



ФИСТ
УлГТУ

Machine Learning

Lesson 04. Basics of Machine Learning

Kirill Svyatov

Ulyanovsk State Technical University,
Faculty of Information Systems and Technologies

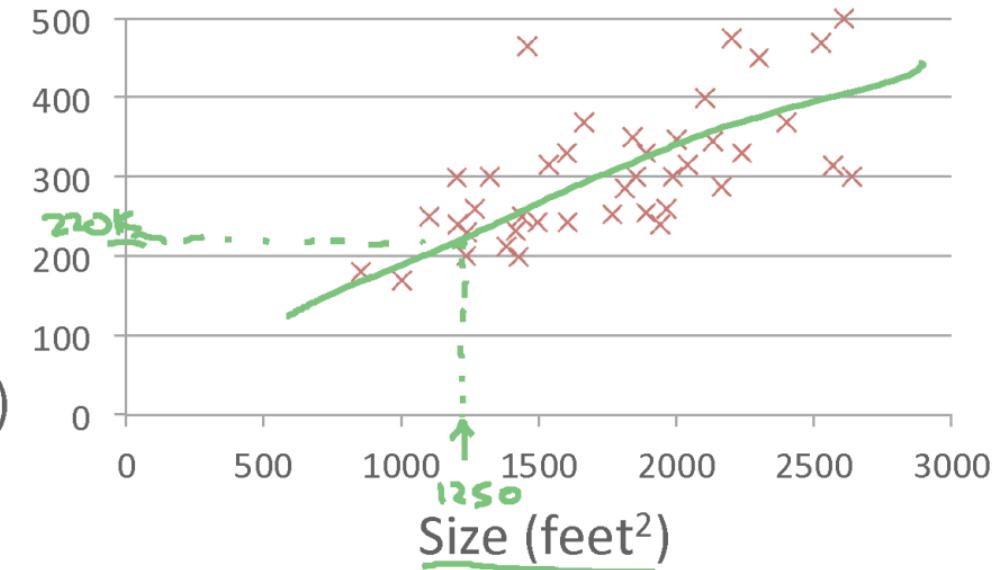


Linear regression with one variable

Model representation

Housing Prices

Price
(in 1000s
of dollars)



Supervised Learning

Given the “right answer” for each example in the data.

Regression Problem

Predict real-valued output

Classification: Discrete-valued output

Training set of housing prices
(Portland, OR)

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

Notation:

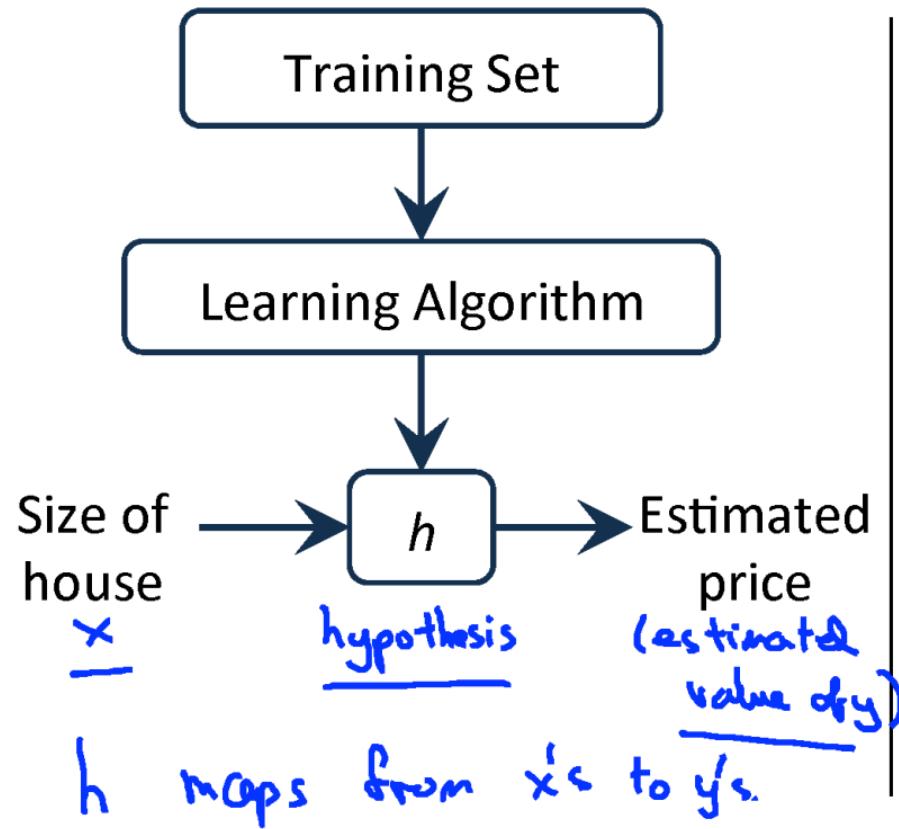
- m = Number of training examples
- x's = "input" variable / features
- y's = "output" variable / "target" variable

(x, y) - one training example

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

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

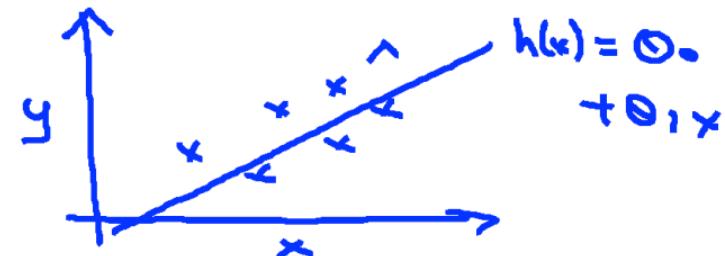
$m = 47$



How do we represent h ?

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

Shorthand: $h(x)$



Linear regression with one variable. (x)

Univariate linear regression.

one variable

Cost function



Training Set

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

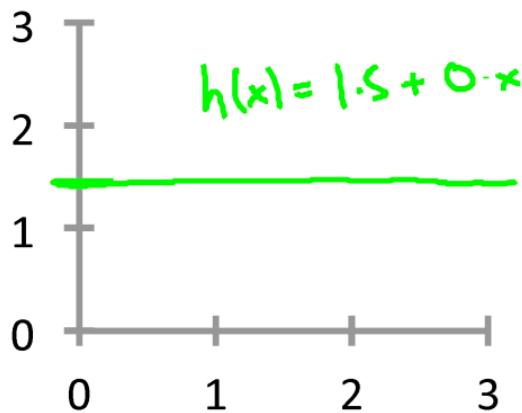
$m = 47$

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

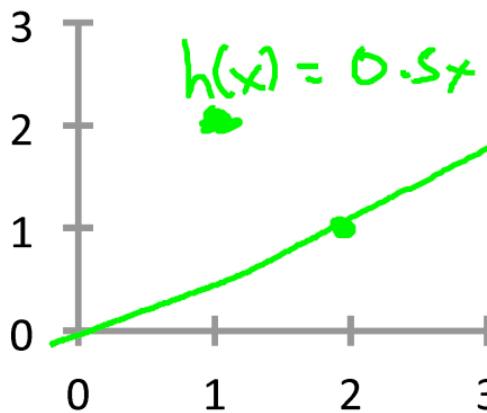
θ_i 's: Parameters

How to choose θ_i 's ?

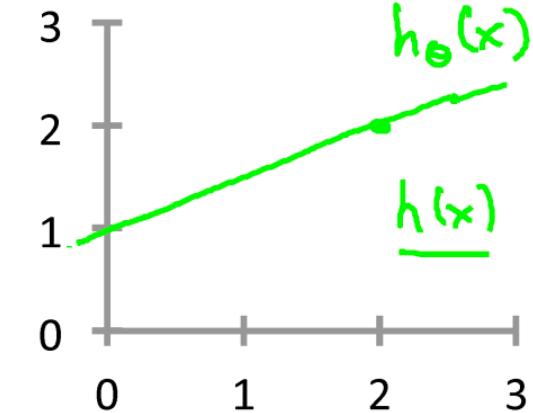
$$\underline{h_{\theta}(x)} = \theta_0 + \theta_1 x$$



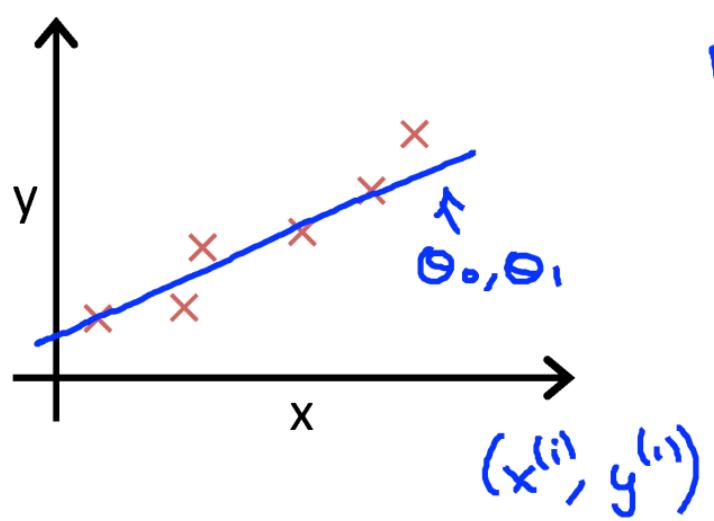
$$\begin{aligned} \rightarrow \theta_0 &= 1.5 \\ \rightarrow \theta_1 &= 0 \end{aligned}$$



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



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



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

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

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

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

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

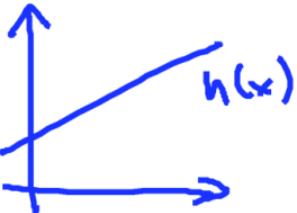
minimize $\underline{\theta_0, \theta_1}$ $J(\underline{\theta_0, \theta_1})$
 Cost function
 Squared error function

Hypothesis:

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

Parameters:

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



Cost Function:

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

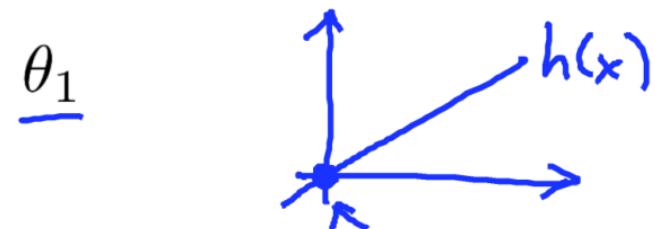
Goal: minimize $J(\theta_0, \theta_1)$

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

Simplified

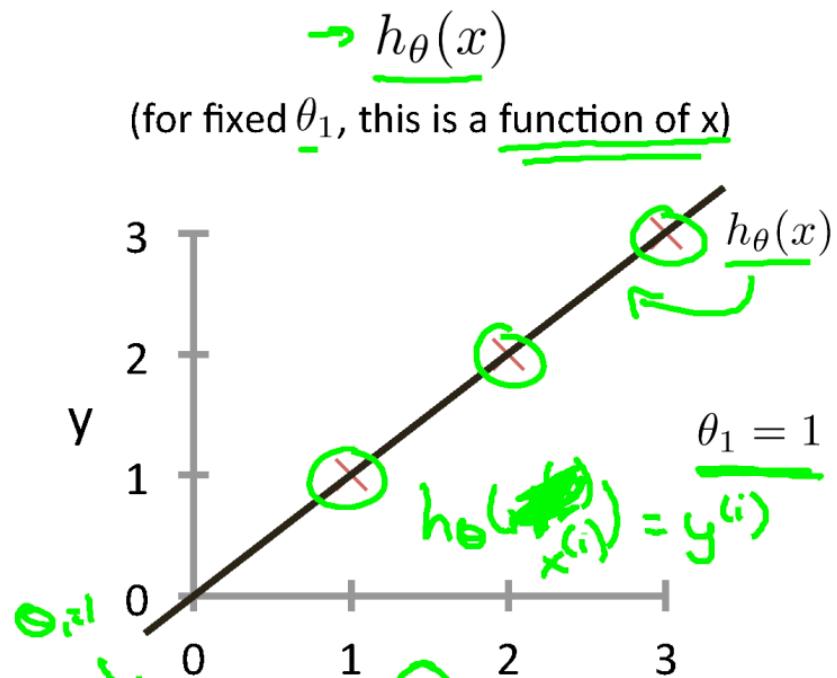
$$h_{\theta}(x) = \underline{\theta_1 x}$$

$$\theta_0 = 0$$

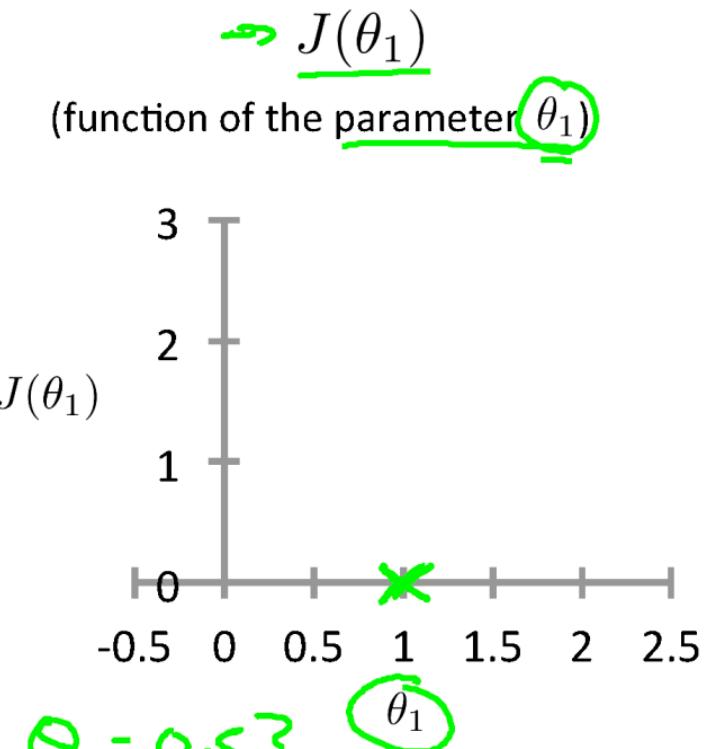


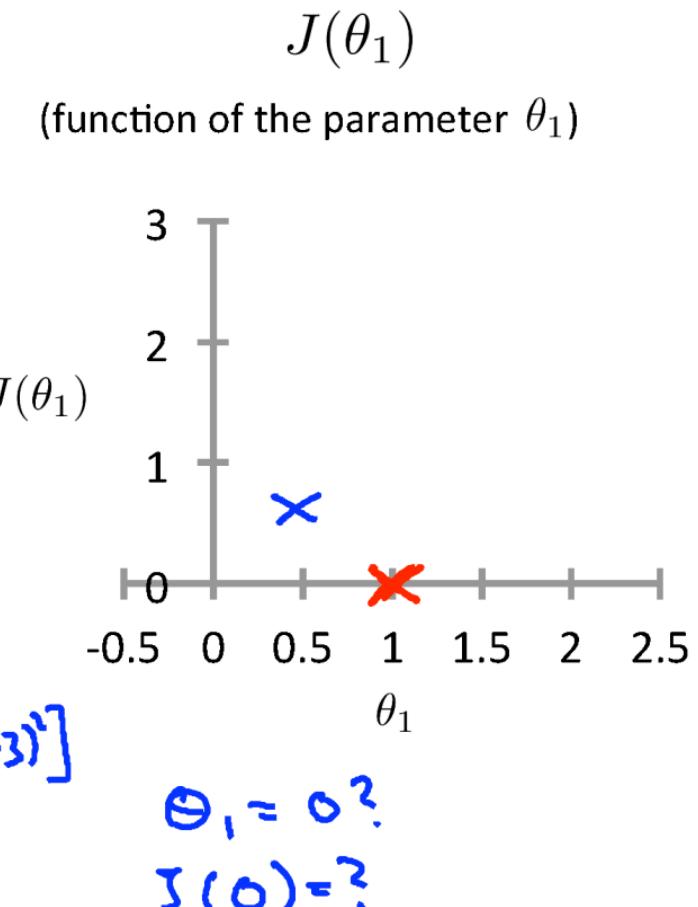
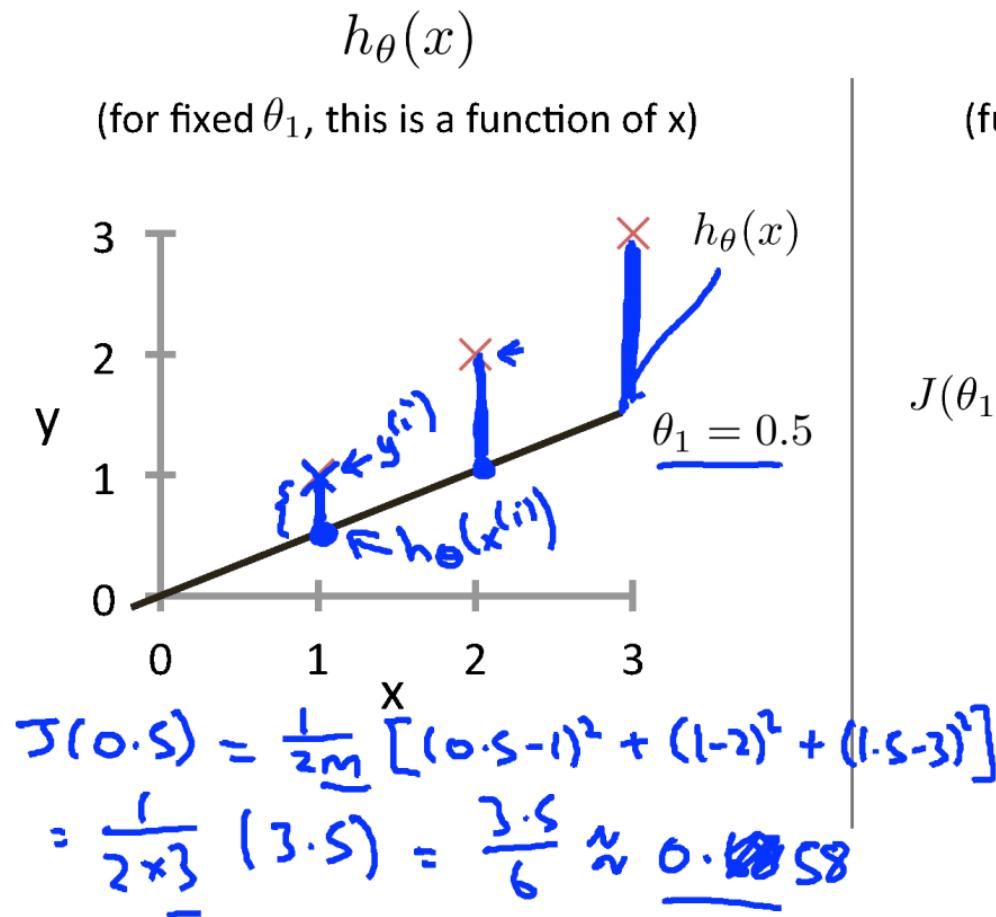
$$J(\theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

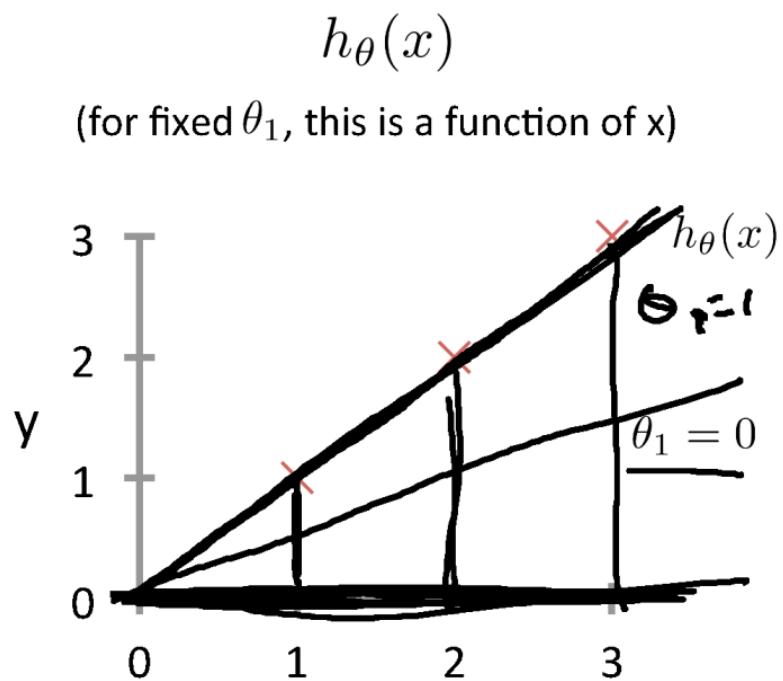
$$\underline{\text{minimize}} \underline{\underline{J(\theta_1)}} \quad \theta_1 \quad \theta_1, x^{(i)}$$



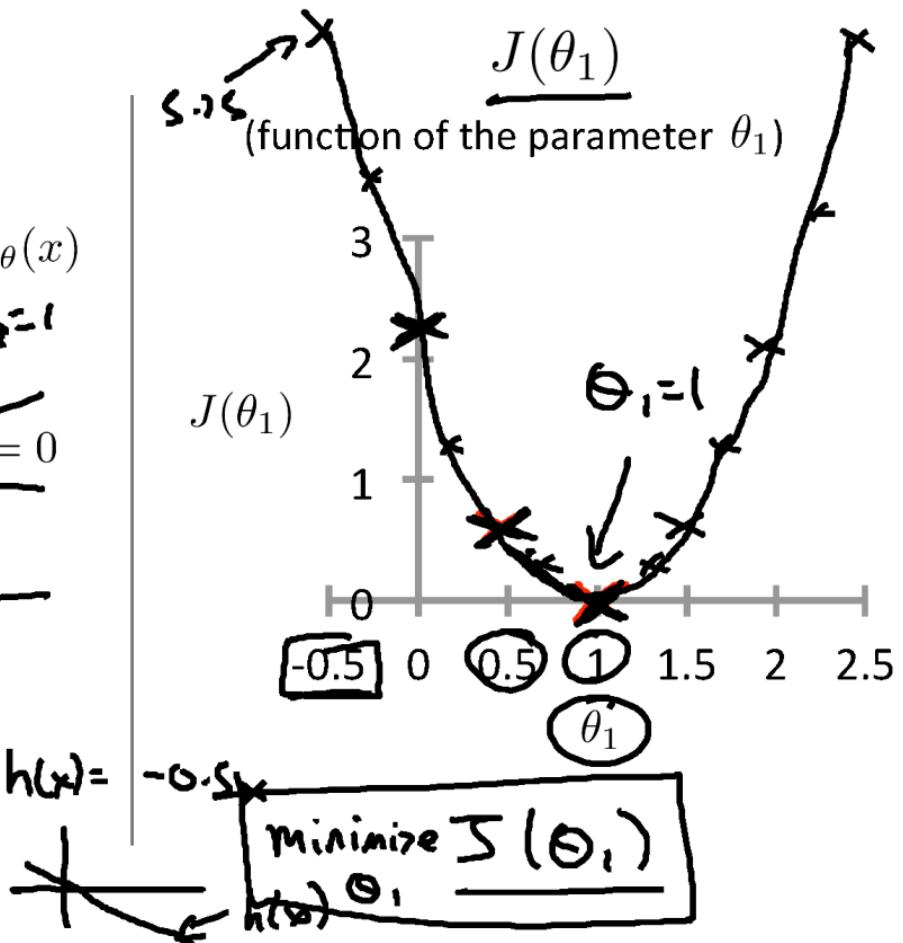
$$\begin{aligned}
 J(\theta_1) &= \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 \\
 &= \frac{1}{2m} \sum_{i=1}^m (\theta_1 x^{(i)} - y^{(i)})^2 \approx \frac{1}{2m} (0^2 + 0^2 + 0^2) = 0^2
 \end{aligned}$$







$$\mathcal{J}(\theta) = \frac{1}{2m} (1^2 + 2^2 + 3^2) \\ = \frac{1}{6} \cdot 14 \approx 2.3$$



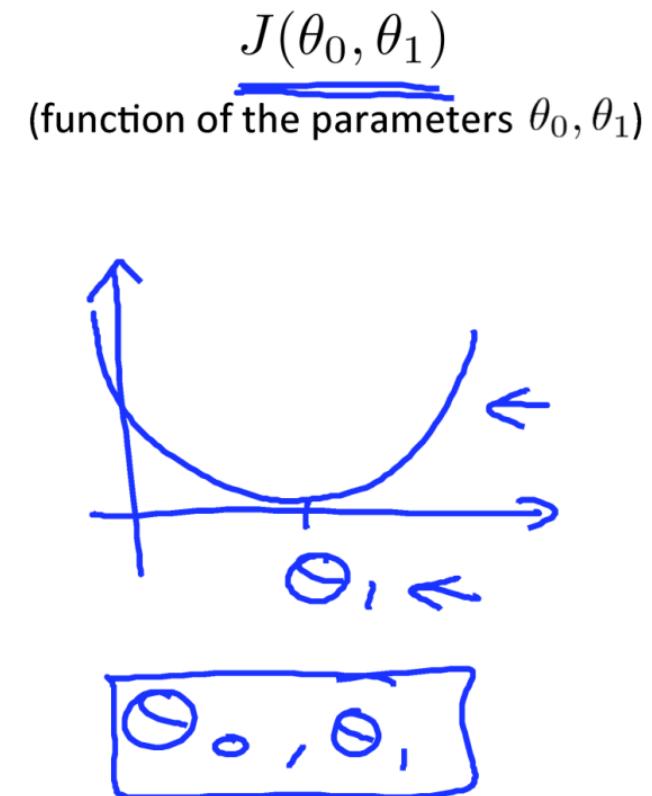
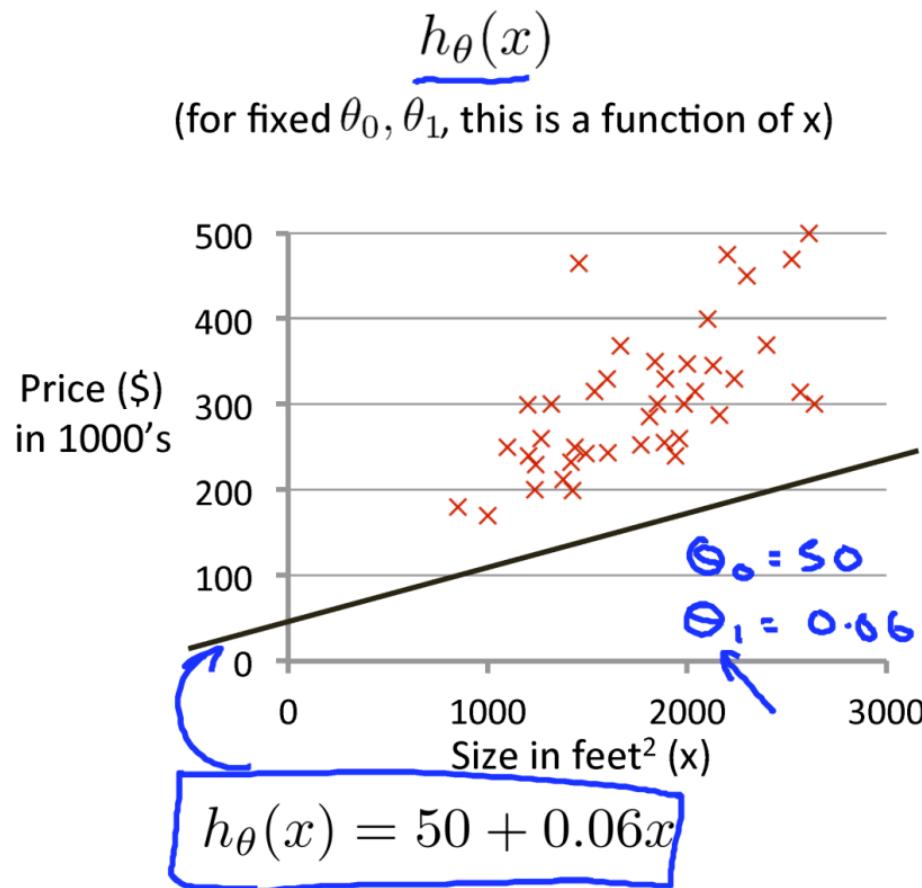
Hypothesis: $h_{\theta}(x) = \theta_0 + \theta_1 x$

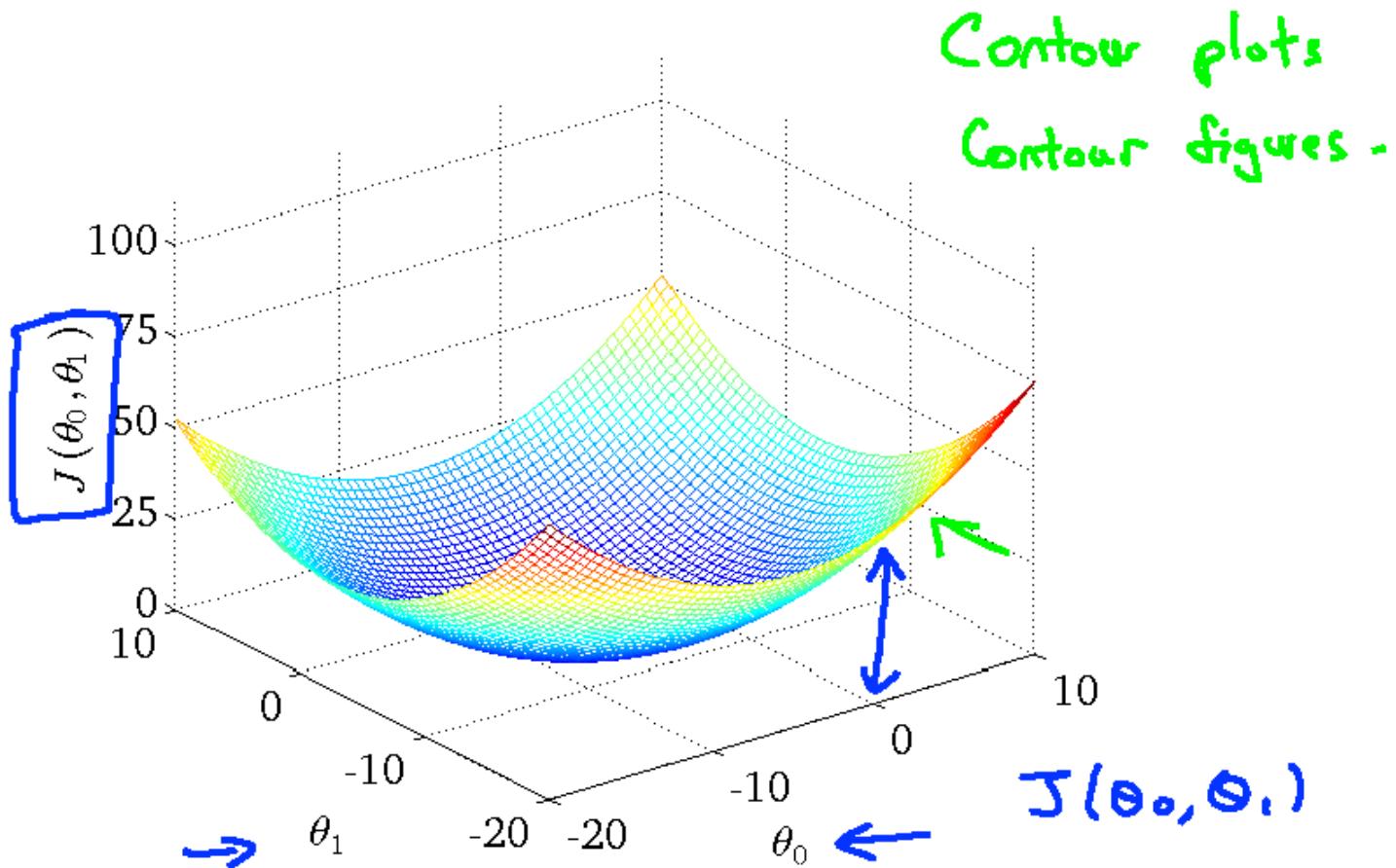
Parameters: θ_0, θ_1

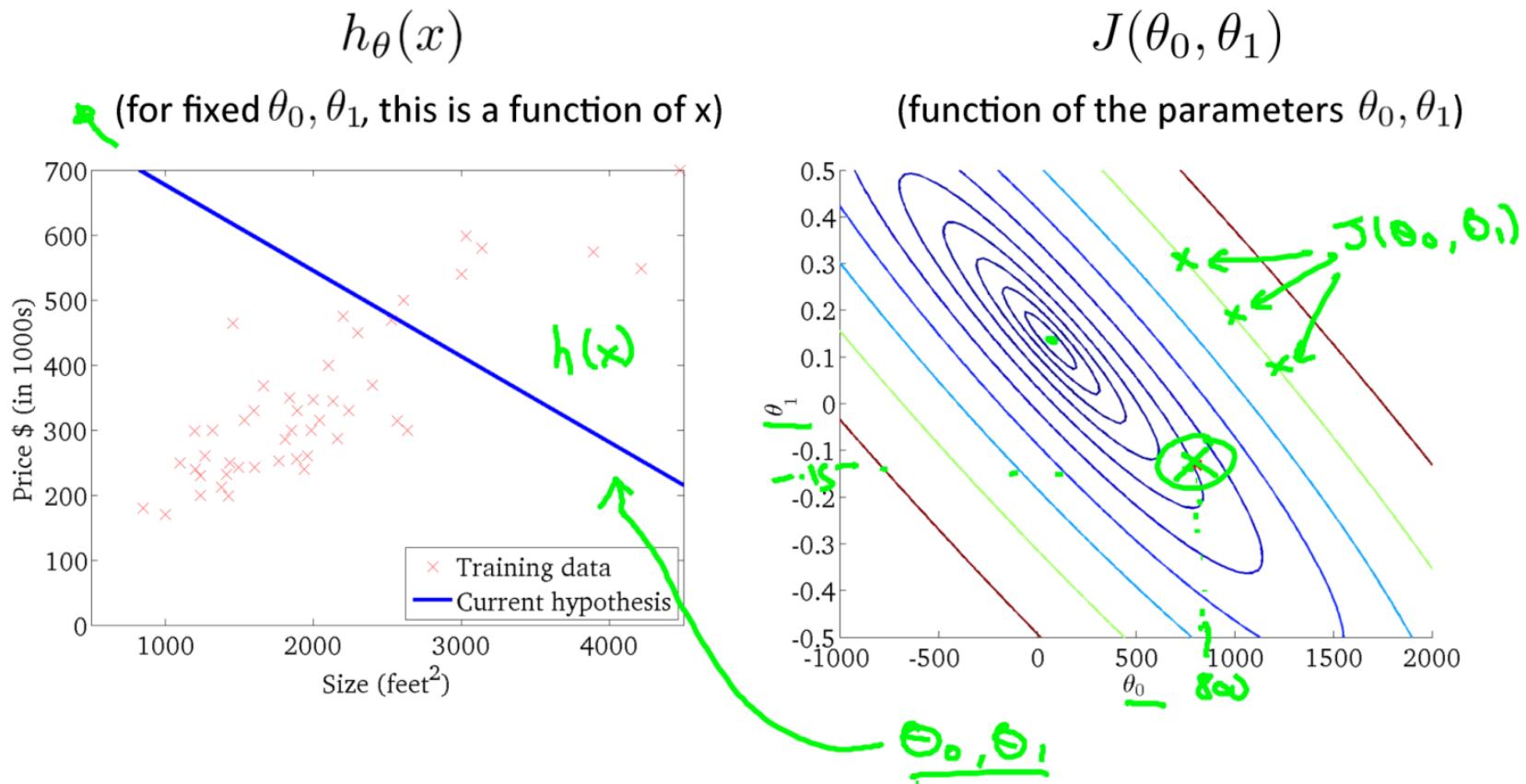
Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$

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

.

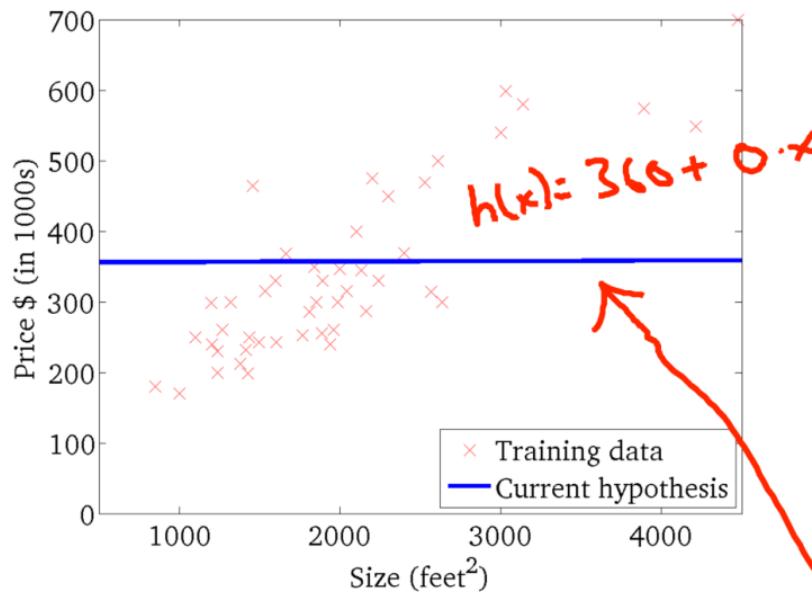






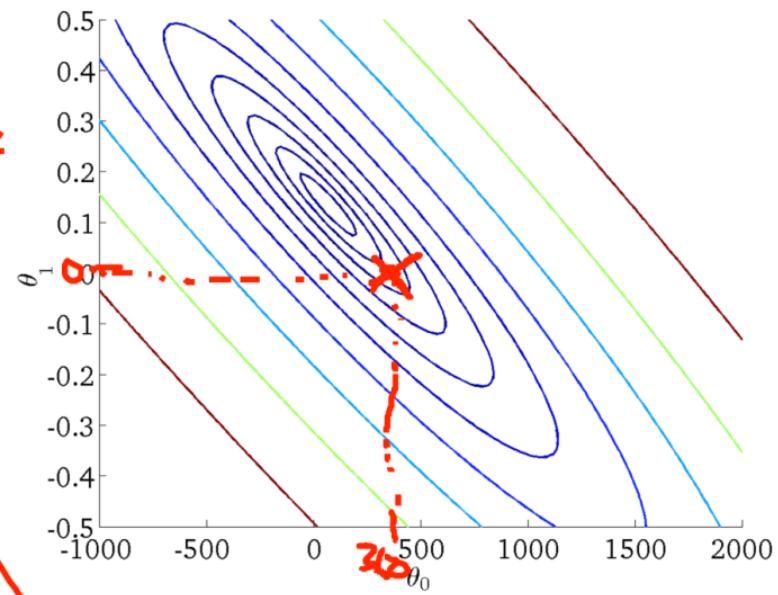
$$h_{\theta}(x)$$

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



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

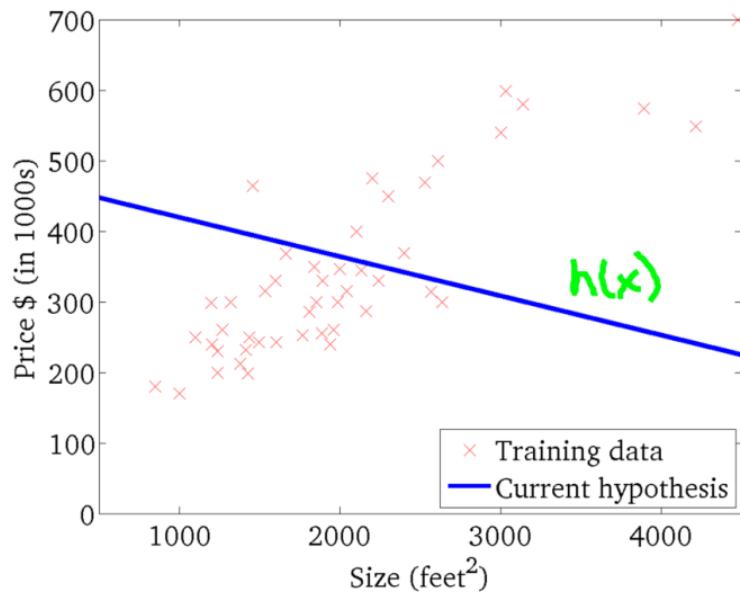
(function of the parameters θ_0, θ_1)



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

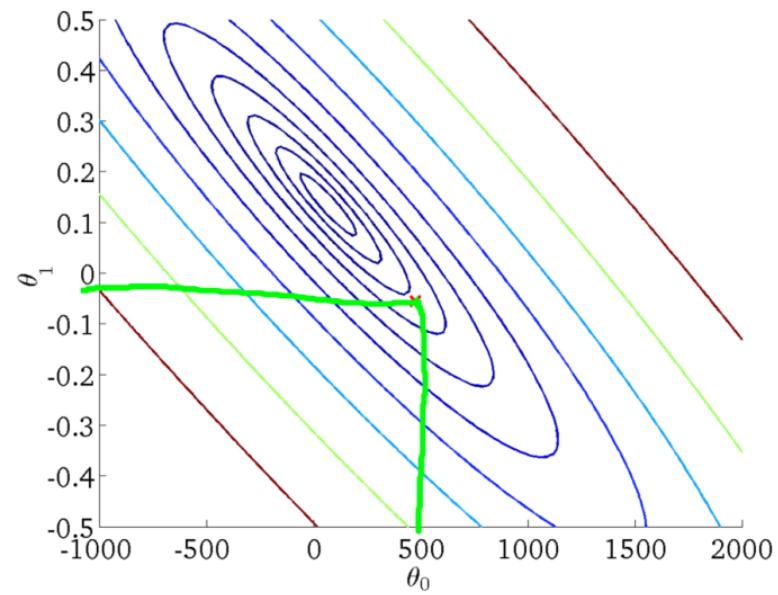
$$h_{\theta}(x)$$

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



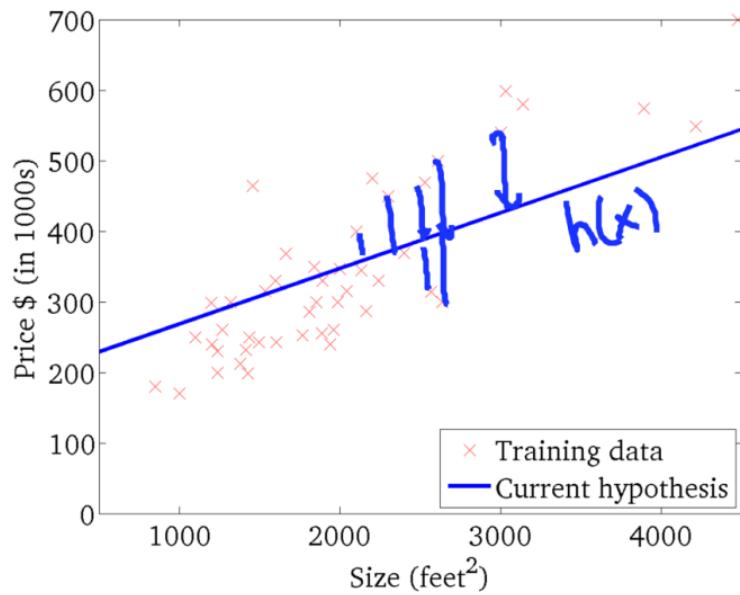
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



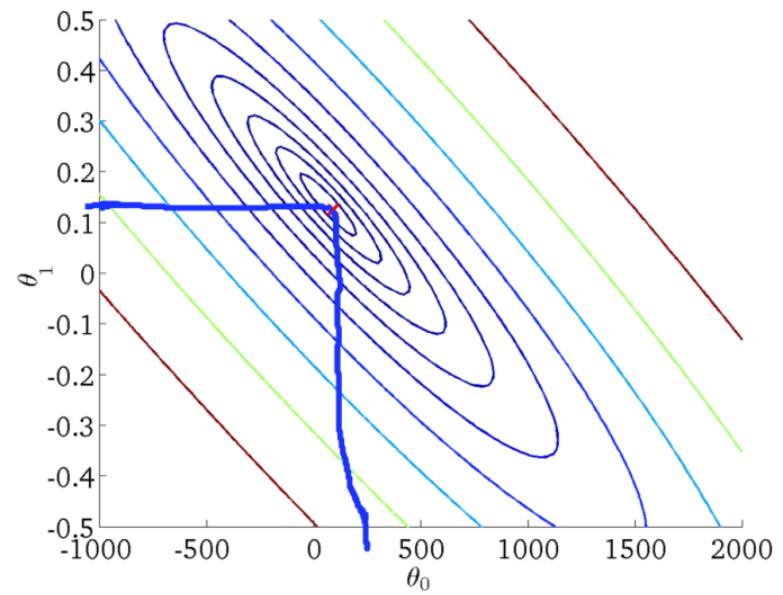
$$h_{\theta}(x)$$

(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)



Gradient descent

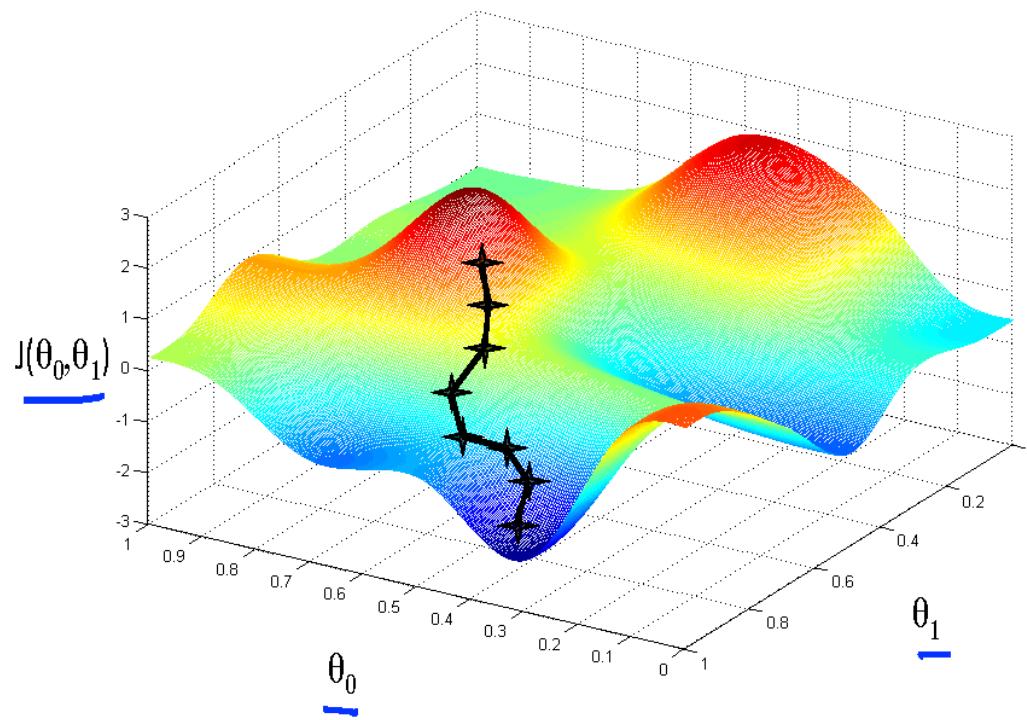


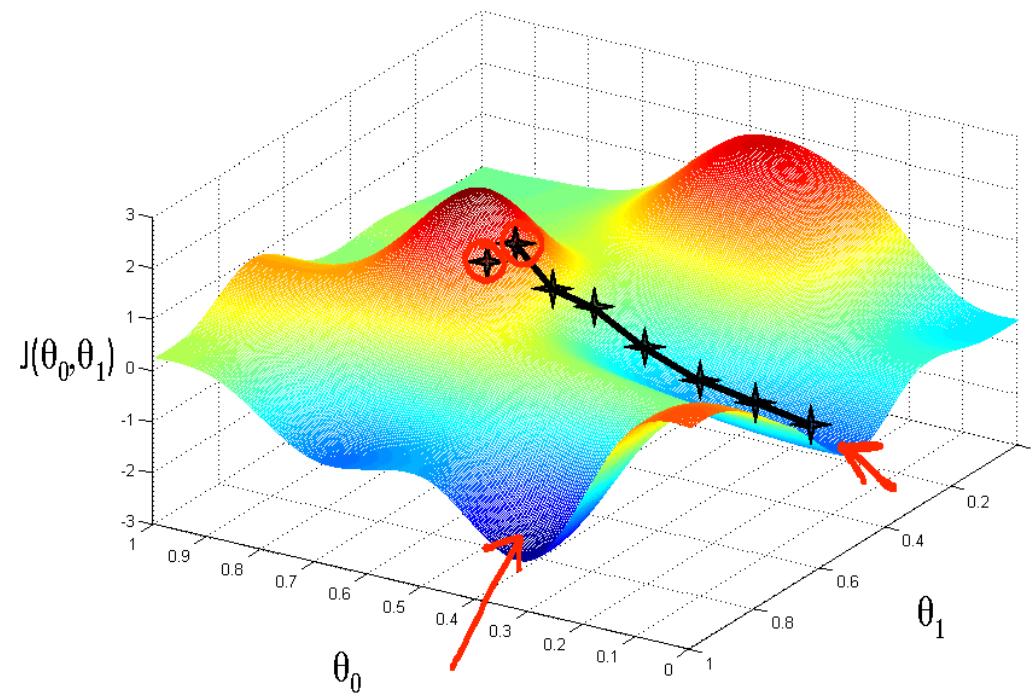
Have some function $\underline{J(\theta_0, \theta_1)}$ $\mathcal{J}(\theta_0, \theta_1, \theta_2, \dots, \theta_n)$

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

Outline:

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





Gradient descent algorithm

repeat until convergence {

```

    [ →  $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ 
    } ← learning rate
  }
```

θ_0, θ_1

Assignment

$$\begin{array}{l} a := b \\ \quad \uparrow \\ a := a + 1 \end{array}$$

Truth assertion

$$\left| \begin{array}{l} a = b \leftarrow \\ a = a + 1 \times \end{array} \right.$$

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

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 {

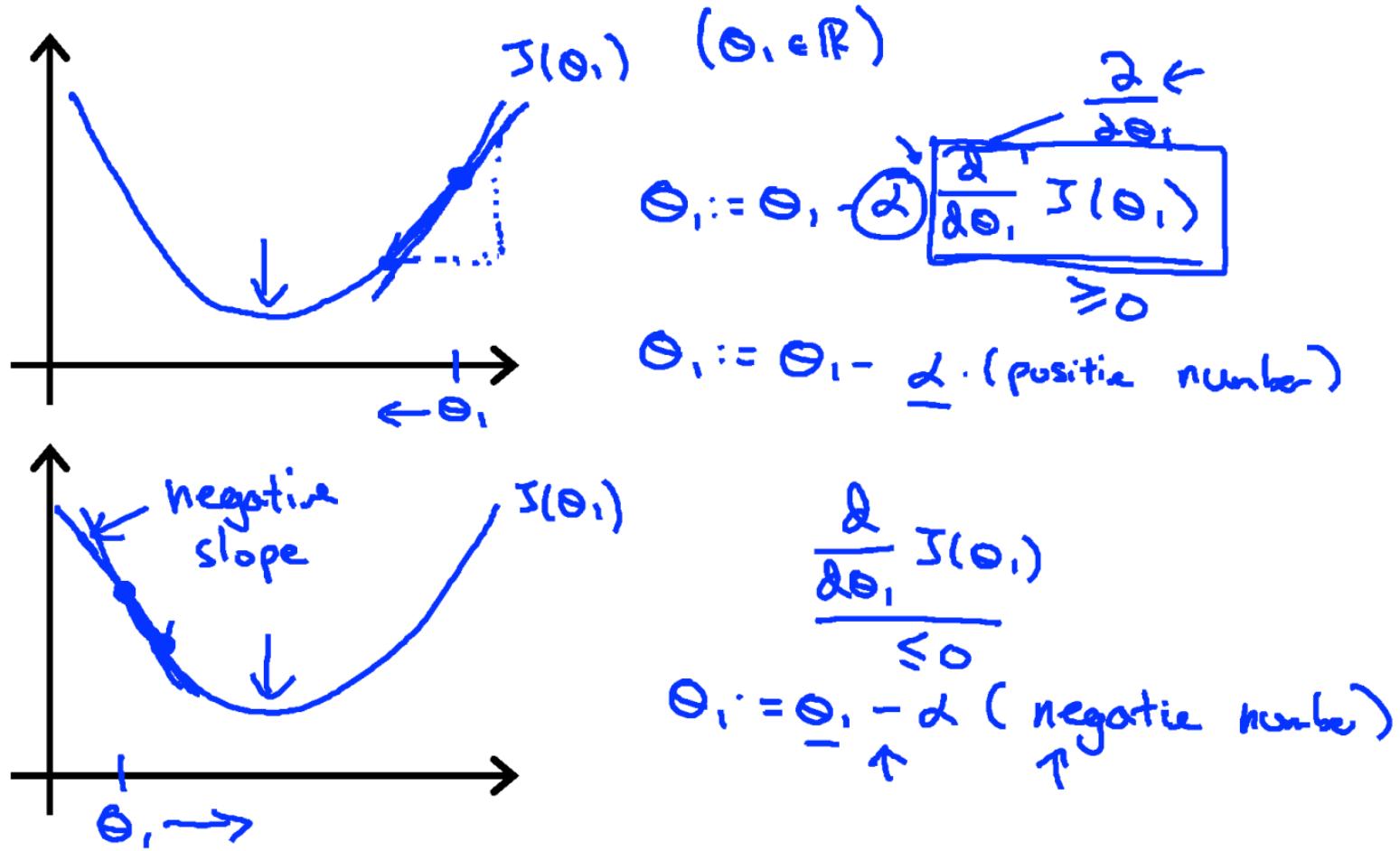
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

}

learning rate *derivative*

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

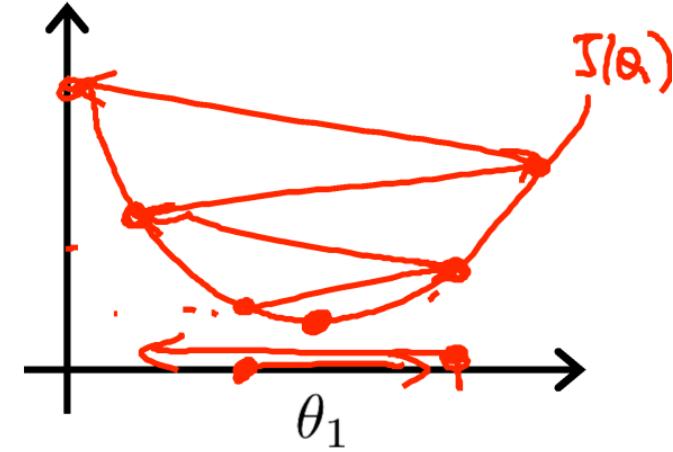
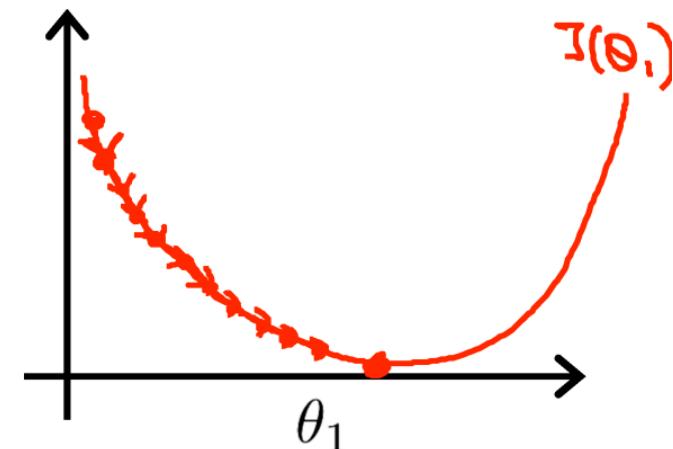
$$\min_{\theta_1} J(\theta_1) \quad \theta_1 \in \mathbb{R}.$$

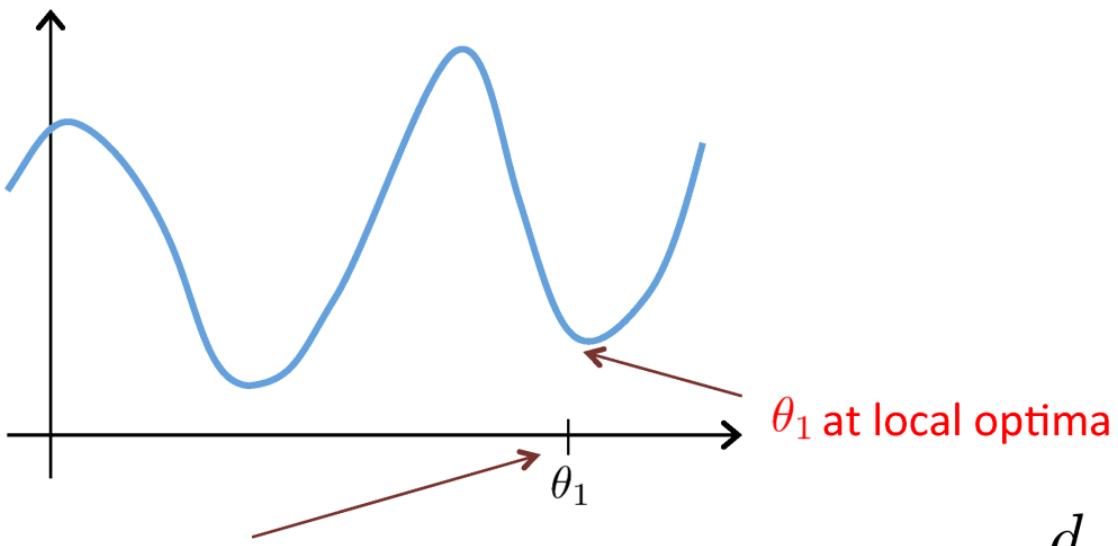


$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

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.



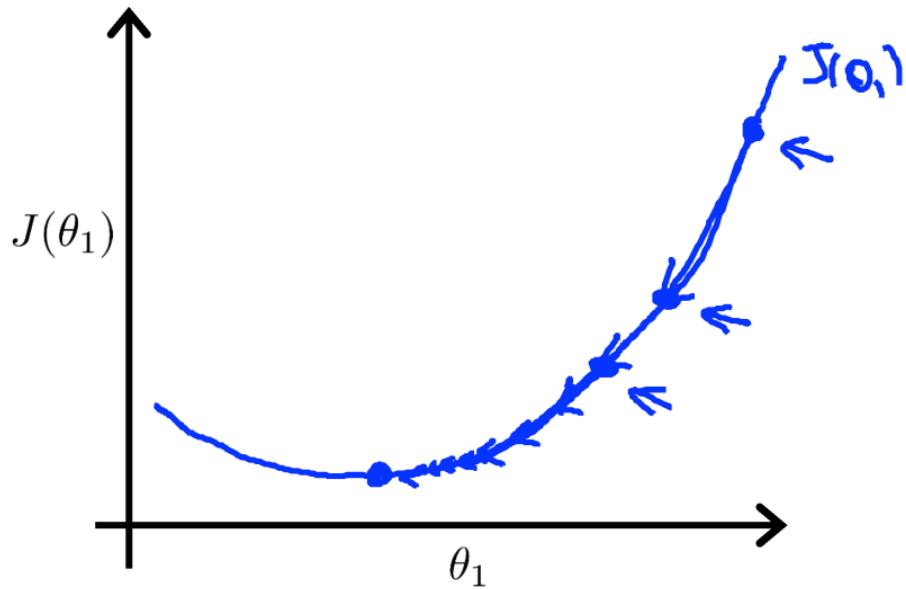


$$\theta_1 := \theta_1 - \alpha \frac{d}{d\theta_1} J(\theta_1)$$

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

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

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

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

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

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

Gradient descent algorithm

repeat until convergence {

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

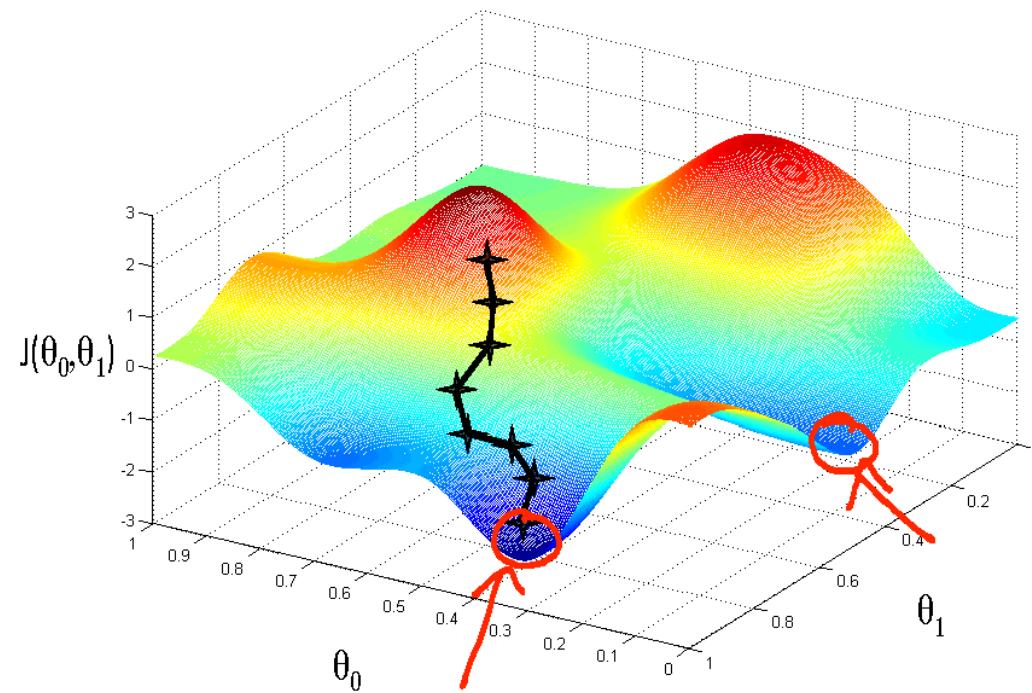
$$\theta_1 := \theta_1 - \alpha \left[\frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)} \right]$$

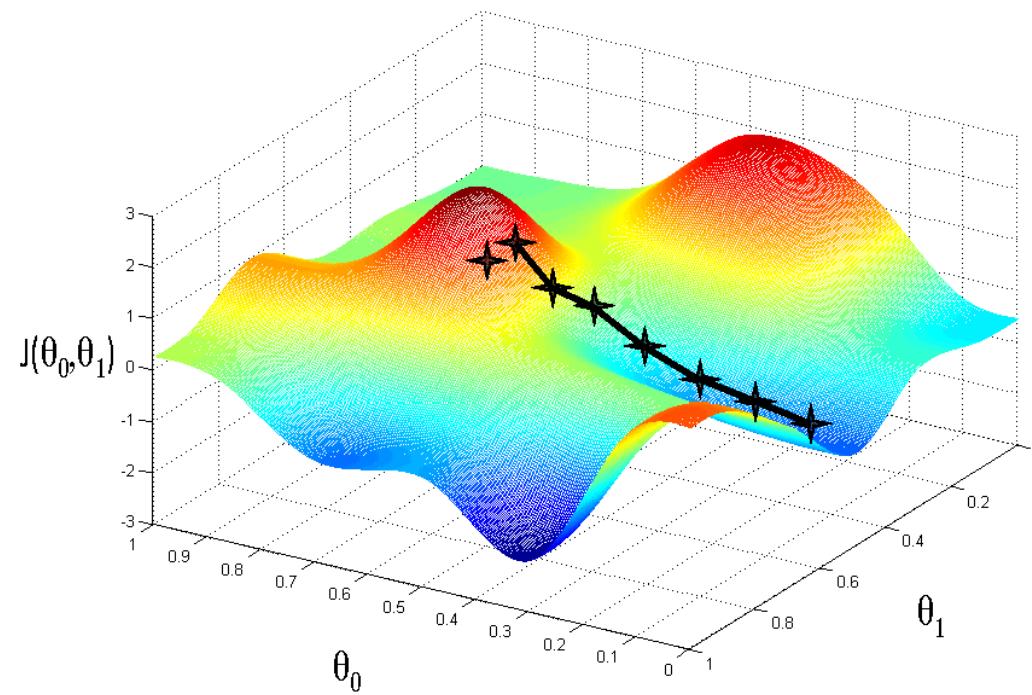
}

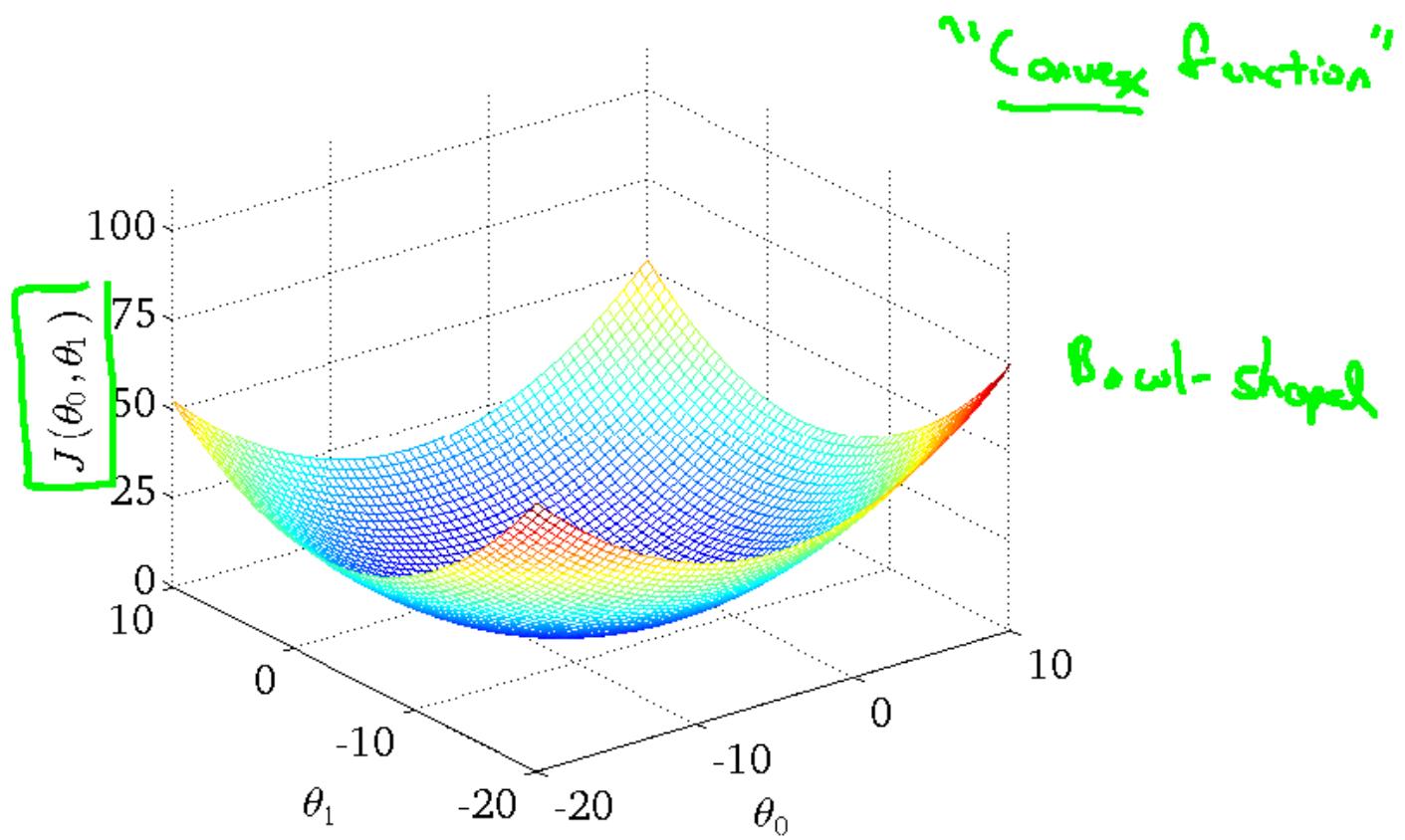
$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

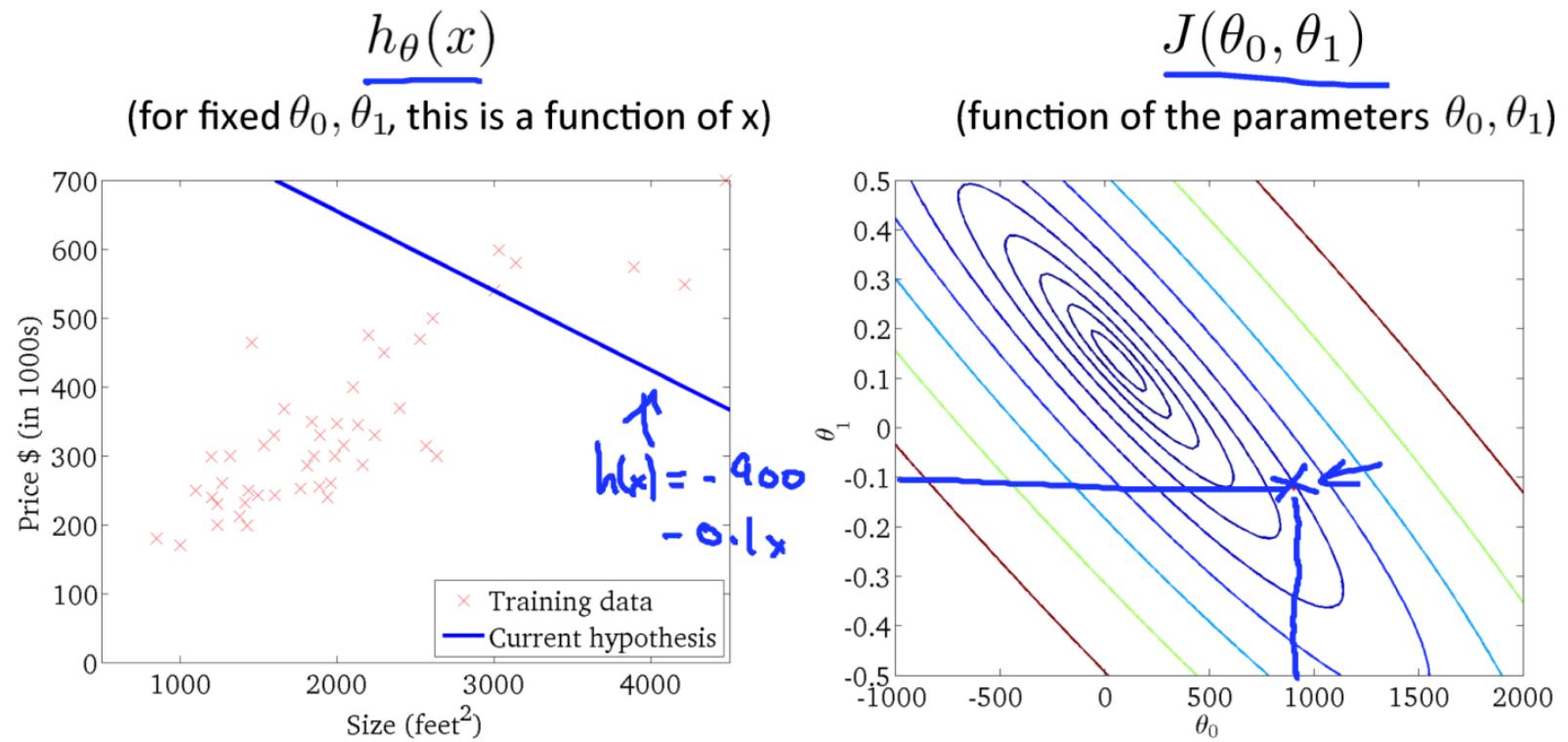
update
 θ_0 and θ_1
simultaneously

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



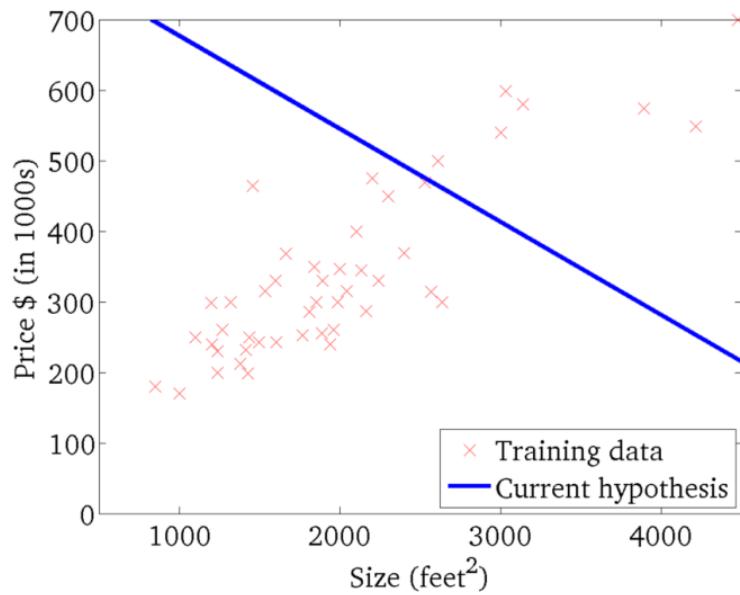






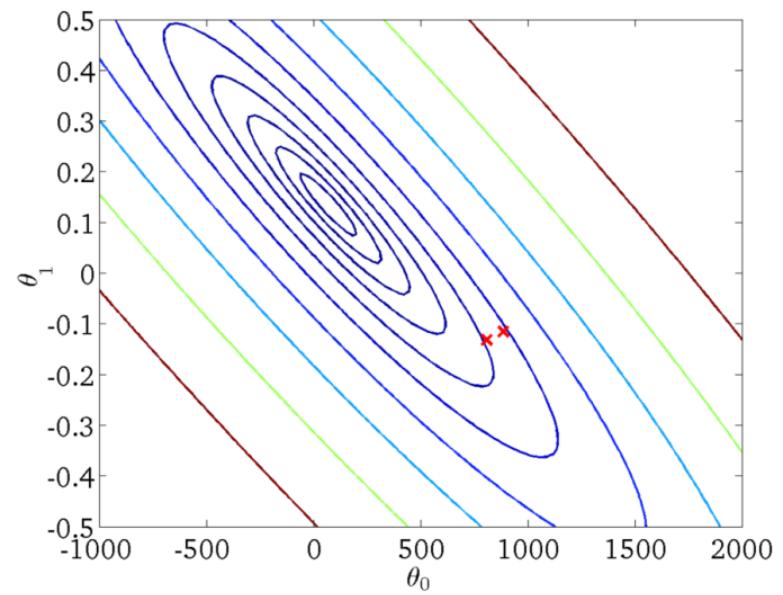
$$h_{\theta}(x)$$

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



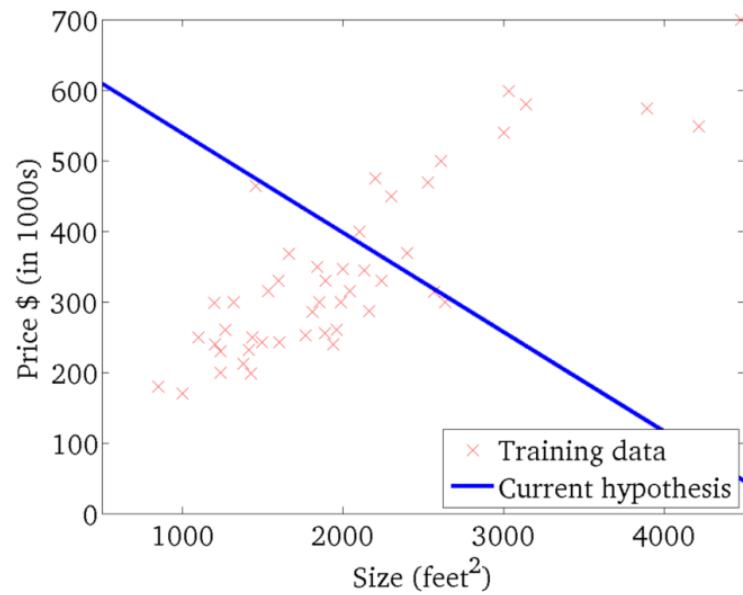
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



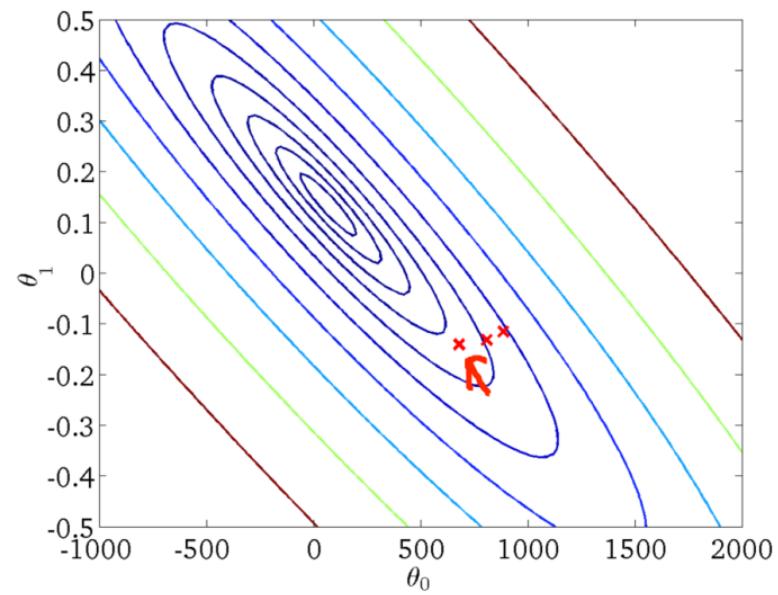
$$h_{\theta}(x)$$

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



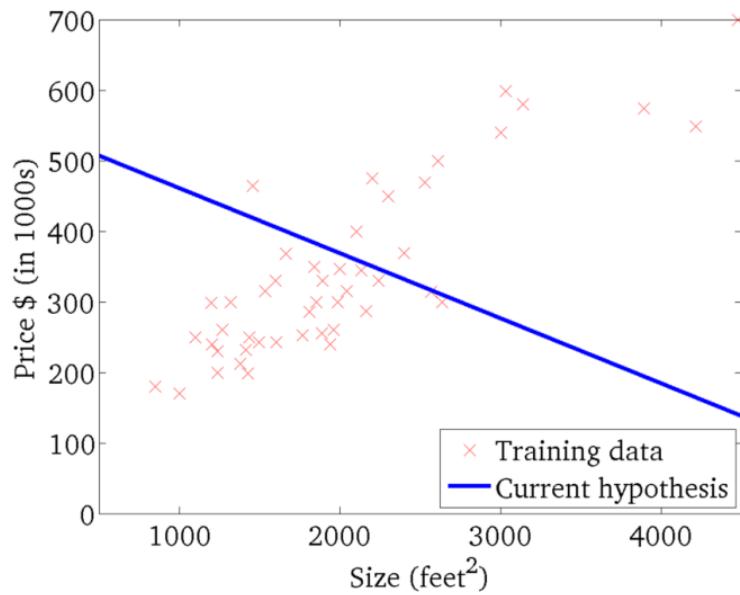
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



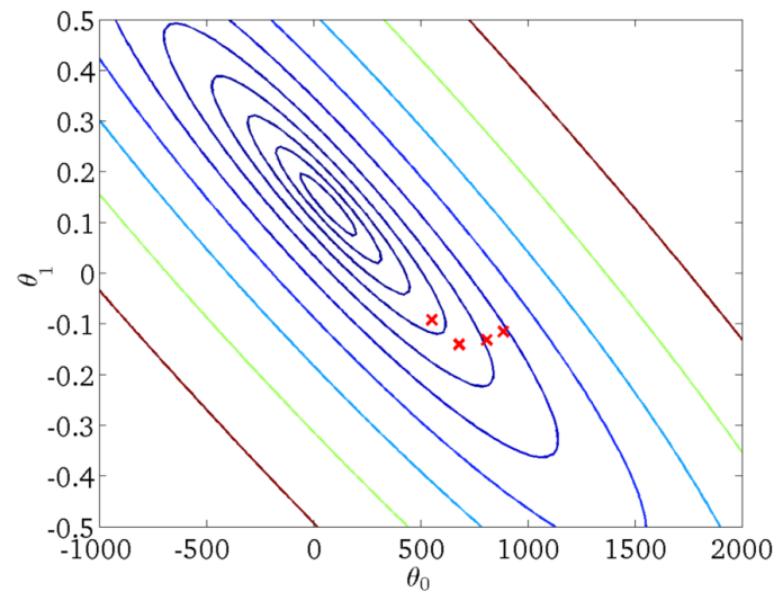
$$h_{\theta}(x)$$

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



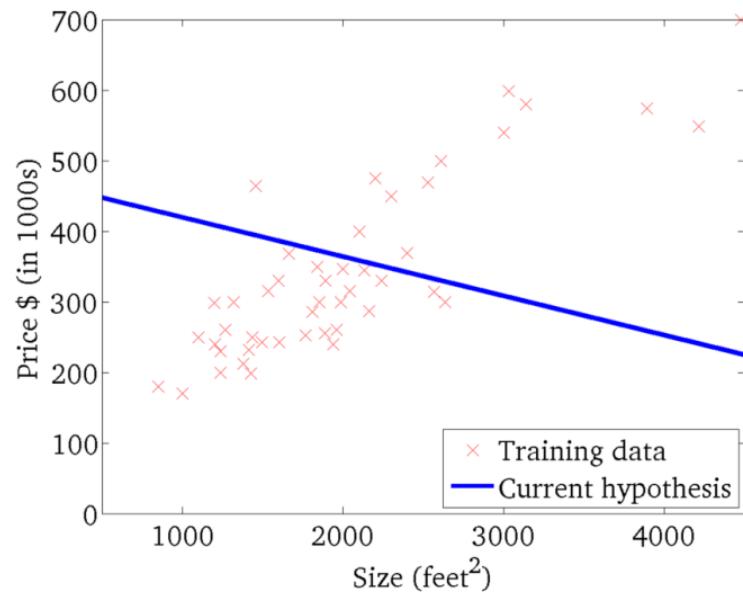
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



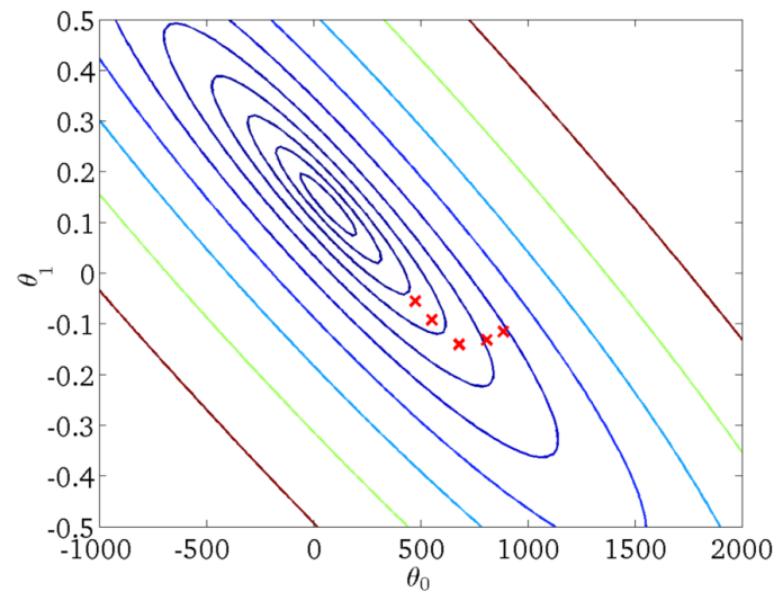
$$h_{\theta}(x)$$

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



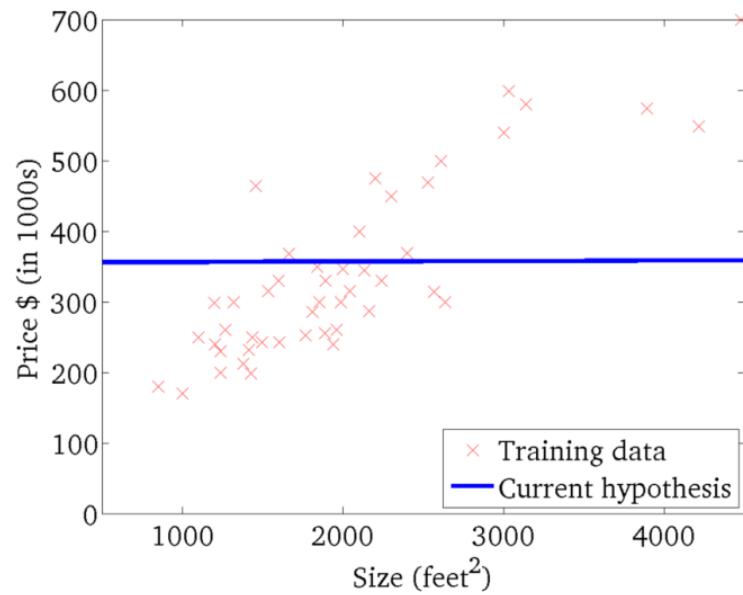
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



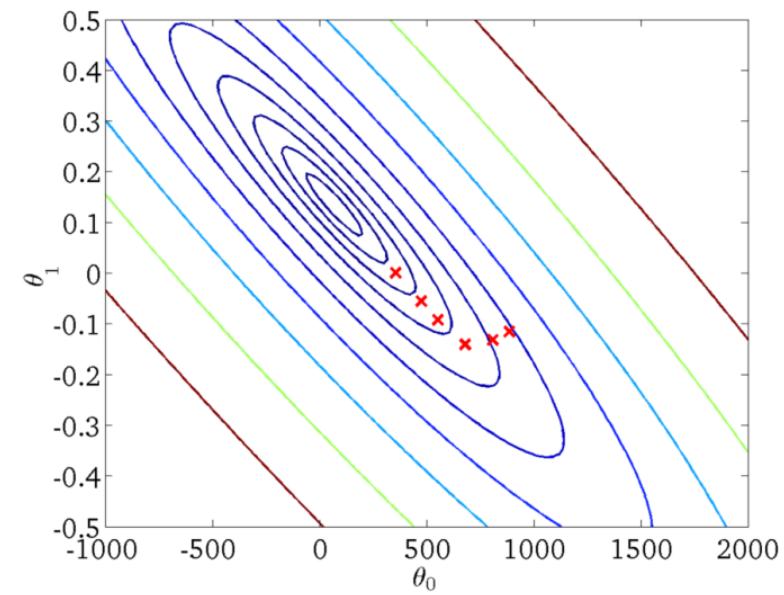
$$h_{\theta}(x)$$

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



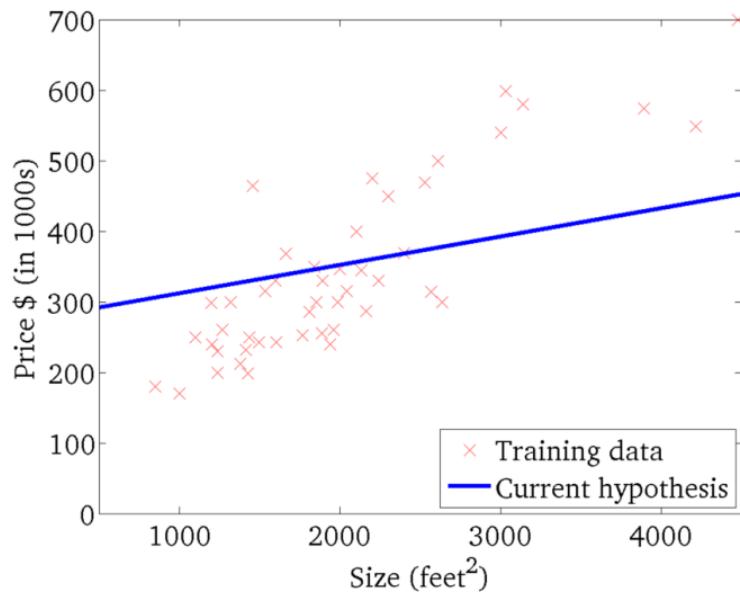
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



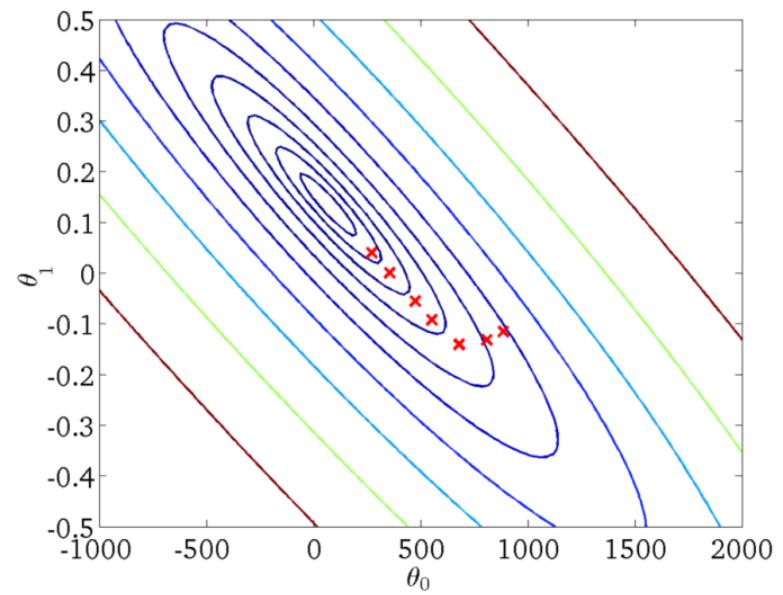
$$h_{\theta}(x)$$

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



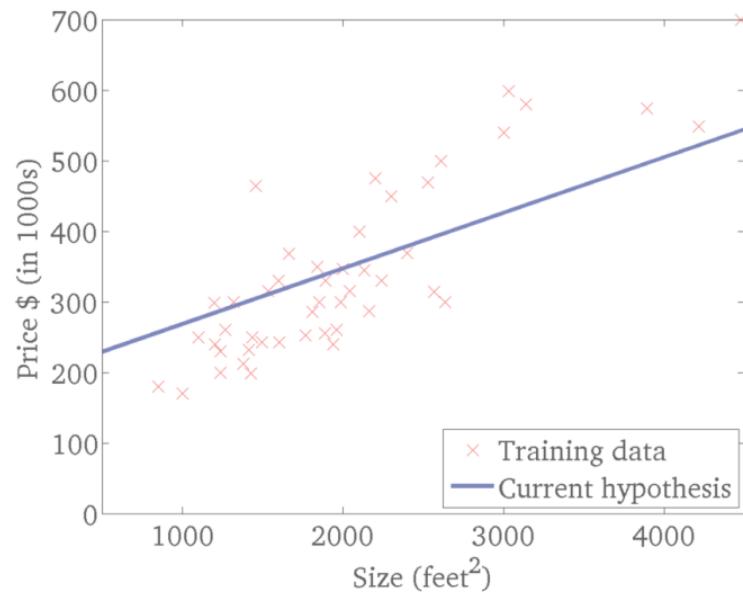
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



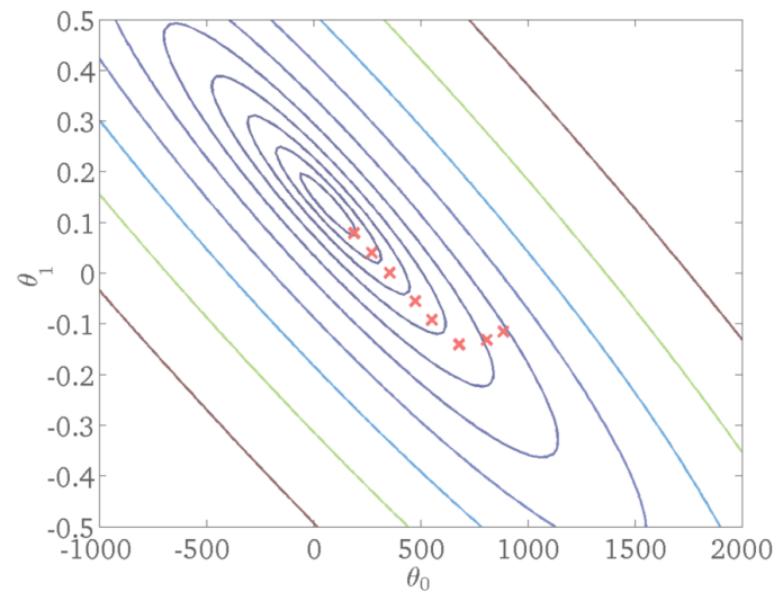
$$h_{\theta}(x)$$

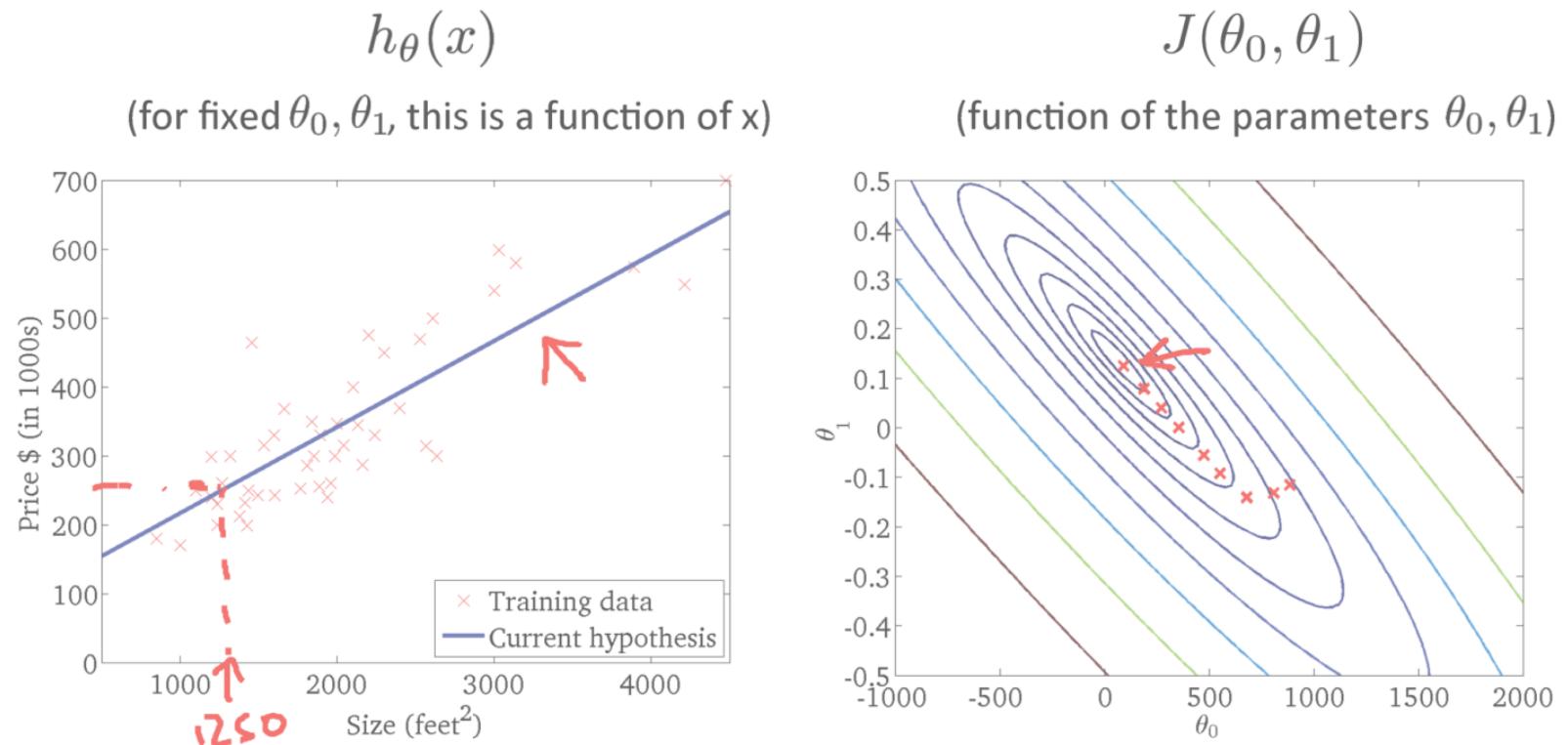
(for fixed θ_0, θ_1 , this is a function of x)



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

(function of the parameters θ_0, θ_1)





“Batch” Gradient Descent

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

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

Linear Regression with multiple variables

Multiple features

Multiple features (variables).

Size (feet ²)	Price (\$1000)
\xrightarrow{x}	$y \xleftarrow{ }$
2104	460
1416	232
1534	315
852	178
...	...

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

Multiple features (variables).

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

Notation:

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

$\left[\begin{array}{c} x^{(2)} \\ x_3^{(2)} \end{array} \right] = \left[\begin{array}{c} 1416 \\ 3 \\ 2 \\ 40 \end{array} \right]$

$x_3^{(2)} = 2$

Hypothesis:

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

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

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

↑ ↑ ↑
 age

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

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

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

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

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

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

Multivariate linear regression. 

Gradient descent for multiple variables



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

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

Cost function:

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

Gradient descent:

Repeat {

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

}

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

Gradient Descent

Previously (n=1):

Repeat {

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

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

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

(simultaneously update θ_0, θ_1)

}

New algorithm ($n \geq 1$):

Repeat {

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

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

}

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

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

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

...



Gradient descent in practice I: Feature Scaling

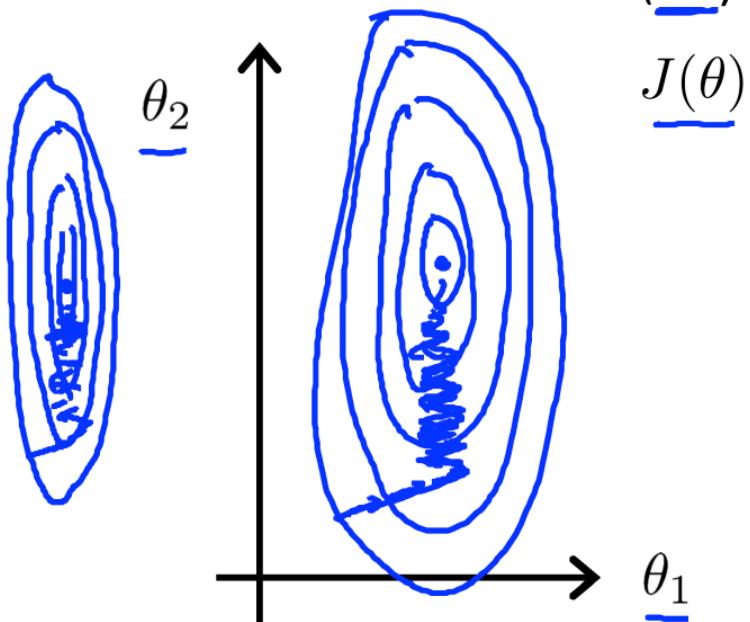


Feature Scaling

Idea: Make sure features are on a similar scale.

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

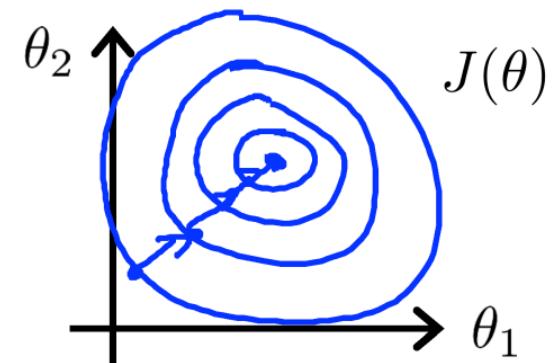
$$x_2 = \text{number of bedrooms (1-5)}$$



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

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

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



Feature Scaling

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

$$x_0 = 1$$

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

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

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

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

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

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

$$-\frac{1}{3} \text{ to } \frac{1}{3} \quad \checkmark$$

Mean normalization

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

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

Average size ≈ 100

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

1-5 bedrooms

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

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

avg value of x_1 in training set
 range (max - min)
 (or standard deviation)

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

Gradient descent in practice II: Learning rate

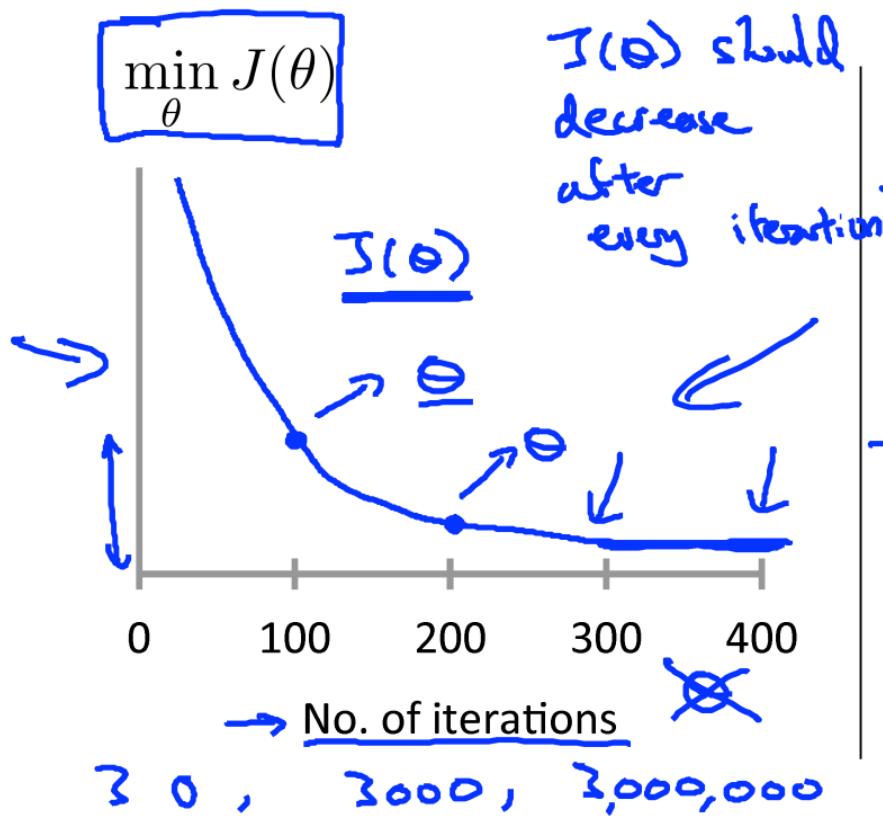


Gradient descent

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

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

Making sure gradient descent is working correctly.

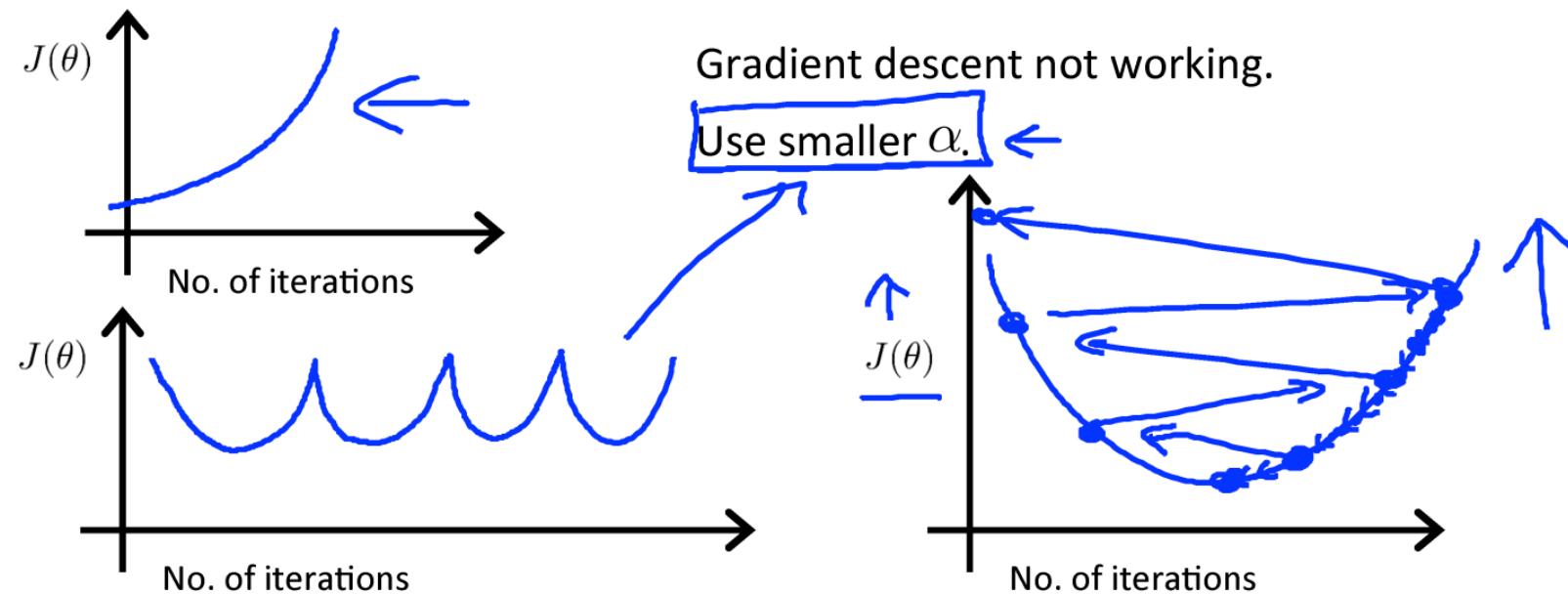


→ Example automatic convergence test:

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

$$\Sigma$$

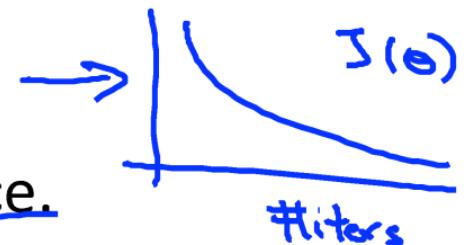
Making sure gradient descent is working correctly.



- 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

$$\dots, \underbrace{0.001}_{\uparrow}, \underbrace{0.003}_{\approx 2x}, \underbrace{0.01}_{\approx 2x}, \underbrace{0.03}_{3x}, \underbrace{0.1}_{\approx 3x}, \underbrace{0.3}_{10x}, \underbrace{1}_{\dots}$$

Features and polynomial regression



Housing prices prediction

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

$\underline{x_1}$



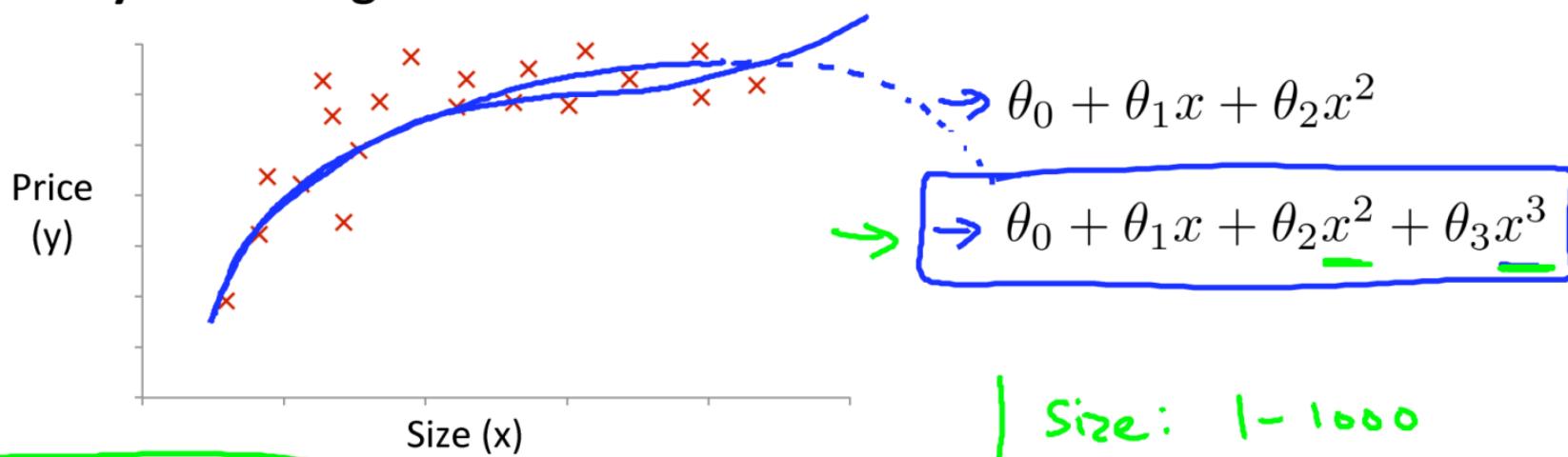
Area

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

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

\curvearrowleft land area

Polynomial regression

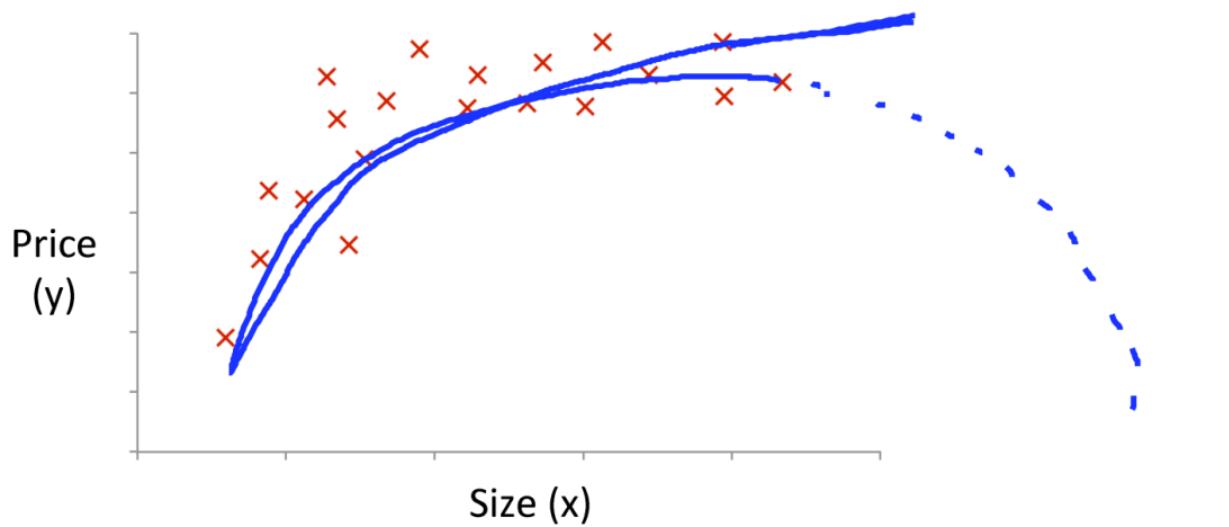


$$\begin{aligned}
 h_{\theta}(x) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \\
 &= \theta_0 + \theta_1(\text{size}) + \theta_2(\text{size})^2 + \theta_3(\text{size})^3
 \end{aligned}$$

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

Size: 1 - 100
 Size²: 1 - 100,000
 Size³: 1 - 10⁹

Choice of features



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

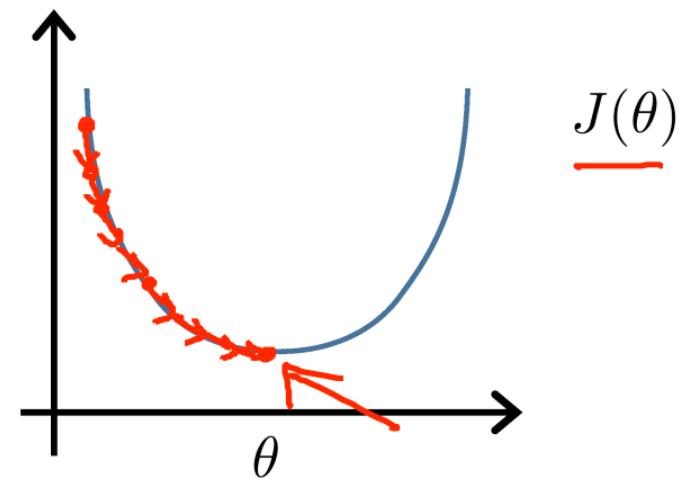
$$\rightarrow h_{\theta}(x) = \theta_0 + \theta_1(\text{size}) + \theta_2 \sqrt{(\text{size})}$$



Normal equation



Gradient Descent



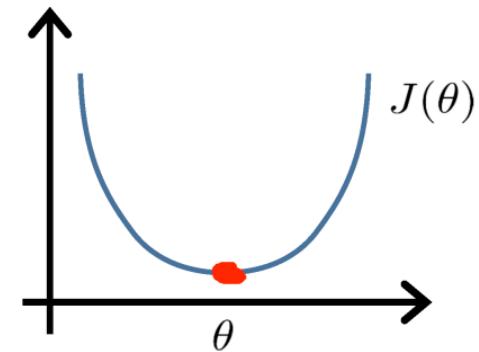
Normal equation: Method to solve for θ
analytically.

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

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

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

Solve for θ



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

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

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

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

Examples: $m = 4$.

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

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

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

$m \times (n+1)$

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

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

$$\underline{x}^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1} \quad | \quad \underline{X} = \begin{bmatrix} \underline{(x^{(1)})^\top} \\ \underline{(x^{(2)})^\top} \\ \vdots \\ \underline{(x^{(m)})^\top} \end{bmatrix}$$

(design matrix)

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

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

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

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

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

Set $A = X^T X$

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

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

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

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

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

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

m training examples, n features.

Gradient Descent

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

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



Normal Equation

- • No need to choose α .
- • Don't need to iterate.
- Need to compute

$$\rightarrow \boxed{(X^T X)^{-1}} \quad \underline{\text{nxn}} \quad \underline{O(n^3)}$$

- Slow if n is very large.

$$\underline{n = 100}$$

$$\underline{n = 1000}$$

$$\dots \underline{n = 10000}$$

Normal equation and non-invertibility (optional)



Normal equation

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

$X^T X$

- What if $X^T X$ is non-invertible? (singular/degenerate)

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



$\xrightarrow{\text{pinv}}$
 $\xrightarrow{\text{inv}}$

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



- Redundant features (linearly dependent).

E.g. $\underline{x_1} = \text{size in feet}^2$

$\underline{x_2} = \text{size in m}^2$

$$\underline{x_1} = (3.28)^2 \underline{x_2}$$

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

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

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

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

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

- Delete some features, or use regularization.

↓ later

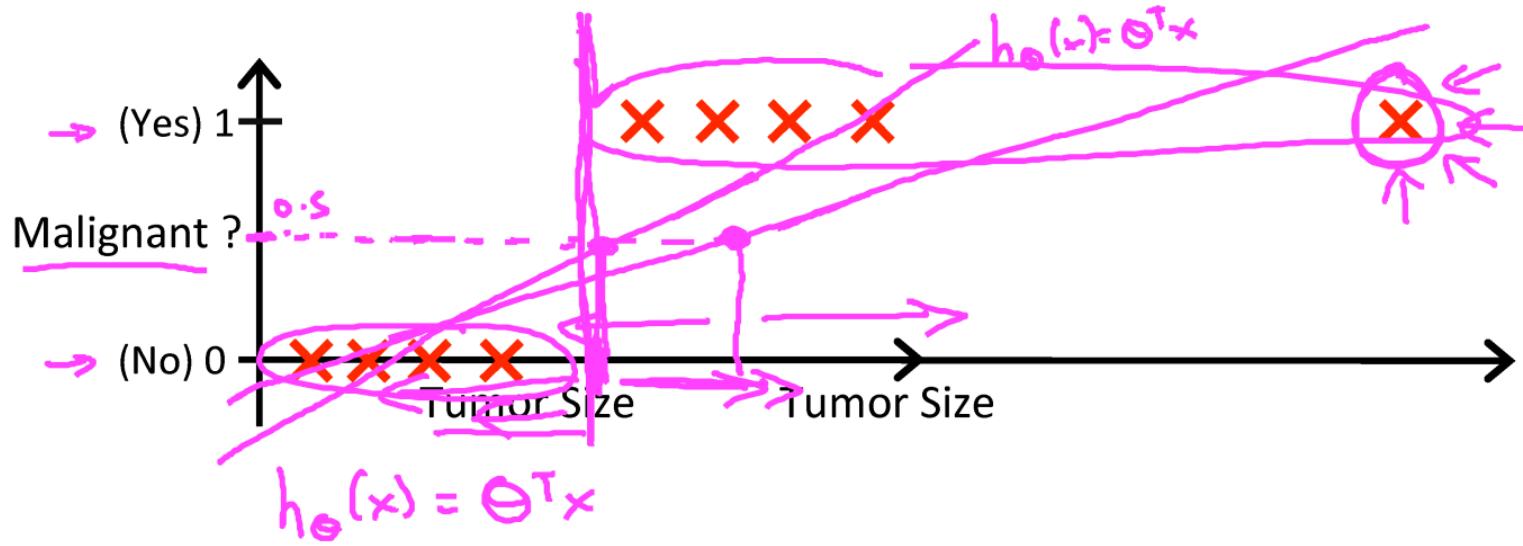
Logistic Regression

Classification



Classification

- Email: Spam / Not Spam?
 - Online Transactions: Fraudulent (Yes / No)?
 - Tumor: Malignant / Benign ?
- $y \in \{0, 1\}$
- 0: “Negative Class” (e.g., benign tumor)
 - 1: “Positive Class” (e.g., malignant tumor)
- $y \in \{0, 1, 2, 3\}$



→ Threshold classifier output $h_{\theta}(x)$ at 0.5:

→ If $h_{\theta}(x) \geq 0.5$, predict "y = 1"

If $h_{\theta}(x) < 0.5$, predict "y = 0"

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

$\underline{h_\theta(x)}$ can be $\underline{> 1}$ or $\underline{< 0}$

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

Classification

Logistic Regression

Hypothesis Representation

Logistic Regression Model

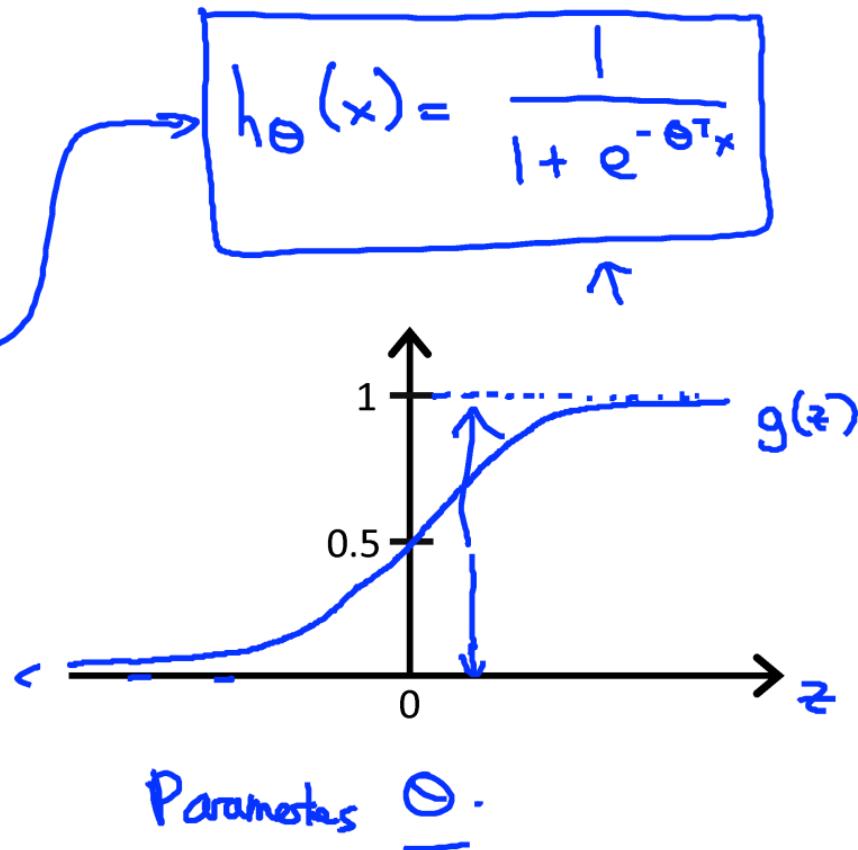
Want $0 \leq h_\theta(x) \leq 1$

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

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

$\theta^T x$

- Sigmoid function
- Logistic function



Interpretation of Hypothesis Output

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

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

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

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

Tell patient that 70% chance of tumor being malignant

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

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

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

$$\rightarrow \underline{P(y = 0|x; \theta) = 1 - P(y = 1|x; \theta)}$$

Logistic Regression

Decision boundary

Logistic regression

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

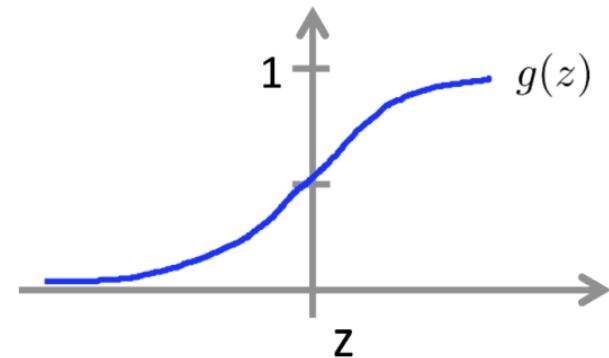
$$g(z) = \frac{1}{1+e^{-z}}$$

Suppose predict “ $y = 1$ ” if $h_{\theta}(x) \geq 0.5$

$$\theta^T x \geq 0$$

predict “ $y = 0$ ” if $h_{\theta}(x) < 0.5$

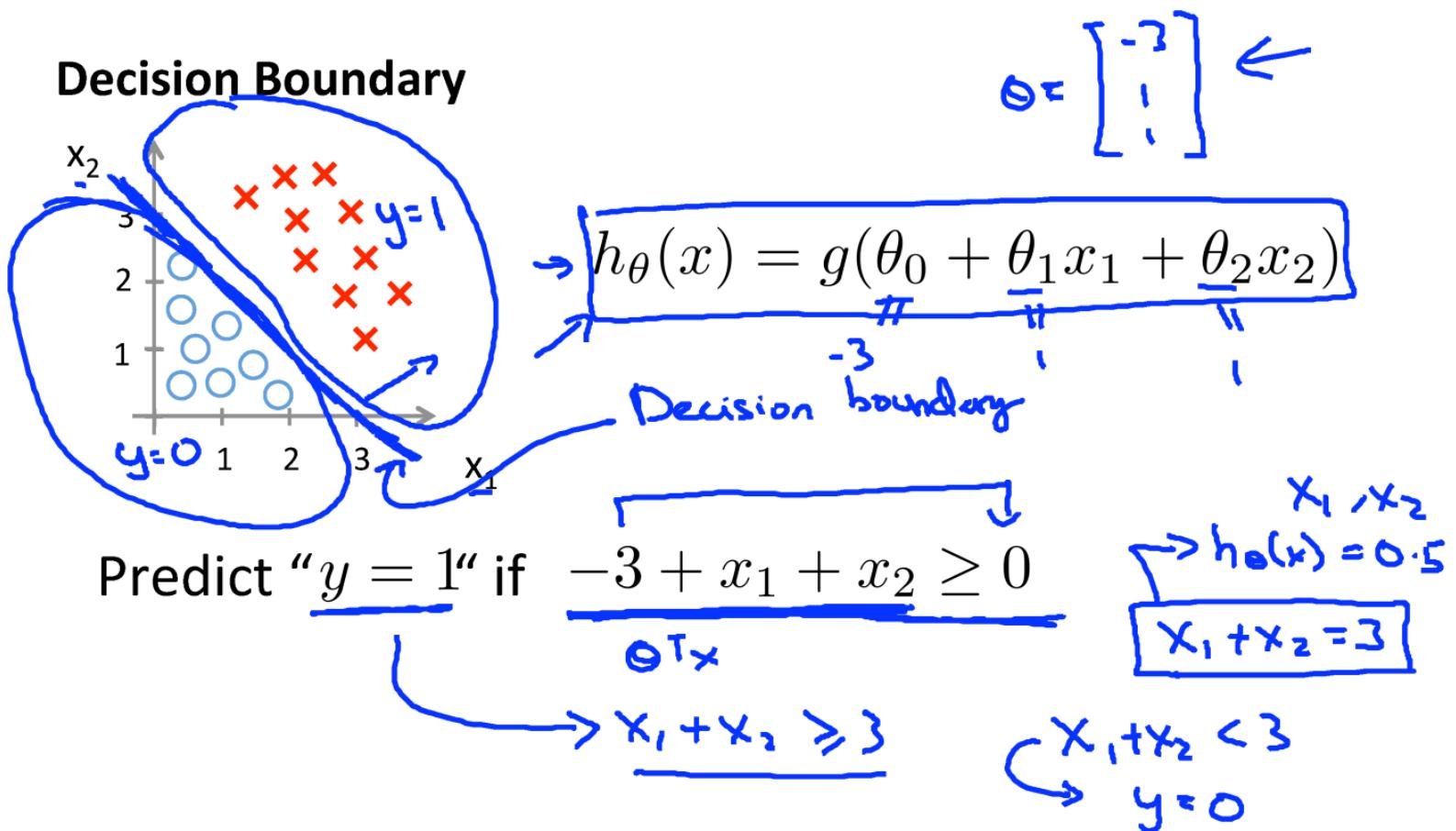
$$\theta^T x < 0$$



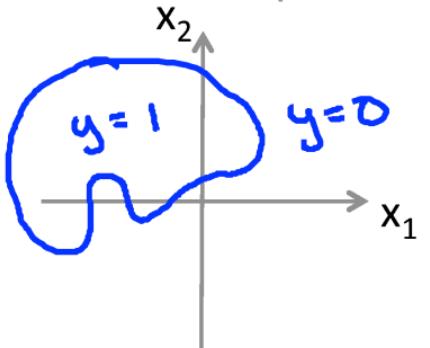
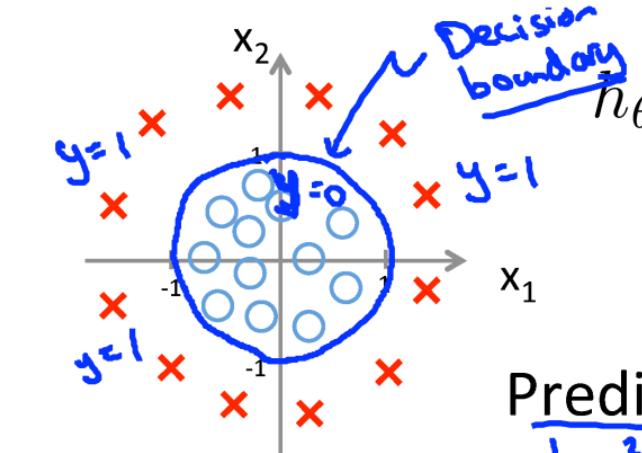
$$g(z) \geq 0.5 \\ \text{when } z \geq 0$$

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

$$g(z) < 0.5 \\ \text{when } z < 0$$



Non-linear decision boundaries



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

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

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

$$x_1^2 + x_2^2 = 1$$

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

Logistic Regression

Cost function

Training set:

m examples

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

$$x \in \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_n \end{bmatrix} \quad \mathbb{R}^{n+1}$$

$$x_0 = 1, y \in \{0, 1\}$$

How to choose parameters θ ?

Cost function

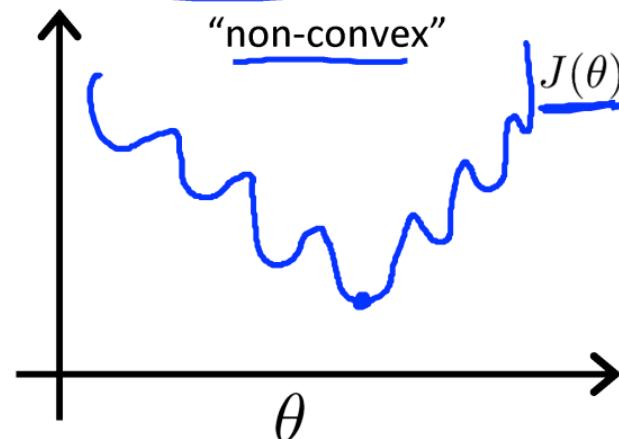
→ ~~Linear~~
logistic

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

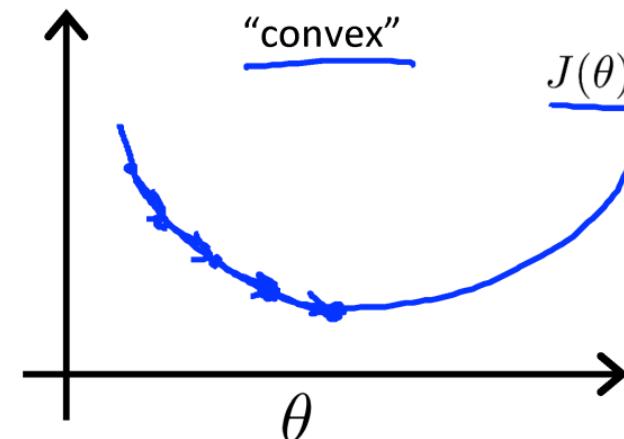
cost($h_\theta(x^{(i)})$, $y^{(i)}$)

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

$$\frac{1}{2} h_\theta(x^{(i)})^2$$



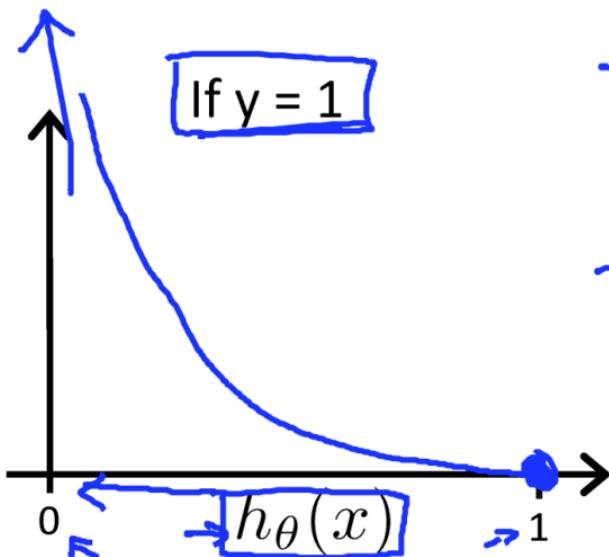
"non-convex"



"convex"

Logistic regression cost function

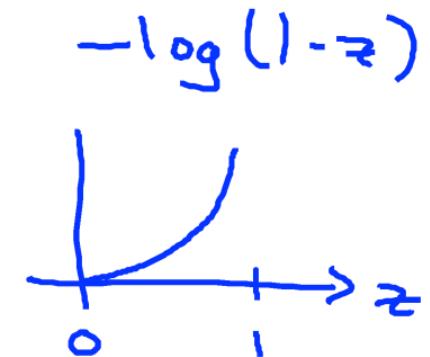
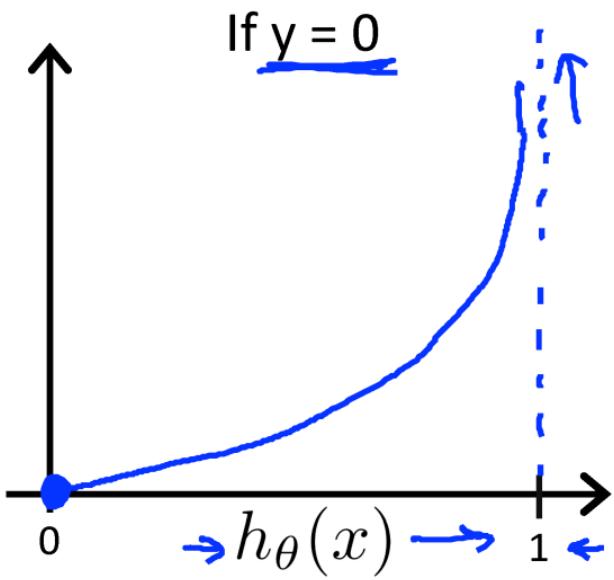
$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



- Cost = 0 if $y = 1, h_\theta(x) = 1$
But as $h_\theta(x) \rightarrow 0$
 $\underline{\text{Cost}} \rightarrow \infty$
- 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

Simplified cost function and gradient decent



Logistic regression cost function

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

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

Note: y = 0 or 1 always

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

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

$$\begin{aligned}
 J(\theta) &= \frac{1}{m} \sum_{i=1}^m \text{Cost}(h_\theta(x^{(i)}), y^{(i)}) \\
 &= -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)})) \right]
 \end{aligned}$$

To fit parameters θ :

$$\min_{\theta} J(\theta) \quad \text{Get } \Theta$$

To make a prediction given new x :

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

$p(y=1 | x; \theta)$

Gradient Descent

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

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

Repeat {

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

}

(simultaneously update all θ_j)

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

Gradient Descent

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

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

Repeat {

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

(simultaneously update all θ_j)

$$\Theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \quad \text{for } i=0 \text{ to } n$$

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

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

Algorithm looks identical to linear regression!

Logistic Regression

Advanced optimization

Optimization algorithm

Cost function $\underline{J(\theta)}$. Want $\min_{\theta} \underline{J(\theta)}$.

Given θ , we have code that can compute

$$\begin{aligned} &\rightarrow - J(\theta) \\ &\rightarrow - \frac{\partial}{\partial \theta_j} J(\theta) \quad (\text{for } j = 0, 1, \dots, n) \end{aligned}$$

Gradient descent:

Repeat {

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

}

Optimization algorithm

Given θ , we have code that can compute

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

(for $j = 0, 1, \dots, n$)

Optimization algorithms:

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

Advantages:

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

Disadvantages:

- More complex

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

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

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

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

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

```

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);

```

$$\rightarrow \text{options} = \text{optimset}('GradObj', 'on', 'MaxIter', '100');$$

$$\rightarrow \text{initialTheta} = \text{zeros}(2,1);$$

$$[\text{optTheta}, \text{functionVal}, \text{exitFlag}] ...$$

$$= \text{fminunc}(@\text{costFunction}, \text{initialTheta}, \text{options});$$

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

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

theta(1) ←
theta(2)
theta(n+1)

```
function [jVal gradient] = costFunction(theta)
    jVal = [code to compute  $J(\theta)$ ];
    gradient(1) = [code to compute  $\frac{\partial}{\partial \theta_0} J(\theta)$ ];
    gradient(2) = [code to compute  $\frac{\partial}{\partial \theta_1} J(\theta)$ ];
    :
    gradient(n+1) = [code to compute  $\frac{\partial}{\partial \theta_n} J(\theta)$ ];
```

Logistic Regression

Multi-class classification: One-vs-all

Multiclass classification

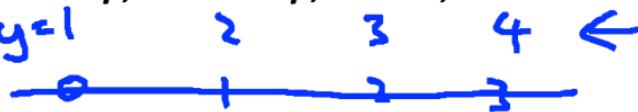
Email foldering/tagging: Work, Friends, Family, Hobby

$$y=1 \quad y=2 \quad y=3 \quad y=4$$

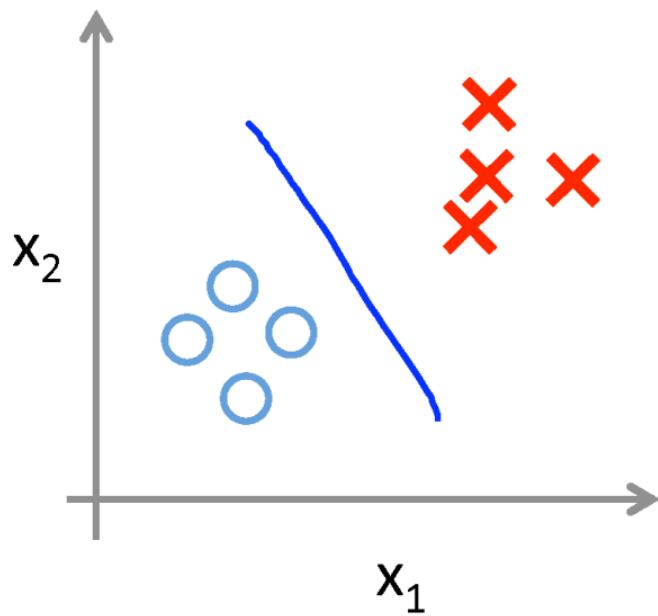
Medical diagrams: Not ill, Cold, Flu

$$y=1 \quad 2 \quad 3$$

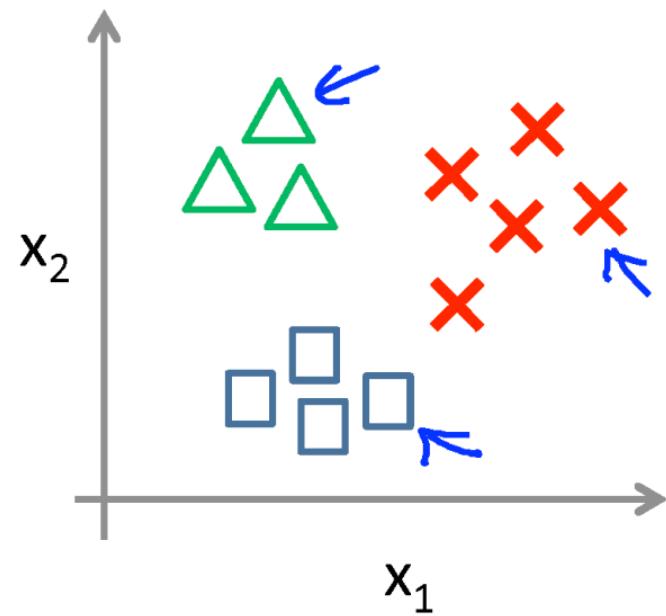
Weather: Sunny, Cloudy, Rain, Snow

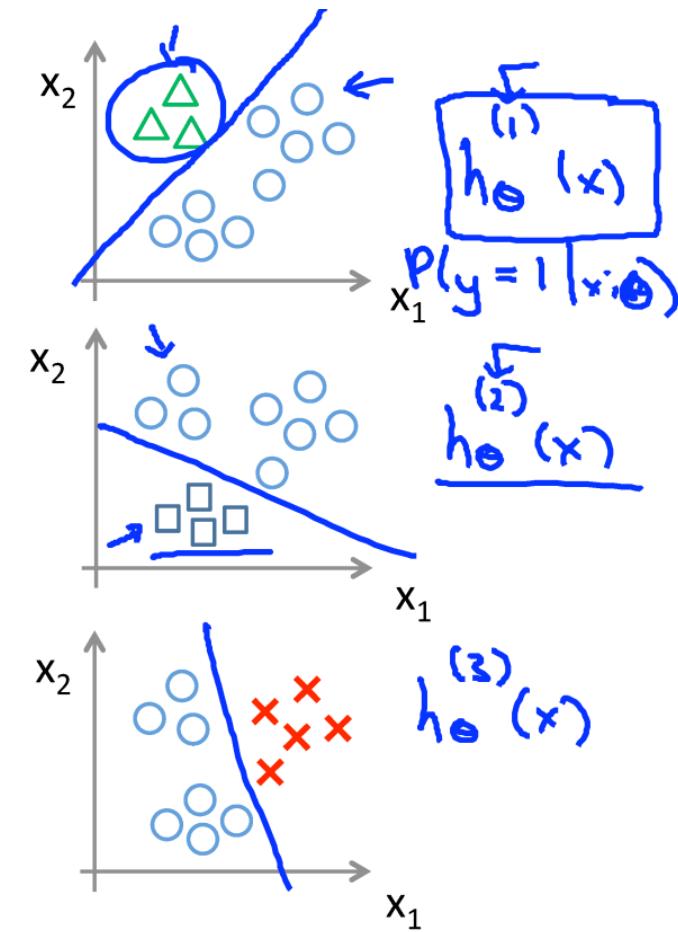
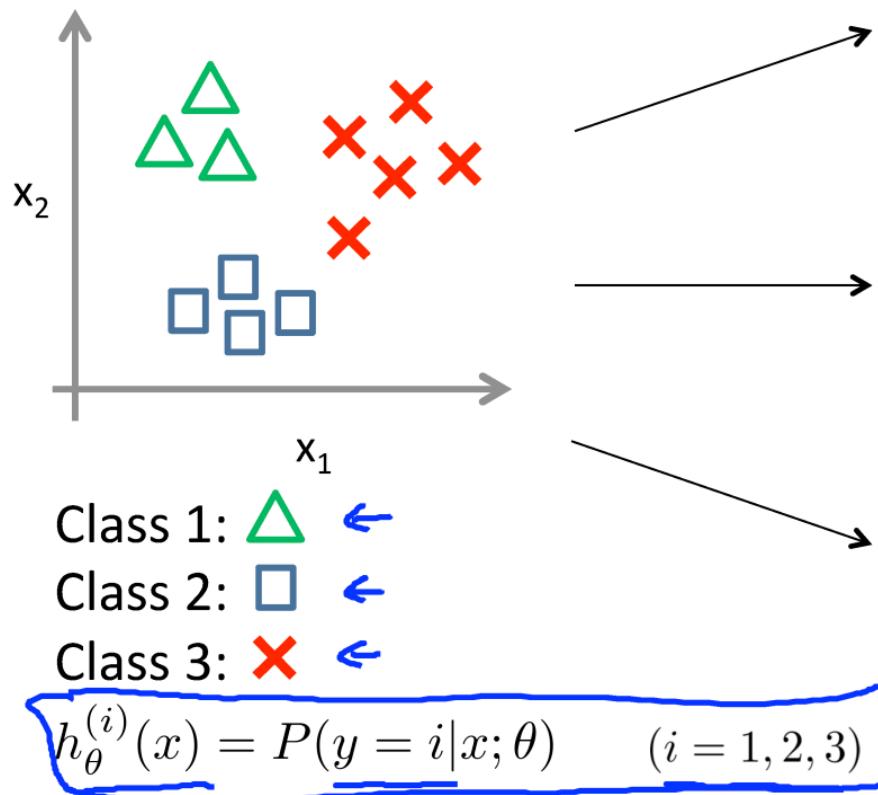
$$y=1 \quad 2 \quad 3 \quad 4 \leftarrow$$


Binary classification:



Multi-class classification:



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

One-vs-all

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

On a new input \underline{x} , to make a prediction, pick the class i that maximizes

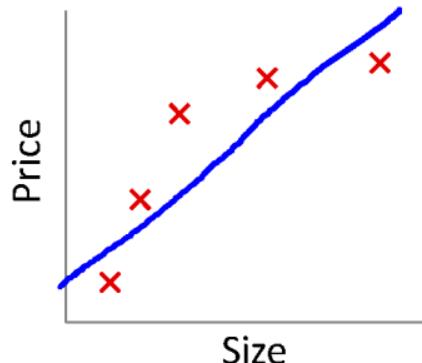
$$\max_i \underline{\underline{h_{\theta}^{(i)}(x)}}$$

Regularization

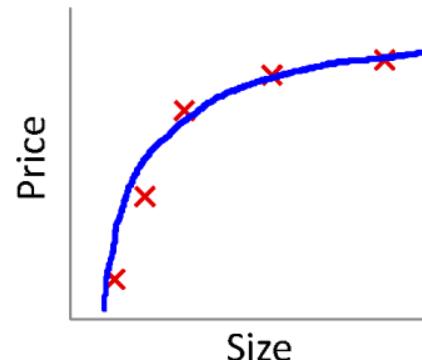
The problem of overfitting



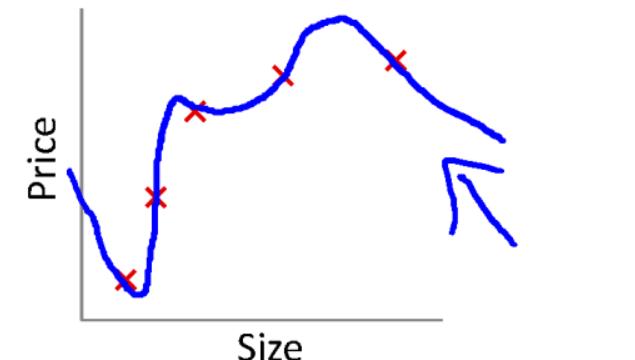
Example: Linear regression (housing prices)



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



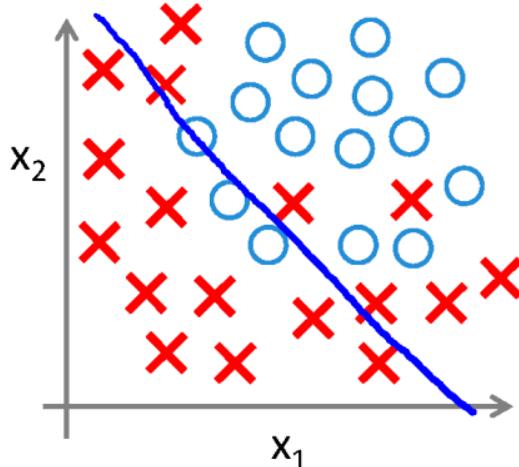
$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2$
 "Just right"



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

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

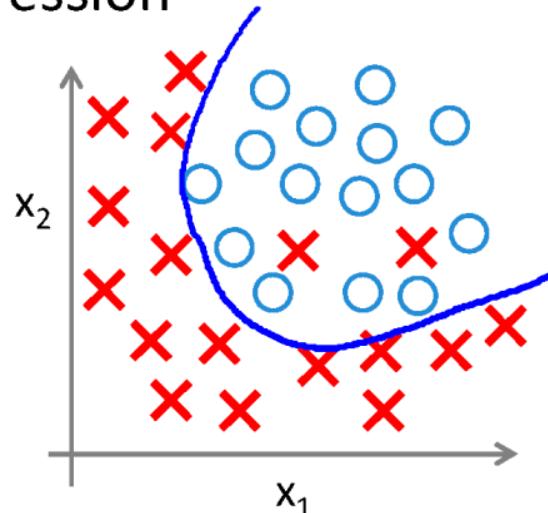
Example: Logistic regression



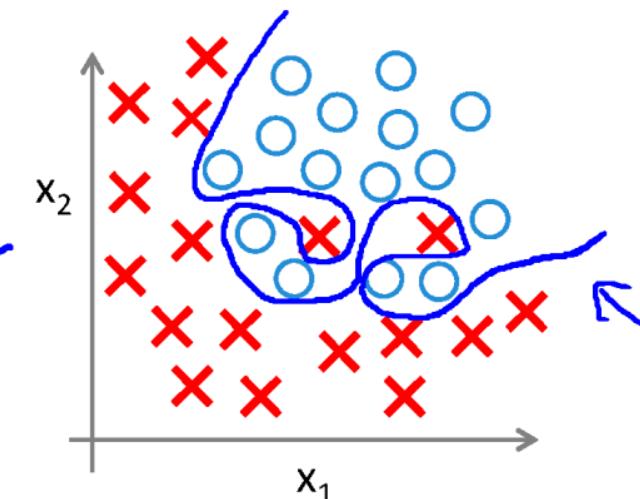
$$\rightarrow h_{\theta}(x) = g(\underline{\theta_0 + \theta_1 x_1 + \theta_2 x_2})$$

(g = sigmoid function)

"Under-fit"



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

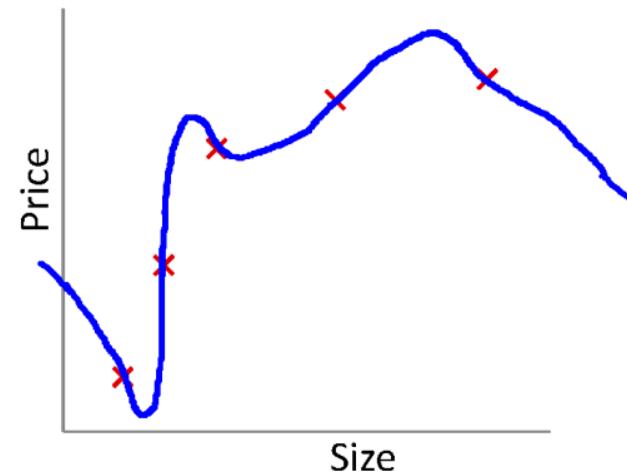


$$g(\underline{\theta_0 + \theta_1 x_1 + \theta_2 x_1^2} \\ + \underline{\theta_3 x_1^2 x_2} + \underline{\theta_4 x_1^2 x_2^2} \\ + \underline{\theta_5 x_1^2 x_2^3} + \underline{\theta_6 x_1^3 x_2} + \dots)$$

"Over-fit"

Addressing overfitting:

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



Addressing overfitting:

Options:

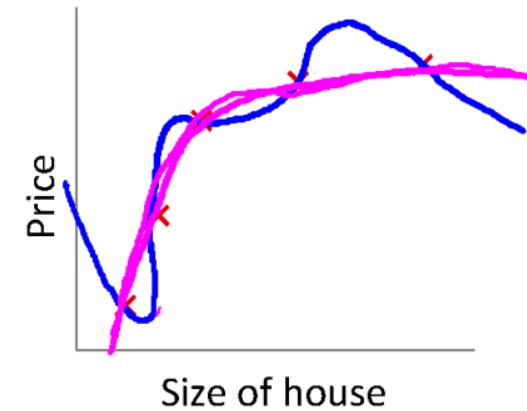
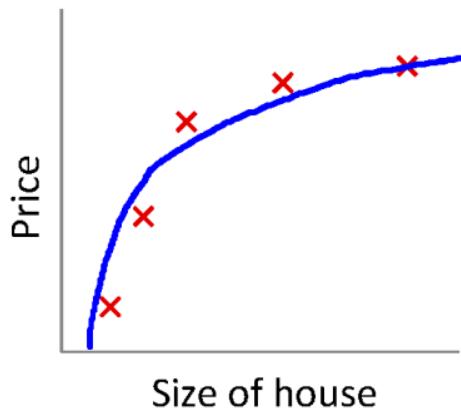
1. Reduce number of features.
 - — Manually select which features to keep.
 - — Model selection algorithm (later in course).
2. Regularization.
 - — Keep all the features, but reduce magnitude/values of parameters θ_j .
 - Works well when we have a lot of features, each of which contributes a bit to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

Regularization

Cost function



Intuition



Suppose we penalize and make θ_3, θ_4 really small.

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

$\underline{\theta_3 \approx 0}$ $\underline{\theta_4 \approx 0}$

Regularization.

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

- “Simpler” hypothesis 
- Less prone to overfitting 

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

Housing:

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

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

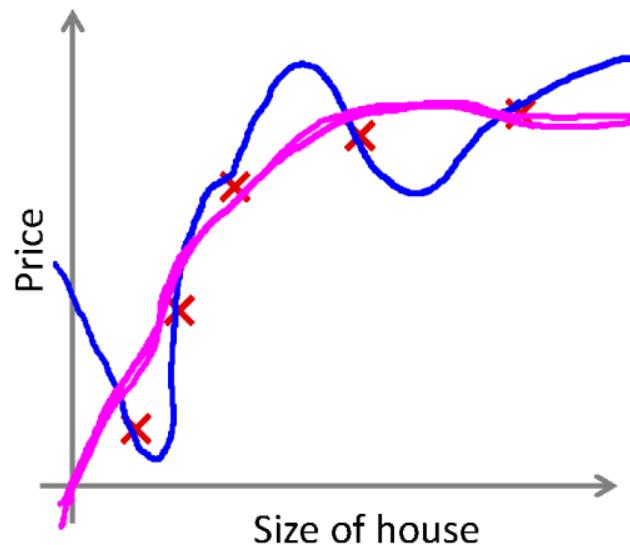
~~$\theta_1, \theta_2, \theta_3, \dots, \theta_{100}$~~

Regularization.

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

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

regularization parameter



In regularized linear regression, we choose θ to minimize

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

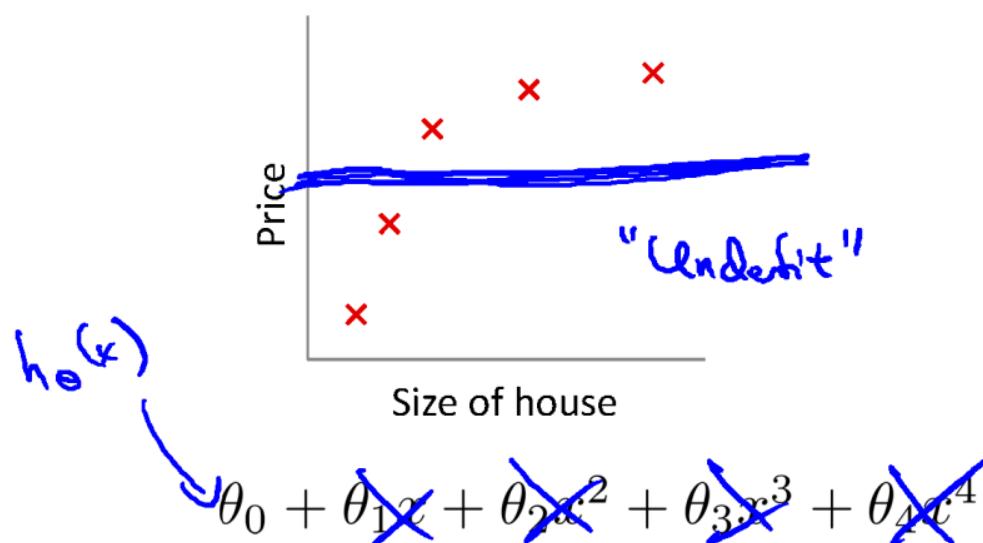
What if λ is set to an extremely large value (perhaps for too large for our problem, say $\lambda = 10^{10}$)?

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

In regularized linear regression, we choose 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 $\lambda = 10^{10}$)?



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

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

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

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

Regularization

Regularized linear regression



Regularized linear regression

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

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

↑

Gradient descent

Repeat {

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

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

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

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

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

}

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

$$1 - \alpha \frac{\lambda}{m} < 1 \quad \underline{0.99} \quad \theta_j \times 0.99$$

$$\theta_j^*$$

Normal equation

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

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

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

$$\frac{\partial}{\partial \theta_j} J(\theta) \stackrel{\text{set}}{=} 0 \quad \text{← } n$$

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

E.g. $n=2$

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

Non-invertibility (optional/advanced).

Suppose $m \leq n$, \leftarrow
 (#examples) (#features)

$$\theta = \frac{(X^T X)^{-1} X^T y}{\text{non-invertible / singular}}$$

pinv $\frac{\text{inv}}{\kappa}$

If $\lambda > 0$,

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

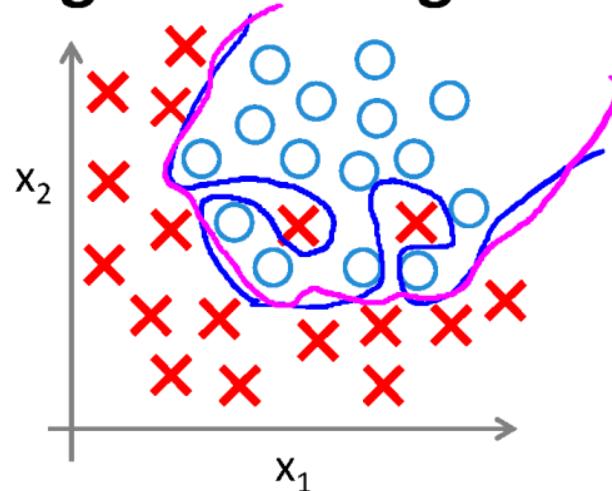
invertible.

Regularization

Regularized logistic regression



Regularized logistic regression.



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

Cost function:

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

$\boxed{\theta_1, \theta_2, \dots, \theta_n}$

Gradient descent

Repeat {

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

$$\rightarrow \theta_j := \theta_j - \alpha \left[\underbrace{\frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}}_{(j = \cancel{0}, 1, 2, 3, \dots, n)} - \frac{\lambda}{m} \theta_j \right] \leftarrow$$

$\theta_1, \dots, \theta_n$

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

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

Advanced optimization

\rightarrow function [jVal, gradient] = costFunction(theta)

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

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

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

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

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

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

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

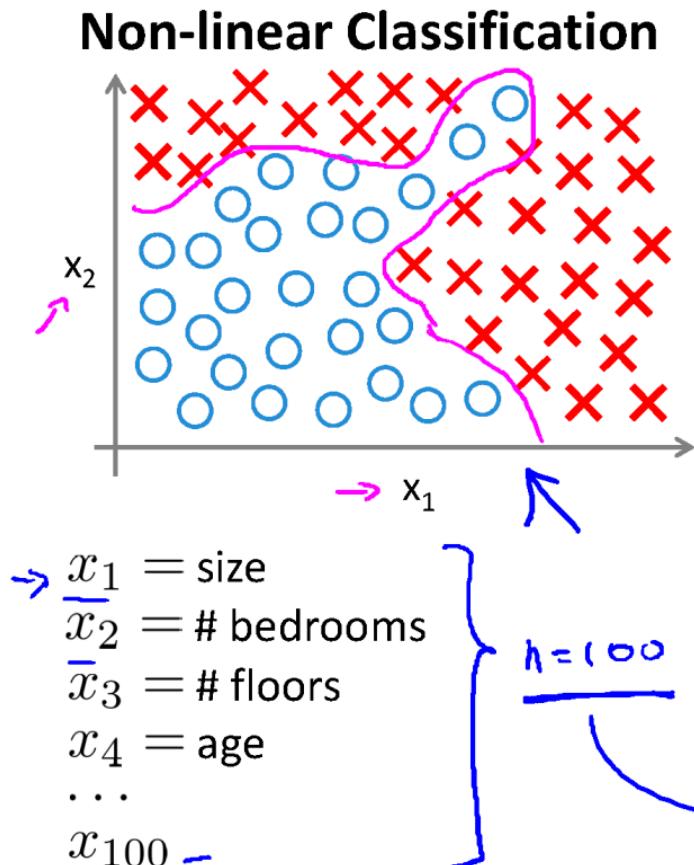
$$\vdots \quad \left(\frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)} \right) - \frac{\lambda}{m} \theta_2$$

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

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

Neural Networks: Representation

Non-linear hypotheses



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

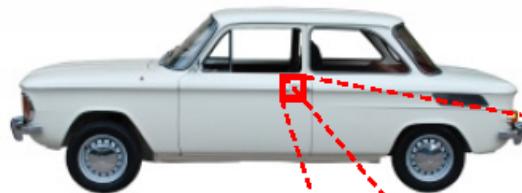
$\rightarrow \underline{x_1^2}, \underline{x_1 x_2}, \underline{x_1 x_3}, \underline{x_1 x_4} \dots \underline{x_1 x_{100}}$
 $\underline{x_2^2}, \underline{x_2 x_3} \dots$
 $\approx 5000 \text{ feature}$ $\mathcal{O}(n^2)$

$\rightarrow \underline{x_1^2}, \underline{x_2^2}, \underline{x_3^2}, \dots, \underline{x_{100}^2}$
 $\underline{x_1 x_2 x_3}, \underline{x_1^2 x_2}, \underline{x_{10} x_{11} x_{12}}, \dots$ $\frac{n^2}{2}$
~~10~~

$\mathcal{O}(n^3)$ 170,000

What is this?

You see this:



But the camera sees this:

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



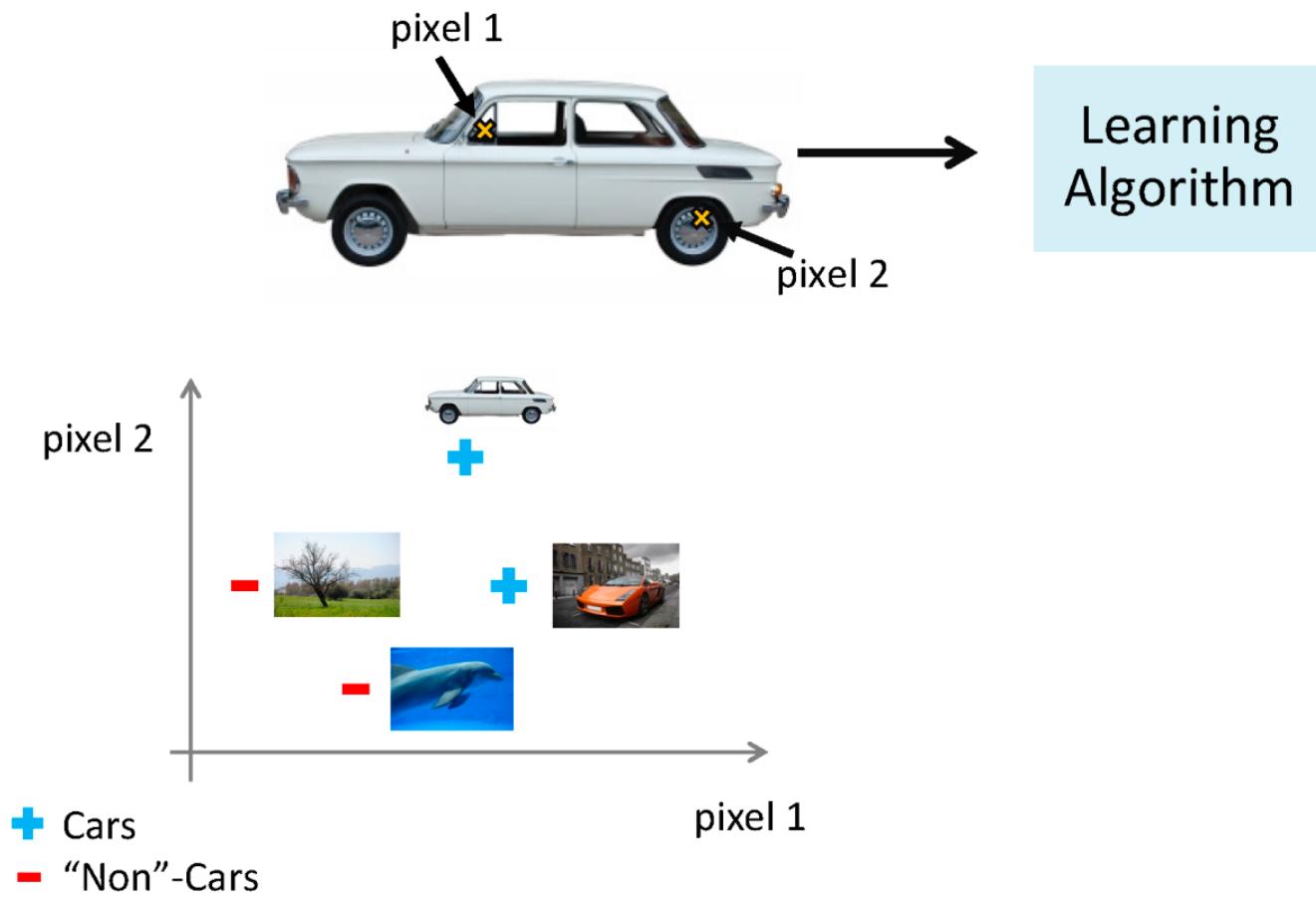
Computer Vision: Car detection

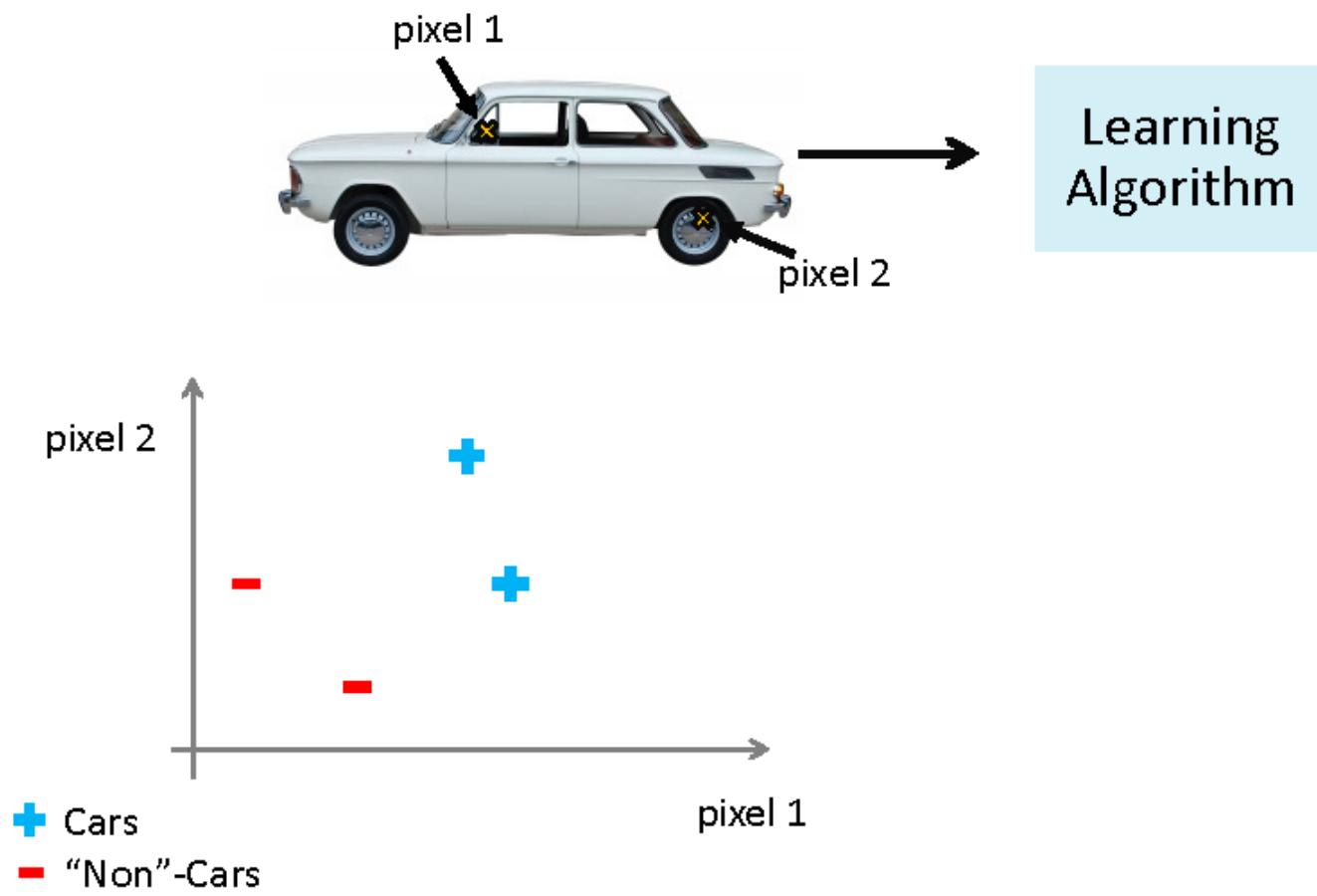


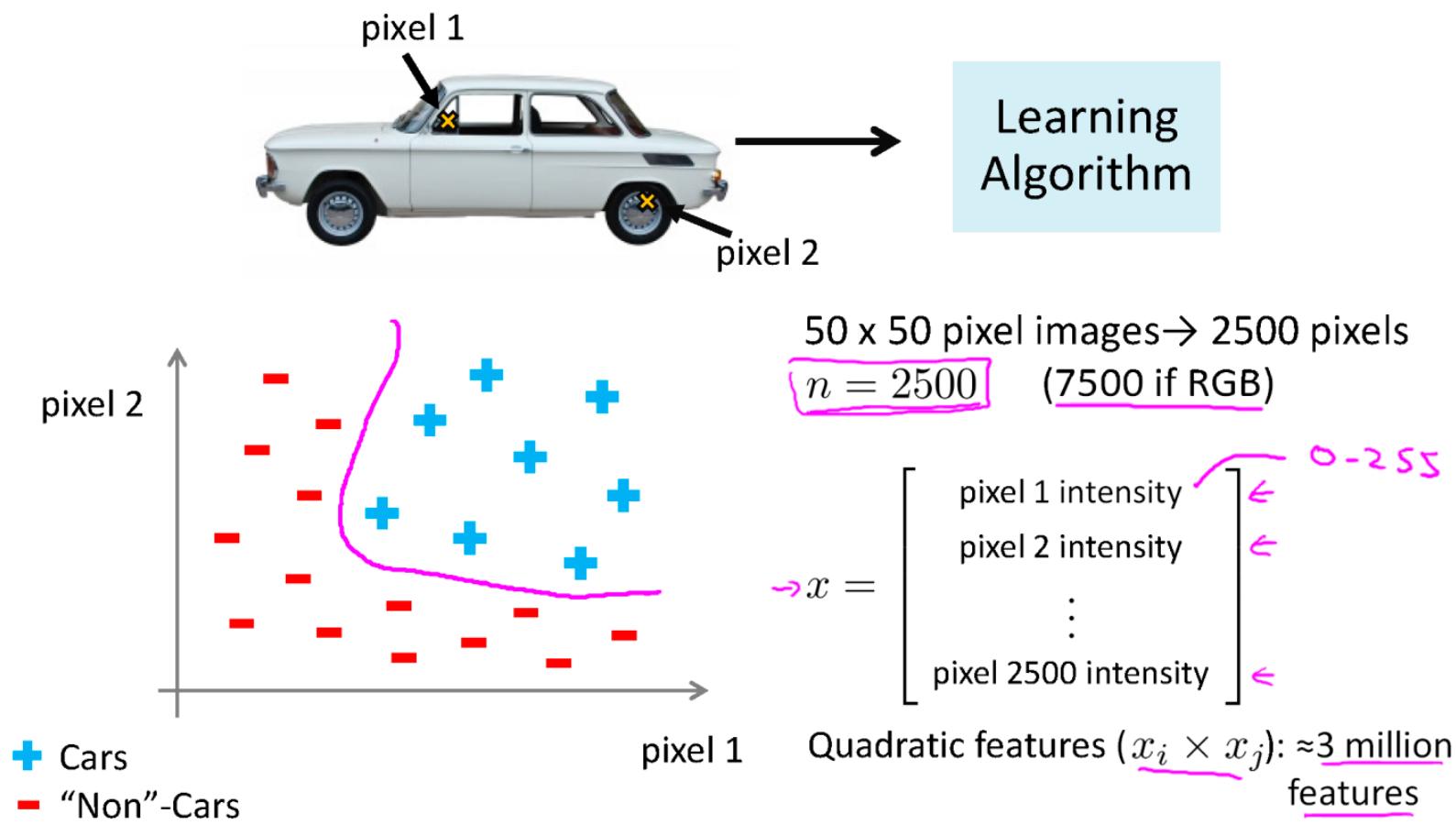
Testing:



What is this?







Neural Networks: Representation

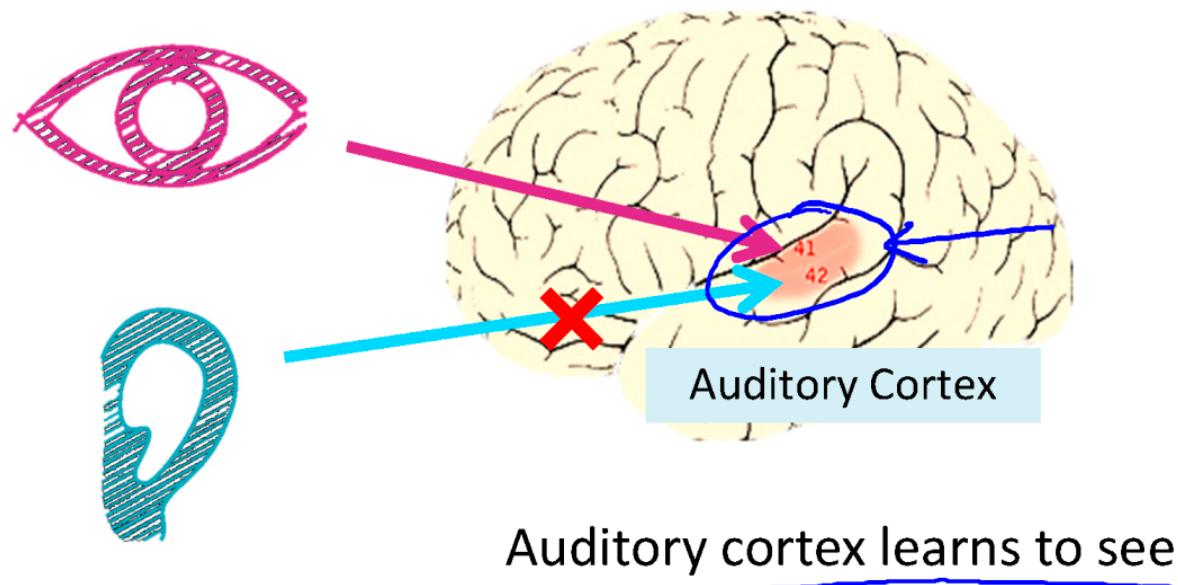
Neurons and the brain



Neural Networks

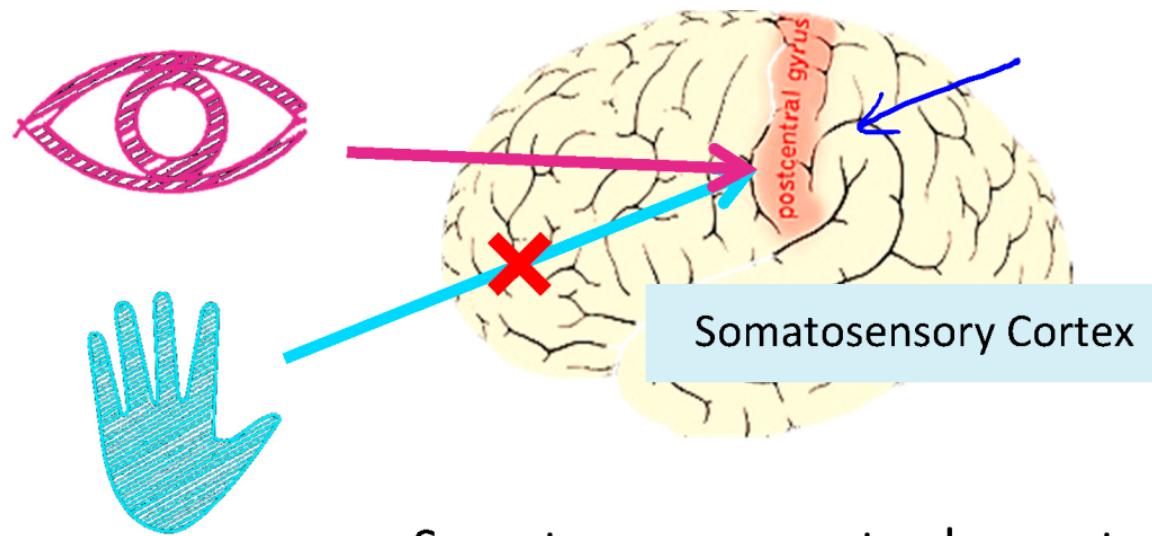
- Origins: Algorithms that try to mimic the brain.
- Was very widely used in 80s and early 90s; popularity diminished in late 90s.
- Recent resurgence: State-of-the-art technique for many applications

The “one learning algorithm” hypothesis



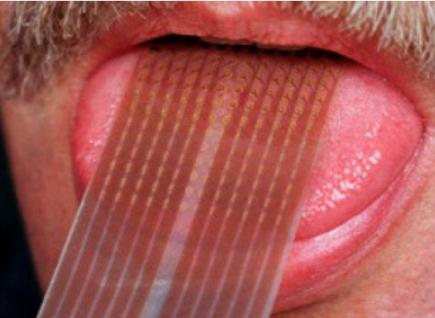
[Roe et al., 1992]

The “one learning algorithm” hypothesis



[Metin & Frost, 1989]

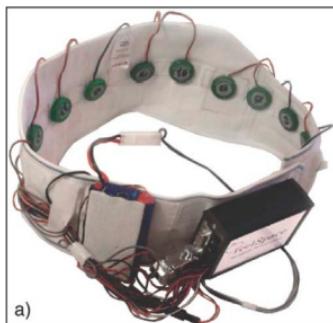
Sensor representations in the brain



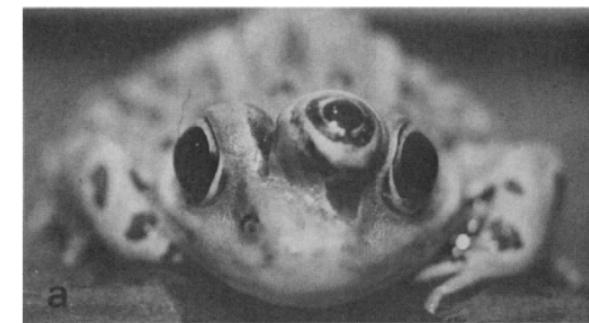
Seeing with your tongue



Human echolocation (sonar)



Haptic belt: Direction sense



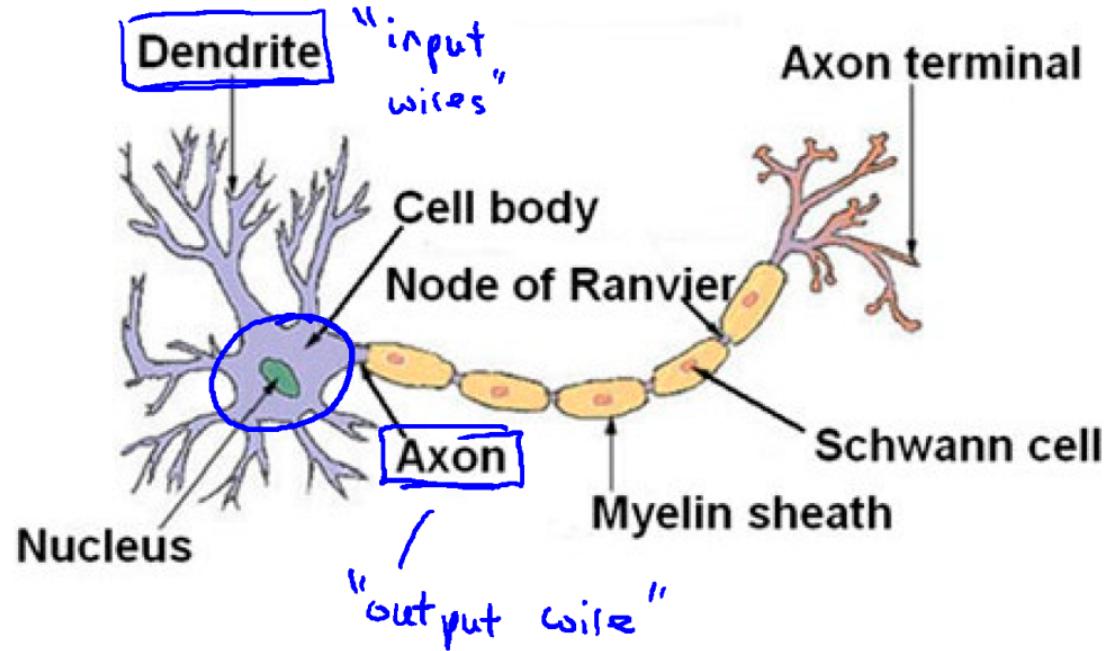
Implanting a 3rd eye

[BrainPort; Welsh & Blasch, 1997; Nagel et al., 2005; Constantine-Paton & Law, 2009]

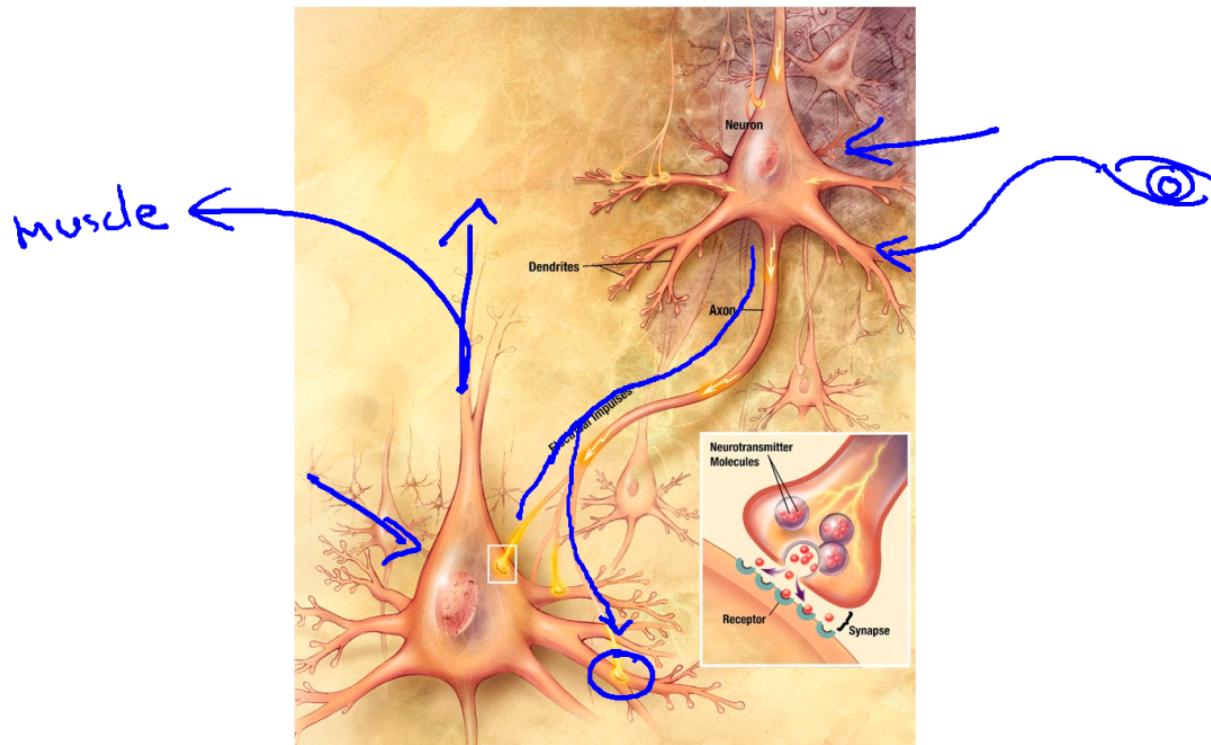
Neural Networks: Representation

Model representation I

Neuron in the brain

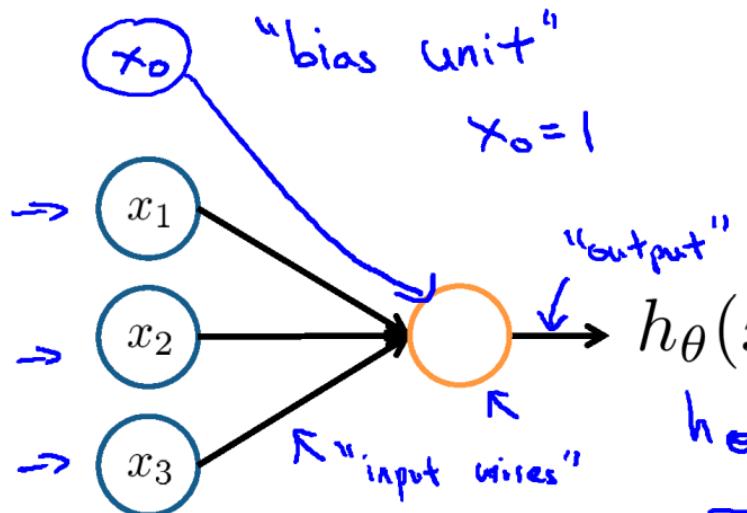


Neurons in the brain



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

Neuron model: Logistic unit



$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

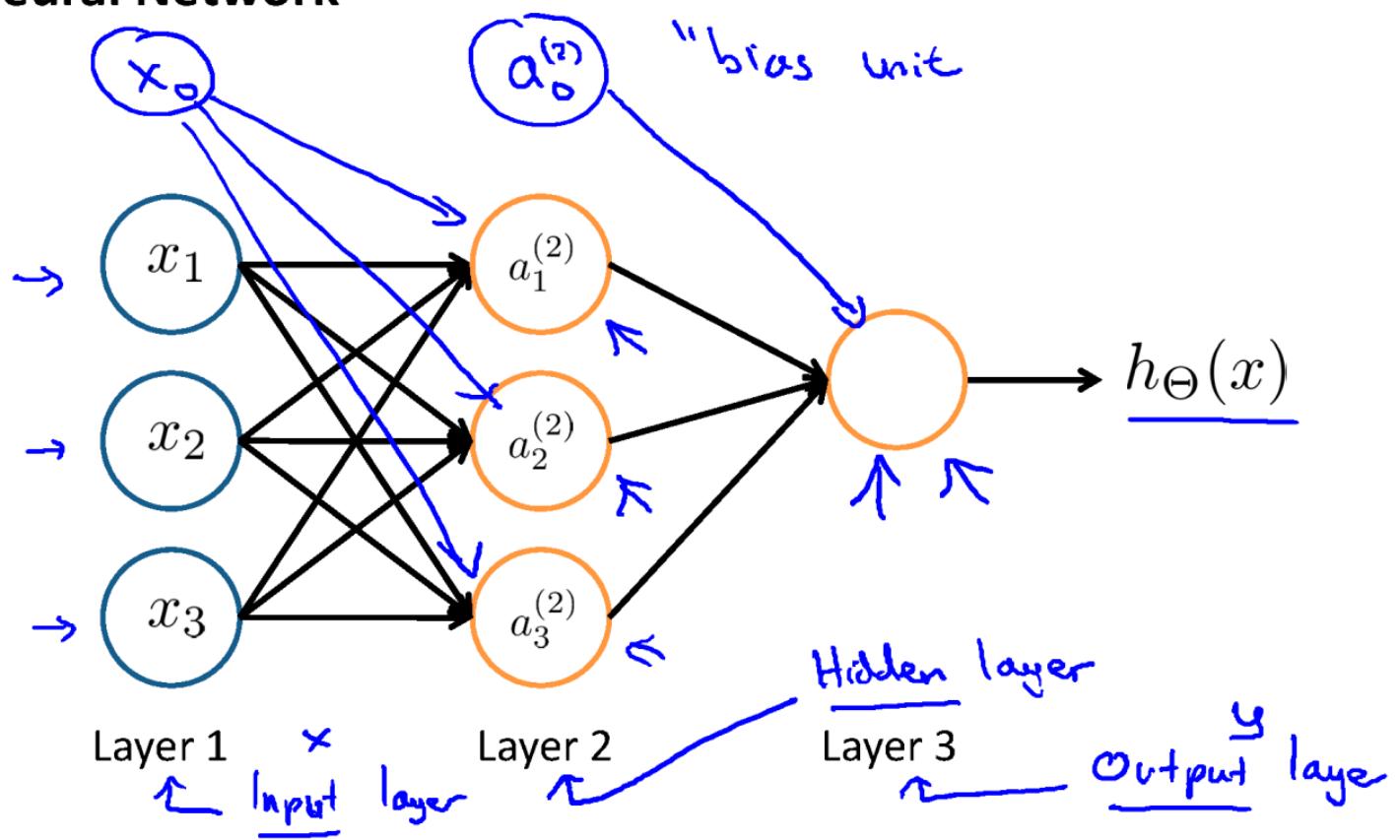
↑
"weights" ←
(parameters ↓

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

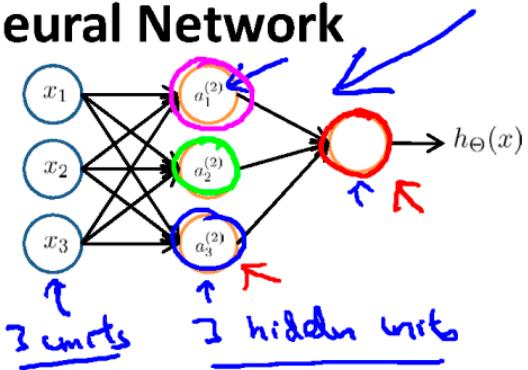
Sigmoid (logistic) activation function.

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

Neural Network



Neural Network



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

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

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

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

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

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

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

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

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

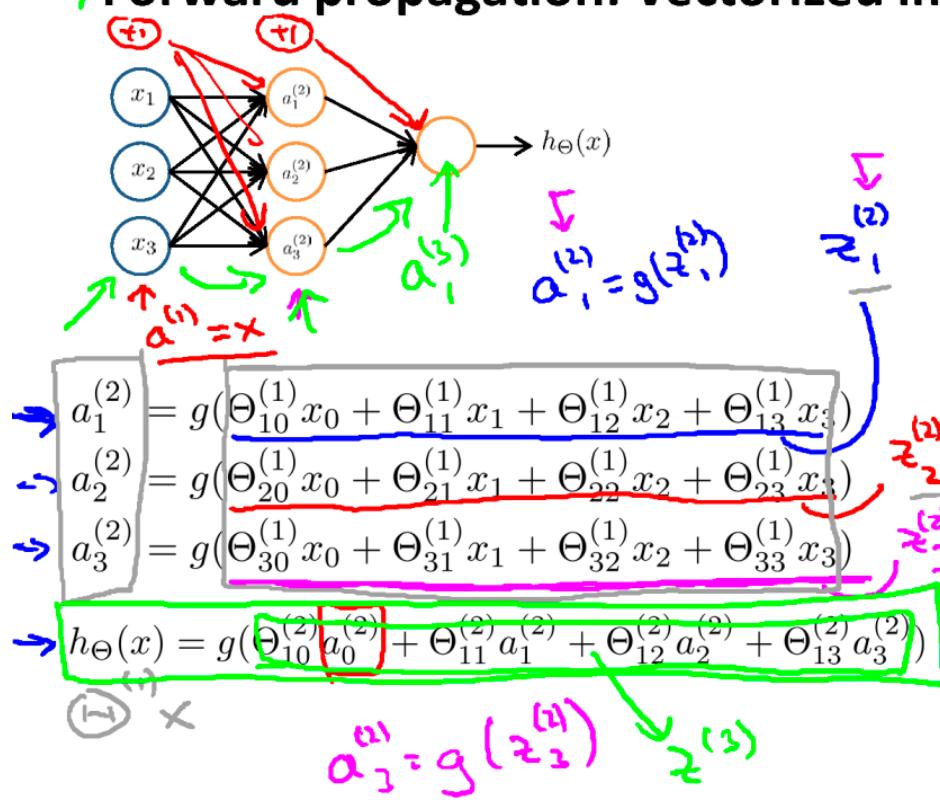
$$s_{j+1} \times (s_j + 1)$$

Neural Networks: Representation

Model representation II



Forward propagation: Vectorized implementation



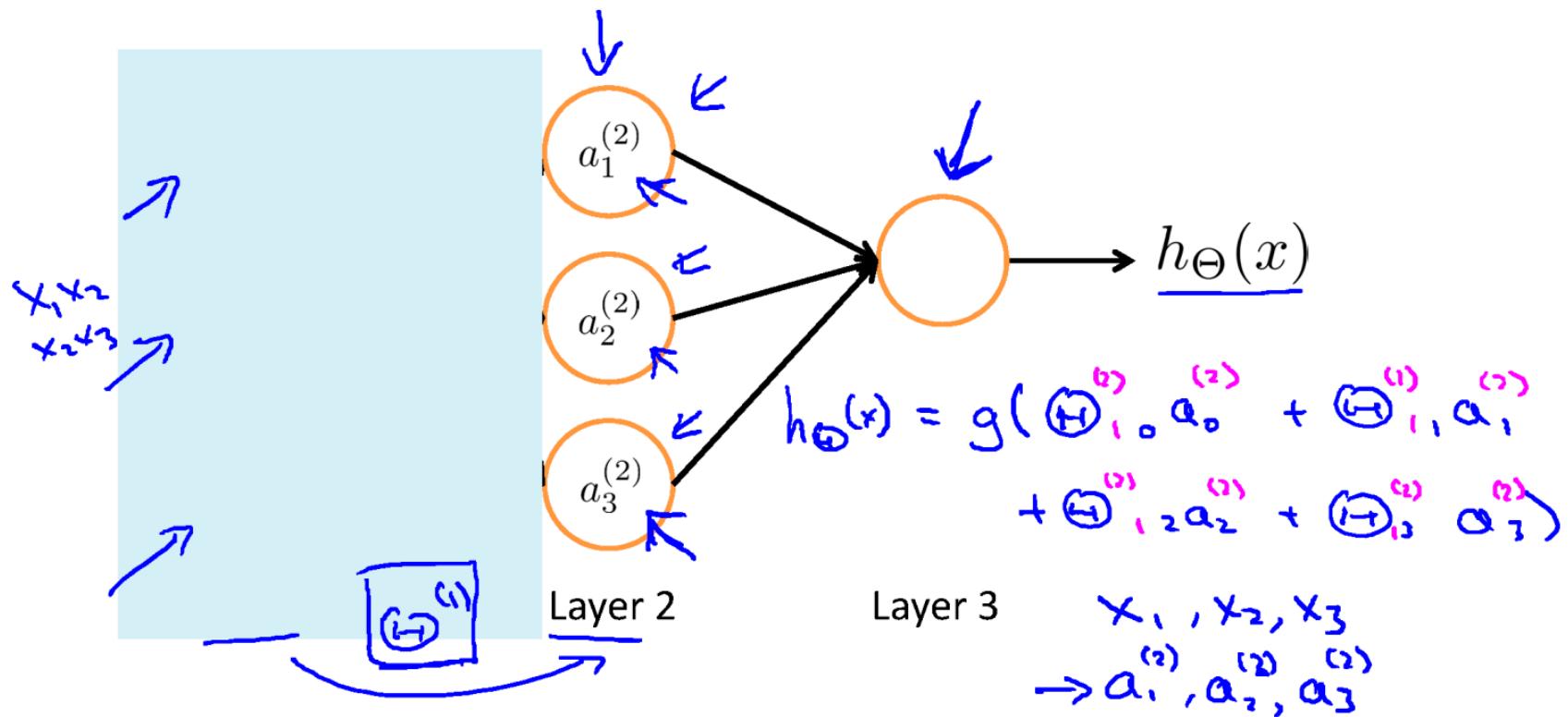
$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$$

$$\begin{aligned} z^{(2)} &= \Theta^{(1)} \times \alpha^{(1)} \\ a^{(2)} &= g(z^{(2)}) \\ &\in \mathbb{R}^3 \end{aligned}$$

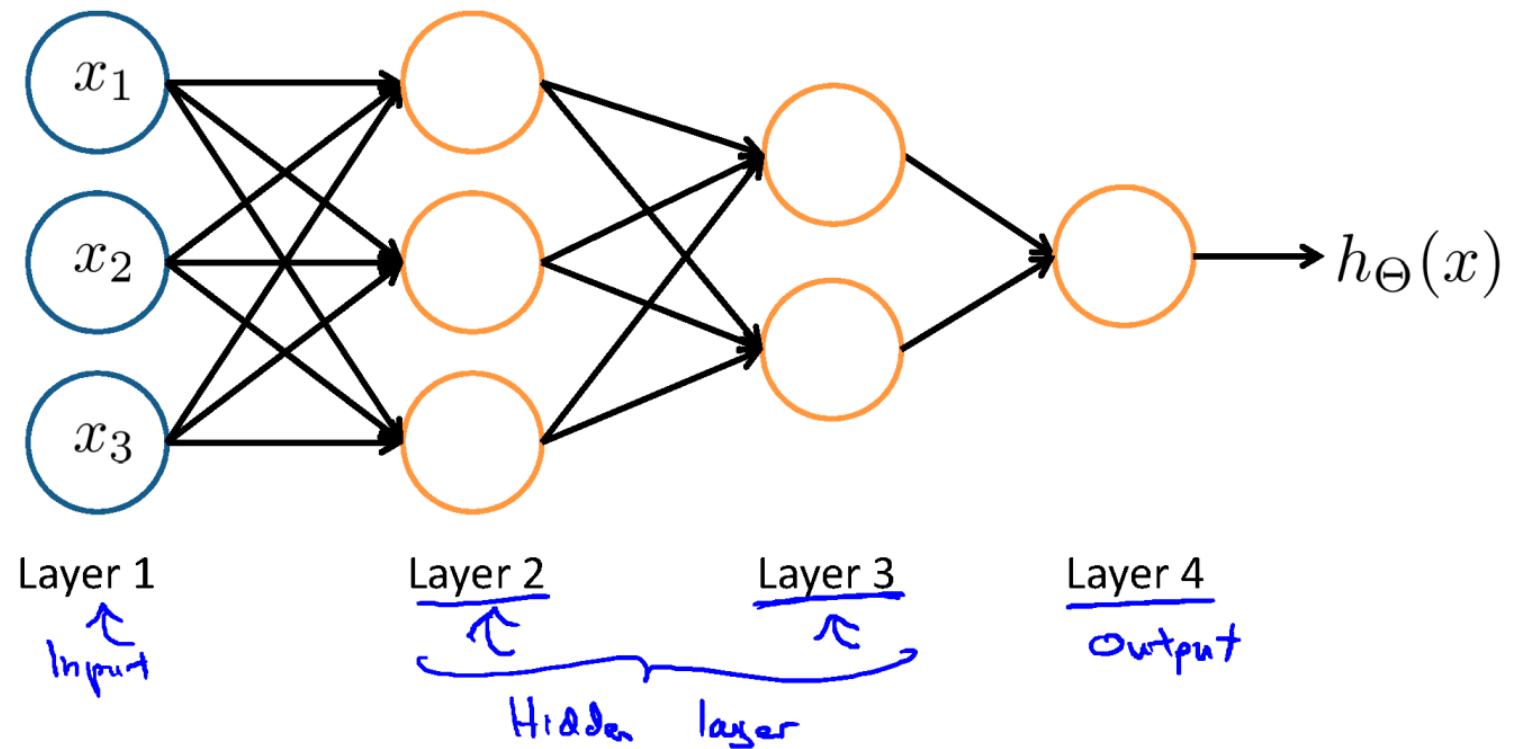
Add $a_0^{(2)} = 1.$ $\rightarrow a^{(2)}$ $\in \mathbb{R}^4$

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

Neural Network learning its own features



Other network architectures

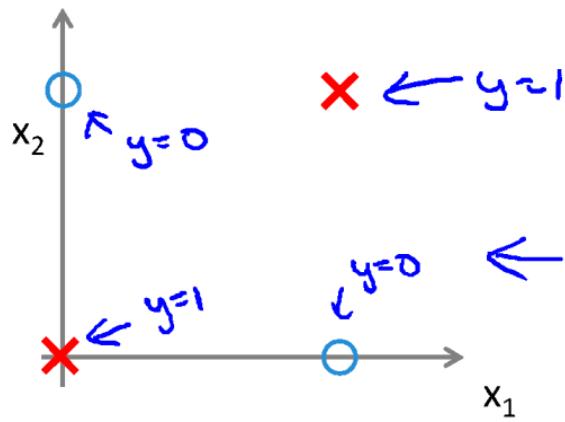


Neural Networks: Representation

Examples and intuitions I

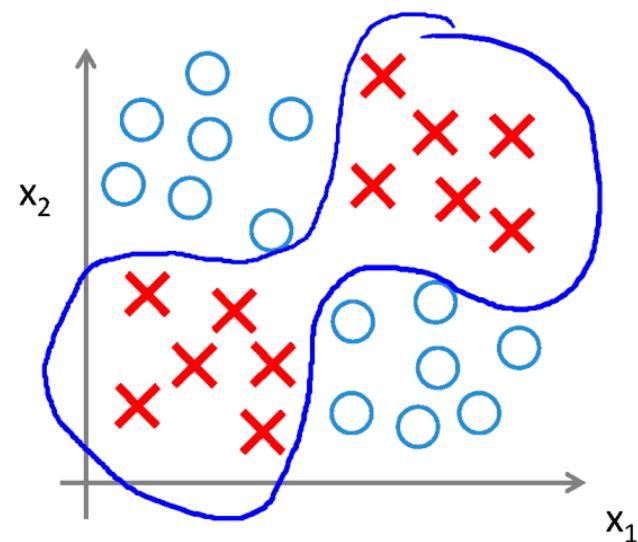
Non-linear classification example: XOR/XNOR

→ x_1, x_2 are binary (0 or 1).



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

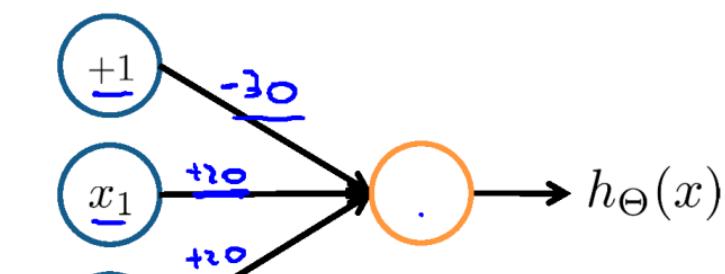
→ $\underline{\underline{\text{NOT} (x_1 \text{ XOR } x_2)}}$



Simple example: AND

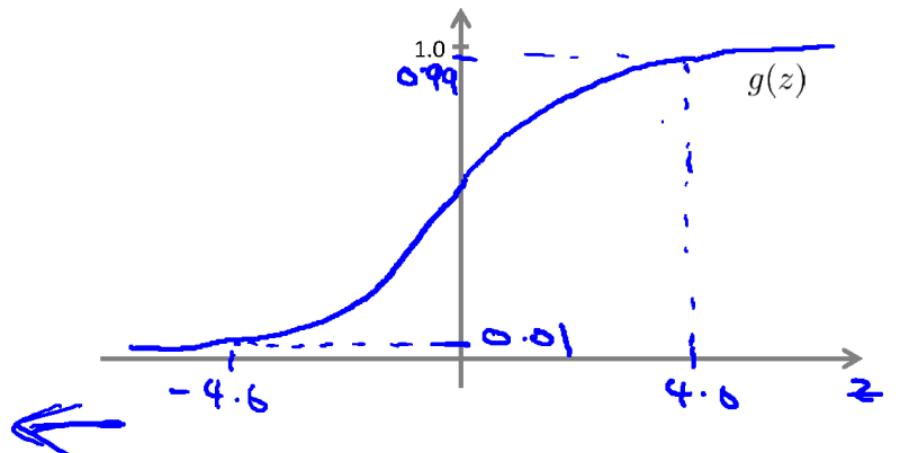
$$\rightarrow x_1, x_2 \in \{0, 1\}$$

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



$$\rightarrow h_{\Theta}(x) = g\left(\frac{-30}{\pi} + \frac{10x_1}{\pi} + \frac{20x_2}{\pi}\right)$$

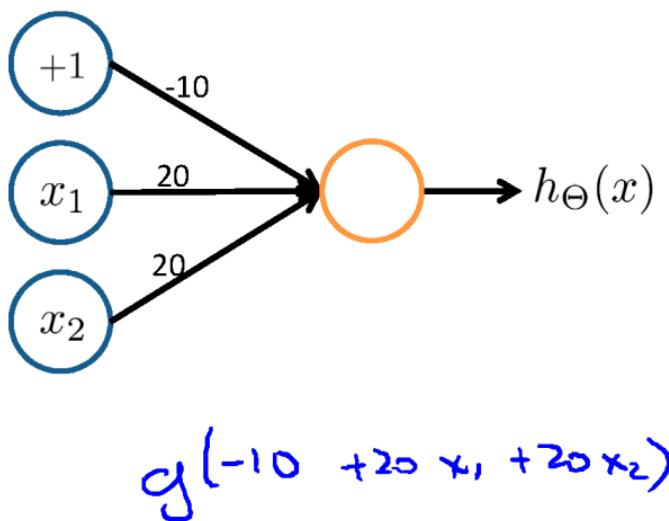
$\Theta_{10}^{(1)}$ $\Theta_{11}^{(1)}$ $\Theta_{12}^{(1)}$



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

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

Example: OR function



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

Neural Networks: Representation

Examples and intuitions II

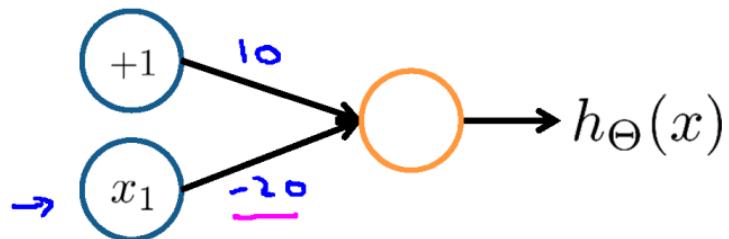
$\rightarrow x_1 \text{ AND } x_2$

$\rightarrow x_1 \text{ OR } x_2$

$\{0, 1\}$.

Negation:

NOT x_1

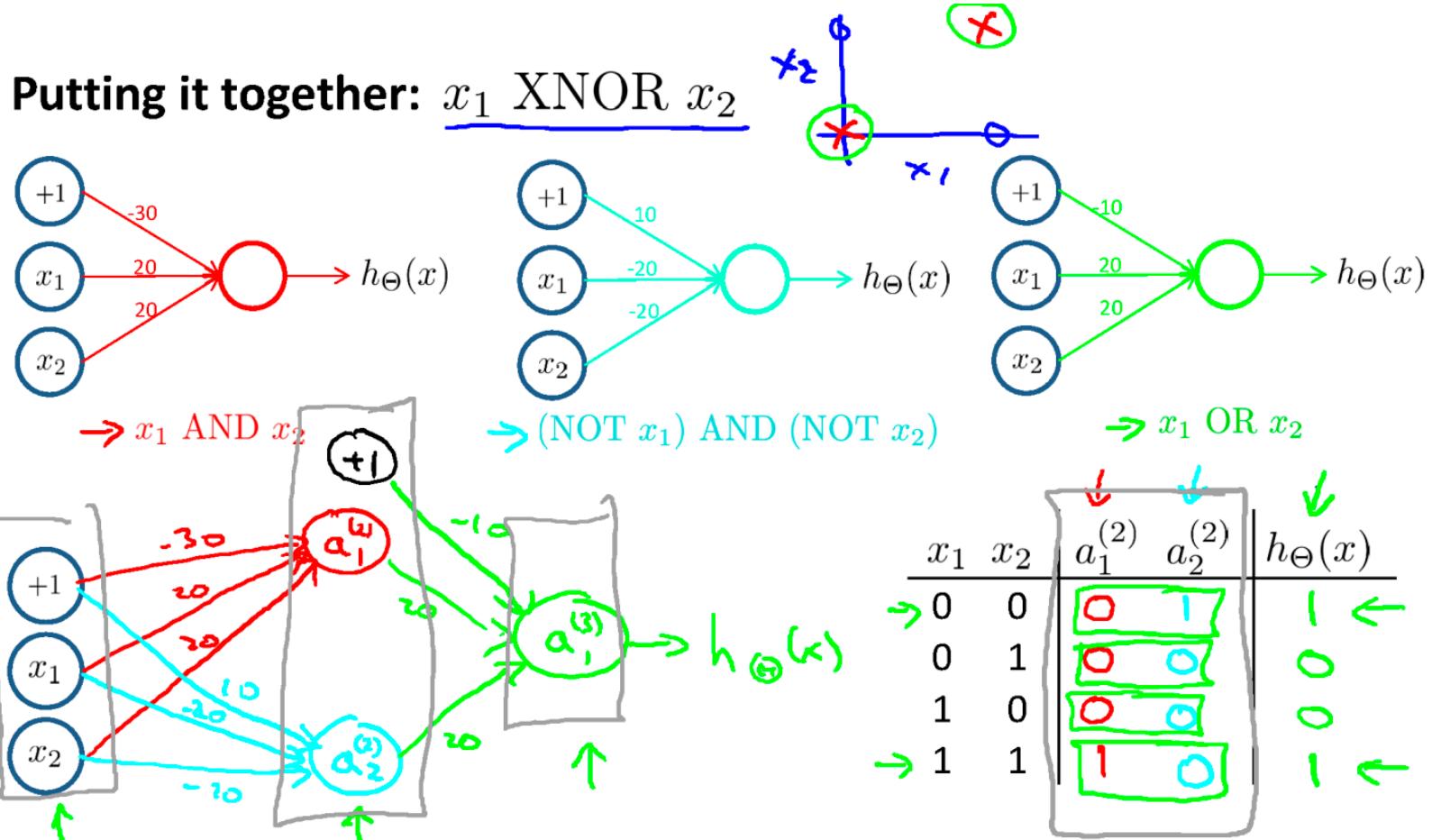


x_1	$h_\Theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

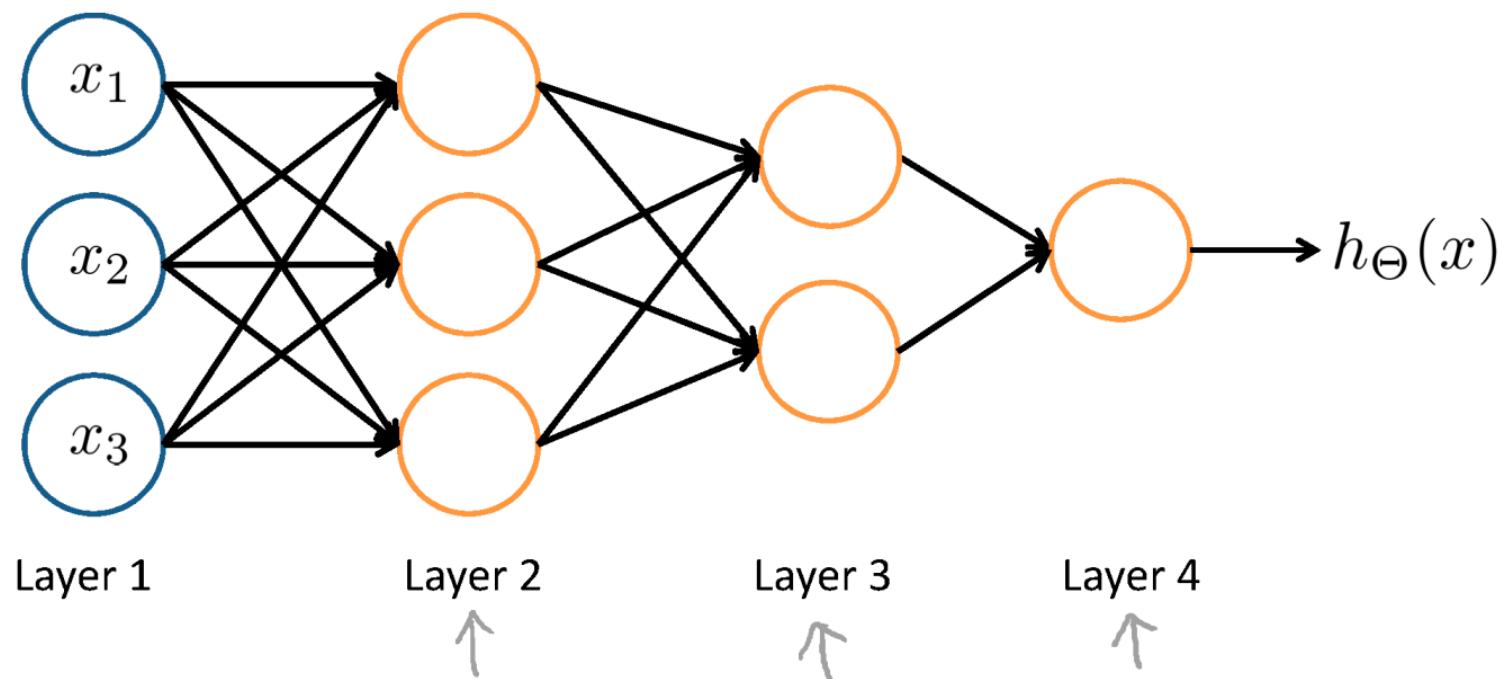
$$h_\Theta(x) = g(10 - 20x_1)$$

$\rightarrow (\text{NOT } x_1) \text{ AND } (\text{NOT } x_2)$
 $\underline{\underline{=1}} \text{ if and only if}$
 $\rightarrow x_1 = x_2 = 0$

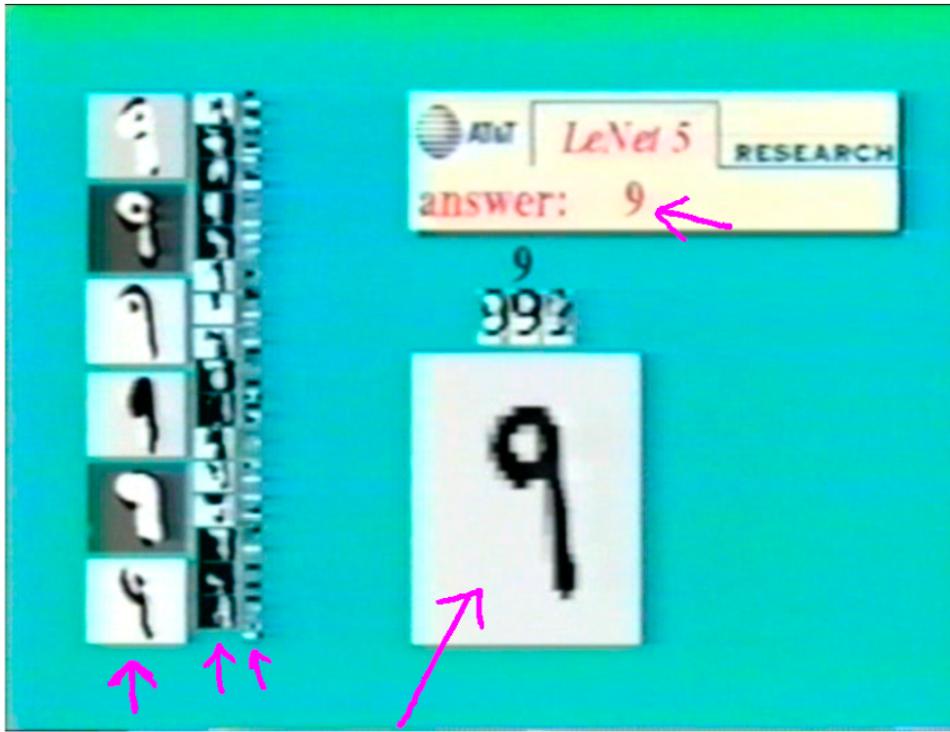
Putting it together: $x_1 \text{ XNOR } x_2$



Neural Network intuition

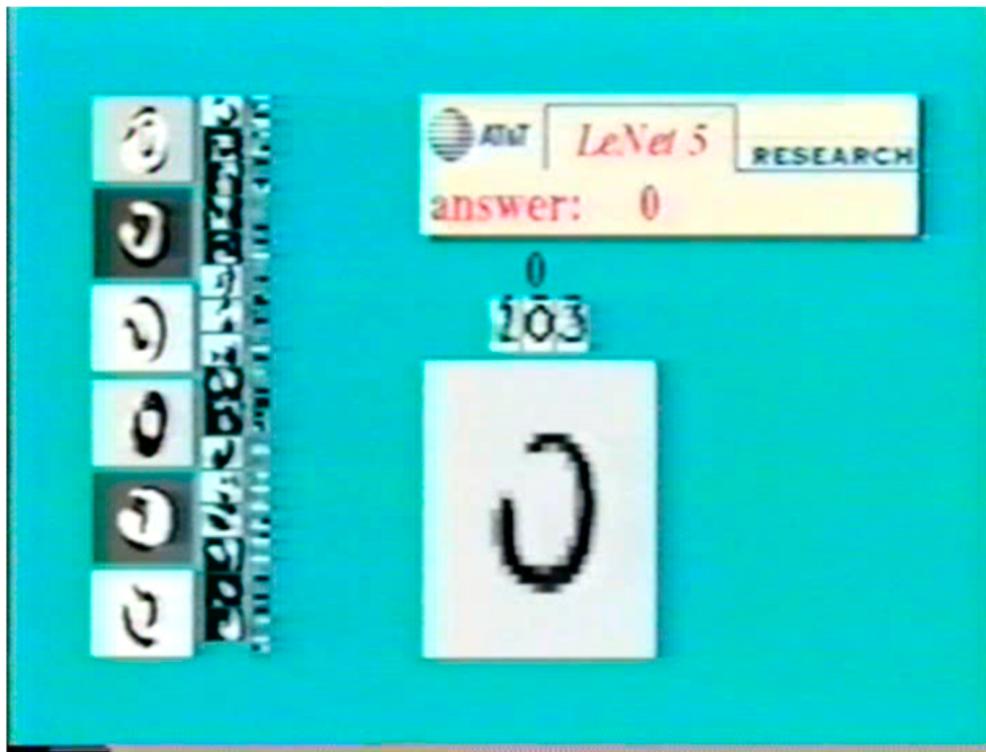


Handwritten digit classification



[Courtesy of Yann LeCun]

Handwritten digit classification



[Courtesy of Yann LeCun]

Neural Networks: Representation

Multi-class classification

Multiple output units: One-vs-all.

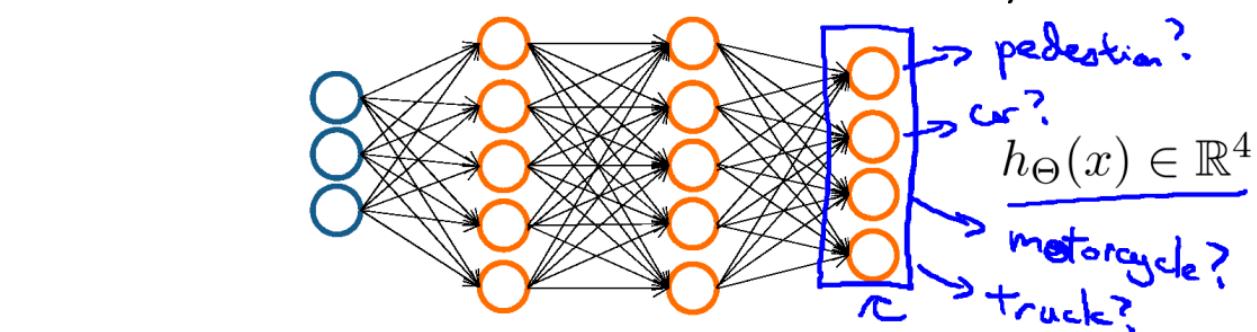


Pedestrian

Car

Motorcycle

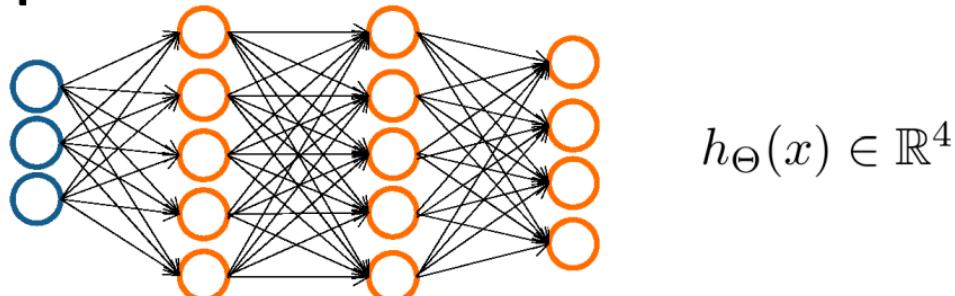
Truck



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

when pedestrian when car when motorcycle

Multiple output units: One-vs-all.



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

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

$\Rightarrow y^{(i)}$ 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

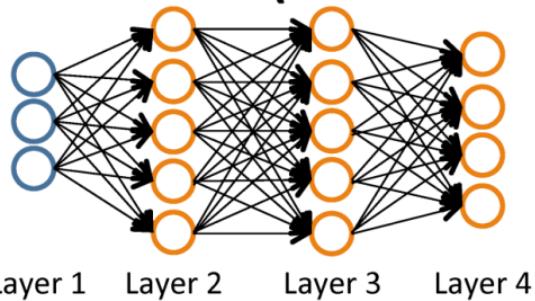
~~Previously
 $y \in \{1, 2, 3, 4\}$~~

$\underbrace{h_{\Theta}(x^{(i)})}_{\in \mathbb{R}^4} \approx y^{(i)}$

Neural Networks: Learning

Cost function

Neural Network (Classification)



Binary classification

$y = 0 \text{ or } 1$

1 output unit

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

$L =$ total no. of layers in network

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

Multi-class classification (K classes)

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

pedestrian car motorcycle truck

K output units

Cost function

Logistic regression:

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

Neural network:

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

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



Neural Networks: Learning

Backpropagation algorithm



Gradient computation

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

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

Need code to compute:

$$\rightarrow -\underline{J(\Theta)}$$

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

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

Gradient computation

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

Forward propagation:

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

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

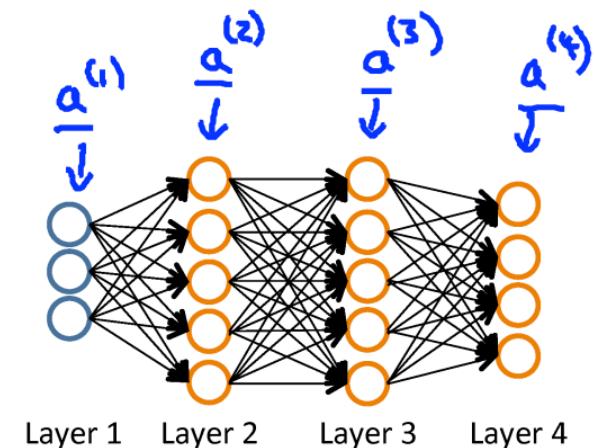
$$\rightarrow \underline{a}^{(2)} = g(\underline{z}^{(2)}) \quad (\text{add } \underline{a}_0^{(2)})$$

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

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

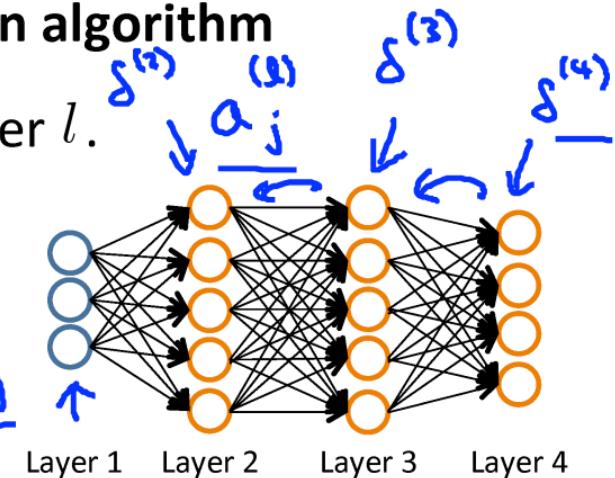
$$\rightarrow \underline{z}^{(4)} = \Theta^{(3)} \underline{a}^{(3)}$$

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



Gradient computation: Backpropagation algorithm

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



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

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

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

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

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

(ignoring λ ; if $\lambda = 0$)

Backpropagation algorithm

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

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

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

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

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

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

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

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

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

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

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

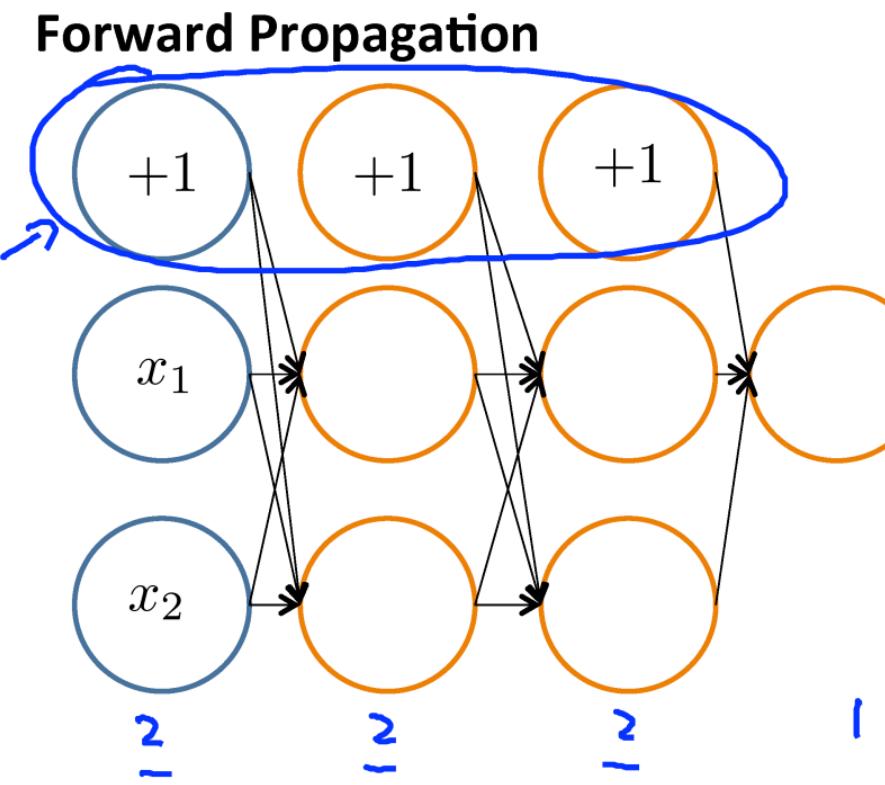
$\rightarrow D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$ if $j = 0$

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

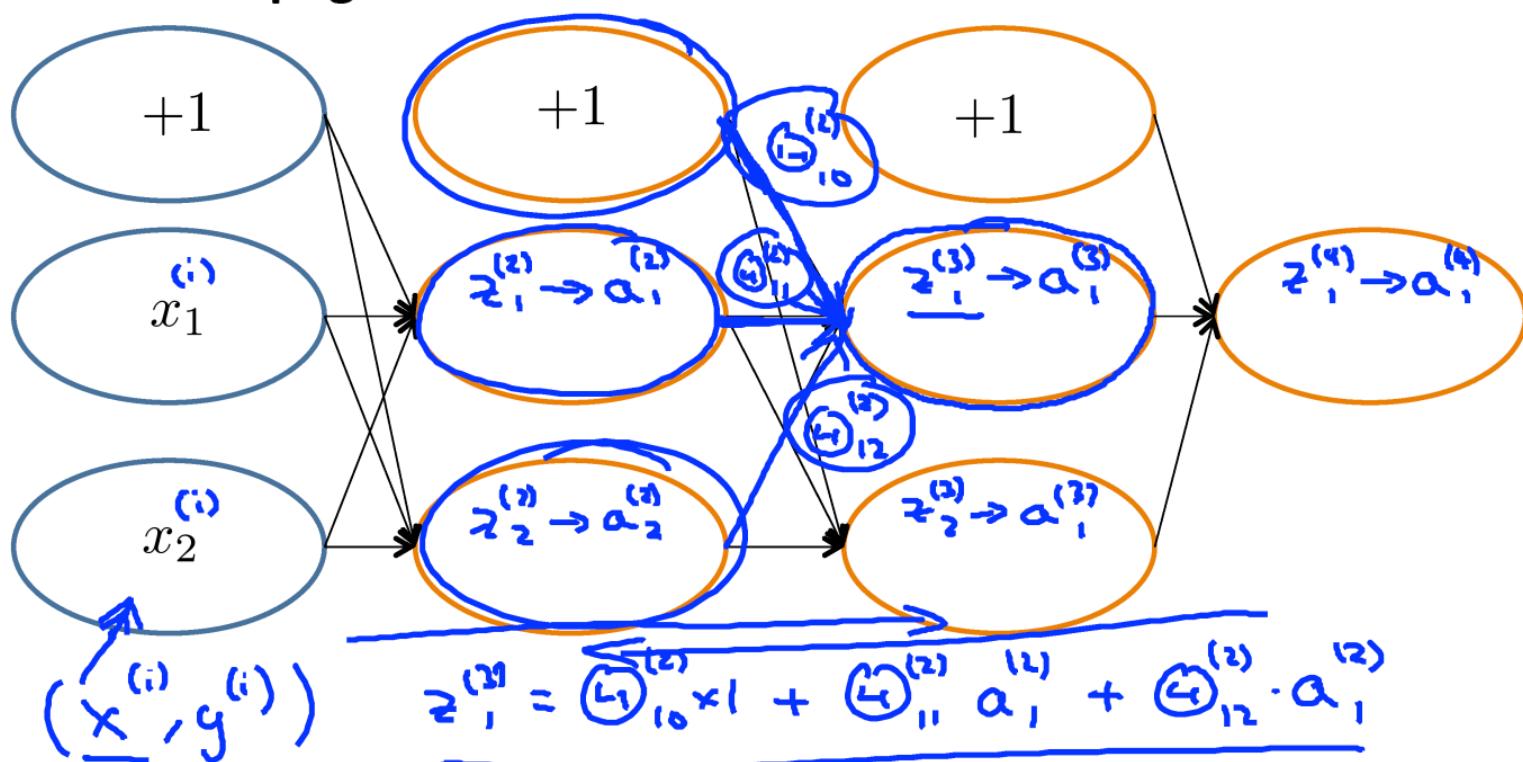
Neural Networks: Learning

Backpropagation intuition





Forward Propagation



What is backpropagation doing?

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

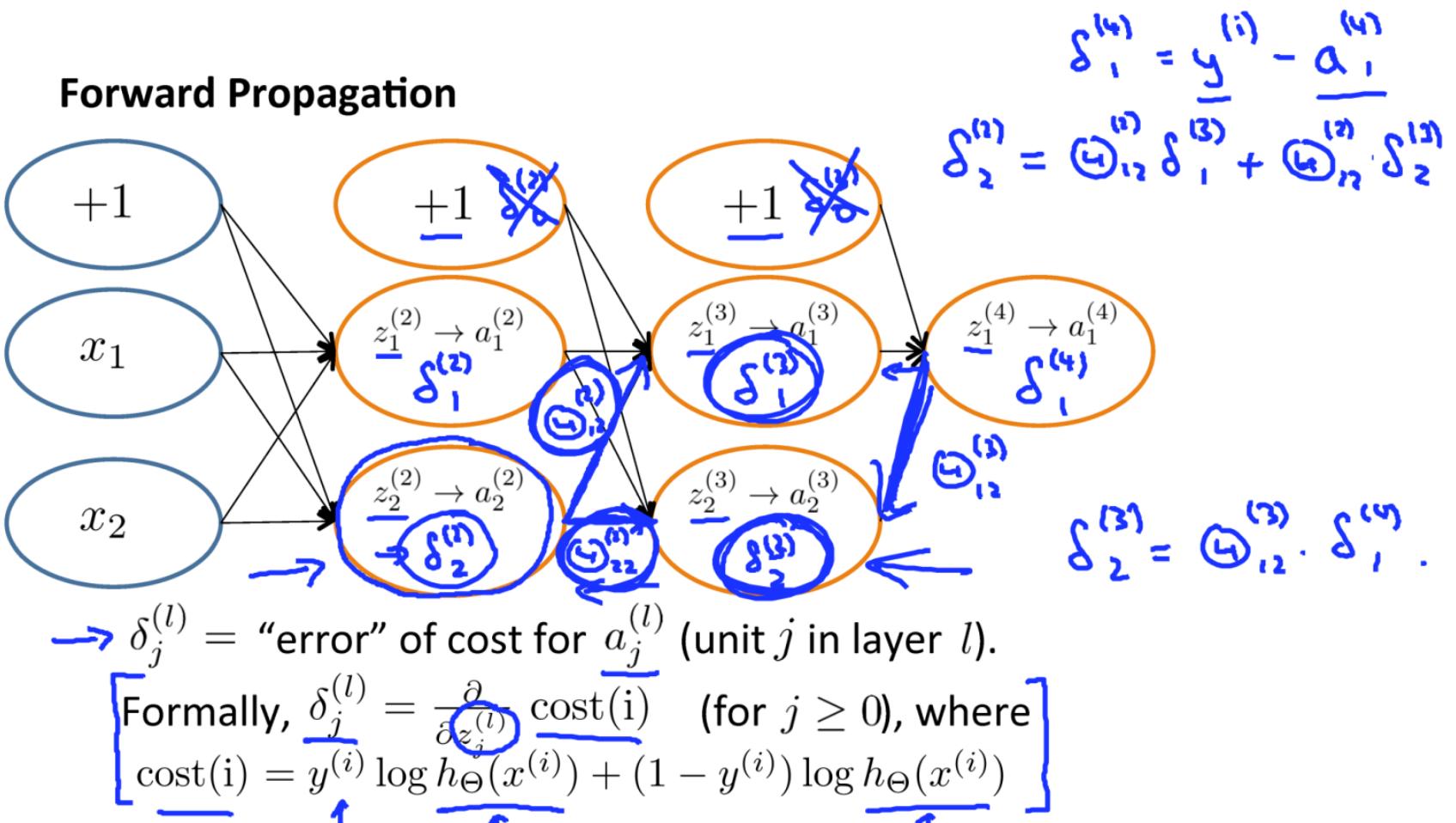
$(x^{(i)}, y^{(i)})$

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

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

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

I.e. how well is the network doing on example i? $y^{(i)}$



Neural Networks: Learning

Implementation note: Unrolling parameters



Advanced optimization

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

Neural Network (L=4):

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

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

“Unroll” into vectors

Example

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

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

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

$\Theta^{(1)}$ $\Theta^{(2)}$ $\Theta^{(3)}$

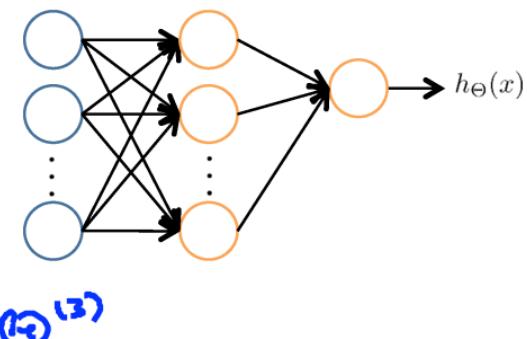
$\rightarrow \text{thetaVec} = [\Theta^{(1)}(:); \Theta^{(2)}(:); \Theta^{(3)}(:)]$

$\rightarrow \text{DVec} = [D^{(1)}(:); D^{(2)}(:); D^{(3)}(:)]$

$\Theta^{(1)} = \text{reshape}(\text{thetaVec}(1:110), 10, 11);$

$\Theta^{(2)} = \text{reshape}(\text{thetaVec}(111:220), 10, 11);$

$\Theta^{(3)} = \text{reshape}(\text{thetaVec}(221:231), 1, 11);$



Learning Algorithm

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

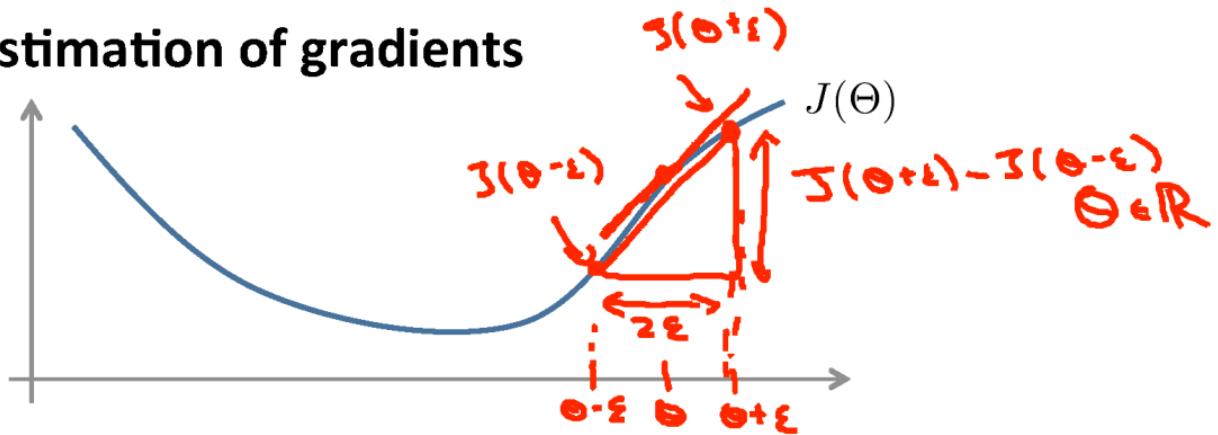
```
function [jval, gradientVec] = costFunction(thetaVec)
    → From thetaVec, get  $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ . reshape
    → Use forward prop/back prop to compute  $D^{(1)}, D^{(2)}, D^{(3)}$   $J(\Theta)$ 
    and  $D_1^{(1)}, D_2^{(2)}, D_3^{(3)}$ 
    Unroll  $\underline{\quad}$  to get gradientVec.
```

Neural Networks: Learning

Gradient checking



Numerical estimation of gradients



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

$\epsilon = 10^{-4}$

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

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

Parameter vector θ

→ $\theta \in \mathbb{R}^n$ (E.g. θ is “unrolled” version of $\underline{\Theta^{(1)}}, \underline{\Theta^{(2)}}, \underline{\Theta^{(3)}}$)

$$\rightarrow \theta = [\theta_1, \theta_2, \theta_3, \dots, \theta_n]$$

$$\rightarrow \frac{\partial}{\partial \underline{\theta_1}} J(\theta) \approx \frac{J(\theta_1 + \epsilon, \theta_2, \theta_3, \dots, \theta_n) - J(\theta_1 - \epsilon, \theta_2, \theta_3, \dots, \theta_n)}{2\epsilon}$$

$$\rightarrow \frac{\partial}{\partial \underline{\theta_2}} J(\theta) \approx \frac{J(\theta_1, \theta_2 + \epsilon, \theta_3, \dots, \theta_n) - J(\theta_1, \theta_2 - \epsilon, \theta_3, \dots, \theta_n)}{2\epsilon}$$

⋮

$$\rightarrow \frac{\partial}{\partial \underline{\theta_n}} J(\theta) \approx \frac{J(\theta_1, \theta_2, \theta_3, \dots, \theta_n + \epsilon) - J(\theta_1, \theta_2, \theta_3, \dots, \theta_n - \epsilon)}{2\epsilon}$$

```

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

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

$$\begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_i + \epsilon \\ \vdots \\ \theta_n \end{bmatrix} \rightarrow \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_i - \epsilon \\ \vdots \\ \theta_n \end{bmatrix}$$

Check that $\boxed{\text{gradApprox}} \approx \boxed{\text{DVec}}$ ←
 ↑
From back prop.

Implementation Note:

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

↓ ↓ ↓

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

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.

Neural Networks: Learning

Random initialization

Initial value of Θ

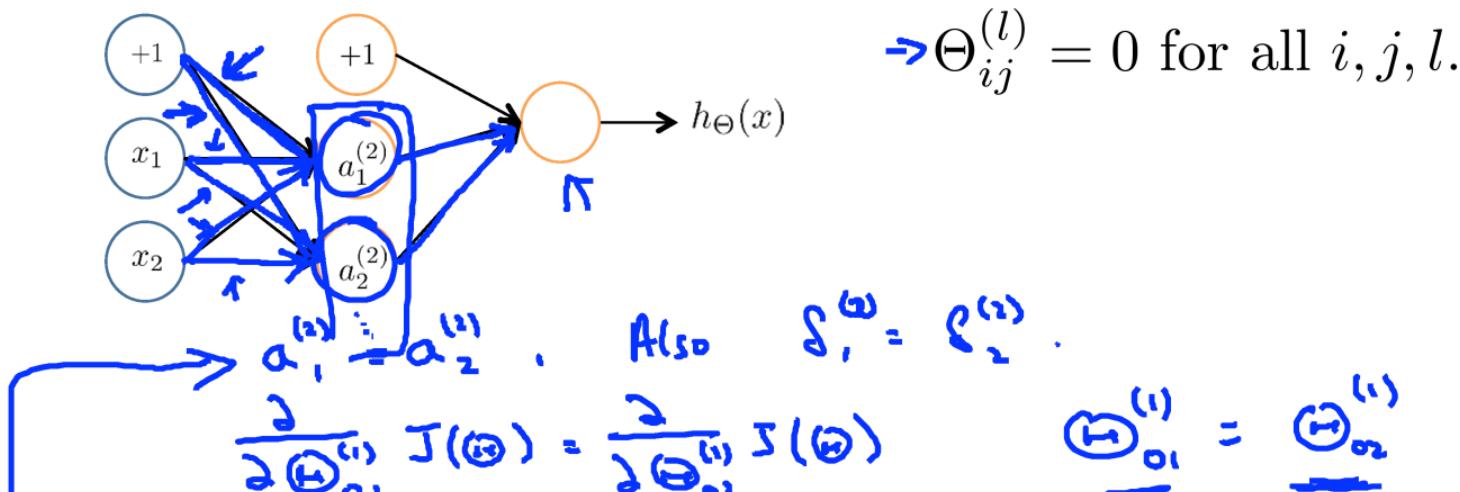
For gradient descent and advanced optimization method, need initial value for Θ .

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

Consider gradient descent

Set initialTheta = zeros(n,1) ?

Zero initialization



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

$$\underline{\alpha_1^{(2)}} = \underline{\alpha_2^{(2)}}$$

Random initialization: Symmetry breaking

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

E.g.

Random 10x11 matrix (betw. 0 and 1)

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

$[-\epsilon, \epsilon]$

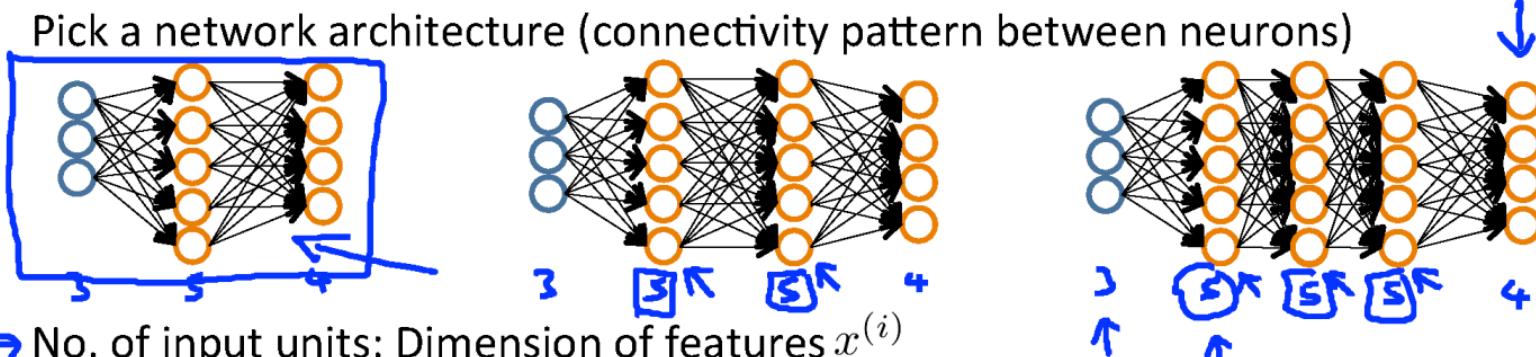
→ Theta2 = rand(1,11) * (2*INIT_EPSILON)
- INIT_EPSILON;

Neural Networks: Learning

Putting it together

Training a neural network

Pick a network architecture (connectivity pattern between neurons)



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

→ No. output units: Number of classes

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

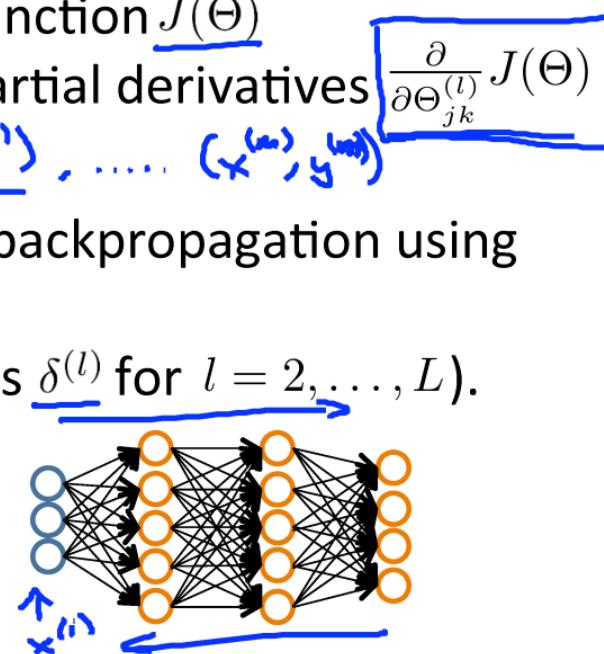
$$y \in \{1, 2, 3, \dots, 103\}$$

~~$y \in \mathbb{R}$~~

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

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 $\frac{\partial}{\partial \Theta_j^{(l)}} J(\Theta)$
- **for** $i = 1:m$ { $(x^{(i)}, y^{(i)})$ $(x^{(i)}, y^{(i)})$, ..., $(x^{(m)}, y^{(m)})$
- Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$
- Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \dots, L$.
- $\Delta^{(1)} := \Delta^{(1)} + \delta^{(1)} (a^{(1)})^T$
 \vdots
 $\Delta^{(L)} := \Delta^{(L)} + \delta^{(L)} (a^{(L)})^T$
- Compute $\frac{\partial}{\partial \Theta_j^{(k)}} J(\Theta)$.

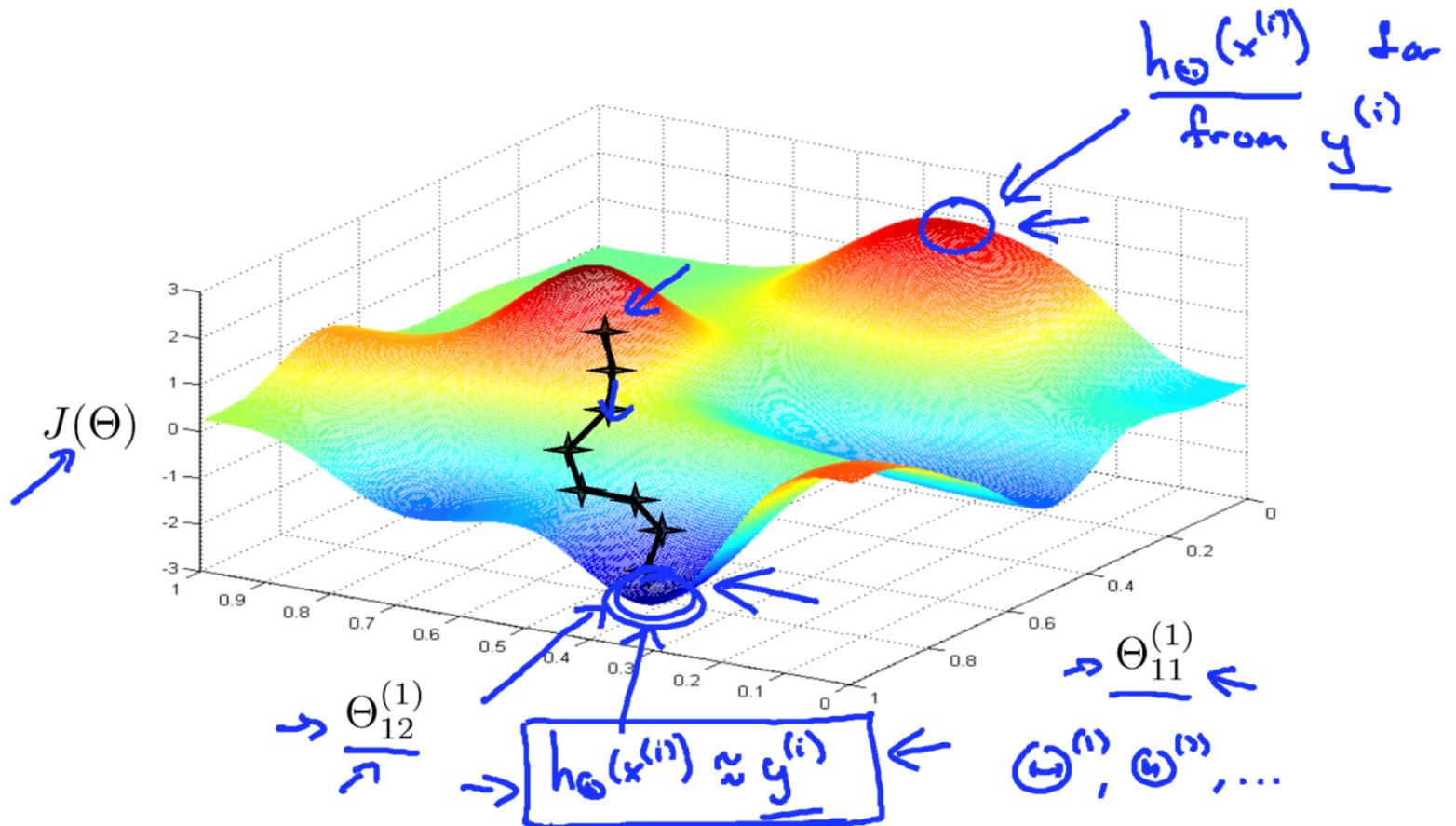


Training a neural network

- 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

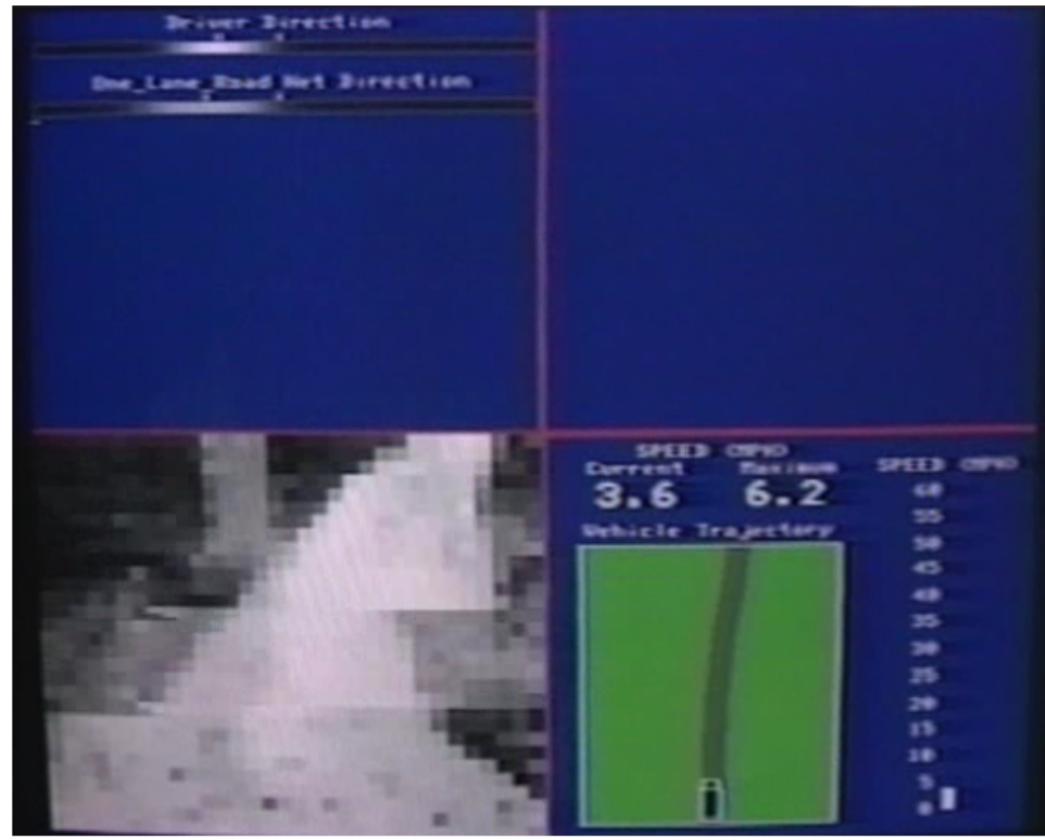
$$\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta) \quad \uparrow$$

$J(\Theta)$ — non-convex.



Neural Networks: Learning

Backpropagation example: Autonomous driving (optional)



[Courtesy of Dean Pomerleau]