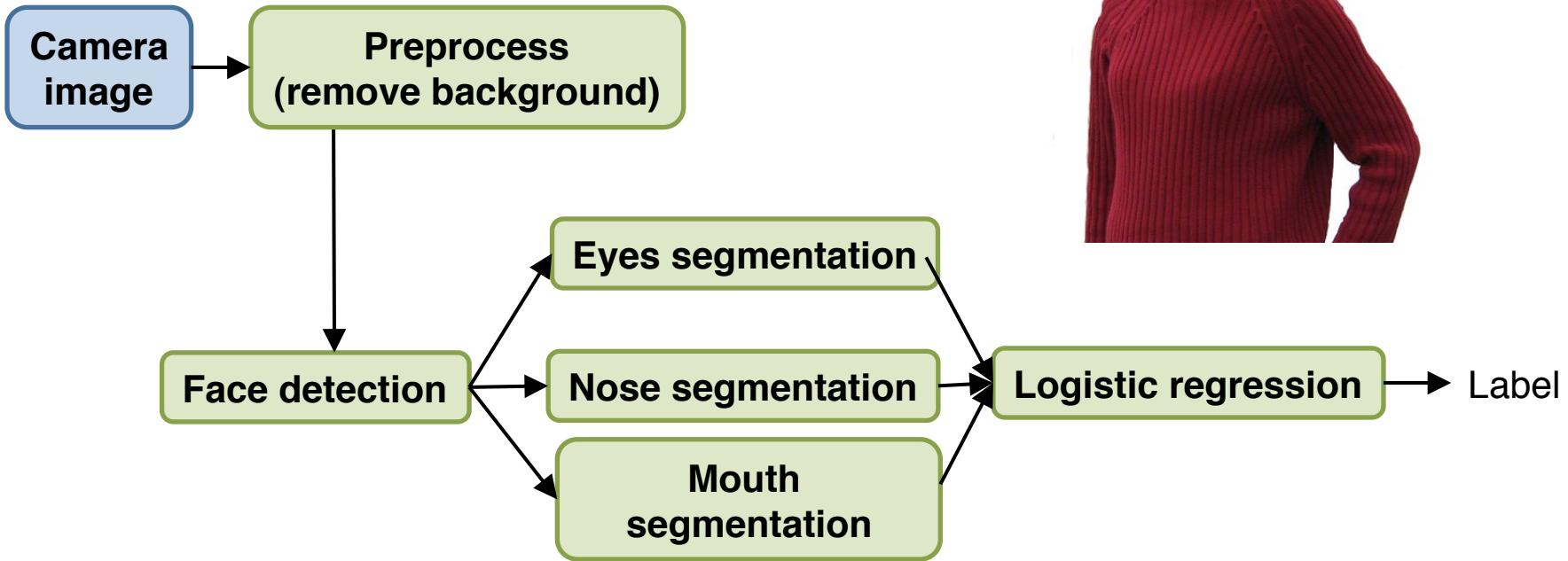
What part of the pipeline should you spend the most time trying to improve?

Component	Accuracy
Overall system	72%
→ Text detection	89%
Character segmentation	90%
Character recognition	100%

Annotations: Blue arrows point from the 'Accuracy' column to the 'Overall system' and 'Text detection' rows. Handwritten blue arrows show a downward arrow from '72%' to '89%', another from '89%' to '90%', and a final one from '90%' to '100%'. Handwritten blue numbers '17%', '1%', and '10%' are placed next to the downward arrows.

# Another ceiling analysis example

Face recognition from images  
(Artificial example)



# Another ceiling analysis example

