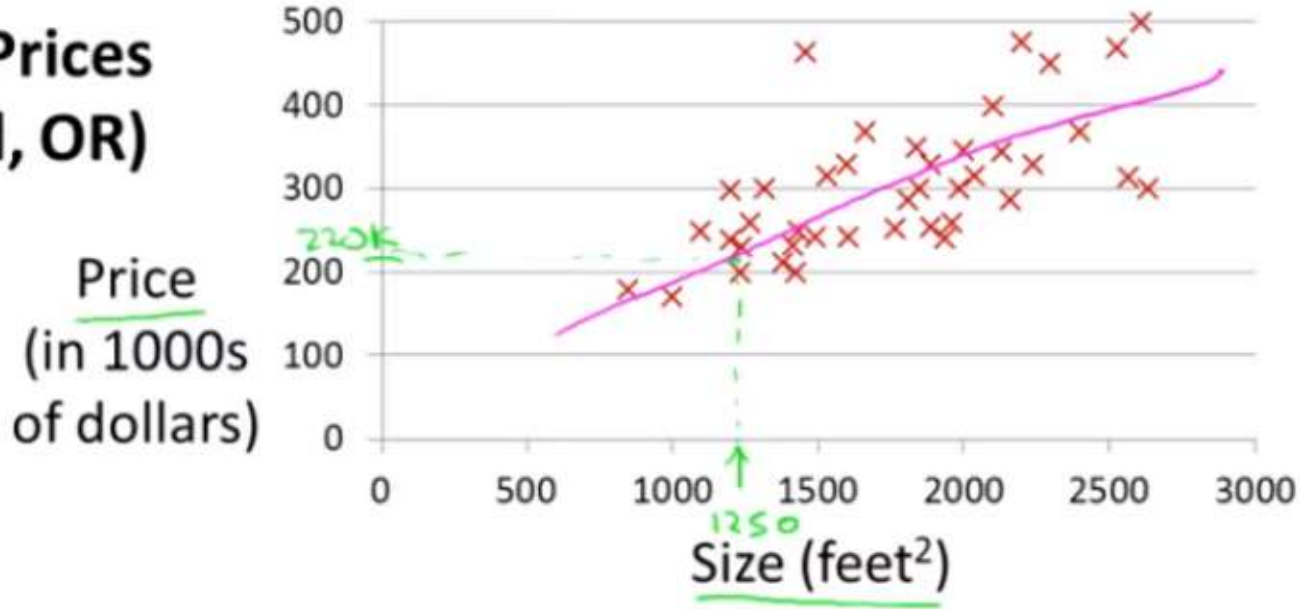
Housing Prices (Portland, OR)



Supervised Learning

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

Regression Problem

Predict real-valued output

Classification: Discrete-valued output

Training set of housing prices (Portland, OR)

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

m = 47

Notation:

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

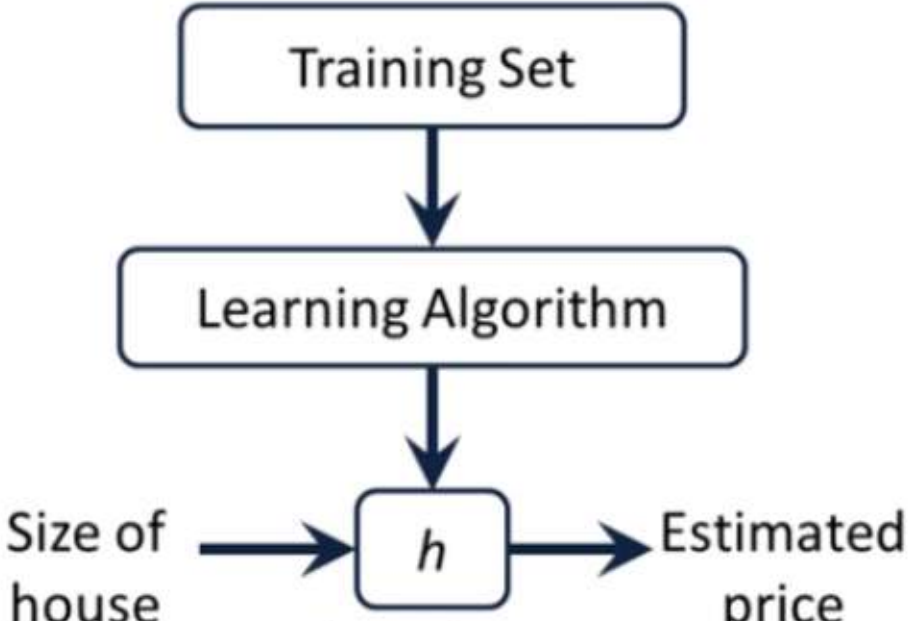
(x, y) - one training example

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

$x^{(1)} = 2104$

$x^{(2)} = 1416$

$y^{(1)} = 460$

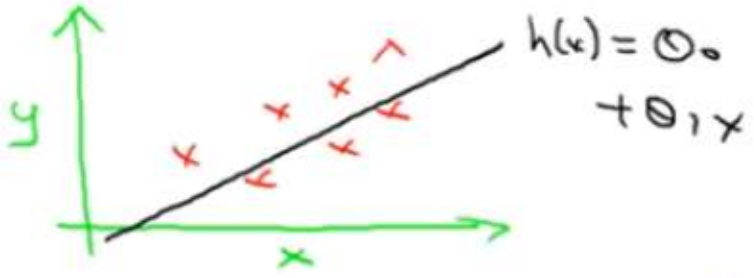


h maps from x's to y's.

How do we represent *h* ?

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

Shorthand: $h(x)$



Linear regression with one variable. (x)

Univariate linear regression.

└ one variable

Training Set

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

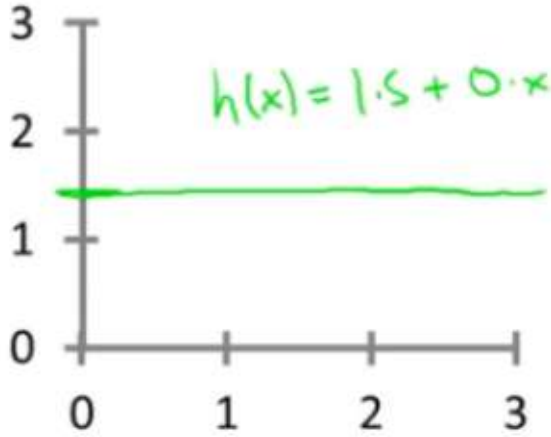
$m = 47$

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

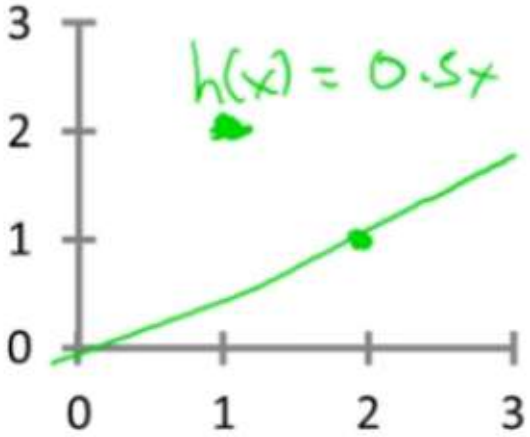
θ_i's: Parameters

How to choose θ_i's ?

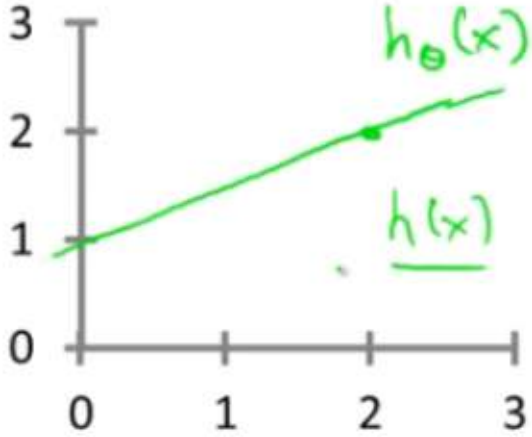
$h_\theta(x) = \theta_0 + \theta_1 x$



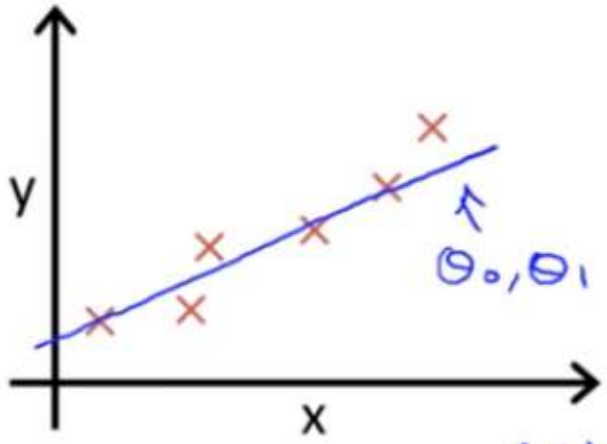
→ $\theta_0 = 1.5$
→ $\theta_1 = 0$



→ $\theta_0 = 0$
→ $\theta_1 = 0.5$



→ $\theta_0 = 1$
→ $\theta_1 = 0.5$



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

Idea: Choose θ_0, θ_1 so that $h_\theta(x)$ is close to y for our training examples (x, y)

x, y

#training examples

minimize θ_0, θ_1 $\frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$

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

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

minimize θ_0, θ_1 $J(\theta_0, \theta_1)$

Cost function

Squared error function

Hypothesis:

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

Parameters:

θ_0, θ_1

Cost Function:

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

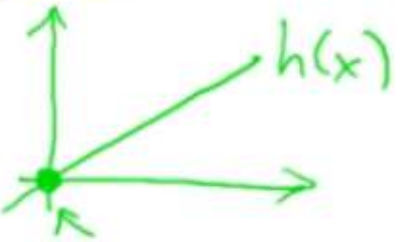
Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

Simplified

$$h_\theta(x) = \theta_1 x$$

$\theta_0 = 0$

θ_1



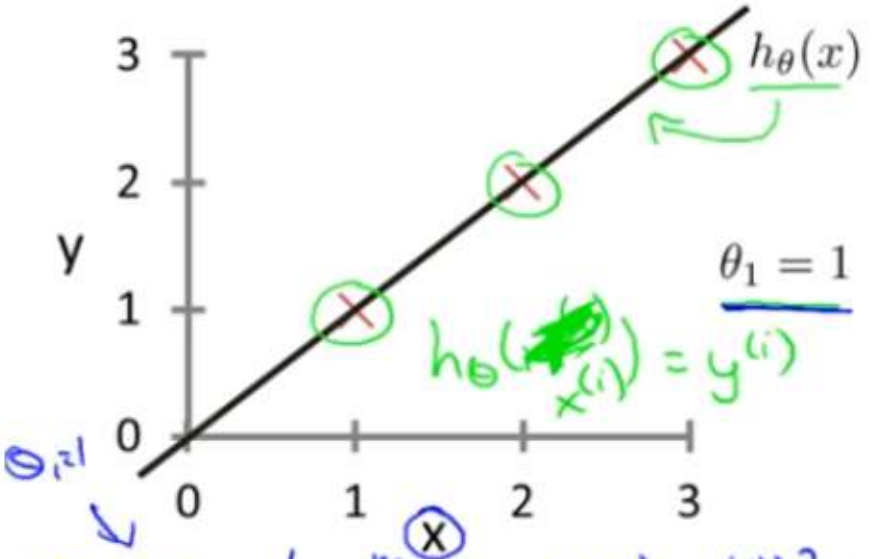
$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

$\theta, x^{(i)}$

$\underset{\theta_1}{\text{minimize}}\ J(\theta_1)$

$h_\theta(x)$

(for fixed θ_1, this is a function of x)



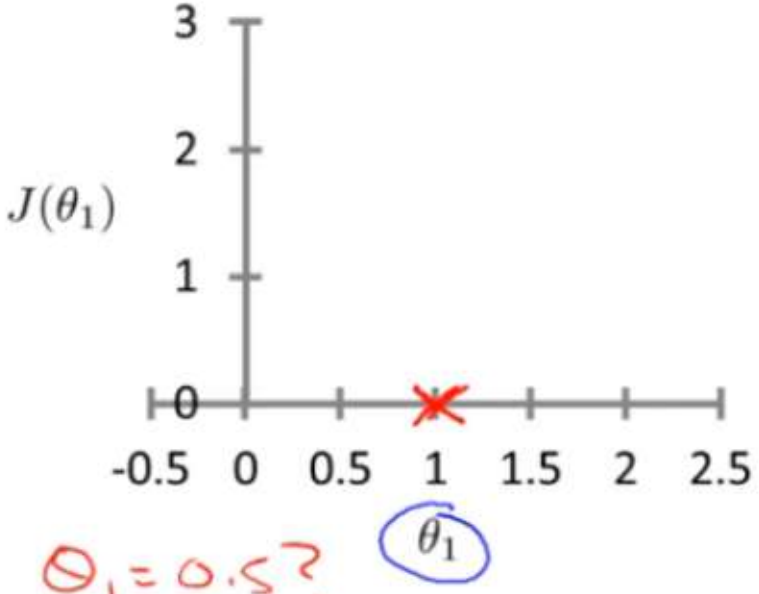
$\theta_1 = 1$

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

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

$J(\theta_1)$

(function of the parameter θ_1)

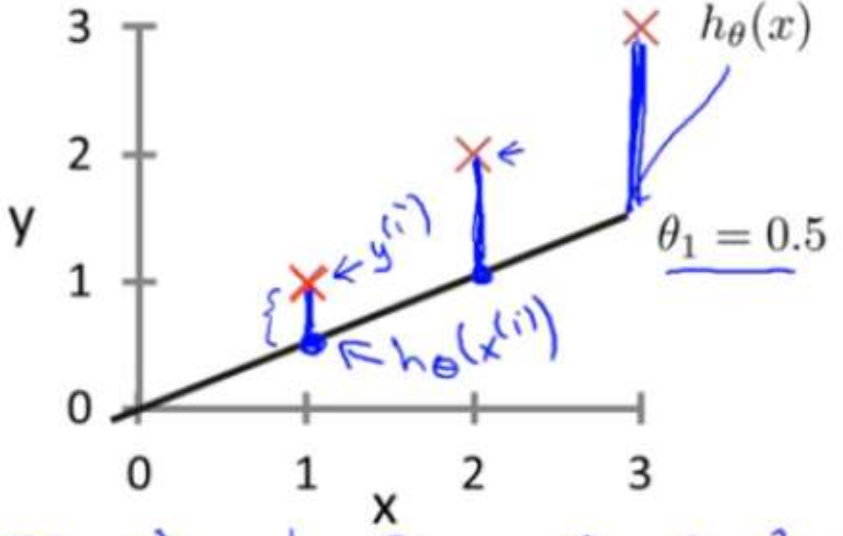


$\theta_1 = 0.5$?

$J(1) = 0$

$h_\theta(x)$

(for fixed θ_1, this is a function of x)

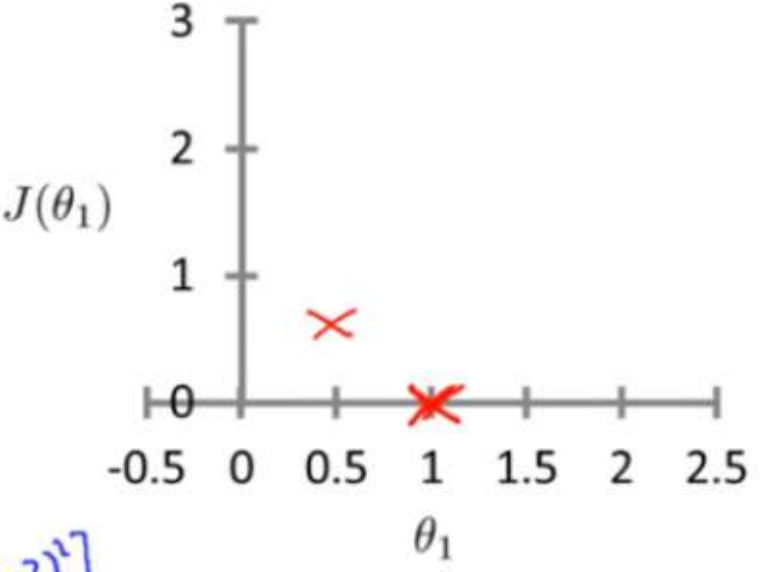


$J(0.5) = \frac{1}{2m}\left[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2\right]$

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

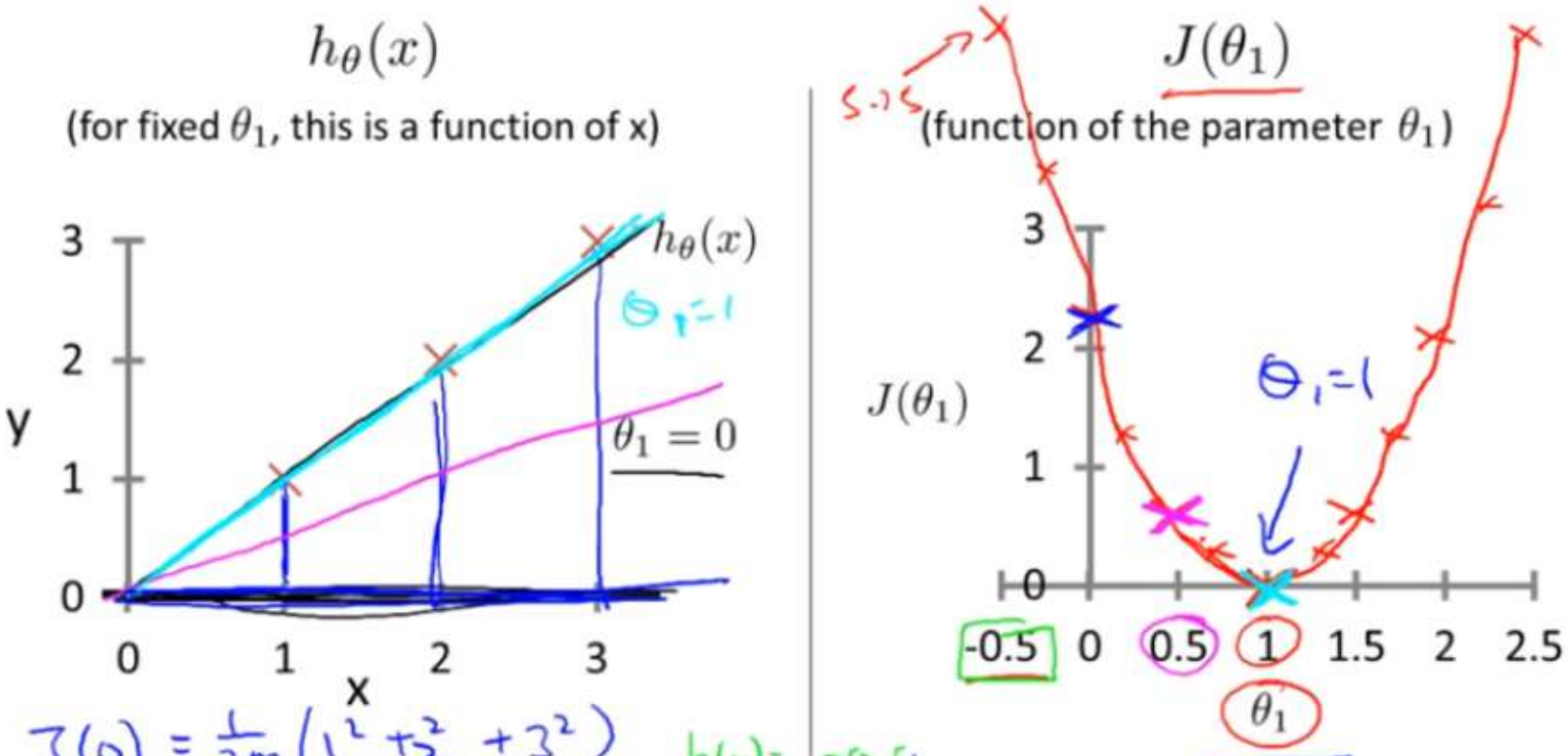
$J(\theta_1)$

(function of the parameter θ_1)



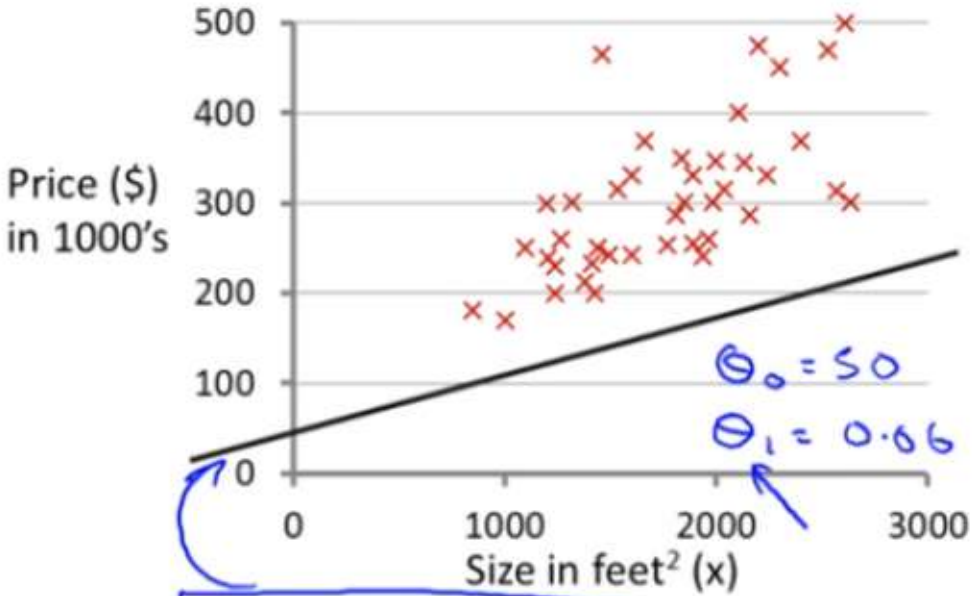
$\theta_1 = 0$?

$J(0) = ?$


$h_\theta(x)$
(for fixed θ_1, this is a function of x)
3
2
1
0
y
0
1
2
3
x
$h_\theta(x)$
$\theta_1 = 1$
$\theta_1 = 0$
$J(\theta_1)$
(function of the parameter θ_1)
5.25
3
2
1
0
$J(\theta_1)$
$\theta_1 = 1$
-0.5
0
0.5
1
1.5
2
2.5
θ_1
$J(0) = \frac{1}{2m}(1^2 + 2^2 + 3^2)$
$= \frac{1}{6} \cdot 14 \approx 2.3$
$h(x) = -0.5x$
Minimize $J(\theta_1)$
θ_1

$h_\theta(x)$

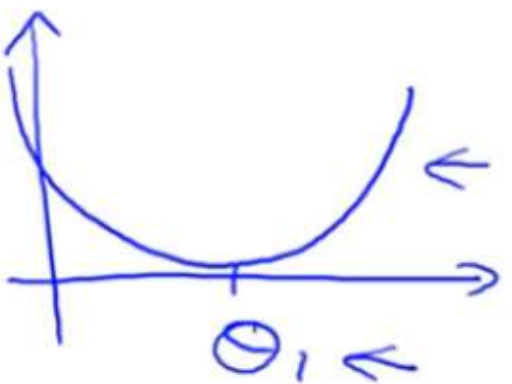
(for fixed θ_0, θ_1, this is a function of x)



$$h_\theta(x) = 50 + 0.06x$$

$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



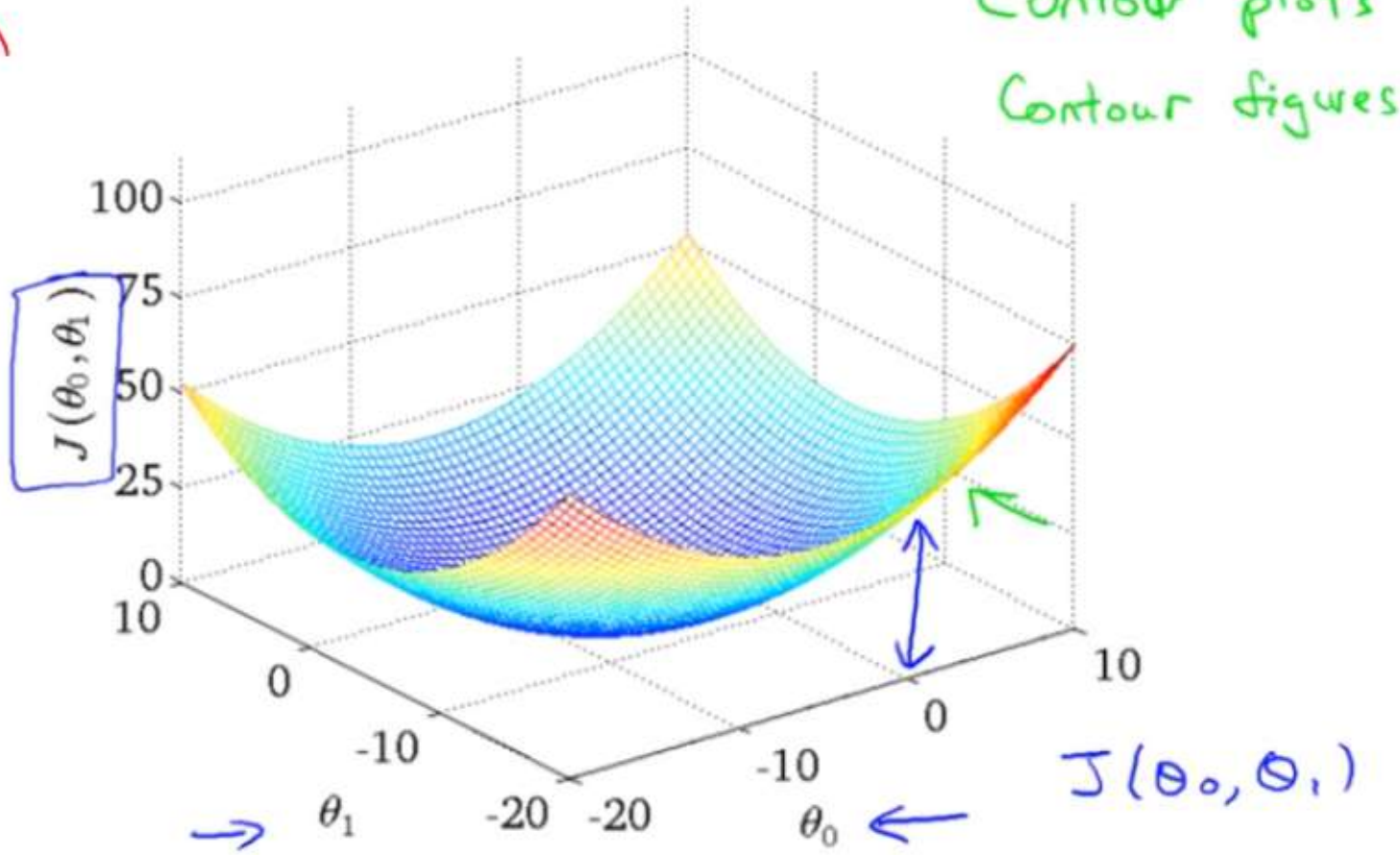
θ_0, θ_1

→ graph on next PAGE

Graph

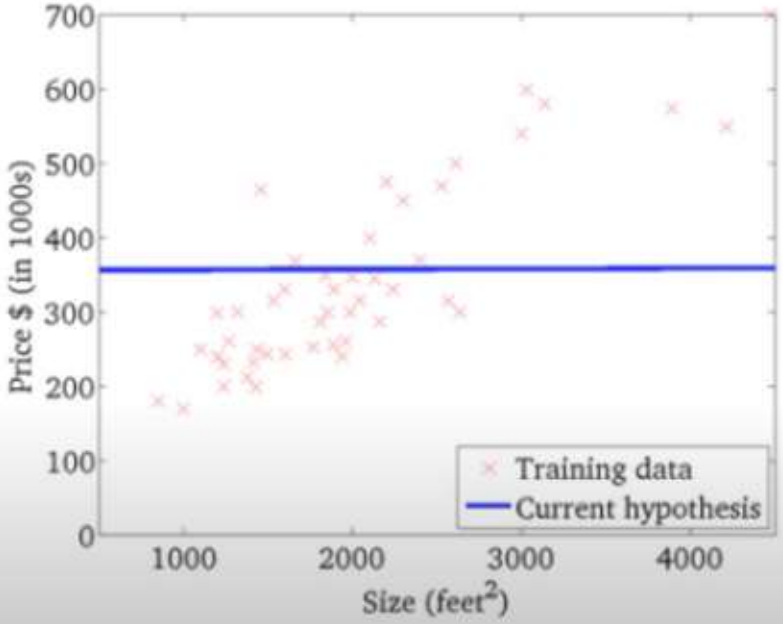
Contour plots

Contour figures -



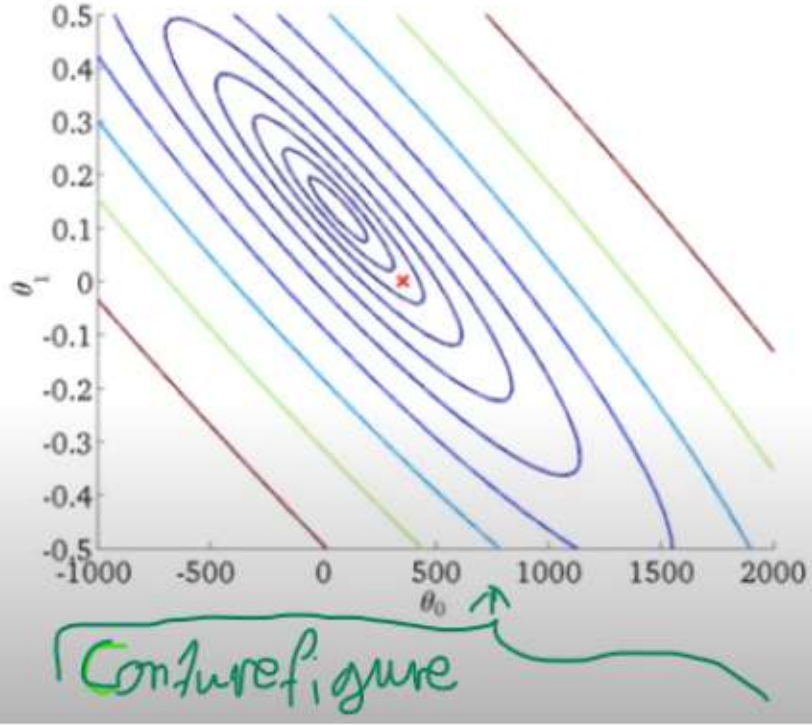
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

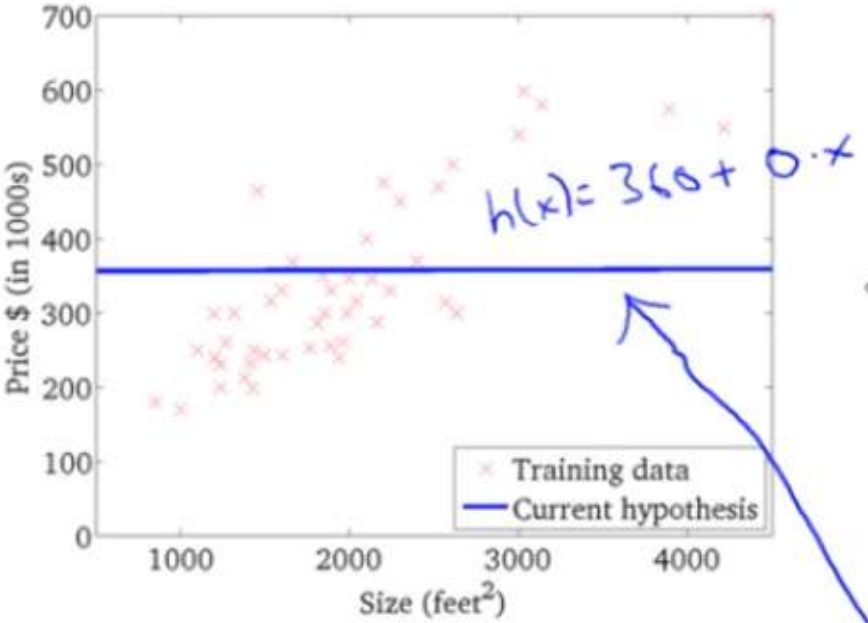


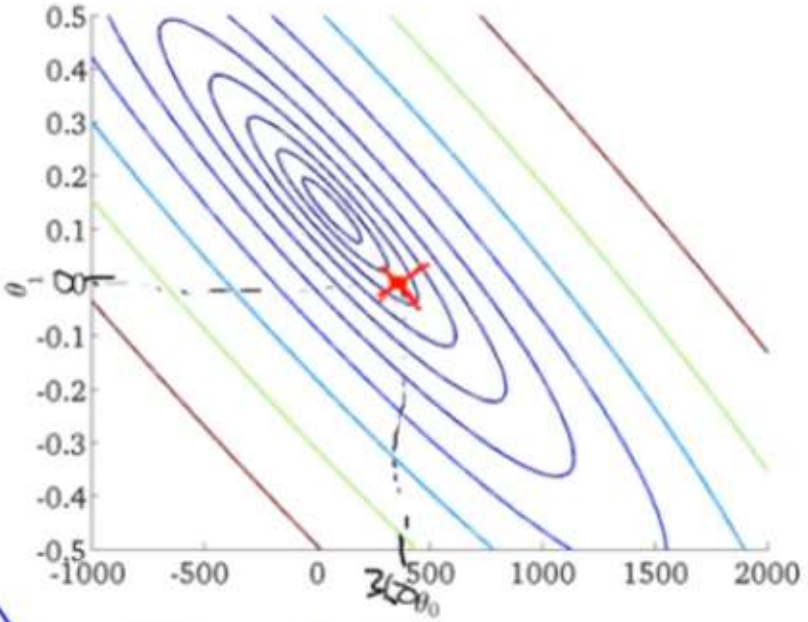
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)

$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



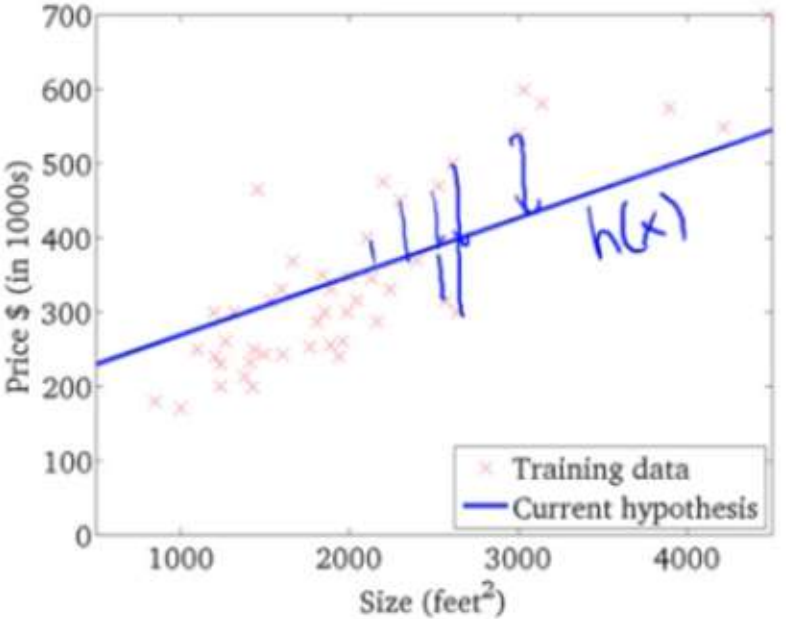


$\theta_0 = 360$

$\theta_1 = 0$

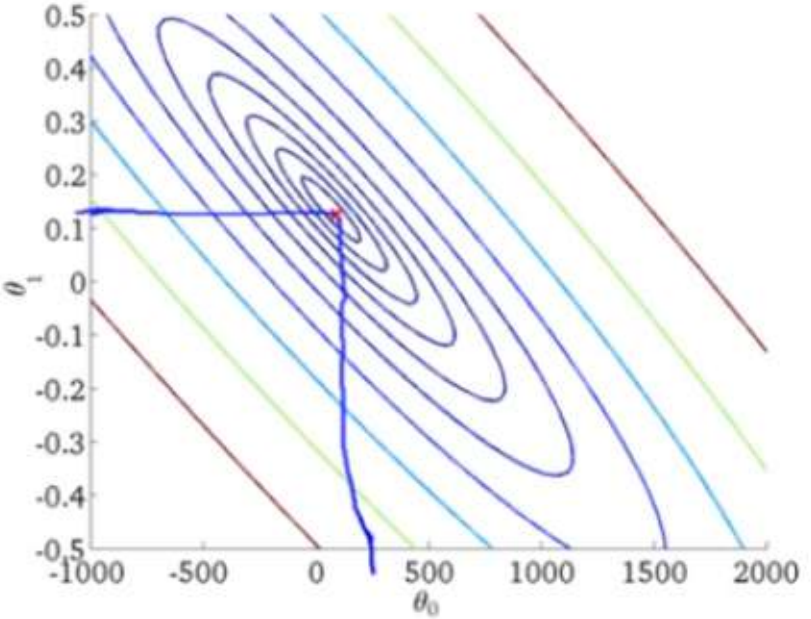
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

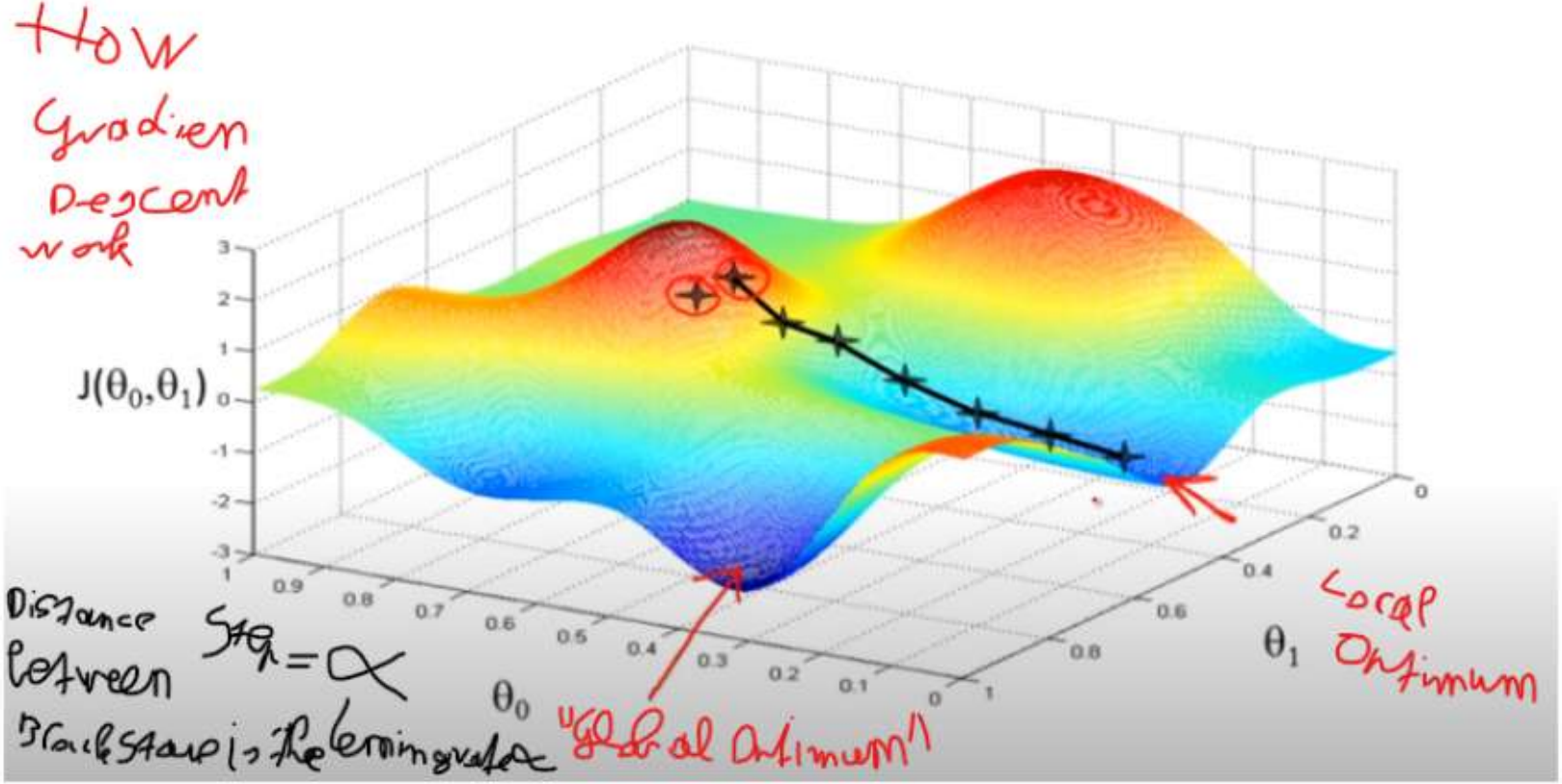


Have some function $J(\theta_0, \theta_1)$ $J(\theta_0, \theta_1, \theta_2, \ldots, \theta_n)$

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

Outline:

- Start with some θ_0, θ_1 (Say $\theta_0 = 0, \theta_1 = 0$)
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum


How Gradient Descent work
$J(\theta_0, \theta_1)$
θ_0
θ_1
Distance between Step = α
Backstep is the derivative
"Global Optimum"
Local Optimum

Gradient descent algorithm

Assignment: $a := b$, $a := a+1$

Truth assertion: $a = b$, $a = a+1$ ✗

θ_0, θ_1

repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (\text{for } j = 0 \text{ and } j = 1)$$

}

learning rate

Simultaneously update θ_0 and θ_1

Correct: Simultaneous update

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

Incorrect:

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

Gradient descent algorithm

repeat until convergence {

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

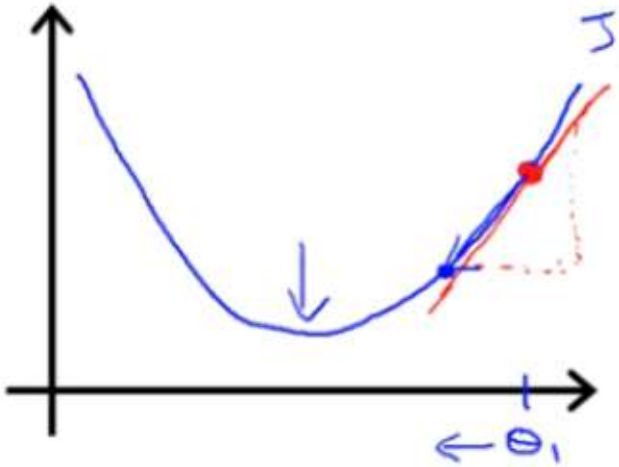
(simultaneously update $j = 0$ and $j = 1$)

}

learning rate

derivative

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

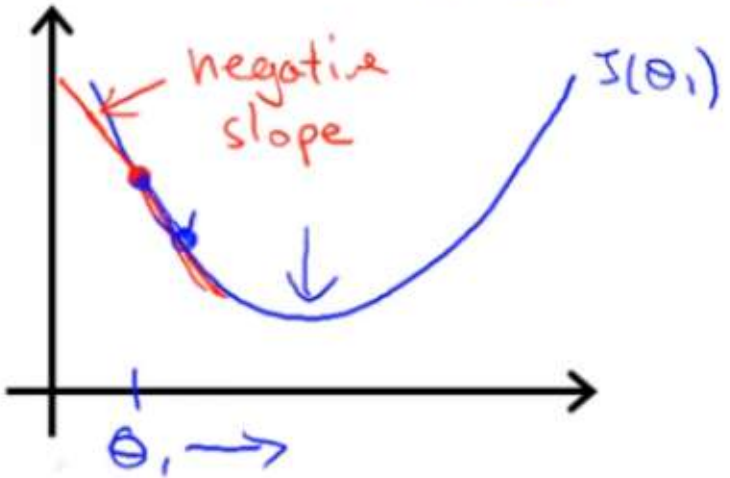


$(\theta_1 \in \mathbb{R})$

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

$\frac{d}{d\theta_1} J(\theta_1) \geq 0$

$\theta_1 := \theta_1 - \alpha \cdot$ (positive number)



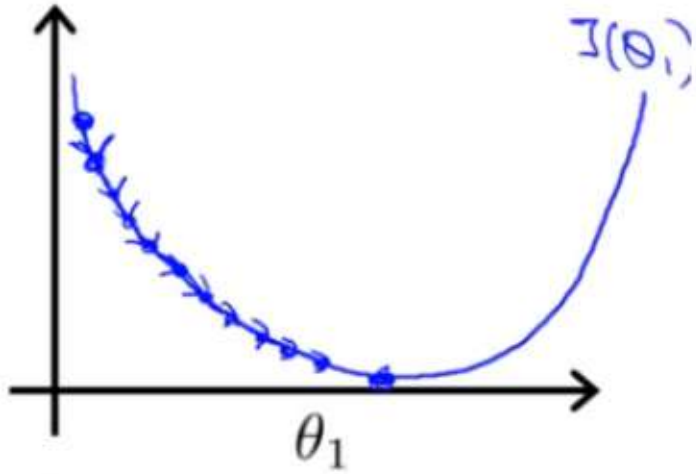
$\frac{d}{d\theta_1} J(\theta_1) \leq 0$

$\theta_1 := \theta_1 - \alpha$ (negative number)

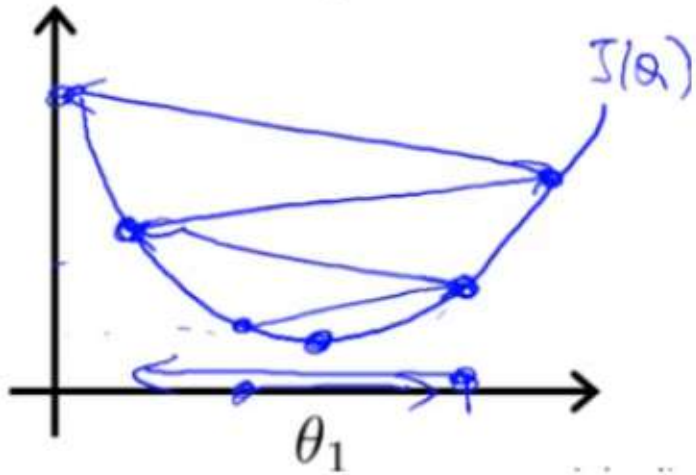
Andrew N

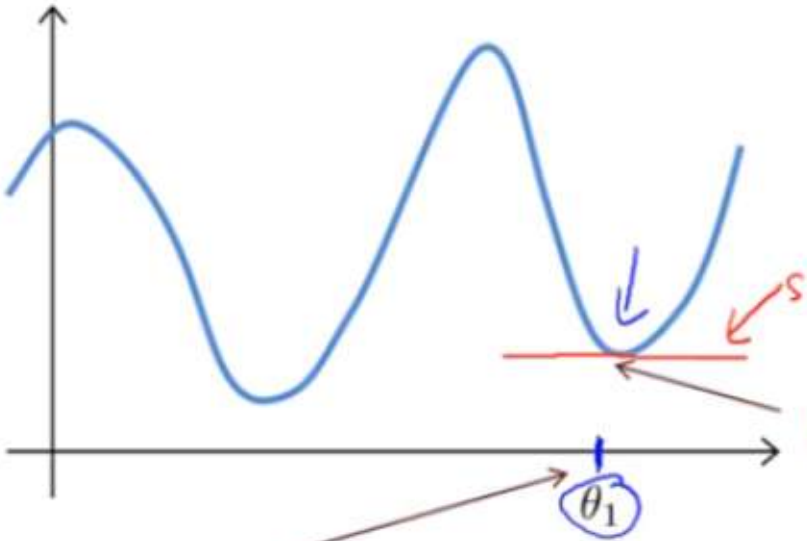
$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.



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





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

$= 0$

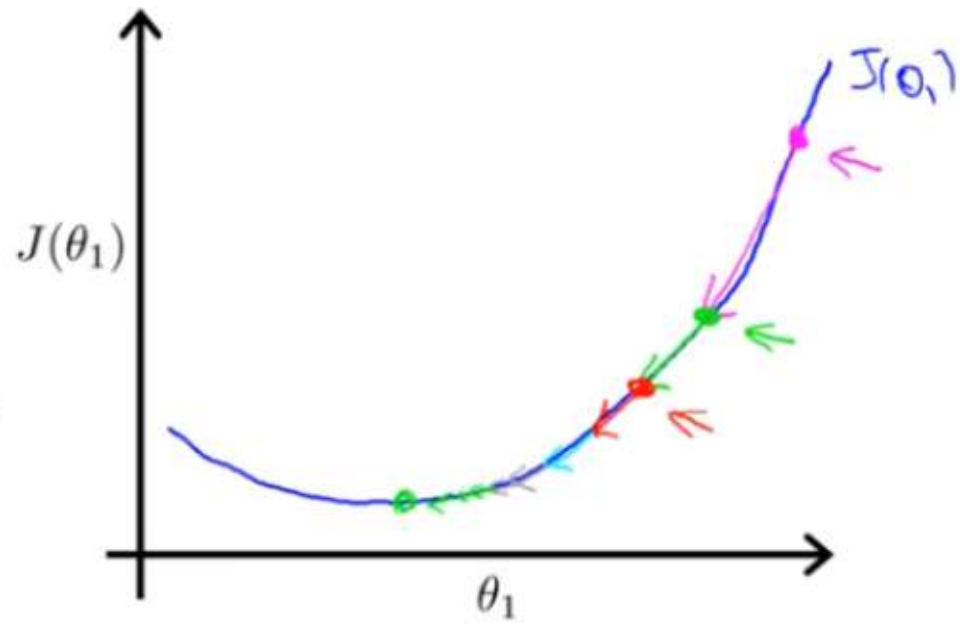
$\theta_1 := \theta_1 - \alpha \cdot 0$

$\theta_1 := \theta_1$

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

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

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



Gradient descent algorithm

repeat until convergence {

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

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

}

Linear Regression Model

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

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

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

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

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

θ_0 $j = 0$: $\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$

θ_1 $j = 1$: $\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$

Gradient descent algorithm

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

repeat until convergence {

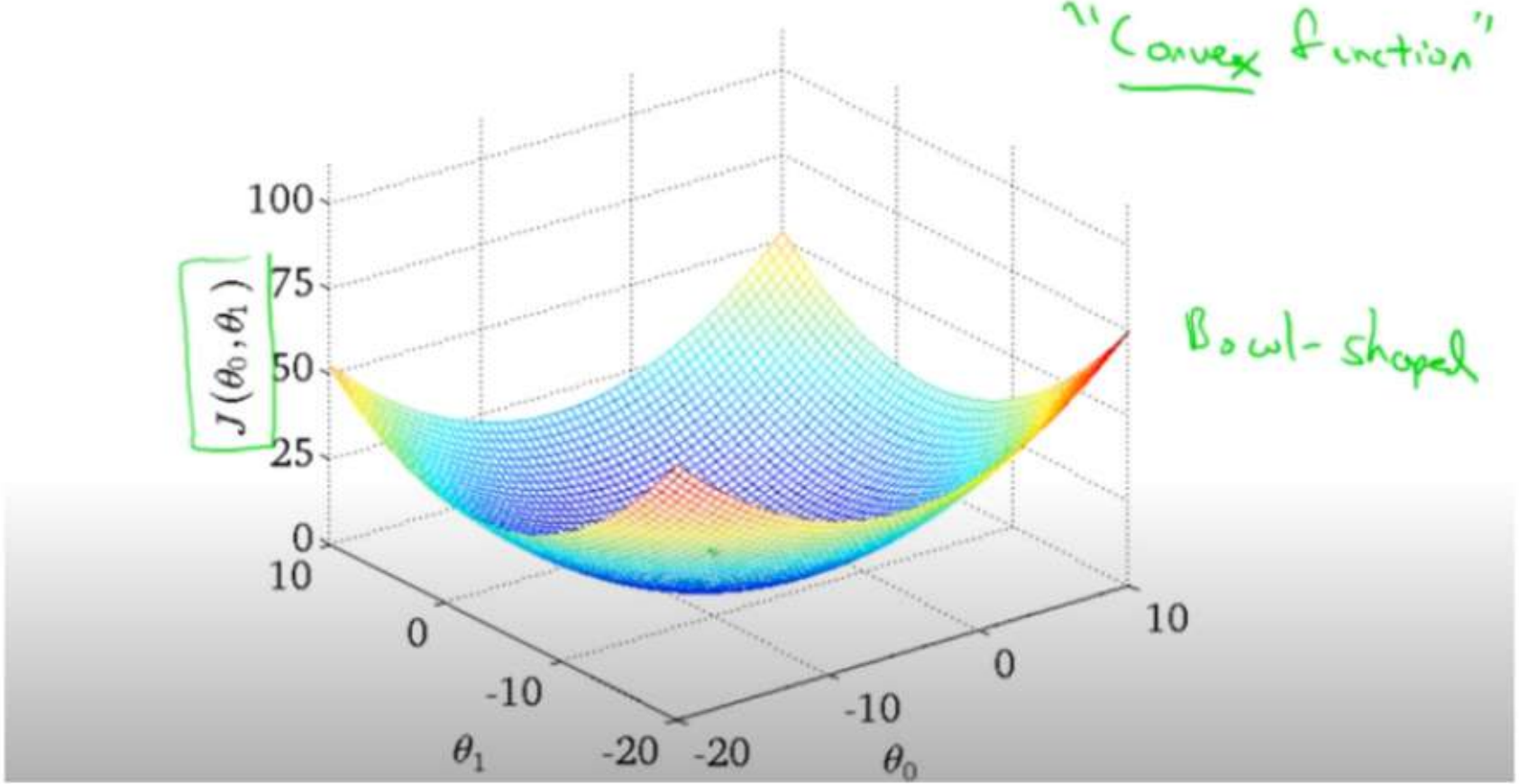
$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

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

}

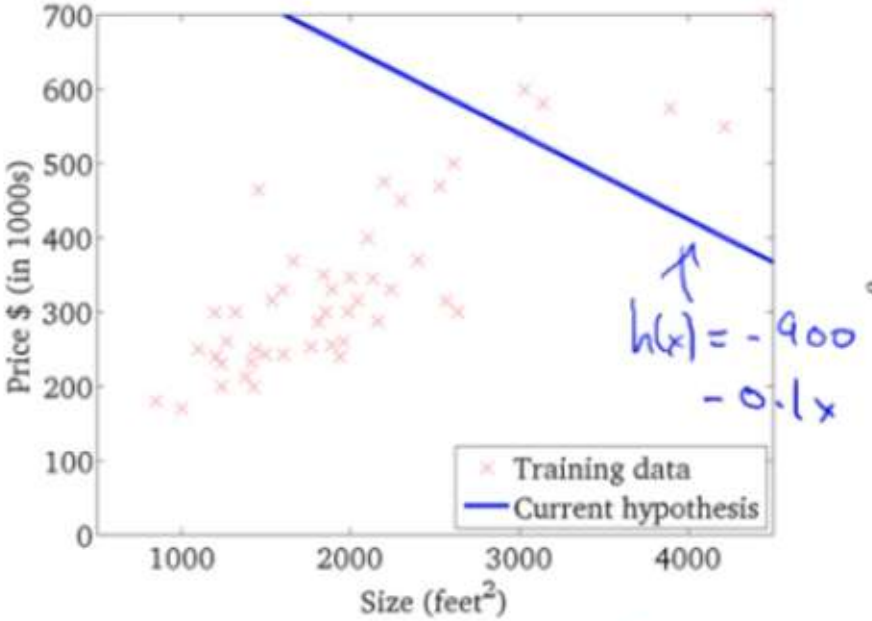
update θ_0 and θ_1 simultaneously

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


"Convex function"
Bowl-shaped
$J(\theta_0, \theta_1)$
100
75
50
25
0
10
0
-10
-20
θ_1
-20
-10
0
10
θ_0

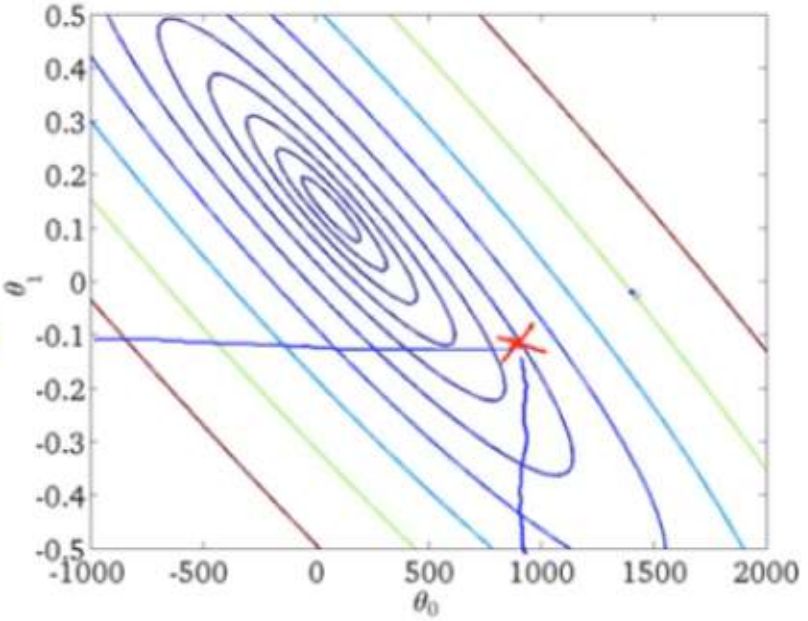
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

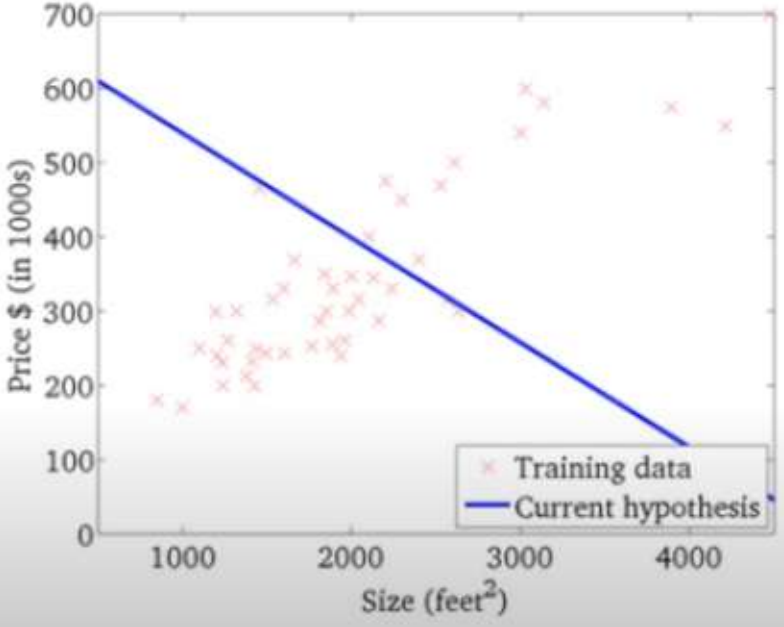
(function of the parameters θ_0, θ_1)



FROM THIS TO... (NEXT SLIDE)

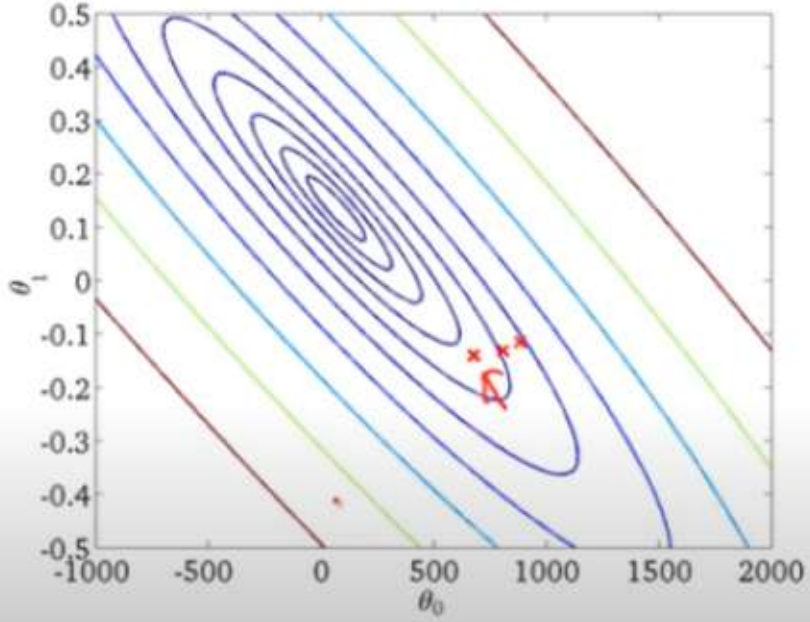
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



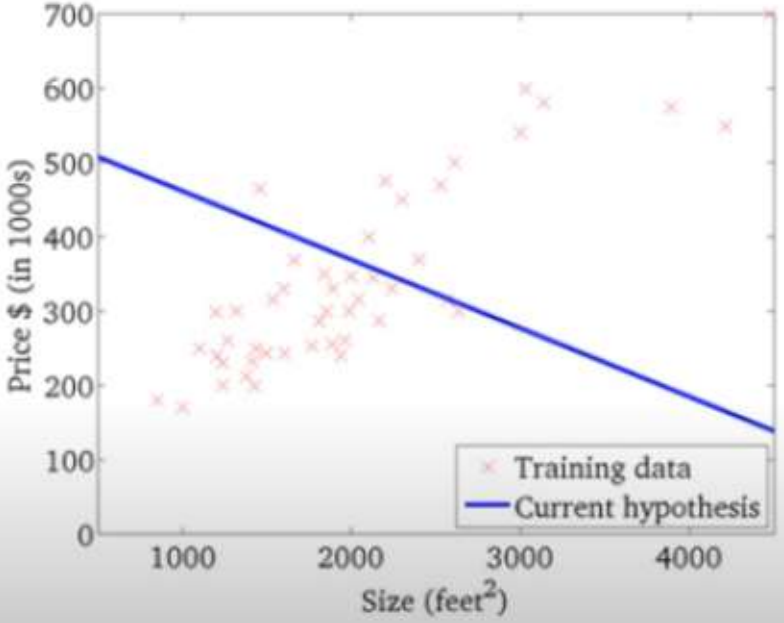
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



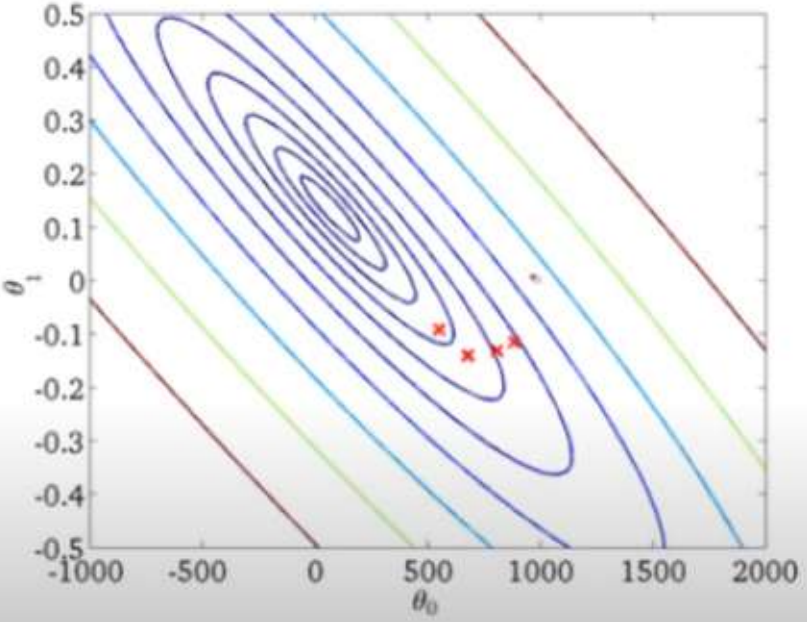
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



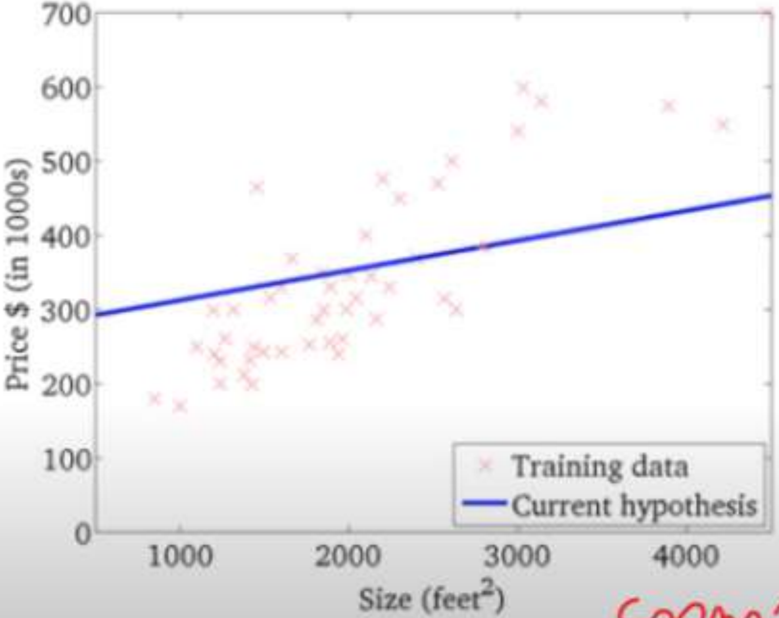
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



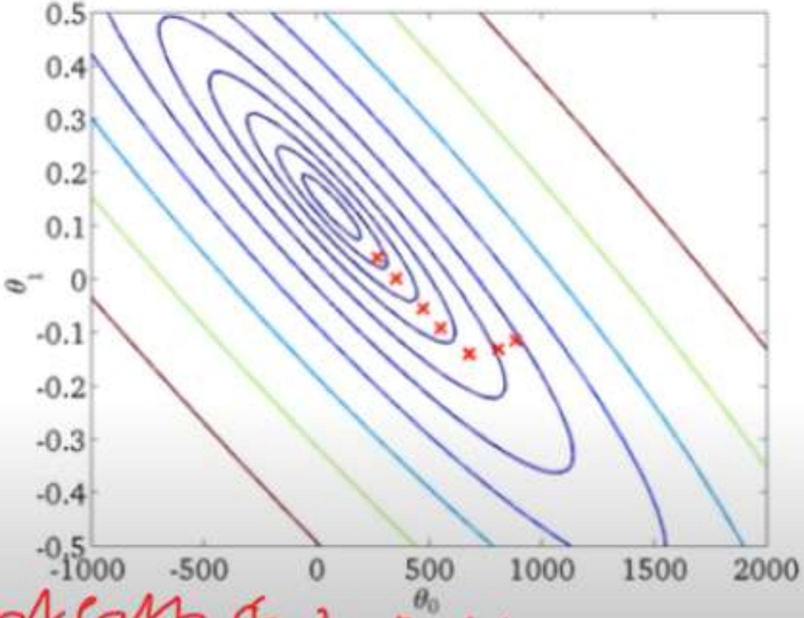
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

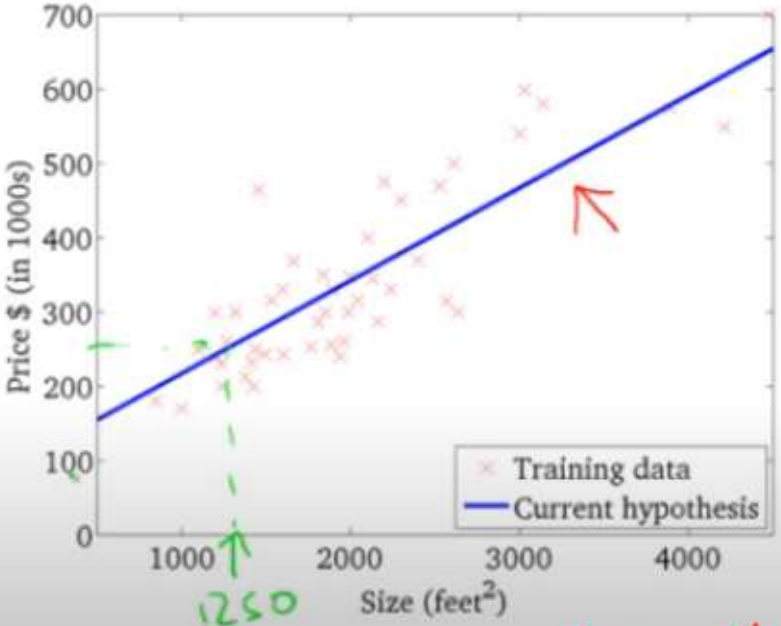
(function of the parameters θ_0, θ_1)



seem to get better fit DATA

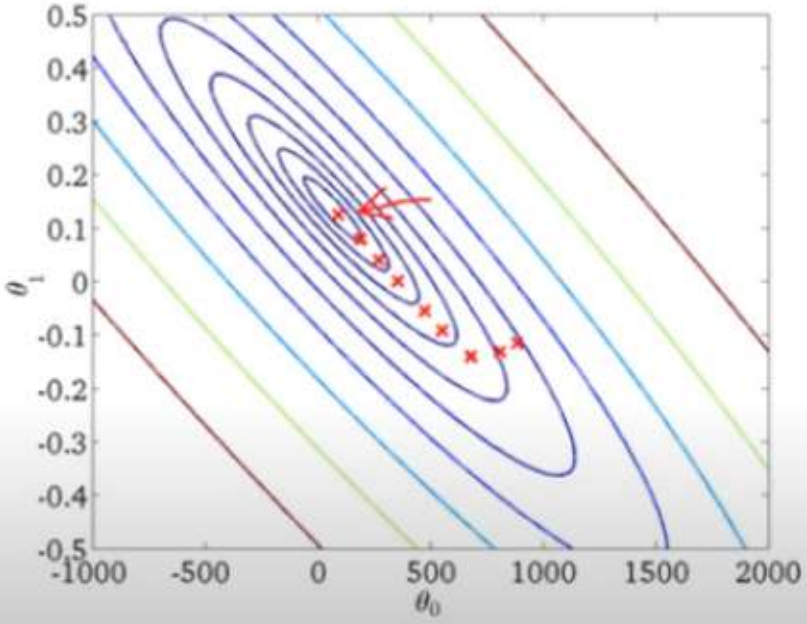
$h_\theta(x)$

(for fixed θ_0, θ_1, this is a function of x)



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



Finally -

"Batch" Gradient Descent

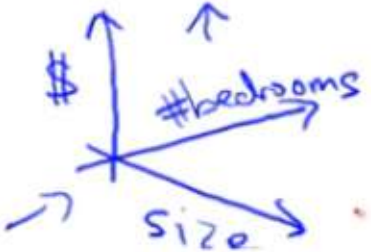
"Batch": Each step of gradient descent uses all the training examples.

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

Two extensions:

1. In $\min J(\theta_0, \theta_1)$, solve for θ_0, θ_1 exactly, without needing iterative algorithm (gradient descent).
2. Learn with larger number of features.

Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178



Linear Algebra

Notation and set of the things you can do with matrices and vectors.

Rest explained in next PDF

Matrix:

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

Vector:

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 172 \end{bmatrix}$$

Multiple features (variables).

Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

$m = 47$

Notation:

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

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

$x_3^{(2)} = 2$

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

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

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n \quad (x_0 = 1)$$

$$= \theta^T x$$

$[\theta_0\ \theta_1 \cdots \theta_n]$ $= \theta^T$: $(n+1) \times 1$ matrix

$\theta^T x$

Multivariate linear regression.

Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$ ($x_0 = 1$)

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

Cost function:

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

Gradient descent:

Repeat {

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

}

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

Gradient Descent

Previously (n=1):

Repeat {

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

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

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

(simultaneously update θ_0, θ_1)

}

New algorithm $(n \geq 1)$:

Repeat {

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

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

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

}

$x_0^{(i)} = 1$

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

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

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

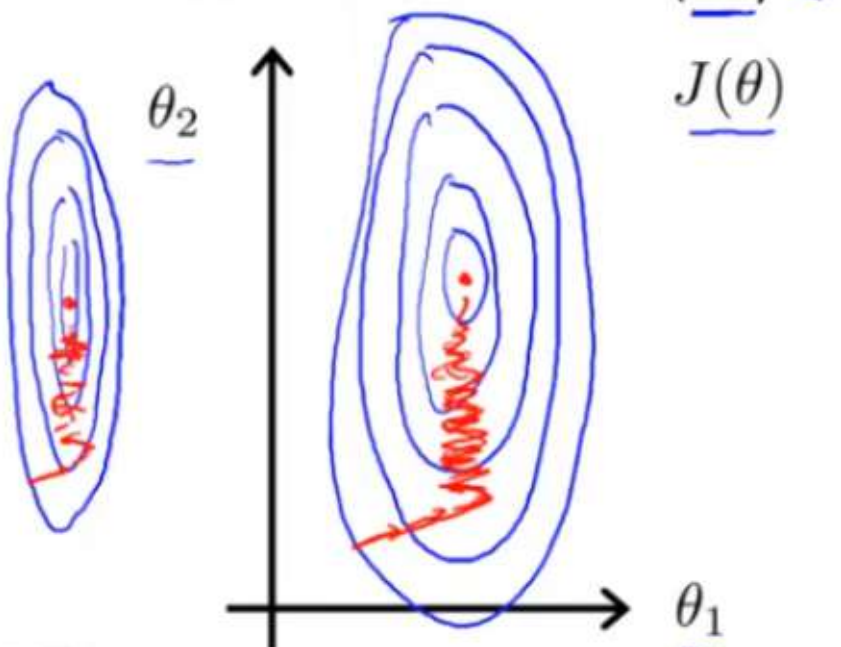
...

Feature Scaling

Idea: Make sure features are on a similar scale.

E.g. x_1 = size (0-2000 feet^2)

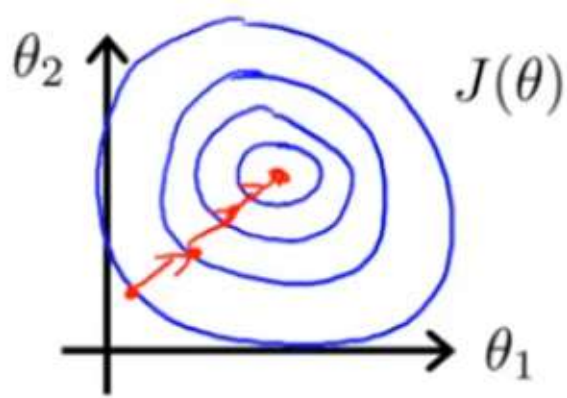
x_2 = number of bedrooms (1-5)



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

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

$0 \le x_1 \le 1$ $\quad$ $0 \le x_2 \le 1$



Feature Scaling

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

$x_0 = 1$

- $0 \leq x_1 \leq 3$ ✓
- $-2 \leq x_2 \leq 0.5$ ✓
- $-100 \leq x_3 \leq 100$ ✗
- $-0.0001 \leq x_4 \leq 0.0001$ ✗

-3 to 3 ✓

$-\frac{1}{3}$ to $\frac{1}{3}$ ✓

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

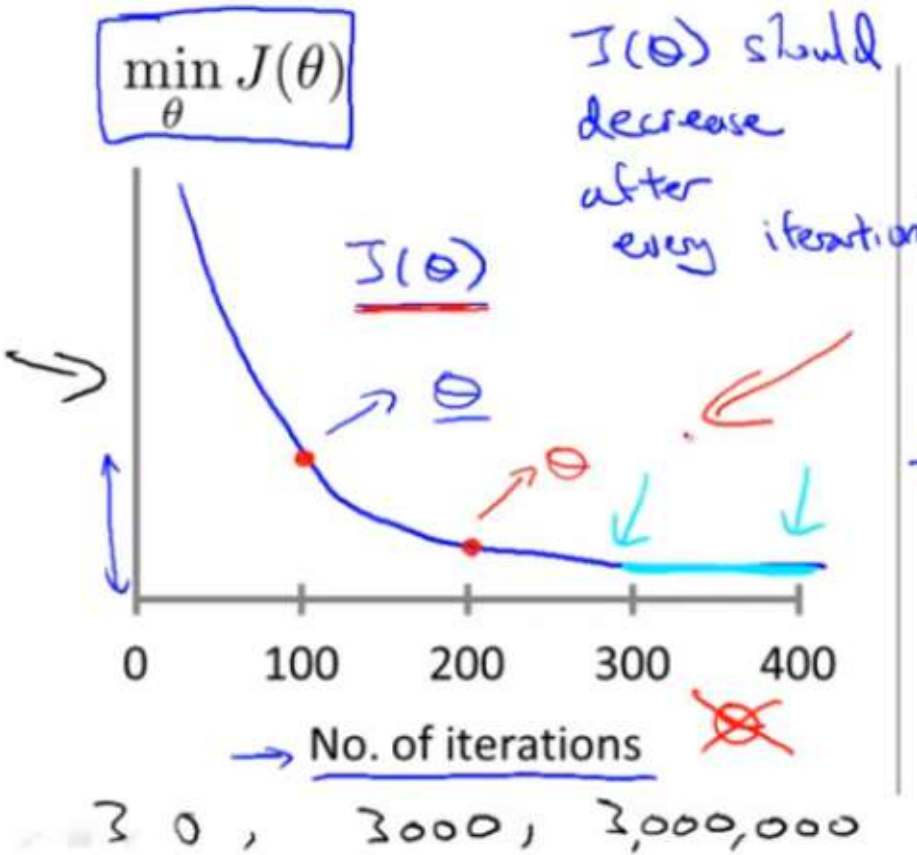
$-0.5 \leq x_1 \leq 0.5$, $-0.5 \leq x_2 \leq 0.5$

$x_1 \leftarrow \frac{x_1 - \mu_1}{s_1}$ ← avg value of x_1 in training set

s_1 = range (max - min) (or standard deviation)

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

Making sure gradient descent is working correctly.

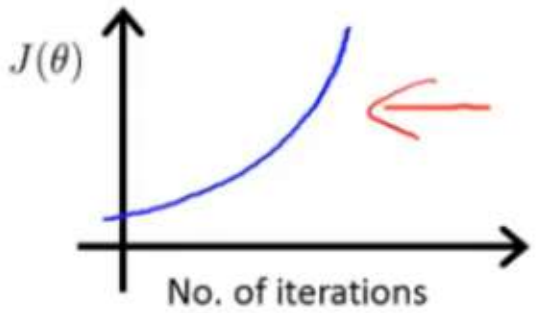


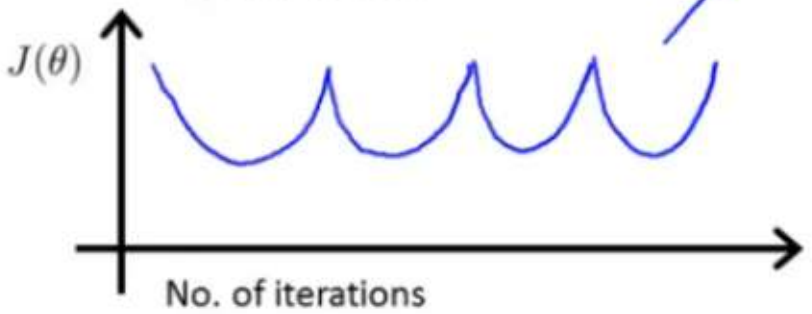
→ Example automatic convergence test:

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

ε

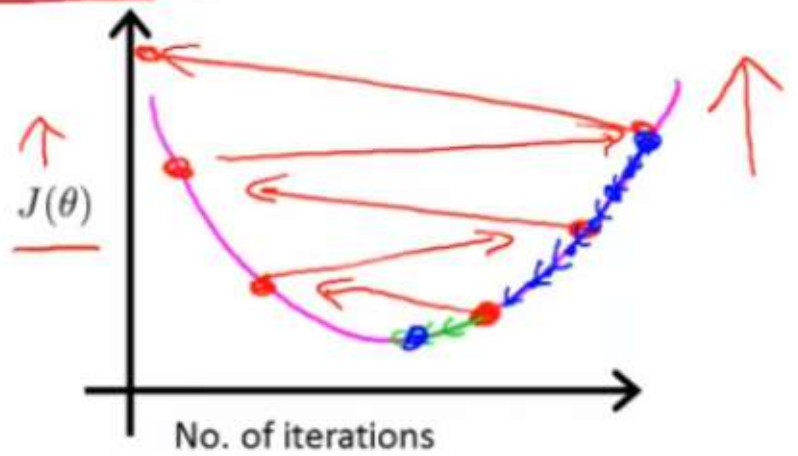
Making sure gradient descent is working correctly.





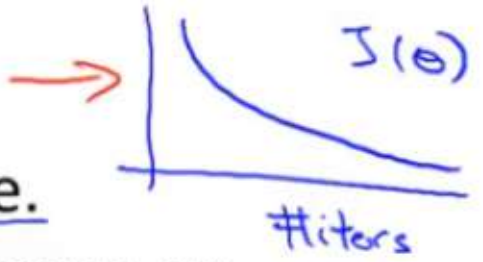
Gradient descent not working.

Use smaller α.



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

Summary:

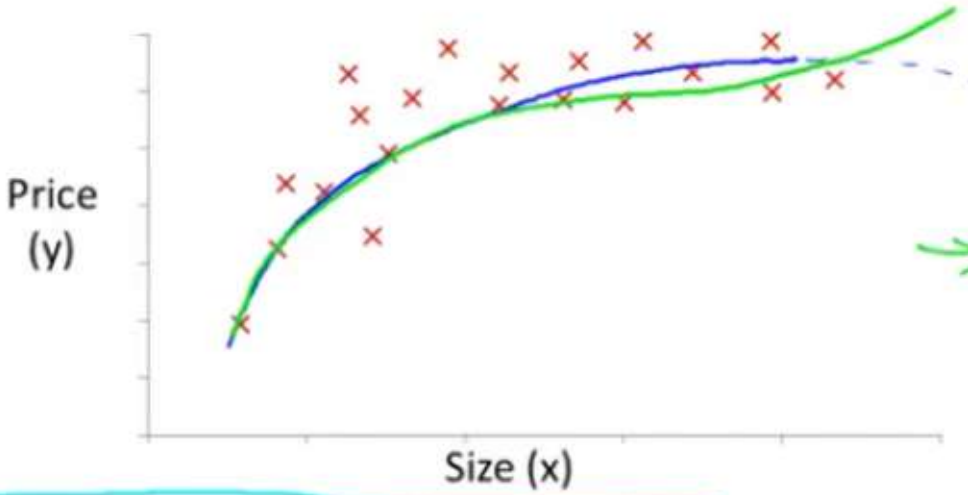


- If α is too small: slow convergence.
- If α is too large: $J(\theta)$ may not decrease on every iteration; may not converge. (Slow converge also possible)

To choose α, try

$$\ldots, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, \ldots$$

Polynomial regression



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

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

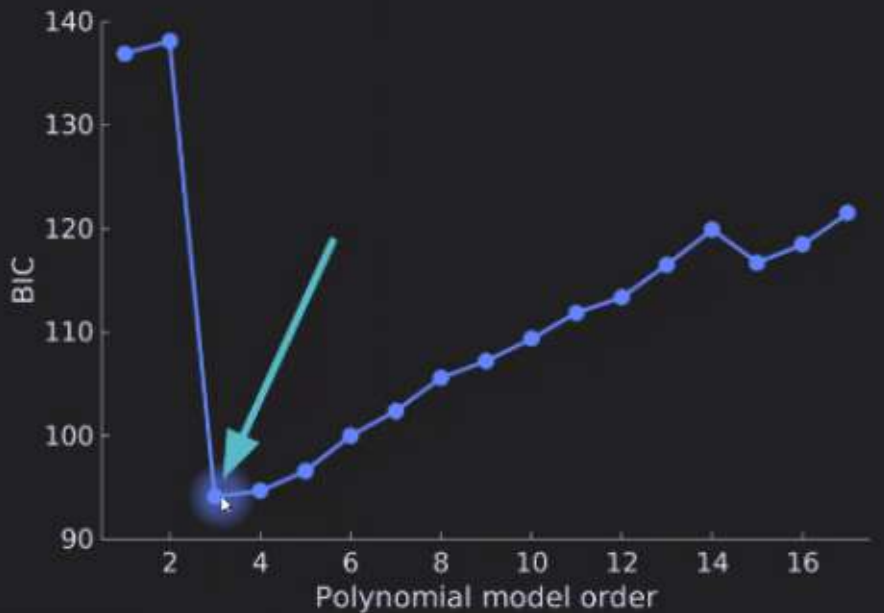
$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$
$$= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3$$

$x_1 = (size)$
$x_2 = (size)^2$
$x_3 = (size)^3$

Size: 1 - 1000
Size^2: 1 - 1000,000
Size^3: 1 - 10^9

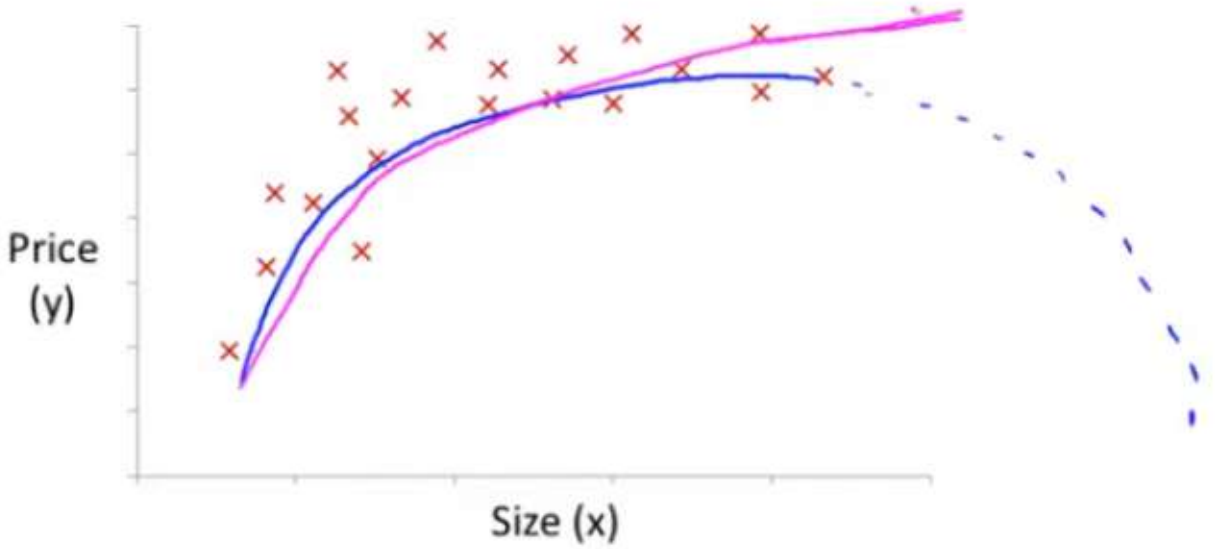
Model order selection, thanks to Bayes

$$\mathrm{BIC}_k = n \log(\mathrm{SS}_\epsilon) + k \log(n)$$



The lowest value on y-axis is the best choice of Polynomial model order that fits with the data best

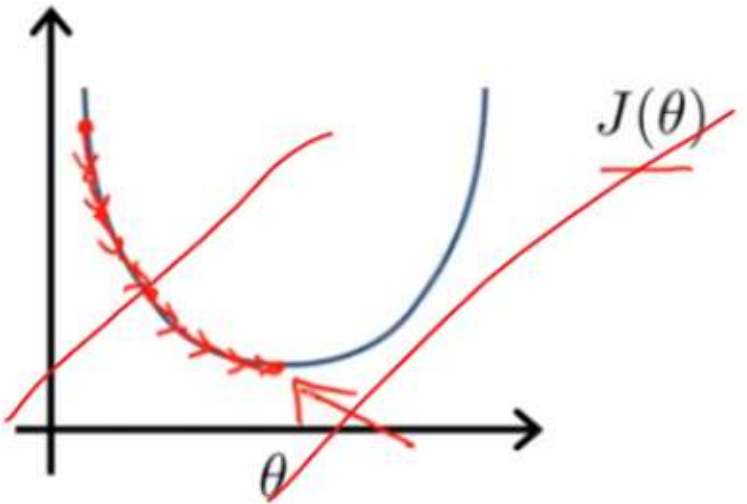
Choice of features



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

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

~~Gradient Descent~~



New Topic
It is another Method

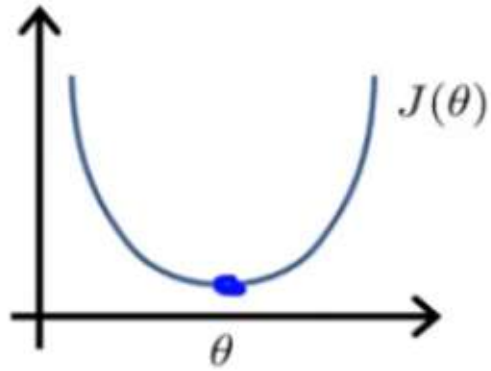
Normal equation: Method to solve for θ analytically.

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

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

$\frac{d}{d\theta} J(\theta) = \dots \overset{\text{set}}{=} 0$

Solve for θ



$\theta \in \mathbb{R}^{n+1}$ $\quad J(\theta_0, \theta_1, \dots, \theta_m) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

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

Solve for $\theta_0, \theta_1, \dots, \theta_n$

Examples: $m = 4$.

x_0	Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

$m \times (n+1)$

m-dimensional vector

$\theta = (X^T X)^{-1} X^T y$ = the value of θ that minimize your cost function

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

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

X (design matrix) $=$

$$\begin{bmatrix} \text{---} & (x^{(1)})^T & \text{---} \\ \text{---} & (x^{(2)})^T & \text{---} \\ & \vdots & \\ \text{---} & (x^{(m)})^T & \text{---} \end{bmatrix}$$

$m \times (n+1)$

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

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

$m \times 2$

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

Andrew N

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

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

Set $A = X^T X$

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

Octave: `pinv(X'*X)*X'*y`

`pinv(X'*X)*X'*y`

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

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

X' X^T

~~Feature Scaling~~

$0 \le x_1 \le 1$

$0 \le x_2 \le 1000$

$0 \le x_3 \le 10^{-5}$ ✓

m training examples, n features.

PRO VS Cons

Gradient Descent

- Need to choose α.
- Needs many iterations.
- Works well even when n is large.

$n = 10^6$

Normal Equation

not really works

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

$n = 100$

$n = 1000$

$n = 10000$

Always invertible matrix

Logistic Regression Model

Want $0 \le h_\theta(x) \le 1$

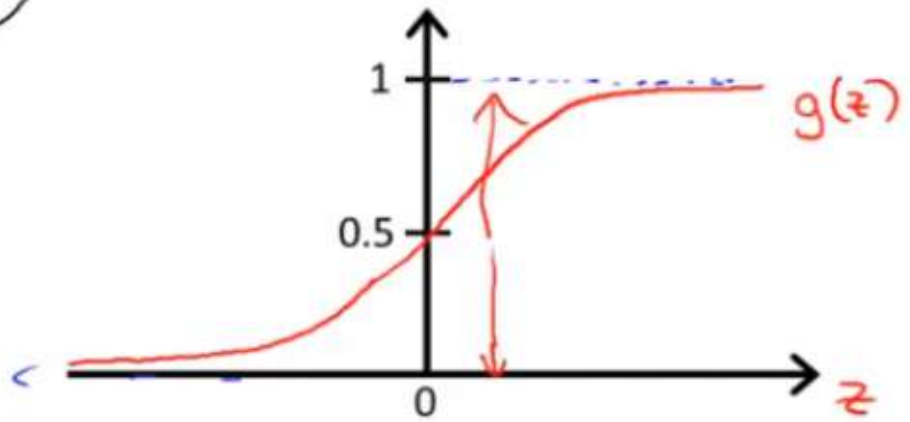
$$h_\theta(x) = g(\theta^T x)$$

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

$z = \theta^T x$

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

- Sigmoid function
- Logistic function



Parameters θ.

Interpretation of Hypothesis Output

$h_\theta(x)$

$h_\theta(x)$ = estimated probability that y = 1 on input x

Example: If $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorSize} \end{bmatrix}$

$h_\theta(x) = 0.7$ $\quad y=1$

Tell patient that 70% chance of tumor being malignant

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

$y = 0$ or 1

"probability that y = 1, given x, parameterized by θ"

Always the P() calc, will be equal to 1, since it is a classification function

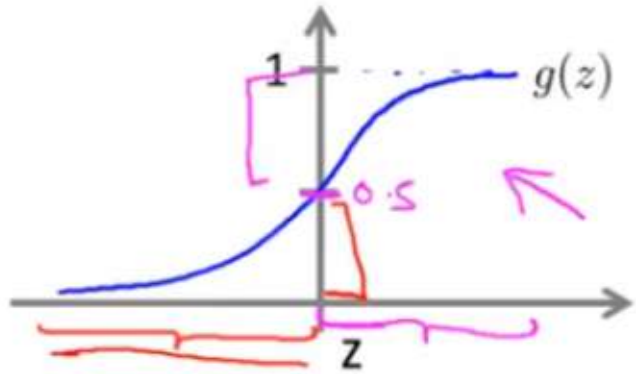
$$P(y=0|x;\theta) + P(y=1|x;\theta) = 1$$

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

Logistic regression

$h_\theta(x) = g(\theta^T x) = P(y=1 \mid x; \theta)$

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



Suppose predict "$y = 1$" if $h_\theta(x) \geq 0.5$

$\theta^T x \geq 0$

$g(z) \geq 0.5$ when $z \geq 0$

$h_\theta(x) = g(\theta^T x) \geq 0.5$ whenever $\theta^T x \geq 0$

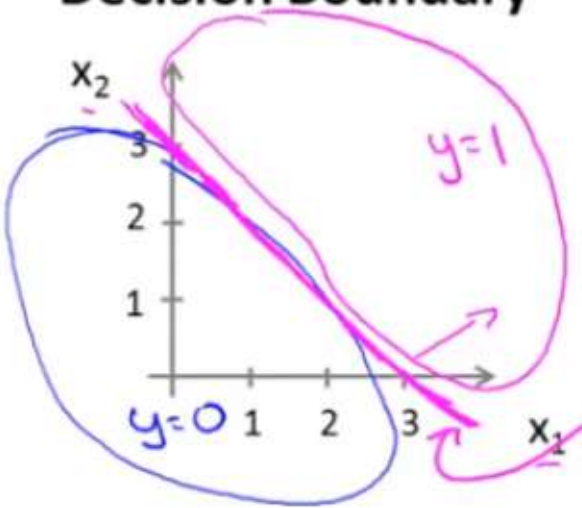
predict "$y = 0$" if $h_\theta(x) < 0.5$

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

$\theta^T x < 0$

$g(z) < 0.5$

Decision Boundary



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

$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$$

$\theta_0 = -3$, $\theta_1 = 1$, $\theta_2 = 1$

Decision boundary

Predict "$y = 1$" if $-3 + x_1 + x_2 \geq 0$

$\theta^T x$

$x_1 + x_2 \geq 3$

x_1, x_2

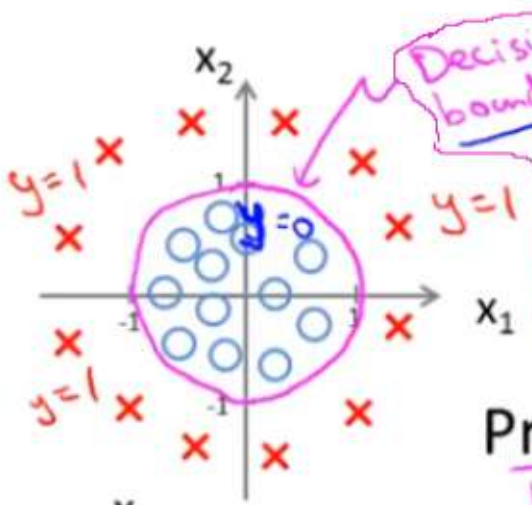
$h_\theta(x) = 0.5$

$x_1 + x_2 = 3$

$x_1 + x_2 < 3$

$y = 0$

Non-linear decision boundaries



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

$\theta = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}$

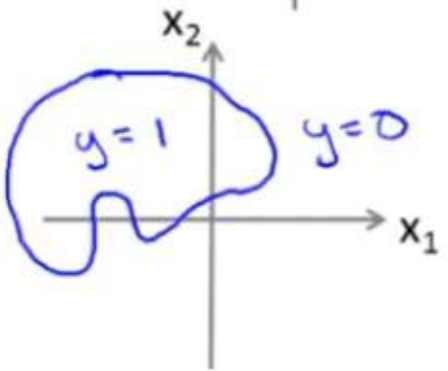
The db in this example is defined by the parameters theta, and not by the (big) data

FOR NOW → See NEXT slide

Predict "$y = 1$" if $-1 + x_1^2 + x_2^2 \geq 0$

$x_1^2 + x_2^2 = 1$

$x_1^2 + x_2^2 \geq 1$



$$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1^2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^2 x_2^2 + \theta_6 x_1^3 x_2 + \dots)$$

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

m examples $\quad x \in \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \quad x_0 = 1, y \in \{0, 1\}$

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

How to choose parameters θ ?

Cost function

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

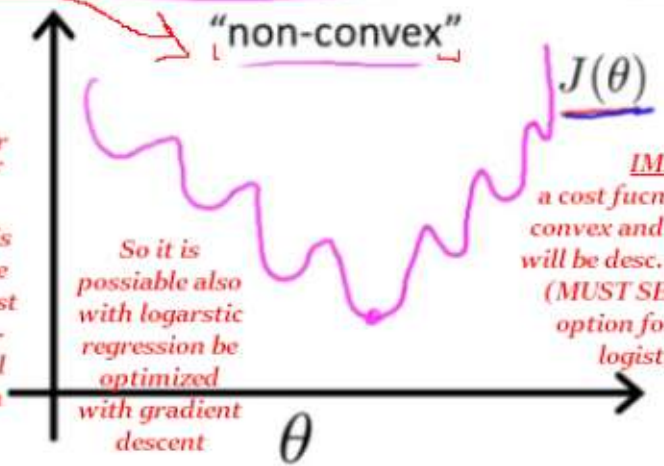
Logistic

$\rightarrow \text{Cost}(h_\theta(x^{(i)}), y^{(i)}) = \frac{1}{2} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$ ←

$\rightarrow \text{cost}(h_\theta(x^{(i)}), y)$

$\frac{1}{1 + e^{-\theta^T x}}$

"non-convex"

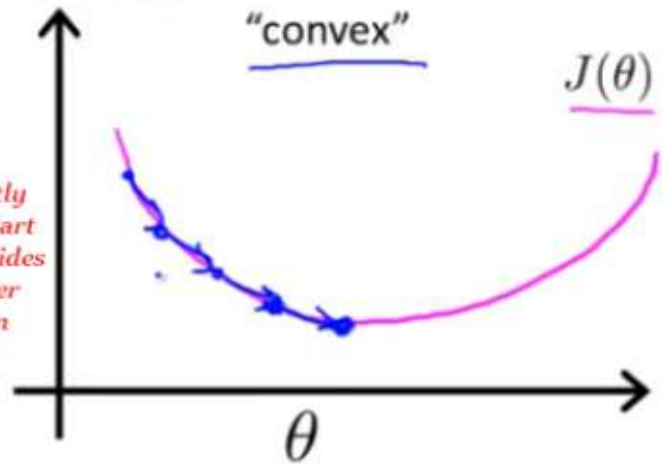


But it is possible, with a particular choice of cost function - is is possible to get a cost function - with local optimum

So it is possiable also with logarstic regression be optimized with gradient descent

IMPORTANT: a cost fucntion that is mostly convex and is used as standart will be desc. after next 2-3 slides (MUST SEE the much better option for cost funciotn in logistic regression)

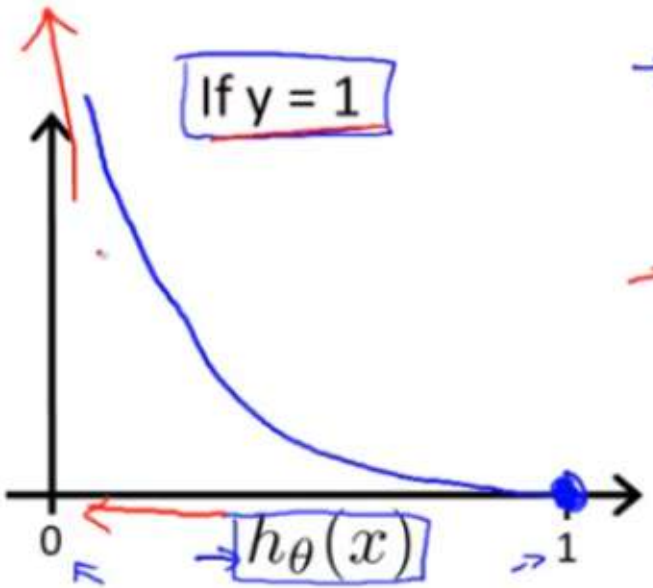
"convex"



Andrew

Logistic regression cost function

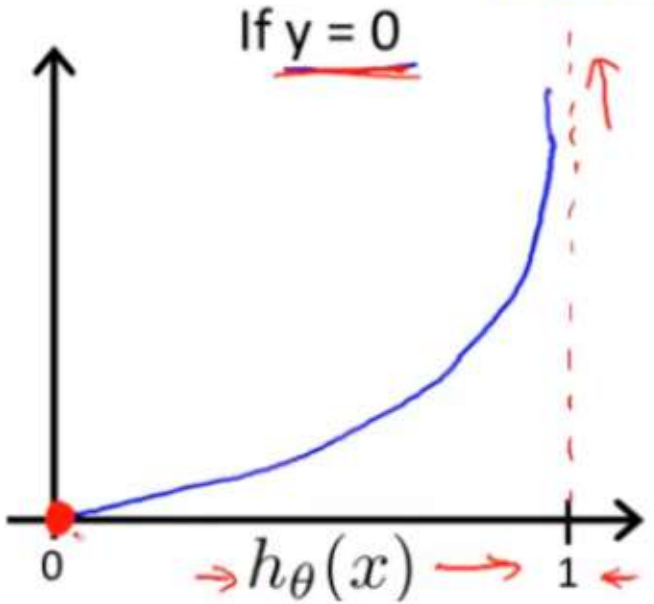
$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

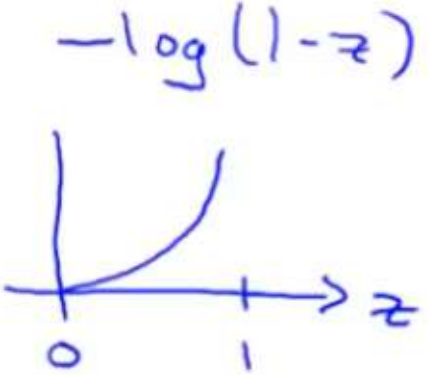


- Cost $= 0$ if $y = 1, h_\theta(x) = 1$
  But as $h_\theta(x) \to 0$
  $Cost \to \infty$
- Captures intuition that if $h_\theta(x) = 0$, (predict $P(y = 1|x; \theta) = 0$), but $y = 1$, we'll penalize learning algorithm by a very large cost.

Logistic regression cost function

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





Logistic regression cost function — Simplified ver.

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

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

"SAME"

Note: $y = 0$ or 1 always

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

(y or $(1-y)$ $=0$ or $=1$)

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

If $y=0$: $\text{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x))$

Logistic regression cost function

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

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

To fit parameters θ:

$$\min_\theta J(\theta)$$ Get θ

To make a prediction given new x:

Output $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ $\Rightarrow$ $p(y=1 \mid x;\theta)$

Gradient Descent

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

Want $\min_\theta J(\theta)$:

Repeat {

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

(simultaneously update all θ_j)

}

see next slide

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

Gradient Descent

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

The cost function of logistic regression

Want $\min_\theta J(\theta)$:

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

for $i = 0$ to n

Although is is possible to calc it with a for loop, mush better and faster solution is the vectorzied calc as you learned

Repeat {

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

(simultaneously update all θ_j)

}

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

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

Algorithm looks identical to linear regression!

They look identical but are not, since the hypothese differ in liner reageression and logistic regression,

The difference is defined with the red and blue color above

Optimization algorithm

Advance Optimization

Given θ, we have code that can compute

- $J(\theta)$
- $\frac{\partial}{\partial \theta_j} J(\theta)$ (for $j = 0, 1, \ldots, n$)

Optimization algorithms:

- Gradient descent
- Conjugate gradient
- BFGS
- L-BFGS

Advantages:

- No need to manually pick α
- Often faster than gradient descent.

Disadvantages:

- More complex

In OCTAVE API

Example:

$\min_\theta J(\theta)$

$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix}$ $\theta_1 = 5, \theta_2 = 5.$

$J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$

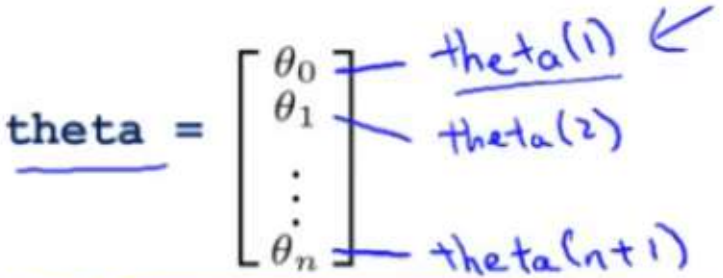
$\frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5)$

$\frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5)$

```
function [jVal, gradient]
            = costFunction(theta)
  jVal = (theta(1)-5)^2 + ...
         (theta(2)-5)^2;
  gradient = zeros(2,1);
  gradient(1) = 2*(theta(1)-5);
  gradient(2) = 2*(theta(2)-5);
```

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

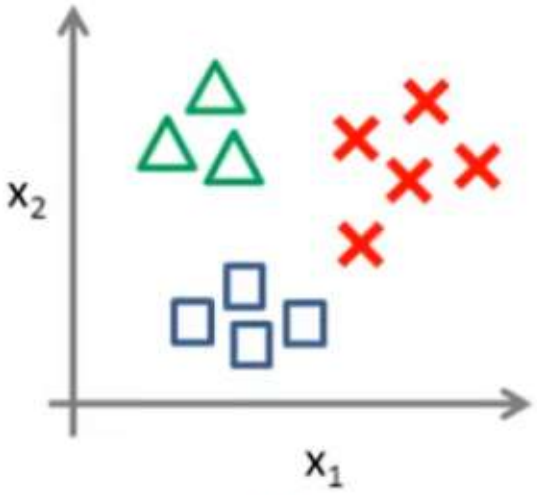
$\theta \in \mathbb{R}^d \quad d \geq 2.$




```
function [jVal, gradient] = costFunction(theta)
  jVal = [code to compute J(θ)];
  gradient(1) = [code to compute ∂/∂θ0 J(θ)];
  gradient(2) = [code to compute ∂/∂θ1 J(θ)];
  :
  gradient(n+1) = [code to compute ∂/∂θn J(θ)];
```

END

One-vs-all (one-vs-rest):

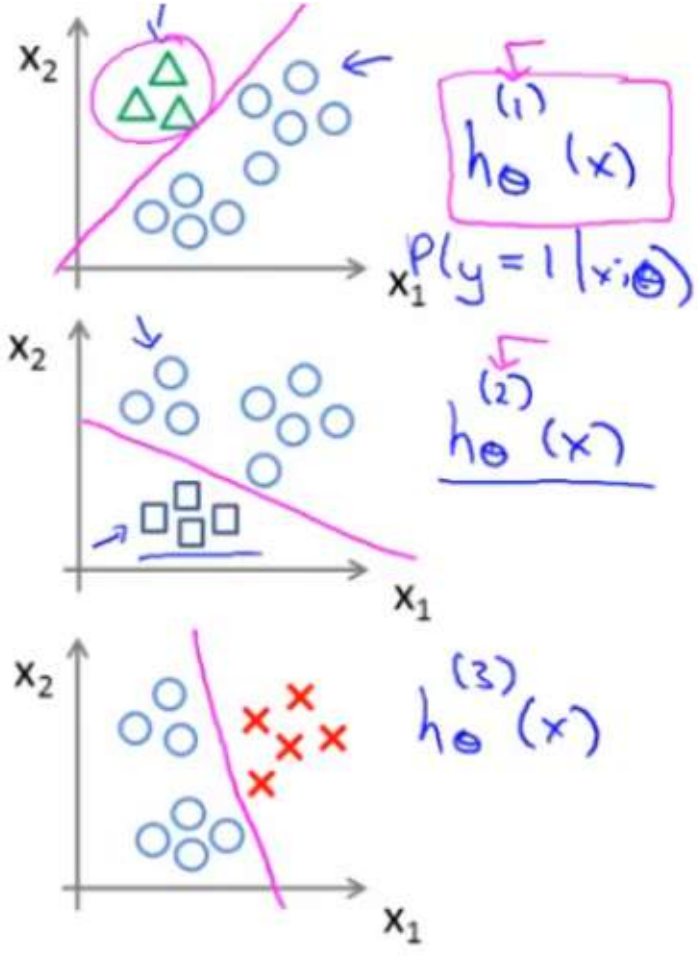


Class 1: △ ←

Class 2: □ ←

Class 3: × ←

$$h_\theta^{(i)}(x) = P(y = i|x;\theta) \qquad (i = 1, 2, 3)$$



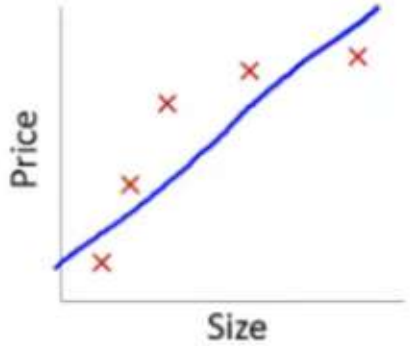
One-vs-all

Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$.

On a new input x, to make a prediction, pick the class i that maximizes

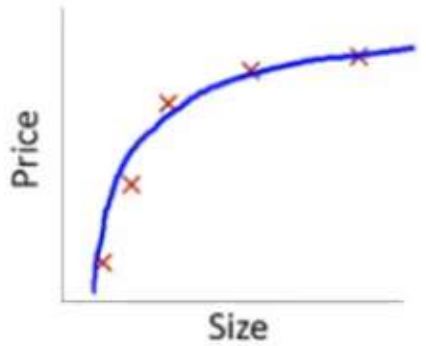
$$\max_i h_\theta^{(i)}(x)$$

Example: Linear regression (housing prices)



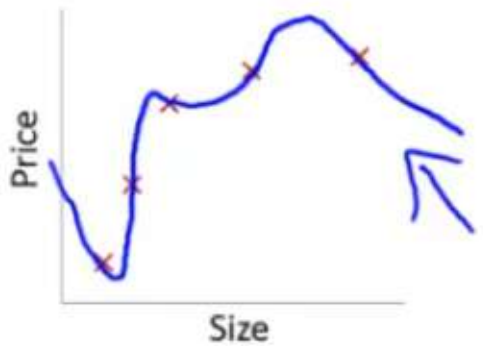
$\rightarrow \theta_0 + \theta_1 x$

"Underfit" "High bias"



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

"Just right"

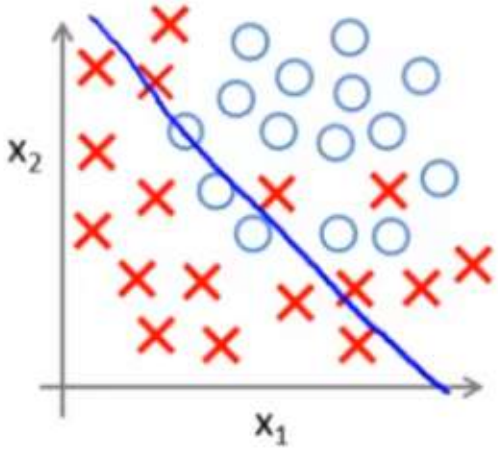


$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x$

"Overfit" "High variance"

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

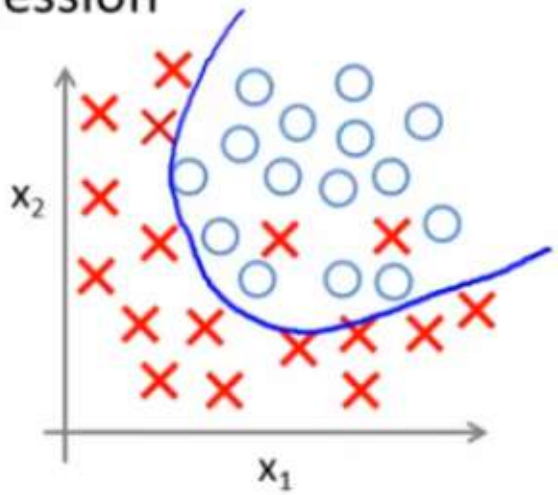
Example: Logistic regression



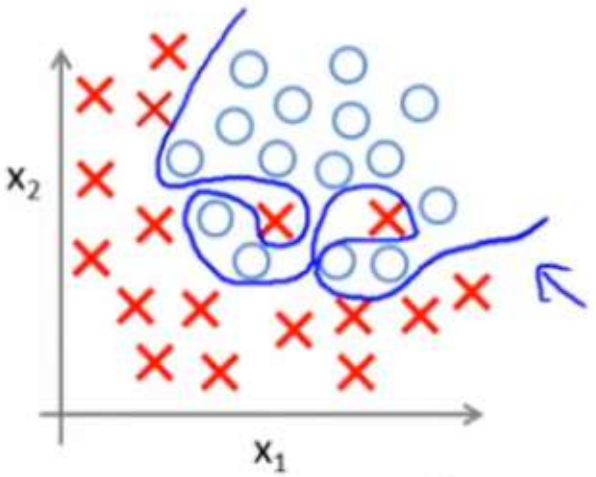
$h_\theta(x) = g(\theta_0 + \theta_1 x_1 + \theta_2 x_2)$

(g = sigmoid function)

"Underfit"



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



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

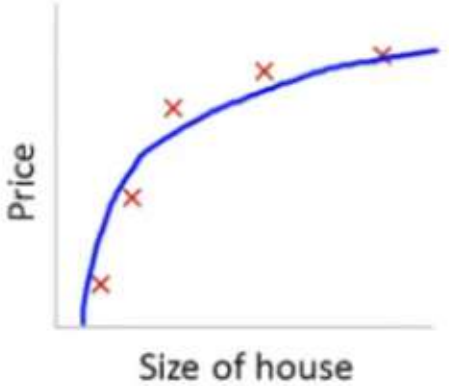
"Overfit"

Addressing overfitting:

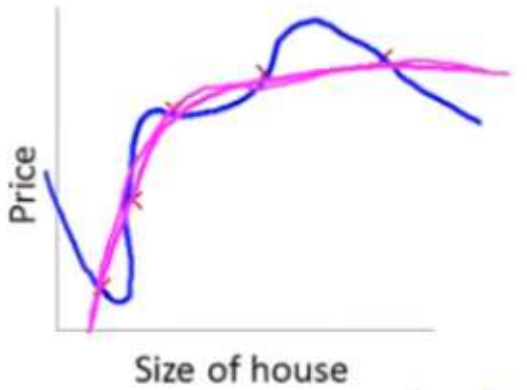
Options:

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

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.

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

This the goal: $\theta_3 \approx 0$ $\theta_4 \approx 0$

Regularization.

Small values for parameters $\theta_0, \theta_1, \ldots, \theta_n$
- "Simpler" hypothesis
- Less prone to overfitting

$\theta_3, \theta_4 \approx 0$

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

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

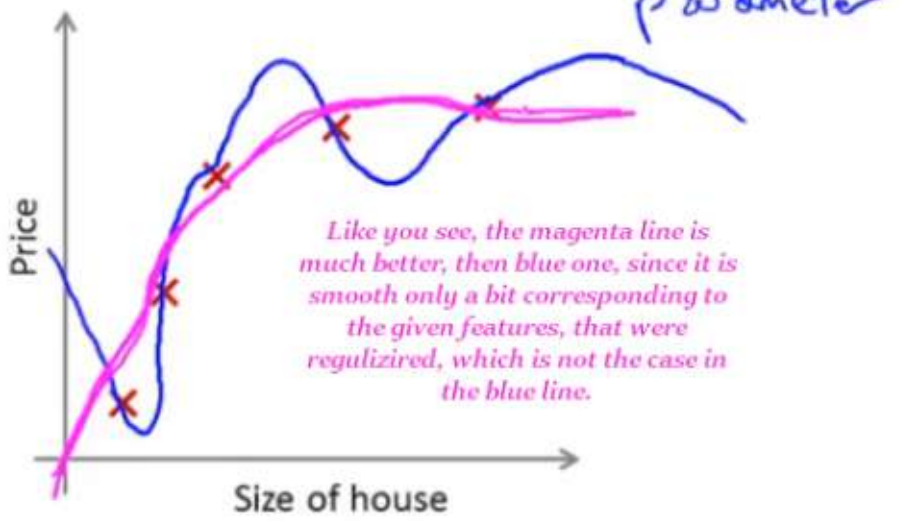
$\theta_1, \theta_2, \theta_3, \ldots, \theta_{100}$

Regularization.

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

regularization parameter

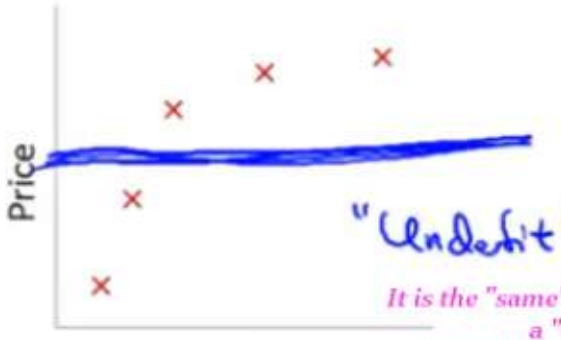
$$\min_\theta J(\theta)$$



In regularized linear regression, we choose θ to minimize

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

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



"Underfit"

The "inverse" effect as in the slide before the features are the small that they do not even matter, and therefore do not predict very well, if this occure we call this UNDERFITTING

It is the "same": like you would just have a "too high bais"
SO REMEMBER, THE DIFFERENCE between "bais and variacne"

$\theta_1, \theta_2, \theta_3, \theta_4$

$\theta_1 \approx 0$, $\theta_2 \approx 0$

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

$h_\theta(x) = \theta_0$

$h_\theta(x)$

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

Gradient descent

Repeat { *Seperetly, defined the value for theta_o, since in "j" the o^th index is excluded*

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

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

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

}

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

so slowly decreseing the parameter also on the lamba (I guess) so able you to aply. also in linerar regression.

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

0.99

$\theta_j \times 0.99$

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

$J(\theta)$

θ_j^2

Normal equation

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

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

$\rightarrow \min_\theta J(\theta)$

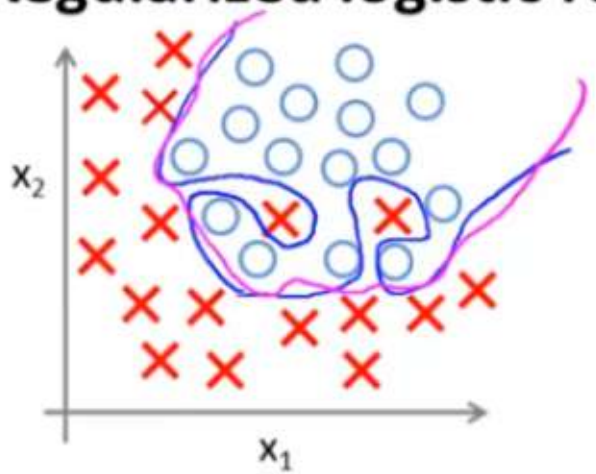
$\frac{\partial}{\partial \theta_j} J(\theta) \overset{\text{set}}{=} 0$ for all j

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

$(n+1) \times (n+1)$

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

Regularized logistic regression.



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

Cost function:

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

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

Gradient descent

Repeat {

Seperetly, define the value for theta_0, since in field "j" the index "zero" is excluded see below

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

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

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

$\theta_1 \ldots \theta_n$

}

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

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

Advanced optimization

fminunc (@costFunction, θ)

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$ — theta(1), theta(2), theta(n+1)

```
function [jVal, gradient] = costFunction(theta)
    jVal = [code to compute J(θ)];
```

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

```
    gradient(1) = [code to compute ∂/∂θ0 J(θ)];
```

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

```
    gradient(2) = [code to compute ∂/∂θ1 J(θ)];
```

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

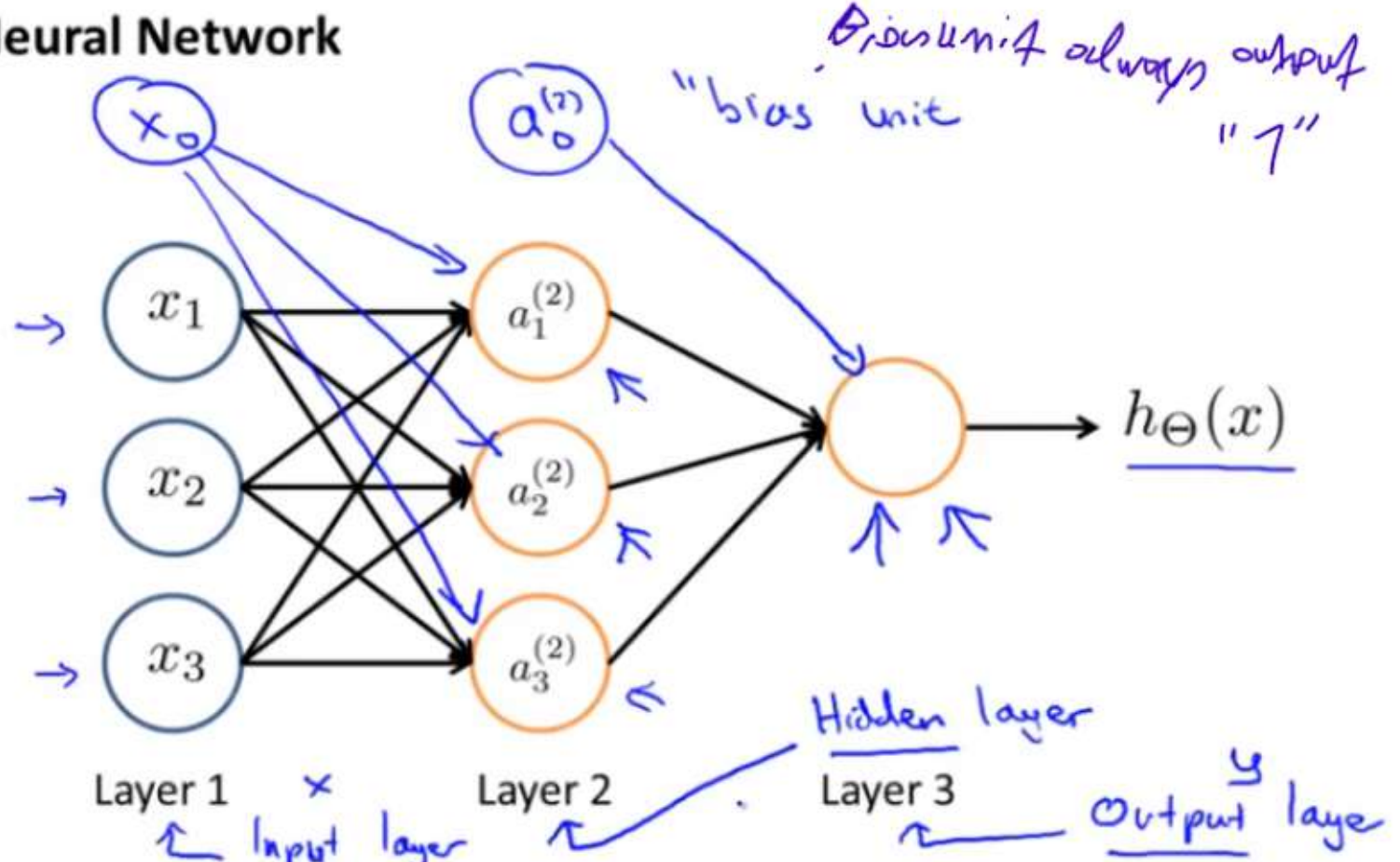
```
    gradient(3) = [code to compute ∂/∂θ2 J(θ)];
    :
```

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

```
    gradient(n+1) = [code to compute ∂/∂θn J(θ)];
```

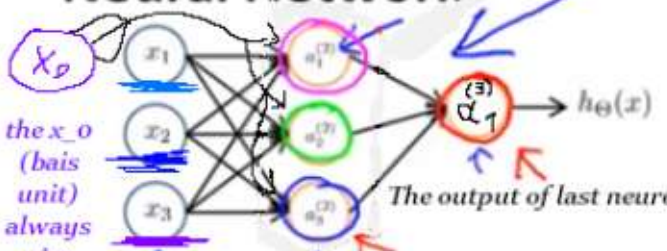
J(θ)

Neural Network



MODEL I

Neural Network



the x_0 (bais unit) always give ouput "1"

3 units

2 hidden units

Activation (=receiving "input" -> compute the value -> output (to another neuron))

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

Weigths = Parameters

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

The output of last neuron = h_Theta (x)

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

$h_\Theta(x)$

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

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

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

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

$\Theta^{(2)}$

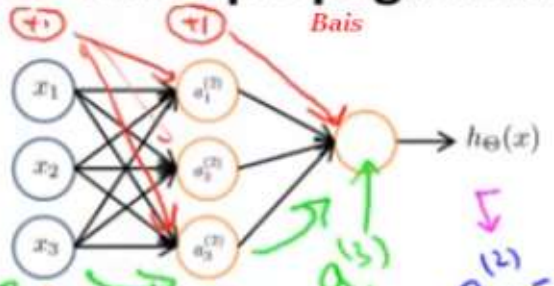
Weights are the value with the algor. will multiply with of the given x_i etc. so weights are the cost of line => this is explanined after 2 slide

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

$S_{j+1} \times (S_j + 1)$

Andrew Ng

Forward propagation: Vectorized implementation



$a^{(1)} = x$

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

$z^{(2)}$

Not correlated to rest of red

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

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

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

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

This red is correlated

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

$z^{(3)}$

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

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

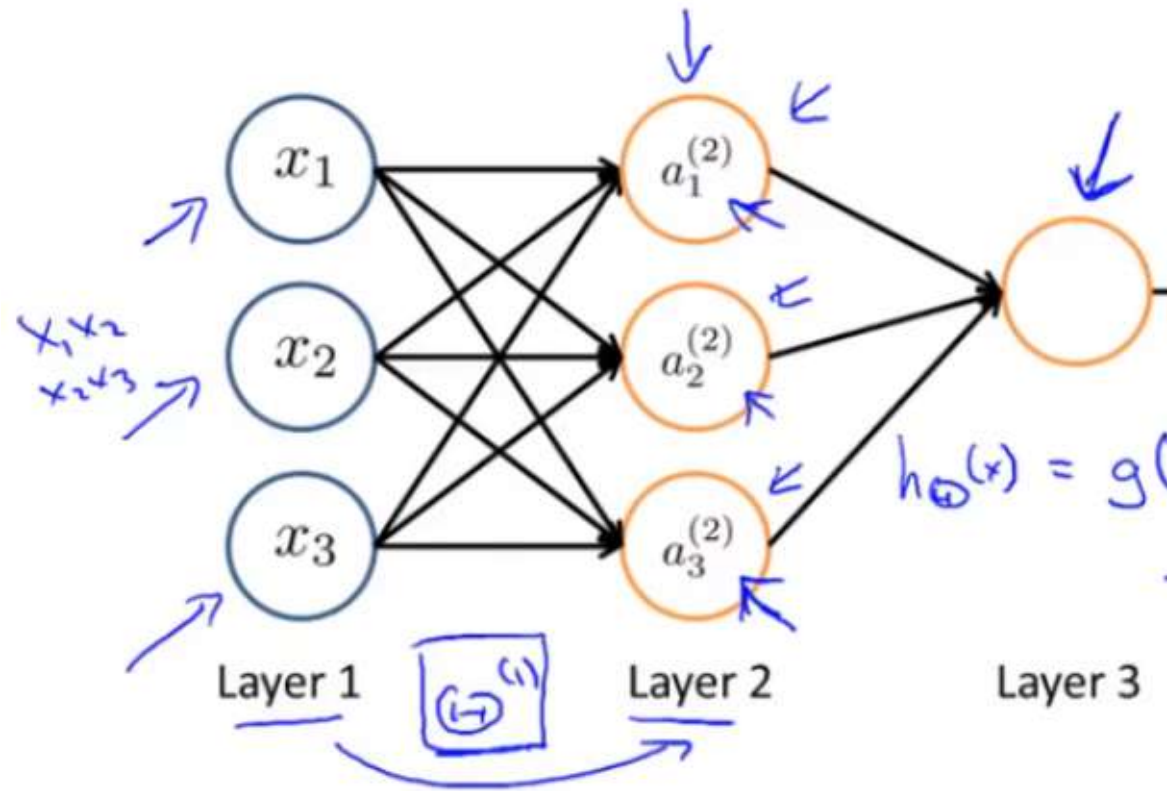
$a^{(2)} = g(z^{(2)})$ $\quad a^{(2)} \in \mathbb{R}^3$

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

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

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

Neural Network learning its own features



It work just the same as the logistic regression, only here the "input" value for the logistic regression were already been leraed through the algor., since that is how the Neural Netweork is working it compute each value and successive, and therefore it choose the most fit ("edited") data/values at the end, and that result in the approx. best hypoteses at end

$h_\Theta(x)$

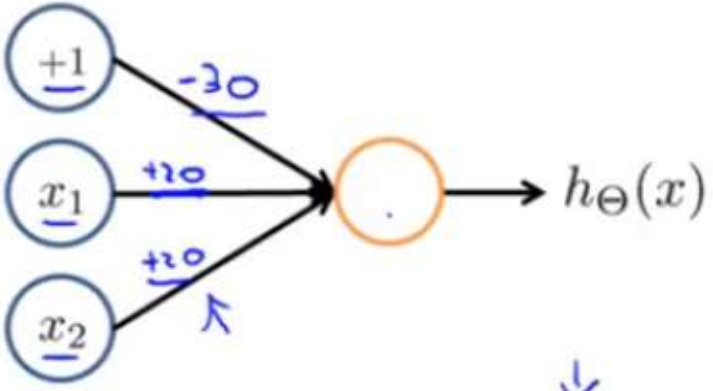
$h_\Theta(x) = g(\Theta^{(2)}_{1,0} a_0^{(2)} + \Theta^{(1)}_{1,1} a_1^{(1)} + \Theta^{(2)}_{1,2} a_2^{(2)} + \Theta^{(1)}_{1,3} a_3^{(2)})$

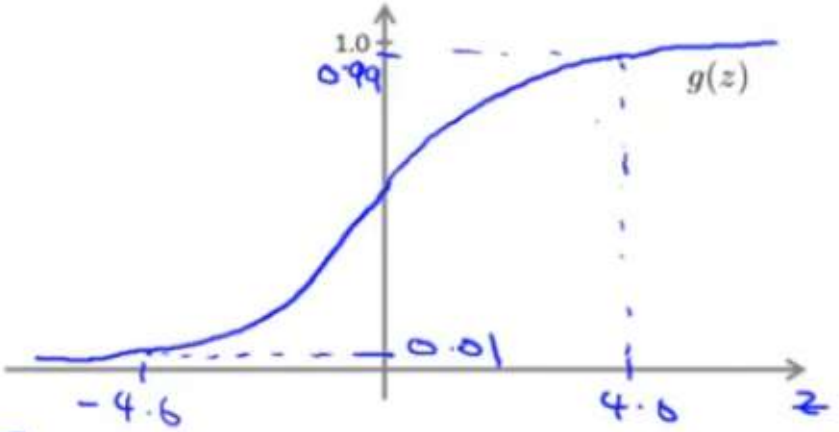
x_1, x_2, x_3

$\rightarrow a_1^{(2)}, a_2^{(2)}, a_3^{(2)}$

Simple example: AND

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





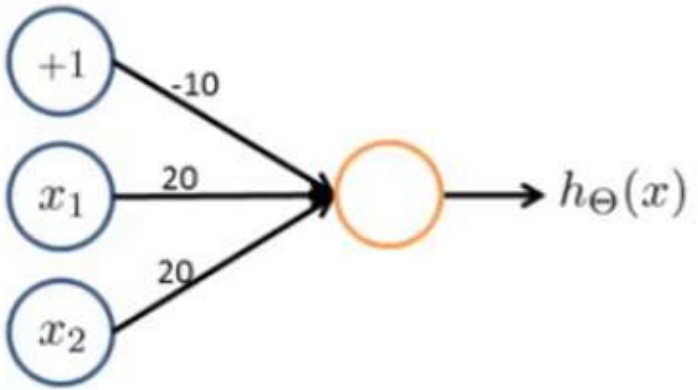
$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$

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

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	$g(10) \approx 1$

$h_\Theta(x) \approx x_1$ AND x_2

Example: OR function



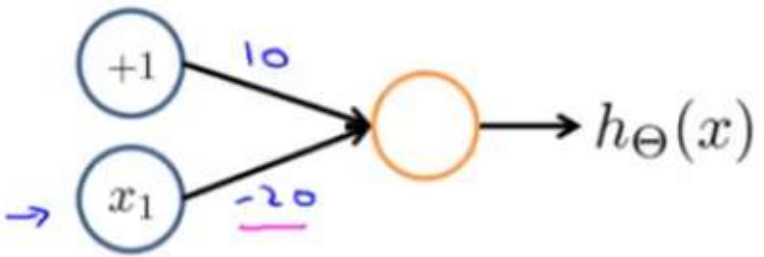
$g(-10 + 20x_1 + 20x_2)$

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

→ x_1 AND x_2 → x_1 OR x_2 {0,1}.

Negation:

NOT x_1



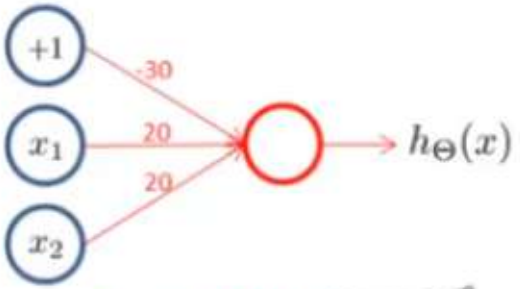
x_1	$h_\Theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

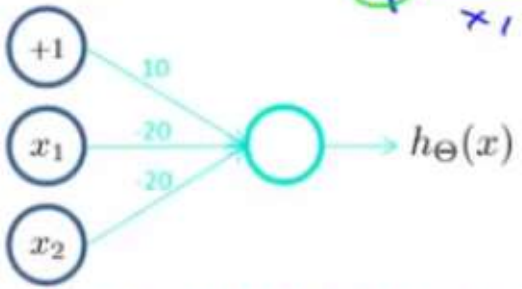
$h_\Theta(x) = g(10 - 20x_1)$

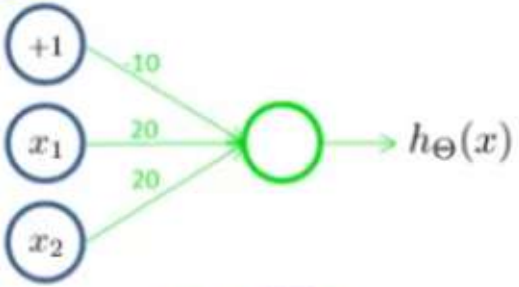
→ (NOT x_1) AND (NOT x_2)

(= 1 if and only if → $x_1 = x_2 = 0$

Putting it together: x_1 XNOR x_2



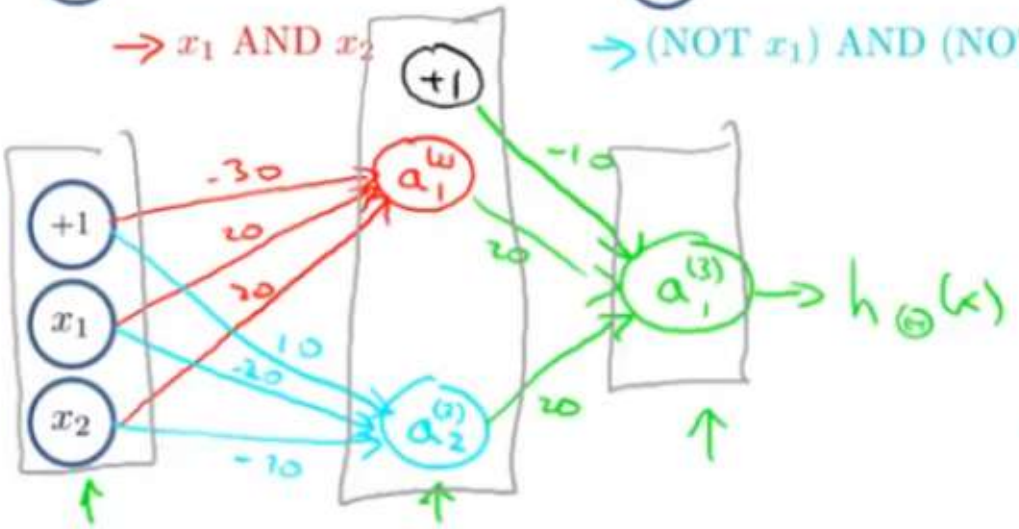




→ x_1 AND x_2

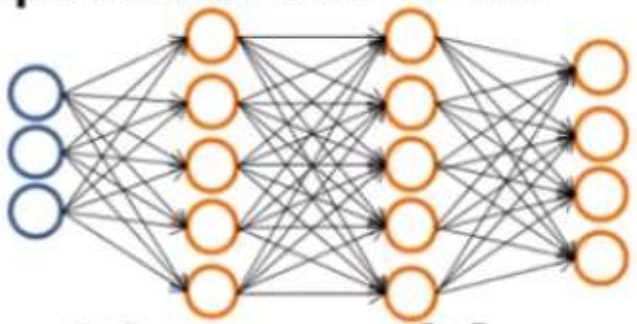
→ (NOT x_1) AND (NOT x_2)

→ x_1 OR x_2



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

Multiple output units: One-vs-all.



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

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

when pedestrian when car when motorcycle

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

$y^{(i)}$ one of $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$

pedestrian car motorcycle truck

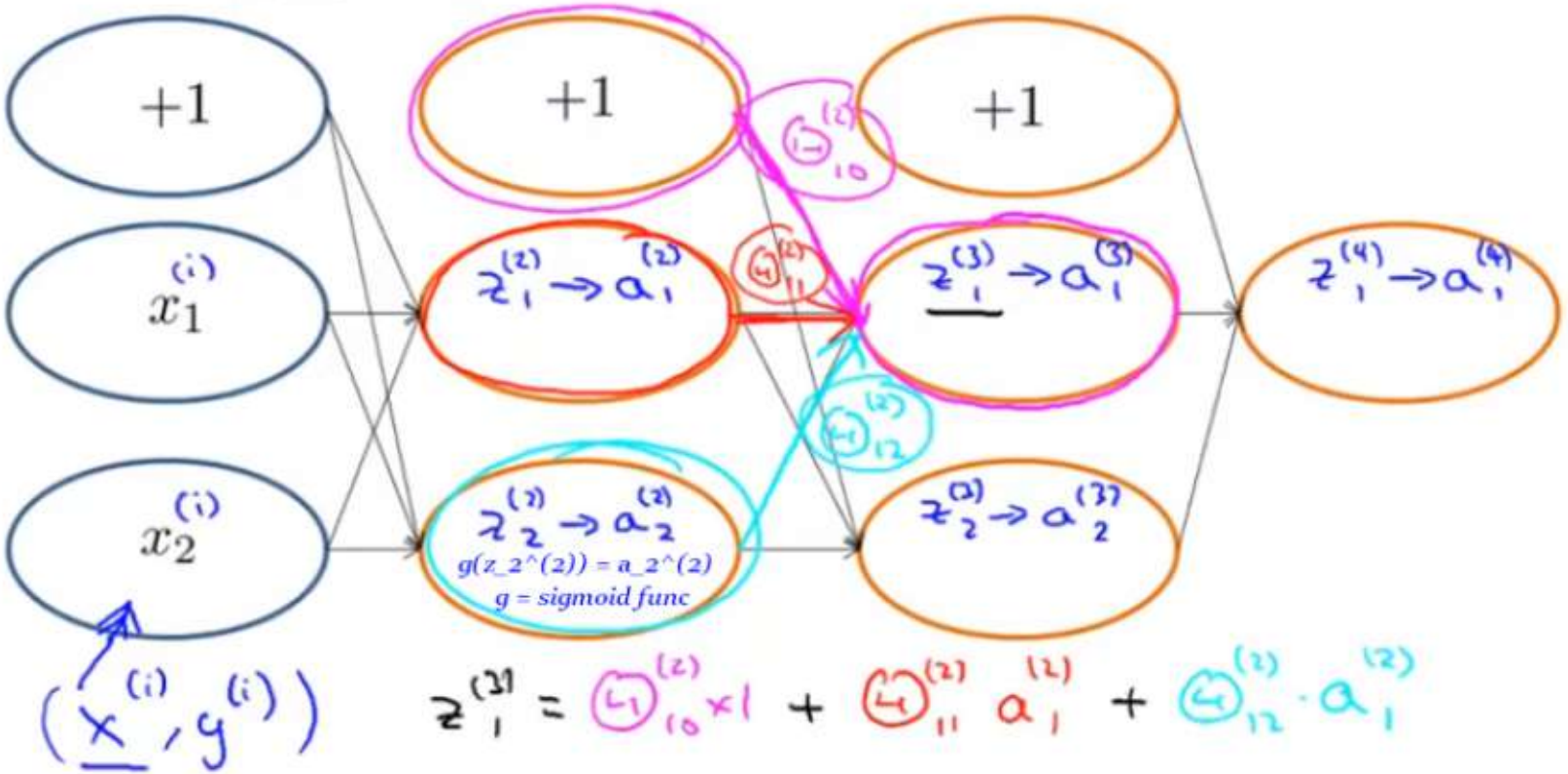
Previously $y \in \{1, 2, 3, 4\}$

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

$h_\Theta(x^{(i)}) \approx y^{(i)}$

$\in \mathbb{R}^4$

Forward Propagation



Before was the Intution behind the Neural Networks,
Now we will deal with how the Neural Networks, indeed, learn in algor.

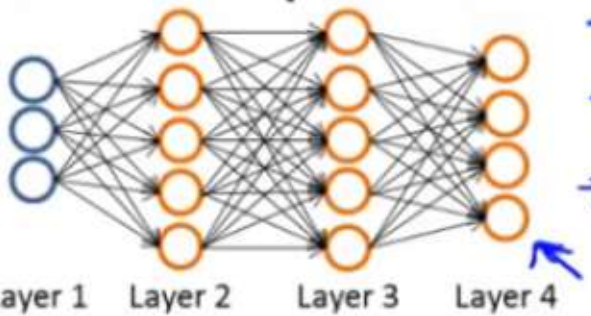
THIS IS IMPORTANT

Neural Networks: Learning

Cost function

Machine Learning

Neural Network (Classification)



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

$L =$ total no. of layers in network $L = 4$

$s_l =$ no. of units (not counting bias unit) in layer l $s_1 = 3,\ s_2 = 5,\ s_4 = s_L = 4$

Binary classification

$y = 0$ or 1

1 output unit

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

$s_L = 1, \quad K = 1$

Multi-class classification (K classes)

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

pedestrian car motorcycle truck

K output units

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

$s_L = K \quad (K \geq 3)$

Cost function

Logistic regression:

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

= do not sum up bais

Neural network:

Out put a K vector, if mulitply output

Select out the i^th elelemtn of the output that were computed

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

Same as above only we also have to sum the K units as well in the cost function

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k)\right]$$

SO that the sum up of the K units that were outputed

If lambda = 0, then no optimalization

Regularizaiton for neural Network

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

it is possiable to sum up the bias, but no oft

Execpt the bais unit, mean it is not sum up

Each of K

y_k

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

Gradient computation

sum over K mean we have more then one output units

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log h_\theta(x^{(i)})_k + (1 - y_k^{(i)}) \log(1 - h_\theta(x^{(i)})_k)\right]$$

If lambd equals zero, then the algor. do not optimized

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_j^{(l)})^2$$

Regularization Terms

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

Need code to compute:

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

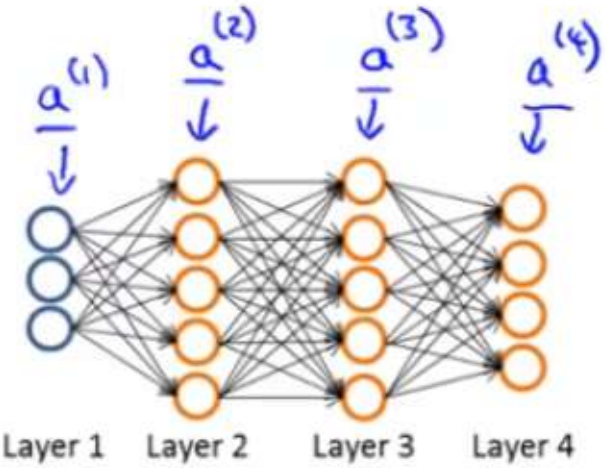
$\Theta_{ij}^{(l)} \in \mathbb{R}$

Gradient computation

Given one training example (x, y):

Forward propagation:

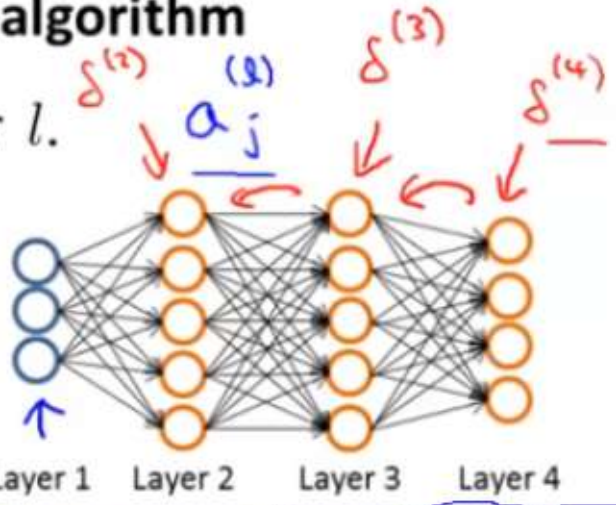
$$a^{(1)} = x$$
$$z^{(2)} = \Theta^{(1)} a^{(1)}$$
$$a^{(2)} = g(z^{(2)}) \quad (\text{add } a_0^{(2)})$$
$$z^{(3)} = \Theta^{(2)} a^{(2)}$$
$$a^{(3)} = g(z^{(3)}) \quad (\text{add } a_0^{(3)})$$
$$z^{(4)} = \Theta^{(3)} a^{(3)}$$
$$a^{(4)} = h_\Theta(x) = g(z^{(4)})$$



To get the derivatives of the neural network values, that were already computed by algor.

Gradient computation: Backpropagation algorithm

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



For each output unit (layer L = 4)

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

$(h_\Theta(x))_j$

Vectorized Version

$\delta^{(4)} = a^{(4)} - y$

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

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

Elementwise Mulitplication

Do Calculus and as result you should get the follwing right equation: For the derivatives for prime sigmoid function –

Same

$$a^{(3)} .* (1 - a^{(3)})$$

$$a^{(2)} .* (1 - a^{(2)})$$

a^(3) is : Vector for activations values of that layer & "1" is the vector of ones

DO not exitst, since we do not want to change the original values

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

Fazit: it is possiable to prove that: after the above calcs you get

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

(ignoring λ; if $\lambda = 0$)

Backpropagation algorithm

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

Set $\Delta_{ij}^{(l)} = 0$ (for all l, i, j). (used to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$)

For $i = 1$ to m ← $(x^{(i)}, y^{(i)})$.

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

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

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

We do not change the original input

Vectorzied implementation in loop

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

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

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

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

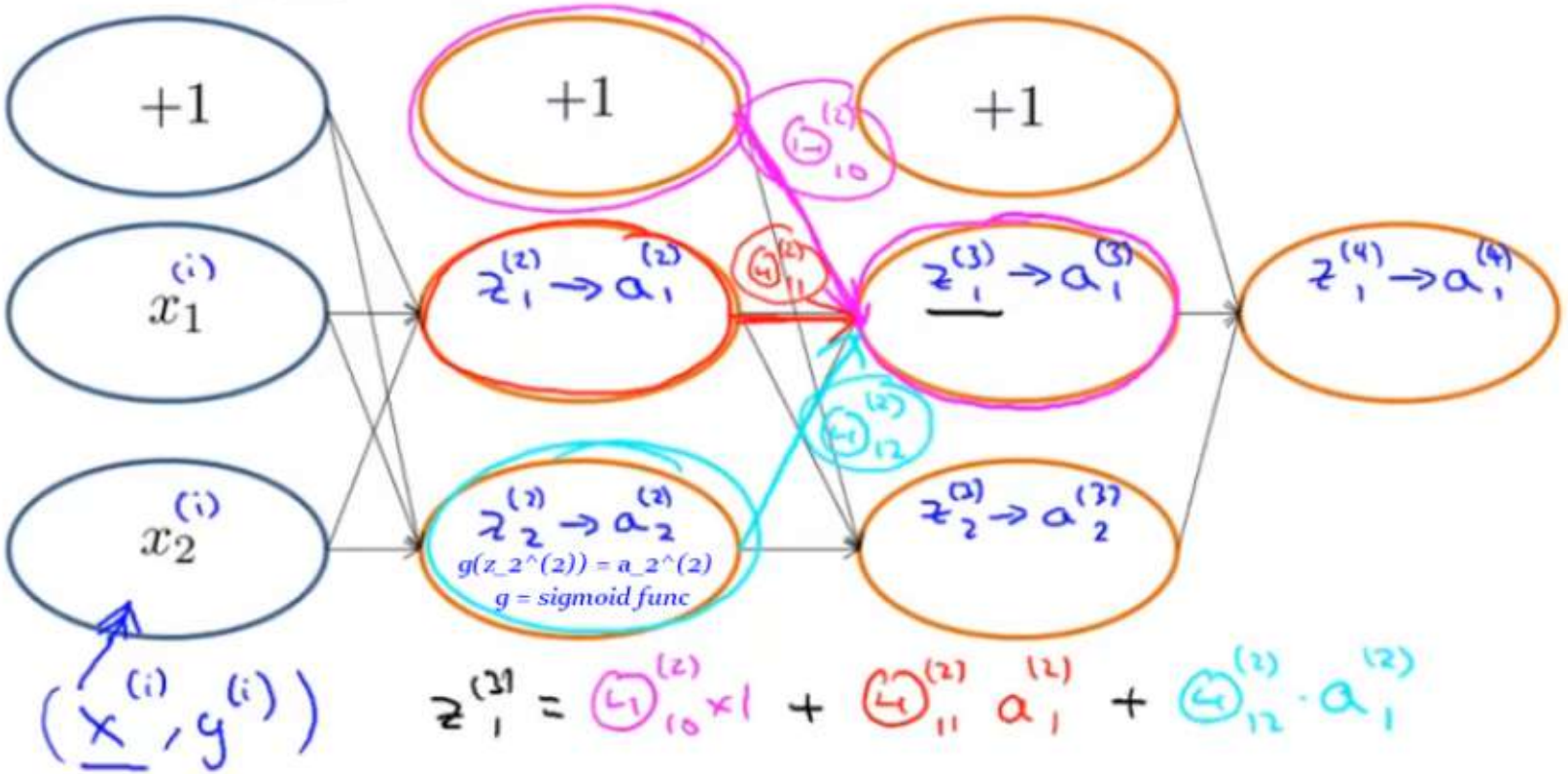
→ $D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$ if $j = 0$

if compute this, therefore: Fazit

Fazit

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

Forward Propagation



What is backpropagation doing?

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

Assoz. with the values on right

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

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

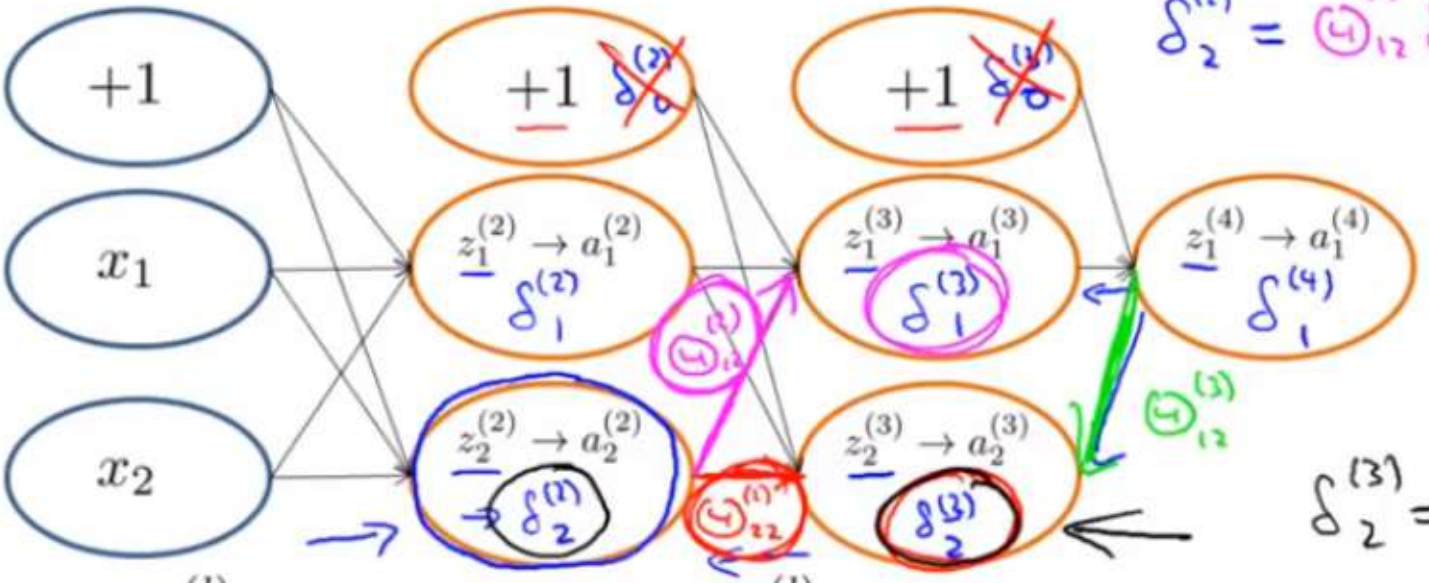
$$\text{cost(i)} = y^{(i)} \log h_\Theta(x^{(i)}) + (1 - y^{(i)}) \log h_\Theta(x^{(i)})$$

the standart cost func

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

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

Forward Propagation *To Backward Propagation*



$\delta_1^{(4)} = y^{(i)} - a_1^{(4)}$

$\delta_2^{(2)} = \Theta_{12}^{(2)} \delta_1^{(3)} + \Theta_{22}^{(2)} \cdot \delta_2^{(3)}$

$\delta_2^{(3)} = \Theta_{12}^{(3)} \cdot \delta_1^{(4)} \cdot$

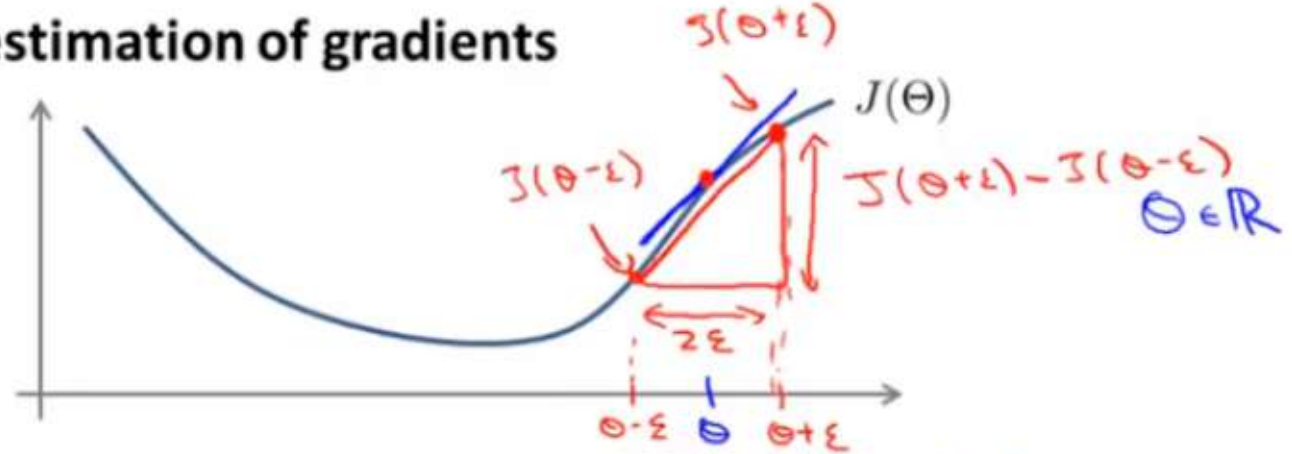
Edit

→ $\delta_j^{(l)}$ = "error" of cost for $a_j^{(l)}$ (unit j in layer l).

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

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

Numerical estimation of gradients

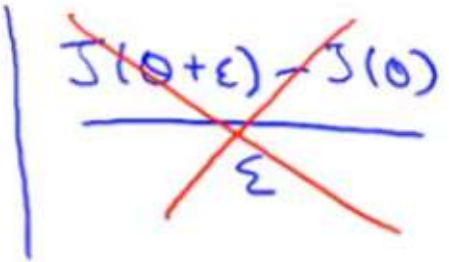


Gradien Cheking

It make sure that, your algor. do the right calcs for Neural Network

$$\frac{d}{d\theta} J(\theta) \approx \frac{J(\theta+\varepsilon) - J(\theta-\varepsilon)}{2\varepsilon} \leftarrow$$

$$\varepsilon = 10^{-4} \leftarrow$$



And solve most of the error while computing

Alos make sure that your opimization work correctly... etc

Implement:

```
gradApprox = (J(theta + EPSILON) - J(theta - EPSILON))
             /(2*EPSILON)
```

Parameter vector θ

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

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

$\theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_i \\ \vdots \\ \theta_n \end{bmatrix}$ $\theta_i + \varepsilon$, $\theta_i - \varepsilon$

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

Check that gradApprox $\approx$ DVec

From backprop.

Implementation Note: Gradient check

- 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.

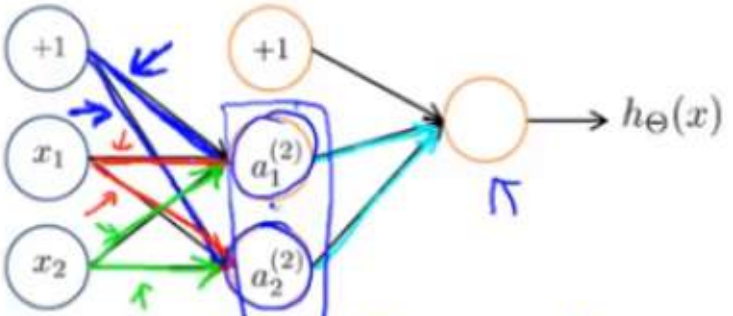
(→ DVec $\delta^{(4)}, \delta^{(3)}, \delta^{(2)}$

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.

Zero initialization

Very bad Idea, since your algor. will always computed the same value, therefore it will by higly redundent and completly not efficent



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

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

Stay the same and this is bad thing

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

$$\Theta_{01}^{(1)} = \Theta_{02}^{(1)}$$

It will reapeat

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

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

Good Idea

Random initialization: Symmetry breaking

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

(i.e. $-\epsilon \leq \Theta_{ij}^{(l)} \leq \epsilon$)

The weights value will be randomized

E.g.

random 10×11 matrix (betw. 0 and 1)

→ `Theta1 = rand(10,11)*(2*INIT_EPSILON) - INIT_EPSILON;`

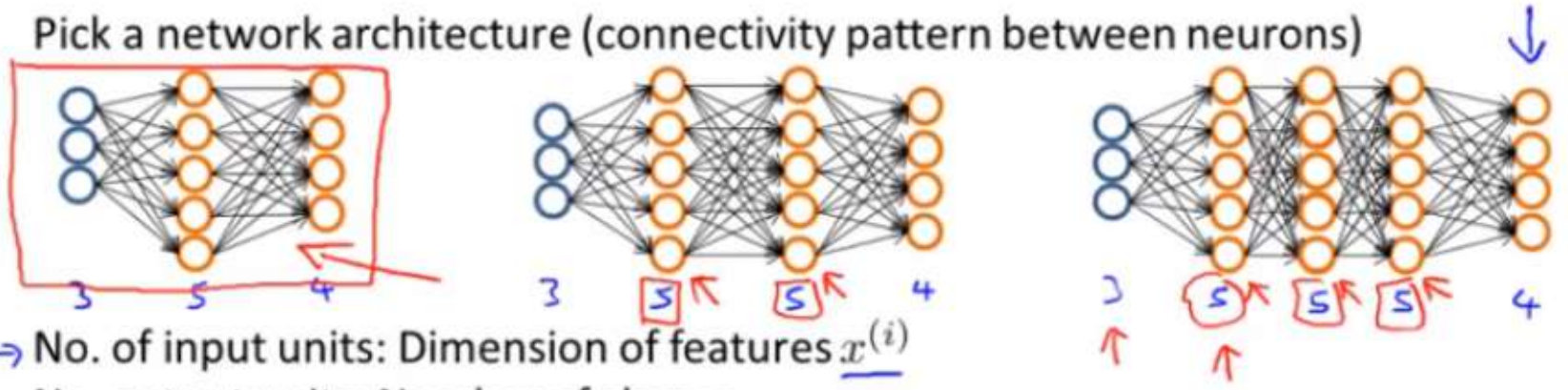
$[-\epsilon, \epsilon]$

→ `Theta2 = rand(1,11)*(2*INIT_EPSILON) - INIT_EPSILON;`

Used as a stadnrt nowadays

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)

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

~~$y = 5$~~

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

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

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

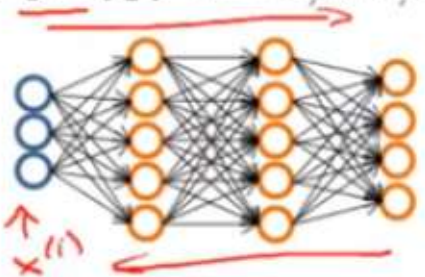
$\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

....

}

...

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



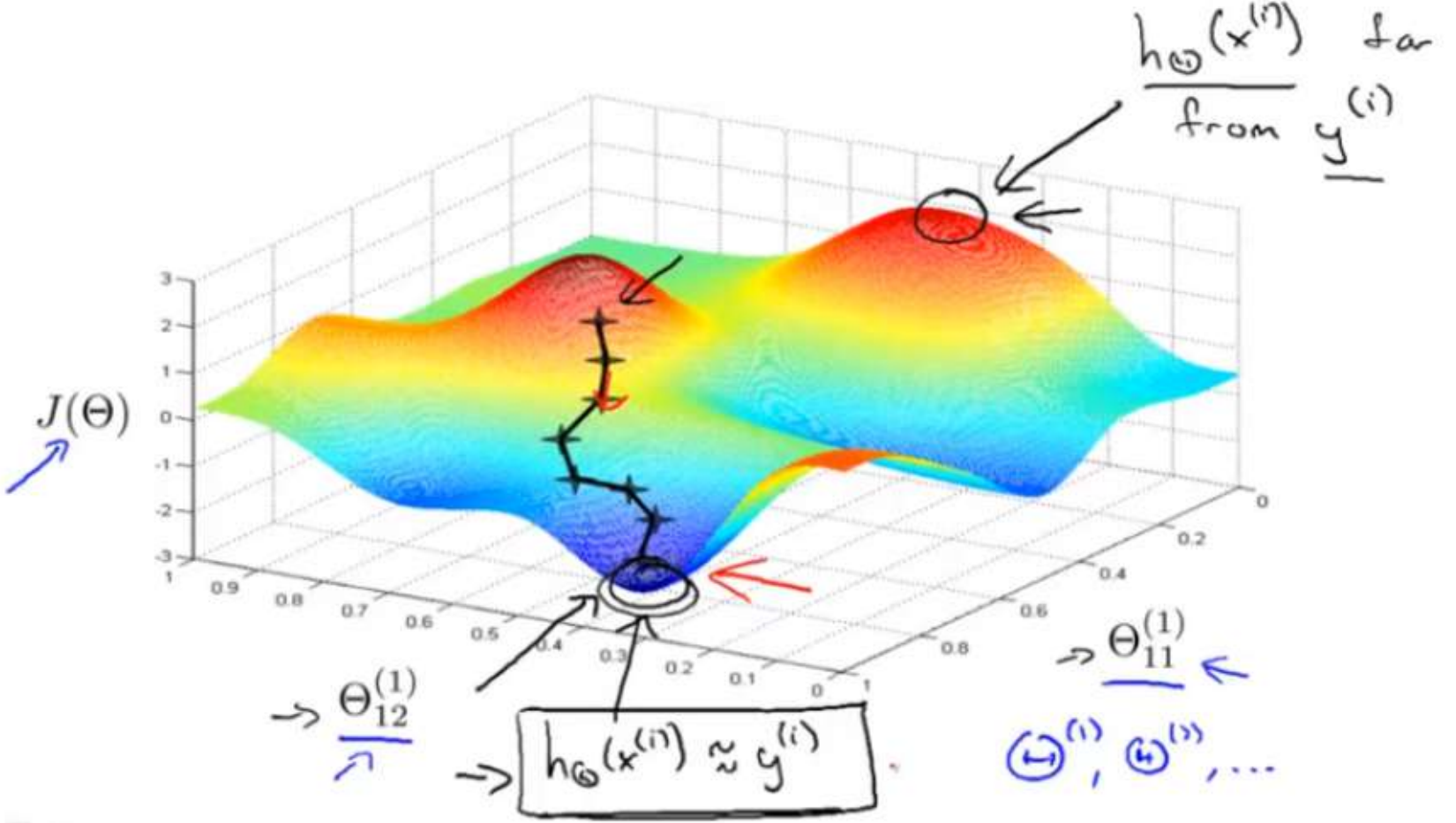
Training a neural network

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

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

$J(\Theta)$

$J(\Theta)$ – non-convex.


$h_\Theta(x^{(i)})$ far from $y^{(i)}$
$J(\Theta)$
$\Theta_{11}^{(1)}$
$\Theta_{12}^{(1)}$
$h_\Theta(x^{(i)}) \approx y^{(i)}$
$\Theta^{(1)}, \Theta^{(2)}, \ldots$

Debugging a learning algorithm: *IMPORTANT: This slide will be revisted after you will read the whole pdf: so on this slide are just show the genaral idea, the true things to consider will be on (fore)last slides*

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

SO IMPORTANT: THE RESULT OF ALL THIS IS LOCATED ON THE LAST SLIDE OF THE PDF FILE SEE THERE

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

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

- Get more training examples
- Try smaller sets of features $x_1, x_2, x_3, \ldots, x_{100}$
- Try getting additional features
- Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, \text{etc.})$
- Try decreasing λ
- Try increasing λ

Machine learning diagnostic:

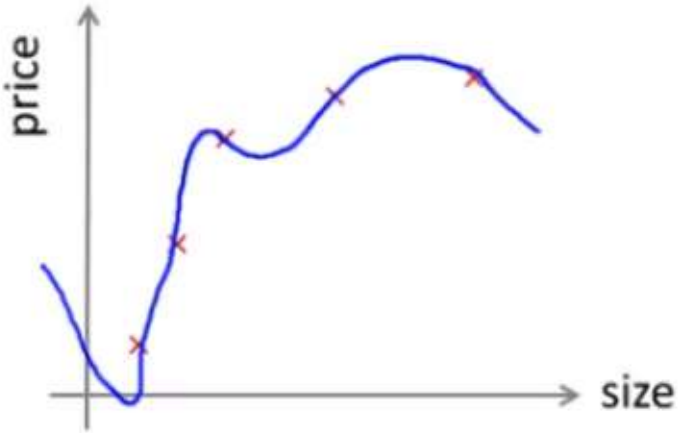
Diagnostic: A test that you can run to gain insight what is/isn't working with a learning algorithm, and gain guidance as to how best to improve its performance.

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

to improve your Algor.

It helps to decide what to do next

Evaluating your hypothesis



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

Fails to generalize to new examples not in training set.

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

Evaluating your hypothesis

Dataset:

	Size	Price	
60%	2104	400	Training set
	1600	330	
	2400	369	
	1416	232	
	3000	540	
	1985	300	
20%	1534	315	Cross validation (CV) set
	1427	199	
20%	1380	212	test set
	1494	243	

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

$(x_{cv}^{(1)}, y_{cv}^{(1)})$
$(x_{cv}^{(2)}, y_{cv}^{(2)})$
$\vdots$
$(x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$

m_{cv} = no. of CV example $(x_{cv}^{(i)}, y_{cv}^{(i)})$

$(x_{test}^{(1)}, y_{test}^{(1)})$
$(x_{test}^{(2)}, y_{test}^{(2)})$
$\vdots$
$(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

m_{test}

Training/testing procedure for linear regression

- Learn parameter θ from training data (minimizing training error $J(\theta)$) — 70%

- Compute test set error:

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

Training/testing procedure for logistic regression

- Learn parameter θ from training data
- Compute test set error:

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

- Misclassification error (0/1 misclassification error):

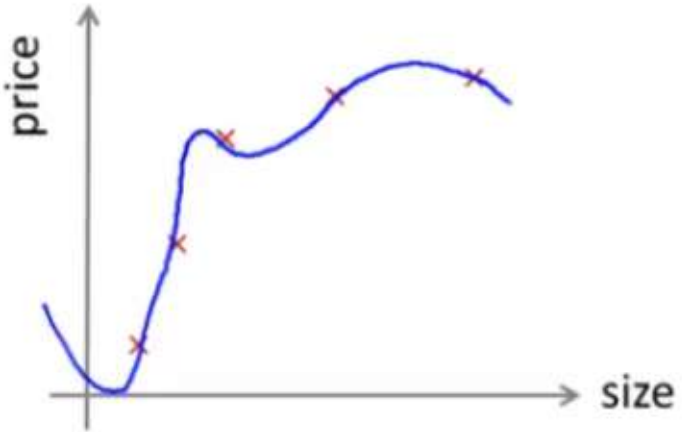
$$err(h_\theta(x), y) = \begin{cases} 1 & \text{if } h_\theta(x) \geq 0.5, \; y = 0 \\ & \text{or if } h_\theta(x) < 0.5, \; y = 1 \\ 0 & \text{otherwise} \end{cases}$$

error function

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

another func

Overfitting example



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

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

Model selection

d = degree of polynomial

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

⋮ ⋮ ⋮

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

The Theta notation is not corralted with the privious means of it, in this slide it is just a set of resultet parametetrs.

Choose $\theta_0 + \ldots \theta_5 x^5$

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

$\Theta_0, \Theta_1 \ldots$

$\Theta^{(5)}$

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.

Evaluating your hypothesis

Dataset:

	Size	Price	
60%	2104	400	Training set
	1600	330	
	2400	369	
	1416	232	
	3000	540	
	1985	300	
20%	1534	315	Cross validation (CV) set
	1427	199	
20%	1380	212	test set
	1494	243	

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

$(x_{cv}^{(1)}, y_{cv}^{(1)})$
$(x_{cv}^{(2)}, y_{cv}^{(2)})$
$\vdots$
$(x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$

m_{cv} = no. of CV example $(x_{cv}^{(i)}, y_{cv}^{(i)})$

$(x_{test}^{(1)}, y_{test}^{(1)})$
$(x_{test}^{(2)}, y_{test}^{(2)})$
$\vdots$
$(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

m_{test}

Train/validation/test error

Training error:

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

$J(\theta)$

Cross Validation error:

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

Test error:

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

Model selection

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

⋮

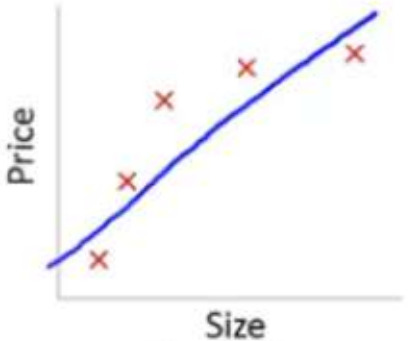
10. $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_{10} x^{10}$ → $\theta^{(10)}$ → $J_{cv}(\theta^{(10)})$

$d = 4$

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

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

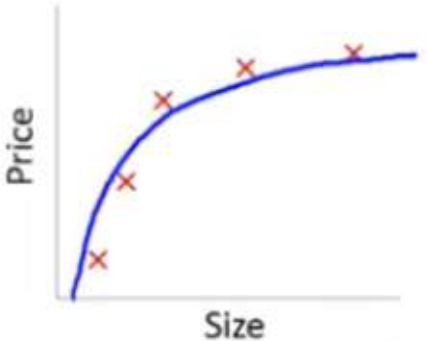
Bias/variance



$\theta_0 + \theta_1 x$

High bias
(underfit)

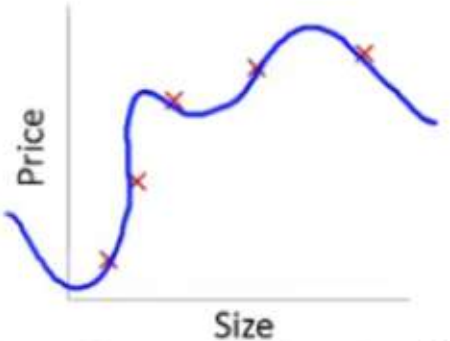
$d = 1$



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

"Just right"

$d = 2$



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

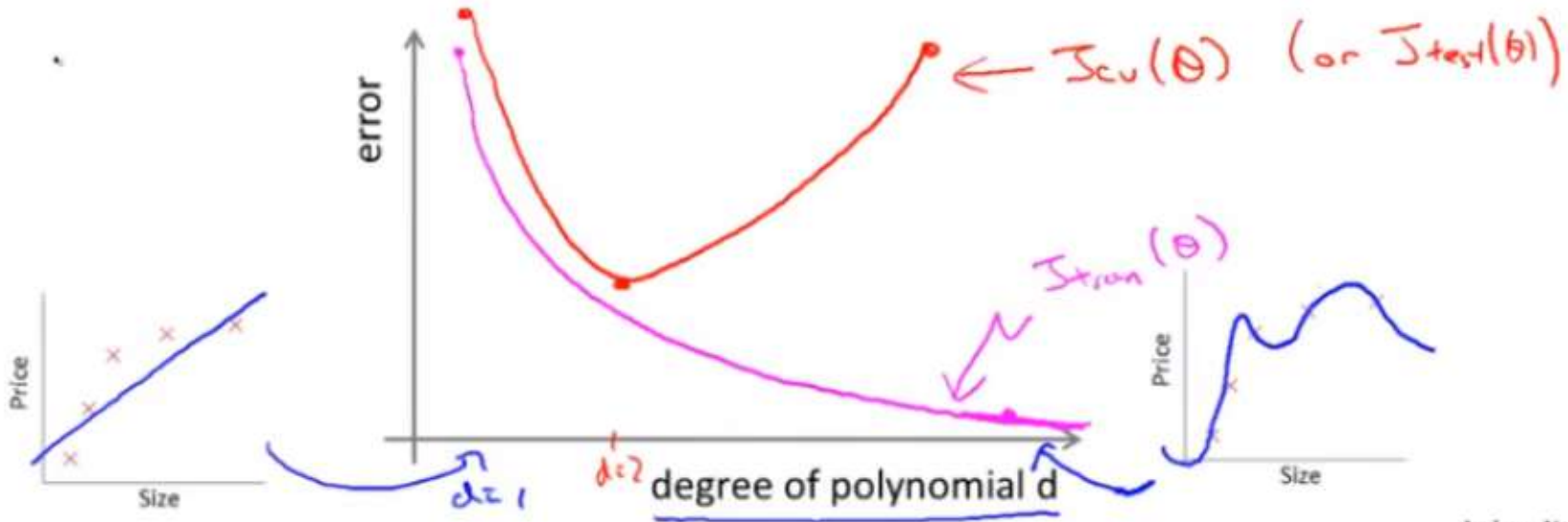
High variance
(overfit)

$d = 4$

Bias/variance

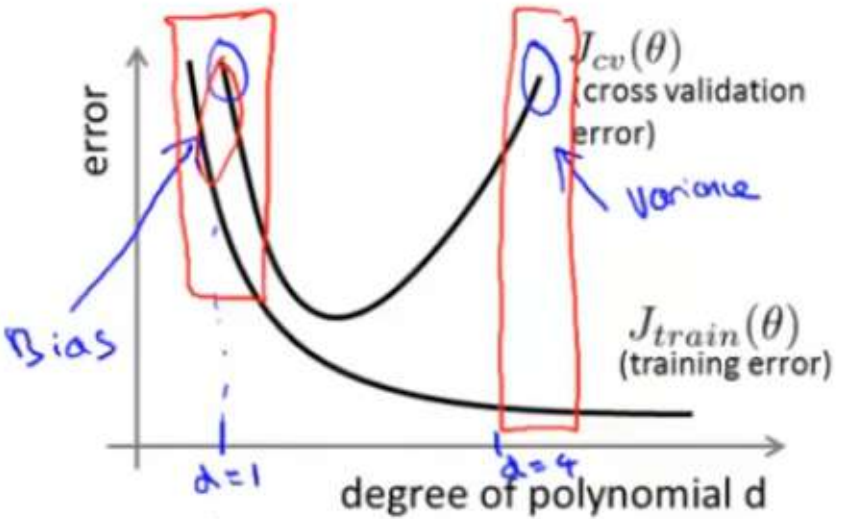
Training error: $J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

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



Diagnosing bias vs. variance

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



Bias (underfit):

→ $J_{train}(\theta)$ will be high

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

Variance (overfit):

→ $J_{train}(\theta)$ will be low

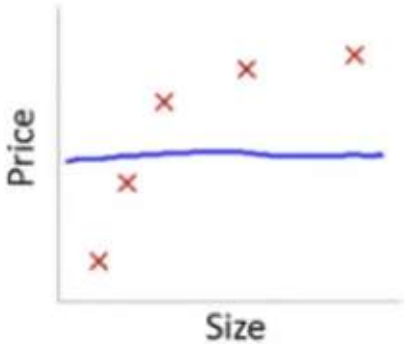
$J_{cv}(\theta) >> J_{train}(\theta)$

Much greater than >>

Linear regression with regularization

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

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

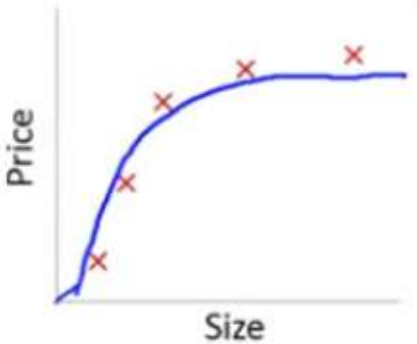


Large λ

High bias (underfit)

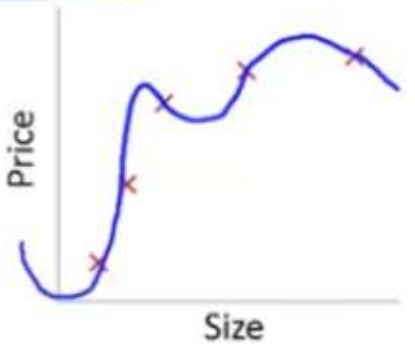
$\lambda = 10000.\ \theta_1 \approx 0, \theta_2 \approx 0, \ldots$

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



Intermediate λ

"Just right"



Small λ

High variance (overfit)

$\lambda = 0$

Andrew

Choosing the regularization parameter λ

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

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

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

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

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

Without Regularization / Adding λ sets $J(\theta)$

J_{train}
J_{cv}
J_{test}

Choosing the regularization parameter λ

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

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

The Thetas notation are not correlted with others meanigs, in this slide it is just, the set of resulted parameters

1. Try $\lambda = 0$ → $\min_\theta J(\theta) \rightarrow \theta^{(1)} \rightarrow J_{cv}(\theta^{(1)})$
2. Try $\lambda = 0.01$ → $\min_\theta J(\theta) \rightarrow \theta^{(2)} \rightarrow J_{cv}(\theta^{(2)})$
3. Try $\lambda = 0.02$ → $\theta^{(3)} \rightarrow J_{cv}(\theta^{(3)})$
4. Try $\lambda = 0.04$
5. Try $\lambda = 0.08$ → $\theta^{(5)}$ $J_{cv}(\theta^{(5)})$

⋮

12. Try $\lambda = 10$ → $\theta^{(12)} \rightarrow J_{cv}(\theta^{(12)})$

↑ 10.24

Pick (say) $\theta^{(5)}$. Test error: $J_{test}(\theta^{(5)})$

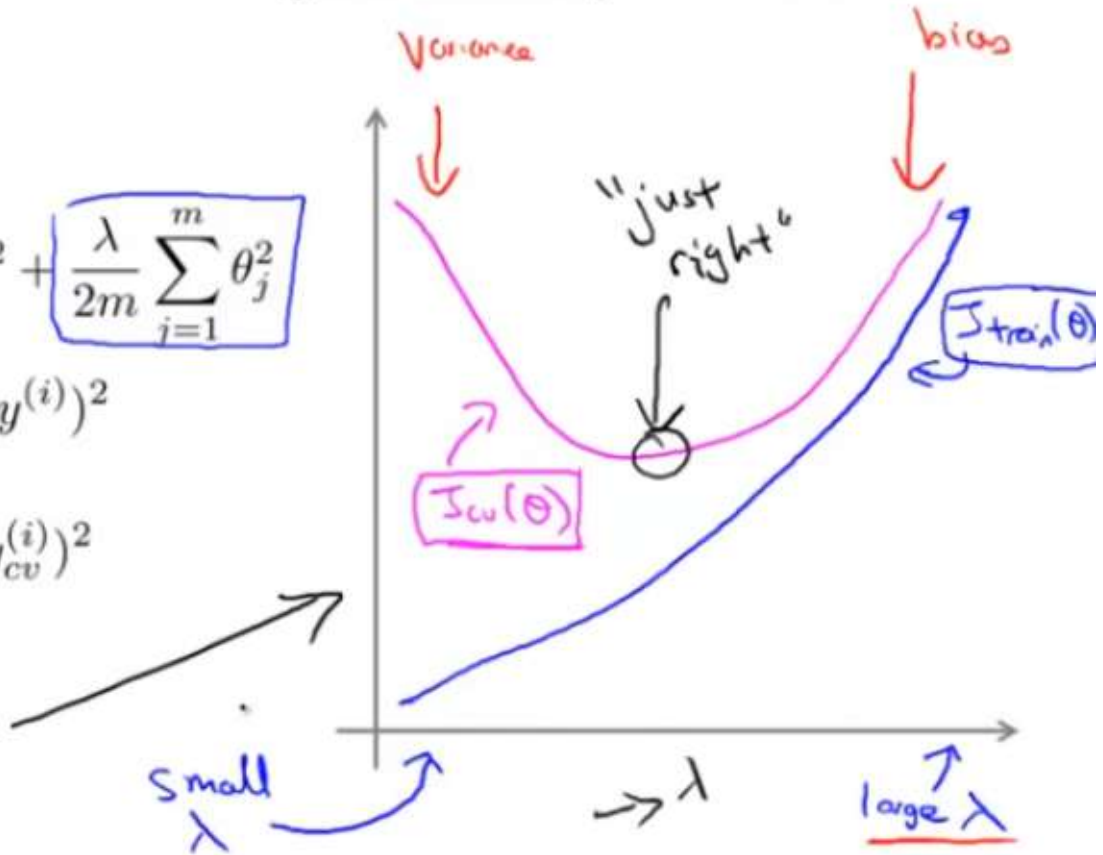
Andrew N

Bias/variance as a function of the regularization parameter λ

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

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

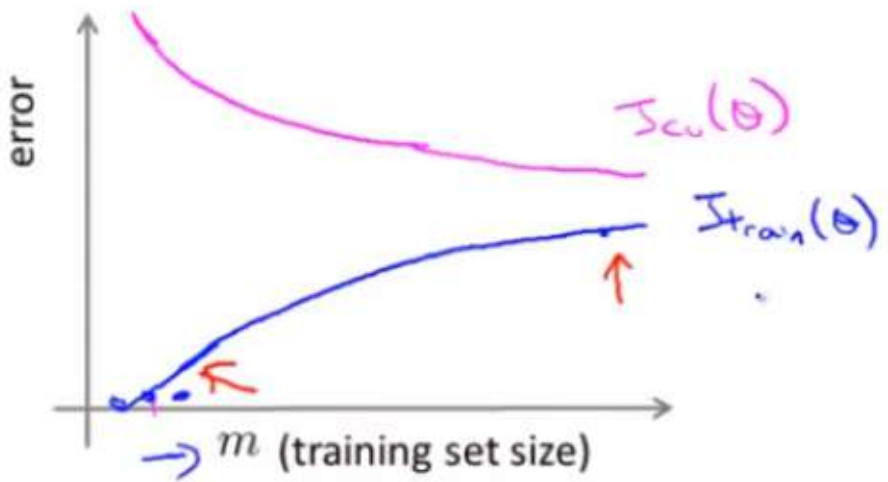
$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$



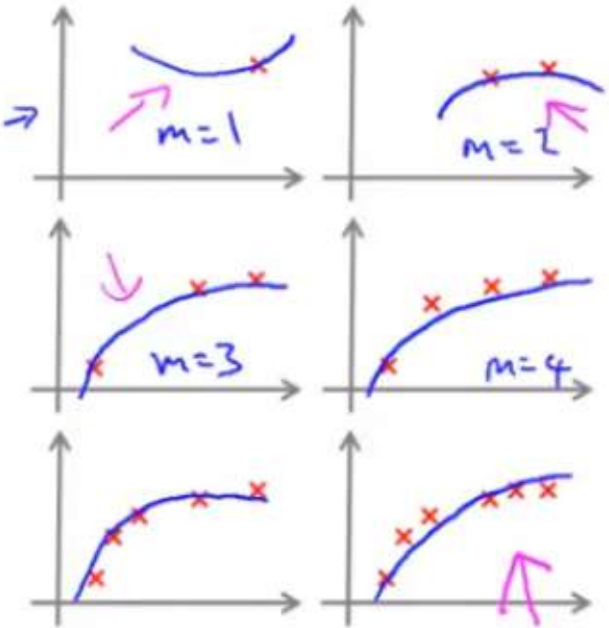
Learning curves

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

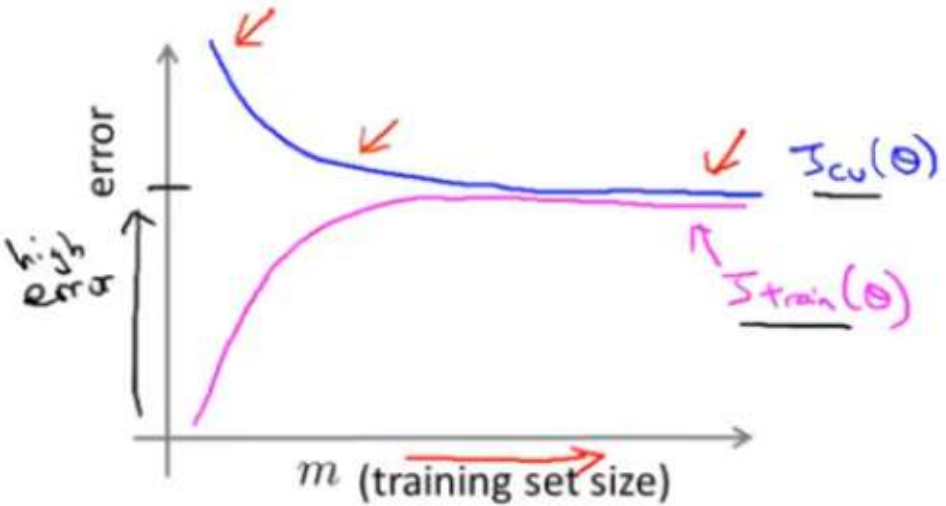
$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$



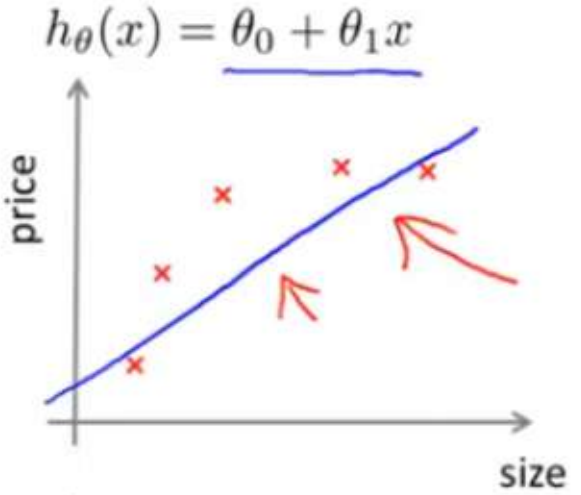
$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2$$

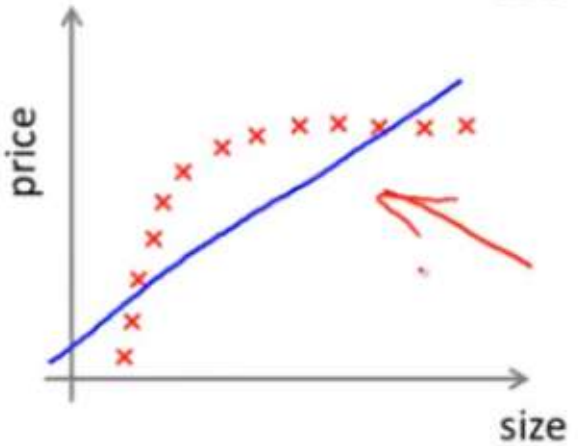


High bias



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





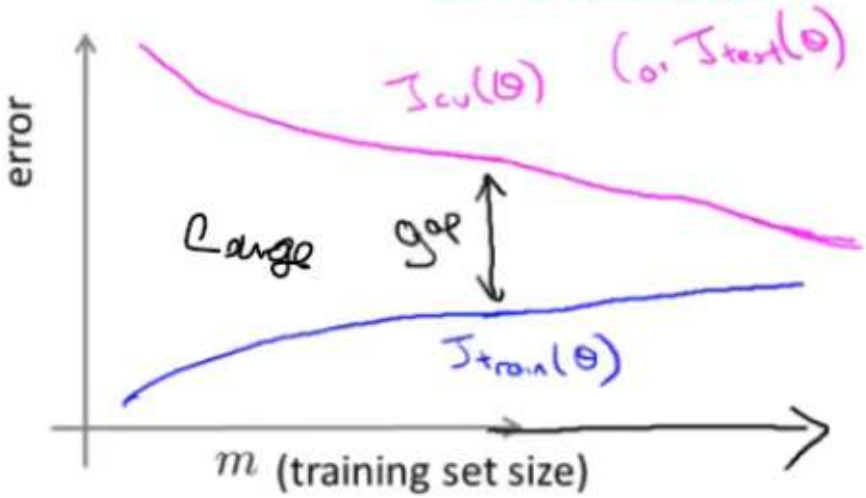
FAZIT ====>
of Learning Cureves

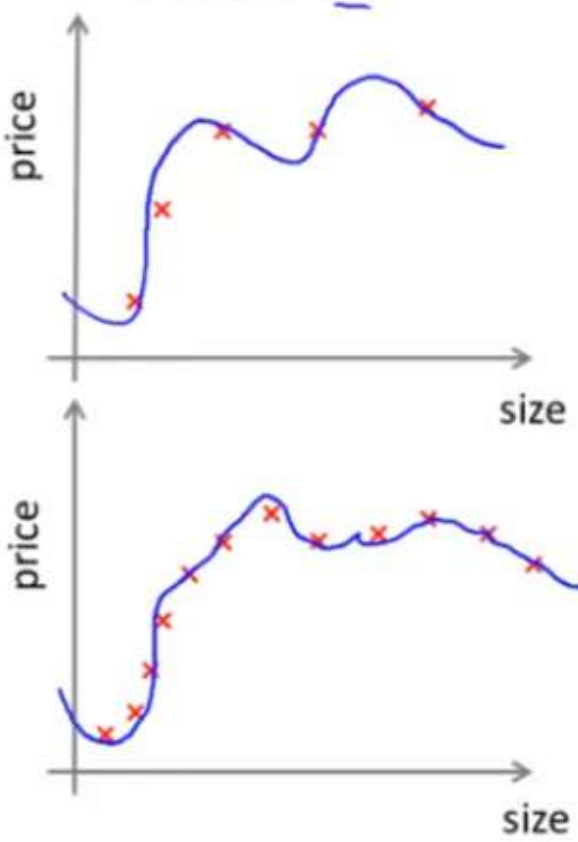
Do ALWAYS plot the learning Curves
It is worth it, in deed, it save you a lot of time and prevent you that you may start on workint on a probelm that is not the reatl threat, that cause the bad preformence of algot.

High variance

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

(and small λ)





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

Debugging a learning algorithm:

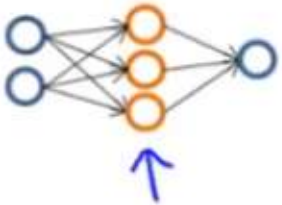
Suppose you have implemented regularized linear regression to predict housing prices. However, when you test your hypothesis in a new set of houses, you find that it makes unacceptably large errors in its prediction. What should you try next?

VERY VERY IMPORTANT:
SEE THE SLIDE AND CONSIDER YOU ALGOR., THEN YOU CAN CHOOSE WAHT DO TO NEXT TO IMPROVE YOU ALGOR.

- 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

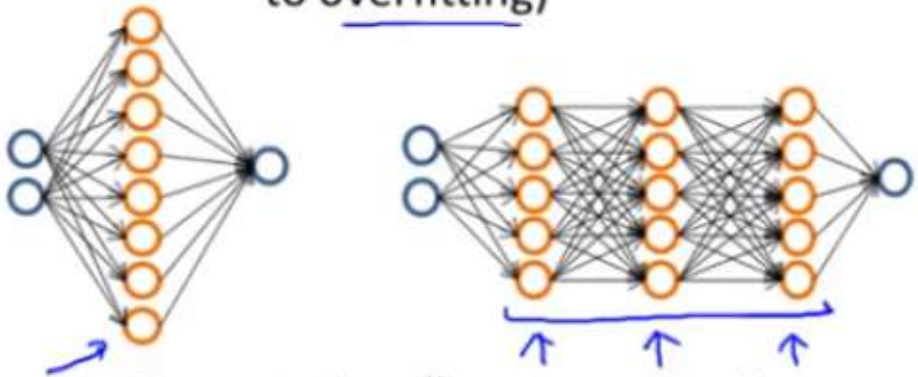
Neural networks and overfitting

"Small" neural network
(fewer parameters; more prone to underfitting)



Computationally cheaper

"Large" neural network
(more parameters; more prone to overfitting)



Computationally more expensive.

Use regularization (λ) to address overfitting.

$J_{c_0}(\theta)$

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, ...

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

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

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.)

Recommended approach

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

Error Analysis

$m_{CV} =$ 500 examples in cross validation set

Algorithm misclassifies 100 emails.

Manually examine the 100 errors, and categorize them based on:

(i) What type of email it is — pharma, replica, steal passwords, ...

(ii) What cues (features) you think would have helped the algorithm classify them correctly.

Pharma: 12

Replica/fake: 4

Steal passwords: 53

Other: 31

Deliberate misspellings: 5
(m0rgage, med1cine, etc.)

Unusual email routing: 16

Unusual (spamming) punctuation: 32

The importance of numerical evaluation

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

Can use "stemming" software (E.g. "Porter stemmer")

universe/university.

Error analysis may not be helpful for deciding if this is likely to improve performance. Only solution is to try it and see if it works.

Need numerical evaluation (e.g., cross validation error) of algorithm's performance with and without stemming.

Without stemming: 5% error With stemming: 3% error

Distinguish upper vs. lower case (Mom/mom): 3.2%

Cancer classification example

Train logistic regression model $h_\theta(x)$. ($y = 1$ if cancer, $y = 0$ otherwise)

Find that you got 1% error on test set.

(99% correct diagnoses)

Only 0.50% of patients have cancer.

→ skewed classes.

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

→ 0.5% error

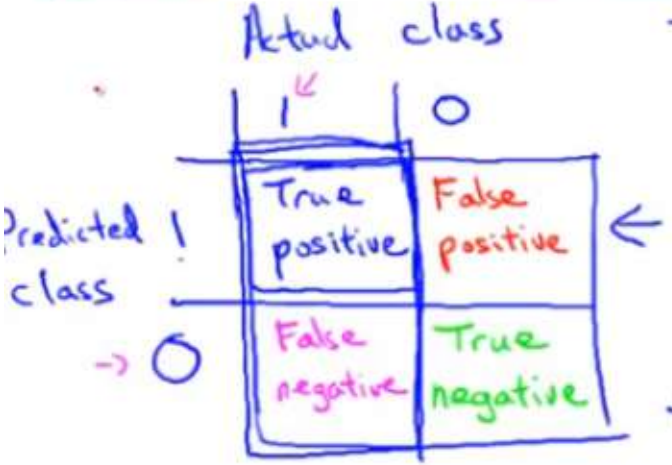
→ 99.2% accuracy (0.8% error)

→ 99.5% accuracy (0.5% error)

Precision/Recall

So Do it

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



→ **Precision** (if both high it is good

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

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

→ **Recall**

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

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

$y = 0$

recall = 0

Trading off precision and recall

- Logistic regression: $0 \leq h_\theta(x) \leq 1$

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

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

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

  → Higher precision, lower recall.

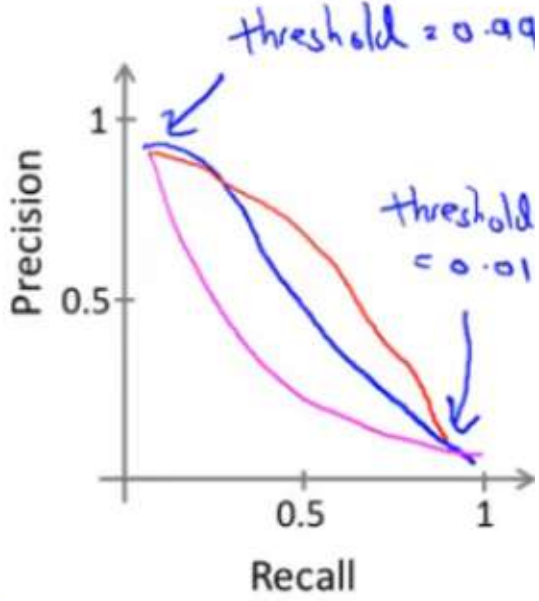
- Suppose we want to avoid missing too many cases of cancer (avoid false negatives).

  → Higher recall, lower precision.

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

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

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



F_1 Score (F score)

How to compare precision/recall numbers?

	Precision(P)	Recall (R)	Average	F_1 Score
Algorithm 1	0.5	0.4	0.45	0.444
Algorithm 2	0.7	0.1	0.4	0.175
Algorithm 3	0.02	1.0	0.51	0.0392

Predict $y=1$ all the time

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

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

$P=0$ or $R=0$ $\Rightarrow$ F-score $=0$. (worst)

$P=1$ and $R=1$ $\Rightarrow$ F-score $=1$ (Best)

Designing a high accuracy learning system

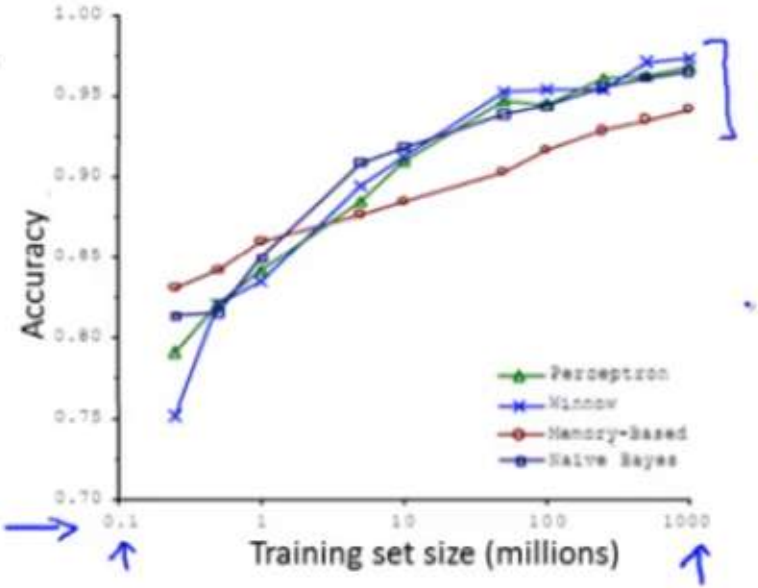
E.g. Classify between confusable words.

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

For breakfast I ate two eggs.

Algorithms

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



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

Banko and Brill, 2001]

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?

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.

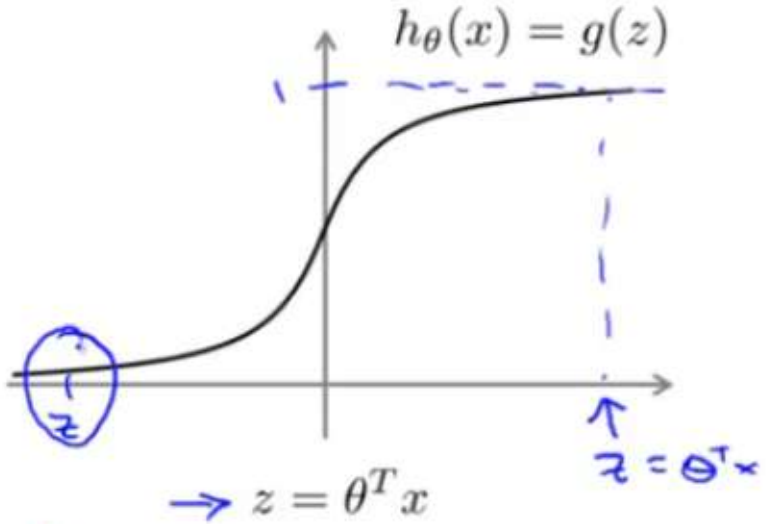
Use a very large training set (unlikely to overfit) low variance ←

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

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

Alternative view of logistic regression

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



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

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

Alternative view of logistic regression

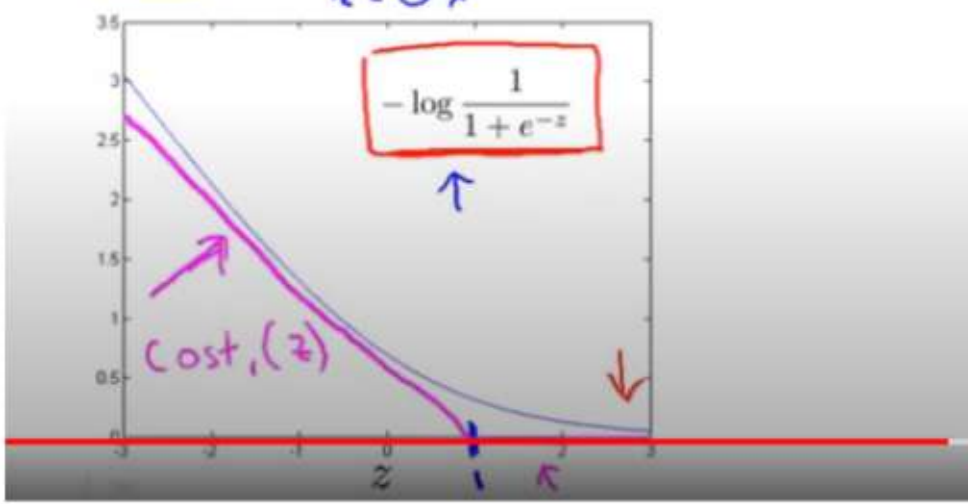
(x, y)

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

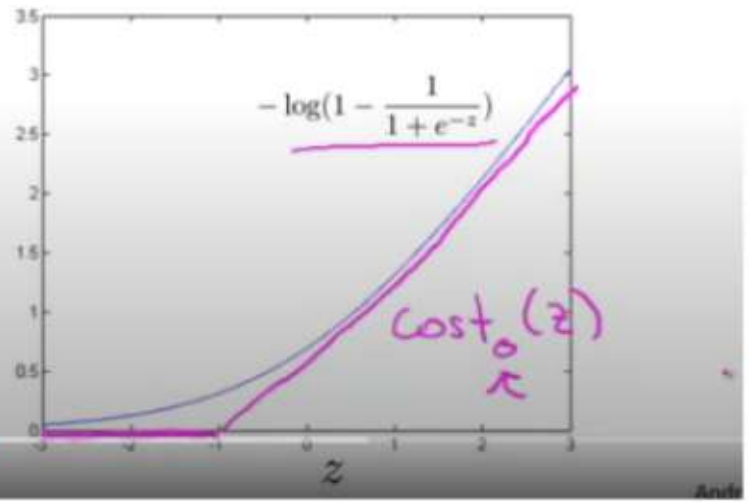
$$= -y \log \frac{1}{1+e^{-\theta^T x}} - (1-y)\log\left(1 - \frac{1}{1+e^{-\theta^T x}}\right)$$

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

$z = \theta^T x$

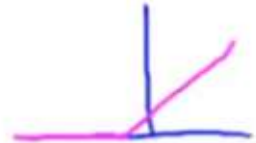


If $y = 0$ (want $\theta^T x \ll 0$):



Support vector machine

Logistic regression:



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

A B

Support vector machine:

$$\min_\theta \cancel{\frac{1}{m}} C \sum_{i=1}^{m} y^{(i)} \text{cost}_1(\theta^T x^{(i)}) + (1-y^{(i)}) \text{cost}_0(\theta^T x^{(i)}) + \cancel{\frac{1}{2m}} \sum_{i=0}^{n} \theta_j^2$$

$\min_u \left((u-5)^2+1\right)$ → $u=5$ ↕ $\min_u 10(u-5)^2+10$ → $u=5$ (×10)

$A + \lambda B$ ← $C = \frac{1}{\lambda}$

$CA + B$ ←

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

SVM hypothesis

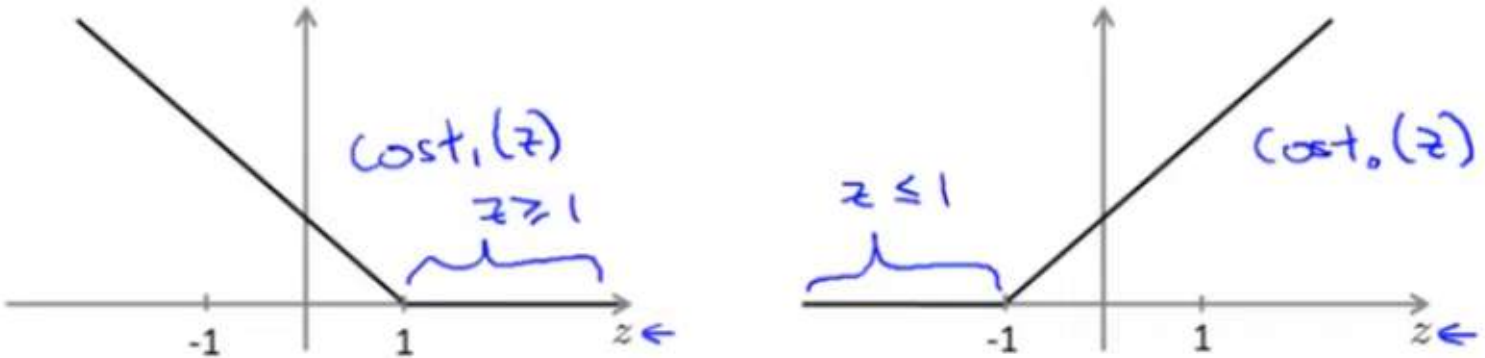
$$\rightarrow \min_{\theta} C \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

Hypothesis:

$$h_\theta(x) \begin{cases} 1 & \text{if } \theta^T x \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

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

$C = 100{,}000$

SVM Decision Boundary

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

$= 0$

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

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

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

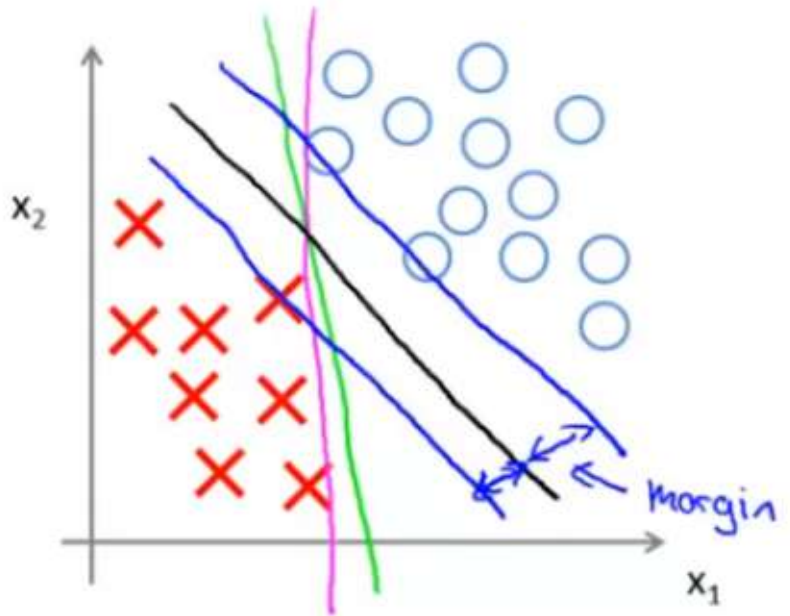
$\theta^T x^{(i)} \leq -1$

$\min_\theta \; C \cdot 0 + \frac{1}{2}\sum_{j=1}^{n}\theta_j^2$

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

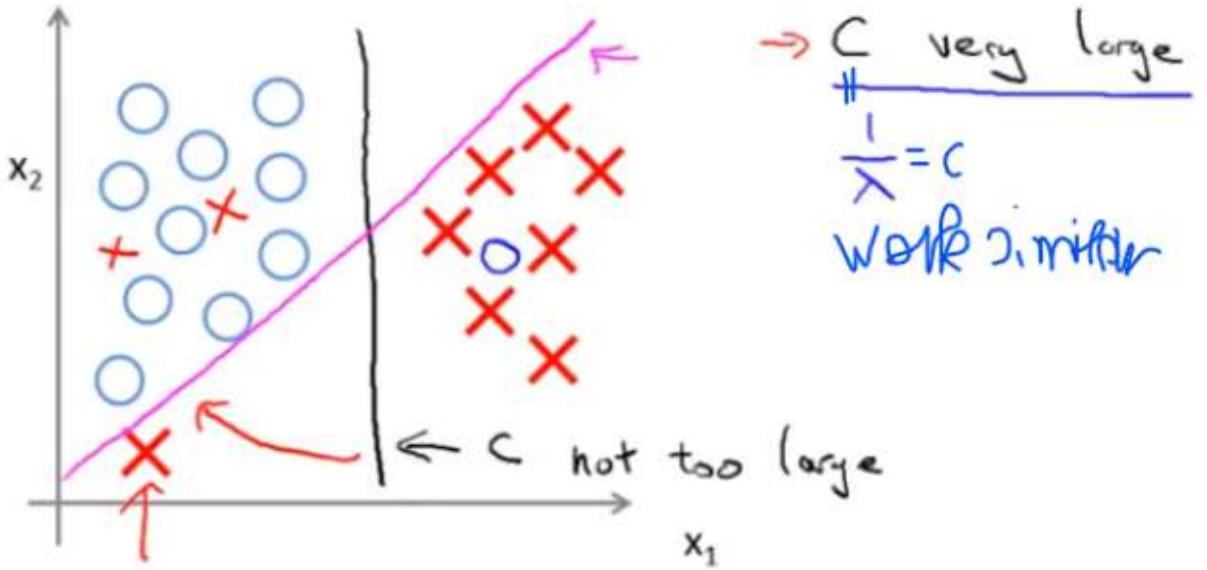
$\theta^T x^{(i)} \leq -1$ if $y^{(i)} = 0$.

SVM Decision Boundary: Linearly separable case

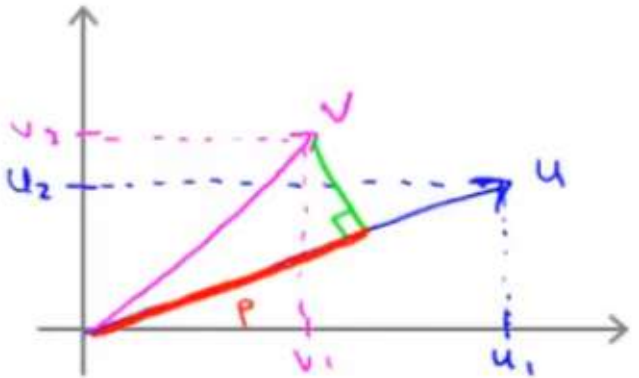


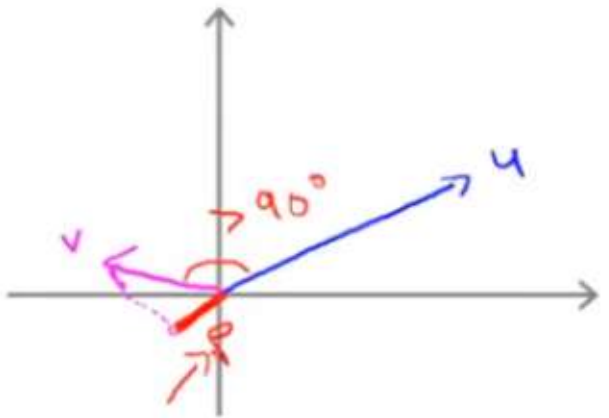
Large margin classifier

Large margin classifier in presence of outliers



Vector Inner Product





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

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

$\|u\|$ = length of vector u

$= \sqrt{u_1^2 + u_2^2} \in \mathbb{R}$

p = length of projection of v onto u

Signed

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

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

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

$p < 0$

SVM Decision Boundary

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

$$= \|\theta\|$$

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

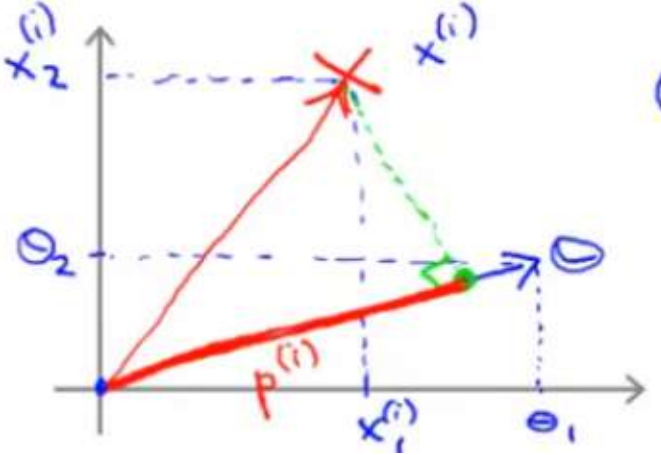
$$\theta^T x^{(i)} \leq -1 \quad \text{if } y^{(i)} = 0$$

Simplication: $\theta_0 = 0$. $n=2$

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

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

$u^T v$



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

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

Andrew N

SVM Decision Boundary

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

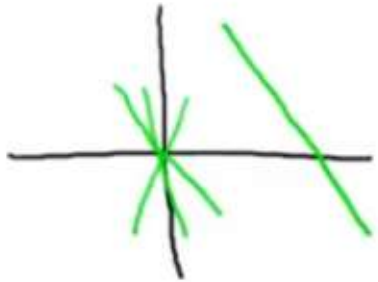
s.t. $p^{(i)} \cdot \|\theta\| \geq 1$ if $y^{(i)} = 1$

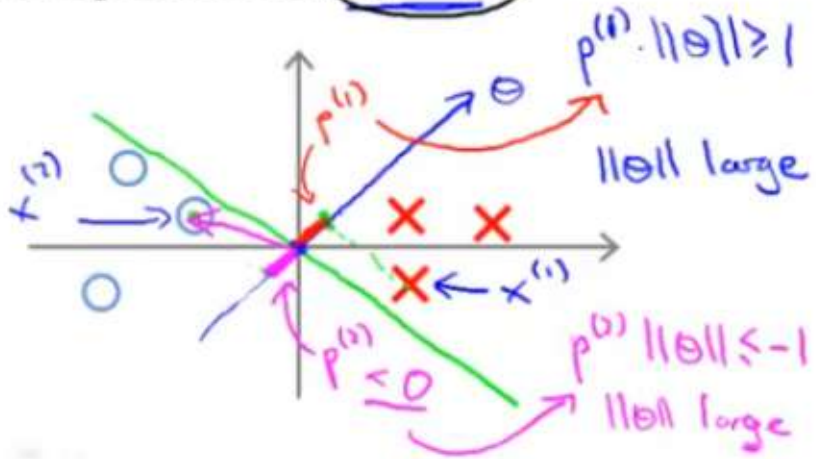
$p^{(i)} \cdot \|\theta\| \leq -1$ if $y^{(i)} = 1$

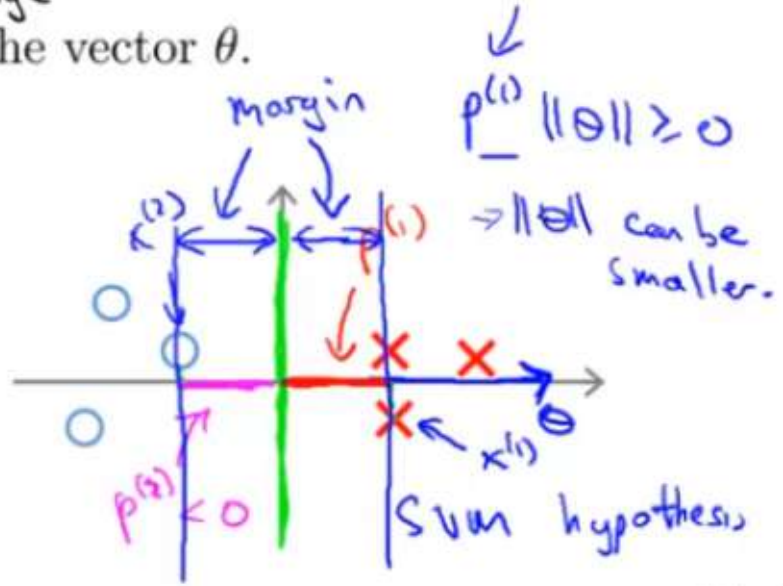
} C very large

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

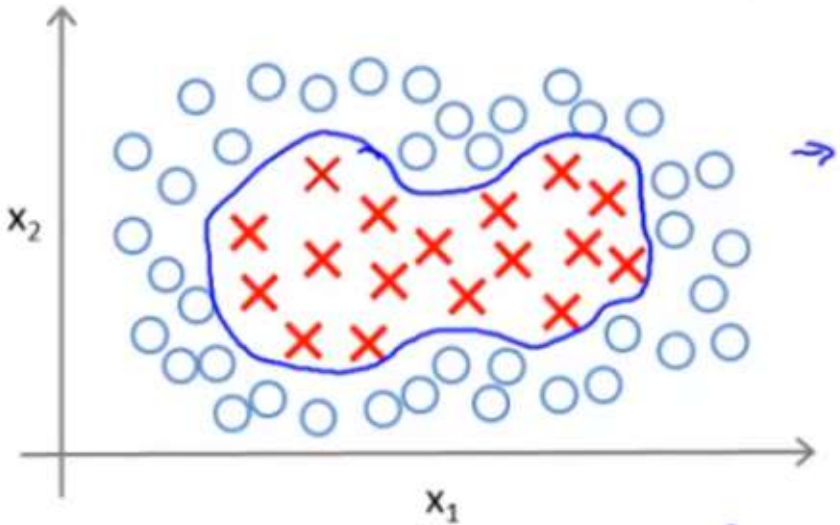
Simplification: $\theta_0 = 0$







Non-linear Decision Boundary



Predict $y = 1$ if

$$\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 + \theta_5 x_2^2 + \cdots \geq 0$$

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

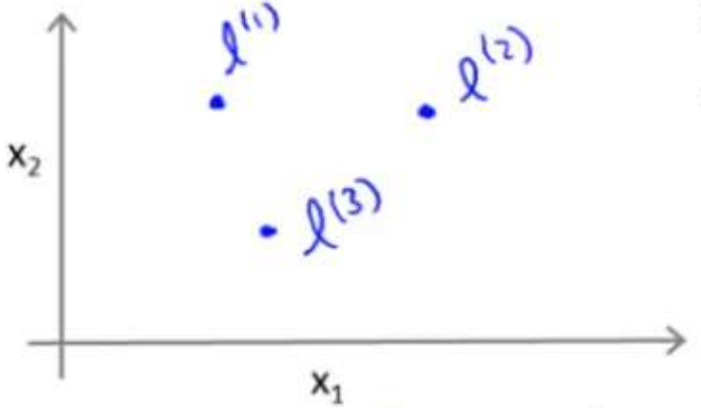
$\rightarrow \theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 + \cdots$

$f_1 = x_1, \quad f_2 = x_2, \quad f_3 = x_1 x_2, \quad f_4 = x_1^2, \quad f_5 = x_2^2, \ldots$

Is there a different / better choice of the features $f_1, f_2, f_3, \ldots$?

Andrew N

Kernel



Given x, compute new feature depending on proximity to landmarks $l^{(1)}, l^{(2)}, l^{(3)}$

$\|w\|$

$\|x - l^{(1)}\|^2$

Given x:

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

Small sigma not

$$f_2 = \text{similarity}(x, l^{(1)}) = \exp\left(-\frac{\|x - l^{(2)}\|^2}{2\sigma^2}\right)$$

Small sigma notation

$$f_3 = \text{similarity}(x, l^{(3)}) = \exp(\dots)$$

Kernel (Gaussian kernels)

$k(x, l^{(i)})$

Kernels and Similarity

Component wise distance between vector "x" and vector "l"

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

The landmars do define new fetuers as you see below

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

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

lower case sigam

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

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

lower case sigam

$l^{(1)} \rightarrow f_1$

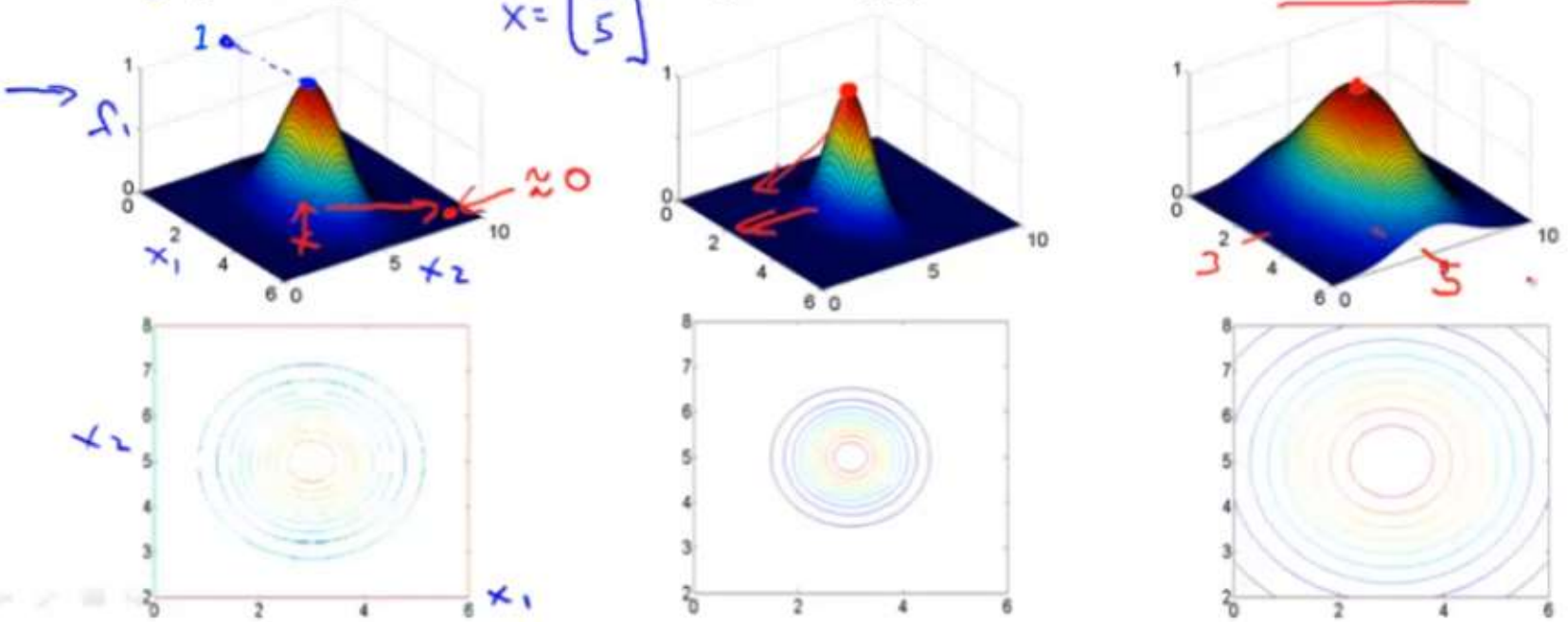
$l^{(2)} \rightarrow f_2$

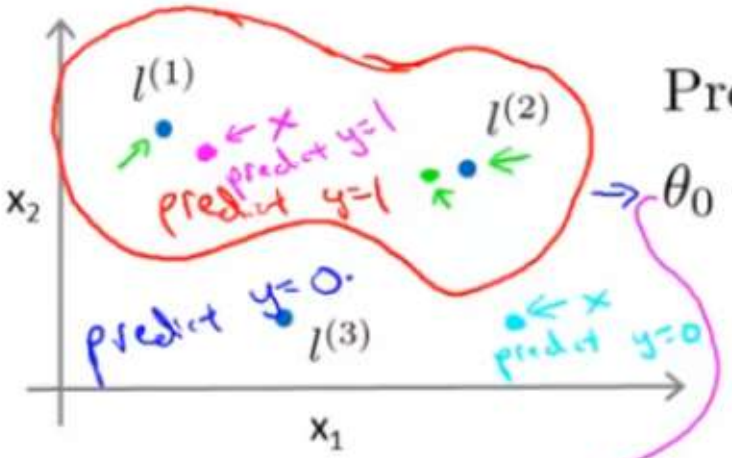
$l^{(3)} \rightarrow f_3$.

Example:

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

$\sigma^2 = 1$ $\quad$ $\sigma^2 = 0.5$ $\quad$ $\sigma^2 = 3$





Predict "1" when

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$$

$\theta_0 = -0.5$, $\theta_1 = 1$, $\theta_2 = 1$, $\theta_3 = 0$

$f_1 \approx 1$, $f_2 \approx 0$, $f_3 \approx 0$.

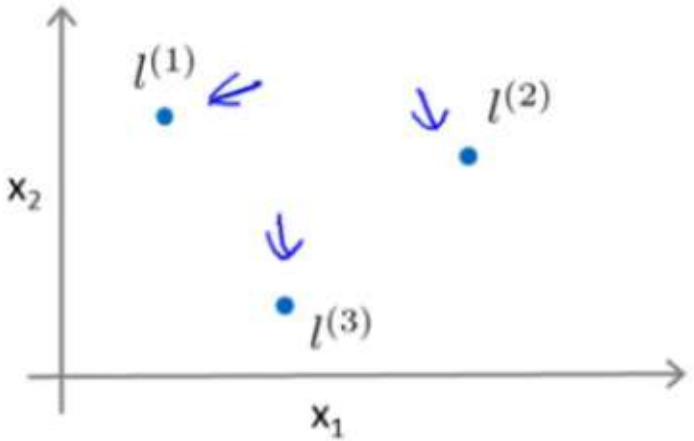
$\theta_0 + \theta_1 \times 1 + \theta_2 \times 0 + \theta_3 \times 0$

$= -0.5 + 1 = 0.5 \geq 0$

$f_1, f_2, f_3 \approx 0$

$\rightarrow \theta_0 + \theta_1 f_1 + \dots \approx -0.5 < 0$

Choosing the landmarks



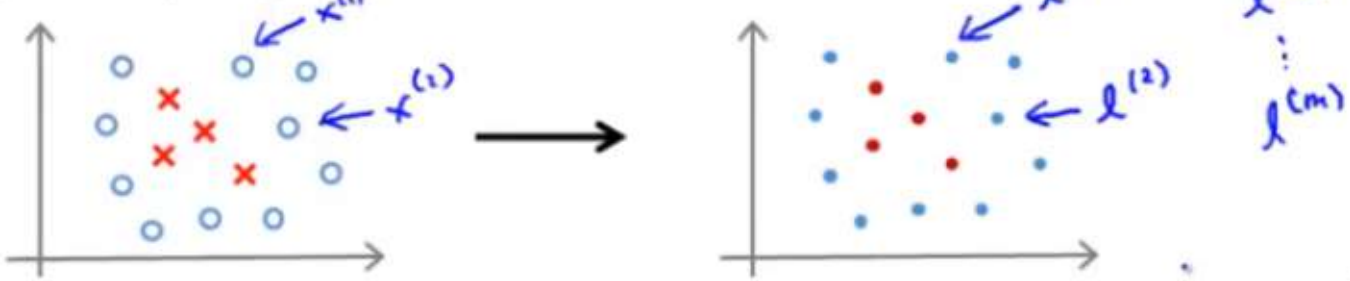
Given x:

$$f_i = \text{similarity}(x, l^{(i)})$$

$$= \exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right)$$

Predict $y = 1$ if $\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$

Where to get $l^{(1)}, l^{(2)}, l^{(3)}, \ldots$?



SVM with Kernels

→ Given $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$,

→ choose $l^{(1)} = x^{(1)}, l^{(2)} = x^{(2)}, \ldots, l^{(m)} = x^{(m)}$.

Given example x:

→ $f_1 = \text{similarity}(x, l^{(1)})$

→ $f_2 = \text{similarity}(x, l^{(2)})$

$\ldots$

$x^{(i)}$

$f = \begin{bmatrix} f_0 \\ f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix}$ $f_0 = 1$

For training example $(x^{(i)}, y^{(i)})$:

$x^{(i)} \rightarrow$ $\begin{bmatrix} f_1^{(i)} \\ f_2^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix}$

$f_1^{(i)} = \text{sim}(x^{(i)}, l^{(1)})$

Similarity $f_2^{(i)} = \text{sim}(x^{(i)}, l^{(2)})$

$f_i^{(i)} = \text{sim}(x^{(i)}, l^{(i)}) = \exp\left(-\frac{0}{2\sigma^2}\right) = 1$

$f_m^{(i)} = \text{sim}(x^{(i)}, l^{(m)})$

One of the f_i will be equal to 1 if $x \approx l$

$x^{(i)} \in \mathbb{R}^{n+1}$ (or $\mathbb{R}^n$)

$f^{(i)} = \begin{bmatrix} f_0^{(i)} \\ f_1^{(i)} \\ \vdots \\ f_m^{(i)} \end{bmatrix}$ $f_0^{(i)} = 1$

SVM with Kernels

the "+1" in dim often refrerse to addional "1ts" vector or for the zero^th index implementation etc.

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

→ Predict "y=1" if $\theta^T f \geq 0$; $\Theta_0 f_0 + \Theta_1 f_1 + \cdots + \Theta_m f_m$

Training:

$n = m$

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

$\rightarrow \Theta_0$

$\Theta^T x^{(i)} \rightarrow \Theta^T f^{(i)}$

Default

$- \sum_j \Theta_j^2 = \Theta^T \Theta \leftarrow \Theta = \begin{bmatrix} \Theta_1 \\ \vdots \\ \Theta_m \end{bmatrix}$ (ignore Θ_0)

$\leftarrow \|\Theta\|^2$

$M = 10{,}000$

$- \Theta^T M \Theta$

And if you wonder, it is possiable to use the Kernels also in a logistic computation, but the idea of kernels do not generelize well with algor. like logistic regression etc. therefore if you use the kernels in logistic or in other function it will just run to slow, so: => IT DO NOT WORTH THE EFFORT

Andrew N

SVM parameters:

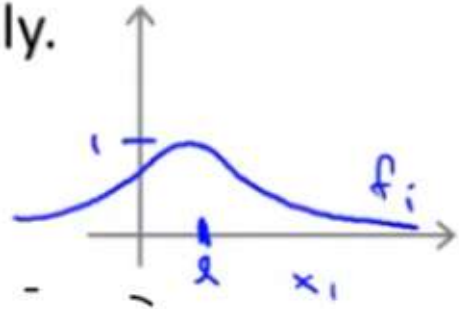
C ($= \frac{1}{\lambda}$). → Large C: Lower bias, high variance. → (small λ)

→ Small C: Higher bias, low variance. → (large λ)

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

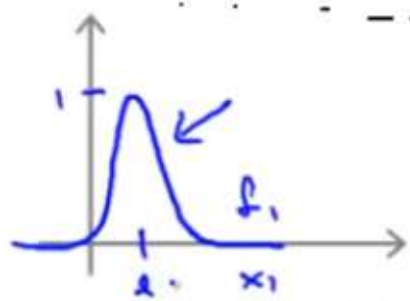
→ Higher bias, lower variance.

$\exp\left(-\frac{\|x-l^{(i)}\|^2}{2\sigma^2}\right)$



Small σ^2: Features f_i vary less smoothly.

Lower bias, higher variance.



Use SVM software package (e.g. liblinear, libsvm, ...) to solve for parameters θ.

Need to specify:

→ Choice of parameter C.

Choice of kernel (similarity function):

E.g. No kernel ("linear kernel")

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

$\theta_0 + \theta_1 x_1 + \cdots + \theta_n x_n \geq 0$

→ n large, m small

$x \in \mathbb{R}^{n+1}$

→ Gaussian kernel:

$$f_i = \exp\left(-\frac{||x - l^{(i)}||^2}{2\sigma^2}\right), \text{ where } l^{(i)} = x^{(i)}.$$

Need to choose σ^2.

$x \in \mathbb{R}^n$, n small and/or m large

Kernel (similarity) functions:

$x^{(i)}$ $\quad l^{(j)} = x^{(j)}$

```
function f = kernel(x1,x2)
```

$$f = \exp\left(-\frac{\|x1 - x2\|^2}{2\sigma^2}\right)$$

f_i

$$x \rightarrow \begin{matrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{matrix}$$

```
return
```

→ Note: Do perform feature scaling before using the Gaussian kernel.

$x \in \mathbb{R}^n$

$\|x - l\|^2$ $\quad v = x - l$

$\|v\|^2 = v_1^2 + v_2^2 + \cdots + v_n^2$

$= (x_1 - l_1)^2 + (x_2 - l_2)^2 + \cdots + (x_n - l_n)^2$

1000 feet^2 $\quad$ 1-5 bedrooms

Other choices of kernel

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

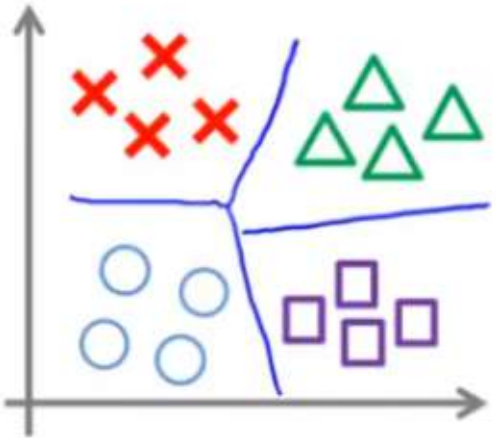
Many off-the-shelf kernels available:

- Polynomial kernel: $k(x, l) = (x^T l)^2$, $(x^T l)^3$, $(x^T l + 1)^3$, $(x^T l + 5)^4$, $(x^T l + \text{constant})^{\text{degree}}$ ↑ parameter of fun

- More esoteric: String kernel, chi-square kernel, histogram intersection kernel, ...

$sim(x, l)$

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 $\theta^{(1)}, \theta^{(2)}, \ldots, \theta^{(K)}$

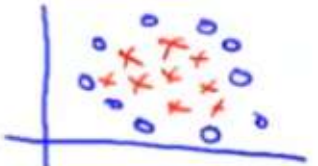
Pick class i with largest $(\theta^{(i)})^T x$

$y=1$ $y=2$ $\ldots$ $\theta=K$

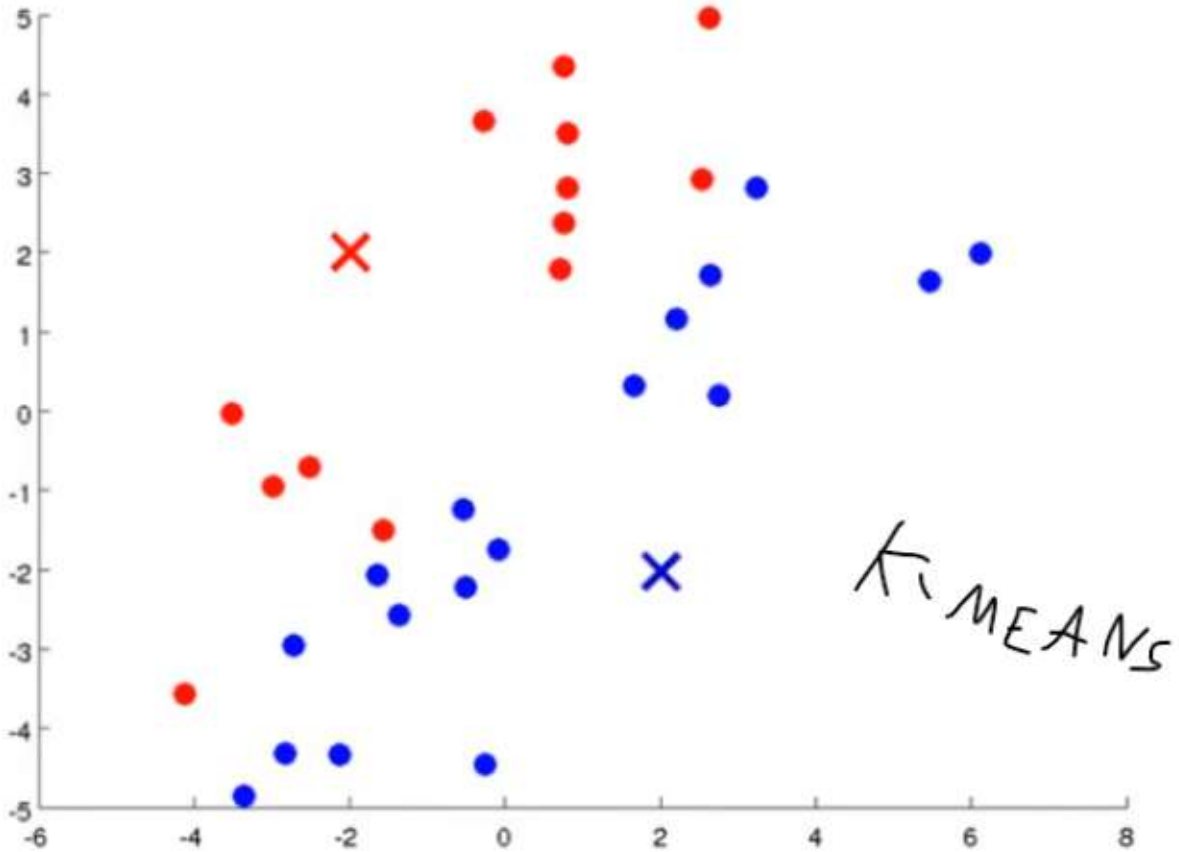
Logistic regression vs. SVMs

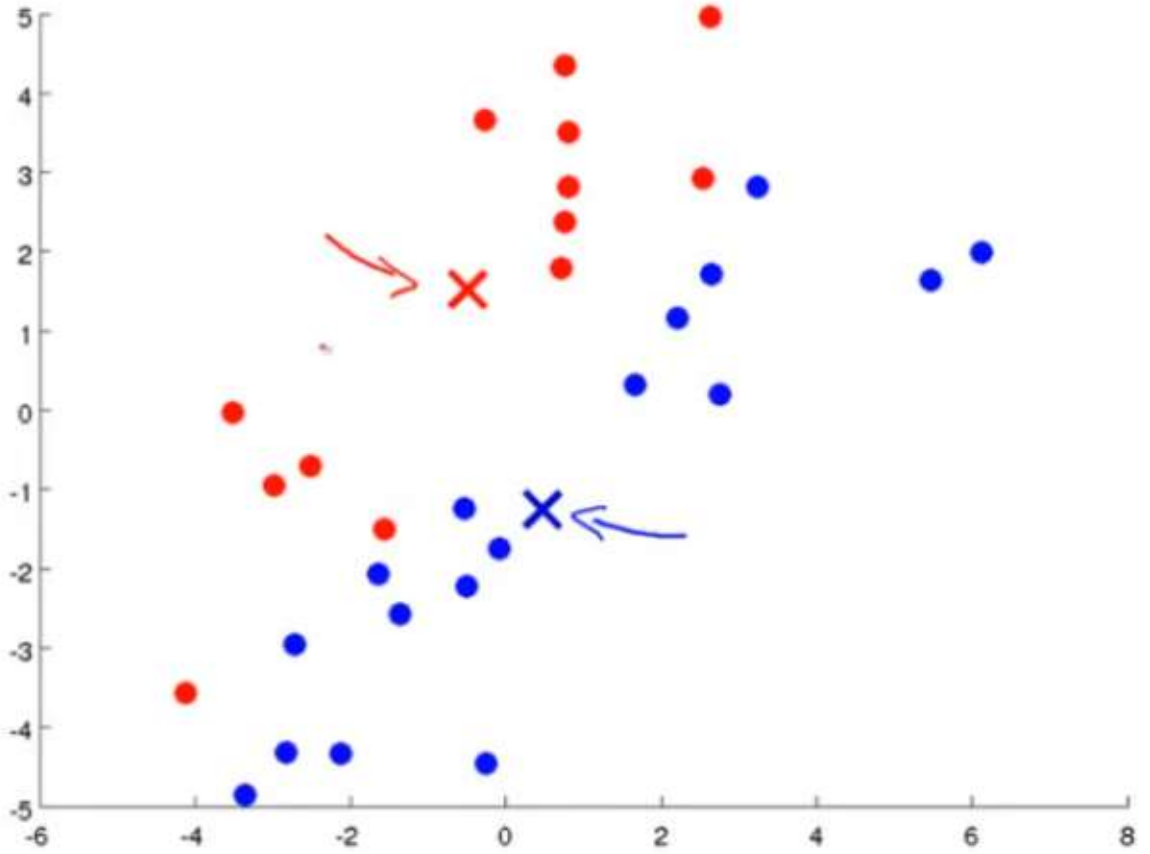
n = number of features ($x \in \mathbb{R}^{n+1}$), m = number of training examples

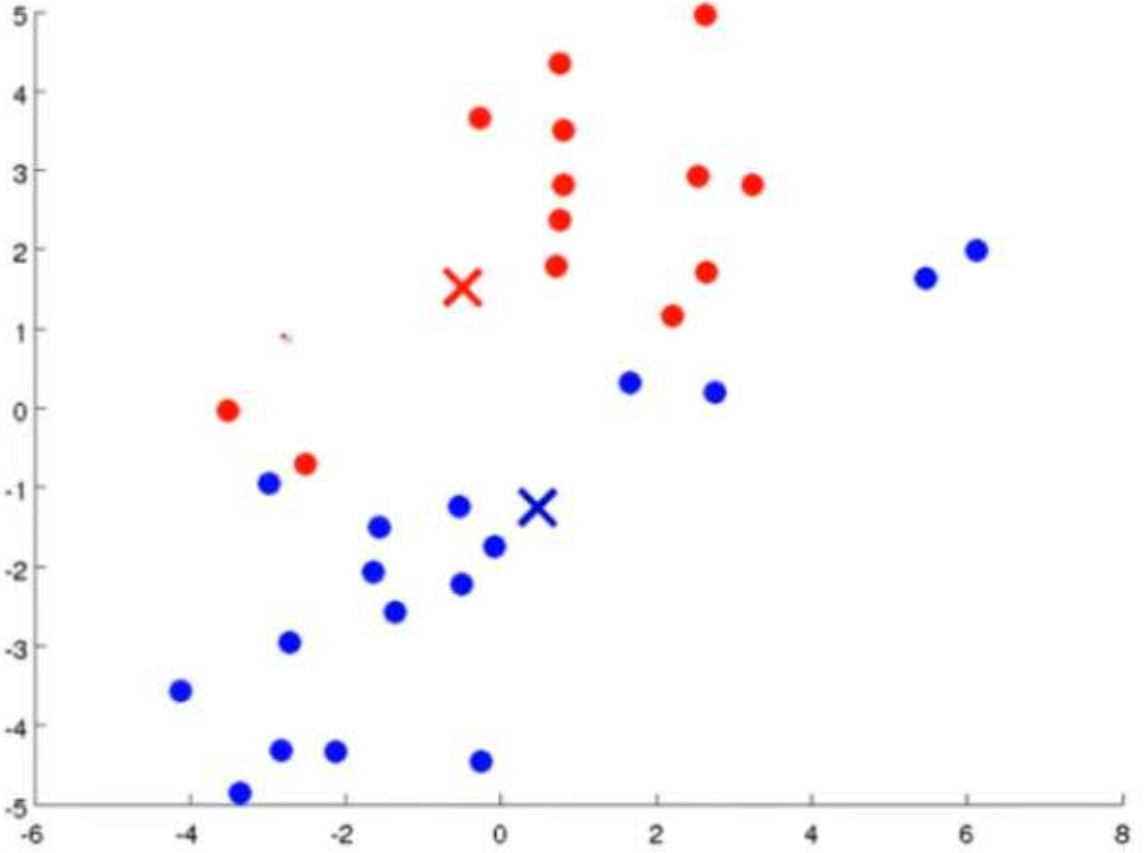
- If n is large (relative to m): (e.g. $n \geq m$, $n = 10{,}000$, $m = 10 \cdots 1000$)
  - Use logistic regression, or SVM without a kernel ("linear kernel")
- If n is small, m is intermediate: ($n = 1 - 1000$, $m = 10 - 10{,}000$)
  - Use SVM with Gaussian kernel
- If n is small, m is large: ($n = 1 - 1000$, $m = 50{,}000+$)
  - Create/add more features, then use logistic regression or SVM without a kernel

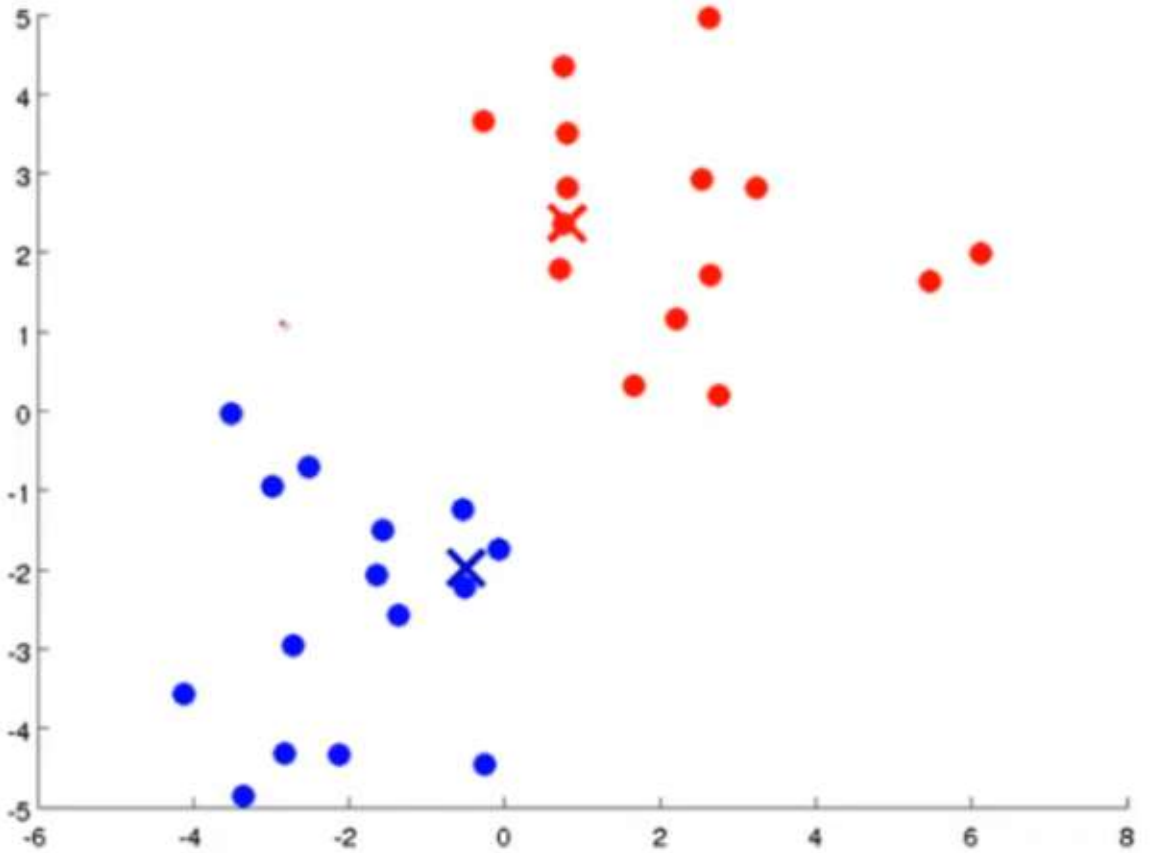


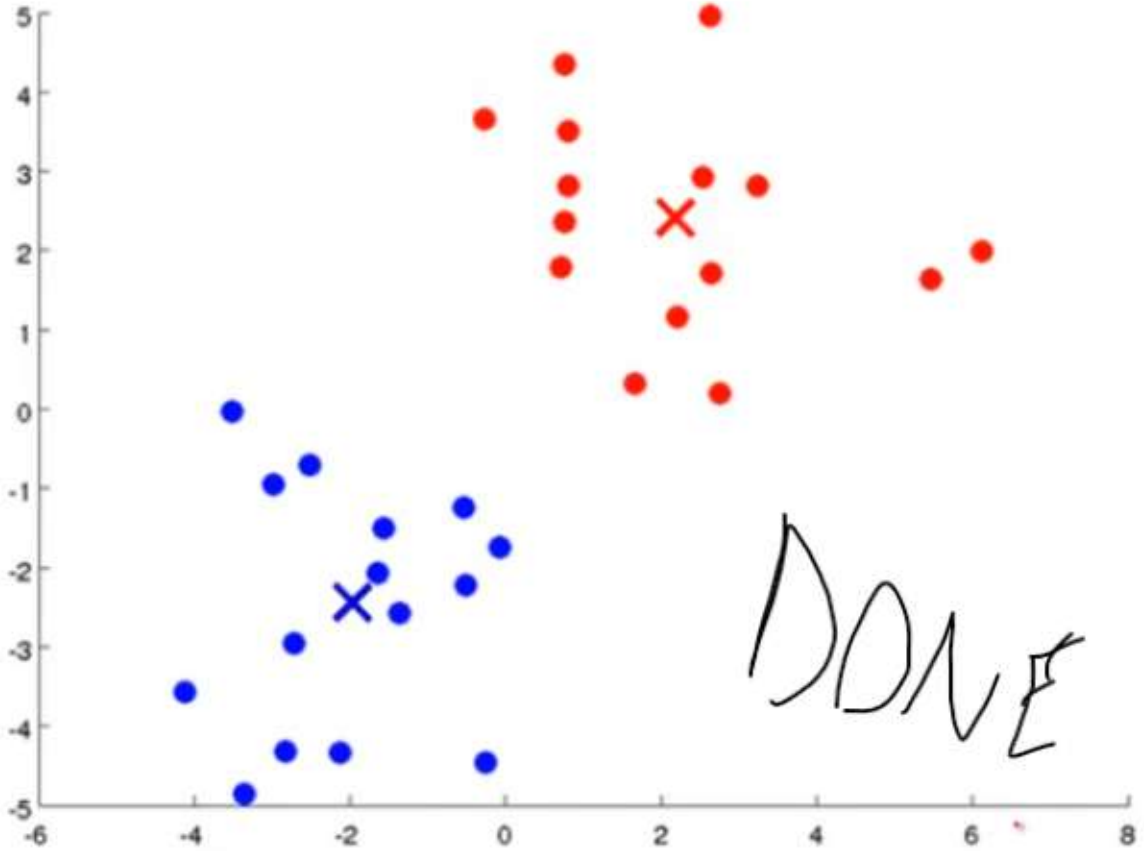
- Neural network likely to work well for most of these settings, but may be slower to train.


5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6
8
K-MEANS


5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6
8


5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6
8


5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6
8


5
4
3
2
1
0
-1
-2
-3
-4
-5
-6
-4
-2
0
2
4
6
8
DONE

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 $x^{(i)}$

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

Move centroid

for $k = 1$ to K

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

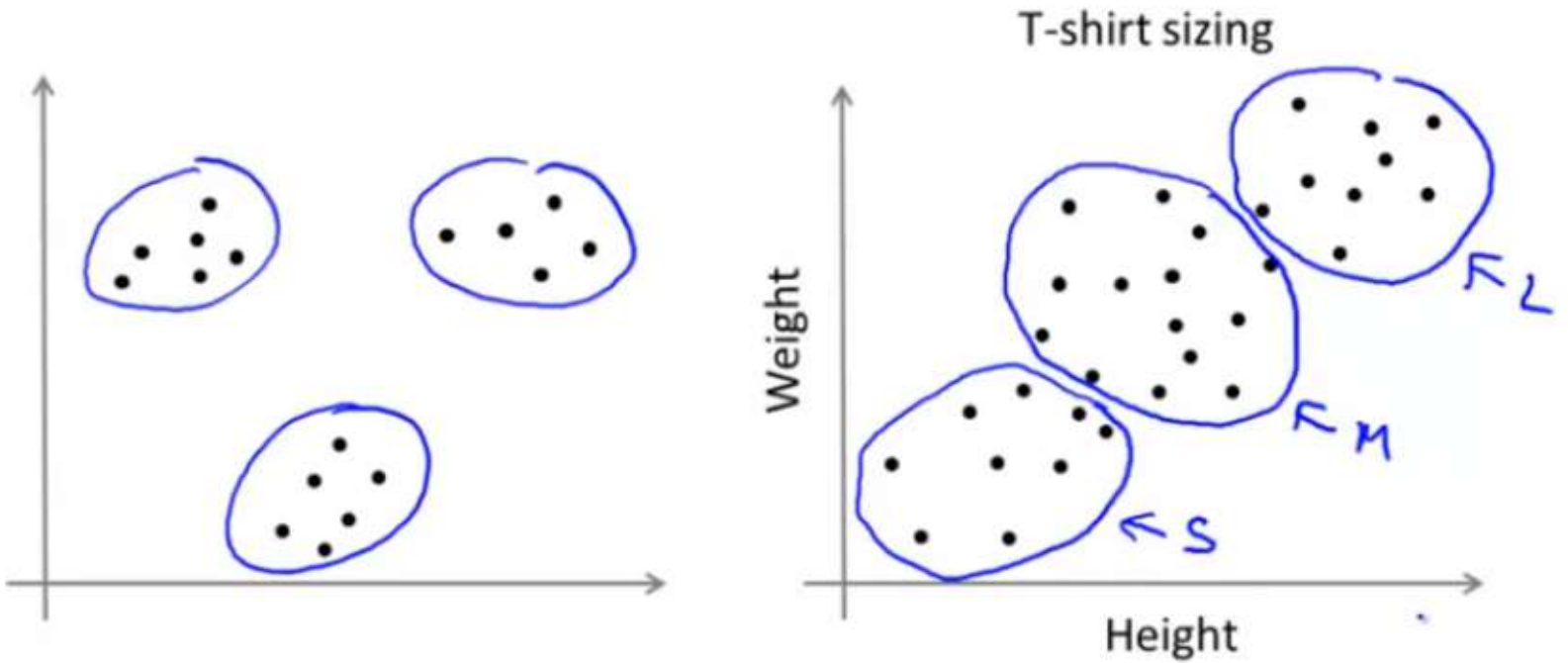
$x^{(1)}, x^{(5)}, x^{(6)}, x^{(10)} \rightarrow c^{(1)}=2,\ c^{(5)}=2,\ c^{(6)}=2,\ c^{(10)}=2$

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

}

K-means for non-separated clusters

S, M, L



K-means optimization objective

→ $c^{(i)}$ = index of cluster (1,2,…,K) to which example $x^{(i)}$ is currently assigned

→ μ_k = cluster centroid k ($\mu_k \in \mathbb{R}^n$)

$\mu_{c^{(i)}}$ = cluster centroid of cluster to which example $x^{(i)}$ has been assigned

K $\quad k \in \{1,2,\ldots,K\}$

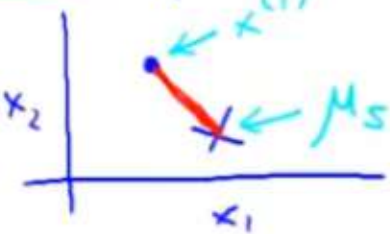
$x^{(i)} \rightarrow 5$ $\quad c^{(i)} = 5$ $\quad \mu_{c^{(i)}} = \mu_5$

Optimization objective:

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

→ $$\min_{\substack{c^{(1)},\ldots,c^{(m)},\\ \mu_1,\ldots,\mu_K}} J(c^{(1)},\ldots,c^{(m)},\mu_1,\ldots,\mu_K)$$

Distortion



K-means algorithm

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

Repeat {

Cluster assignment step

Minimize $J(\ldots)$ wrt $c^{(1)}, c^{(2)}, \ldots, c^{(m)}$ ← (holding $\mu_1, \ldots, \mu_K$ fixed)

for i = 1 to m

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

move centroid

for k = 1 to K

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

}

minimize $J(\ldots)$ wrt $\mu_1, \ldots, \mu_K$

K-means algorithm

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

Repeat {

for i = 1 to m

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

for k = 1 to K

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

}

Random initialization

Should have $K < m$

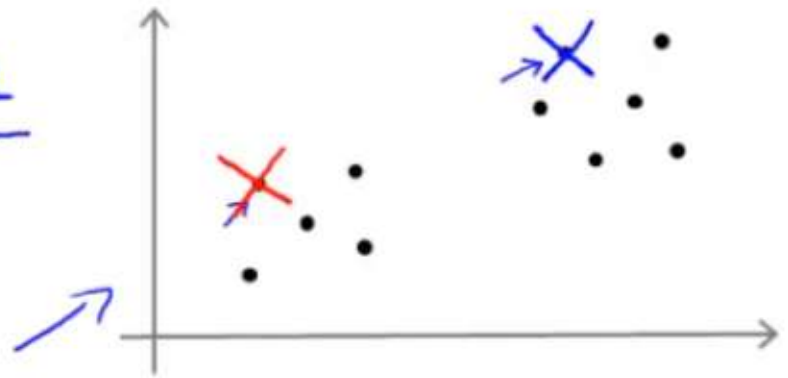
Randomly pick K training examples.

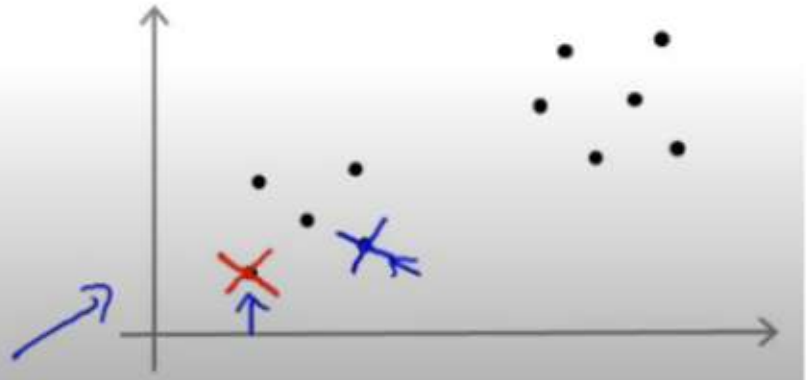
Set $\mu_1, \ldots, \mu_K$ equal to these K examples.

$\mu_1 = x^{(i)}$

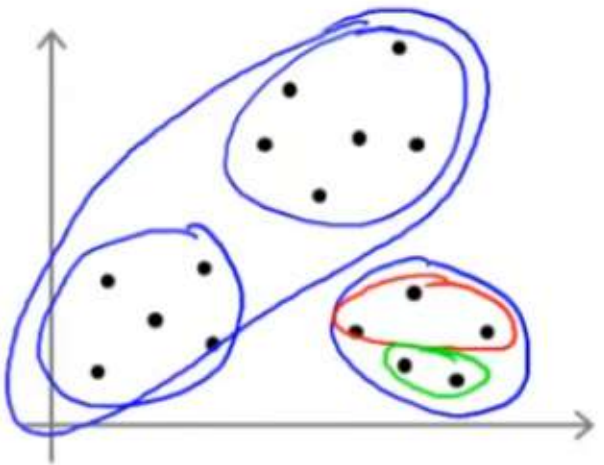
$\mu_2 = x^{(j)}$

$\vdots$

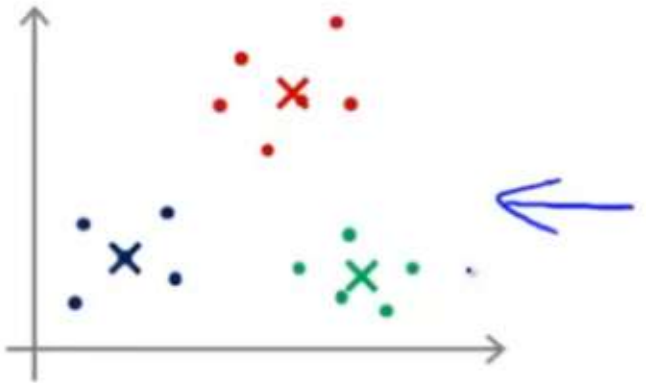


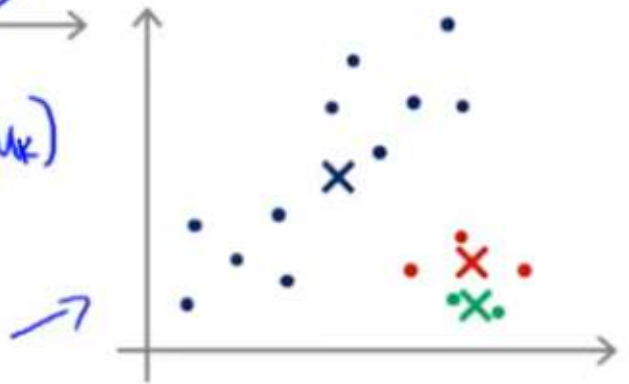


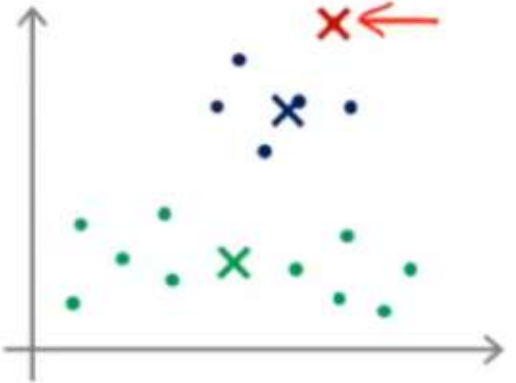
Local optima



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







Random initialization

For i = 1 to 100 {

50 - 1000

Randomly initialize K-means.
Run K-means. Get $c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K$.
Compute cost function (distortion)
$J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K)$
}

Pick clustering that gave lowest cost $J(c^{(1)}, \ldots, c^{(m)}, \mu_1, \ldots, \mu_K)$

K = 2 - 10

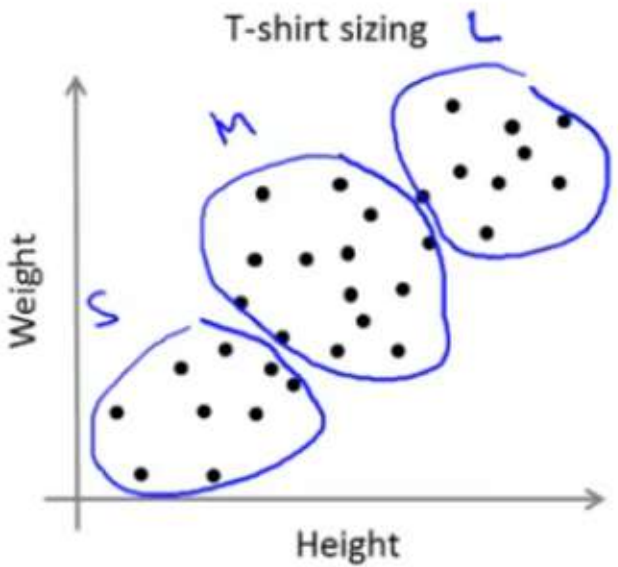
Choosing the value of K

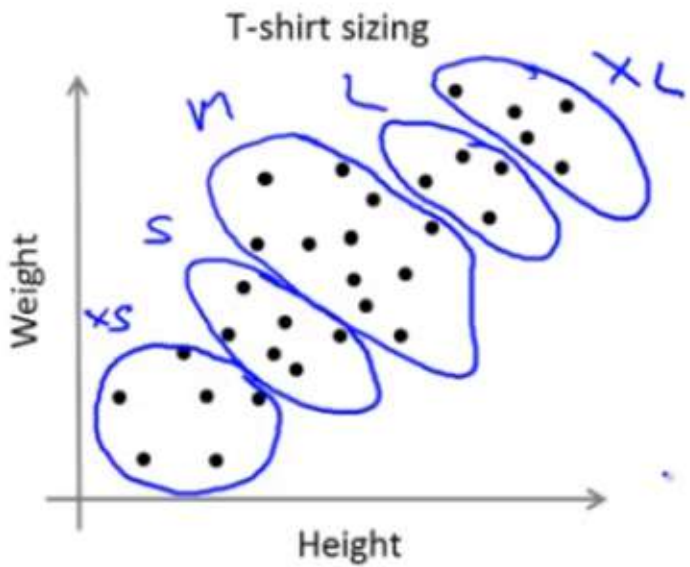
Sometimes, you're running K-means to get clusters to use for some later/downstream purpose. Evaluate K-means based on a metric for how well it performs for that later purpose.

K=3 S, M, L

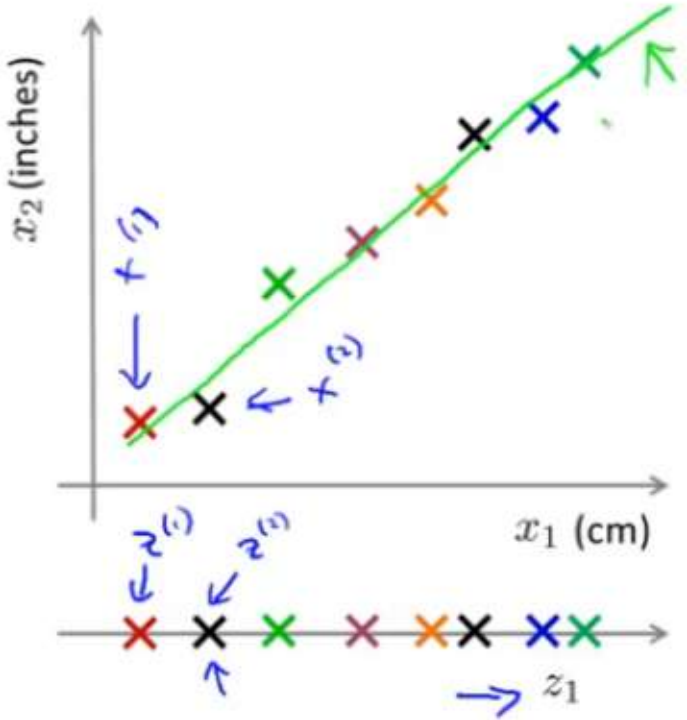
K=5 XS, S, M, L, XL

E.g.





Data Compression



Reduce data from 2D to 1D

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

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

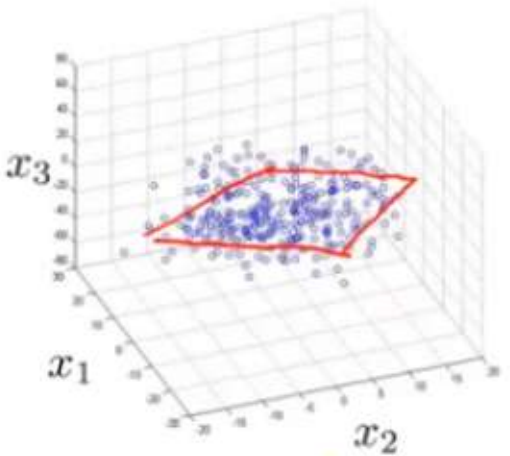
$\vdots$

$x^{(m)} \rightarrow z^{(m)}$

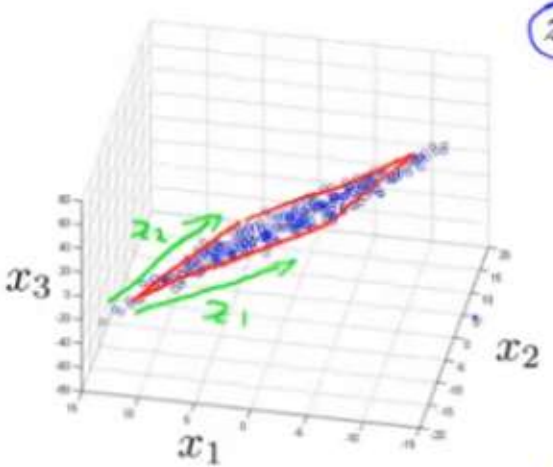
Data Compression

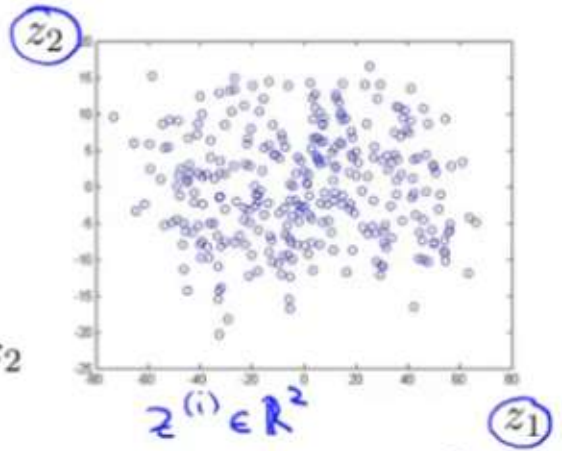
10000D → 1000D

Reduce data from 3D to 2D



$x^{(i)} \in \mathbb{R}^3$

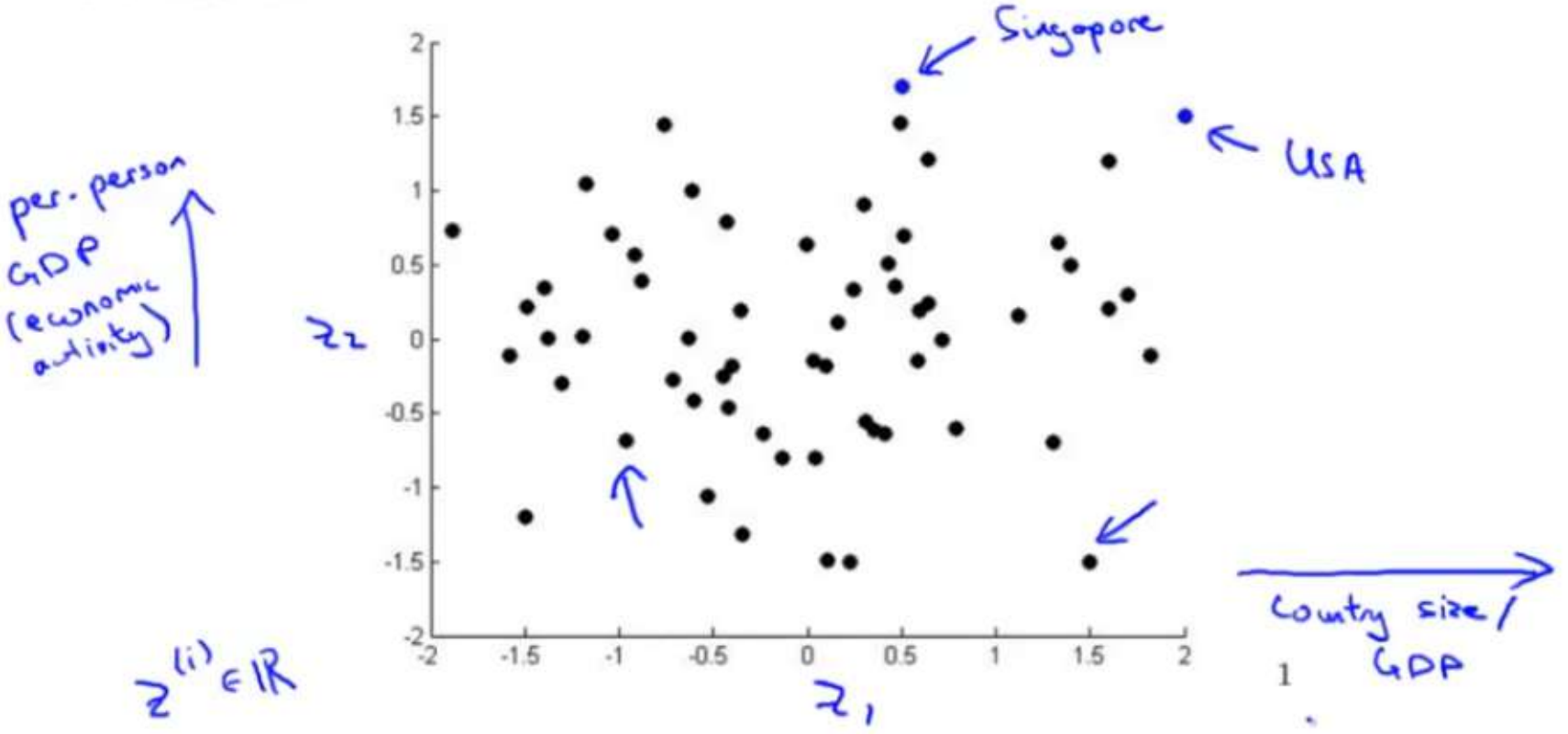




$z^{(i)} \in \mathbb{R}^2$

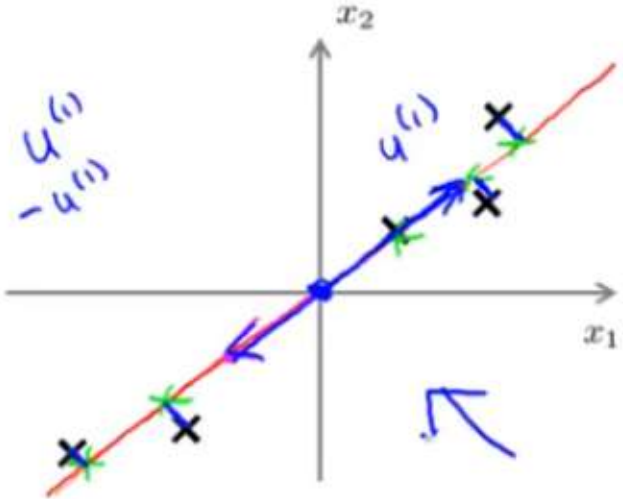
$$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} \qquad z^{(i)} = \begin{bmatrix} z_1^{(i)} \\ z_2^{(i)} \end{bmatrix}$$

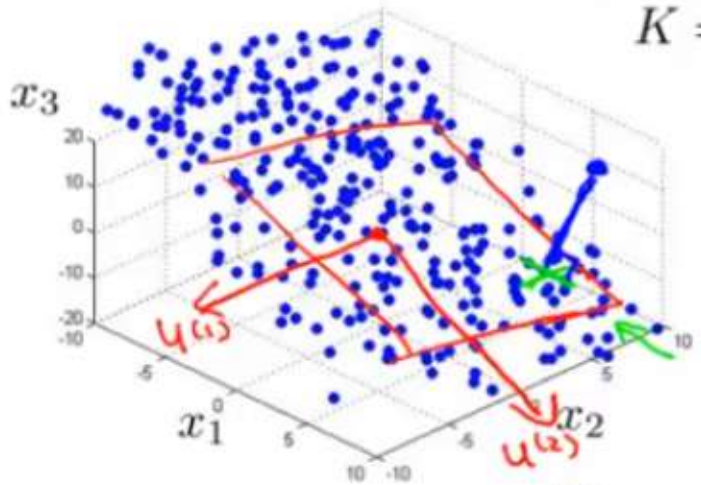
Data Visualization



Principal Component Analysis (PCA) problem formulation

$3D \rightarrow 2D$
$K = 2$

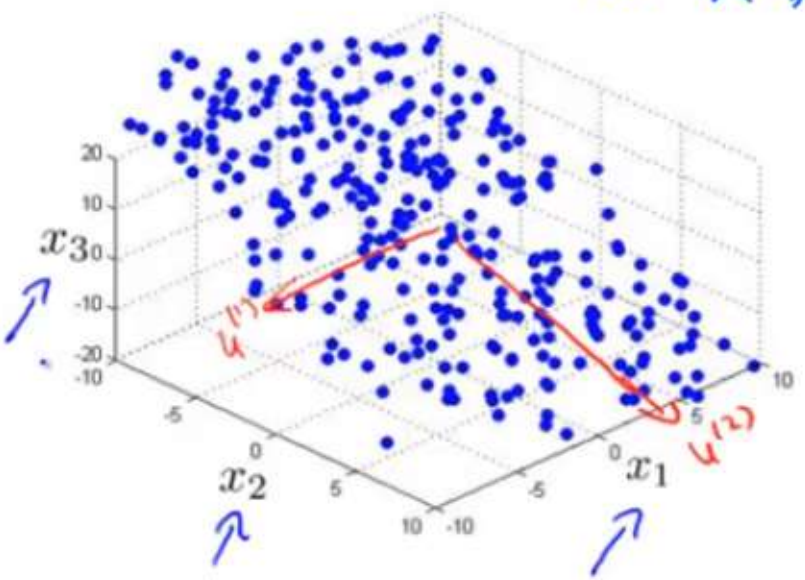


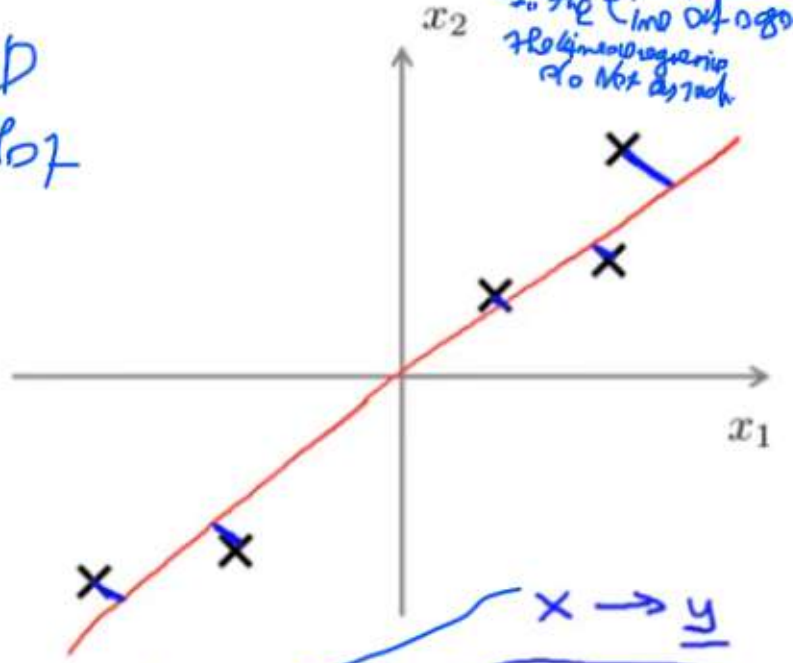


Reduce from 2-dimension to 1-dimension: Find a direction (a vector $u^{(1)} \in \mathbb{R}^n$) onto which to project the data so as to minimize the projection error.

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





Data preprocessing

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

Preprocessing (feature scaling/mean normalization):

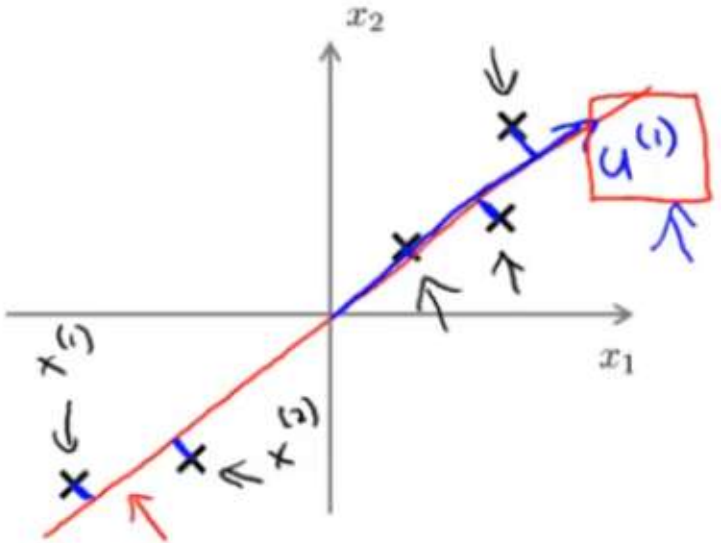
$$\mu_j = \frac{1}{m} \sum_{i=1}^{m} x_j^{(i)}$$

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

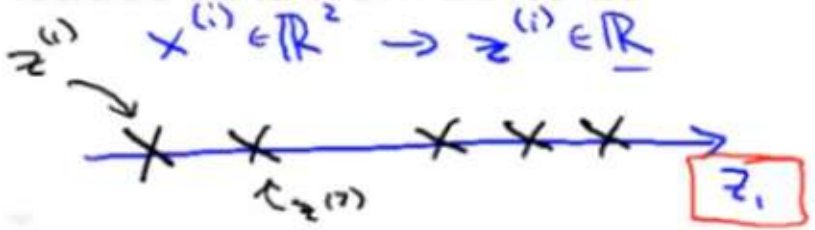
If different features on different scales (e.g., $x_1 =$ size of house, $x_2 =$ number of bedrooms), scale features to have comparable range of values.

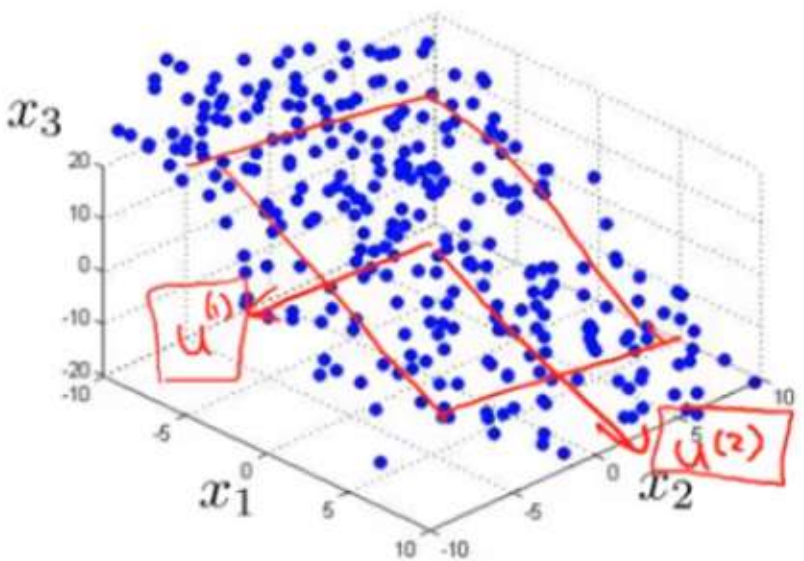
$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j}$$

Principal Component Analysis (PCA) algorithm



Reduce data from 2D to 1D





Reduce data from 3D to 2D

$x^{(i)} \in \mathbb{R}^3 \rightarrow z^{(i)} \in \mathbb{R}^2$

$z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$

Principal Component Analysis (PCA) algorithm

Reduce data from n-dimensions to k-dimensions

Compute "covariance matrix":

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

$n \times 1$ $\quad$ $1 \times n$ $\quad$ $n \times n$

Sigma

Compute "eigenvectors" of matrix Σ:

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

→ Singular value decomposition

eig(Sigma)

$n \times n$ matrix.

$$U = \begin{bmatrix} | & | & | & & | \\ u^{(1)} & u^{(2)} & u^{(3)} & \cdots & u^{(m)} \\ | & | & | & & | \end{bmatrix}$$

k

$U \in \mathbb{R}^{n \times n}$

$u^{(1)}, \ldots, u^{(k)}$

Principal Component Analysis (PCA) algorithm

From `[U,S,V] = svd(Sigma)`, we get:

$$\rightarrow U = \begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(n)} \\ | & | & & | \end{bmatrix} \in \mathbb{R}^{n \times n}$$

k

$x \in \mathbb{R}^n \longrightarrow z \in \mathbb{R}^k$

$$z^{(i)} = \underbrace{\begin{bmatrix} | & | & & | \\ u^{(1)} & u^{(2)} & \cdots & u^{(k)} \\ | & | & & | \end{bmatrix}^T}_{n \times k} \underbrace{\begin{bmatrix} — (u^{(1)})^T — \\ \vdots \\ — (u^{(k)})^T — \end{bmatrix}}_{k \times n} \underbrace{x^{(i)}}_{n \times 1}$$

$z \in \mathbb{R}^k$

U_{reduce}

Example

$k \times 1$

Principal Component Analysis (PCA) algorithm summary

After mean normalization (ensure every feature has zero mean) and optionally feature scaling:

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

```
[U,S,V] = svd(Sigma);
Ureduce = U(:,1:k);
z = Ureduce'*x;
```

$X = \begin{bmatrix} — (x^{(1)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$

Sigma = (1/m) * X' * X;

$x \in \mathbb{R}^n$ $\quad x_0 = 1$

Choosing k (number of principal components)

Average squared projection error: $\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)} - x_{approx}^{(i)}\|^2$

Total variation in the data: $\frac{1}{m}\sum_{i=1}^{m} \|x^{(i)}\|^2$

Typically, choose k to be smallest value so that

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

0.05 0.10 5% (10%)

"99% of variance is retained"

95% 90%

Choosing k (number of principal components)

Algorithm:

Try PCA with $k = 1$ ~~$k=2$~~ ~~$k=3$~~ $k=4$ ...

Compute $U_{reduce}, z^{(1)}, z^{(2)}, \ldots, z^{(m)}, x_{approx}^{(1)}, \ldots, x_{approx}^{(m)}$

Check if

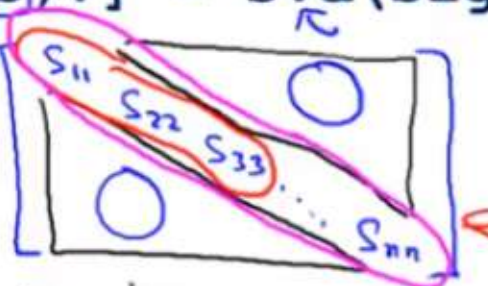
$$\frac{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)} - x_{approx}^{(i)}\|^2}{\frac{1}{m}\sum_{i=1}^{m}\|x^{(i)}\|^2} \leq 0.01?$$

check if 99% of variance is retained (you can choose)

$k = 17$

small as possible

→ [U, S, V] = svd(Sigma)

→ S = 

$k = 3$

For given k

$$1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \leq 0.01$$

$$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}} \geq 0.99$$

of Variance

Choosing k (number of principal components)

→ `[U,S,V] = svd(Sigma)`

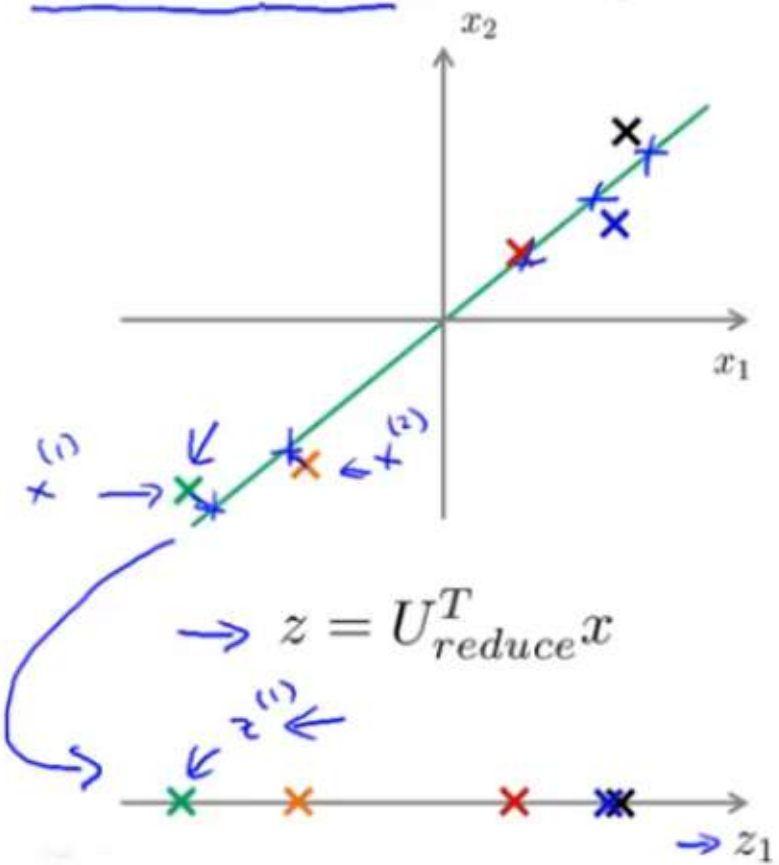
Pick smallest value of k for which

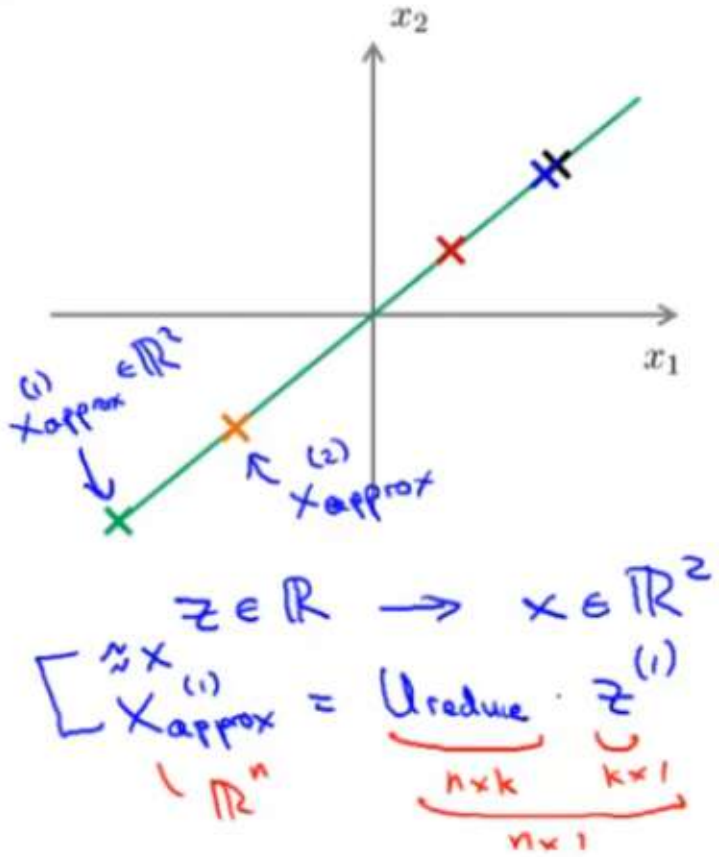
$$\frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{m} S_{ii}} \geq 0.99$$

(99% of variance retained)

k=100

Reconstruction from compressed representation

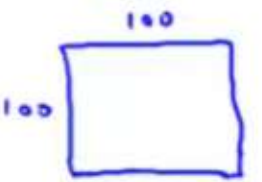




Supervised learning speedup

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

$x^{(i)} \in \mathbb{R}^{10000}$



Extract inputs:

Unlabeled dataset: $x^{(1)}, x^{(2)}, \ldots, x^{(m)} \in \mathbb{R}^{10000}$

$\downarrow PCA$

$z^{(1)}, z^{(2)}, \ldots, z^{(m)} \in \mathbb{R}^{1000}$

$x \rightarrow z$

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

U_{reduce}

Compute

New training set:

$(z^{(1)}, y^{(1)}), (z^{(2)}, y^{(2)}), \ldots, (z^{(m)}, y^{(m)})$

Note: Mapping $x^{(i)} \rightarrow z^{(i)}$ should be defined by running PCA only on the training set. This mapping can be applied as well to the examples $x^{(i)}_{cv}$ and $x^{(i)}_{test}$ in the cross validation and test sets.

$x \rightarrow z$

Application of PCA

- Compression
  - Reduce memory/disk needed to store data
  - Speed up learning algorithm ⇐

Choose k by % of variance retain

- Visualization

$k = 2$ or $k = 3$

Bad use of PCA: To prevent overfitting

→ Use $z^{(i)}$ instead of $x^{(i)}$ to reduce the number of features to $k < n$. — 1000 — 10000

Thus, fewer features, less likely to overfit.

Bad!

This might work OK, but isn't a good way to address overfitting. Use regularization instead. → Good

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

PCA is sometimes used where it shouldn't be

Design of ML system:

- Get training set $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})\}$
- ~~Run PCA to reduce $x^{(i)}$ in dimension to get $z^{(i)}$~~
- Train logistic regression on $\{(z^{(1)}, y^{(1)}), \ldots, (z^{(m)}, y^{(m)})\}$
- Test on test set: Map $x^{(i)}_{test}$ to $z^{(i)}_{test}$. Run $h_\theta(z)$ on $\{(z^{(1)}_{test}, y^{(1)}_{test}), \ldots, (z^{(m)}_{test}, y^{(m)}_{test})\}$

→ How about doing the whole thing without using PCA?

→ Before implementing PCA, first try running whatever you want to do with the original/raw data $x^{(i)}$. Only if that doesn't do what you want, then implement PCA and consider using $z^{(i)}$.

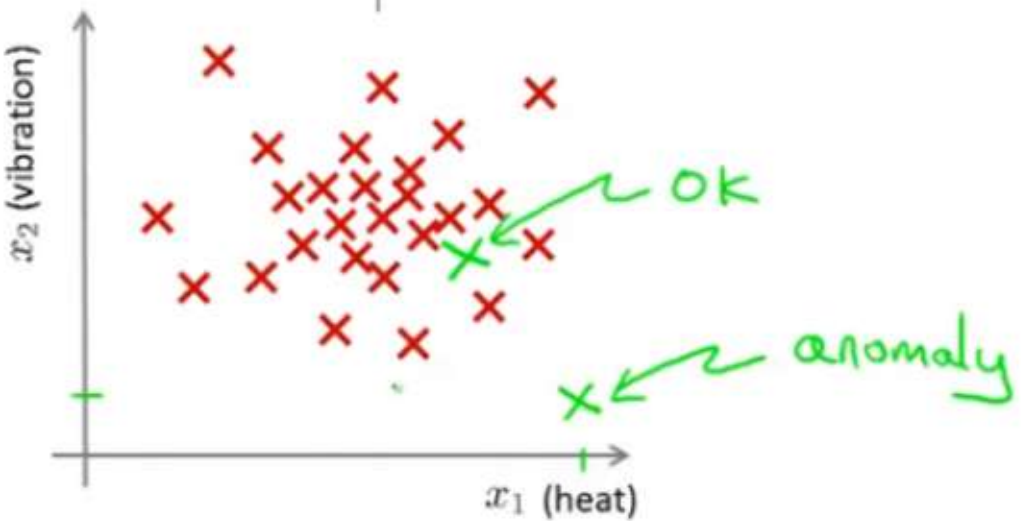
Anomaly detection example

Aircraft engine features:

- x_1 = heat generated
- x_2 = vibration intensity
- ...

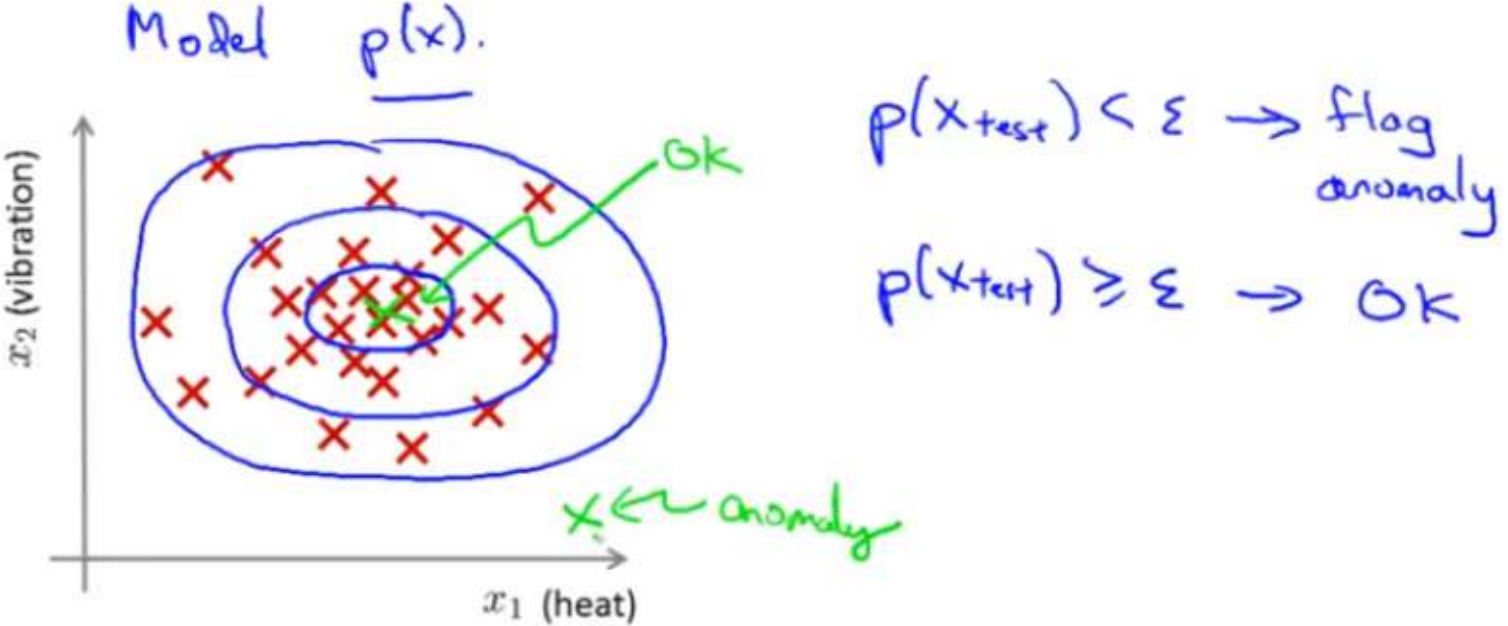
Dataset: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$

New engine: x_{test}



Density estimation

- Dataset: $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$
- Is x_{test} anomalous?



Anomaly detection example

- Fraud detection:
  - $x^{(i)}$ = features of user i's activities
  - Model $p(x)$ from data.
  - Identify unusual users by checking which have $p(x) < \varepsilon$
- Manufacturing
- Monitoring computers in a data center.
  - $x^{(i)}$ = features of machine i

    x_1 = memory use, x_2 = number of disk accesses/sec,

    x_3 = CPU load, x_4 = CPU load/network traffic.

    ...

Gaussian (Normal) distribution

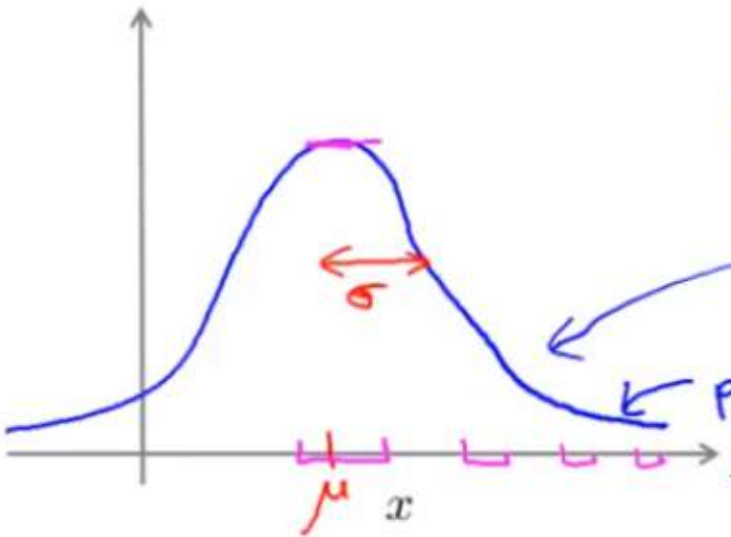
Say $x \in \mathbb{R}$. If x is a distributed Gaussian with mean μ, variance σ^2.

$x \sim \mathcal{N}(\mu, \sigma^2)$

"$\mathcal{N}$"

↑ "distributed as"

σ = standard deviation



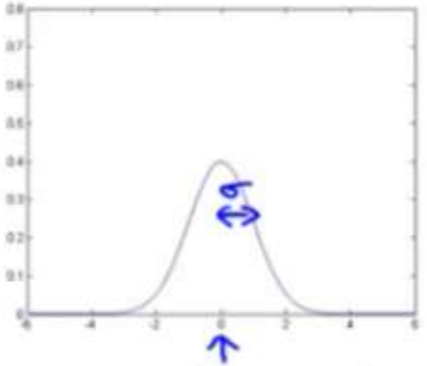
$$p(x; \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\,\sigma} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

$p(x; \mu, \sigma^2)$

x parameterzied by Fixed values of mu and sigma

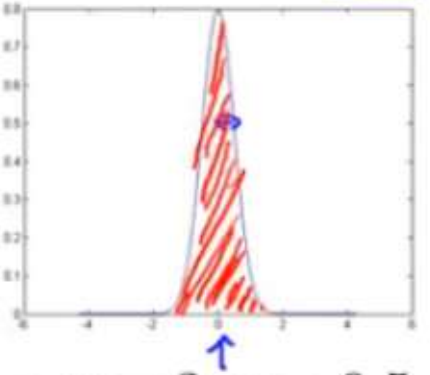
Gaussian distribution example

$\mu = 0, \sigma = 1$



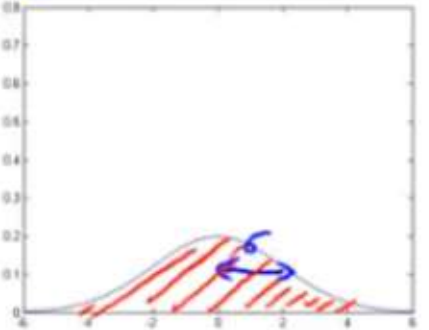
$\mu = 0, \sigma = 0.5$

$\sigma^2 = 0.25$

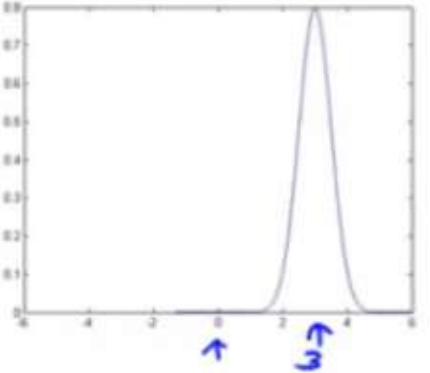


The red area always integret to a whole number, to the "1"

$\mu = 0, \sigma = 2$



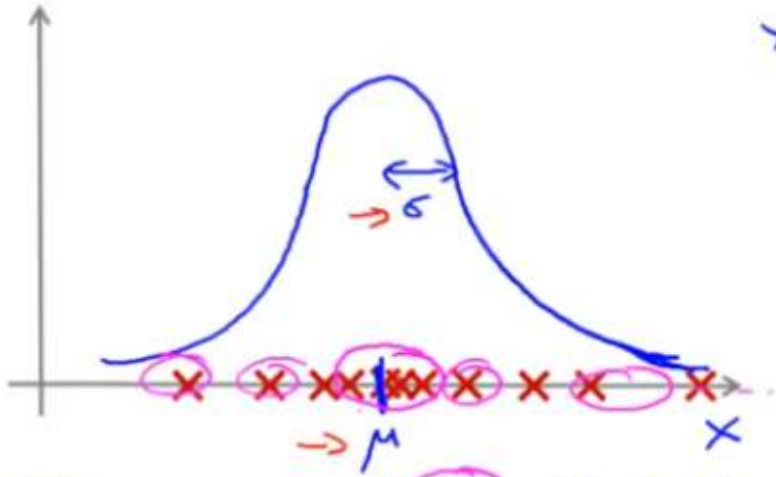
$\mu = 3, \sigma = 0.5$



Parameter estimation

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

$x^{(i)} \sim \mathcal{N}(\mu, \sigma^2)$



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

$$\rightarrow \sigma^2 = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)^2$$

$\frac{1}{m-1}$ ←

Density estimation

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

Each example is $x \in \mathbb{R}^n$

$x_1 \sim \mathcal{N}(\mu_1, \sigma_1^2)$

$x_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$

$x_3 \sim \mathcal{N}(\mu_3, \sigma_3^2)$

$$p(x) = p(x_1; \mu_1, \sigma_1^2)\, p(x_2; \mu_2, \sigma_2^2)\, p(x_3; \mu_3, \sigma_3^2) \cdots p(x_n; \mu_n, \sigma_n^2)$$

$$= \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2)$$

$$\sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + n$$

$$\prod_{i=1}^{n} i = 1 \times 2 \times 3 \times \cdots \times n$$

Anomaly detection algorithm

1. Choose features x_i that you think might be indicative of anomalous examples.
2. Fit parameters $\mu_1, \ldots, \mu_n, \sigma_1^2, \ldots, \sigma_n^2$

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

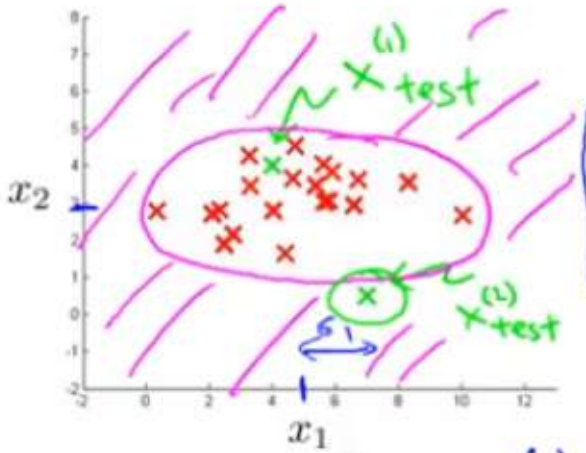
$$\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m} (x_j^{(i)} - \mu_j)^2$$

3. Given new example x, compute $p(x)$:

$$p(x) = \prod_{j=1}^{n} p(x_j; \mu_j, \sigma_j^2) = \prod_{j=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left(-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}\right)$$

Anomaly if $p(x) < \varepsilon$

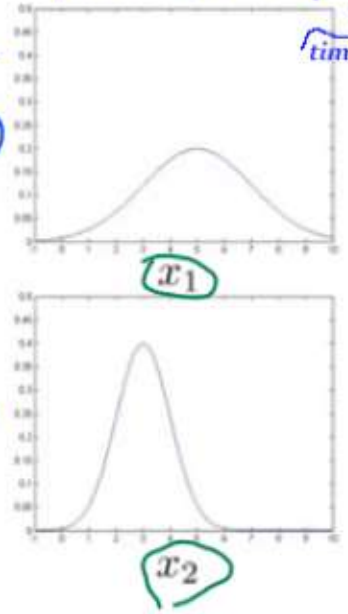
Anomaly detection example



$\mu_1 = 5, \sigma_1 = 2$

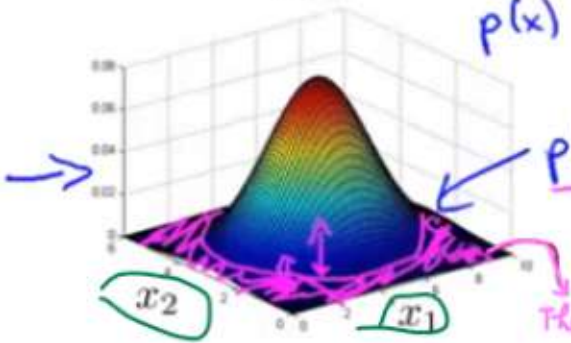
$\mu_2 = 3, \sigma_2 = 1$

The heights of plot → $p(x) = p(x_1; \mu_1, \sigma_1^2)$ times $p(x_2; \mu_2, \sigma_2^2)$



$p(x_1; \mu_1, \sigma_1^2)$

$p(x_2; \mu_2, \sigma_2^2)$



The area of Anormal values

$\varepsilon = 0.02$

$p(x_{test}^{(1)}) = 0.0426 \geq \varepsilon$ Anormal / Non-Anom

$p(x_{test}^{(2)}) = 0.0021 < \varepsilon$ ANormal

The importance of real-number evaluation

When developing a learning algorithm (choosing features, etc.), making decisions is much easier if we have a way of evaluating our learning algorithm.

→ Assume we have some labeled data, of anomalous and non-anomalous examples. ($y = 0$ if normal, $y = 1$ if anomalous).

→ Training set: $x^{(1)}, x^{(2)}, \ldots, x^{(m)}$ (assume normal examples/not anomalous)

→ Cross validation set: $(x_{cv}^{(1)}, y_{cv}^{(1)}), \ldots, (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$

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

$y=1$

Aircraft engines motivating example

10000 good (normal) engines

20 flawed engines (anomalous)

Training set: 6000 good engines ($y = 0$)

CV: 2000 good engines ($y = 0$), 10 anomalous ($y = 1$)

Test: 2000 good engines ($y = 0$), 10 anomalous ($y = 1$)

Alternative:

Training set: 6000 good engines

CV: 4000 good engines ($y = 0$), 10 anomalous ($y = 1$)

Test: 4000 good engines ($y = 0$), 10 anomalous ($y = 1$)

Algorithm evaluation

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

→ On a cross validation/test example x, predict

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

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

$y = 0$

Possible evaluation metrics:

→ - True positive, false positive, false negative, true negative

→ - Precision/Recall

→ - F_1-score ←

CV

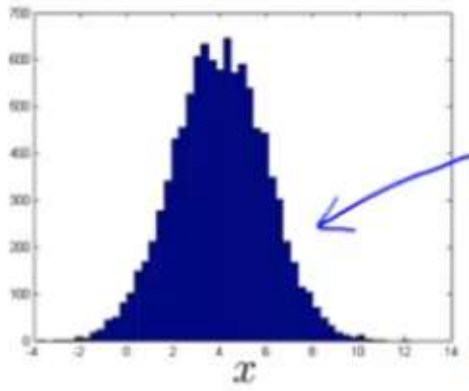
Test set.

Can also use cross validation set to choose parameter ε ←

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

Anomaly detection	vs.	Supervised learning
• Fraud detection $y=1$		• Email spam classification
• Manufacturing (e.g. aircraft engines)		• Weather prediction (sunny/rainy/etc).
• Monitoring machines in a data center		• Cancer classification
⋮		⋮

Non-gaussian features



hist

$p(x_i; \mu_i, \sigma_i^2)$

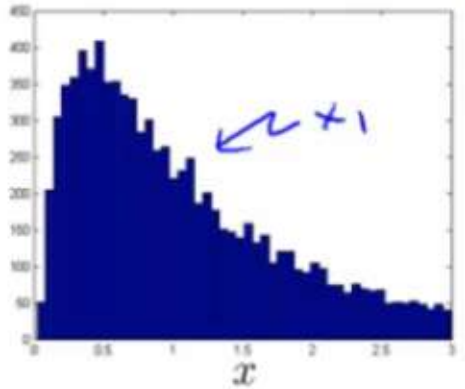
$x_1 \leftarrow \log(x_1)$

$x_2 \leftarrow \log(x_2+1)$ $\quad \log(x_2+c)$

$x_3 \leftarrow \sqrt{x_3} = x_3^{\frac{1}{2}}$

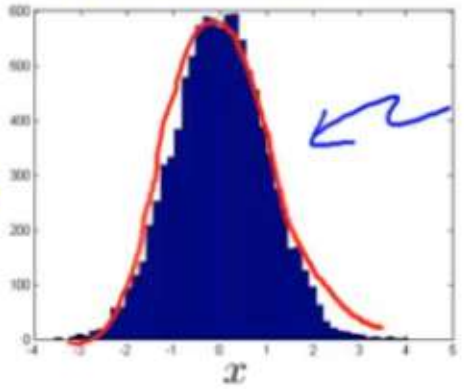
$x_4 \leftarrow x_4^{\frac{1}{3}}$

Play with transformation



x_1

$\log(x)$

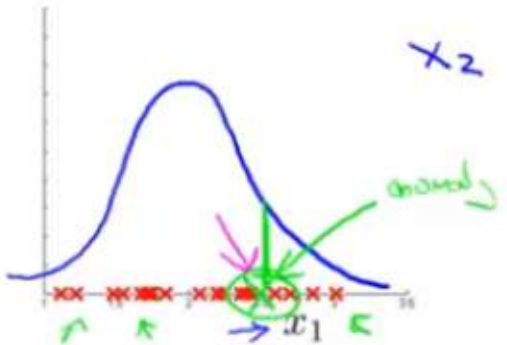


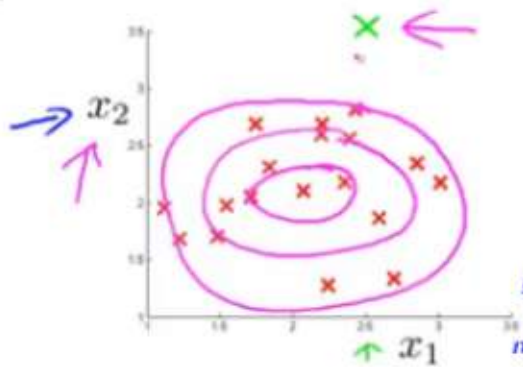
→ Error analysis for anomaly detection

Want $p(x)$ large for normal examples x.
$p(x)$ small for anomalous examples x.

Most common problem:

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





Just search for a inspiratoin: like the anormal would have some feature outscaled or something like this, and use this to detect the anormaly behaivour of the anomals values, like it can be seen on the left-plot on this slide

in fazit to be able to creat new features to the normally detetion later on

Monitoring computers in a data center

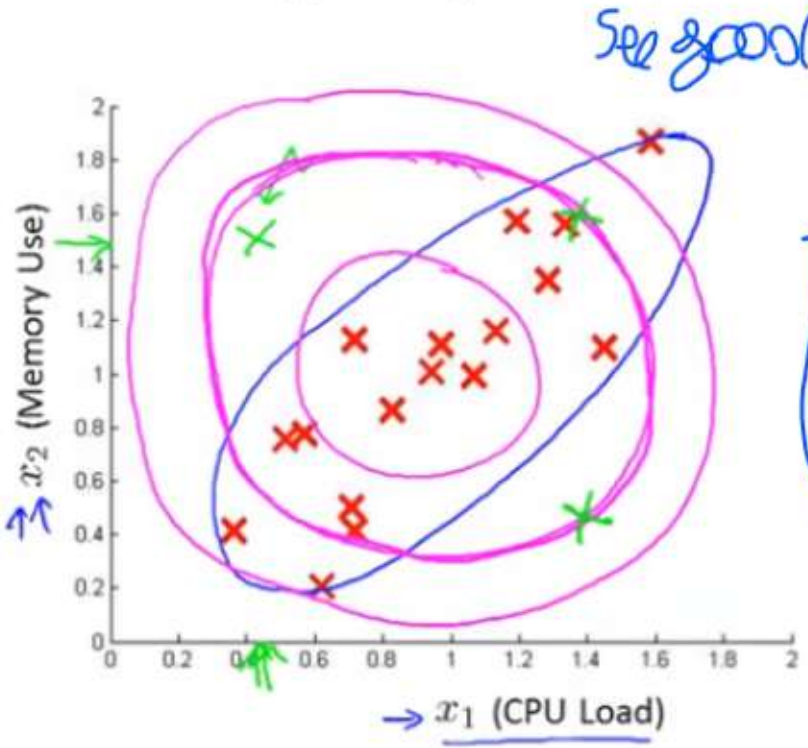
Choose features that might take on unusually large or small values in the event of an anomaly.

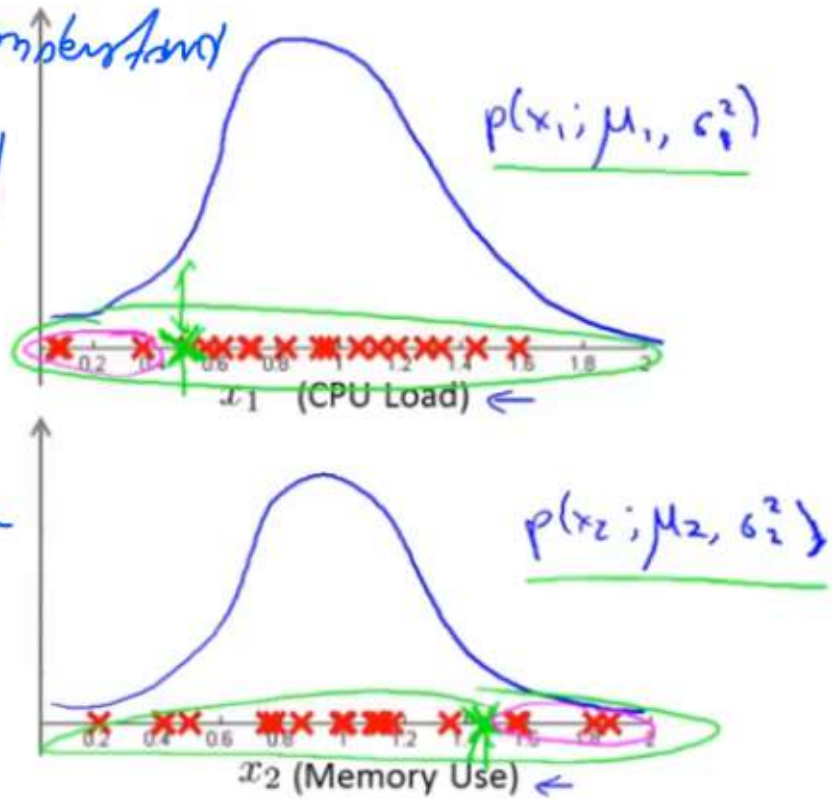
- x_1 = memory use of computer
- x_2 = number of disk accesses/sec
- x_3 = CPU load
- x_4 = network traffic

$$x_5 = \frac{\text{CPU load}}{\text{network traffic}}$$

$$x_6 = \frac{(\text{CPU load})^2}{\text{network traffic}}$$

Motivating example: Monitoring machines in a data center





Multivariate Gaussian (Normal) distribution

→ $x \in \mathbb{R}^n$. Don't model $p(x_1), p(x_2), \ldots$, etc. separately. Model $p(x)$ all in one go.

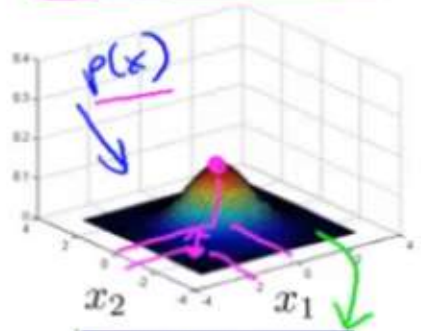
Parameters: $\mu \in \mathbb{R}^n$, $\Sigma \in \mathbb{R}^{n \times n}$ (covariance matrix)

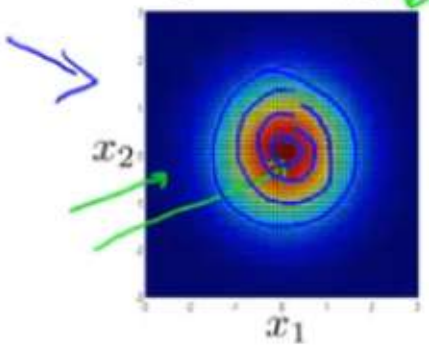
$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)\right)$$

$|\Sigma|$ = determinant of Σ | det (Sigma)

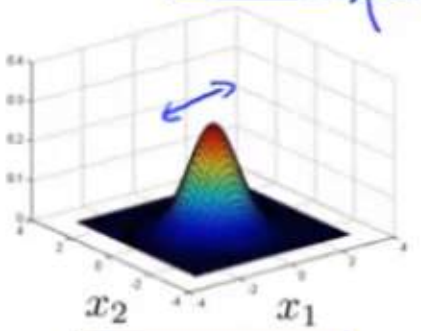
Multivariate Gaussian (Normal) examples

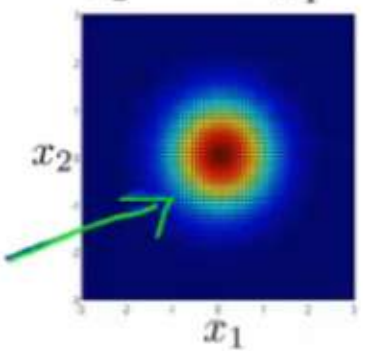
$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$



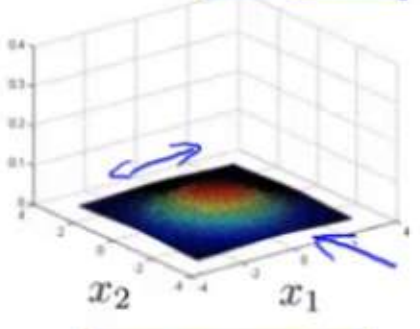


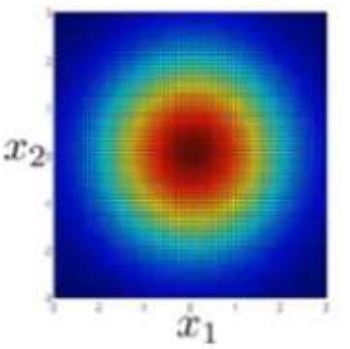
$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 0.6 \end{bmatrix}$





$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 2 \end{bmatrix}$

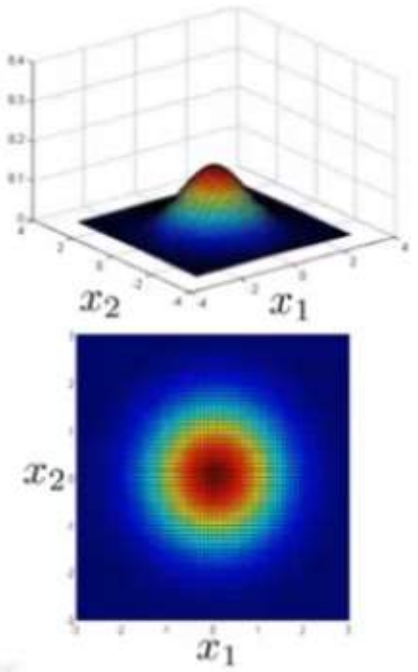




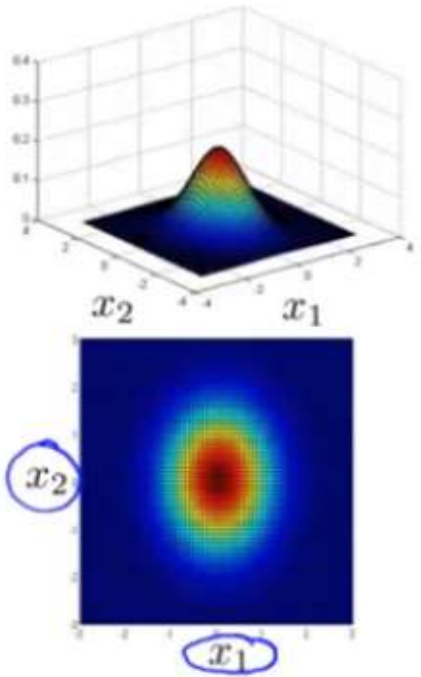
Andre

Multivariate Gaussian (Normal) examples

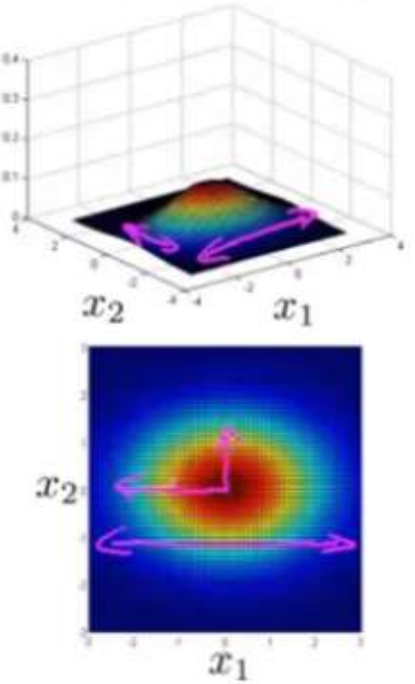
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 0.6 & 0 \\ 0 & 1 \end{bmatrix}$$



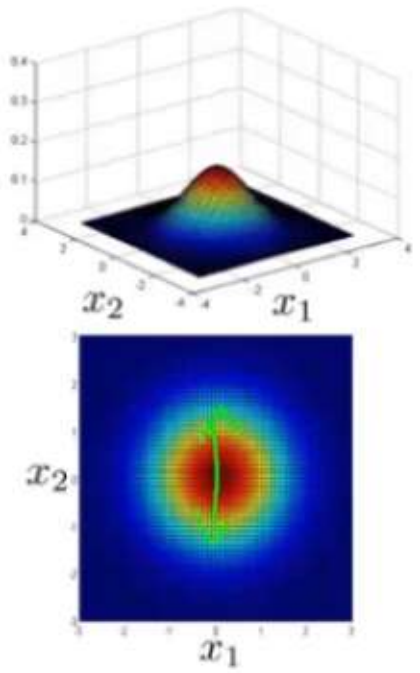
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$



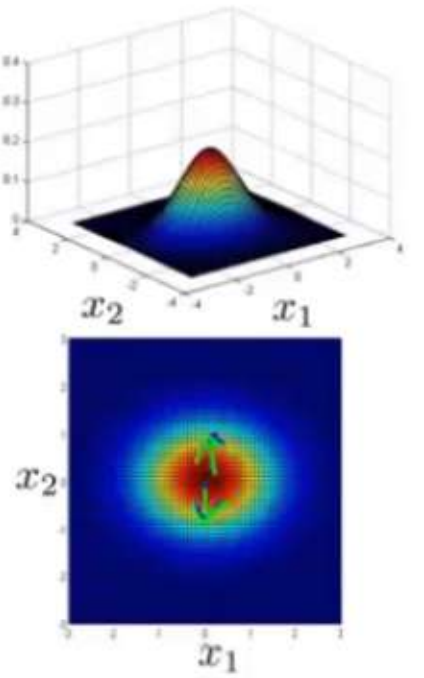
Andr

Multivariate Gaussian (Normal) examples

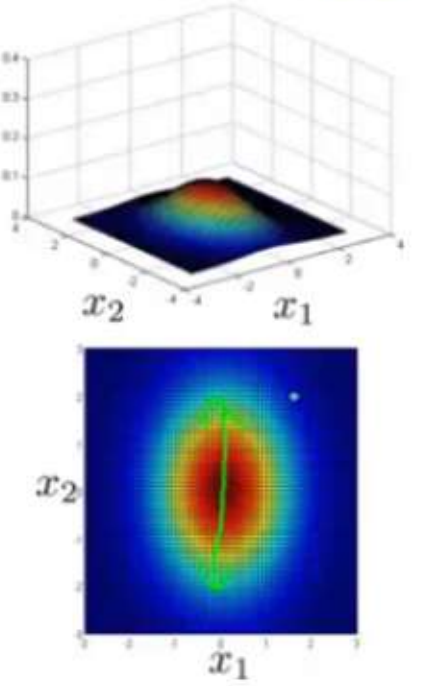
$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$



$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 0.6 \end{bmatrix}$



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



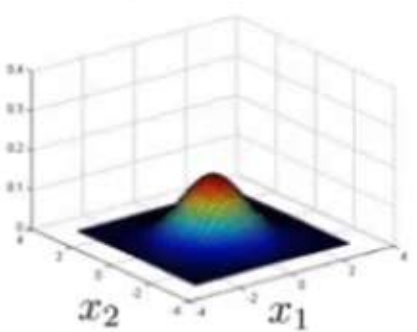
Andrew

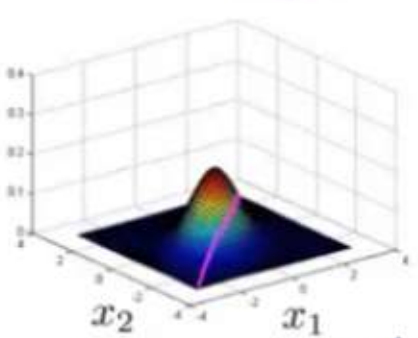
Multivariate Gaussian (Normal) examples

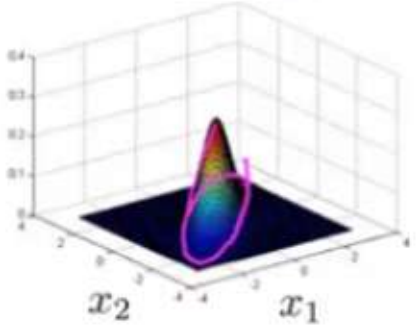
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

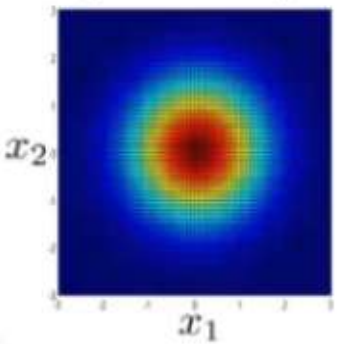
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix}$$

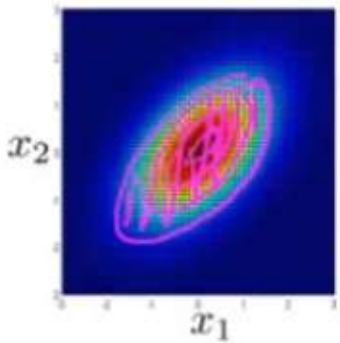
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0.8 \\ 0.8 & 1 \end{bmatrix}$$



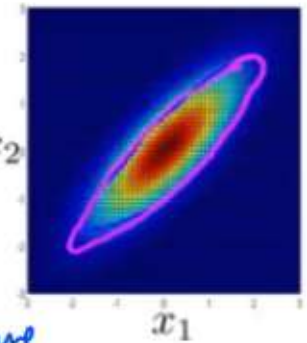






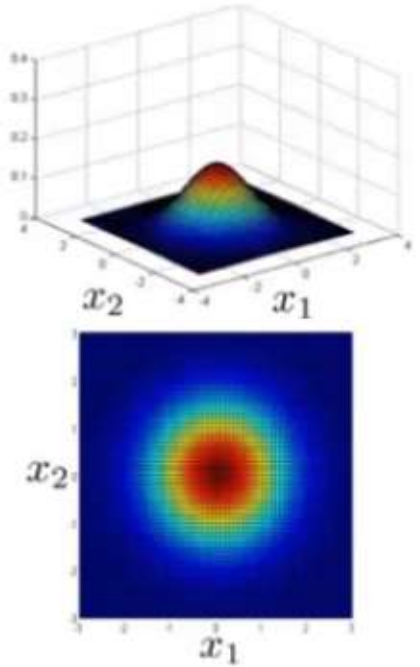


This show the initial problem where you use only one feature

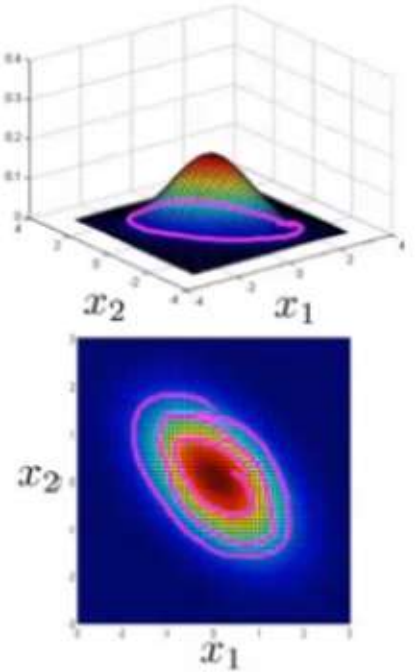


Multivariate Gaussian (Normal) examples

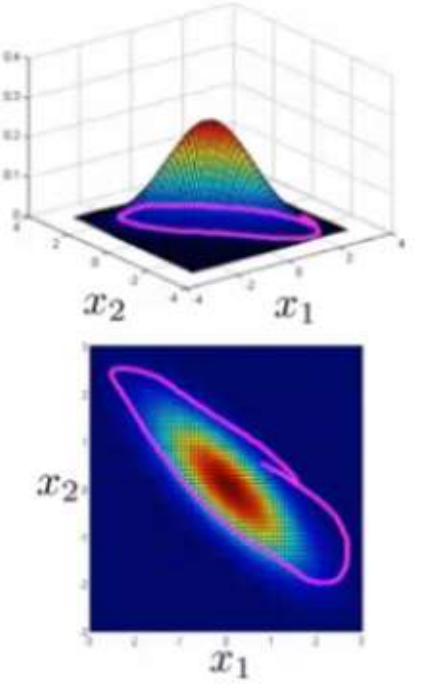
$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$



$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}$$

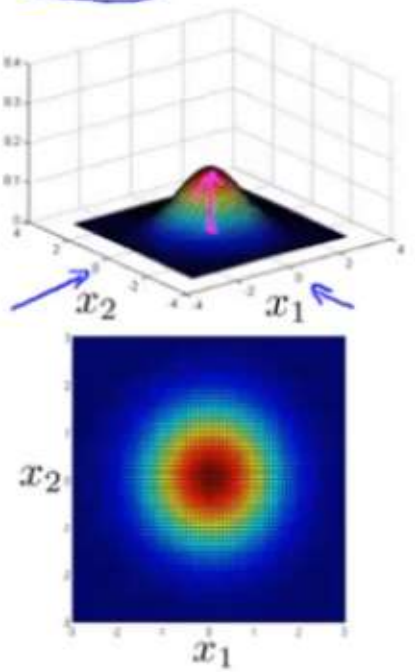


$$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & -0.8 \\ -0.8 & 1 \end{bmatrix}$$

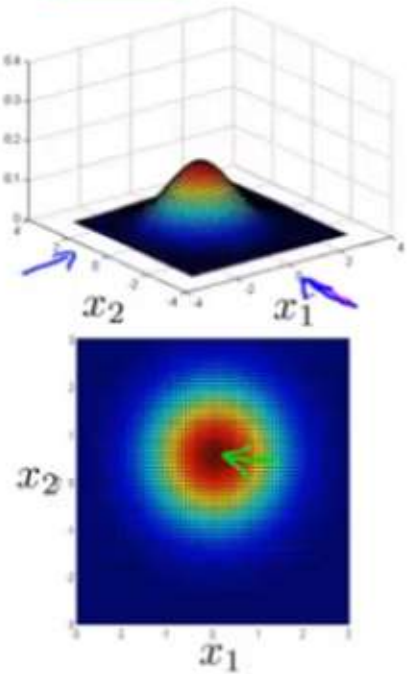


Multivariate Gaussian (Normal) examples

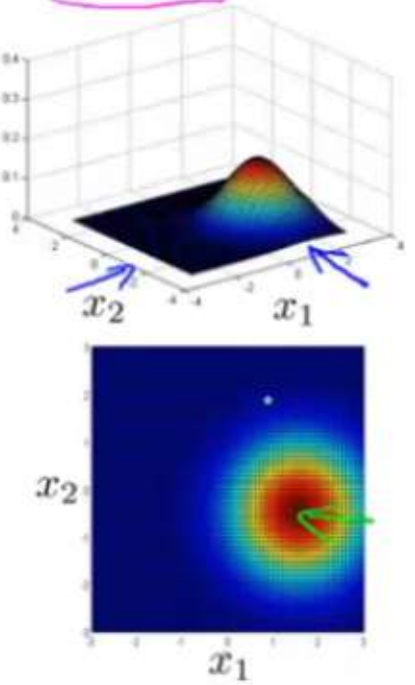
$\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$



$\mu = \begin{bmatrix} 0 \\ 0.5 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$



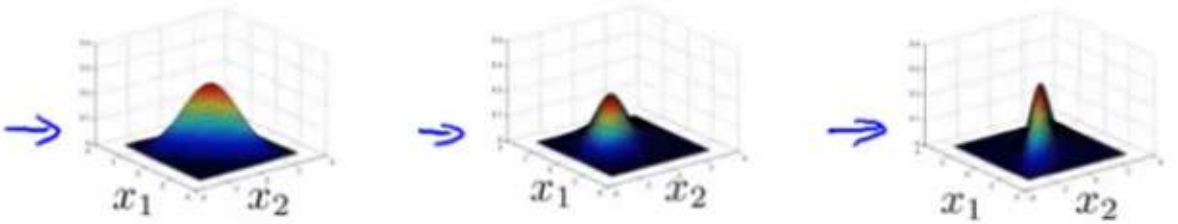
$\mu = \begin{bmatrix} 1.5 \\ -0.5 \end{bmatrix} \Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$



Multivariate Gaussian (Normal) distribution

Parameters μ, Σ $\quad \mu \in \mathbb{R}^n \quad \Sigma \in \mathbb{R}^{n \times n}$

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



Parameter fitting:

Given training set $\{x^{(1)}, x^{(2)}, \ldots, x^{(m)}\}$ $\quad x \in \mathbb{R}^n$

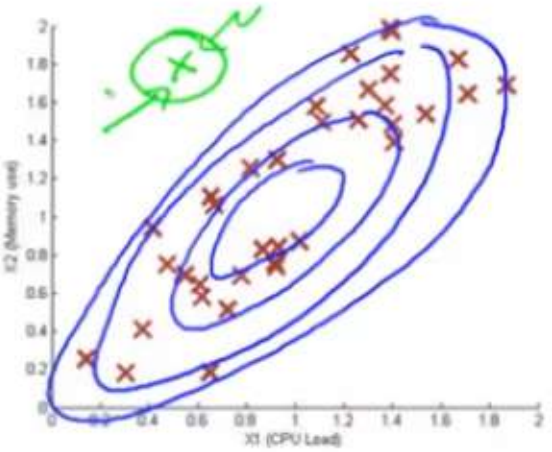
$$\mu = \frac{1}{m} \sum_{i=1}^{m} x^{(i)} \qquad \Sigma = \frac{1}{m} \sum_{i=1}^{m} (x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

Anomaly detection with the multivariate Gaussian

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

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

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



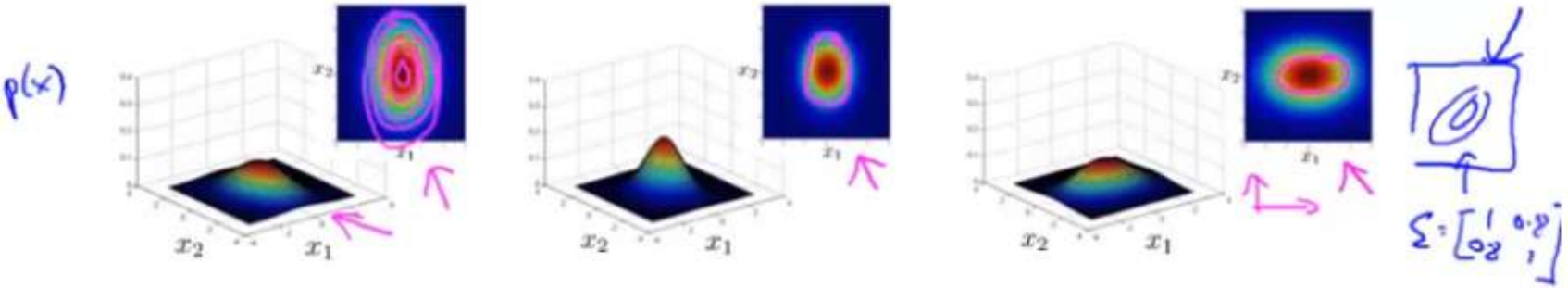
2. Given a new example x, compute

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

Flag an anomaly if $p(x) < \varepsilon$

Relationship to original model

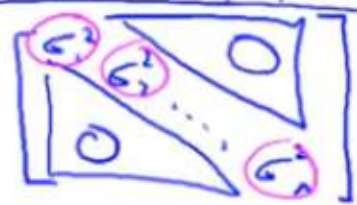
Original model: $p(x) = p(x_1; \mu_1, \sigma_1^2) \times p(x_2; \mu_2, \sigma_2^2) \times \cdots \times p(x_n; \mu_n, \sigma_n^2)$



Corresponds to multivariate Gaussian

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

where $\Sigma =$



Original model vs. Multivariate Gaussian

$$p(x_1;\mu_1,\sigma_1^2)\times\cdots\times p(x_n;\mu_n,\sigma_n^2)$$

Manually create features to capture anomalies where x_1, x_2 take unusual combinations of values.

$x_3 = \frac{x_1}{x_2} = \frac{\text{CPU load}}{\text{memory}}$

This step is very useful, just create your features that fit the anormal values

Computationally cheaper (alternatively, scales better to large n) n=10,000, n=100,000

OK even if m (training set size) is small

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

Automatically captures correlations between features

$\Sigma \in \mathbb{R}^{n\times n}$ Σ^{-1}

Computationally more expensive

$\Sigma \sim \frac{n^2}{2}$

$x_1 = x_2$, $x_3 = x_4 + x_5$

Must have $m > n$ or else Σ is non-invertible.

$m \geq 10n$

Use only if "m" is much greater then "n"

if the above true, the sigam will be non-invertable