ML note of Andrew Ng's course

Table of Contents

W1

1. Fundamentals

- **Supervised Learning**

Regression (prediction)

Classification

...

- **Unsupervised Learning**

Clustering (grouping)

Non-clustering

...

2. Model Representation

Glossary #1

m: the size of training set

$x^{(i)}$: input of the training set

$y^{(i)}$: output of the training set

Linear Regression Problem:


where $h_\theta(x) = \theta_0 + \theta_1 x$

3. Cost Function

a. definition

Training dataset input x and output y known, we need to find θ_0 and θ_1 s.t.

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

get the minimum value

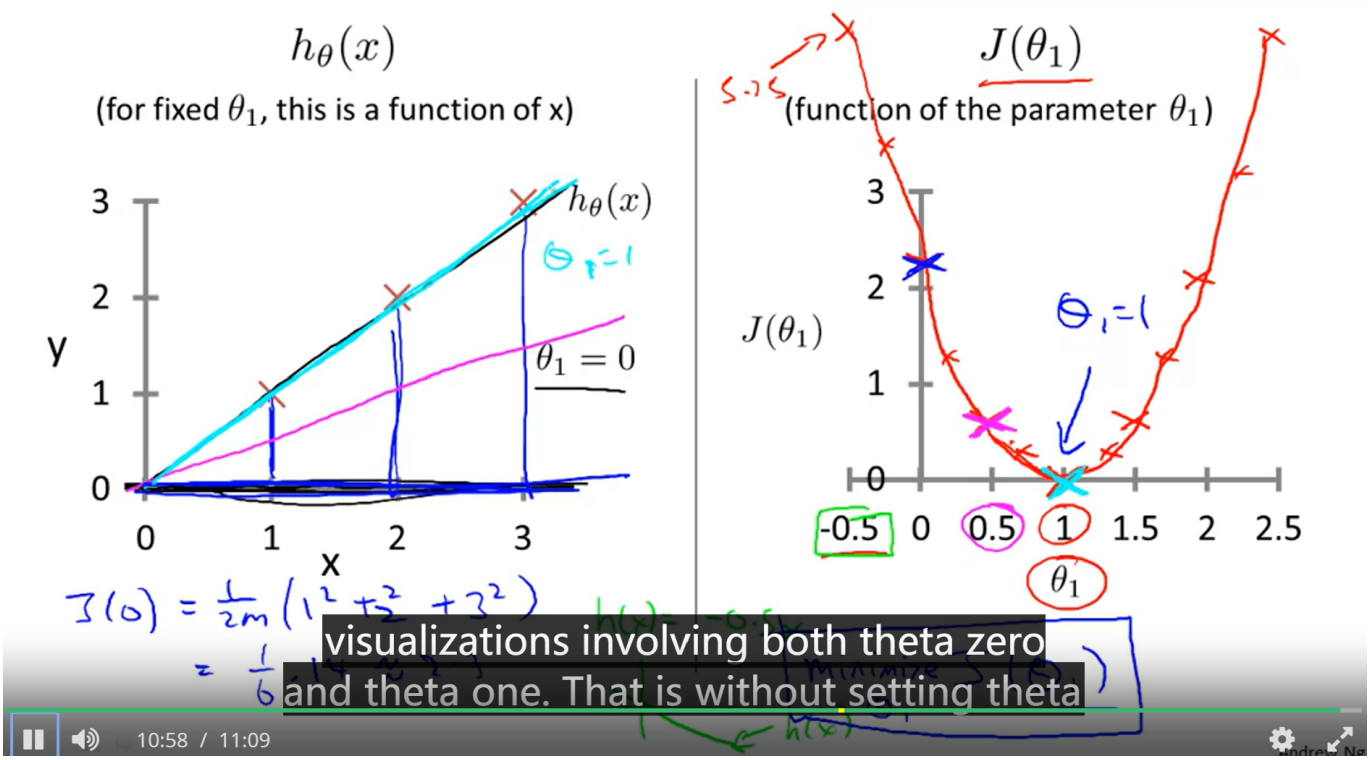
formally, we are going to find

$$\underset{\theta_0, \theta_1}{\operatorname{argmin}} J\left(\theta_0, \theta_1\right)$$

where $J(\theta_0, \theta_1)$ is called **cost function** or **square error cost function**

b. intuition

1. simplified problem:



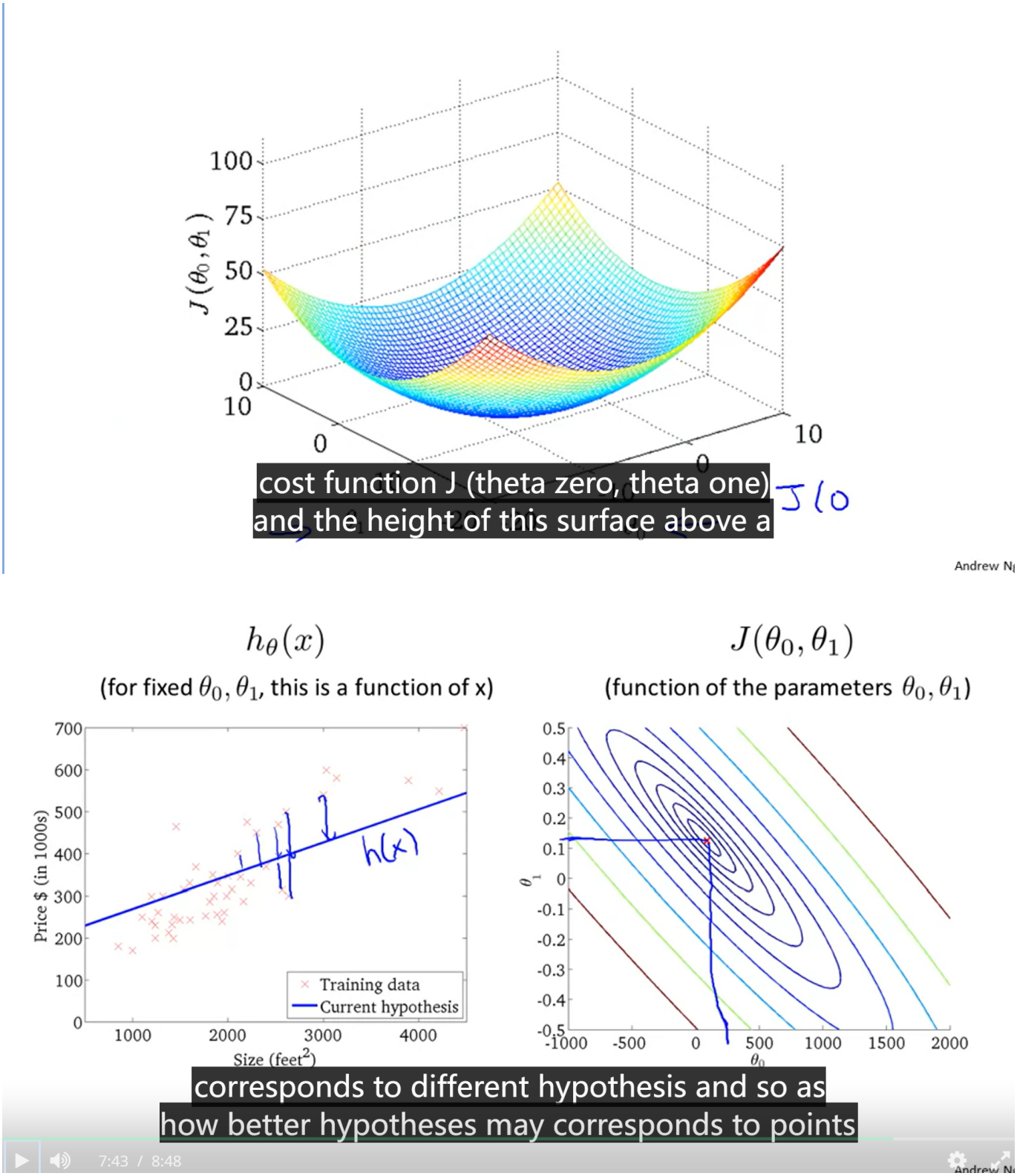
$\theta_0 = 0$

dataset: (1,1), (2,2), (3,3)

the plot of function J is on the right. To solve this question, we have to find θ_1 s.t. $\frac{\mathrm{d}\,J}{\mathrm{d}\,\theta_1} = 0$

and evidently $\theta_1 = 1$

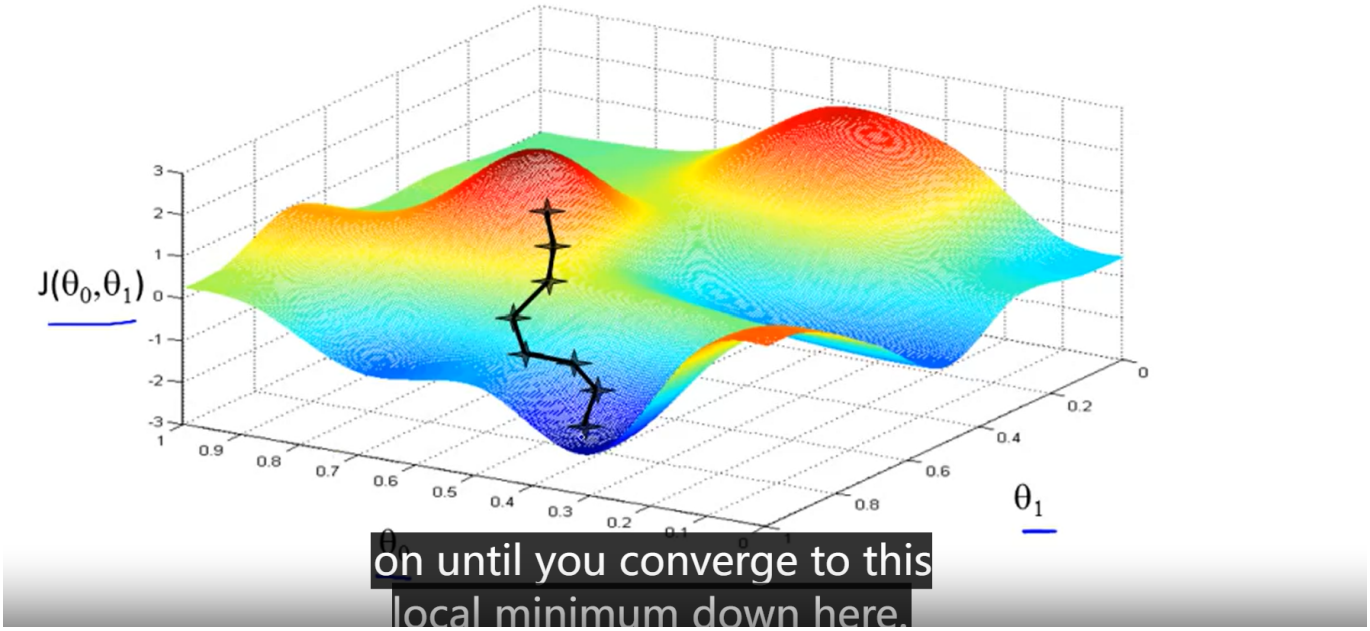
2. original problem:



3-D figure and it is projected to $\theta_0 O \theta_1$ as a **contour plot**. At the current point shown in the figure above, we get the minimum of $J(\theta_0, \theta_1)$, thus find the best $h_\theta(x)$ as the hypothesis function.

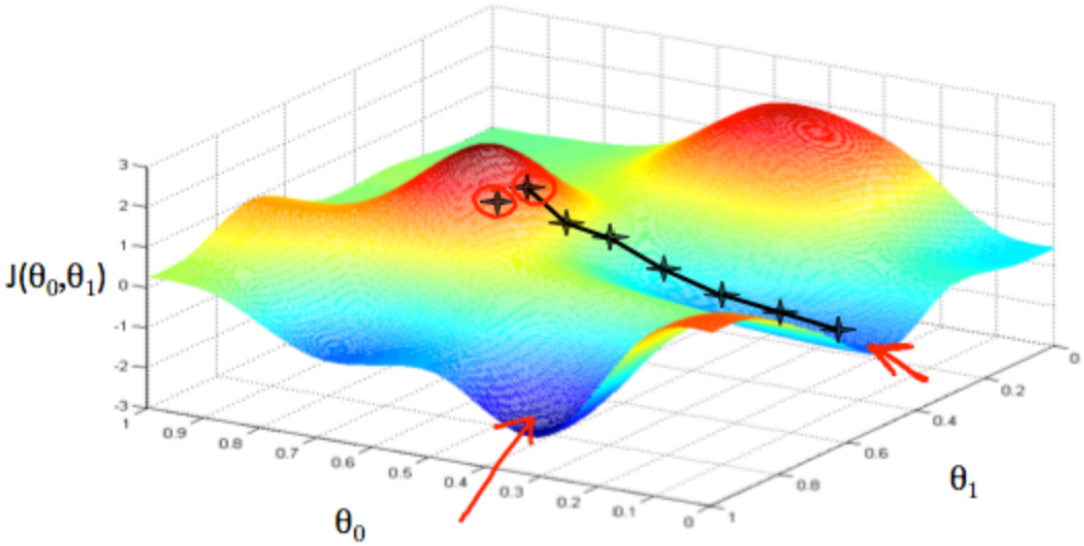
4. Parameter Learning (gradient descent)

a. introduction

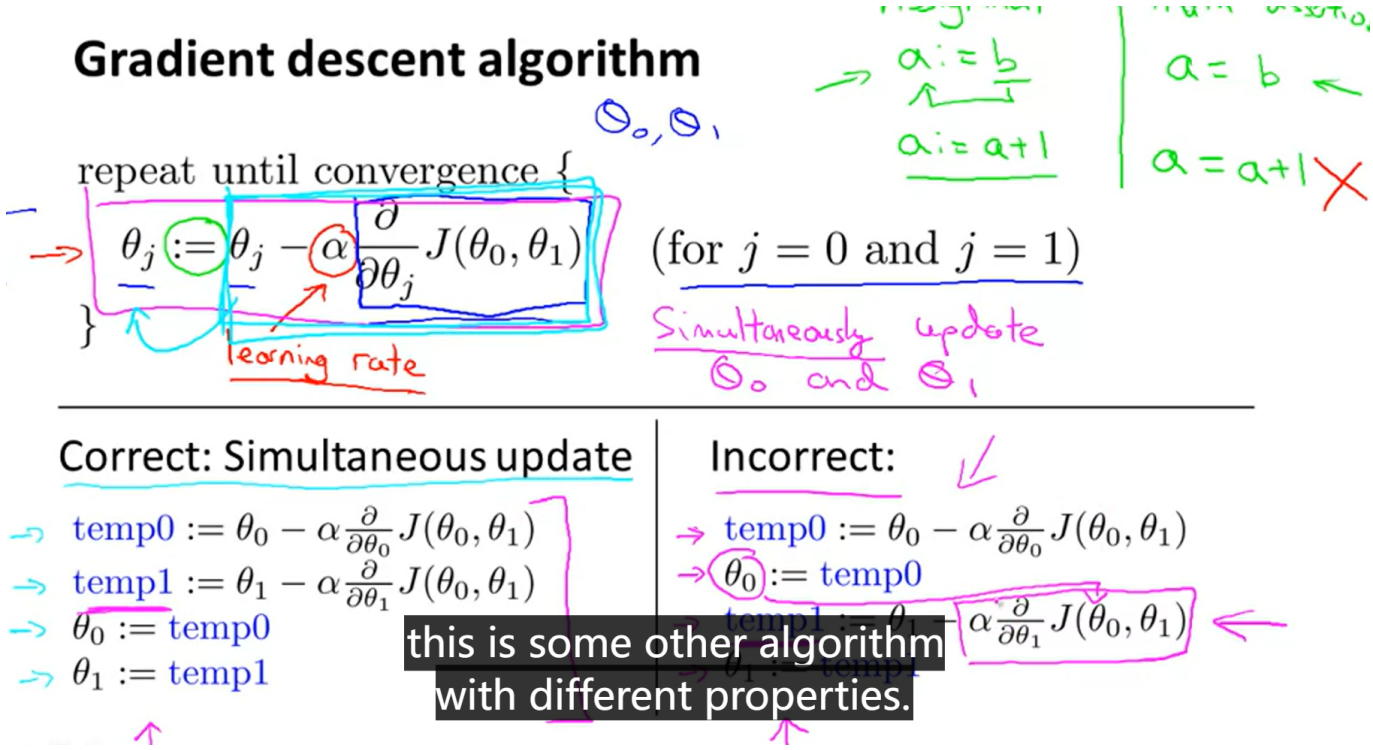


go under the hill on the **steepest way**(partial derivative)

property:



Local optima? Global optima?



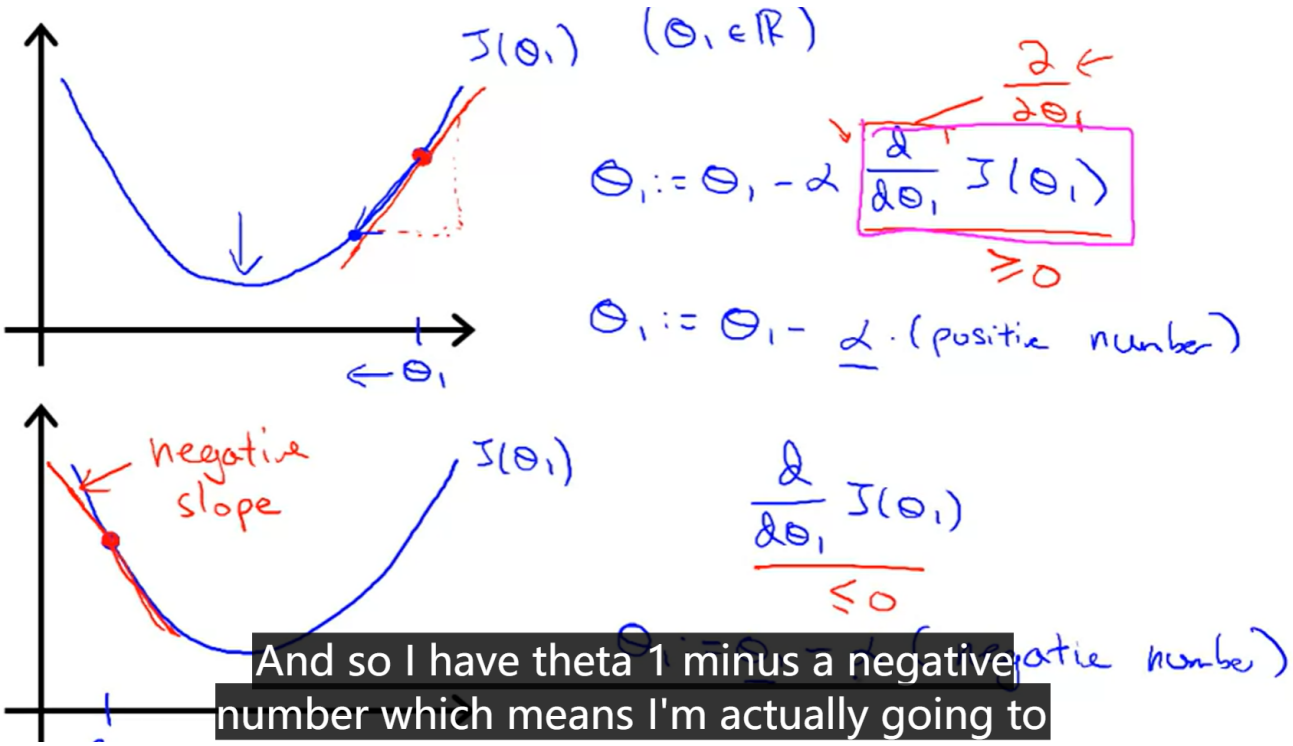
the mathematical implementation of GD

where:

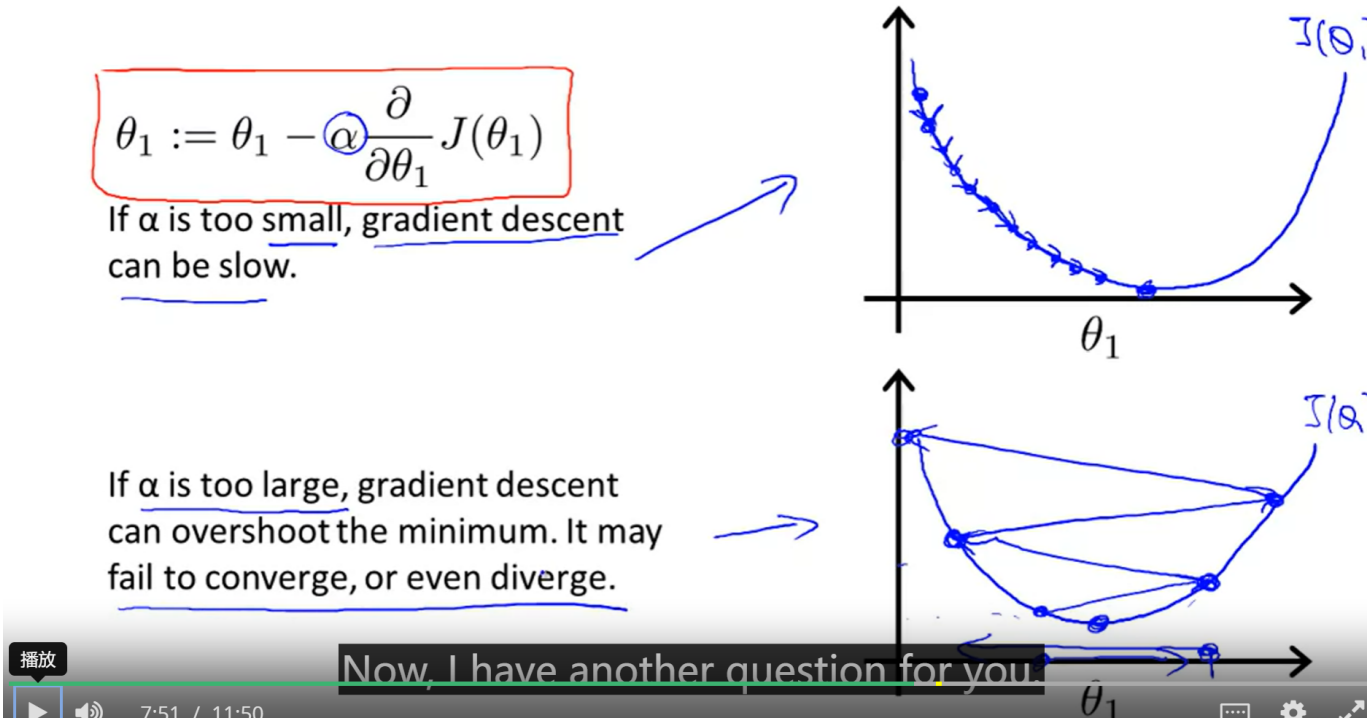
α learning rate: how big a step will be

WARNING: FOCUS ON THE CORRECT UPDATE VERSION

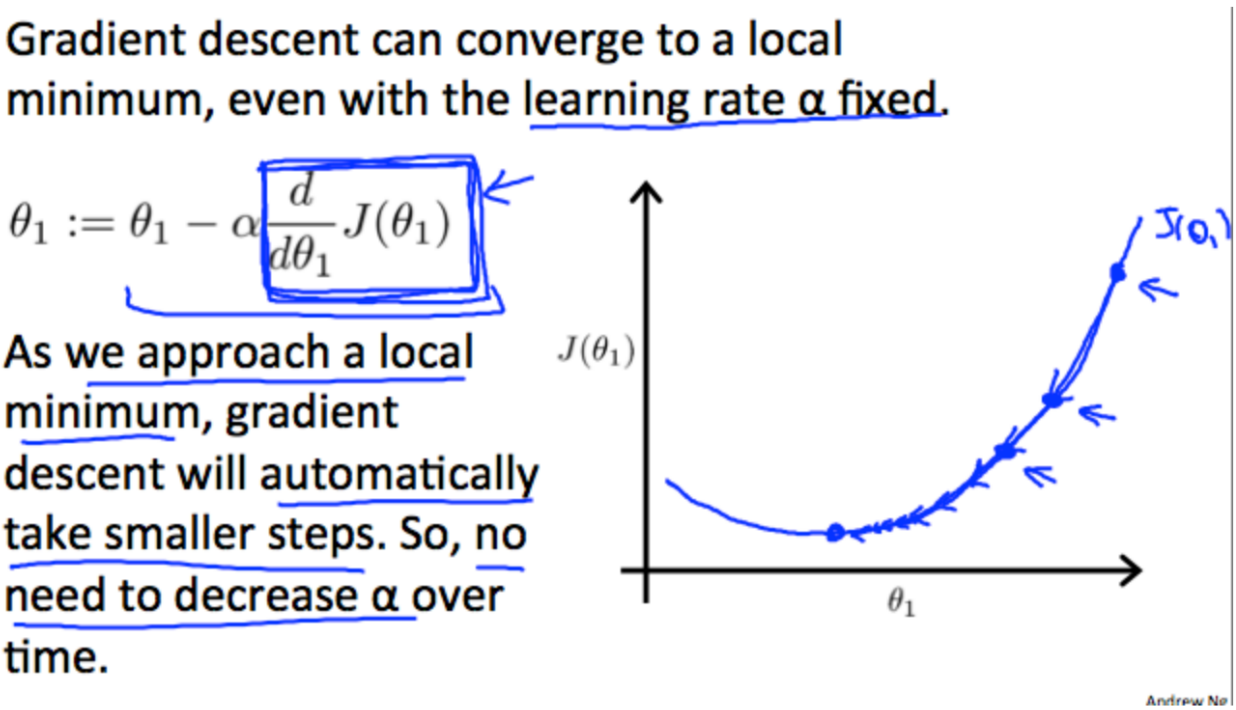
what the derivative does:



what α does:

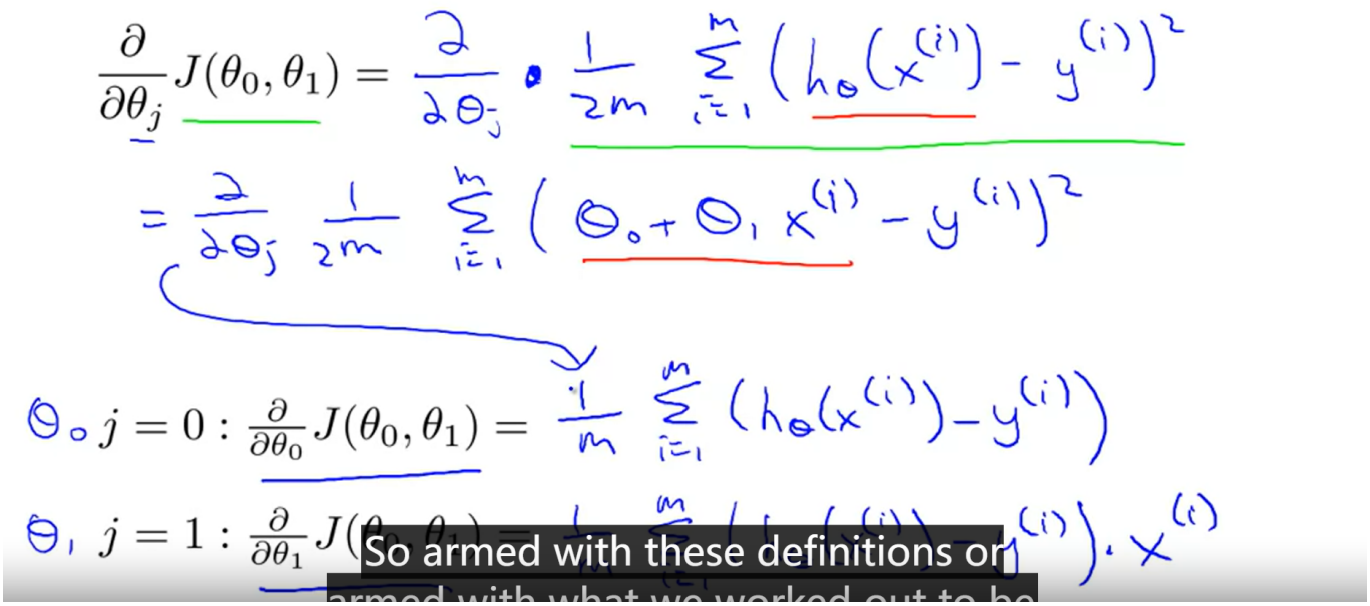


if θ_1 is already the optimal minimum, the algorithm just leave it unchanged, so we get:



b. GD for linear regression

the derivative:



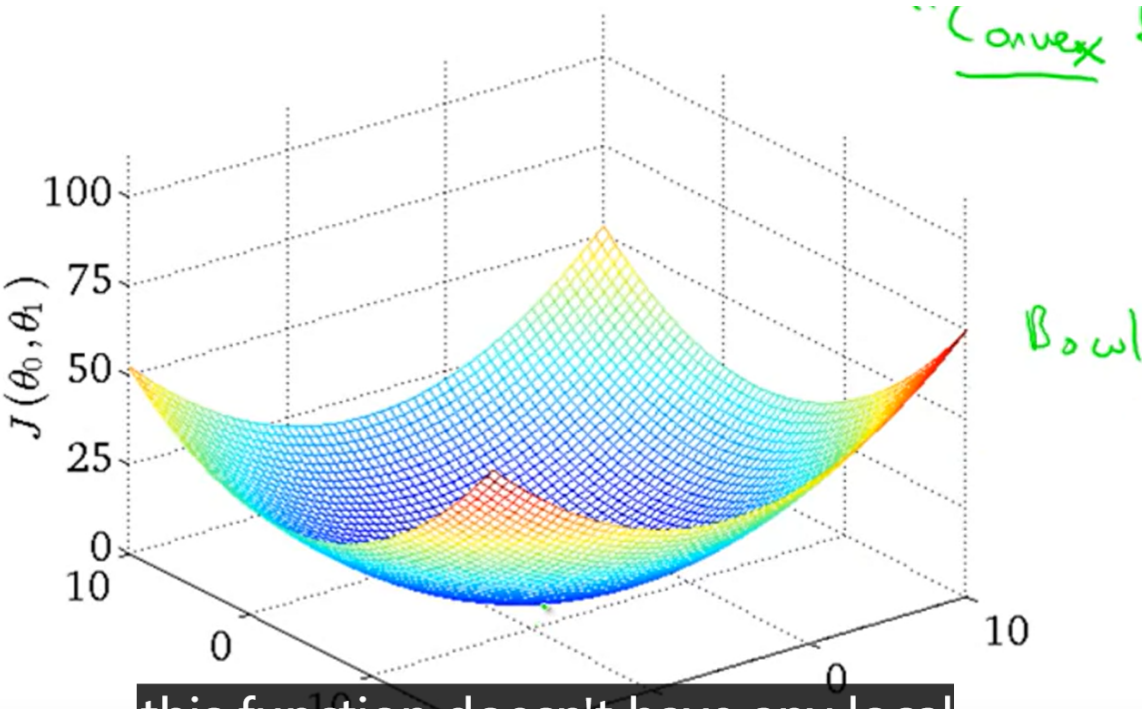
plug back to the GD algorithm:

$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\quad \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \\
&\quad \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)} \\
&\}
\end{aligned}
$$

An example to calculate this(θ_1)

$$
\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} \left(h_\theta(x) - y \right)^2 \\
&= 2 \cdot \frac{1}{2} \left(h_\theta(x) - y \right) \cdot \frac{\partial}{\partial \theta_j} \left(h_\theta(x) - y \right) \\
&= \left(h_\theta(x) - y \right) \cdot \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y \right) \\
&= \left(h_\theta(x) - y \right) x_j
\end{aligned}
$$

the cost function of linear regression is always a **convex function** like this



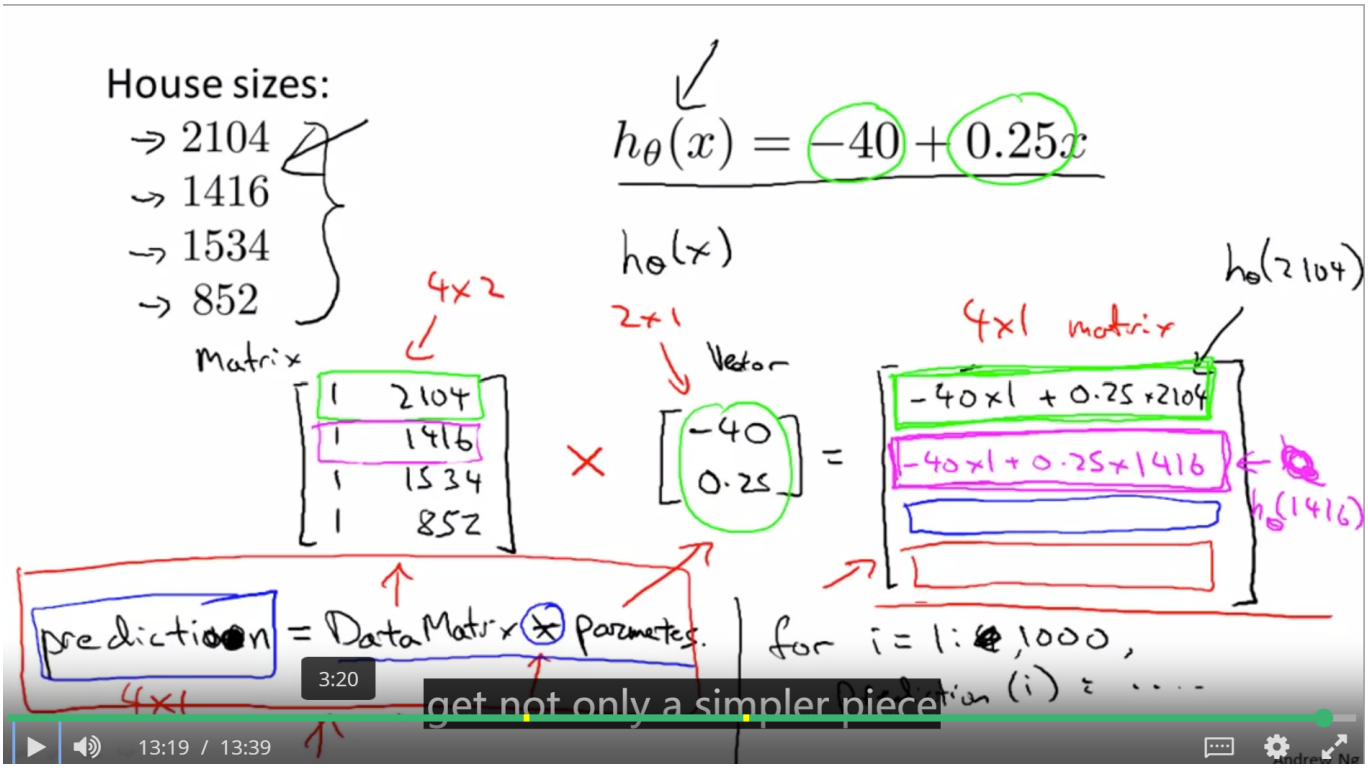
5. Linear Algebra Fundamentals(Review)

a. matrices and vectors(trivial)

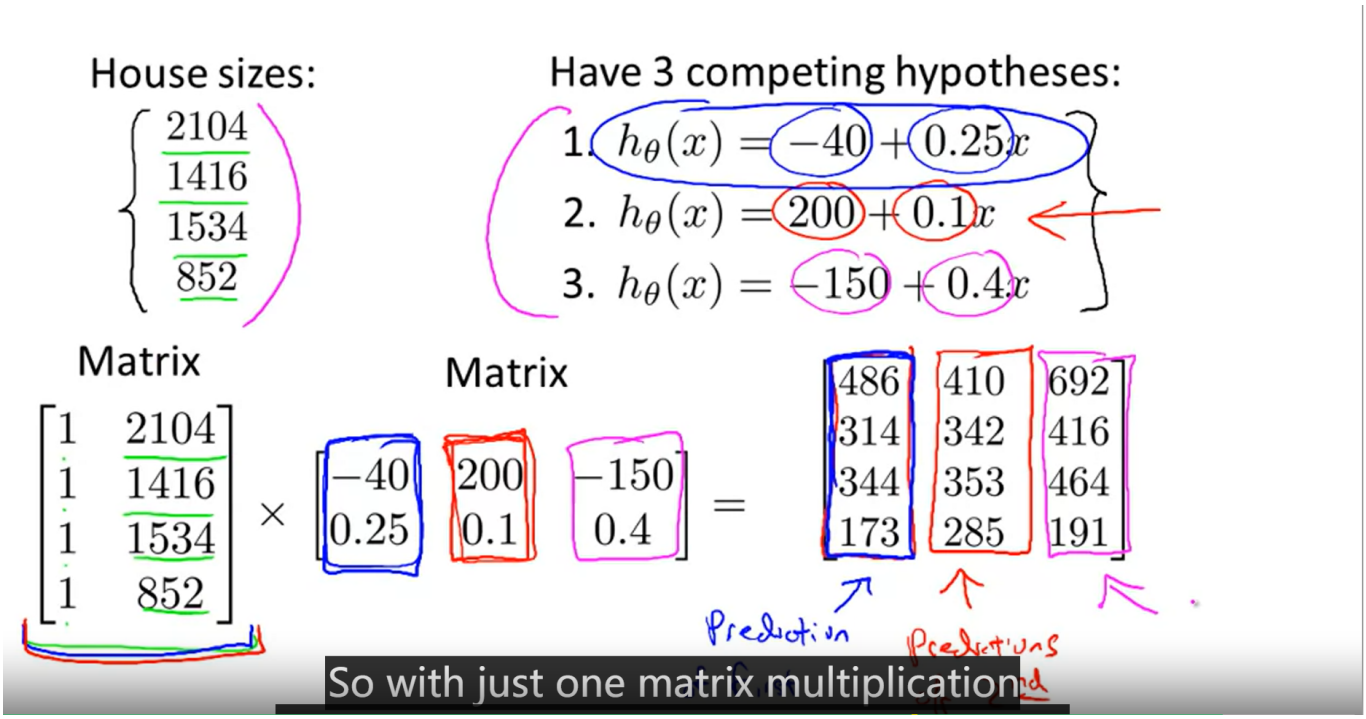
b. addition and scalar multiplication(trivial)

c. matrix matrix multiplication

neat trick to do prediction by one hypothesis function



another neat trick in the situation of many hypothesis functions



properties:

1. Not commutative;
2. Associative;
3. Identity matrix is commutative;

d. inverse and transpose(trivial)

W2

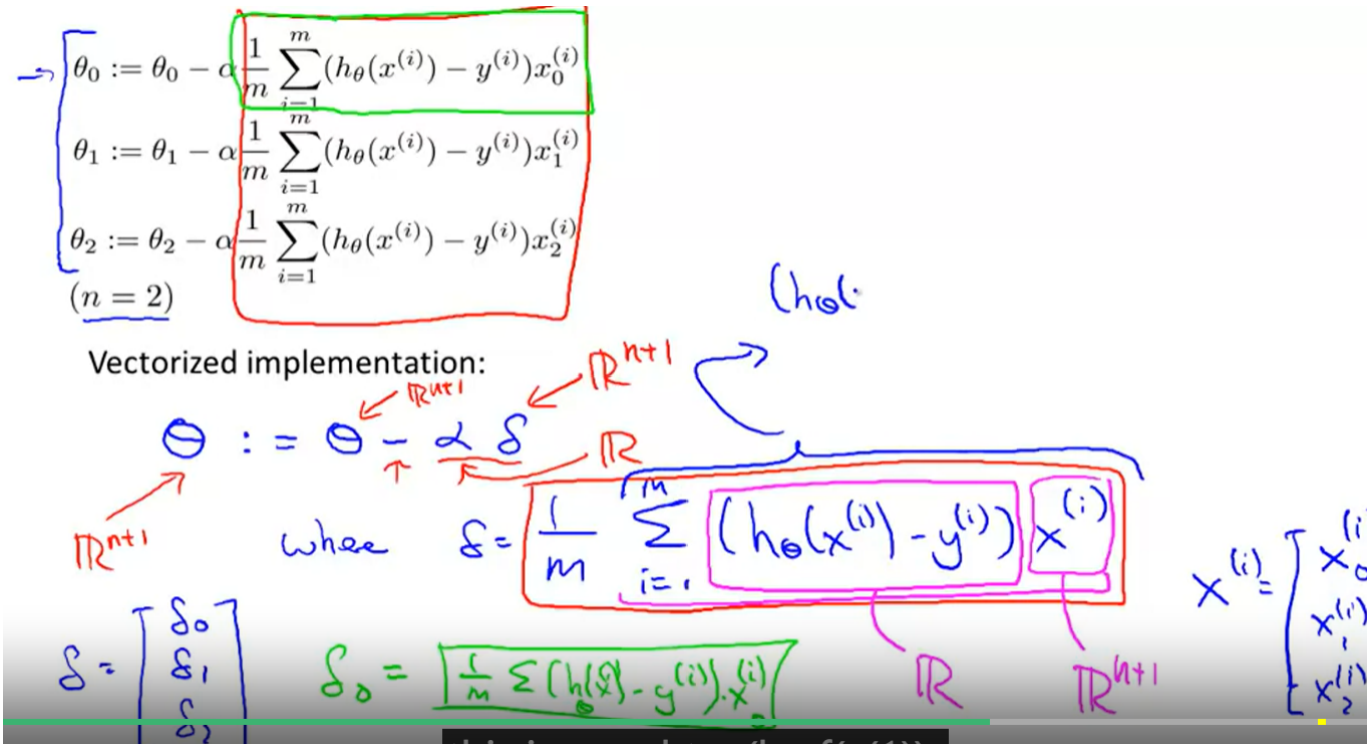
1. Vectorization(intro)

trick 1

$$h_\theta(x) = \sum_{j=\theta}^{n} \theta_j x_j$$

$$= \theta^T x$$

trick 2



note: treat θ as a vector, Σ blah blah x also a vector.

2. Multivariate Linear Regression

Glossary #2

n: number of features

$x^{(i)}$: the i^{th} entry of the training set

$x_j^{(i)}$: the value of feature j of $x^{(i)}$

a. hypothesis function of MLR

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + ... + \theta_n x_n$$

let $x_0 = 1$

let $x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$

let $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$

thus we have

$$h_\theta(x) = \theta^T x$$

b. GD for multiple variables

just the same thing...

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

Parameters: $\theta_0, \theta_1, \ldots, \theta_n$ (n+1 - dimensional vector)

Cost function:

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

Gradient descent:

Repeat {

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

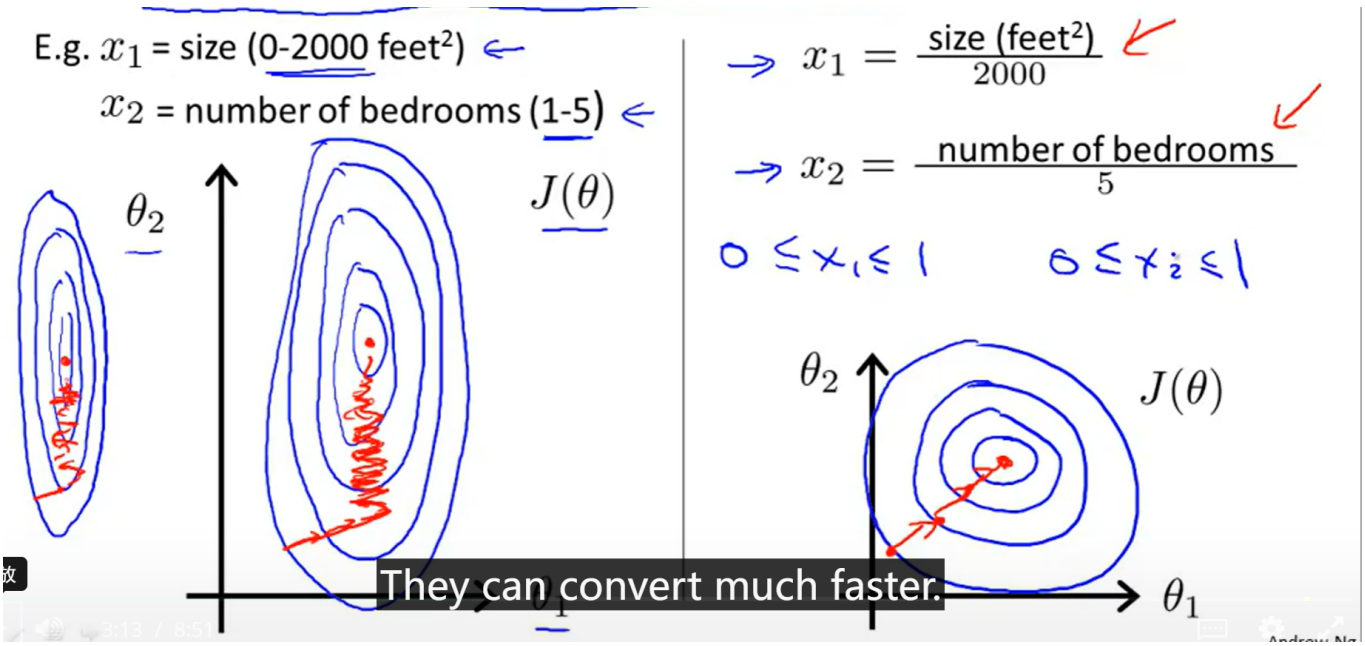
}

(simultaneously update for every $j = 0, \ldots, n$)

c. practical tricks for GD of MLR

1. feature scaling(特征缩放)

idea: make sure features are on the similar scale



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

$x_0 = 1$

$0 \leq x_1 \leq 3$ ✓

$-2 \leq x_2 \leq 0.5$ ✓

$-100 \leq x_3 \leq 100$ ✗

$-0.0001 \leq x_4 \leq 0.0001$ ✗

Mean normalization

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

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

Average size = 100

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

1-5 bedrooms

$-0.5 \leq x_1 \leq 0.5$, $-0.5 \leq x_2 \leq 0.5$

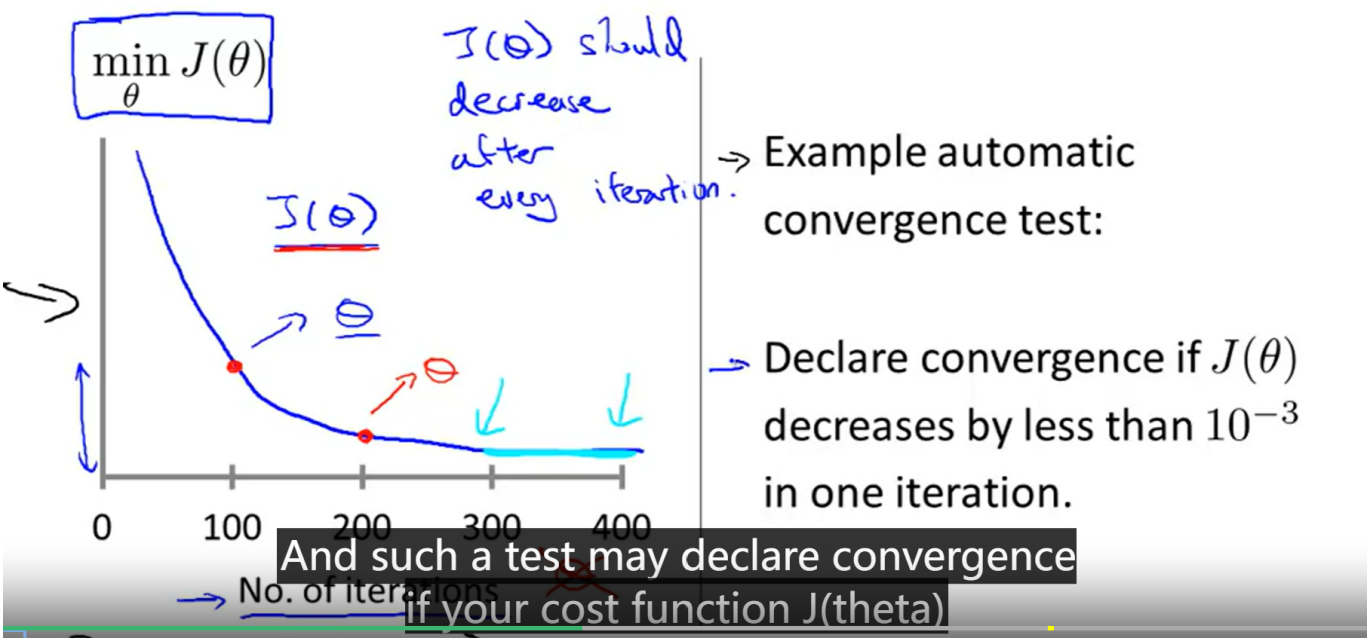
$x_1 \leftarrow \frac{x_1 - \mu_1}{s_1}$ (μ_1: avg value of x_1 in training set; s_1: range (max − min) or standard deviation)

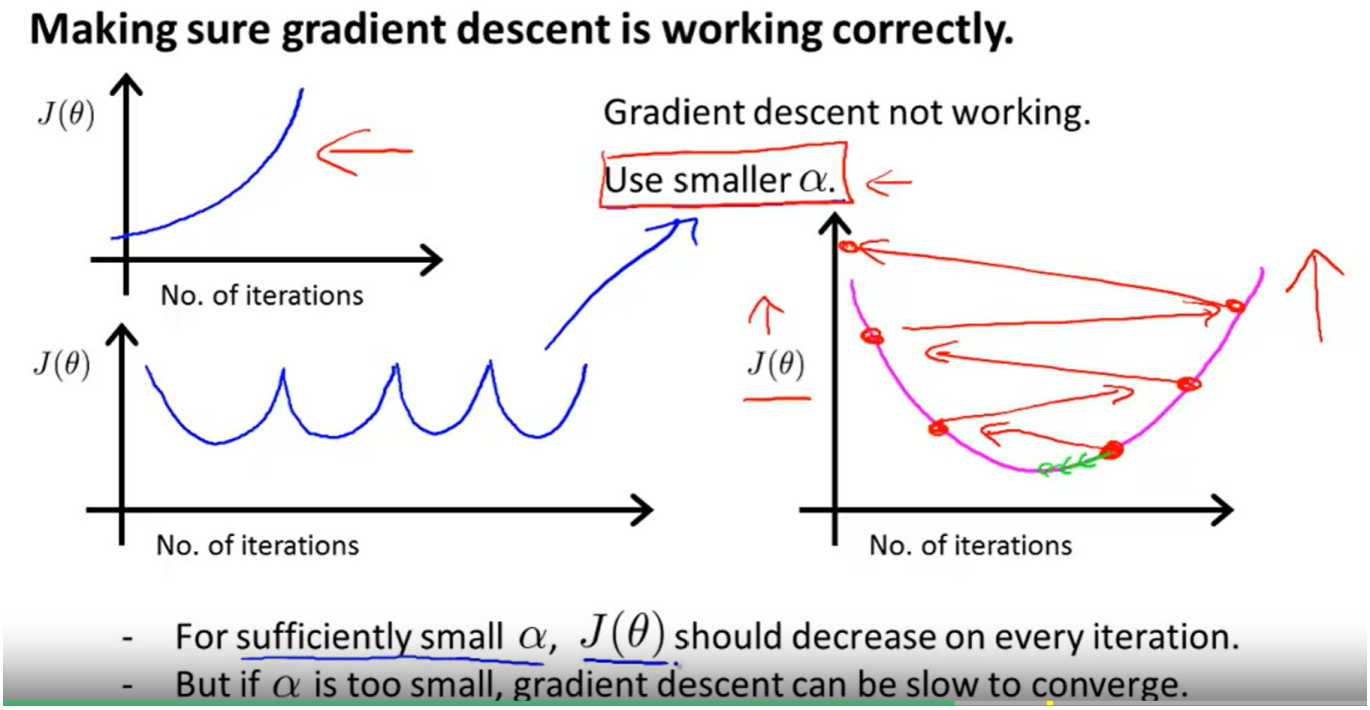
$x_2 \leftarrow \frac{x_2 - \mu_2}{s_2}$

doesn't have to be too exact.

8:27 / 8:51

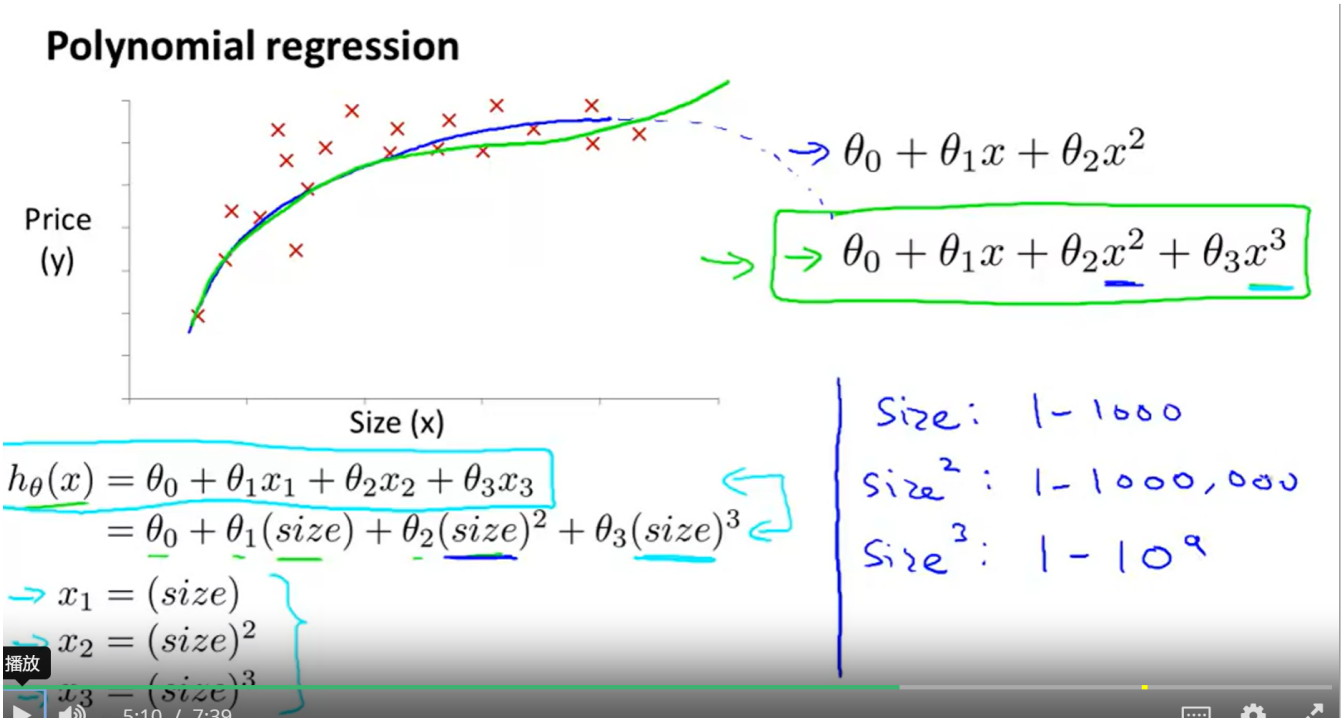
2. learning rate



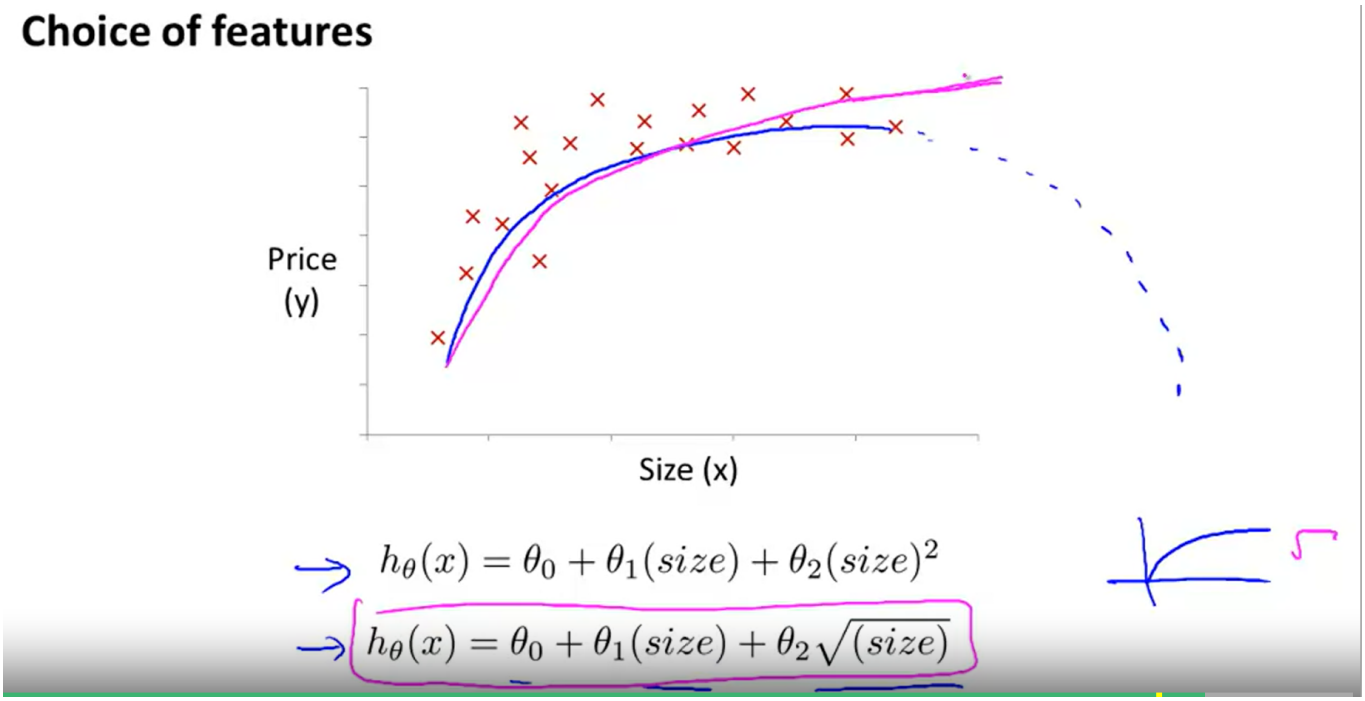


α can neither be too small nor too big

d. Features and Polynomial Regression



note: treat the $size, size^2, size^3$ as x_1, x_2, x_3 and the problem is equivalent to MLR



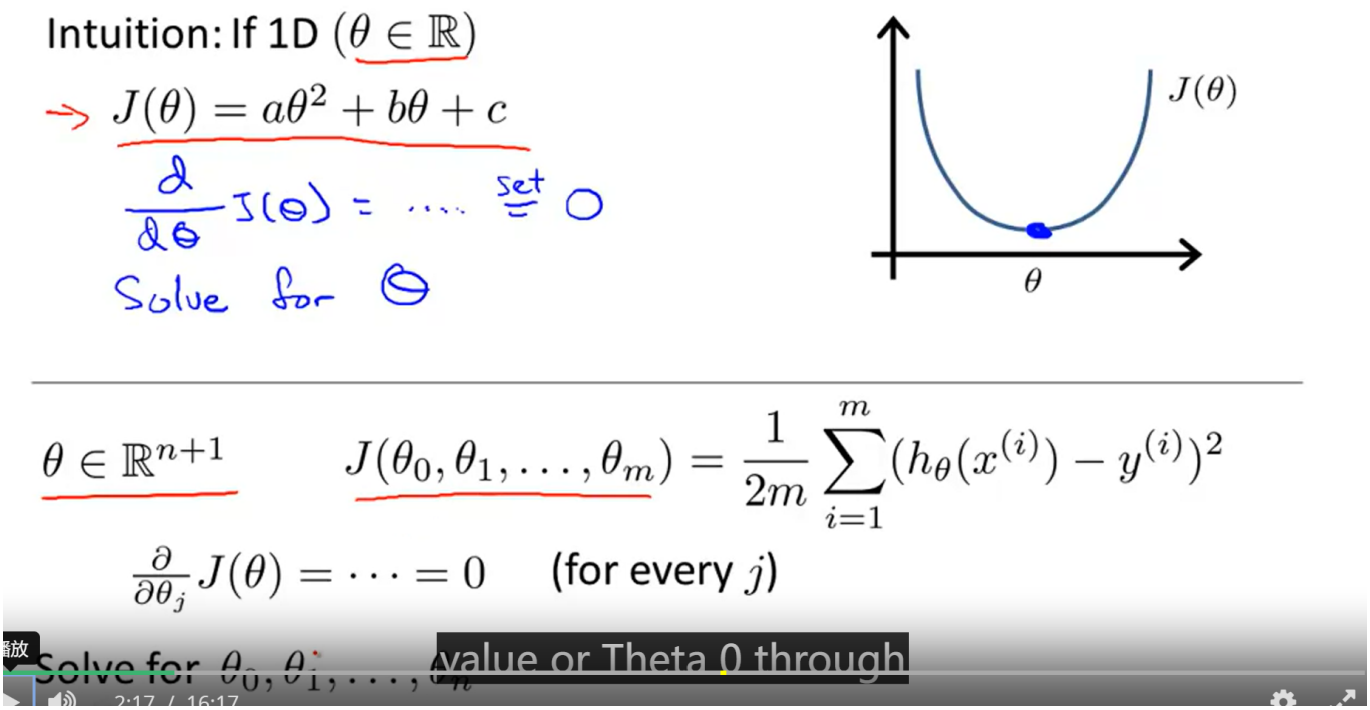
note: an algorithm will help us to choose a feature

idea:

1. we can combine various features to one feature
2. we can create new feature
3. we have multiple choices of features

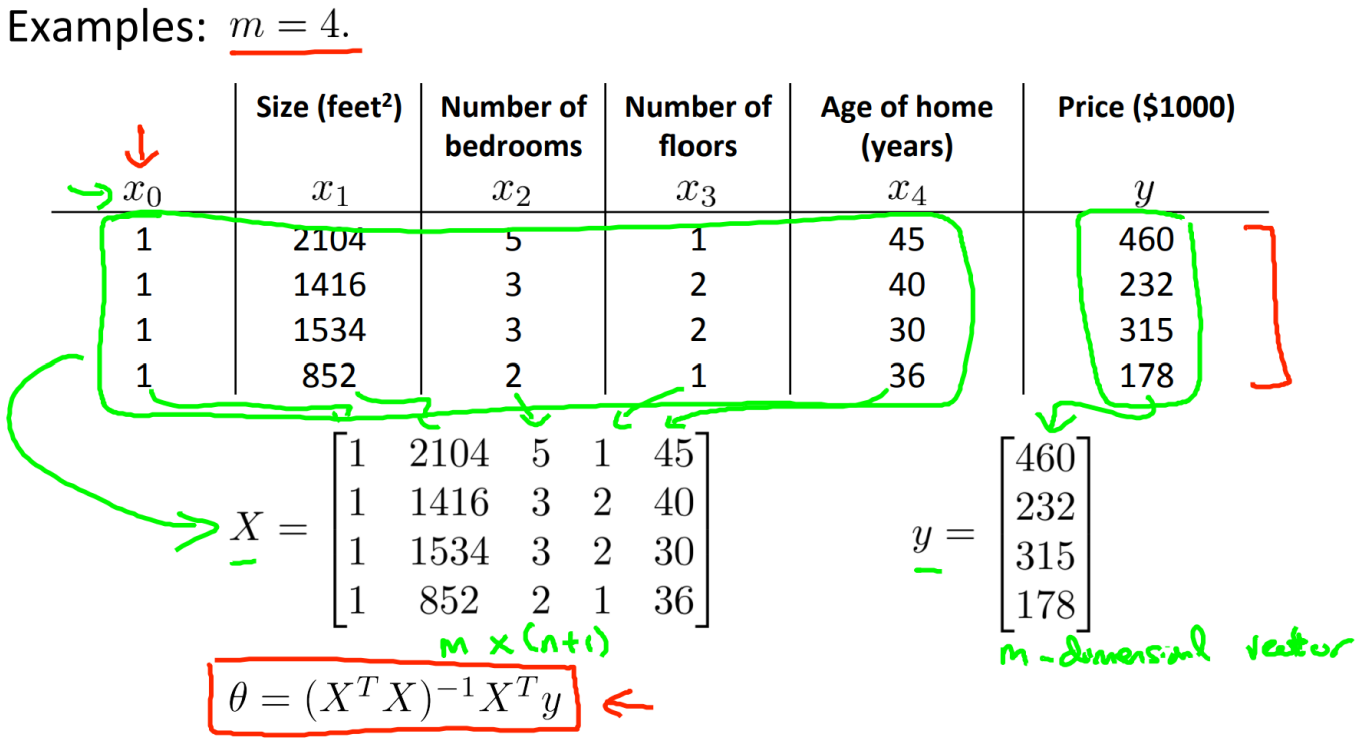
e. normal equation(正规方程)

solve θ analytically



idea: derivation! derivation!

an example:



choice of GD & NE

pros and cons of the two methods

Gradient Descent	Normal Equation
Need to choose alpha	No need to choose alpha
Needs many iterations	No need to iterate
O (kn^2)	O (n^3), need to calculate inverse of X^TX
Works well when n is large	Slow if n is very large

if n is large(probably $n > 10000$), then use GD else use NE

use `pinv()` instead of `inv()` in MATLAB in case of X^TX is non-invertible

W3

1. Classification Introduction

binary-: just$\{0, 1\}$

multi-: $\{0, 1, 2...\}$

using Linear Regression is not a good idea in classification problems

2. Logistic Regression

a. sigmoid function(logistic function)

in classification problems we have:

$h_\theta(x) = g(\theta^Tx)$

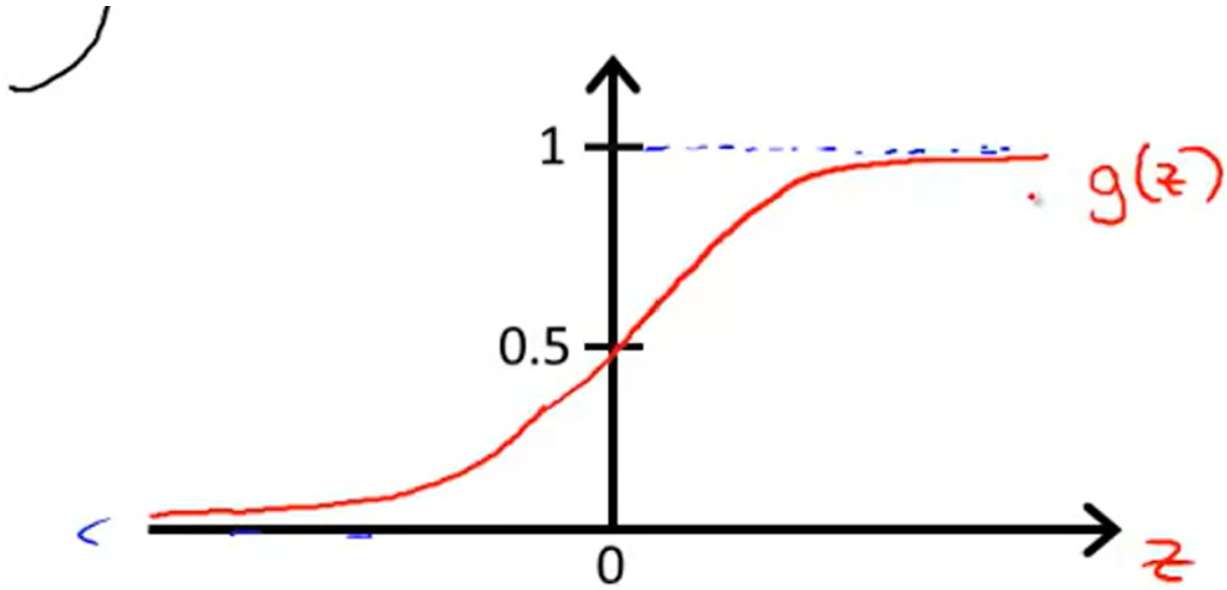
where:

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

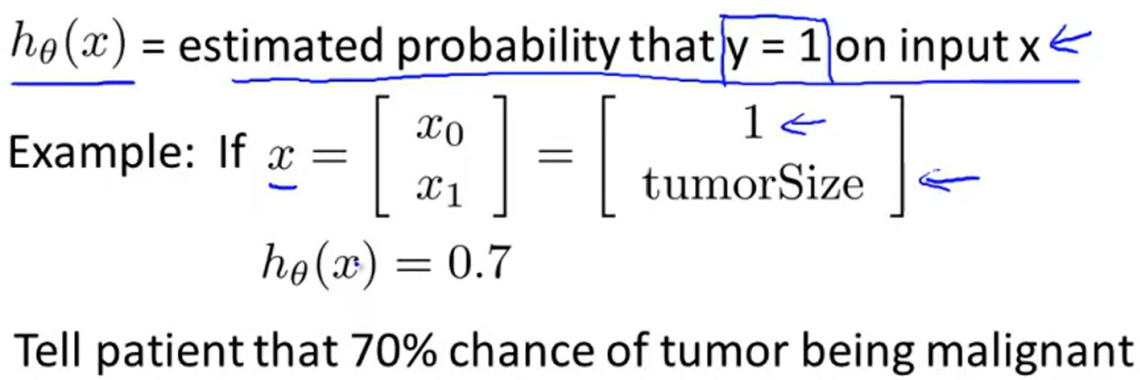
thus, we have

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

the diagram of the sigmoid function:



the output of $h_\theta(x)$ means **probability**, i.e. $P(y=1|x;\theta)$

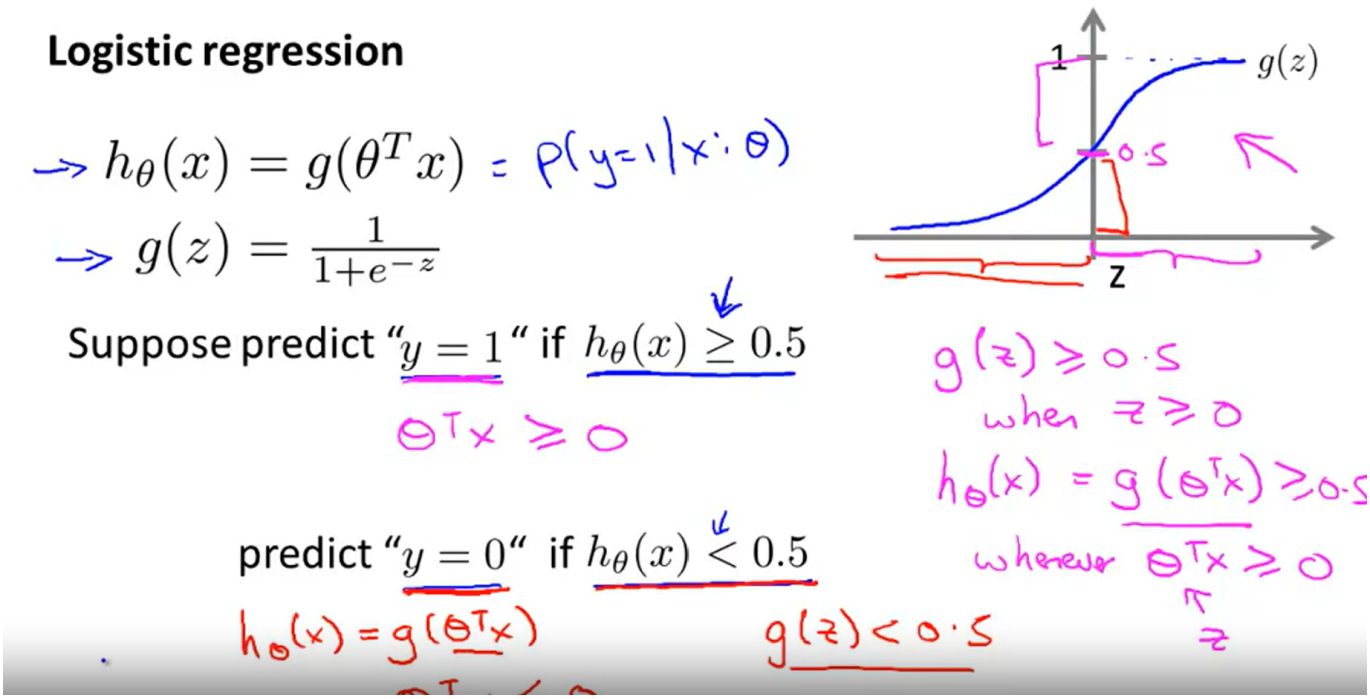


and we have:

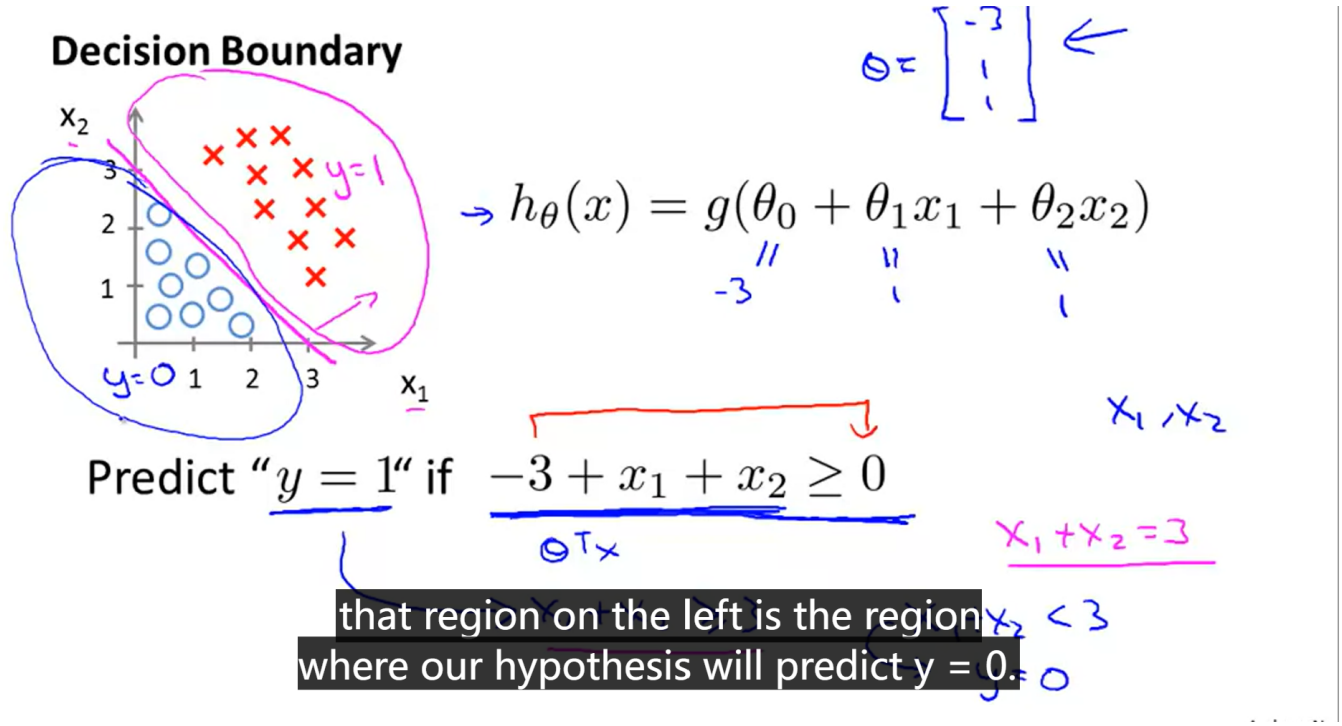
$$h_\theta(x) = P(y=1|x;\theta) = 1 - P(y=0|x;\theta)$$
$$P(y=0|x;\theta) + P(y=1|x;\theta) = 1$$

b. decision boundary(决策边界)

example of logistic regression prediction:

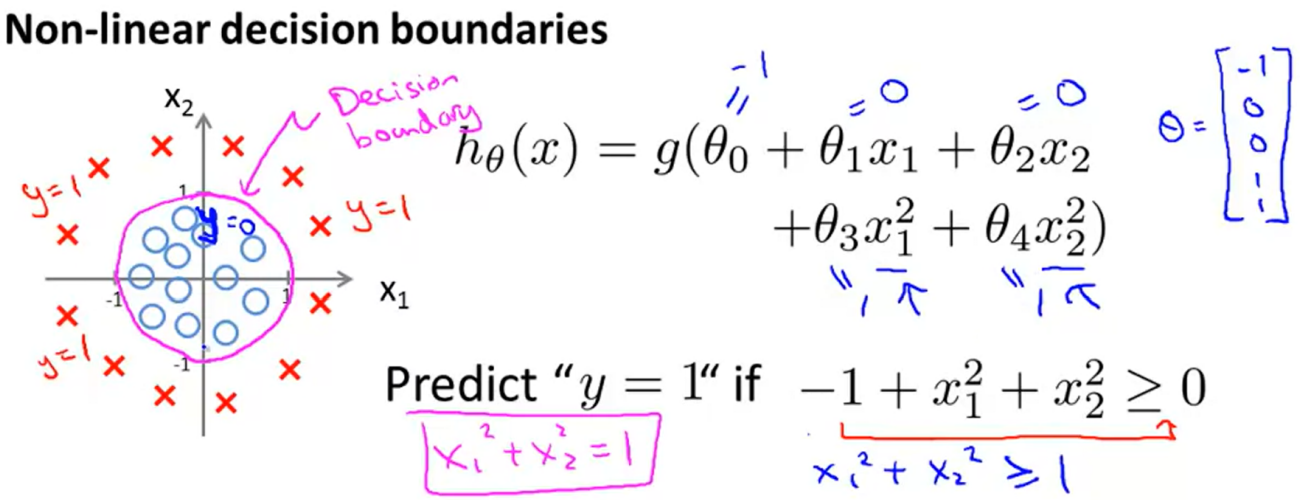


decision boundary example: draw a line(decision boundary) to take different classes apart



that region on the left is the region where our hypothesis will predict y = 0.

non-linear decision boundary:



that I can get in this example.

c. logistic regression cost function & gradient descent

for linear regression we have:

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

rewrite it as:

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x), y)$$

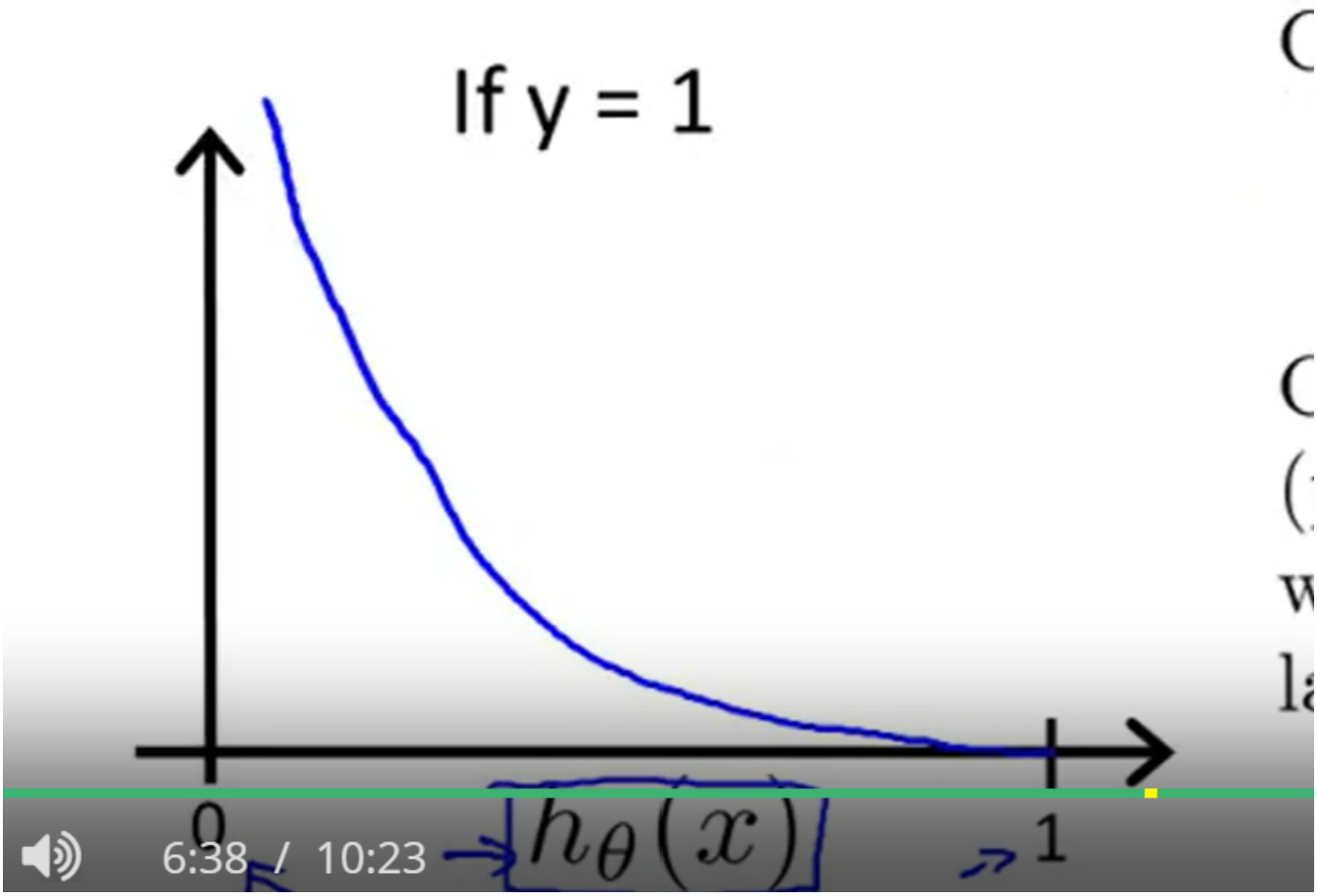
where:

$$\mathrm{Cost}(h_\theta(x), y) = \frac{1}{2}(h_\theta(x^{(i)}) - y^{(i)})^2$$

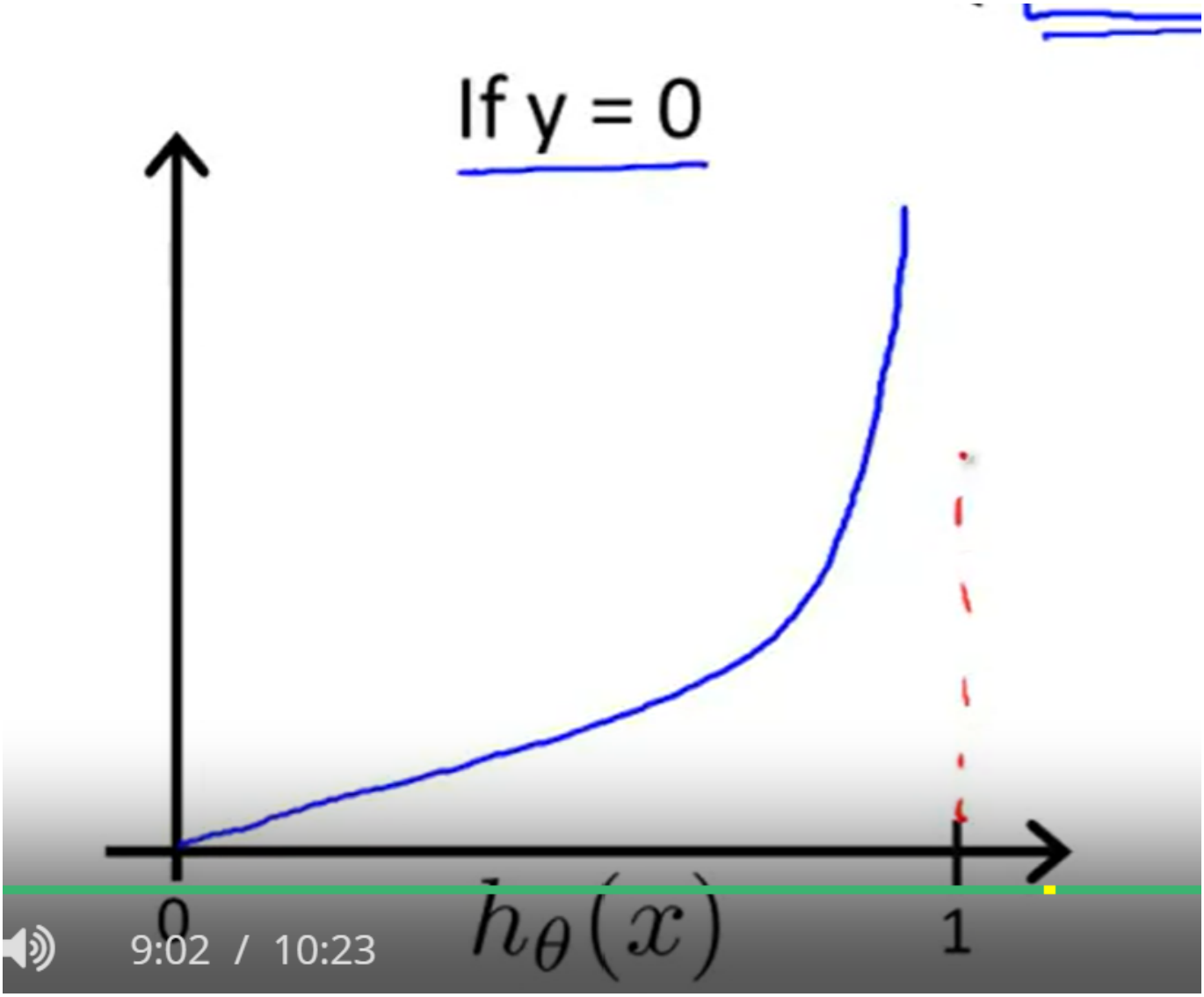
but in logistic regression,we have:

$$\mathrm{Cost}\left(h_\theta(x), y\right) = \begin{cases} -\log\left(h_\theta(x)\right) \text{ if } y = 1 \\ -\log\left(1 - h_\theta(x)\right) \text{ if } y = 0 \end{cases}$$

$\log$ is actually $\ln$, the diagram of the cost function(y==1) is:



the other part of the diagram is:



simplified cost function:

$$\text{Cost}\left(h_\theta(x), y\right) = -y \log\left(h_\theta(x)\right) - (1-y) \log\left(1 - h_\theta(x)\right)$$

as y just has two possible values, it is equivalent to the previous equation.

so we have the final version of the cost function of logistic regression:

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

a vectorized version is:

$$h = g(X\theta)$$
$$J(\theta) = \frac{1}{m} \cdot \left(-y^T \log(h) - (1-y)^T \log(1-h)\right)$$

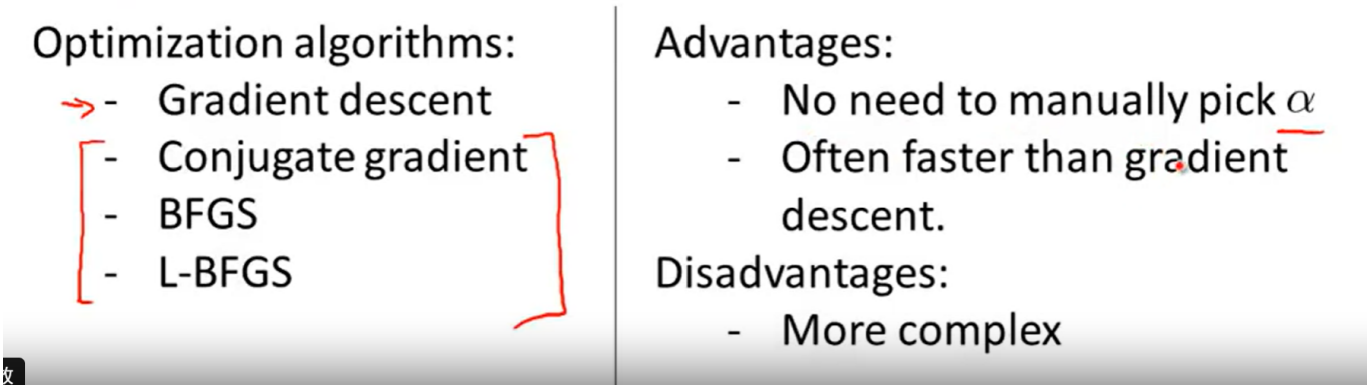
this function is derived from MLE in statistics

and the GD for logistic regression:

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

just like that in linear regression

d. other optimization algorithms



how to use it in MATLAB:

step1: write a function that calculates $J(\theta)$ and $\frac{\partial J}{\partial \theta_j}$

```
function [jVal, gradient] = costFunction(theta)
  jVal = [...code to compute J(theta)...];
  gradient = [...code to compute derivative of J(theta)...];
end
```

step2:

use function `fminunc()` to optimize θ

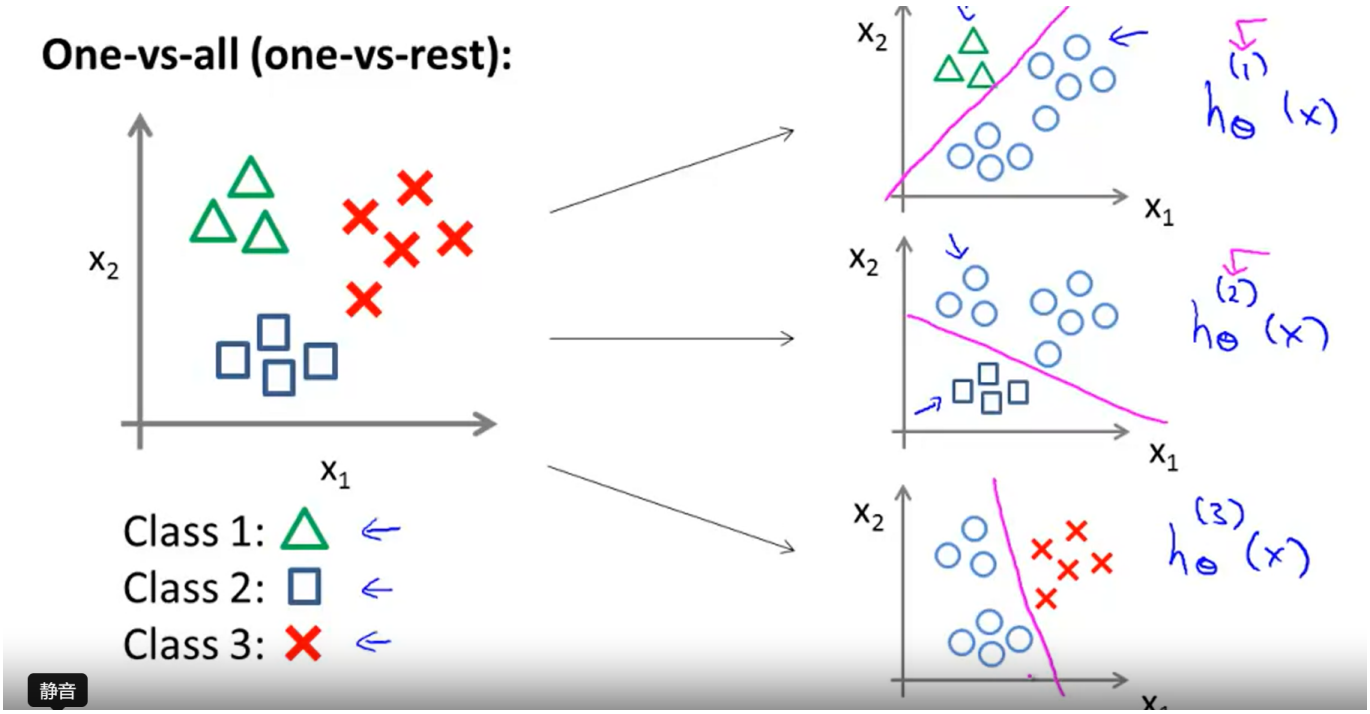
```
options = optimset('GradObj', 'on', 'MaxIter', 100);
initialTheta = zeros(2,1);
   [optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta, options);
```

3. Multi-class Classification

a. one-vs-all classification

idea: separate this problem into different binary-classification problems

example:



i.e. we will train logistic regression classifiers as below:

$$h_\theta^{(i)}(x) = P(y = i|x;\theta)$$

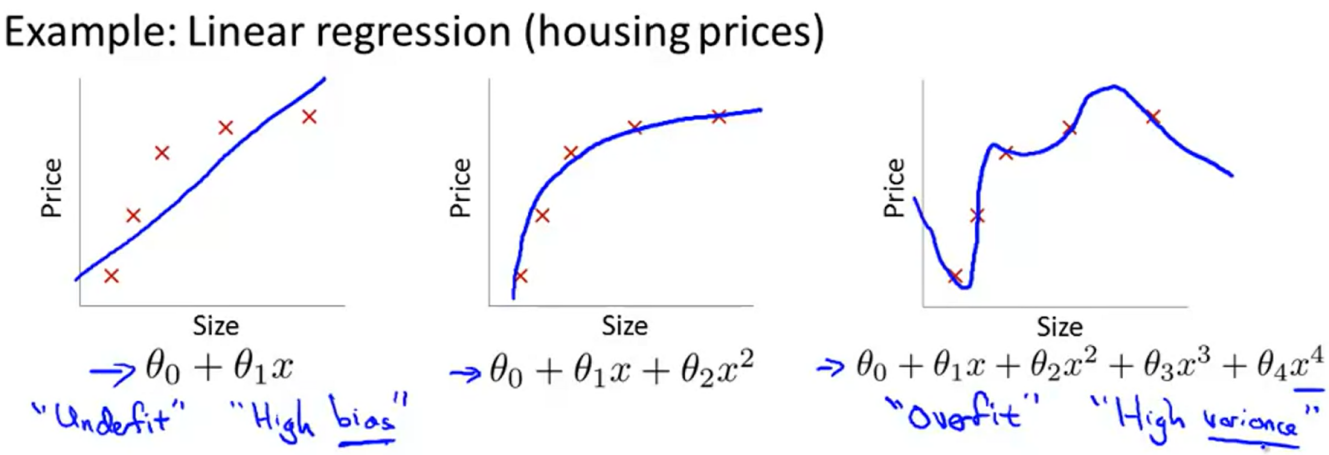
where $i = 1, 2, \dots$

when making predictions, we just have to calculate $\max_i h_\theta^{(i)}(x)$

4. Overfitting

a. introduction

example:



2 ways to solve the problem:

1) Reduce the number of features:

- Manually select which features to keep.
- Use a model selection algorithm (studied later in the course).

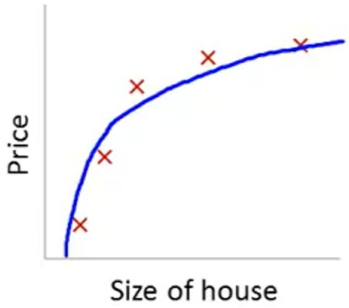
2) Regularization

- Keep all the features, but reduce the magnitude of parameters θ_j.
- Regularization works well when we have a lot of slightly useful features.

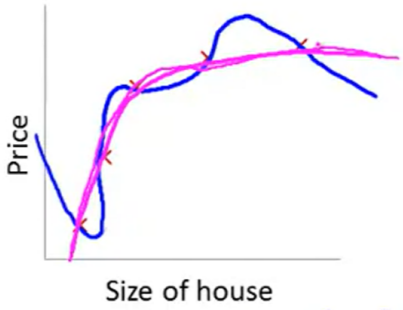
b. regularization

1. intuition&idea:

Intuition



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



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Suppose we penalize and make θ_3, θ_4 really small.

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

Regularization.

Small values for parameters $\theta_0, \theta_1, \ldots, \theta_n$

- "Simpler" hypothesis
- Less prone to overfitting

$\theta_3, \theta_4 \approx 0$

2. cost function

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

where

$$\lambda \sum_{j=1}^{n} \theta_j^2$$

is the regularization term.

λ cannot be too large, otherwise it may cause underfitting

3. regularized linear regression

gradient descent:

Gradient descent

Repeat {

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

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

$$(j = 1, 2, 3, \ldots, n)$$

}

normal equation:

Suppose $m \leq n$,

(#examples) (#features)

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

If $\lambda > 0$,

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

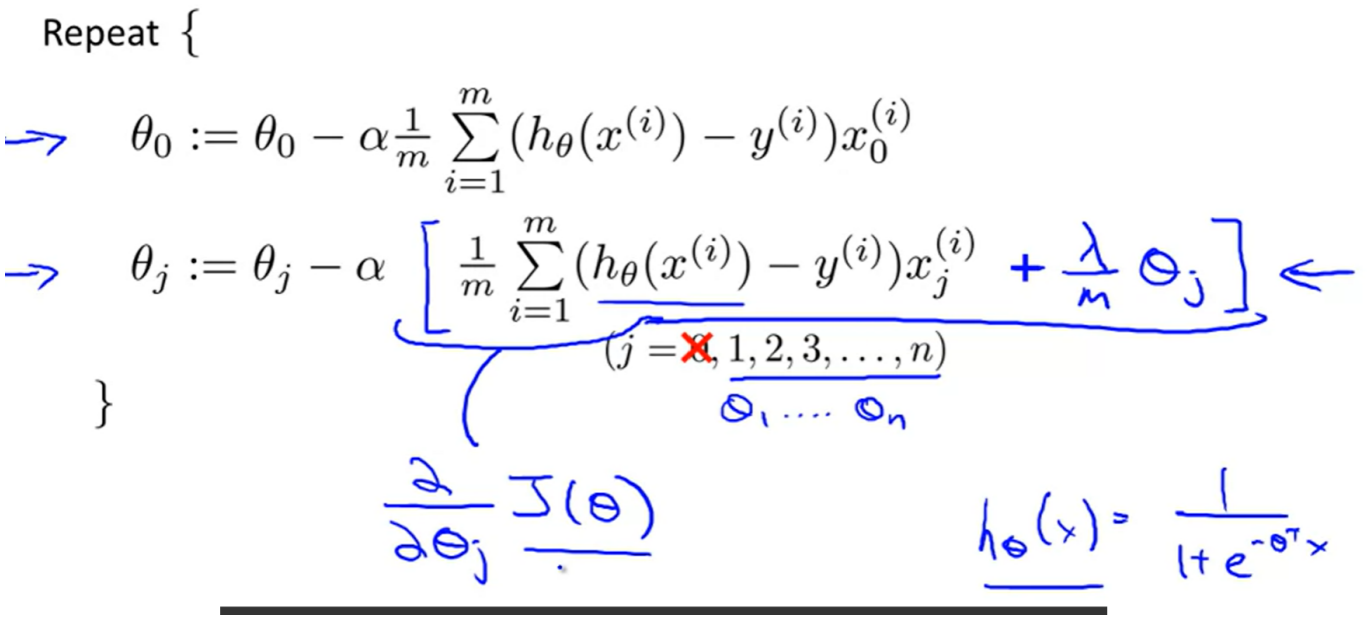
So finally I want to just quickly describe the issue of non-invertibility

note that $X^T X + \lambda \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}$ is always invertable

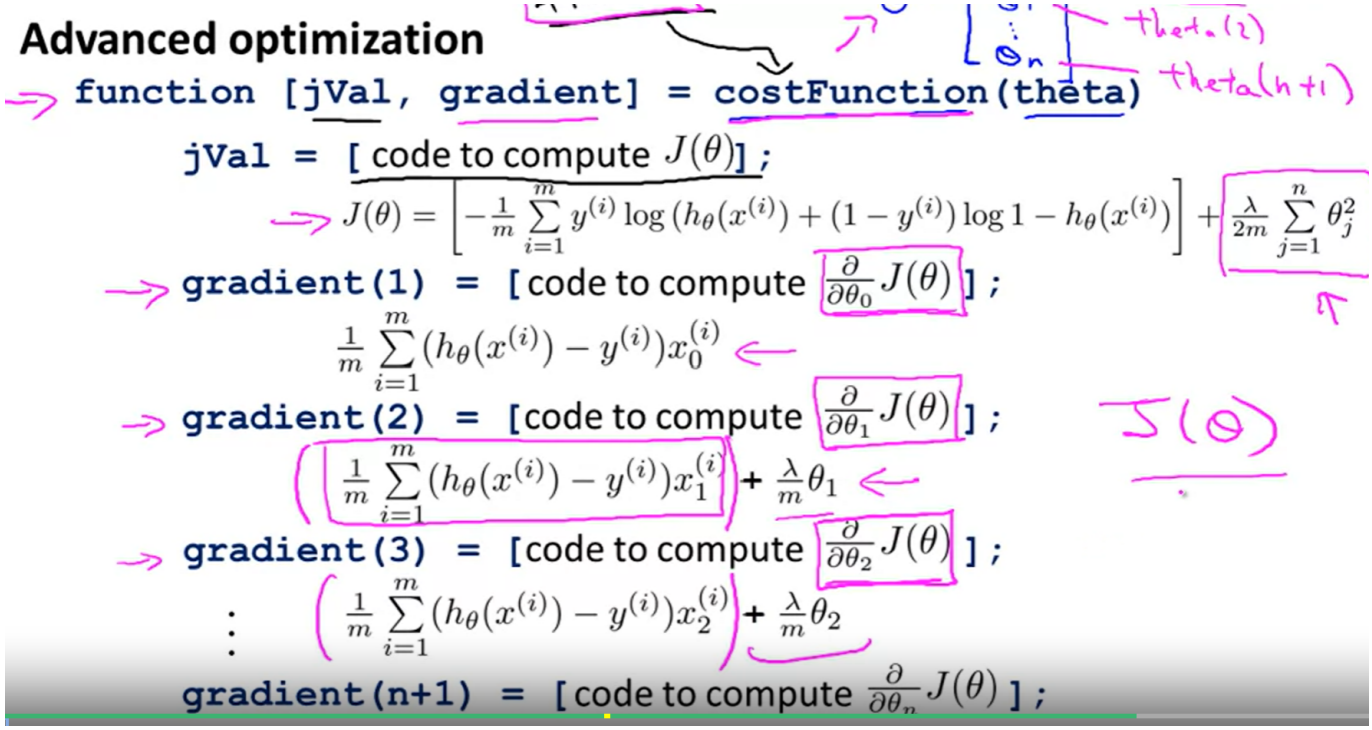
4. regularized logistic regression

gradient descent:

just like that in linear regression, the only difference is the definition of $h_\theta(x)$



advanced algorithms in MATLAB:



just alike.

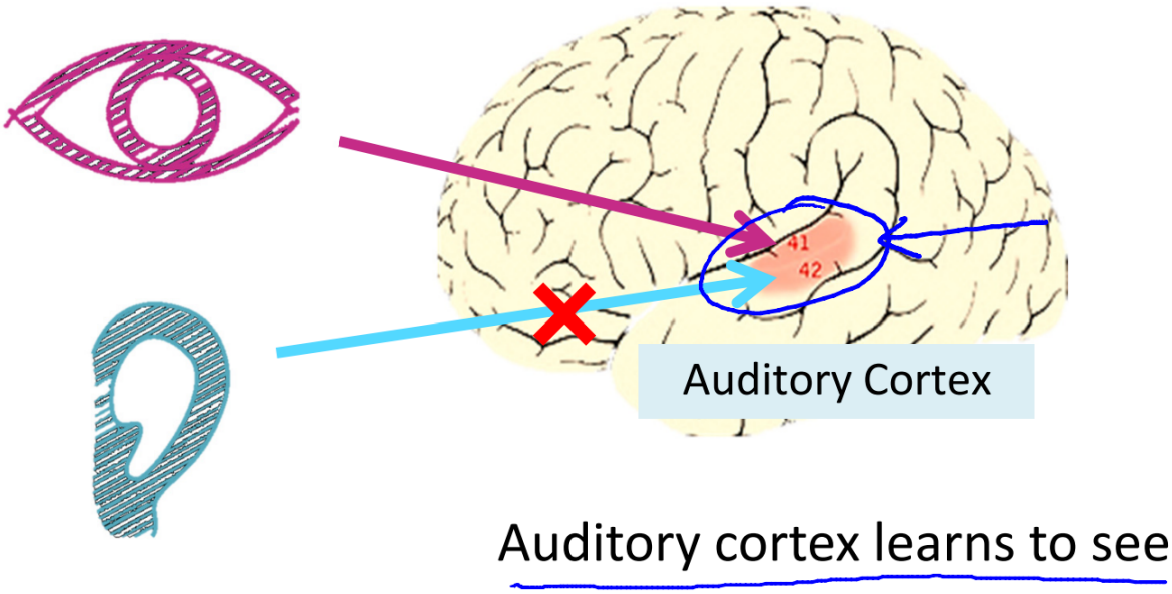
W4

1. Neural Network: Intro & Representation

a. motivation

too many features if we want to train a real classifier like a car detector

brains can actually learn to handle different kinds of signals on the same part once they are connected with different sensors.



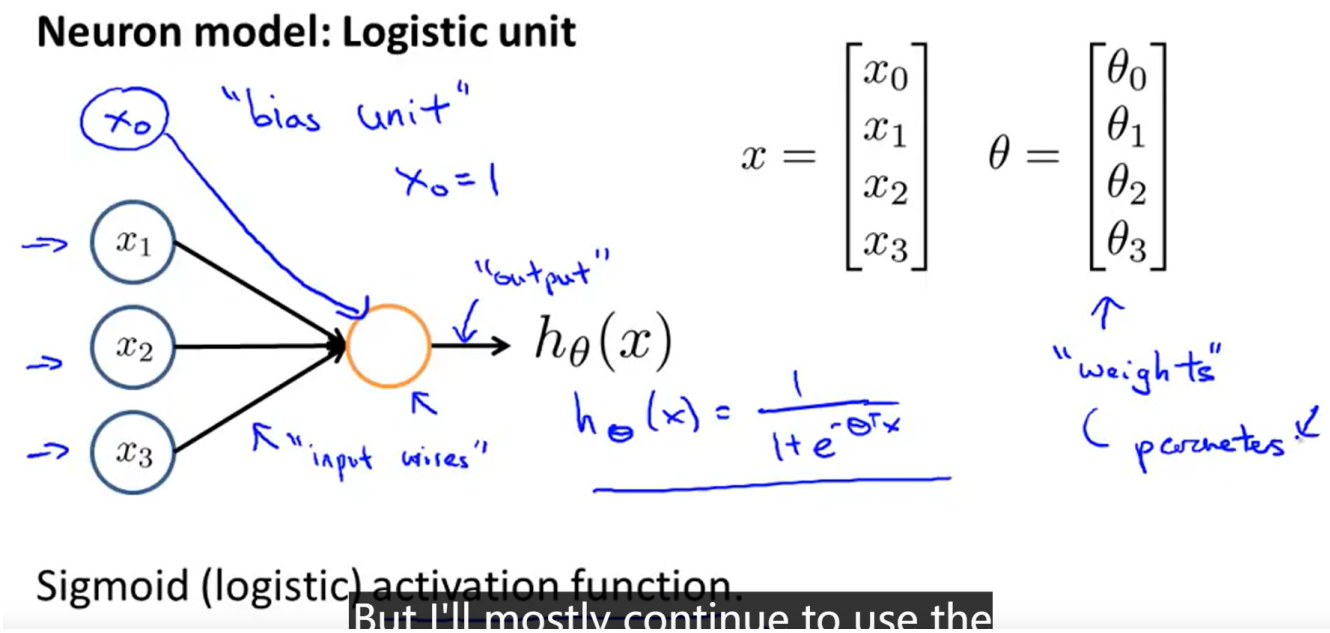
b. model representation

Glossary #3

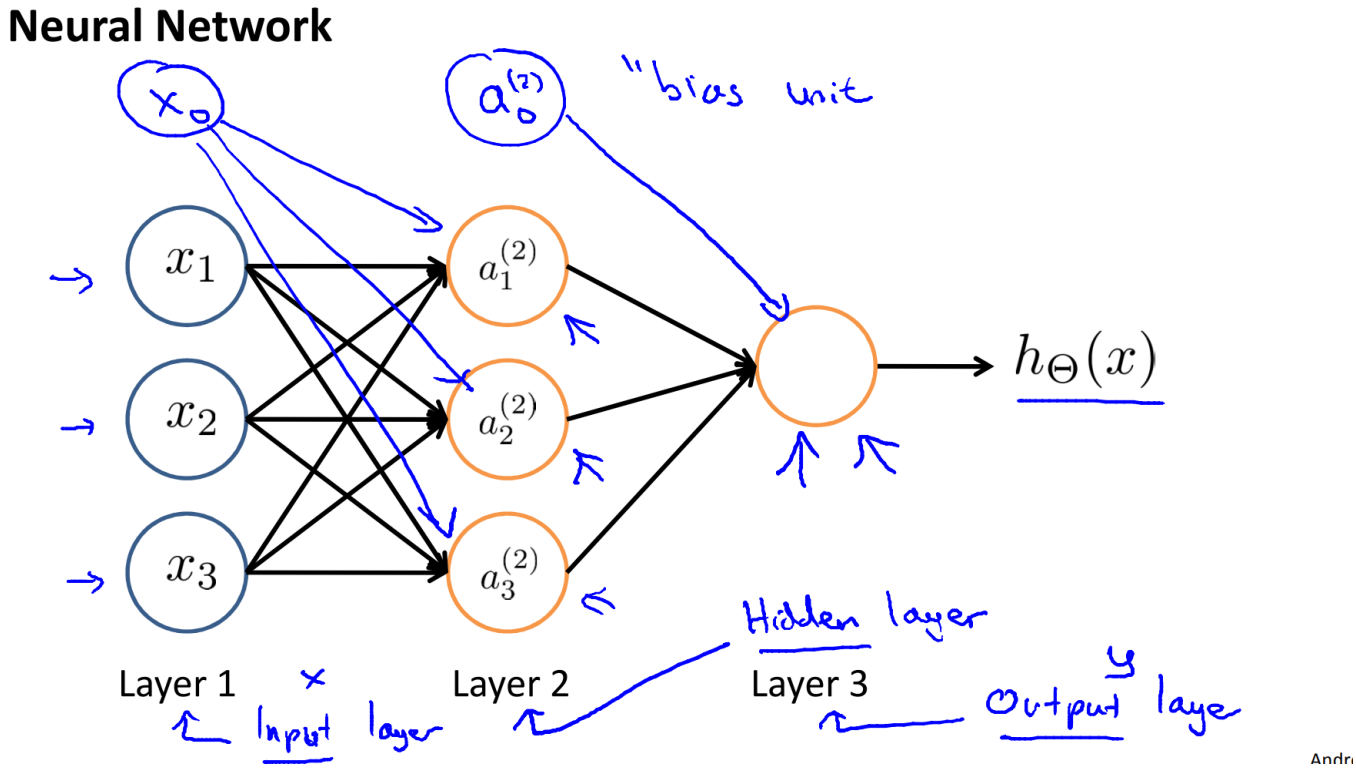
bias unit: input x_0

sigmoid activation function: $g(x) = \frac{1}{1+e^{-x}}$

weights: vector θ

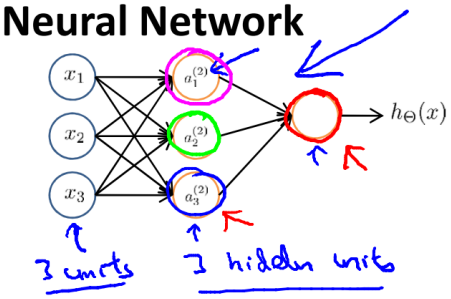


input layer, hidden layer; output layer: as shown in the figure below:



the computational steps:

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 $j+1$

$\Theta^{(1)} \in \mathbb{R}^{3\times 4}$

$h_\Theta(x)$

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

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

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

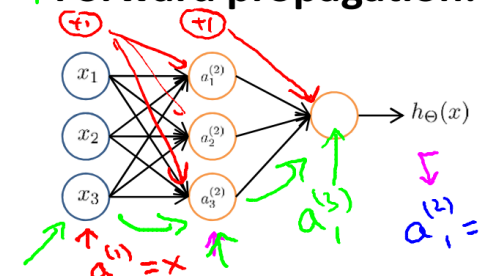
$$\rightarrow h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

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

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

vectorized steps: **forward propagation**

Forward propagation: Vectorized implementation



$$a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$$
$$a_2^{(2)} = g(\Theta_{20}^{(1)}x_0 + \Theta_{21}^{(1)}x_1 + \Theta_{22}^{(1)}x_2 + \Theta_{23}^{(1)}x_3)$$
$$a_3^{(2)} = g(\Theta_{30}^{(1)}x_0 + \Theta_{31}^{(1)}x_1 + \Theta_{32}^{(1)}x_2 + \Theta_{33}^{(1)}x_3)$$
$$h_\Theta(x) = g(\Theta_{10}^{(2)}a_0^{(2)} + \Theta_{11}^{(2)}a_1^{(2)} + \Theta_{12}^{(2)}a_2^{(2)} + \Theta_{13}^{(2)}a_3^{(2)})$$

$a_1^{(2)} = g(z_1^{(2)})$

$z_1^{(2)}$ $z_2^{(2)}$ $z_3^{(2)}$

$z_3^{(2)} = g(z_3^{(2)})$

$z^{(3)}$

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

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

$a^{(2)} \in \mathbb{R}^3$ $a^{(2)} \in \mathbb{R}^3$

Add $a_0^{(2)} = 1.$ → $a^{(2)} \in \mathbb{R}^4$

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

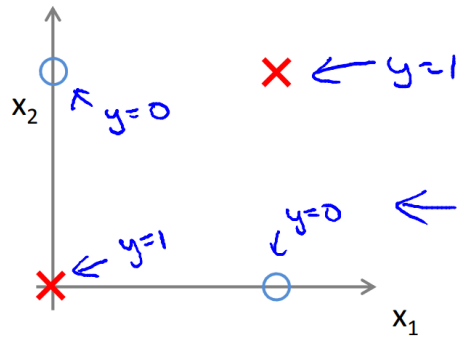
NN just do things like logistic regression except that the features fed to $h_\theta(x)$ are from the hidden layers. **NN learn its own features**

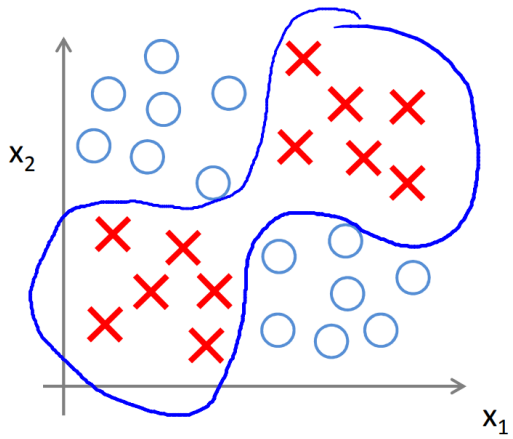
c. intuition & application

intuition: XNOR network

Non-linear classification example: XOR/XNOR

- x_1, x_2 are binary (0 or 1).





$y = x_1 \text{ XOR } x_2$

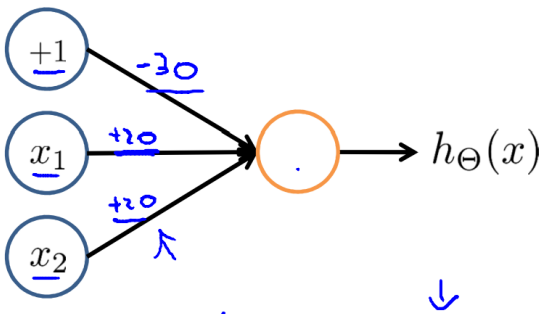
$x_1 \text{ XNOR } x_2$

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

starting from AND

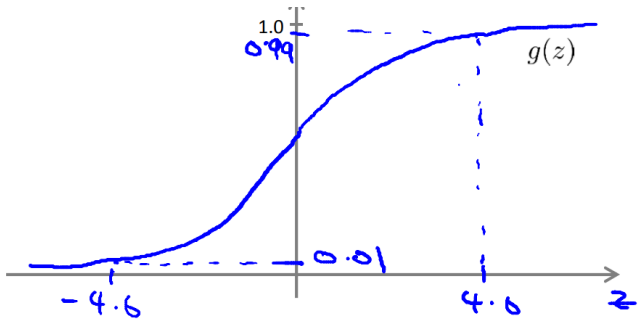
Simple example: AND

- $x_1, x_2 \in \{0, 1\}$
- $y = x_1 \text{ AND } x_2$



$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$

$\Theta^{(1)}_{10}$ $\Theta^{(1)}_{11}$ $\Theta^{(1)}_{12}$

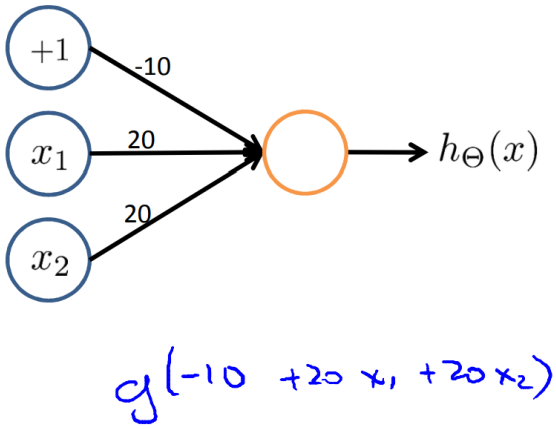


x_1	x_2	$h_\Theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

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

and OR

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	≈ 1
1	1	≈ 1

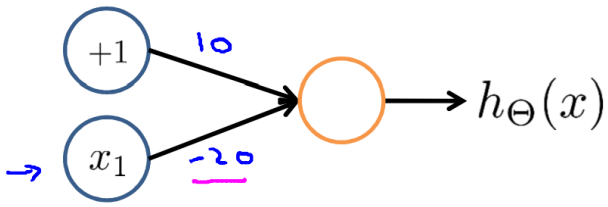
and NOT

$\{0,1\}$.

$\rightarrow x_1$ AND x_2 $\quad\quad$ $\rightarrow x_1$ OR x_2

Negation:

NOT x_1



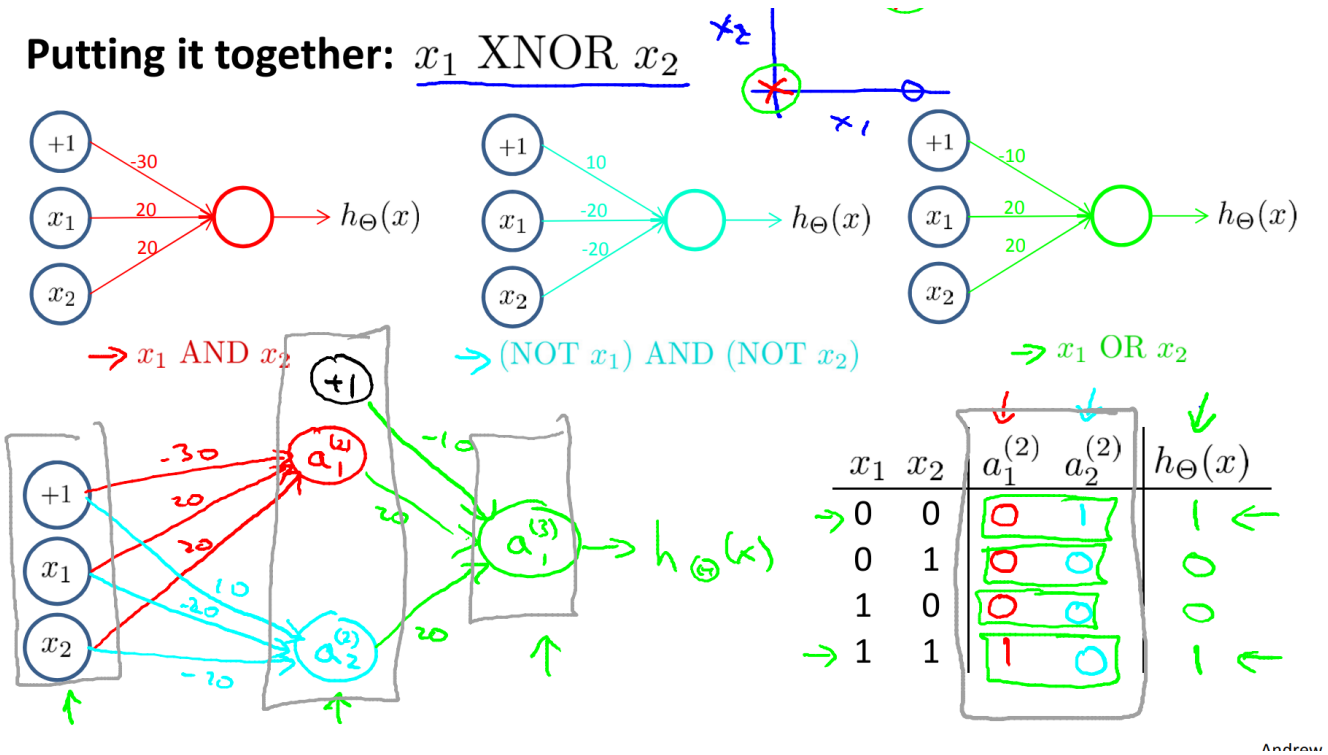
x_1	$h_\Theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

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

$\rightarrow$ (NOT x_1) AND (NOT x_2)

(=1 if and only if $\rightarrow x_1 = x_2 = 0$

finally we can get:



application: handwritten digits classification

W5

1. Neural Network: Learning

a. cost function

Glossary #4

L: the number of layers of a neural network

s_l: the number of units of each layer(without bias unit)

K: the dimension of the output layer

cost function of NN in classification problems:

Neural network:

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

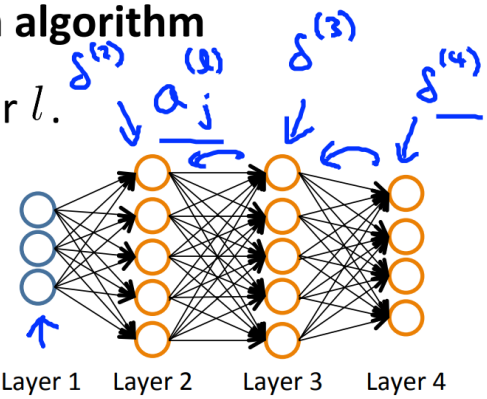
$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)}) \log(1-(h_\Theta(x^{(i)}))_k)\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

b. back propagation: compute the derivatives of $J(\Theta)$

1. compute $\delta^{(j)}$

Gradient computation: Backpropagation algorithm

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



For each output unit (layer L = 4)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

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

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

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

(ignore λ; if $\lambda = 0$)

2. compute derivatives

Backpropagation algorithm

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

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

For $i = 1$ to m

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

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

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

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

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

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

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

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

steps of BP

Given training set $\{(x^{(1)}, y^{(1)}) \cdots (x^{(m)}, y^{(m)})\}$

1. Set $\Delta_{i,j}^{(l)} := 0$ for all (l,i,j), (hence you end up having a matrix full of zeros)

`for` training example t =1 to m:

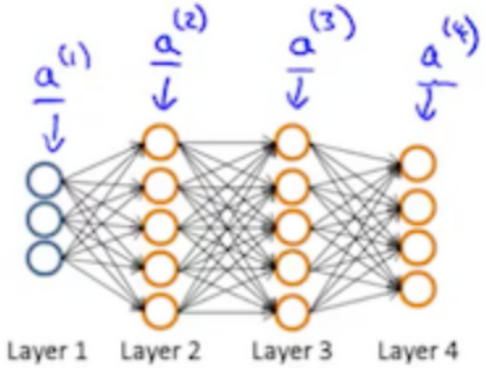
1. Set $a^{(1)} := x^{(t)}$
2. Perform forward propagation to compute $a^{(l)}$ for l=2,3,...,L:

Gradient computation

Given one training example (x, y):

Forward propagation:

$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$$
$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)}) \quad (\text{add } a_0^{(3)})$$
$$z^{(4)} = \Theta^{(3)} a^{(3)}$$
$$a^{(4)} = h_\Theta(x) = g(z^{(4)})$$



3. Using $y^{(t)}$, compute $\delta^{(L)} = a^{(L)} - y^{(t)}$

4. Compute $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$ using $\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) .* a^{(l)} .* (1 - a^{(l)})$

where $a^{(l)} .* (1 - a^{(l)}) = g'(z^{(l)})$ namely the derivative of the sigmoid function

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

end of `for` loop

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

then we get the partial derivative $\frac{\partial}{\partial \Theta_{i,j}^{(l)}} J(\Theta) = D_{i,j}^{(l)}$

c. BP in practice

1. **parameter unrolling from matrices to vectors(系数展开)**

reason: advanced optimizing algorithms treat the inputs as vectors but in NN the inputs are matrices

idea: unroll matrices to vectors

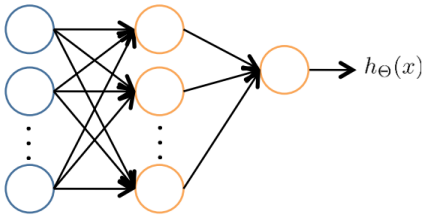
example:

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);
```

usage in advanced learning algorithm

Advanced optimization

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

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

2. **gradient checking**

reason: to debug

idea: check that whether the gradient given by $D^{(l)}$ is similar to the gradient given by its definition(using the formula $\frac{J(\theta+\epsilon)-J(\theta-\epsilon)}{2\epsilon}$)

implementation:

```
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;
```

Check that `gradApprox` ≈ `DVec`

From backprop.

notice that we have to re-initialize `thetaPlus` and `thetaMinus` because we just calculate the partial derivative of each $\theta^{(i)}$

advice on how to use gradient check:

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.

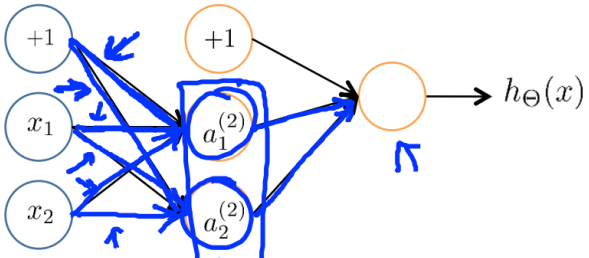
Important:

- 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.

3. **random initialization**

reason:

Zero initialization



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

$a_1^{(2)} = a_2^{(2)}$. Also $\delta_1^{(2)} = \delta_2^{(2)}$.

$\frac{\partial}{\partial \Theta_{01}^{(1)}} J(\Theta) = \frac{\partial}{\partial \Theta_{02}^{(1)}} J(\Theta)$ $\quad \Theta_{01}^{(1)} = \Theta_{02}^{(1)}$

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

$a_1^{(2)} = a_2^{(2)}$

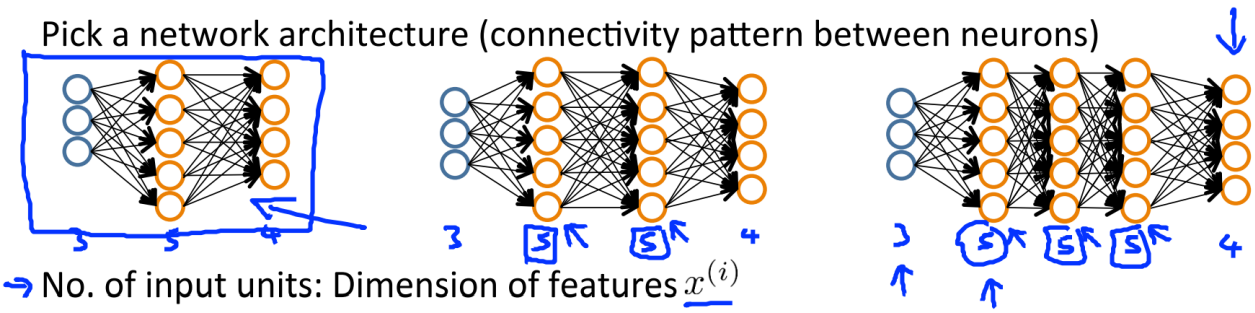
Ar

d. summary: how to train a neural network?

1. pick a network architecture

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)

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

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

2. training through 6 steps

Training a neural network

1. Randomly initialize weights
2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. Implement code to compute cost function $J(\Theta)$
4. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

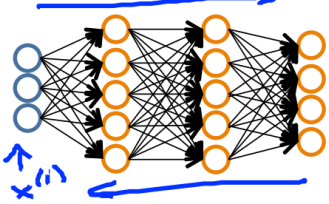
```
for i = 1:m {
```

- Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$
  (Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \ldots, L$).

$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

}

Compute $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$.



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 Θ

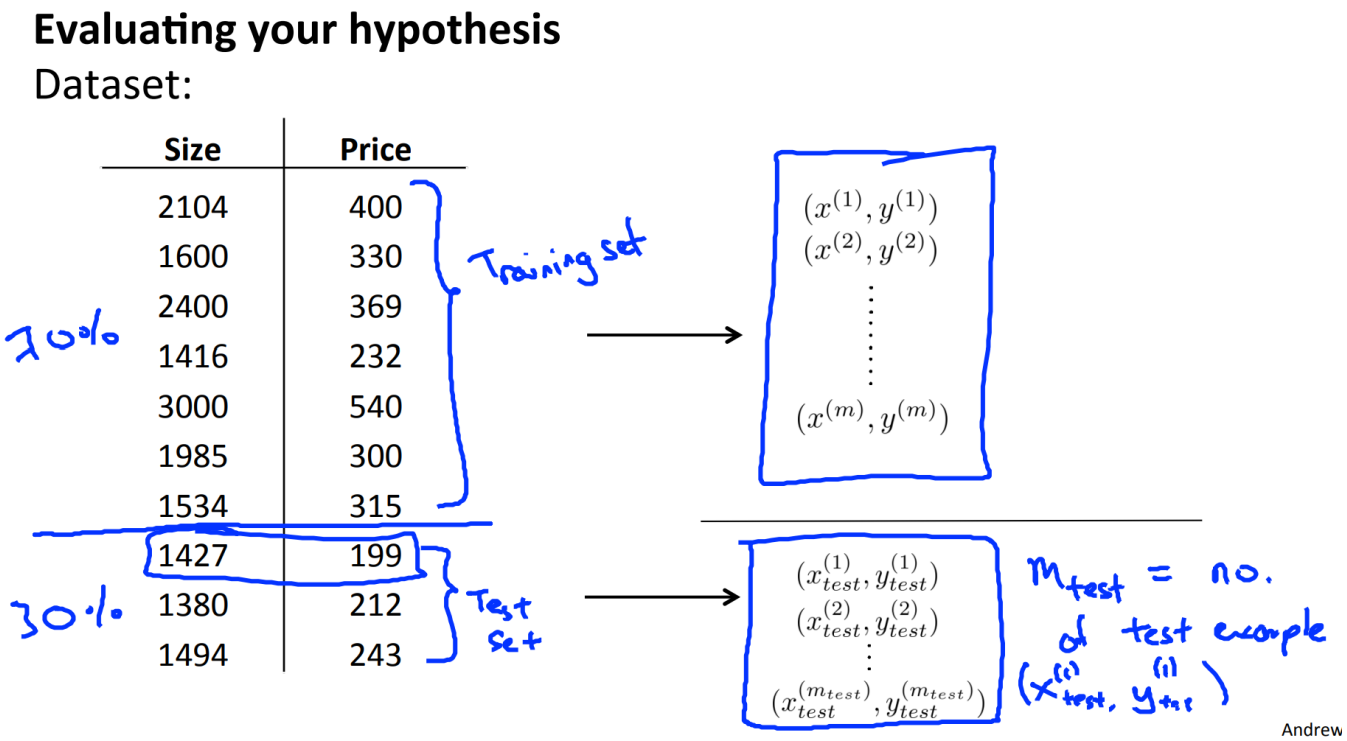
$\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$

$J(\Theta)$ — non-convex.

W6

1. Evaluating a Learning Algorithm

a. test set error



training : test $\approx$ 7 : 3

learn through the training set; compute J_{test} of the test set by the formulas below(varies between linear regression and classification):

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

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

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

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

b. model selection & Train/Validation/Test sets

Glossary #6

d: the degree of the current model

$\theta^{(d)}$: the parameters learnt by the model of degree d

The problem is:

error $J_{test}(\theta^{(5)})$.

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.

and to solve it, we split the dataset into 3 sets: Training/(Cross) Validation/Test sets instead of 2

training : validation : test $\approx$ 6 : 2 : 2

and compute the error of each set

steps of selecting the model using TVT sets

1. learn parameters of each model using the training set
2. compute J_{cv} using the validation set and choose the degree that minimizes J_{cv} and select the model
3. estimate generalized performance using the test set

2. Bias(偏差) & Variance(方差)

a. detection

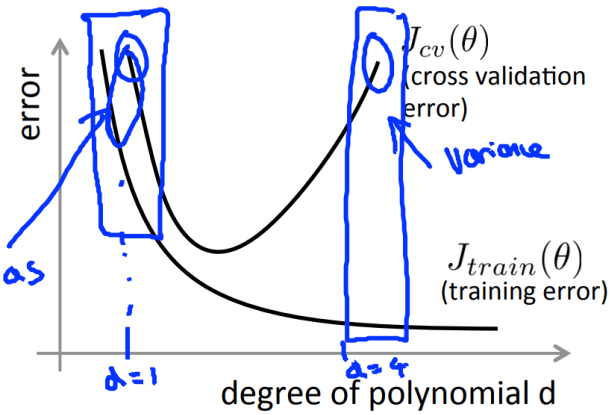
high bias: underfit: $J_{train}(\theta)$

high variance: overfit: $J_{cv}(\theta)$

how to detect:

Diagnosing bias vs. variance

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



Bias (underfit):

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

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

Variance (overfit):

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

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

Andrew N

b. regularization and bias/variance

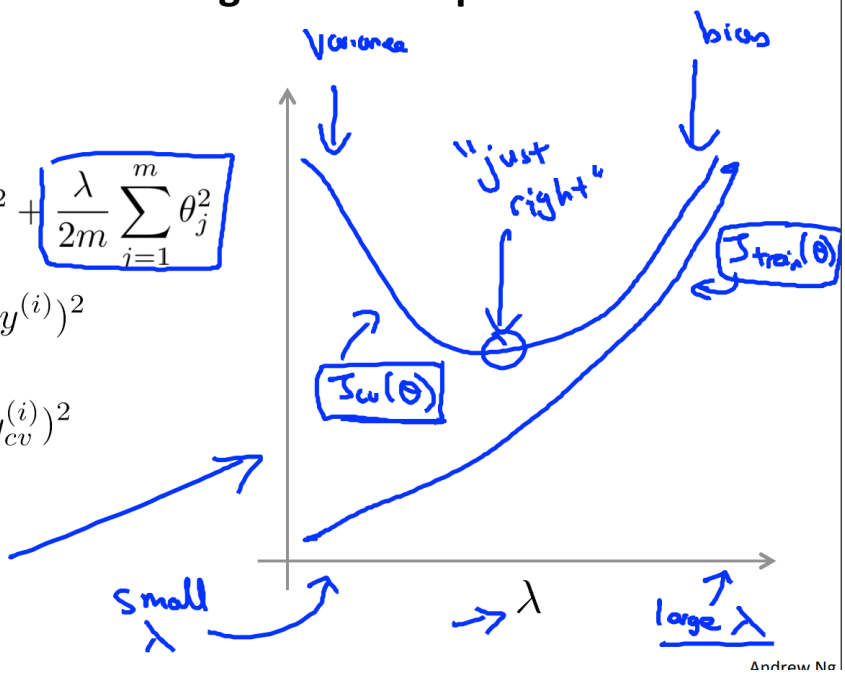
how to choose the regularization parameter λ:

Bias/variance as a function of the regularization parameter λ

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

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

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



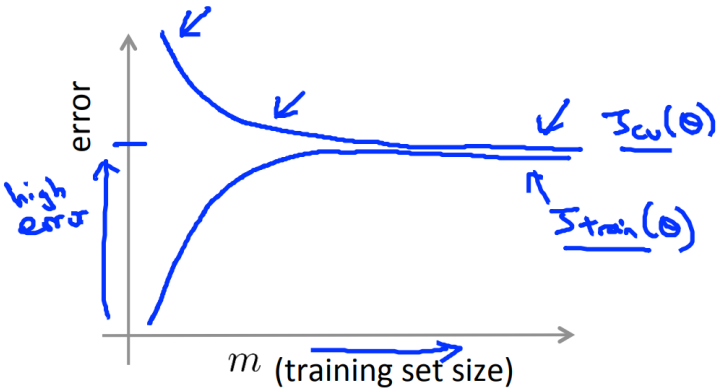
Andrew Ng

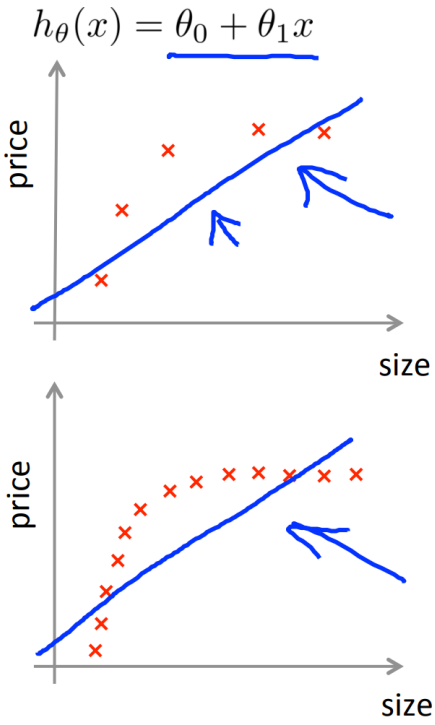
note that all the definition of J_{train} J_{cv} and J_{test} are without the regularization term

c. learning curves

high bias

High bias

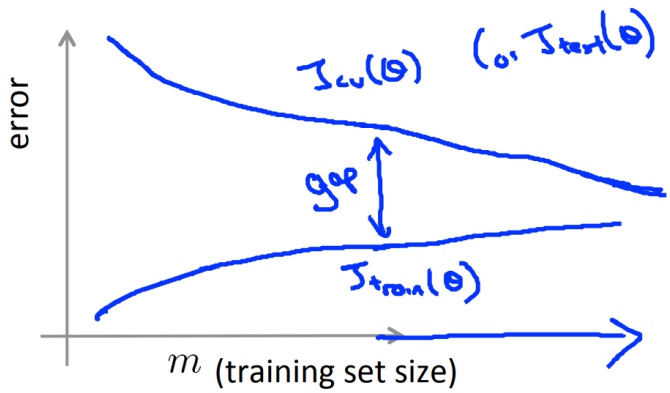


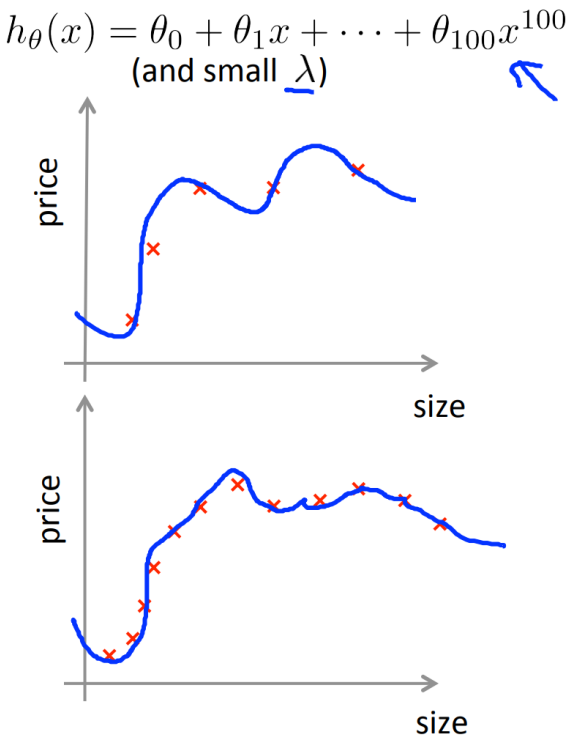


If a learning algorithm is suffering from high bias, getting more training data will not (by itself) help much.

high variance

High variance





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

d. review: what to do next?

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

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

3. ML System Design

a. example: build a spam classifier

prioritizing what to do next

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, discont, andrew, now, ...

$$x = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 1 \\ \vdots \end{bmatrix} \begin{matrix} \text{andrew} \\ \text{buy} \\ \text{deal} \\ \text{discont} \\ \vdots \\ \text{now} \\ \vdots \end{matrix} \qquad x \in \mathbb{R}^{100}$$

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

```
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.)

error analysis

recommended approach when **building a machine learning system**

1. implement a simple algorithm
2. cross validation
3. list possible approaches to improve it

recommended approach when **doing error analysis**

1. find the specific type of the mistaken test cases

e.g.(spam classifier)

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

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

2. list possible cues to improve the performance

doing numerical evaluation is important

It is very important to get error results as a single, numerical value. Otherwise it is difficult to assess your algorithm's performance. I

b. handling skewed data(偏斜数据)

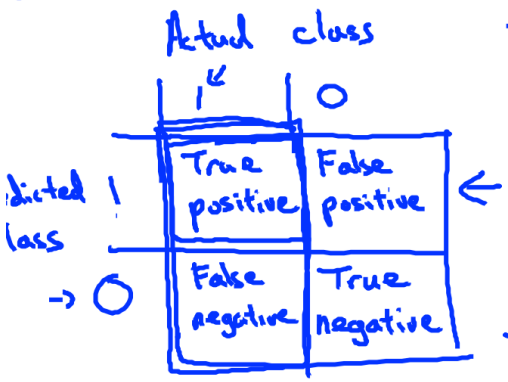
definition

the number of data in one class is much greater than that in another class(e.g. only 0.05% of humanities have cancer) and **causes overfit**

precision(查准率) and recall(召回率)

Precision/Recall

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



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

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

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

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

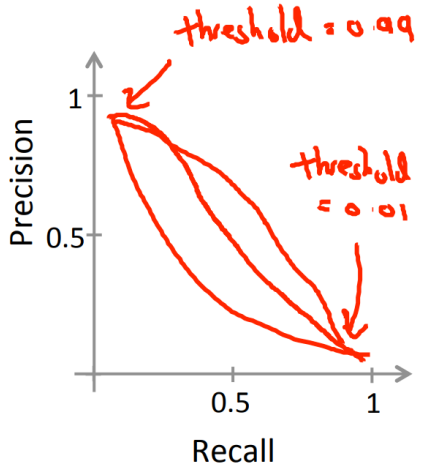
tradeoff between precision and recall

Trading off precision and recall

- Logistic regression: $0 \le h_\theta(x) \le 1$
  Predict 1 if $h_\theta(x) \ge 0.5$ 0.7 0.9 0.3
  Predict 0 if $h_\theta(x) < 0.5$ 0.7 0.9 0.3
- Suppose we want to predict $y = 1$ (cancer) only if very confident.
  → Higher precision, lower recall
- Suppose we want to avoid missing too many cases of cancer (avoid false negatives).
  → Higher recall, lower precision.

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

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



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

Andrew Ng

F score

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

Predict y=1 all the time

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

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

P=0 or R=0 ⇒ F-score = 0

P=1 and R=1 ⇒ F-score = 1

note: choose a proper threshold using **cross validation** set to compute the F score

c. collecting data

rationale

1. sufficient to predict/classify

正在讲话:

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^2) and no other features.

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

2. parameters of the learning algorithm

Large data rationale

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

low bias algorithms.

$J_{train}(\theta)$ will be small.

Use a very large training set (unlikely to overfit)

low variance

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

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

a large scale of data can help in both situations

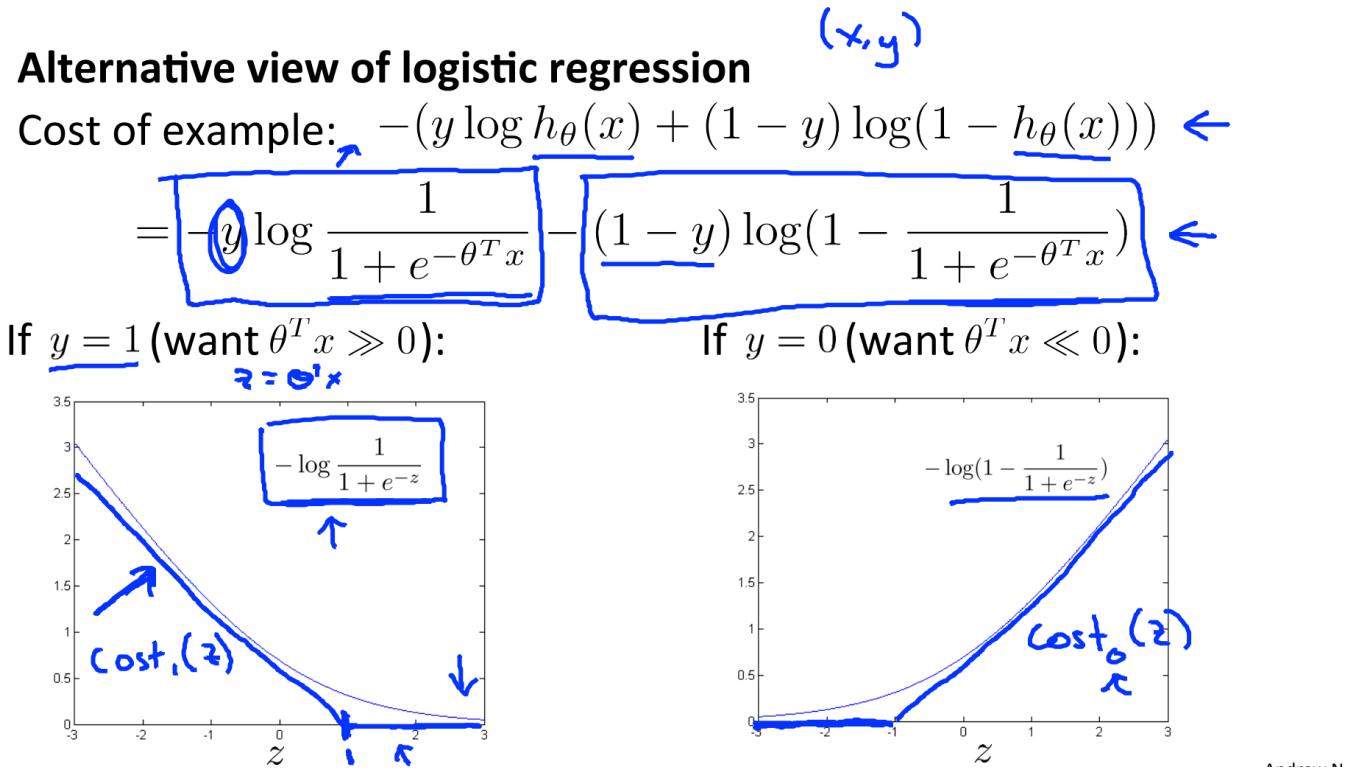
W7

1. SVM(Support Vector Machine)

a. large margin classification(大间距分类)

an alternative viewpoint of logistic regression

a new definition for the cost function of logistic regression $\text{cost}_1(z)$ (y=1) and $\text{cost}_0(z)$(y=0) as the blue line shows :



definition of SVM:

$$\min_\theta C \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

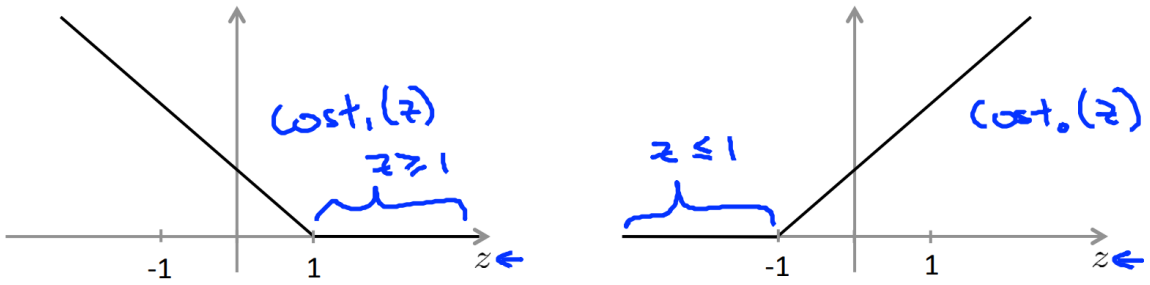
m is removed from the cost function and $C = \frac{1}{\lambda}$, and $h_\theta(x)$ is replaced by $\text{cost}_{0,1}(\theta^T x^{(i)})$

intuition

We want $\theta^T x \geq 1$ instead of 0 if $y = 1$ otherwise we want $\theta^T x \leq -1$ instead of 0 because we want to minimize the cost function by setting the first sum 0:

Support Vector Machine

$$\min_\theta C\sum_{i=1}^{m}\left[y^{(i)}cost_1(\theta^T x^{(i)}) + (1-y^{(i)})cost_0(\theta^T x^{(i)})\right] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$$



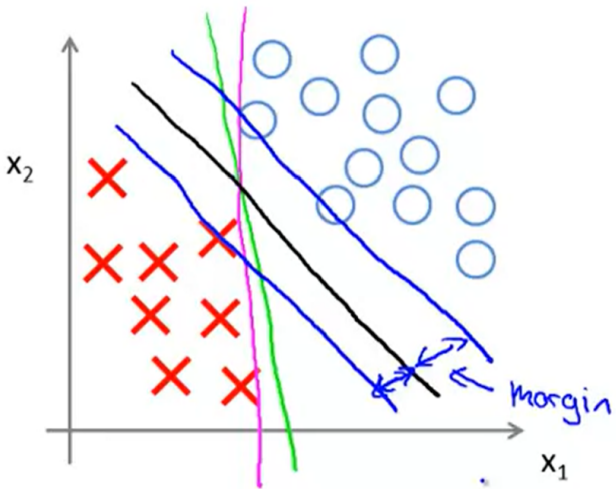
If $y = 1$, we want $\theta^T x \geq 1$ (not just ≥ 0)

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

C = 100,000

SVM choose the best decision boundary with the largest **margin**(the distance between the boundary and the nearest (to the boundary) points of each class):

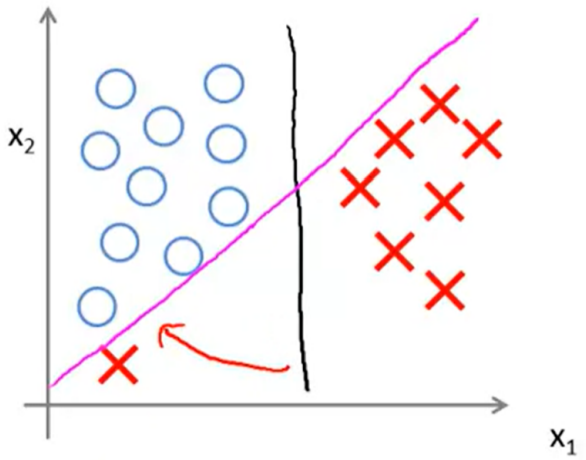
SVM Decision Boundary: Linearly separable case



blue lines are margins and SVM will choose the black one

but in datasets with outliers like this:

Large margin classifier in presence of outliers



if C is very large the SVM will choose the magenta line as the boundary; otherwise it will remain choosing the black one.

math behind LMC

the SVM tries to solve:

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

$$\text{s.t.} \quad \begin{cases} p^{(i)} \cdot \|\theta\| \geq 1 & \text{if } y^{(i)} = 1 \\ p^{(i)} \cdot \|\theta\| \leq -1 & \text{if } y^{(i)} = 1 \end{cases}$$

where $p^{(i)}$ is the projection of $x^{(i)}$ on

i.e. (simplified version)

SVM Decision Boundary

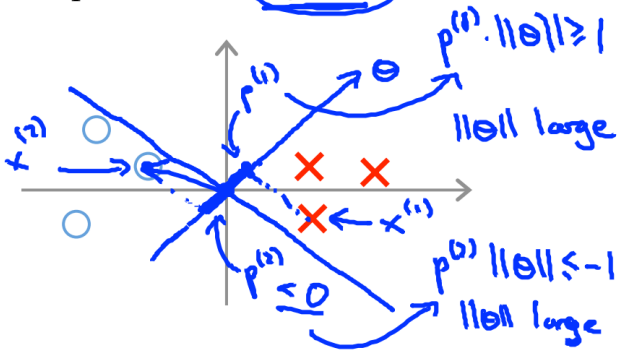
$$\min_{\theta} \frac{1}{2}\sum_{j=1}^{n}\theta_j^2 = \frac{1}{2}\|\theta\|^2$$

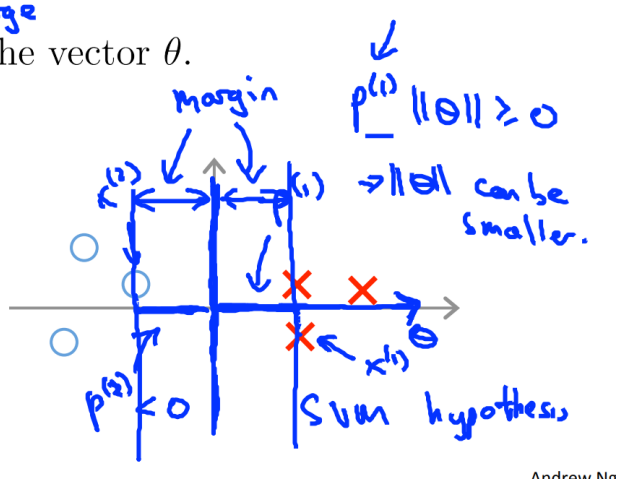
$$\text{s.t.} \quad \begin{cases} p^{(i)} \cdot \|\theta\| \geq 1 & \text{if } y^{(i)} = 1 \\ p^{(i)} \cdot \|\theta\| \leq -1 & \text{if } y^{(i)} = 1 \end{cases}$$

C very large

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

Simplification: $\theta_0 = 0$





Andrew Ng

the reason why SVM doesn't choose the boundary on the left(seems less 'natural' than that on the left) is: $|p^{(i)}|$ is small so that $\|\theta\|$ shall be **large** but that is **contradictory to the first minimizing condition**

here is why θ is vertical to the boundary:

http://cs229.stanford.edu/materials/smo.pdf

Q3) How do we know that the theta is perpendicular to the decision boundary?

(thanks to Mentor Chirag for this derivation)

We know that $\theta' * x = 0$ for any x on the boundary since the boundary is where sigmoid = 0.5.

$$\frac{1}{1+e^{-z}} = \frac{1}{2} => e^{-z} = 1 => -z = 0 => \theta' * x = 0$$

So pick two random points on the boundary a and b, then

$\theta' * a = 0$ and $\theta' * b = 0$

$=> \theta' * (a - b) = 0 => \theta \cdot (a - b) = 0$

and we know that when the dot product of two vectors is 0, the angle between them is 90 degrees. And since the vector (a-b) is on the decision boundary, θ is perpendicular to the decision boundary.

b. kernels(adapting SVM to non-linear classification problems)

Gaussian kernels

Kernels and Similarity

$$f_1 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

If $x \approx l^{(1)}$:

$$f_1 \approx \exp\left(-\frac{0^2}{2\sigma^2}\right) \approx 1$$

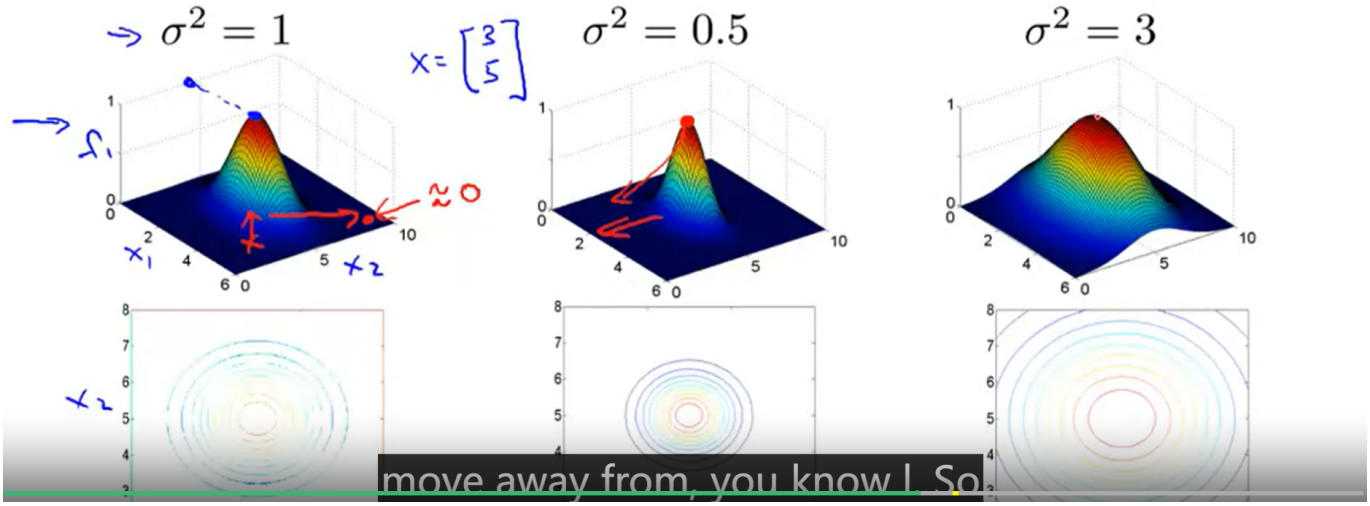
If x if far from $l^{(1)}$:

$$f_1 = \exp\left(-\frac{(\text{large number})^2}{2\sigma^2}\right) \approx 0.$$

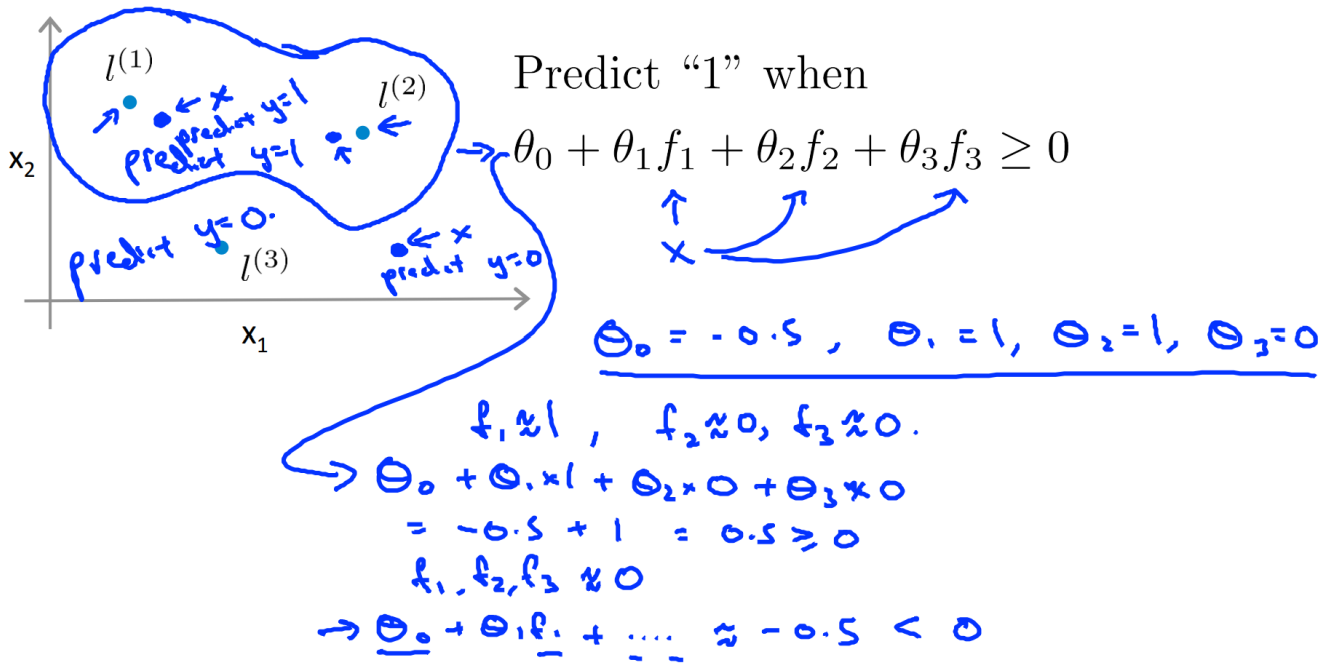
example

Example:

$$l^{(1)} = \begin{bmatrix} 3 \\ 5 \end{bmatrix}, \quad f_1 = \exp\left(-\frac{\|x - l^{(1)}\|^2}{2\sigma^2}\right)$$

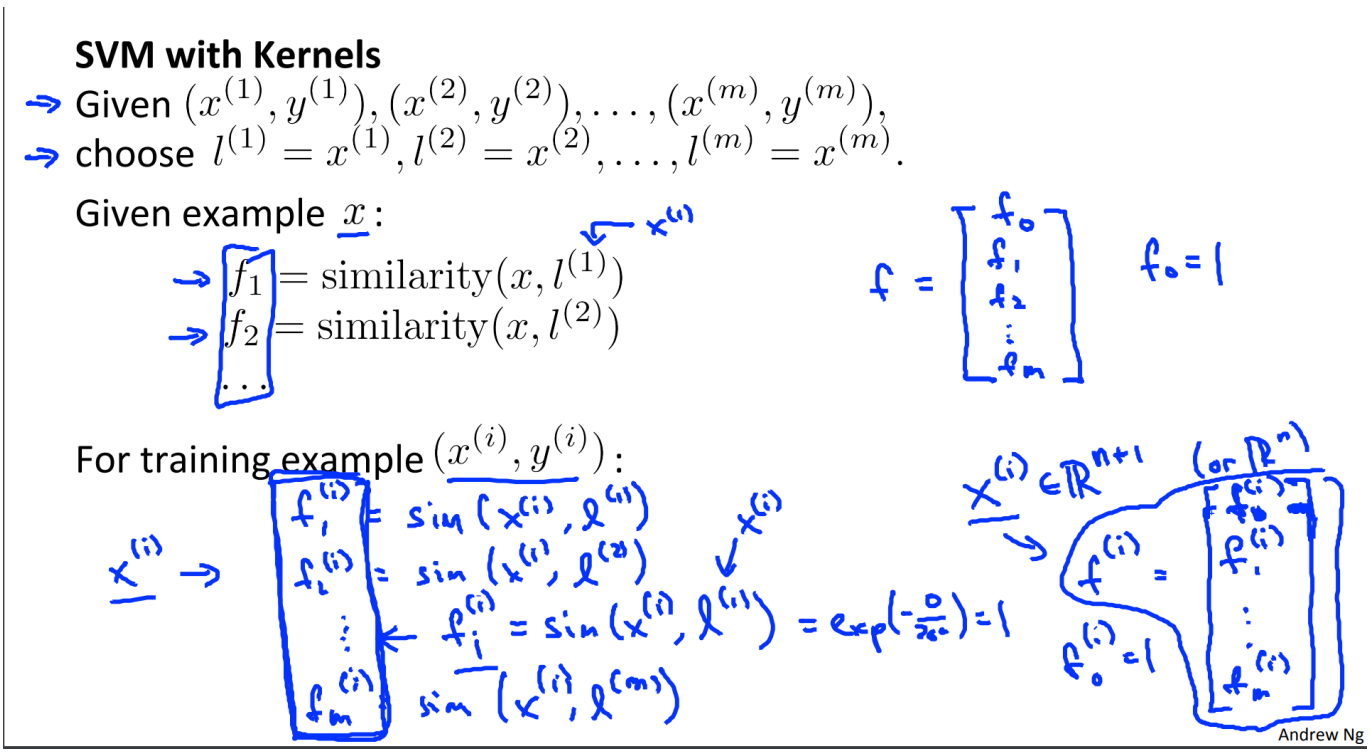


how to predict



choose the landmarks

just choose $l^{(i)} = x^{(i)}$



combining SVM with kernels

just modify $\theta^T x^{(i)}$ to $\theta^T f^{(i)}$ like this:

SVM with Kernels

Hypothesis: Given x, compute features $f \in \mathbb{R}^{m+1}$

→ Predict "y=1" if $\theta^T f \geq 0$

Training:

$$\min_\theta C \sum_{i=1}^{m} y^{(i)} cost_1(\theta^T f^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T f^{(i)}) + \frac{1}{2} \sum_{j=1}^{m} \theta_j^2$$

Andrew N

parameters

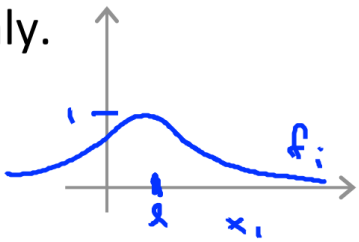
SVM parameters:

C ($= \frac{1}{\lambda}$). → Large C: Lower bias, high variance.

→ Small C: Higher bias, low variance.

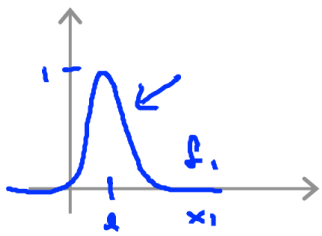
σ^2 Large σ^2: Features f_i vary more smoothly.

→ Higher bias, lower variance.



Small σ^2: Features f_i vary less smoothly.

Lower bias, higher variance.



c. SVM in practice

in programming we don't need to write the SVM functions ourselves but we have to specify two things below:

- the choice of parameter C

- the choice of the kernel(similarity function)
- (when using Gaussian kernel) implement the kernel function

choice of the kernel

1. no kernel(i.e. linear kernel): gives a standard linear classifier i.e. predict $y = 1$ if $\theta^T x \geq 0$. one situation to use it is there are many features but few training data
2. Gaussian kernel: few features and many training data, **do perform feature scaling**!
3. other choices:

Other choices of kernel

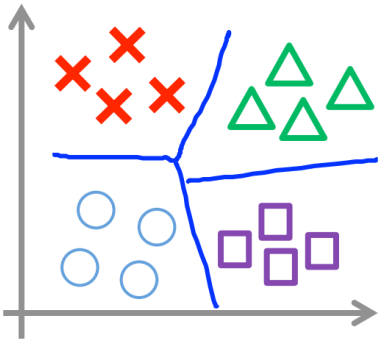
Note: Not all similarity functions $\text{similarity}(x, l)$ make valid kernels.
⇒ (Need to satisfy technical condition called "Mercer's Theorem" to make sure SVM packages' optimizations run correctly, and do not diverge).

Many off-the-shelf kernels available:

- Polynomial kernel: $k(x, l) = (x^T l)^3$, $(x^T l)^2$, $(x^T l + 1)^3$, $(x^T l + 5)^4$, $(x^T l + \text{constant})^{\text{degree}}$
- More esoteric: String kernel, chi-square kernel, histogram intersection kernel, ... $\text{sim}(x, l)$

multi-classification using SVM

Multi-class classification



$$y \in \{1, 2, 3, \ldots, K\}$$

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 $i = 1, 2, \ldots, K$), get $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$
Pick class i with largest $(\theta^{(i)})^T x$

logistic regression vs. SVM

n is the number of features; m is the number of training examples

- n is large relative to m, use logistic regression or SVM without kernels
- n small while m intermediate: SVM with Gaussian kernel
- n small while m large: add more features, use logistic regression or SVM without kernels

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):

→ Use logistic regression, or SVM without a kernel ("linear kernel")

→ If n is small, m is intermediate:

→ Use SVM with Gaussian kernel

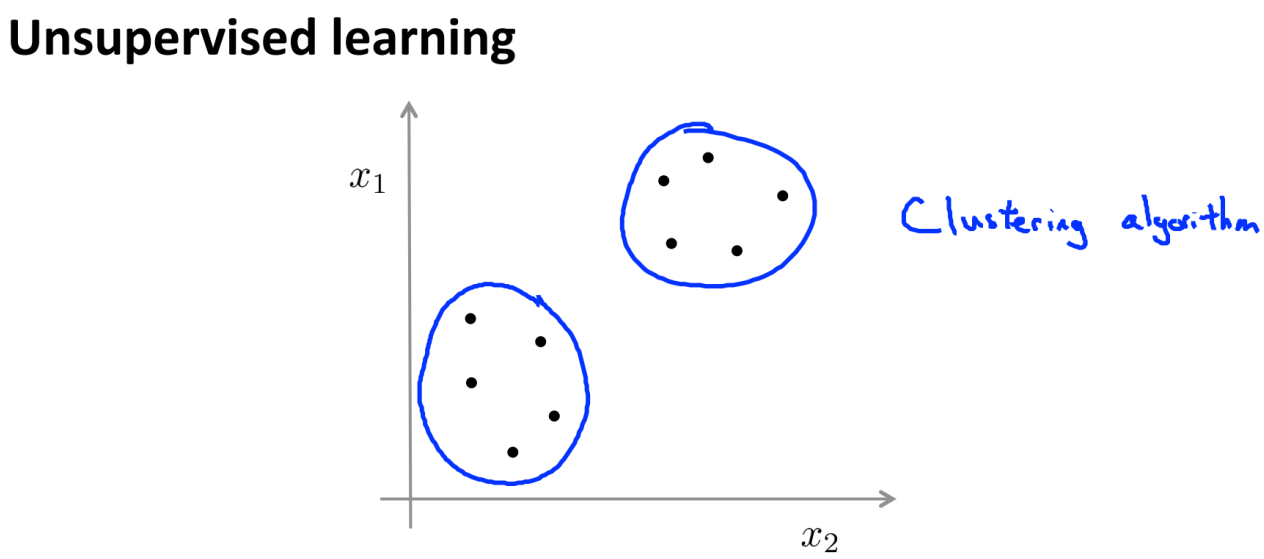
If n is small, m is large:

→ Create/add more features, then use logistic regression or SVM without a kernel

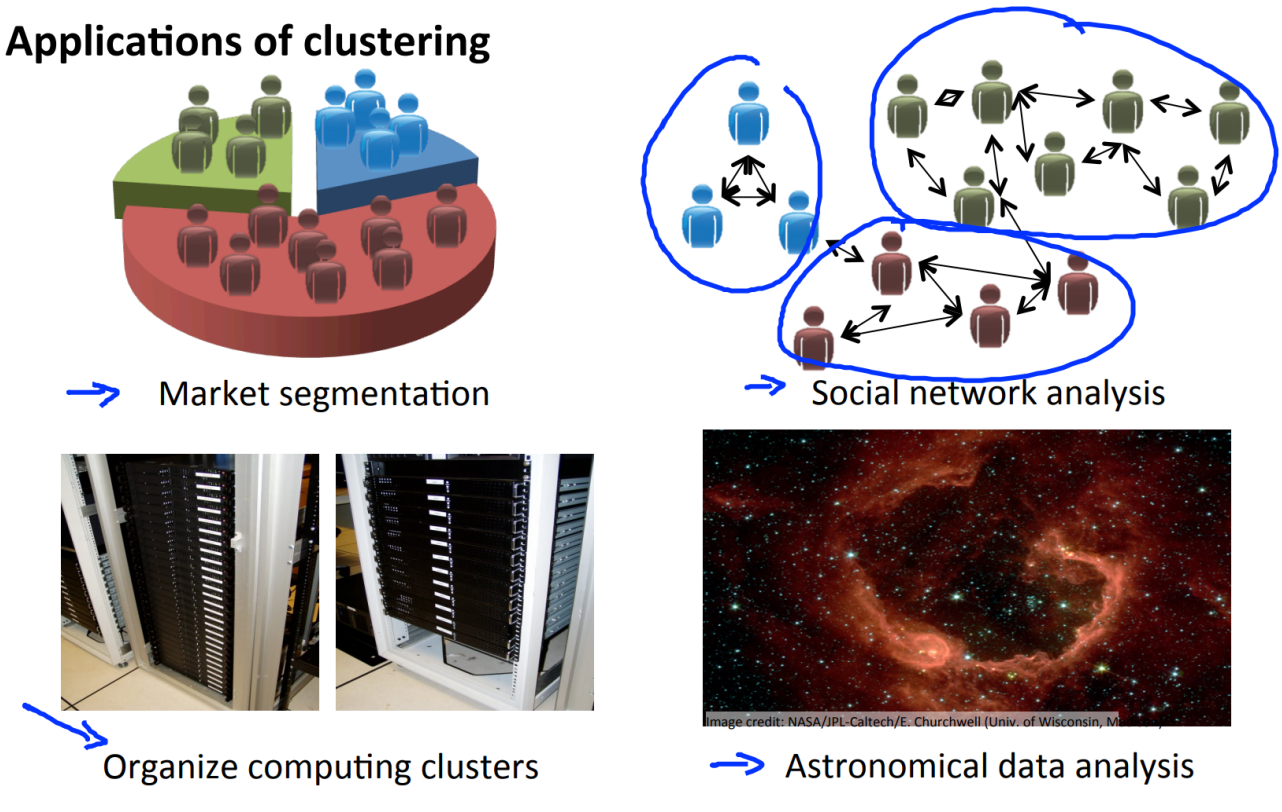
→ Neural network likely to work well for most of these settings, but may be slower to train.

1. Unsupervised learning

what it looks like:



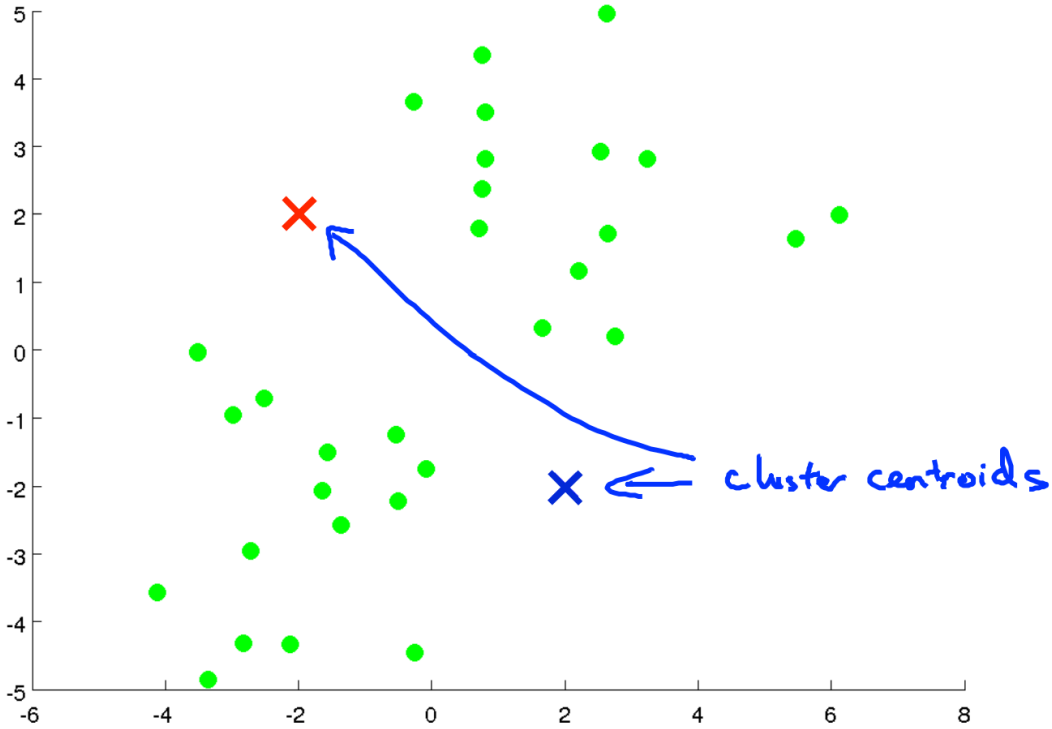
where it is applied:



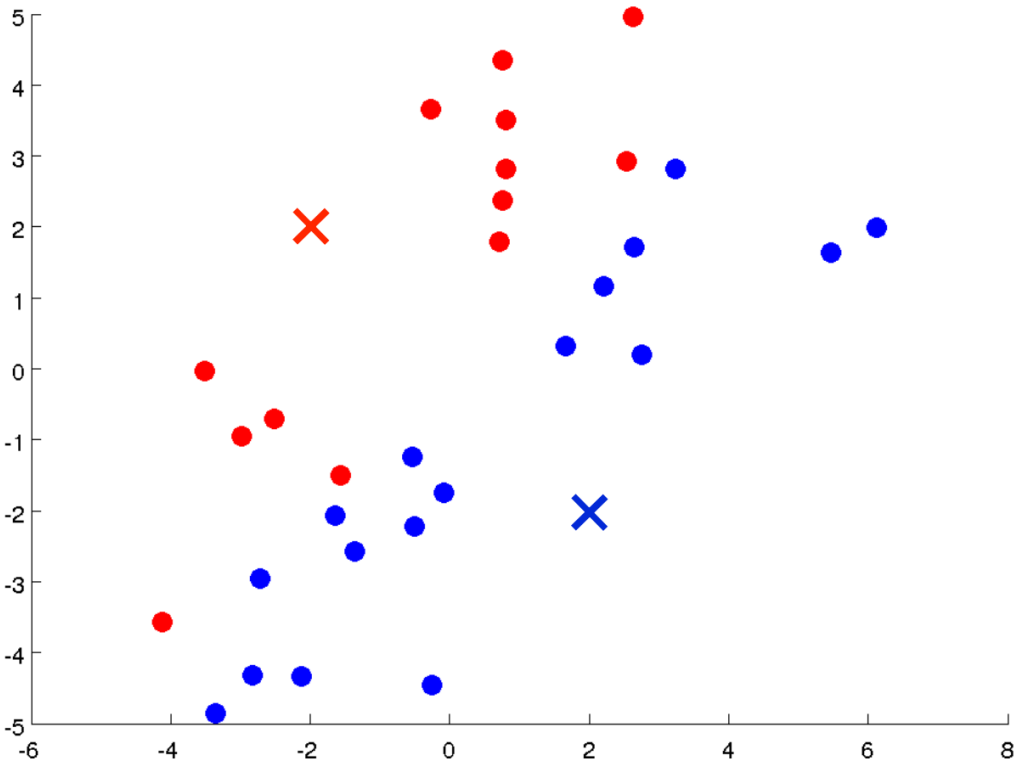
a. K-Means algorithm

steps

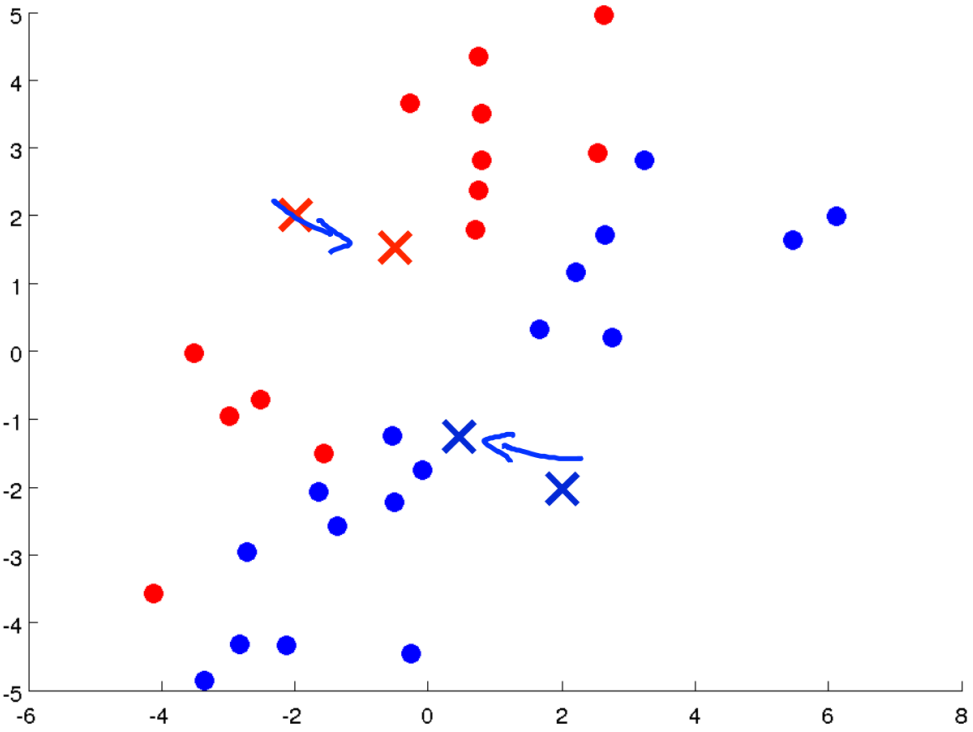
1. randomly choose cluster centroids(聚类中心)



2. assign the training data into clusters by the distance between the data and the cluster centroids



3. move the cluster centroid to the 'average' of each cluster



4. repeat until the centroid remains unchange

more generally, the algorithm goes like this:

Input:

- K (number of clusters)
- Training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

$x^{(i)} \in \mathbb{R}^n$ (drop $x_0 = 1$ convention)

K-means algorithm

Randomly initialize K cluster centroids $\mu_1, \mu_2, \ldots, \mu_K \in \mathbb{R}^n$

Repeat {

Cluster assignment step

for i = 1 to m

$c^{(i)}$:= index (from 1 to K) of cluster centroid closest to $x^{(i)}$

$\min_k \|x^{(i)} - \mu_k\|^2$

move centroid

for k = 1 to K

μ_k := average (mean) of points assigned to cluster k

$x^{(1)}, x^{(5)}, x^{(6)}, x^{(10)}$ → $c^{(1)}=2$, $c^{(5)}=2$, $c^{(6)}=2$, $c^{(10)}=2$

$\mu_2 = \frac{1}{4}\left[x^{(1)} + x^{(5)} + x^{(6)} + x^{(10)}\right] \in \mathbb{R}^n$

}

Note that if no points were assigned to a cluster, a common way to handle this is eliminating the cluster and modify K to $K - 1$. If you really need K clusters, just reinitialize the cluster centroid.

optimization objective

Glossary #7

$c^{(i)}$: index of current cluster to which example $x^{(i)}$ is assigned

K: number of clusters

μ_k: cluster centroid k

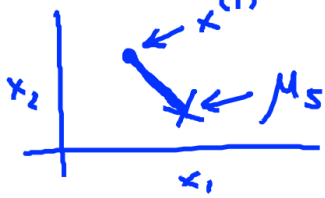
$\mu_{c^{(i)}}$: centroid of cluster to which example $x^{(i)}$ is assigned

the optimization objective(**distortion cost function(失真代价函数)**):

$$J(c^{(1)},\dots,c^{(m)},\mu_1,\dots,\mu_K)=\frac{1}{m}\sum_{i=1}^{m}||x^{(i)}-\mu_{c^{(i)}}||^2$$

$$\min_{c^{(1)},\dots,c^{(m)},\ \mu_1,\dots,\mu_K} J(c^{(1)},\dots,c^{(m)},\mu_1,\dots,\mu_K)$$

Distortion



the cluster assignment step is actually optimizing $c^{(i)}$

the cluster centroid step is actually optimizing μ_k

function J won't sometimes increase!

random initialization

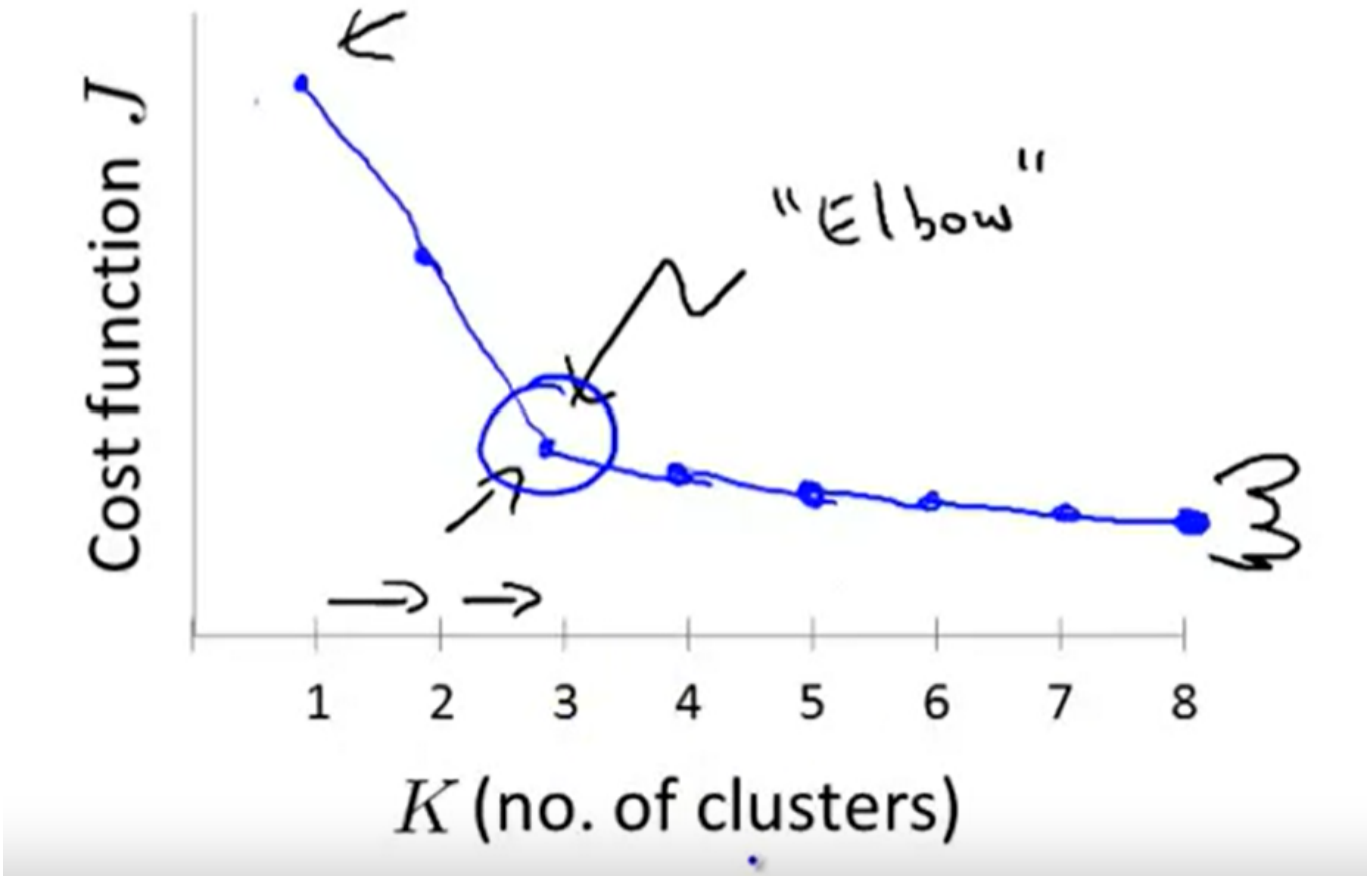
principles:

to avoid local optima, we can **run K-means many times and select the one with the lowest error as the result.**

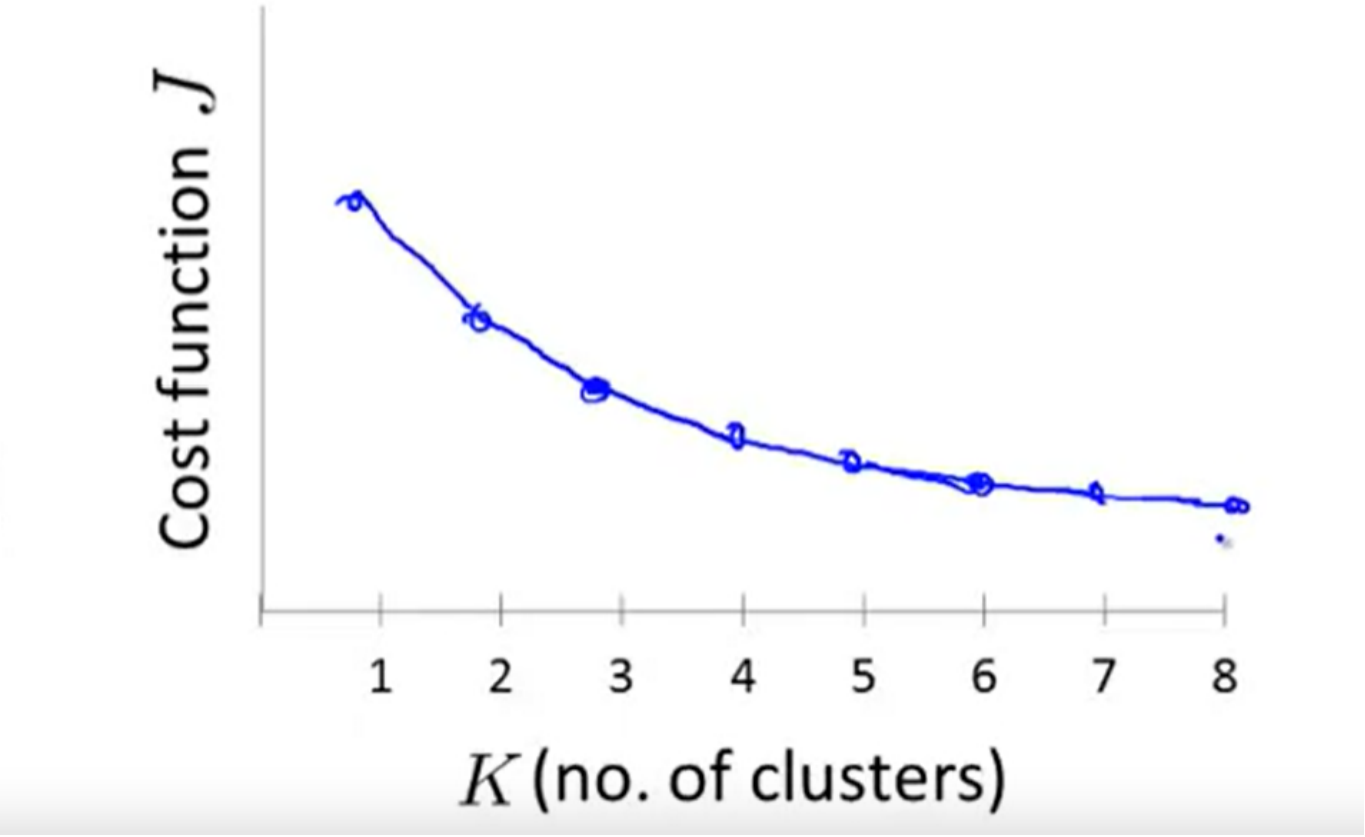
choosing K

Elbow method:

for k=1 to m(exclusive) run K-Means and compute J, correct K shall be the value at the "elbow":



but if the plot is like this:



It is hard to find the "elbow"

2. Dimensionality Reduction

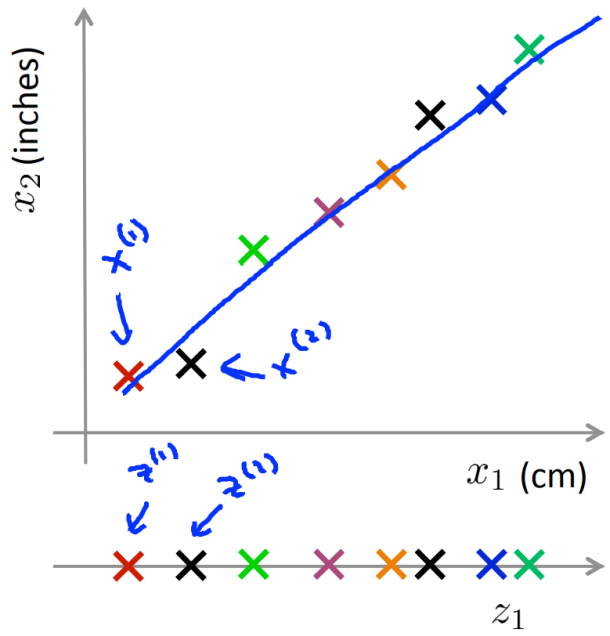
a. motivation

data compression

make the algorithm run faster

(2->1)

Data Compression



Reduce data from 2D to 1D

$x^{(1)} \in \mathbb{R}^2 \rightarrow z^{(1)} \in \mathbb{R}$

$x^{(2)} \in \mathbb{R}^2 \rightarrow z^{(2)} \in \mathbb{R}$

$\vdots$

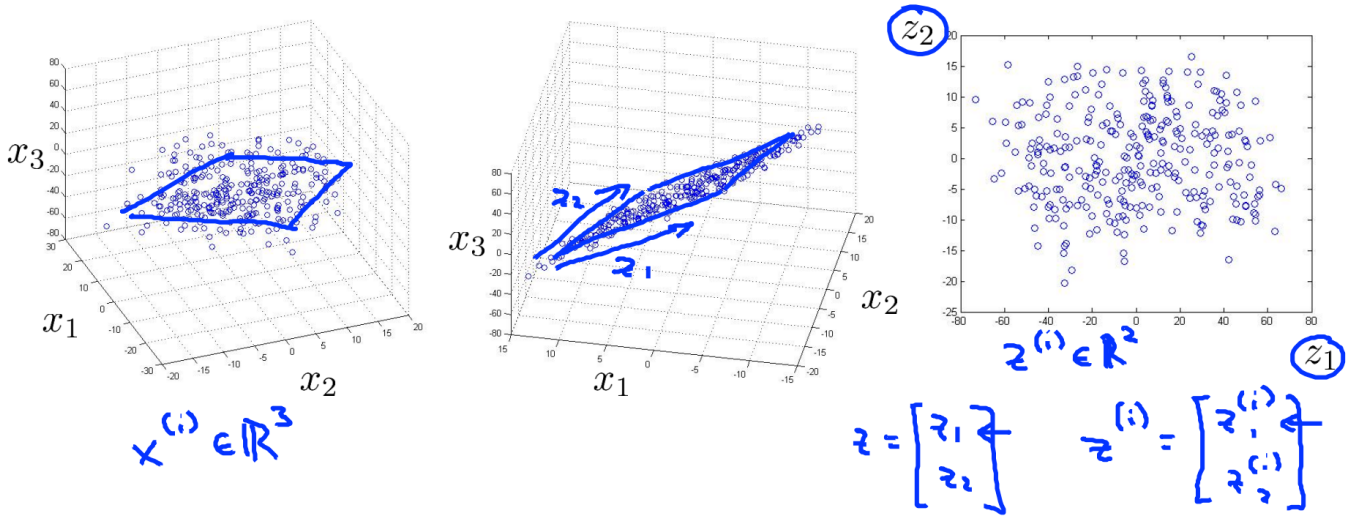
$x^{(m)} \in \mathbb{R}^2 \rightarrow z^{(m)} \in \mathbb{R}$

(3->2)

Data Compression

10000 → 1000

Reduce data from 3D to 2D

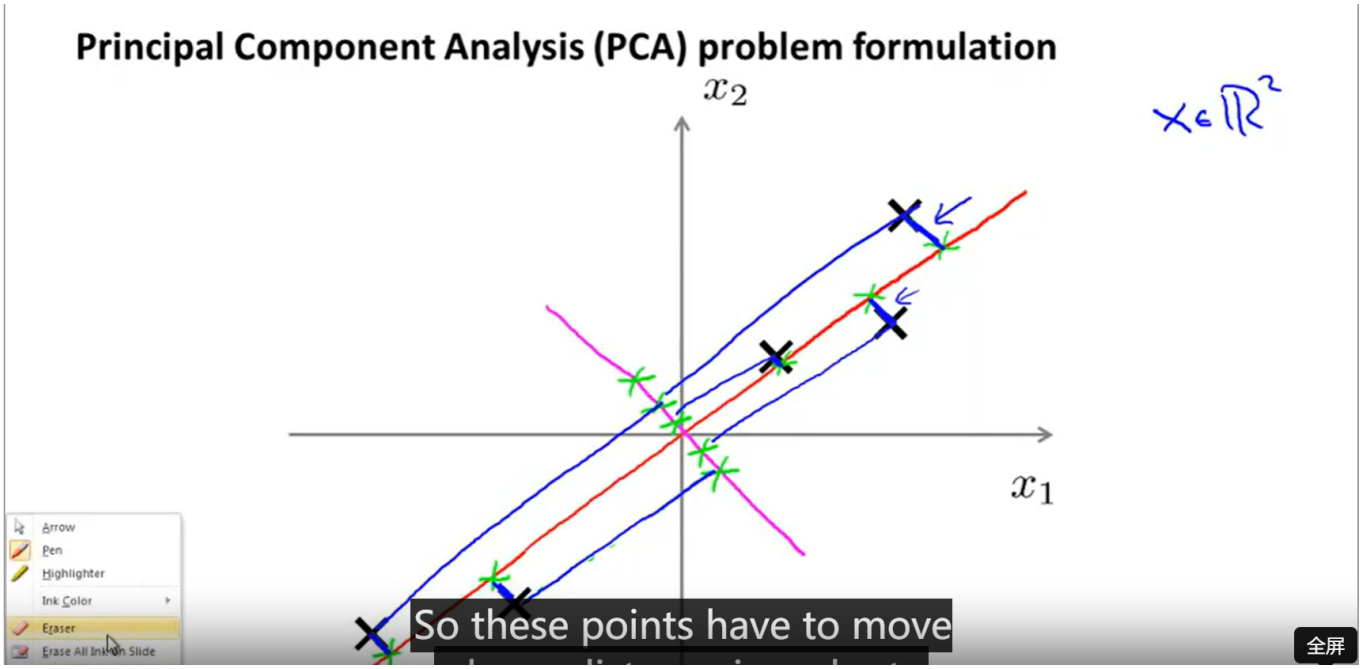


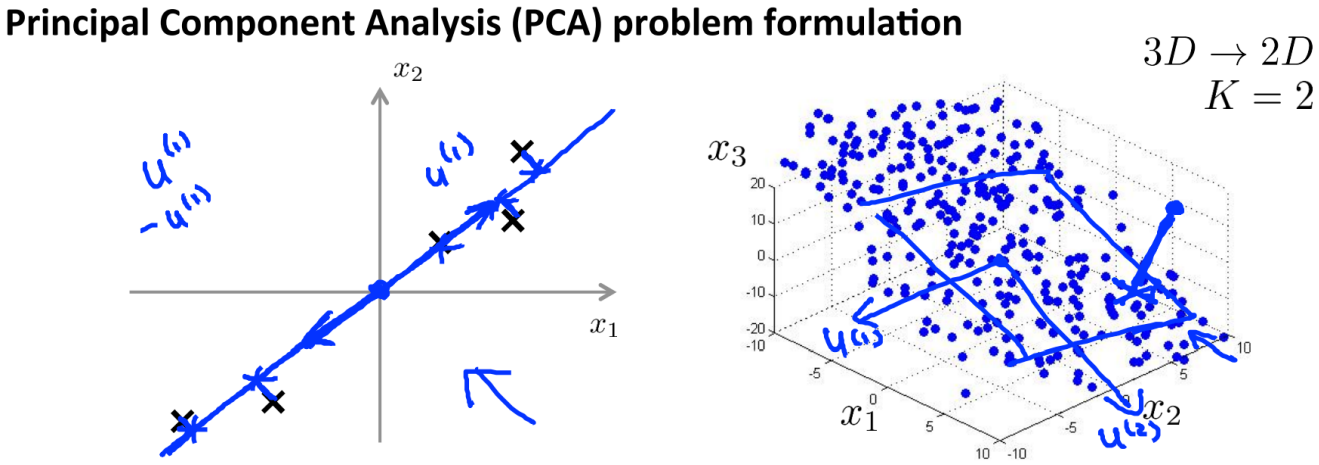
visualization

helps visualize the data easily

b. PCA(Principle Component Analysis)

PCA tries to find a line(super-plain, i.e., k vectors) onto which data is projected where the sum of the **distance** is minimized





Reduce from 2-dimension to 1-dimension: Find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the projection error.

Reduce from n-dimension to k-dimension: Find k vectors $u^{(1)}, u^{(2)}, \ldots, u^{(k)}$ onto which to project the data, so as to minimize the projection error.

PCA is NOT linear regression

- PCA aims to minimize the distance but LR aims to minimize the y-direction square error
- PCA is not for prediction but data dimension reduce

steps of the algorithm

1. data pre-processing: apply mean normalization(must); feature scaling(optional) to all features

Data preprocessing

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

Preprocessing (feature scaling/mean normalization):

$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$

Replace each $x_j^{(i)}$ with $x_j - \mu_j$.

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.

$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$

2. compute covariance matrix and the eigenvectors of the covariance matrix using SVD(Singular-Value Decomposition)

Principal Component Analysis (PCA) algorithm

Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix":

$\Sigma = \frac{1}{m} \sum_{i=1}^{n} (x^{(i)})(x^{(i)})^T$ Sigma

Compute "eigenvectors" of matrix Σ:

```
[U,S,V] = svd(Sigma);
```

Singular value decomposition

eig (Sigma)

The first k columns of the U matrix are exactly $u^{(1)}, u^{(2)}, \ldots u^{(k)}$ and $z = \begin{bmatrix} u^{(1)} & u^{(2)} & \ldots & u^{(k)} \end{bmatrix}^T x$

c. applying PCA

data reconstruction

as $z = \begin{bmatrix} u^{(1)} & u^{(2)} & \dots & u^{(k)} \end{bmatrix}^T x$

$x = (\begin{bmatrix} u^{(1)} & u^{(2)} & \dots & u^{(k)} \end{bmatrix}^T)^{-1} z = \begin{bmatrix} u^{(1)} & u^{(2)} & \dots & u^{(k)} \end{bmatrix} x$ (pseudo inverse here)

Here is why $\begin{bmatrix} u^{(1)} & u^{(2)} & \dots & u^{(k)} \end{bmatrix}^T = \begin{bmatrix} u^{(1)} & u^{(2)} & \dots & u^{(k)} \end{bmatrix}^{-1}$

Neil Ostrove 社区助教

Both are because the eigenvectors returned by Singular Value Decomposition (SVD) form an orthonormal basis.

喜欢 2 个回复 ••• 更多

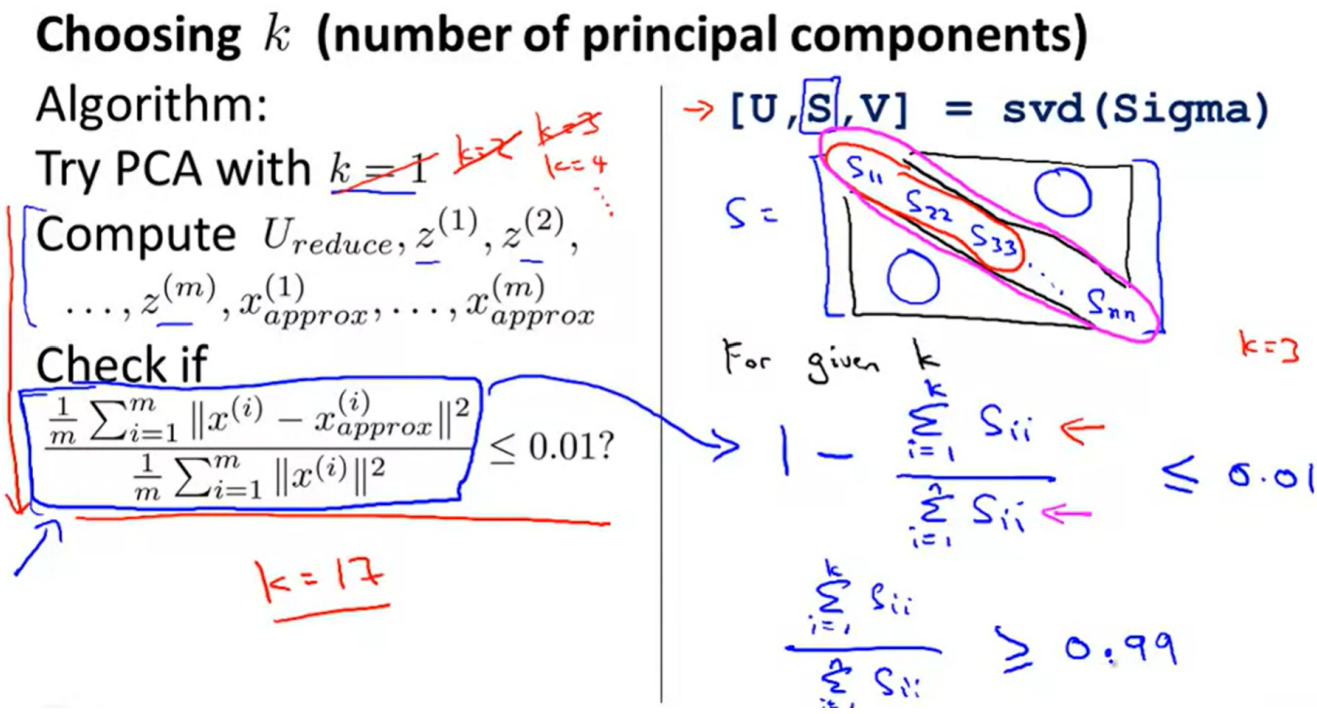
choosing k

principle:

$$\rightarrow \frac{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2} \leq 0.01 \qquad (1\%)$$

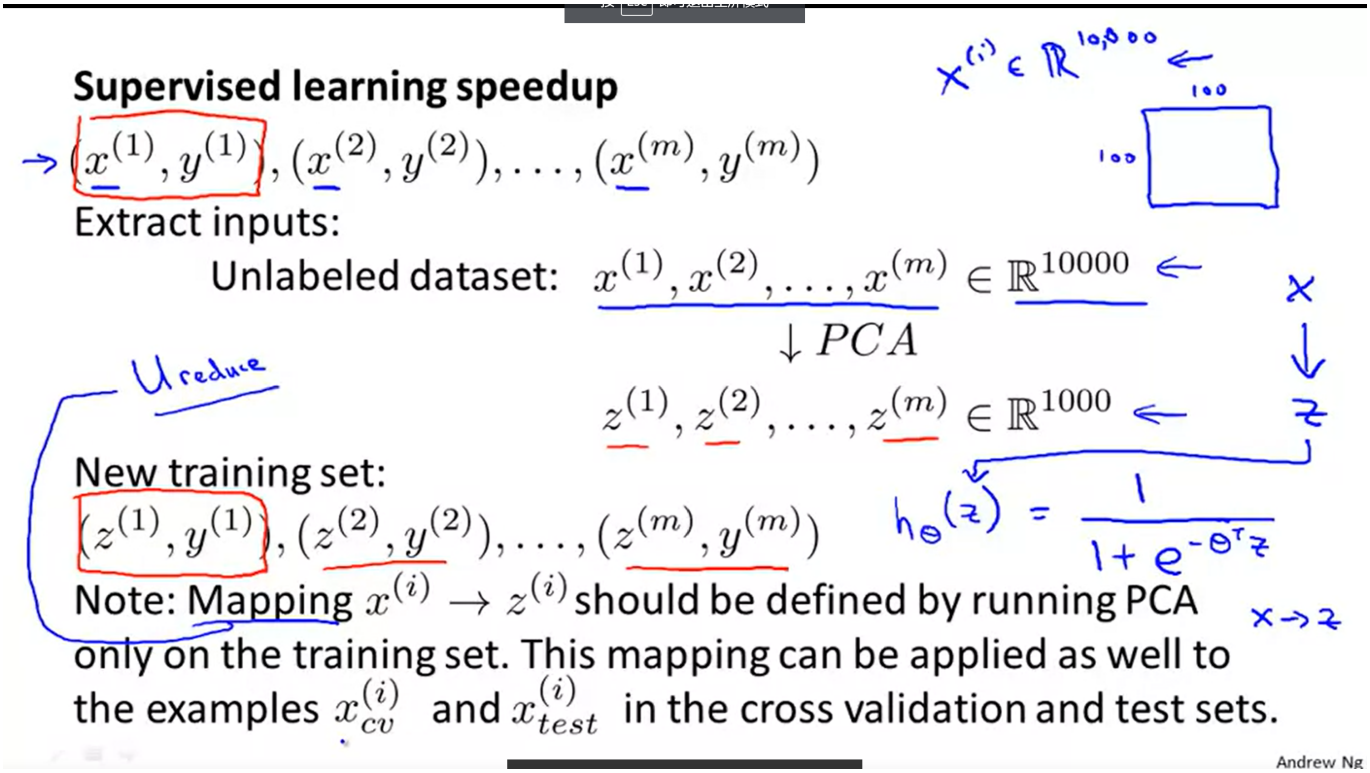
$$\rightarrow$$

algorithm:



application

1. supervised learning speedup



2. **DO NOT** use PCA to prevent overfitting, apply regularization instead
3. **DO NOT** use PCA if unnecessary

W9

1. Anomaly Detection

a. density estimation

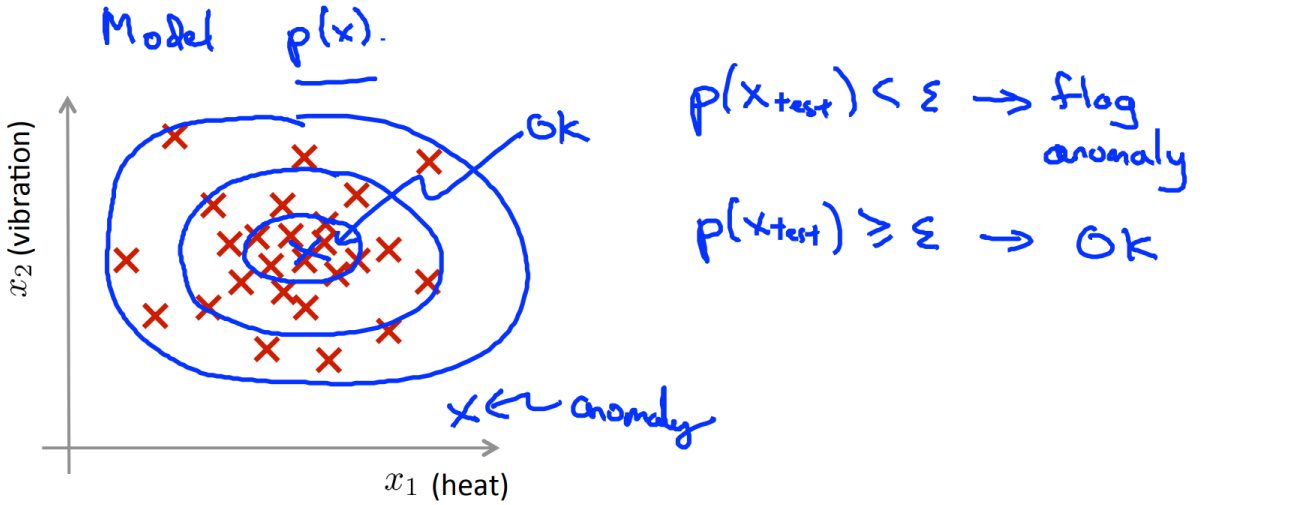
motivation

find out the point that is anomalous:

Density estimation

→ Dataset: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

→ Is x_{test} anomalous?

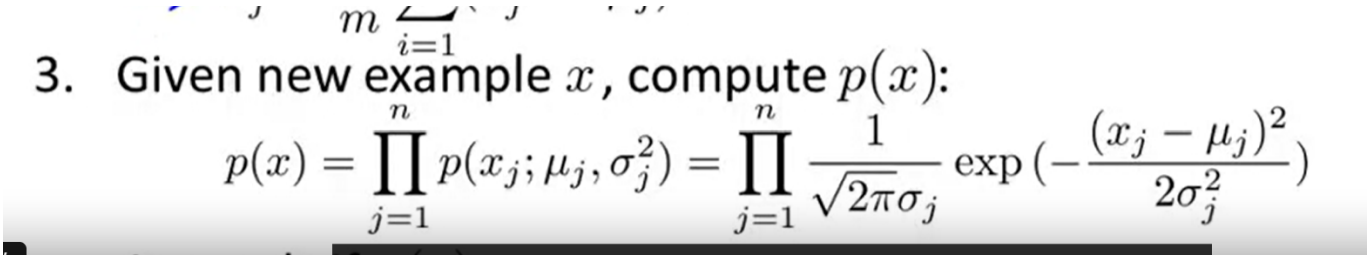


Gaussian distribution(trivial)

algorithm

1. choose probable anomalous features x_i
2. fit μ_j, σ_j^2 using:

3. compute $p(x)$ for a new example x using:



4. anomaly if $p(x) < \varepsilon$

b. building an anomaly detection system

evaluation metrics

Algorithm evaluation

→ Fit model $p(x)$ on training set $\{x^{(1)}, \ldots, x^{(m)}\}$

→ On a cross validation/test example x, predict

$$y = \begin{cases} 1 & \text{if } p(x) < \varepsilon \text{ (anomaly)} \\ 0 & \text{if } p(x) \geq \varepsilon \text{ (normal)} \end{cases}$$

Possible evaluation metrics:

- True positive, false positive, false negative, true negative
- Precision/Recall
- F_1-score

Can also use cross validation set to choose parameter ε

precision/recall/F score(see W6)

Note that **classification accuracy is not used as a metric in case of skewed data**

anomaly detection vs. supervised learning

different features:

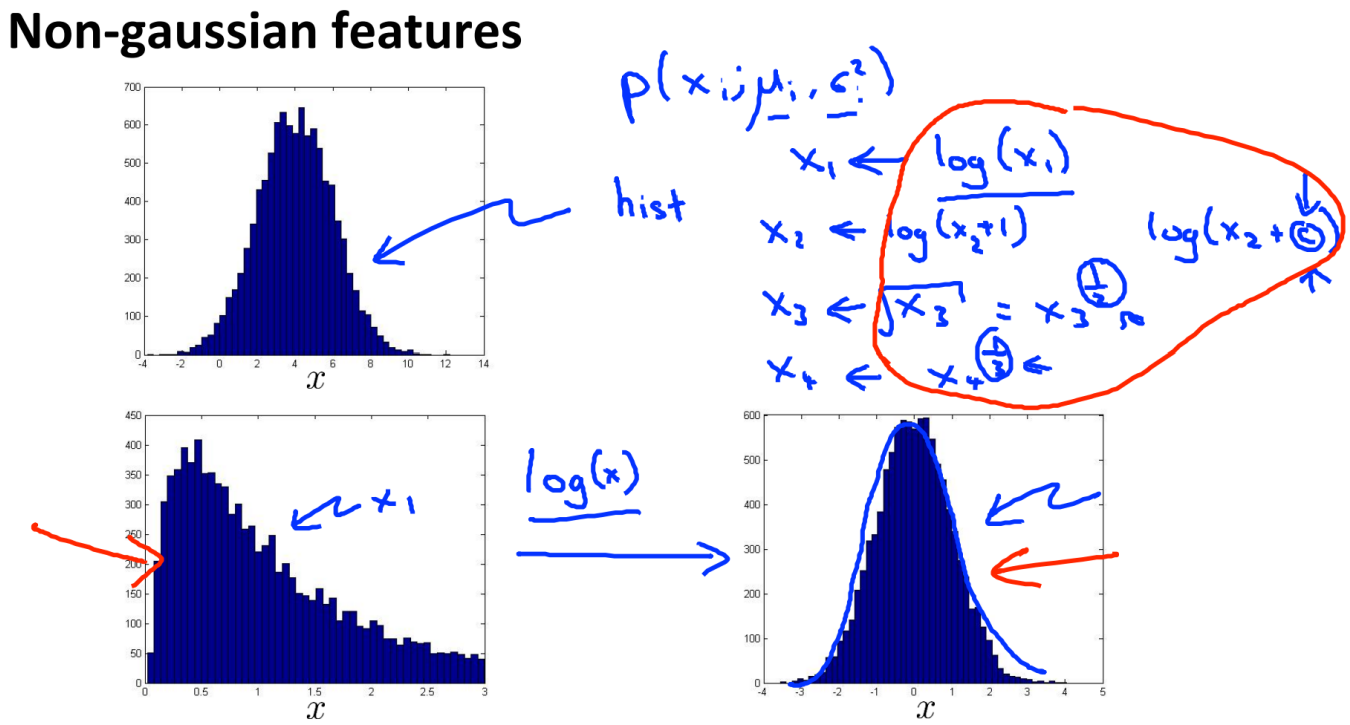
Anomaly detection	**vs.**	**Supervised learning**
→ Very small number of positive examples ($y = 1$). (0-20 is common).		Large number of positive and negative examples.
→ Large number of negative ($y = 0$) examples. $p(x)$		
→ Many different "types" of anomalies. Hard for any algorithm to learn from positive examples what the anomalies look like; future anomalies may look nothing like any of the anomalous examples we've seen so far.		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

different usages:

Anomaly detection	vs.	Supervised learning
• Fraud detection		• Email spam classification
• Manufacturing (e.g. aircraft engines)		• Weather prediction (sunny/ rainy/etc).
• Monitoring machines in a data center		• Cancer classification
⋮		⋮

choosing what features to use

handling non-Gaussian features:



do some transformations to make it more Gaussian:

1. $x := \log(x + c)$
2. $x := x^c$

the constant c is what we adjust.

use `hist(x)` to plot the histogram of the data

error analysis:

what we want:

> → **Error analysis for anomaly detection**
>
> Want $p(x)$ large for normal examples x.
> $p(x)$ small for anomalous examples x.

the problem is:

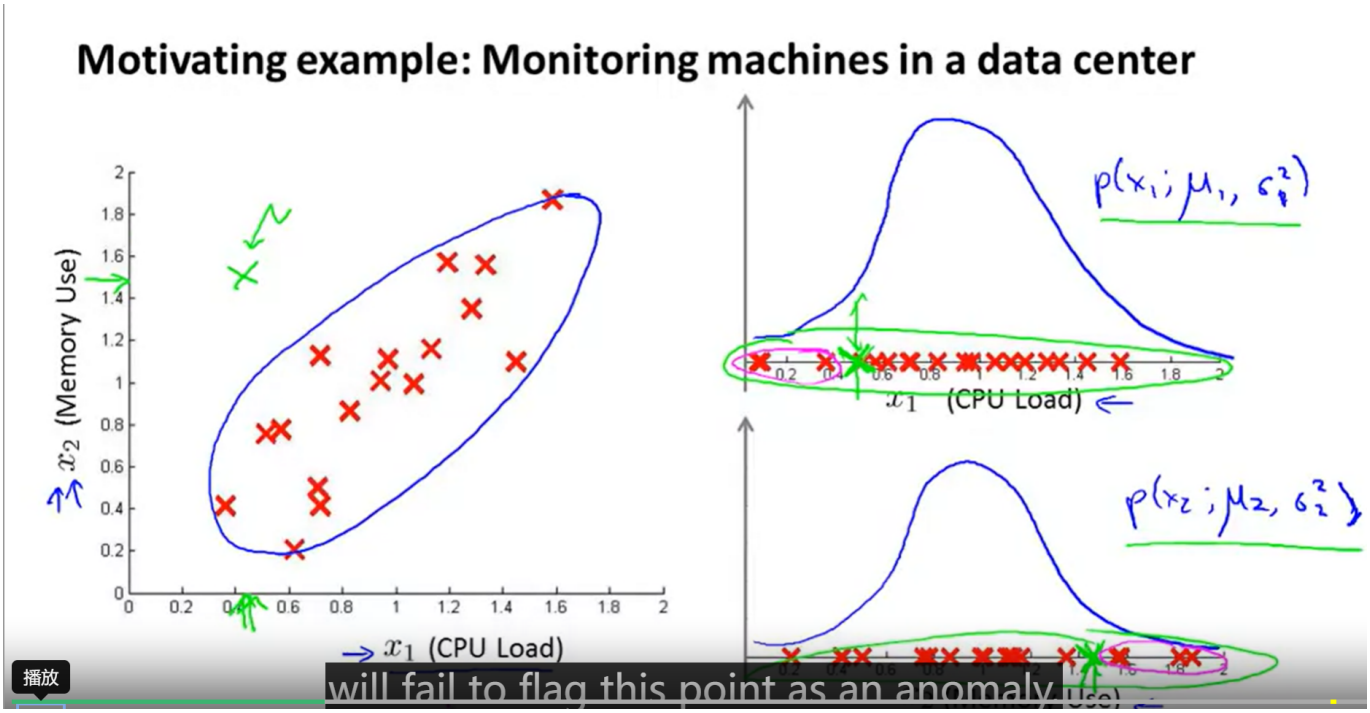
Most common problem:

$p(x)$ is comparable (say, both large) for normal and anomalous examples

solution: combine features that will distinguish between normal cases and anomalous cases

c. multivariant Gaussian distribution

an example when univariant Gaussian fails to detect the anomaly:



the green point is anomalous but the Gaussian values of both features are relatively high.

MGD

Instead of model $p(x_i)$ separately, we model $p(x)$ in one go.

parameters: $\mu \in \mathbb{R}^n$ $\Sigma \in \mathbb{R}^{n \times n}$, where Σ is the covariance matrix

the PDF for MGD:

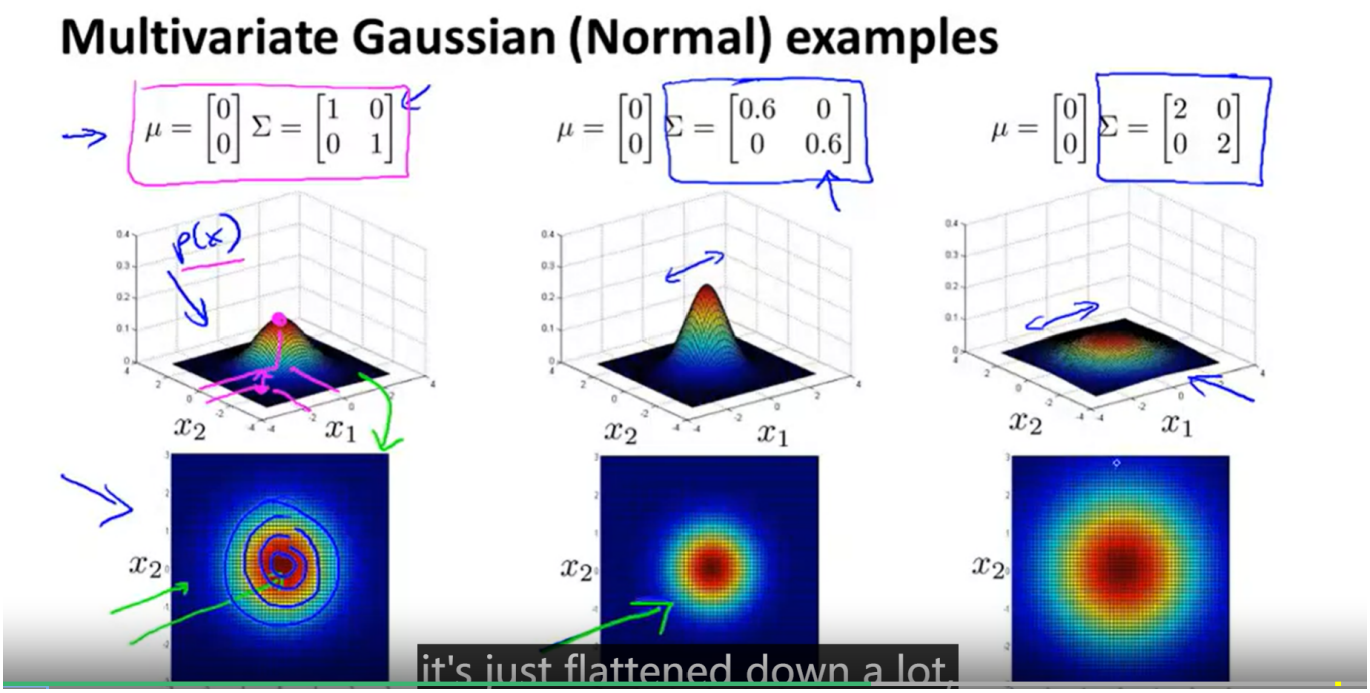
$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{1}{2}} |\Sigma|^{\frac{1}{2}}} \exp\left(\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

where x is a vector

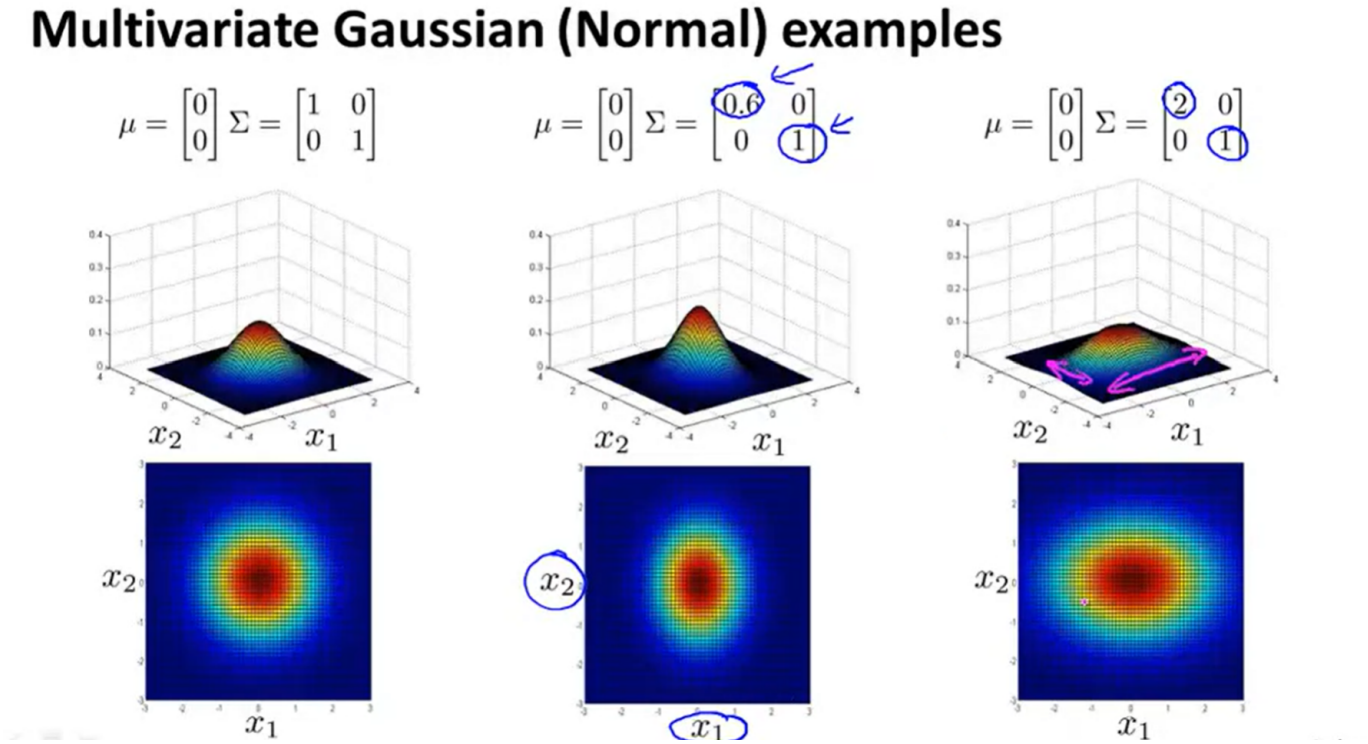
examples

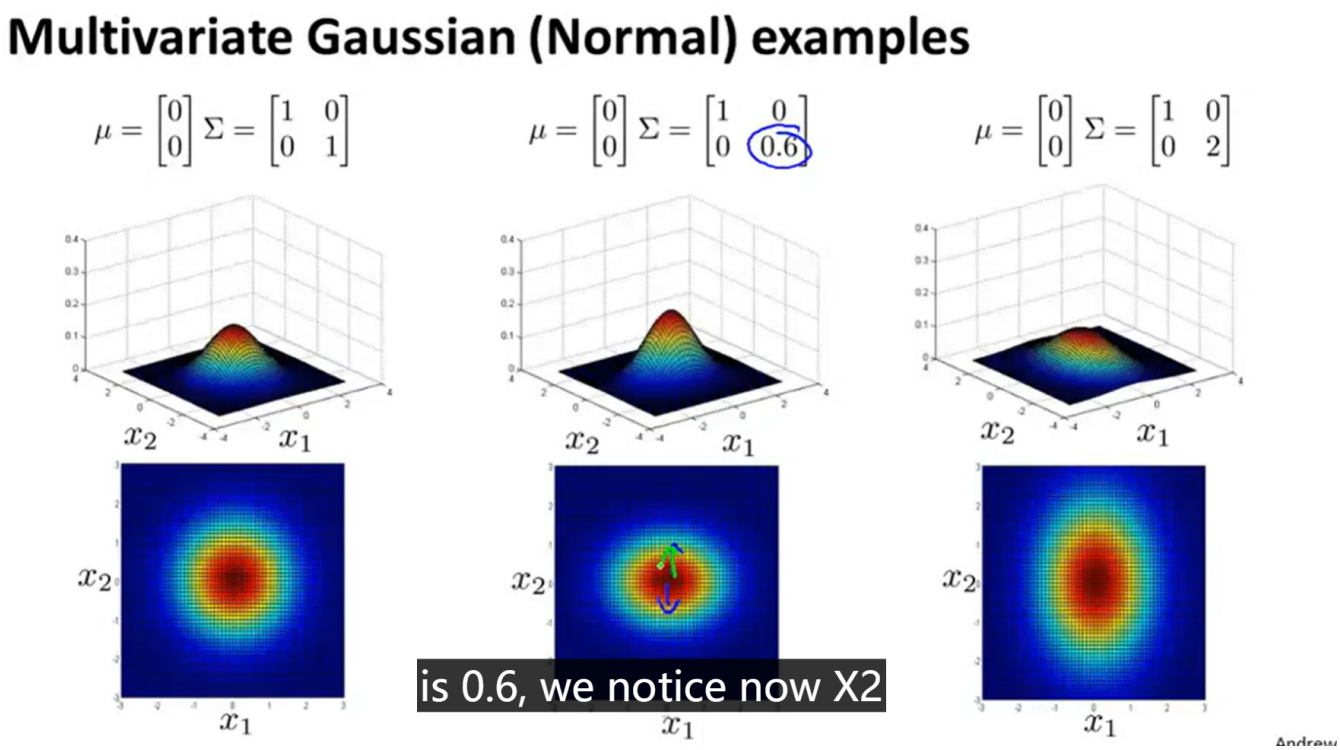
1. varying the diagonal elements of Σ(**the range of each feature**):

synchronized:

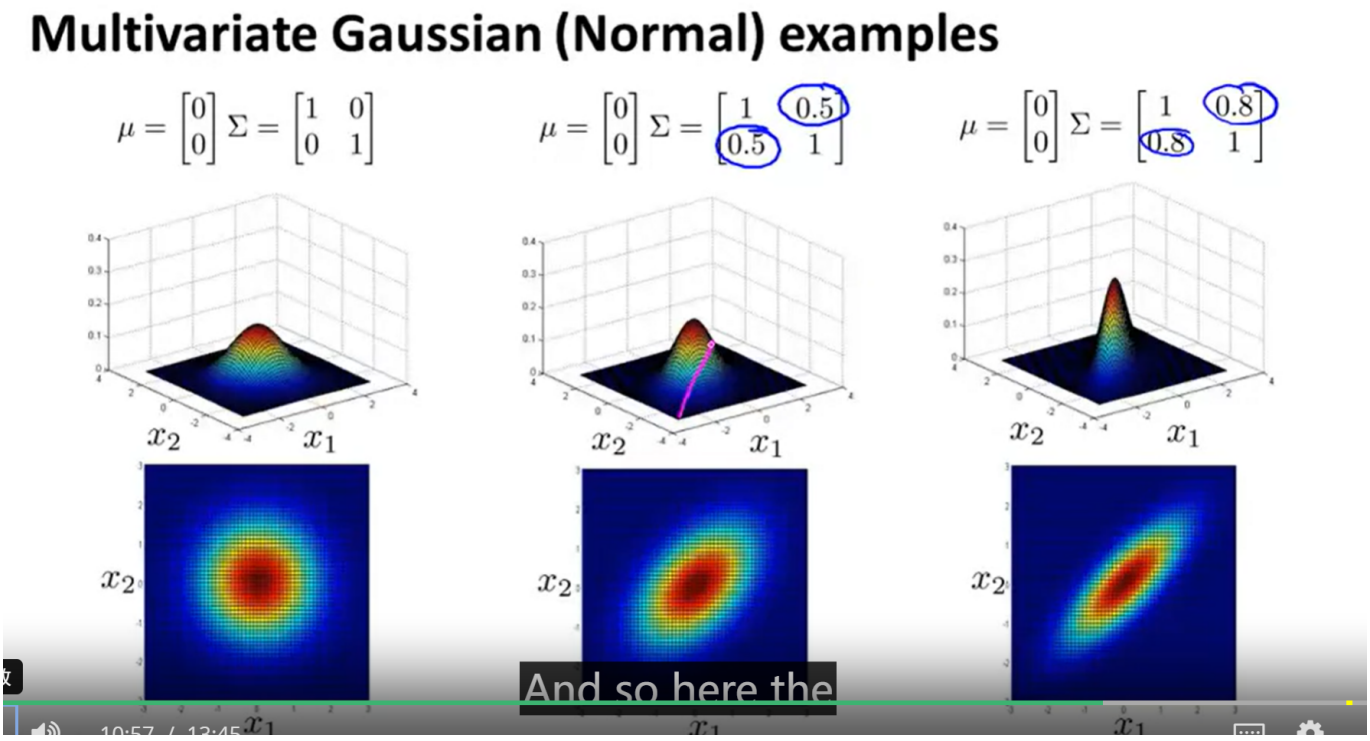


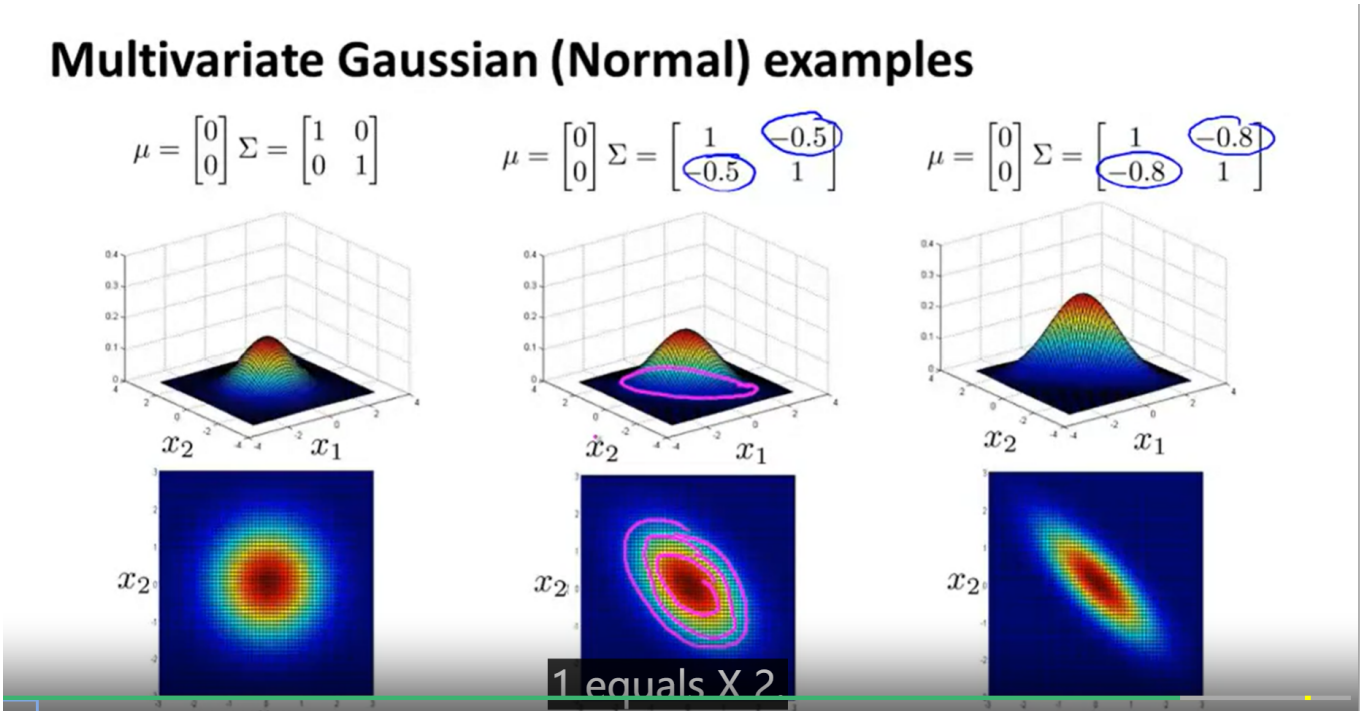
asynchronized:



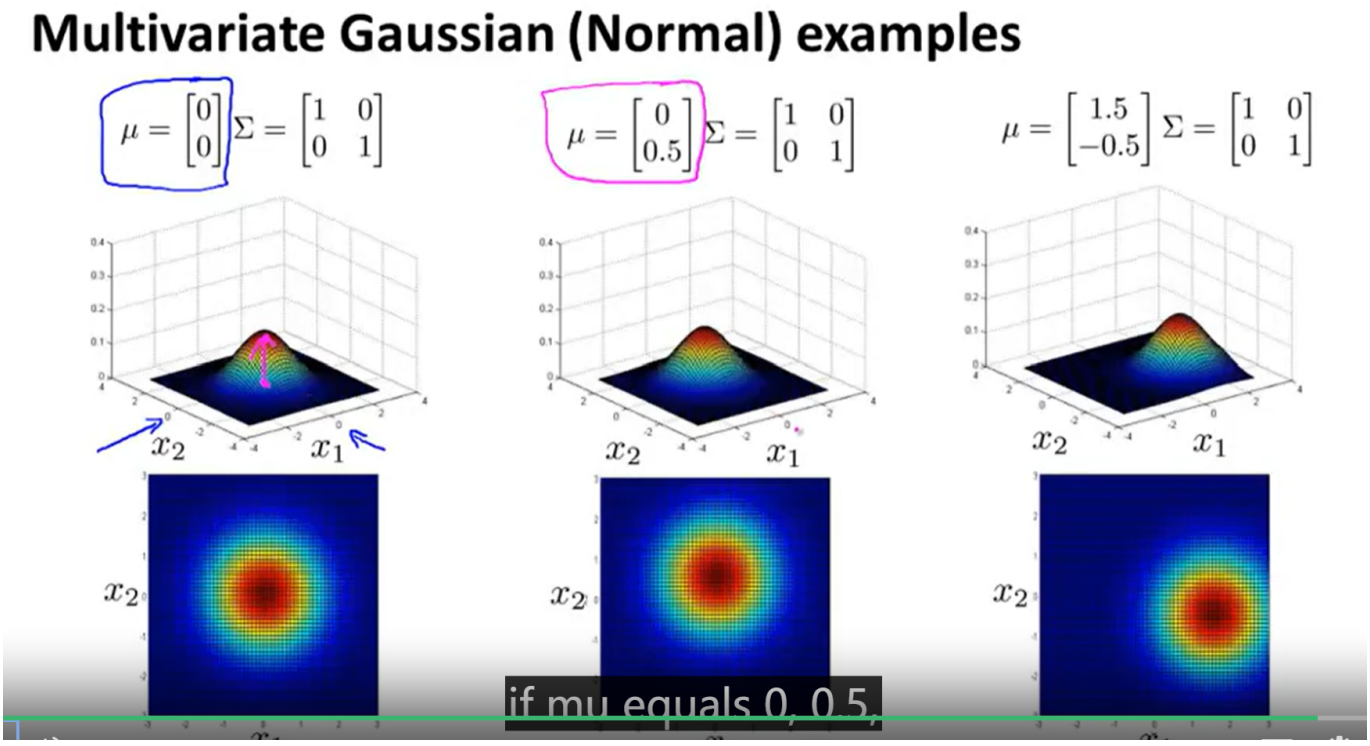


2. varying off-diagonal elements of Σ(**the correlation between each feature**):





3. varying μ(**the location of the peak of the function**)



anomaly detection using MGD

steps:

1. fitting parameters using the same approach as PCA.

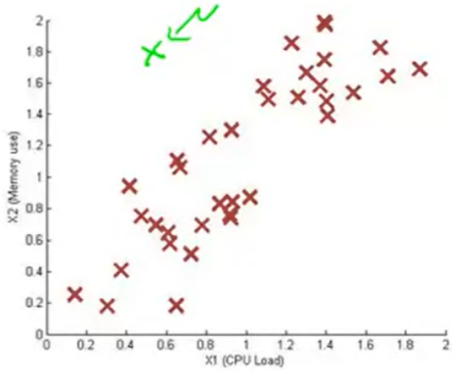
2. compute the PDF of MGD of the new example:

Anomaly detection with the multivariate Gaussian

1. Fit model $p(x)$ by setting

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$\Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)(x^{(i)} - \mu)^T$$



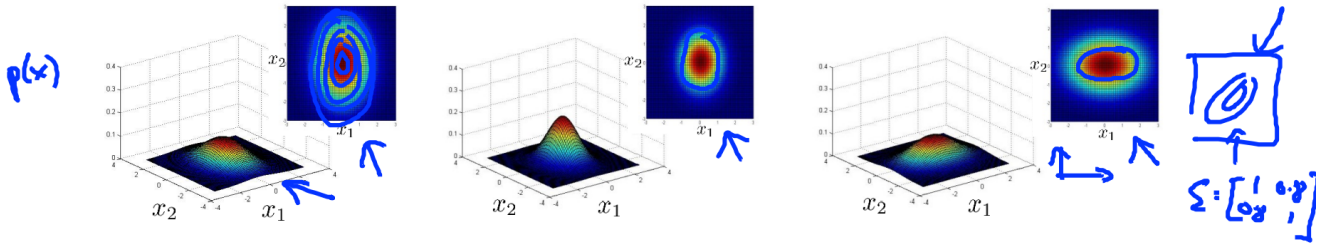
2. Given a new example x, compute

$$p(x) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

relationship to the original model:

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

$$p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

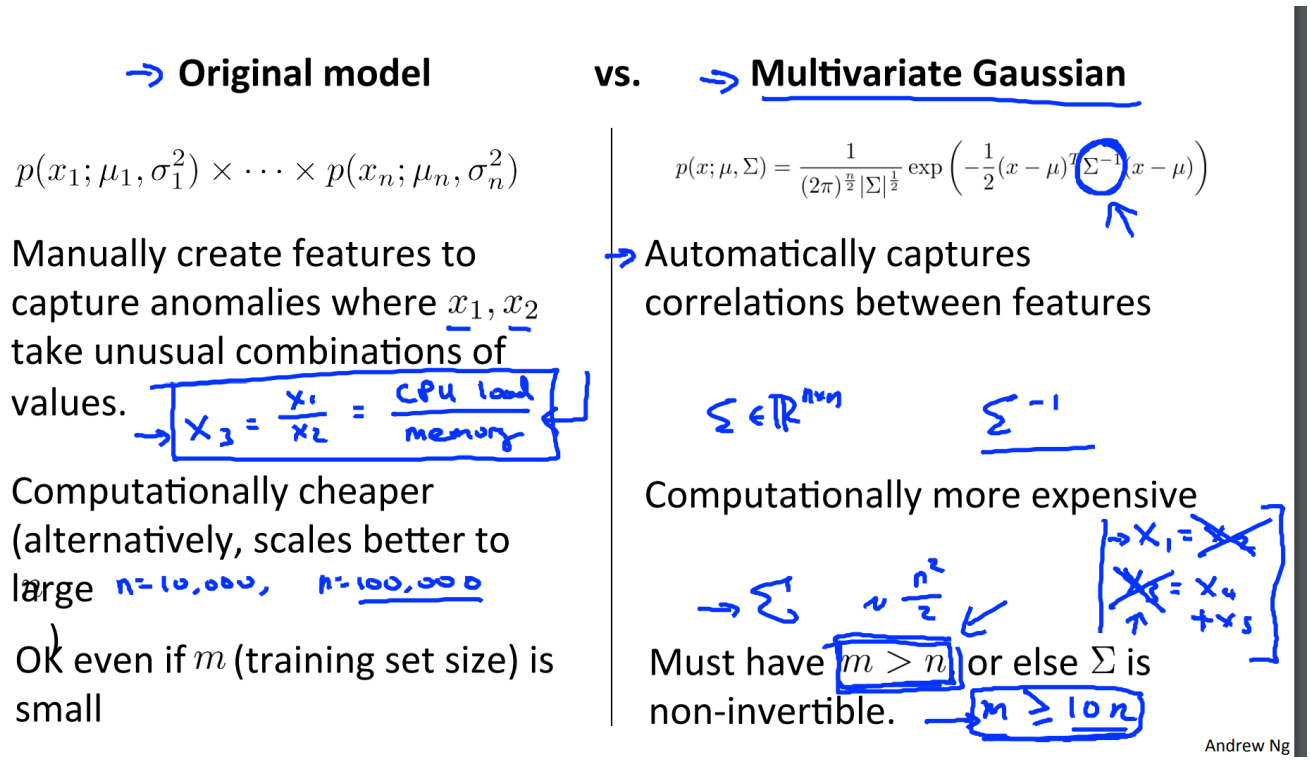
where

Andrew Ng

constraints: the off-diagonal elements shall all be 0s

The MGD automatically take the correlations between features

here is more concrete comparisons:



if Σ is non-invertible, then there are 2 possible reasons:

- $m \leq n$
- redundant features

2. Recommender Systems

a. predicting movie ratings(content-based)

Glossary #8

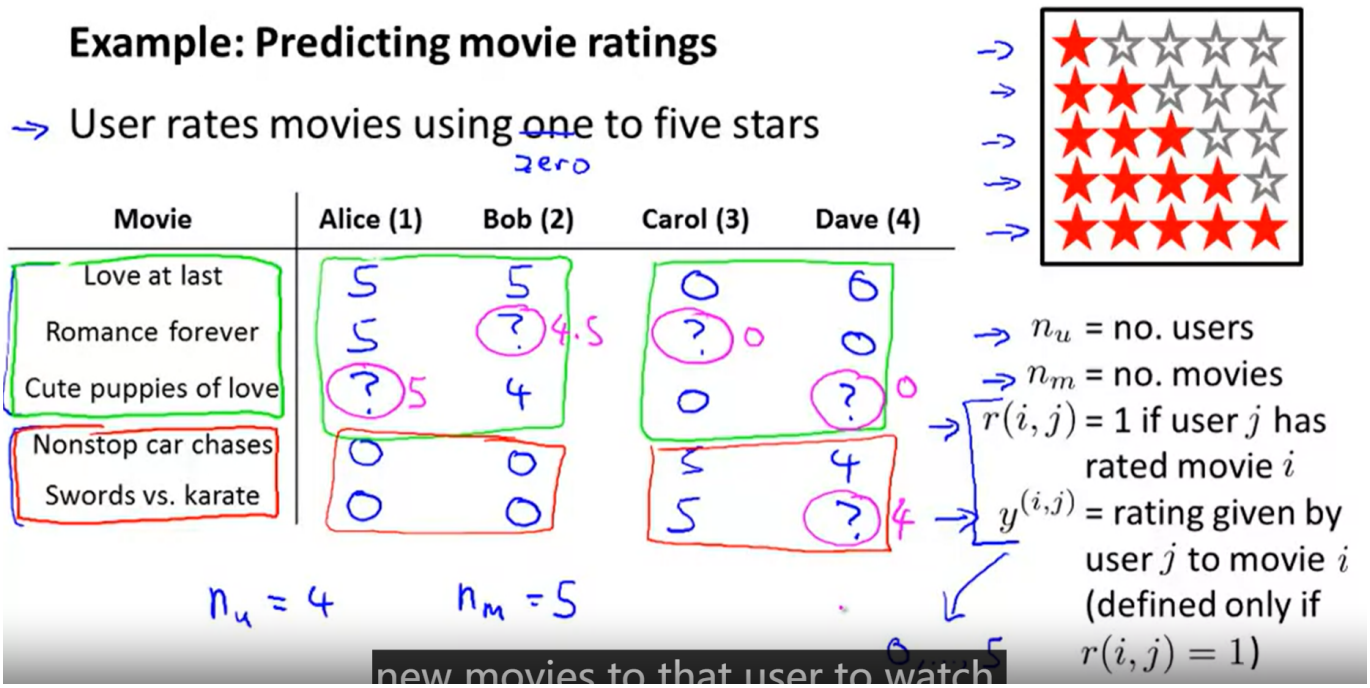
n_u : number of users

n_m : number of movies

$r(i,j)$: 1 if user j has rated movie i

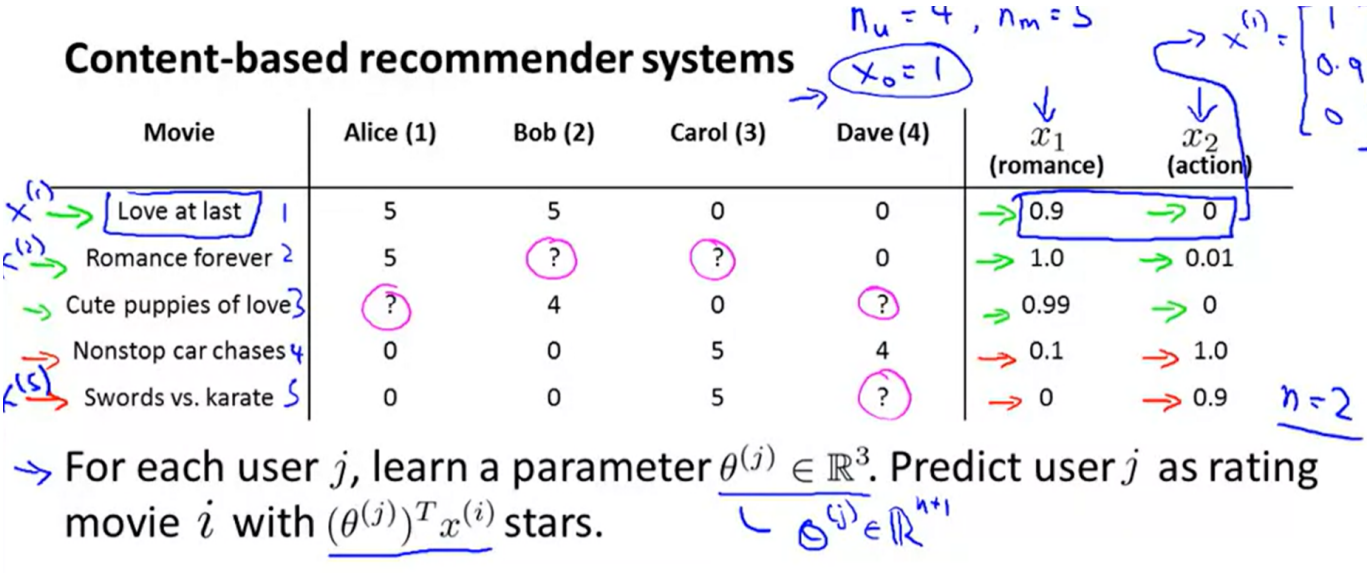
$y^{(i,j)}$ the rate given by user j to movie i (defined only when $r(i,j)=1$)

recommender system tries to fill in the blanks marked '?' and find out what a user likes

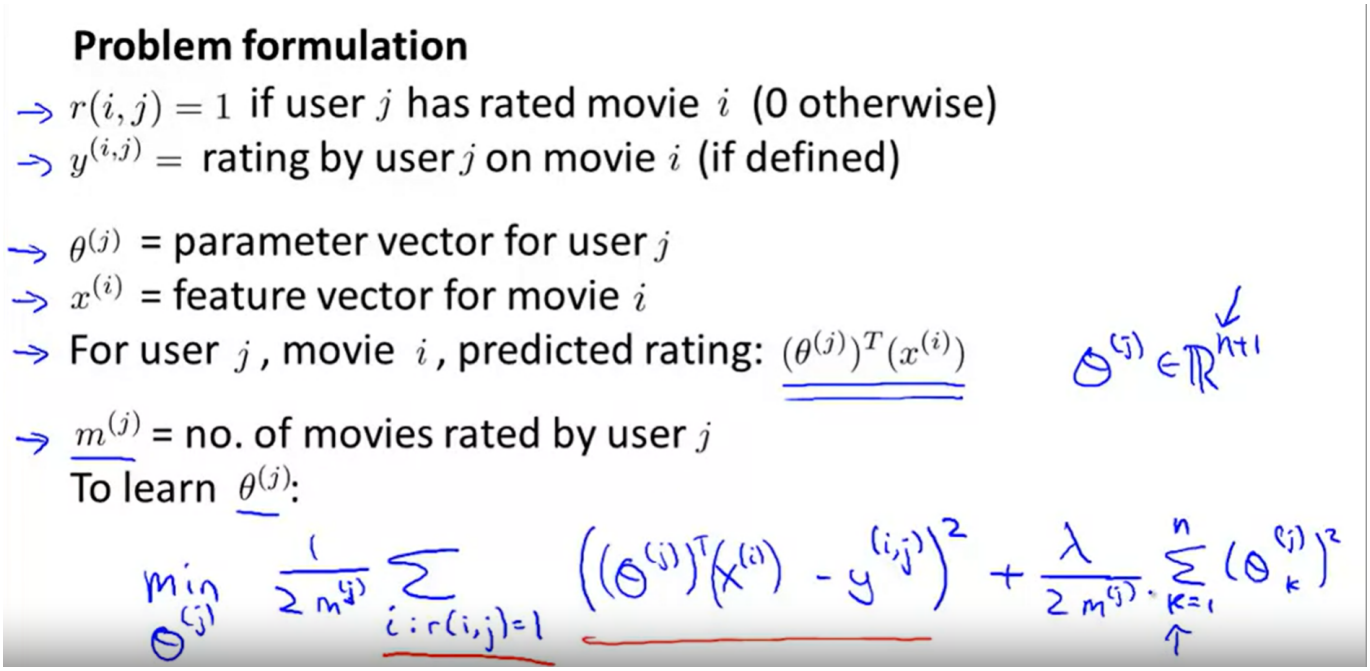


content based recommendations

idea:

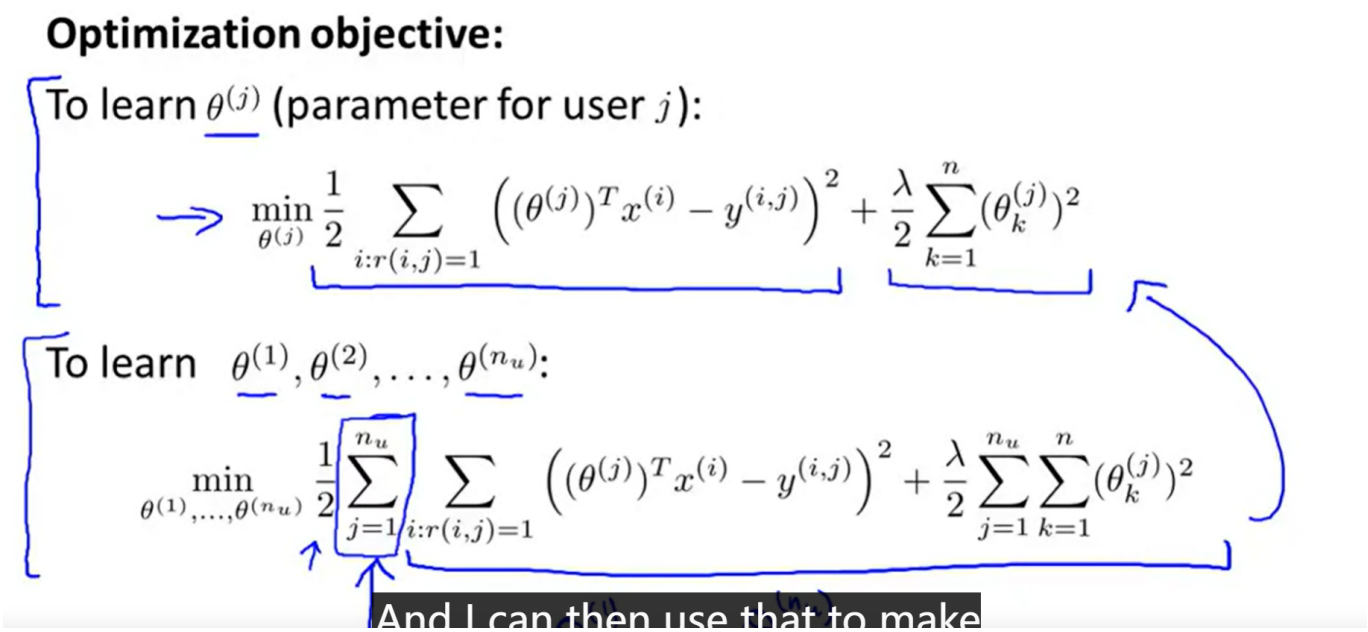


problem formulation

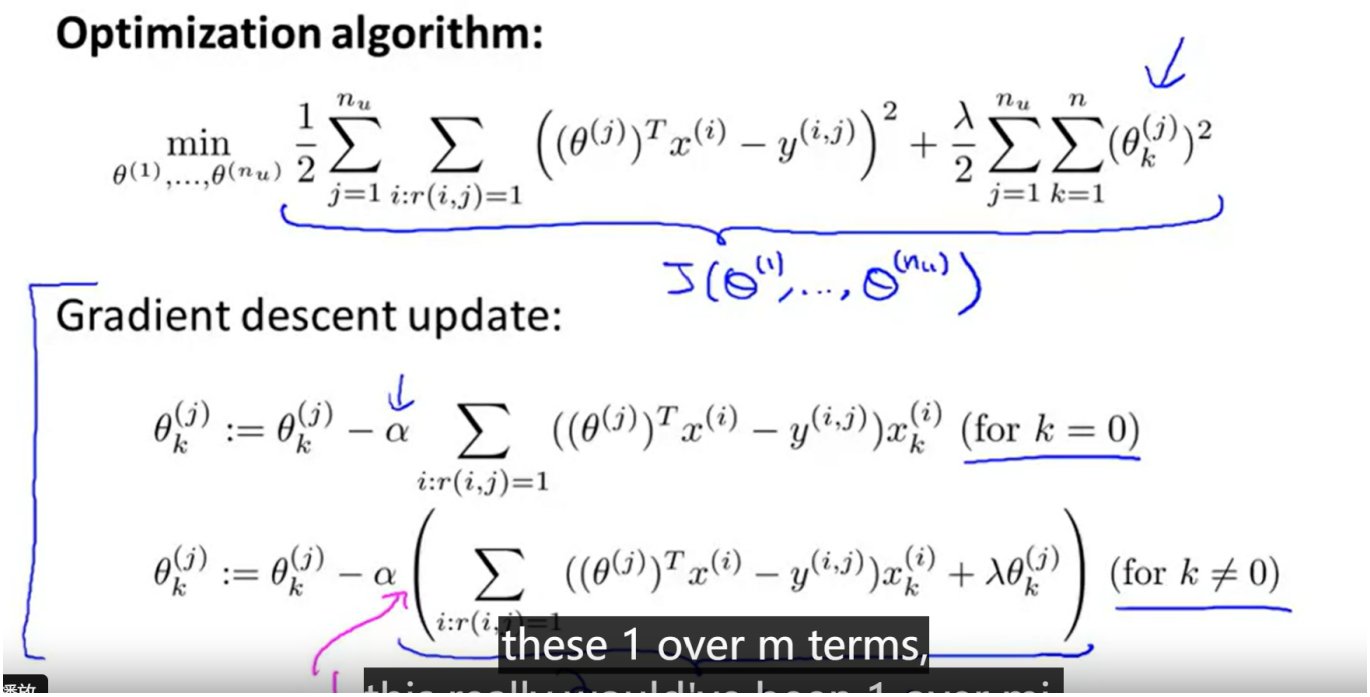


to simplify, $m^{(j)}$ is ignored:

the optimization objective:



algorithm: gradient descent again:



b. collaborative filtering(协同过滤)

idea

assumption #1: we have no idea about the features of the movie(this is more realistic)

Problem motivation

Movie	Alice (1)	Bob (2)	Carol (3)	Dave (4)	x_1 (romance)	x_2 (action)
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	?	?	?

assumption #2: users have told us what kind of movie they like. i.e., the θ parameters are given

Problem motivation

Movie	Alice (1) $\theta^{(1)}$	Bob (2) $\theta^{(2)}$	Carol (3) $\theta^{(3)}$	Dave (4) $\theta^{(4)}$	x_1 (romance)	x_2 (action)
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	?	?	?

$$\theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}, \theta^{(2)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix}, \theta^{(3)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}, \theta^{(4)} = \begin{bmatrix} 0 \\ 0 \\ 5 \end{bmatrix}$$

that she really

we can infer the value of features from the θ values and the rates:

Optimization algorithm

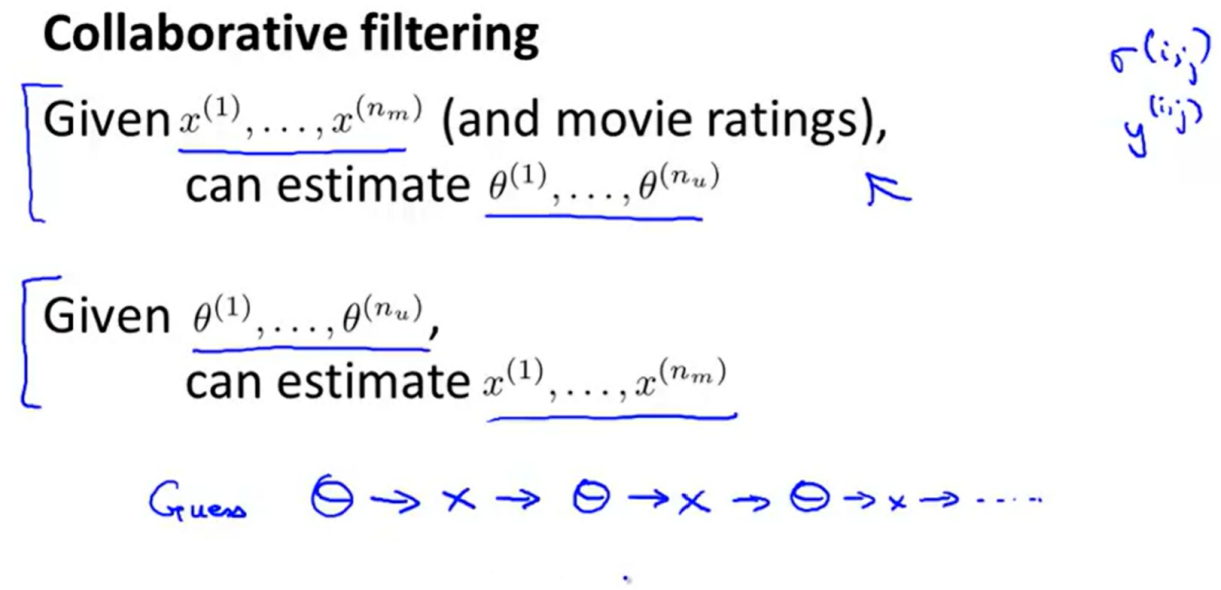
Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$, to learn $x^{(i)}$:

$$\min_{x^{(i)}} \frac{1}{2} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (x_k^{(i)})^2$$

Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$, to learn $x^{(1)}, \ldots, x^{(n_m)}$:

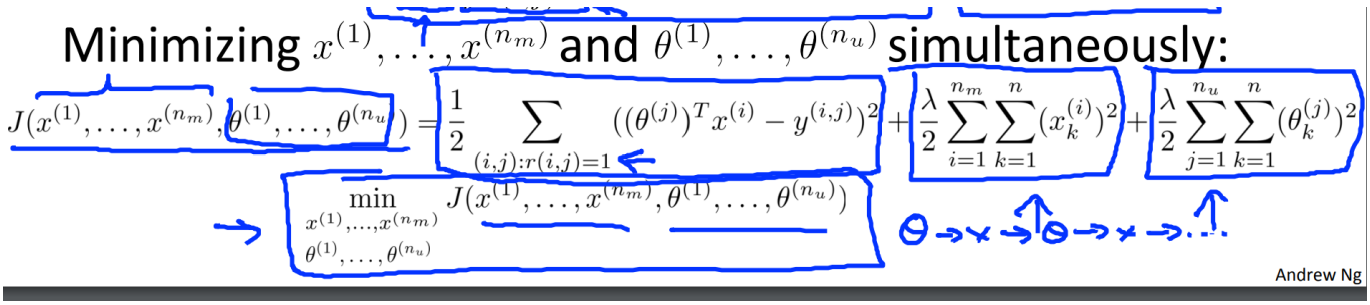
$$\min_{x^{(1)}, \ldots, x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} ((\theta^{(j)})^T x^{(i)} - y^{(i,j)})^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2$$

this is essentially the same algorithm as the content-based one.



the chain is interesting isn't it? But it is not as efficient as it can be. we can solve θ and x simultaneously by combining the cost functions together.

optimization objective



we ignore the convention that $x_0 = 1$ because the algorithm can do this itself.

algorithm

steps:

1. initialize x and θ to small random values(**to ensure that $x^{(1)}$ to $x^{(n_m)}$ can be learnt separately**)
2. use GD to minimize J

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 $j=1,\dots,n_u, i=1,\dots,n_m$:

$$x_k^{(i)} := x_k^{(i)} - \alpha\left(\sum_{j:r(i,j)=1}((\theta^{(j)})^T x^{(i)} - y^{(i,j)})\theta_k^{(j)} + \lambda x_k^{(i)}\right)$$

$$\theta_k^{(j)} := \theta_k^{(j)} - \alpha\left(\sum_{i:r(i,j)=1}((\theta^{(j)})^T x^{(i)} - y^{(i,j)})x_k^{(i)} + \lambda\theta_k^{(j)}\right)$$

$\frac{2}{2x_k^{(i)}} J(\dots)$

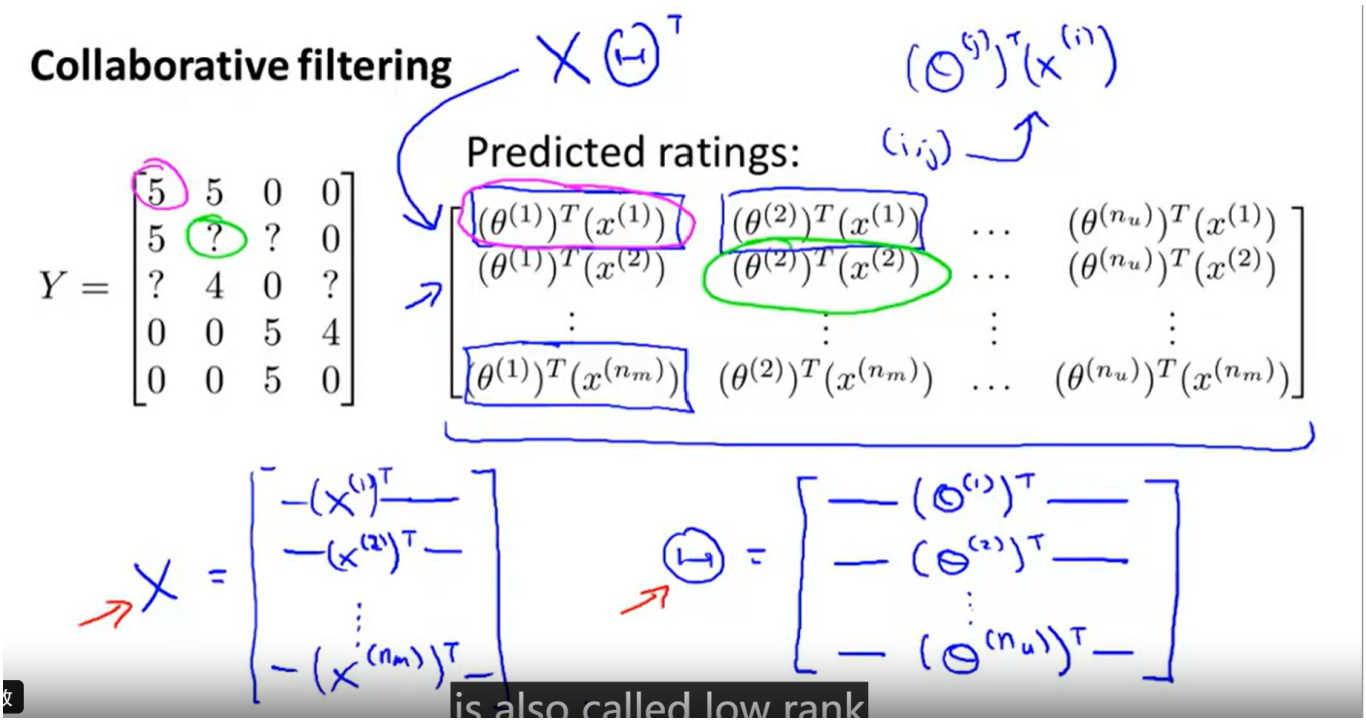
3. predict!

3. For a user with parameters θ and a movie with (learned) features x, predict a star rating of $\theta^T x$.

user J is going to

c. low rank matrix factorization

vectorization: low rank matrix factorization



finding related movies

Finding related movies

For each product i, we learn a feature vector $x^{(i)} \in \mathbb{R}^n$.

→ x_1 = romance, x_2 = action, x_3 = comedy, x_4 = ...

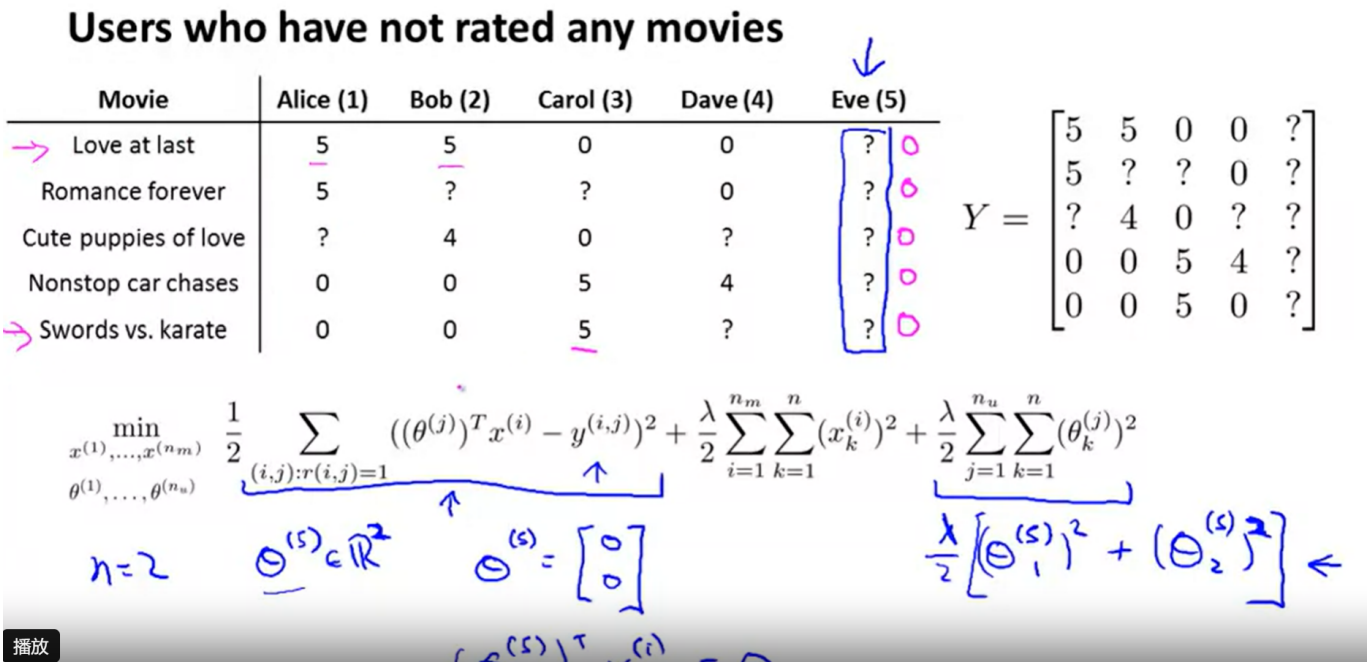
How to find movies j related to movie i?

small $\|x^{(i)} - x^{(j)}\|$ → movie j and i are "similar"

5 most similar movies to movie i:
Find the 5 movies j with the smallest $\|x^{(i)} - x^{(j)}\|$.

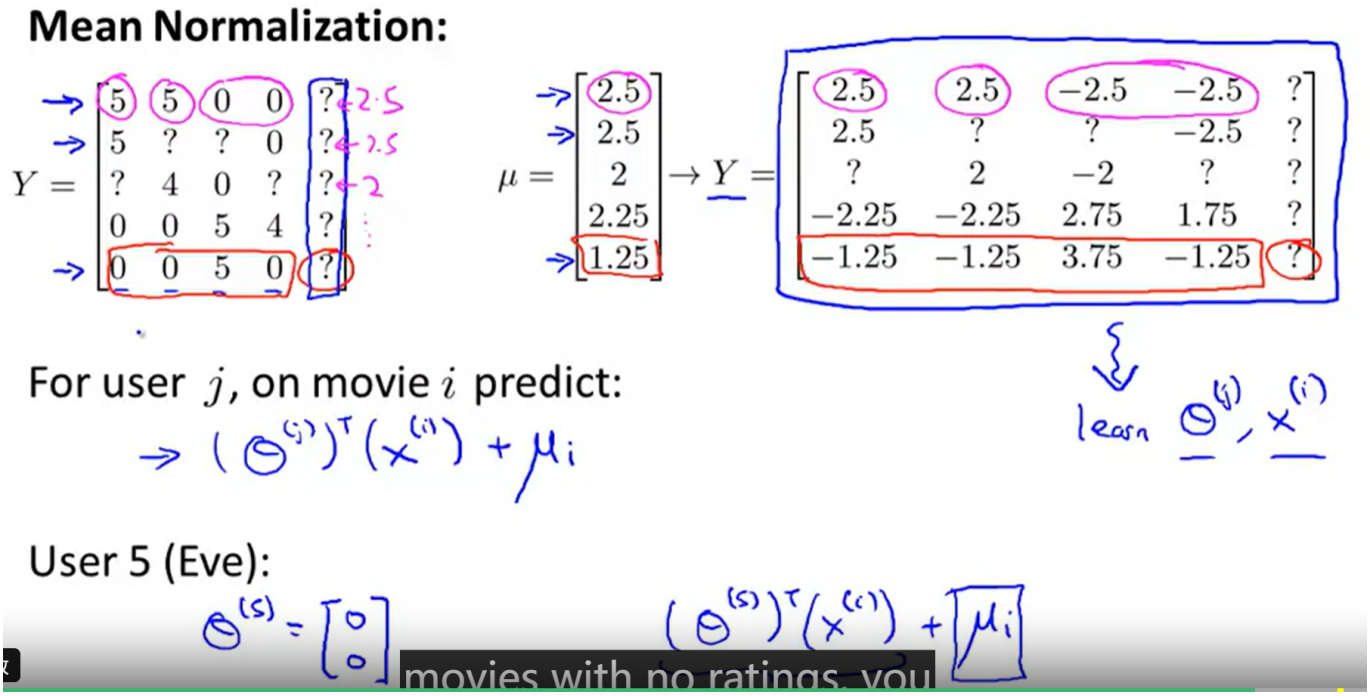
mean normalization

if a user hasn't rated any movies, the θ parameters will eventually go to all zeros because of the regularization term and it is not possible to recommend any movies



what we do:

add the mean rate of each movie to the prediction:



W10

1. Stochastic Gradient Descent(large scale GD)

a. batch GD

just the same as the algorithm in Week 2

Batch gradient descent

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

Repeat {

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

(for every $j = 0, \ldots, n$)

}

b. SGD

algorithm

Stochastic gradient descent

1. Randomly shuffle (reorder) training examples

2. Repeat {

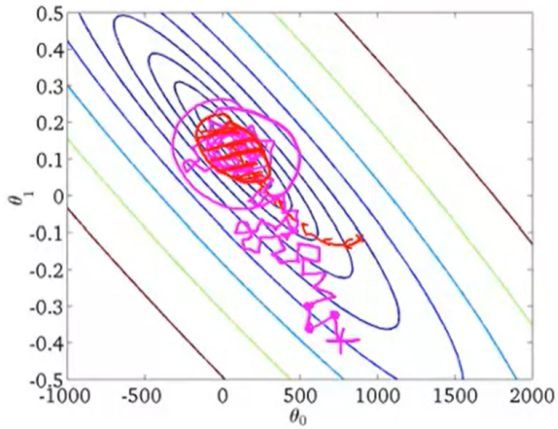
   for $i := 1, \ldots, m$ {

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

   (for every $j = 0, \ldots, n$)

   }

}

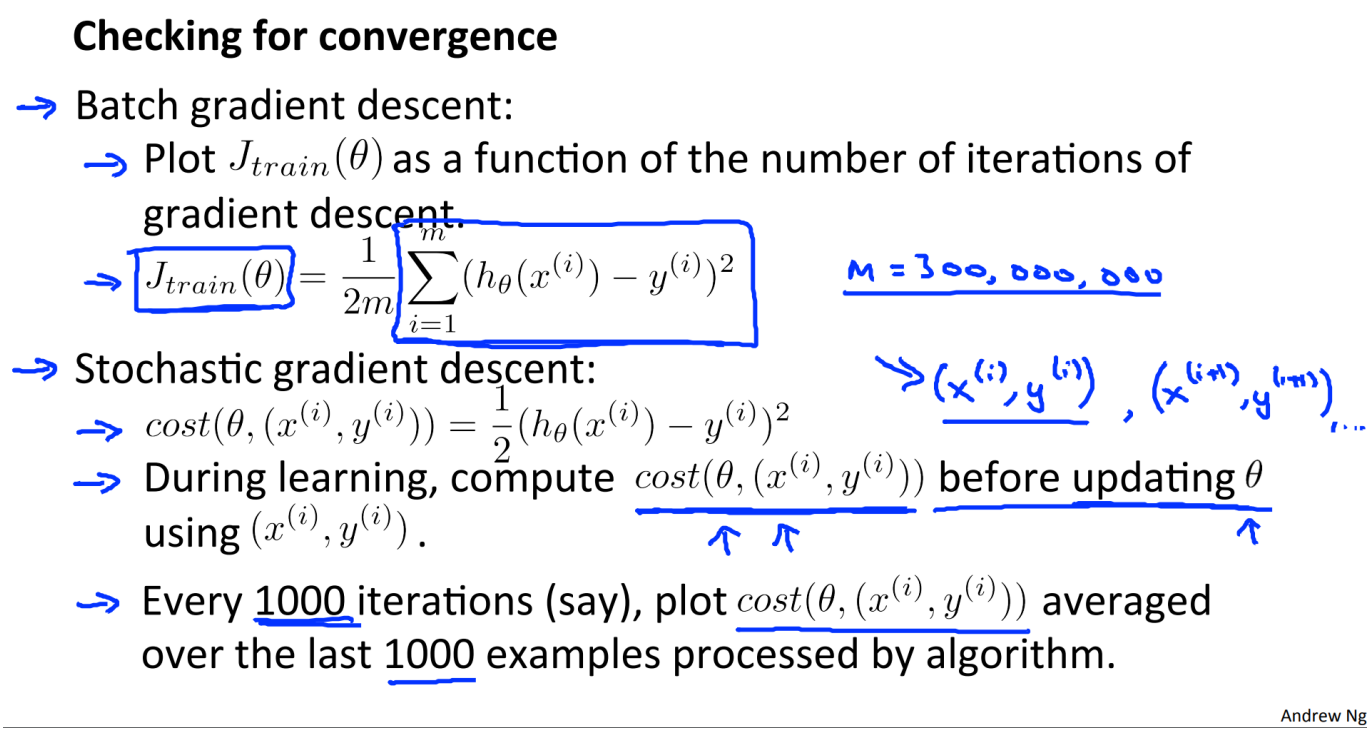


we had this outer loop repeat which says to do this inner loop multiple

Observe one training example every time and the model eventually wanders around the global minimum.

convergence problem

convergence checking:

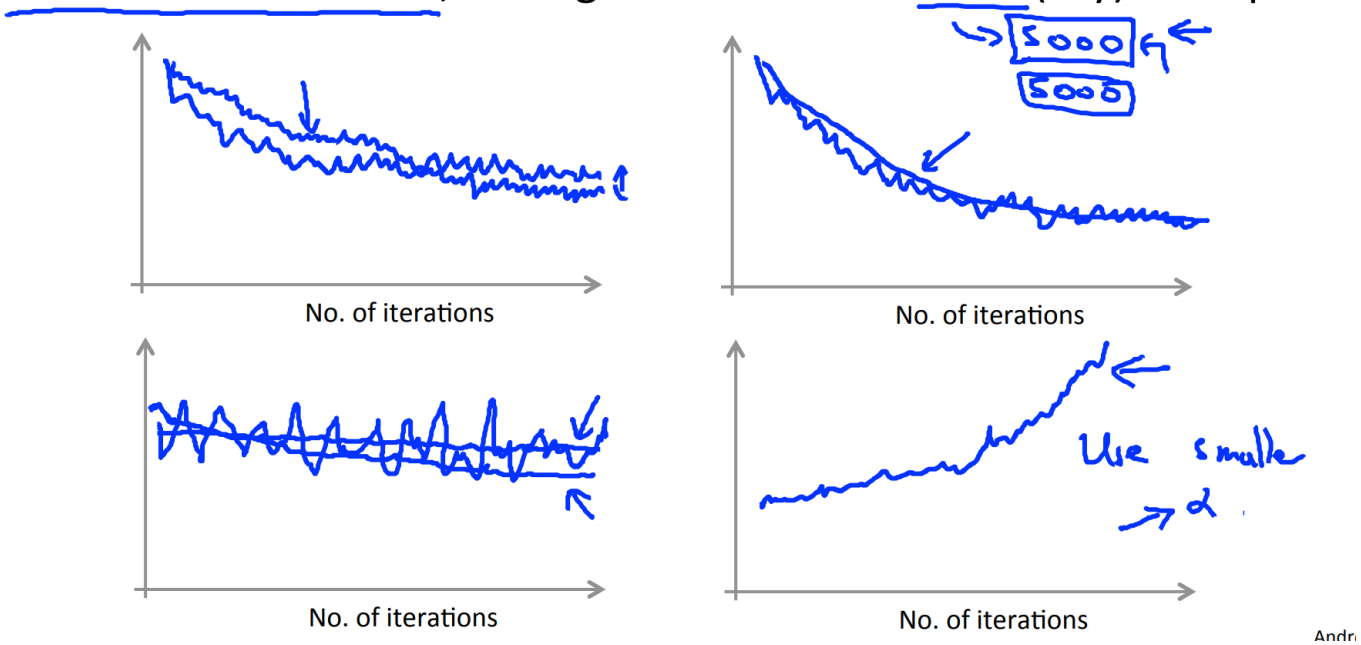


---

4 possible situations:

Checking for convergence

Plot $cost(\theta, (x^{(i)}, y^{(i)}))$, averaged over the last 1000 (say) examples



1 and 2: no bugs

3: may have to change something in the algorithm or set the average number larger(say, 5000)

4: use smaller learning rate α

an approach to make SGD converge to the global minimum

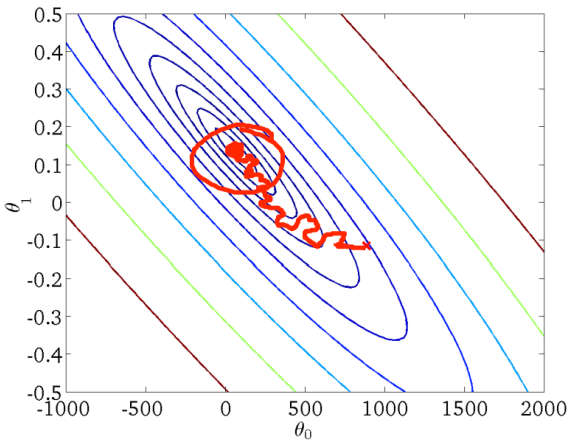
slowly decrease α in each epoch. α can be set to $\alpha = \dfrac{\text{const1}}{\text{itertion number} + \text{const2}}$

the meandering will be smaller and smaller while approaching the global minimum:

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 $i := 1, \ldots, m$ {
   $\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$
   (for $j = 0, \ldots, n$)
   }
   }

Learning rate α is typically held constant. Can slowly decrease α over time if we want θ to converge. (E.g. $\alpha = \frac{\text{const1}}{\text{iterationNumber} + \text{const2}}$) $\alpha \to 0$

c. mini-batch GD

idea

Mini-batch gradient descent

→ Batch gradient descent: Use all m examples in each iteration

→ Stochastic gradient descent: Use 1 example in each iteration

Mini-batch gradient descent: Use b examples in each iteration

b = mini-batch size. b = 10. 2 - 100

algorithm

Mini-batch gradient descent

Say $b = 10, m = 1000$.

Repeat {

for $i = 1, 11, 21, 31, \ldots, 991$ {

$$\theta_j := \theta_j - \alpha \frac{1}{10} \sum_{k=i}^{i+9} (h_\theta(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

(for every $j = 0, \ldots, n$)

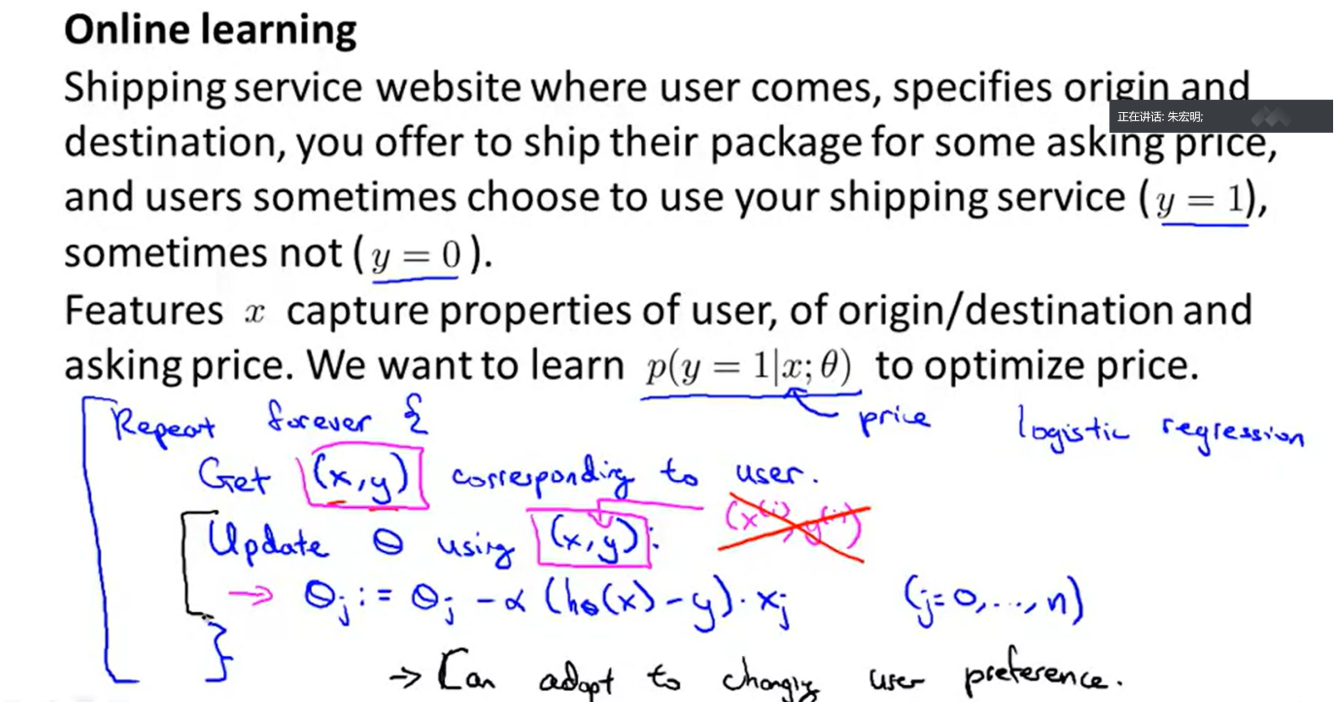
}

}

2. Advanced Topics

a. online learning

learn from a data stream. In each epoch we just consider one training example:



another example: information retrieval

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 = 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. $y = 0$ otherwise.

→ Learn $p(y = 1|x; \theta)$. predicted CTR

→ Use to show user the 10 phones they're most likely to click on.

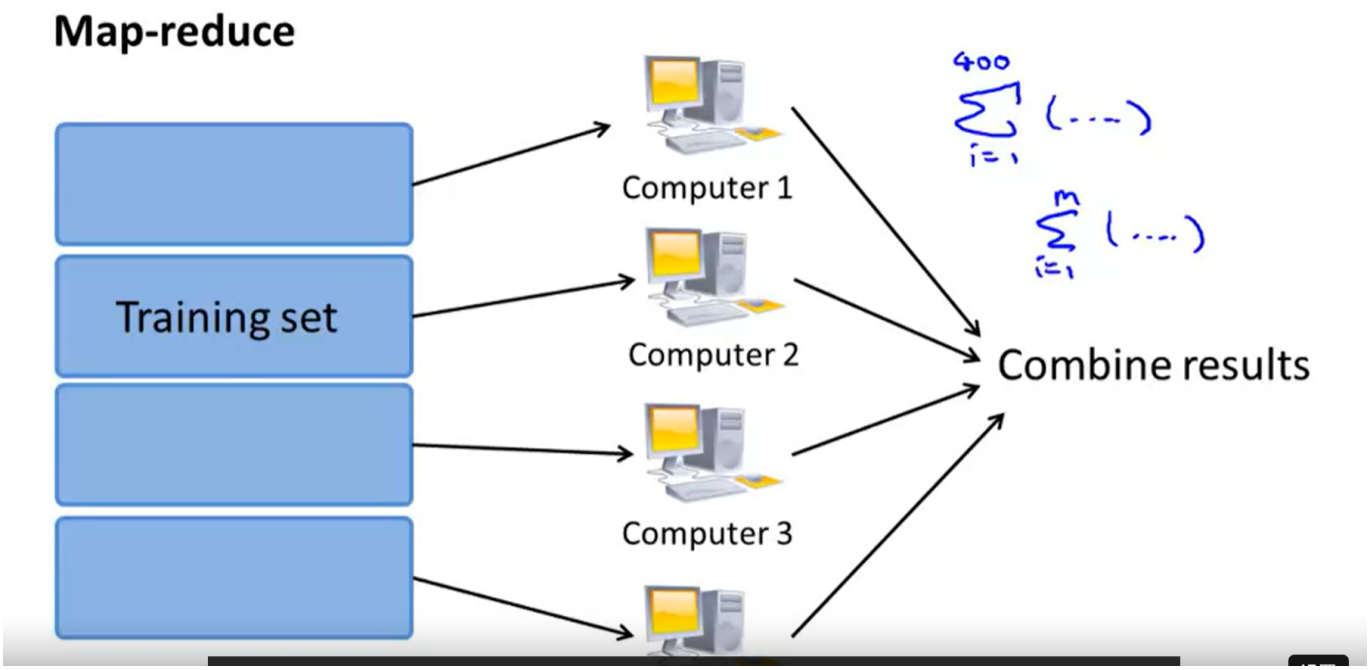
Other examples: Choosing special offers to show user; customized selection of news articles; product recommendation; ...

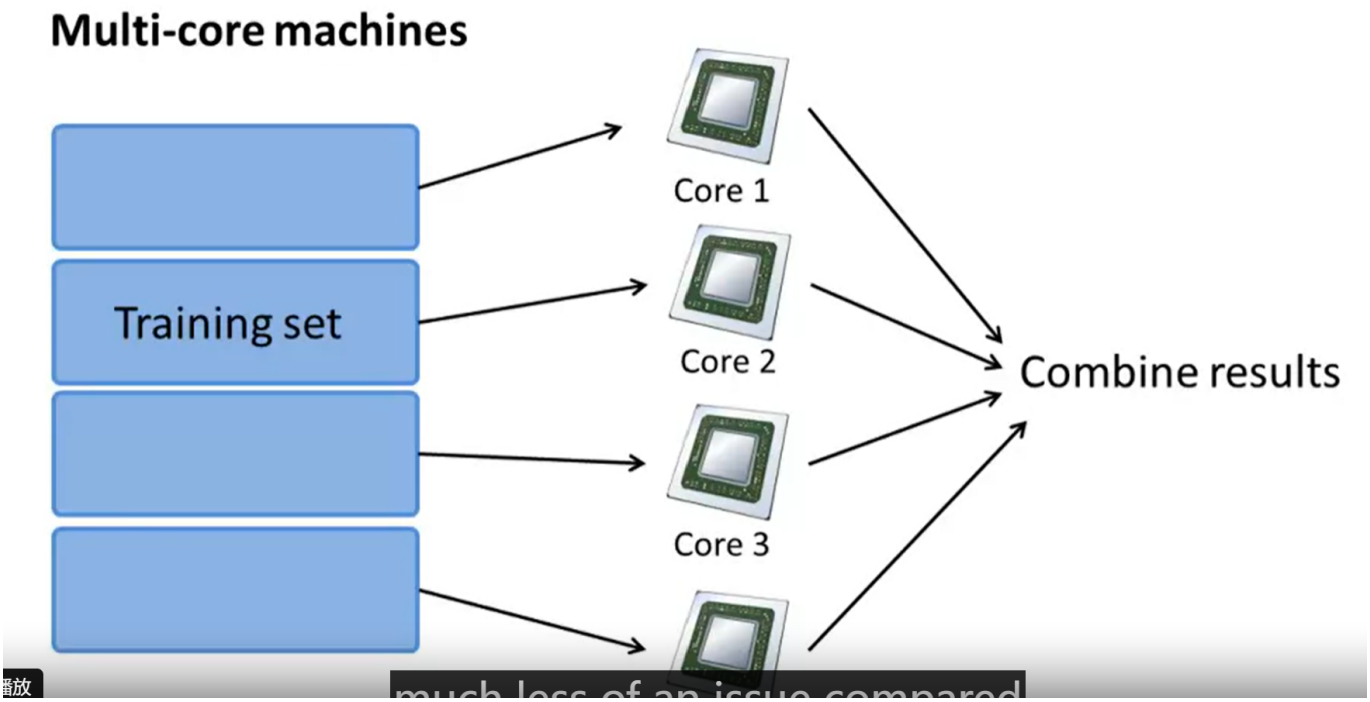
Maybe, you can run your

b. map reduce & data parallelism

map reduce idea:

compute the gradient on different computers(or different cores) and combine them in a master server



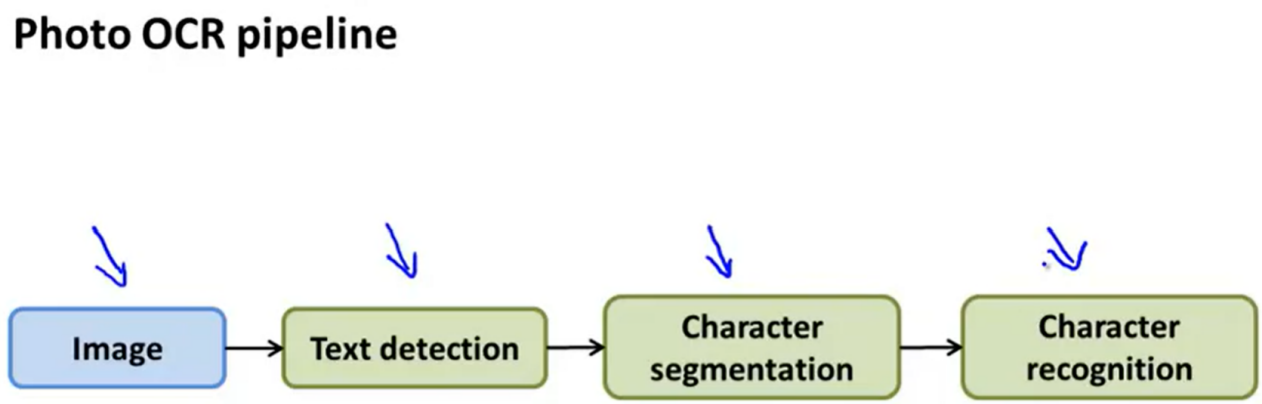


W11

Case Study: Photo OCR

a. pipeline

1. text detection
2. character segmentation
3. character classification
4. (optional) spelling correction



b. sliding window technique

pedestrian detection problem

1. determine the size of the window
2. slide the window from the left top to the right bottom to detect whether there is a pedestrian
3. choose a larger window size(resize it to the image size of the training set, say 80*40) and repeat 1,2.

text detection problem

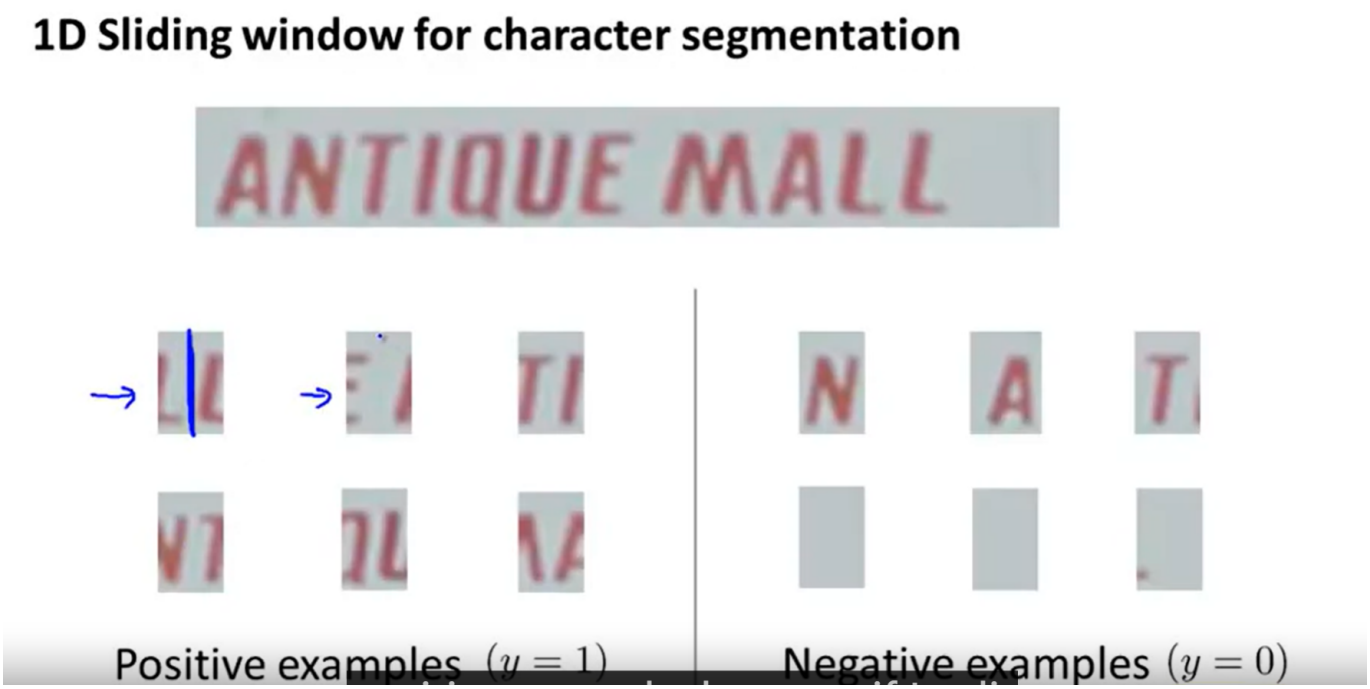
1. use slide window to detect all the text regions in an image:

2. the result may be like this

3. apply expansion operator:

character segmentation

train a classifier to determine whether there is **a split between characters**



applying SW on this:

c. getting data for the system

1. artificial data synthesis

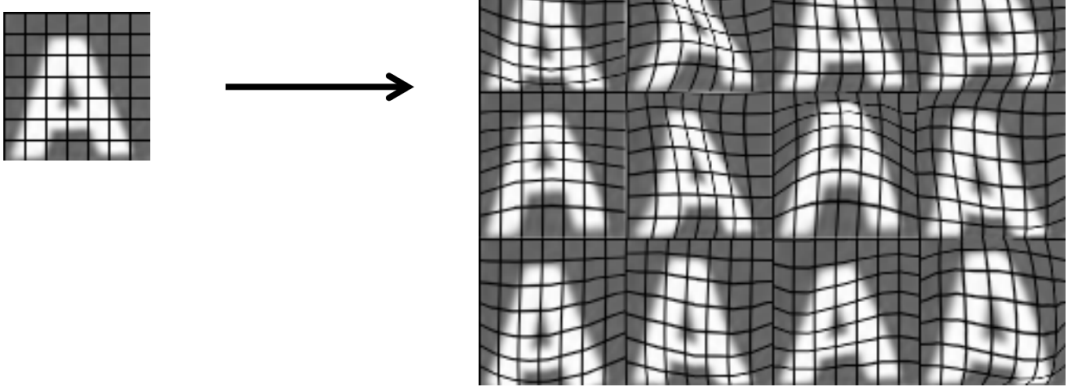
Artificial data synthesis for photo OCR

Real data

Synthetic data

2. introducing distortions:

Synthesizing data by introducing distortions



advice on getting more data

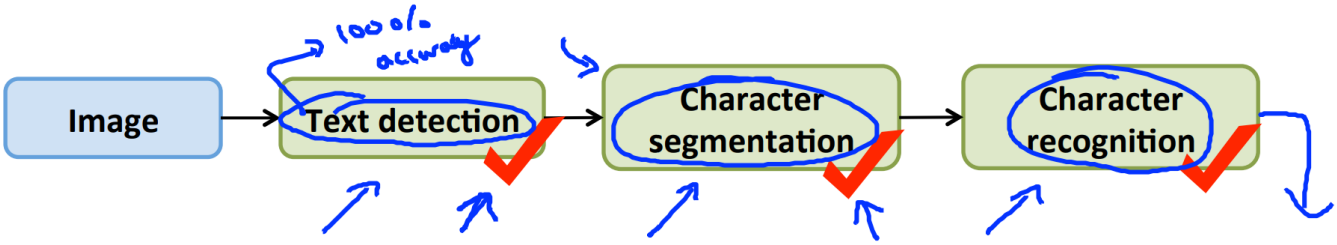
Discussion on getting more data

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)

d. ceiling analysis(上限分析)

examples:

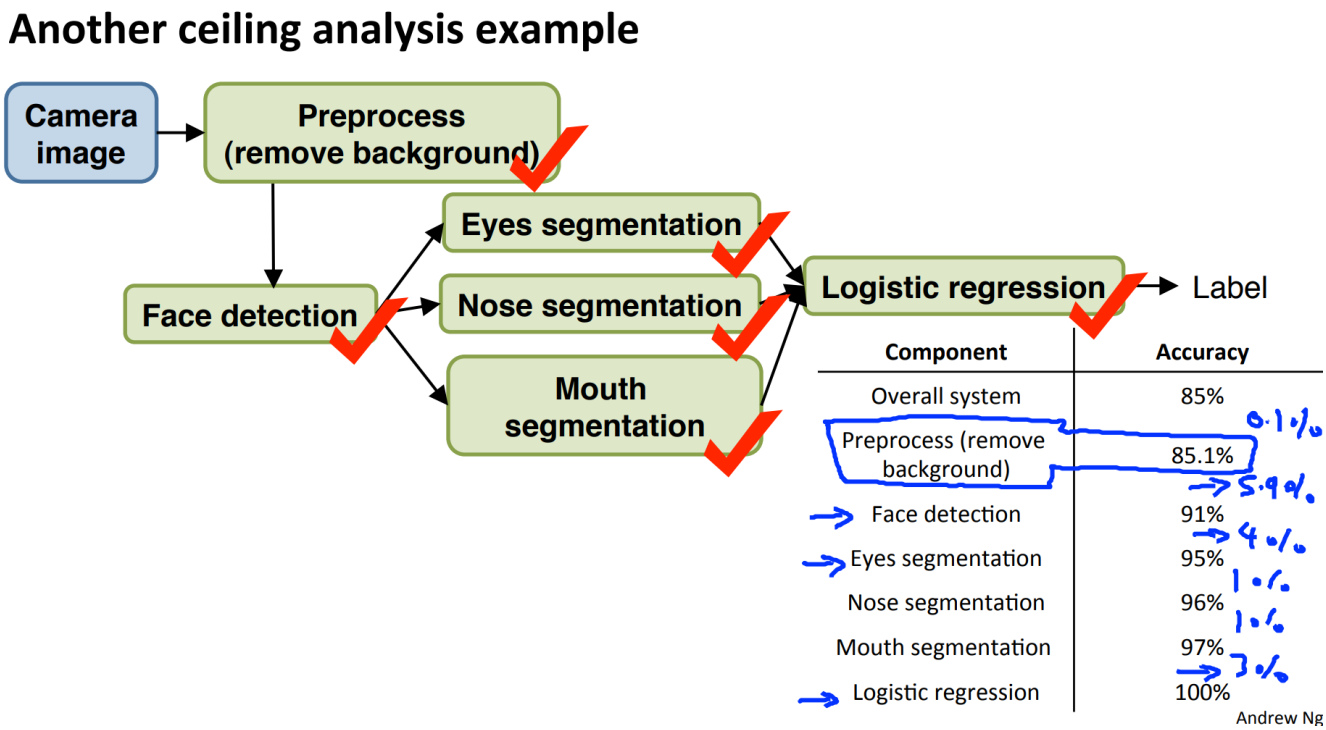
Estimating the errors due to each component (ceiling analysis)



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%

Andrew



Manually make each step of the pipeline to the ground truth and check how much the accuracy improved.

End of Course: Thank you from Andrew Ng