



# Introduction to Data Science

## (Lecture 11)

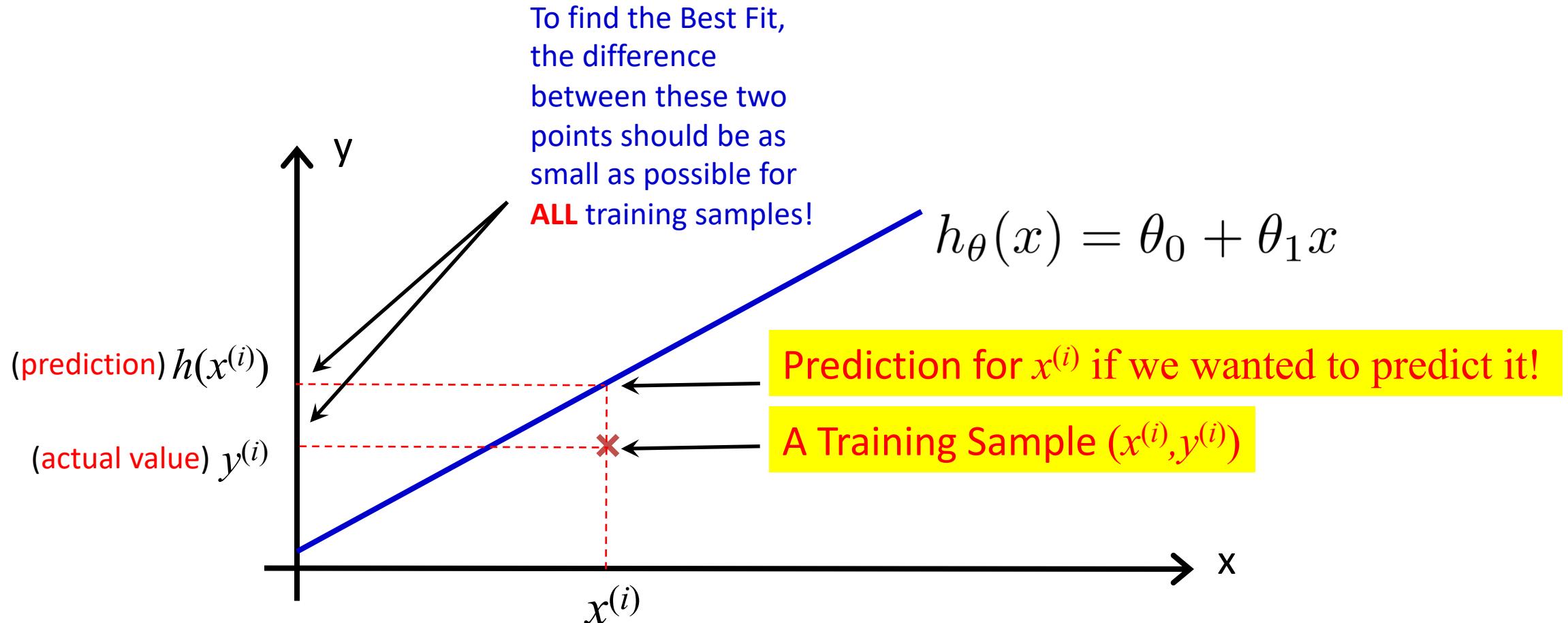
**Dr. Mohammad Pourhomayoun**  
Assistant Professor  
Computer Science Department  
California State University, Los Angeles





# Continue: Gradient Descent for Linear Regression

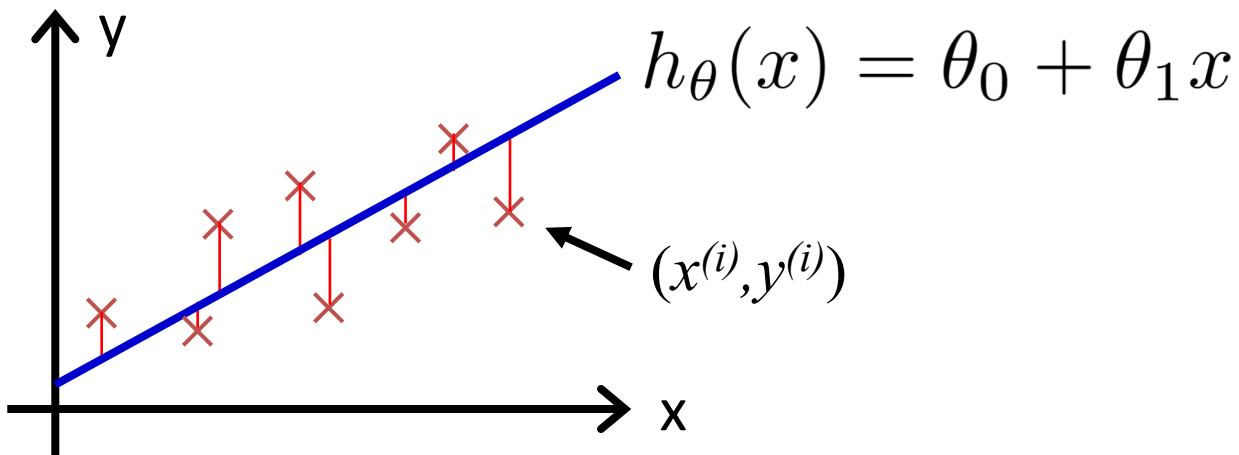
- Thus, we just need to minimize  $(h(x^{(i)}) - y^{(i)})^2$  for every training sample  $i$ .
- Question: Why squared?



Let's define a **Cost Function**  $J(\theta_0, \theta_1)$  as the summation (or average) of all differences between "training samples" and the "regression line".

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

Thus, the Best Fit Line is the line with minimum Cost Function  $J(\theta_0, \theta_1)$ .

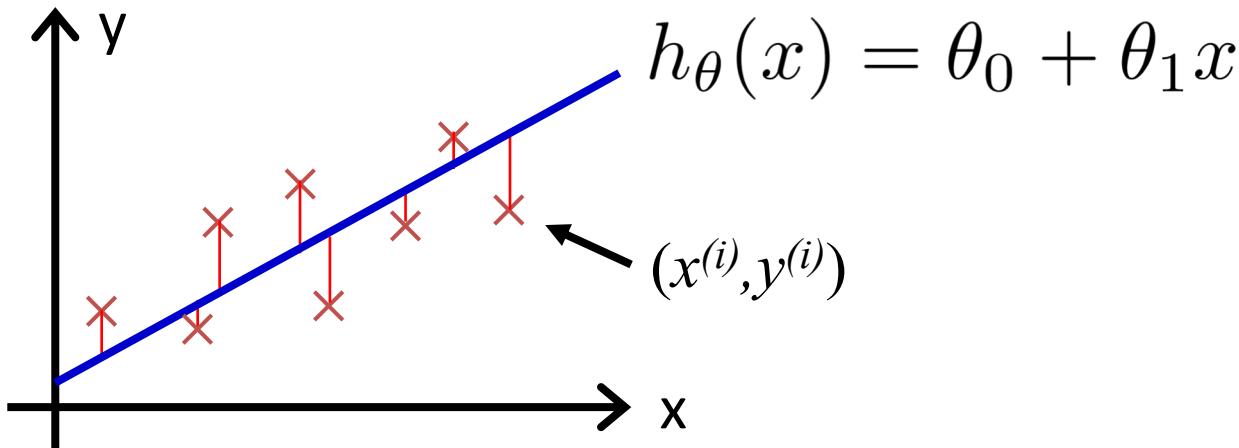


**Training Set:**  $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

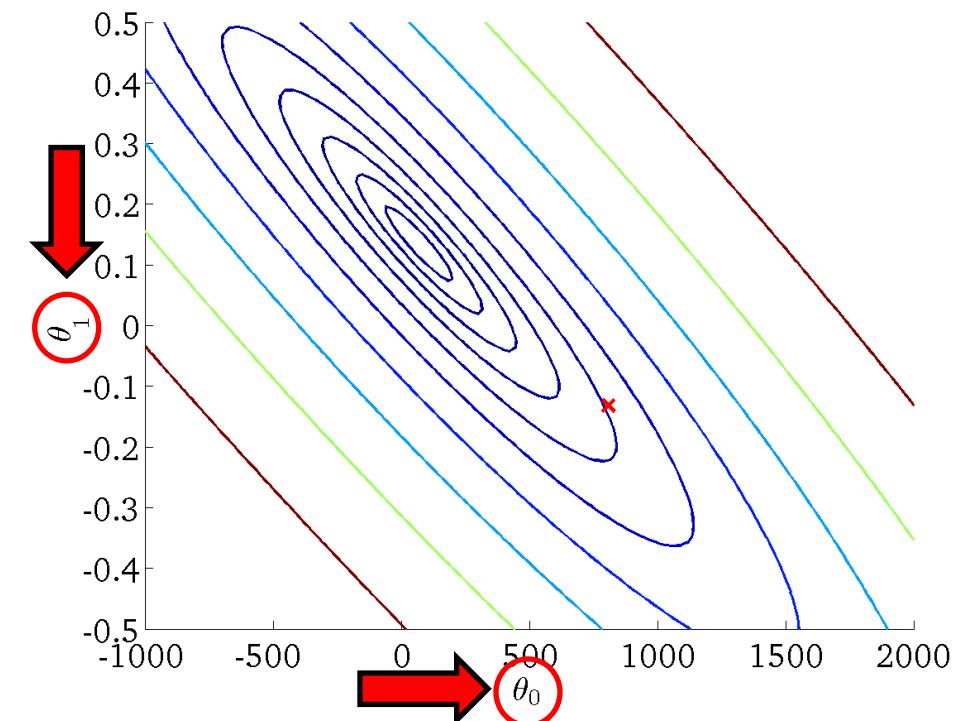
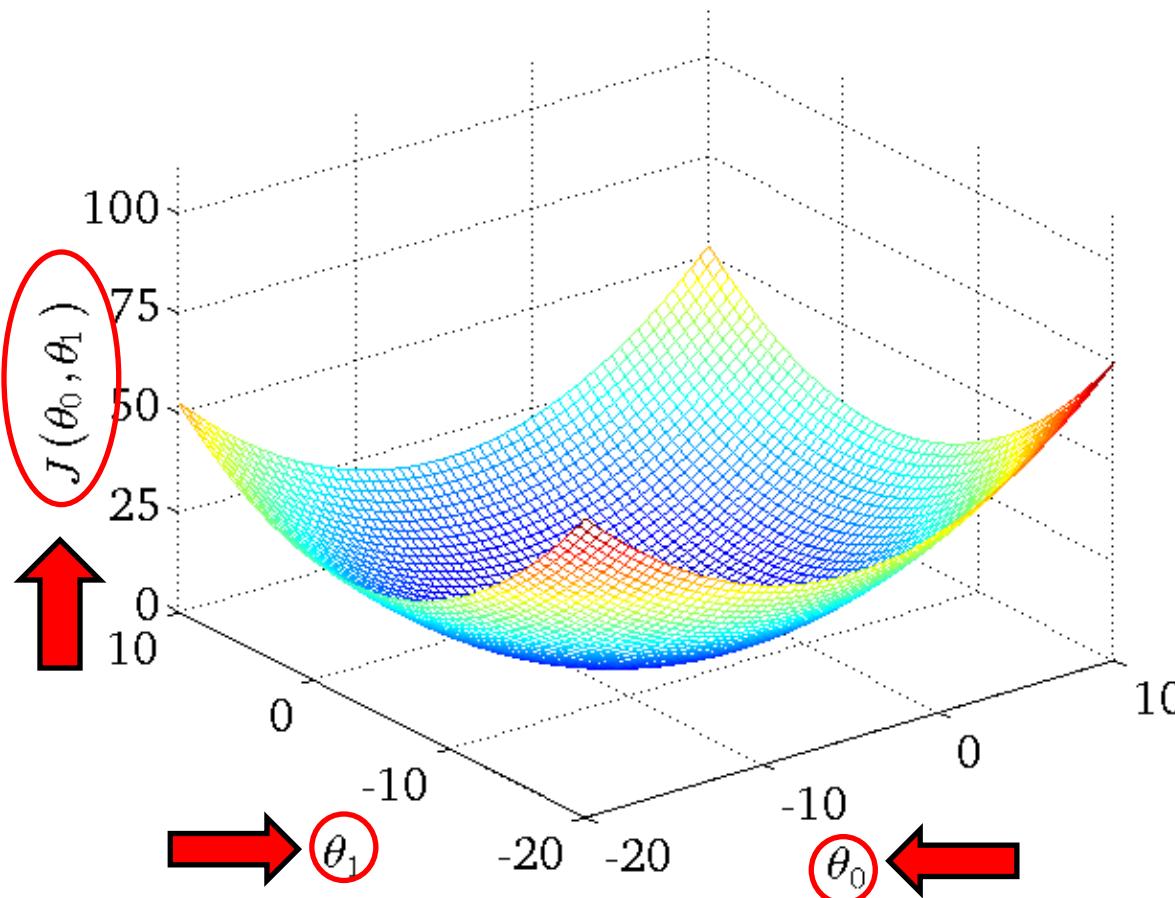
**Cost Function:**  $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$

**Goal:** To find the best parameters  $\theta_0, \theta_1$  that minimizes the **Cost Function**  $J(\theta_0, \theta_1)$ :

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



- It turns out that the **Cost Function**  $J(\theta_0, \theta_1)$  is a Convex (Bowl-Shaped) function. So, it has a global minimum! Example:



\* Reference: Andrew Ng, Machine Learning, Stanford University.

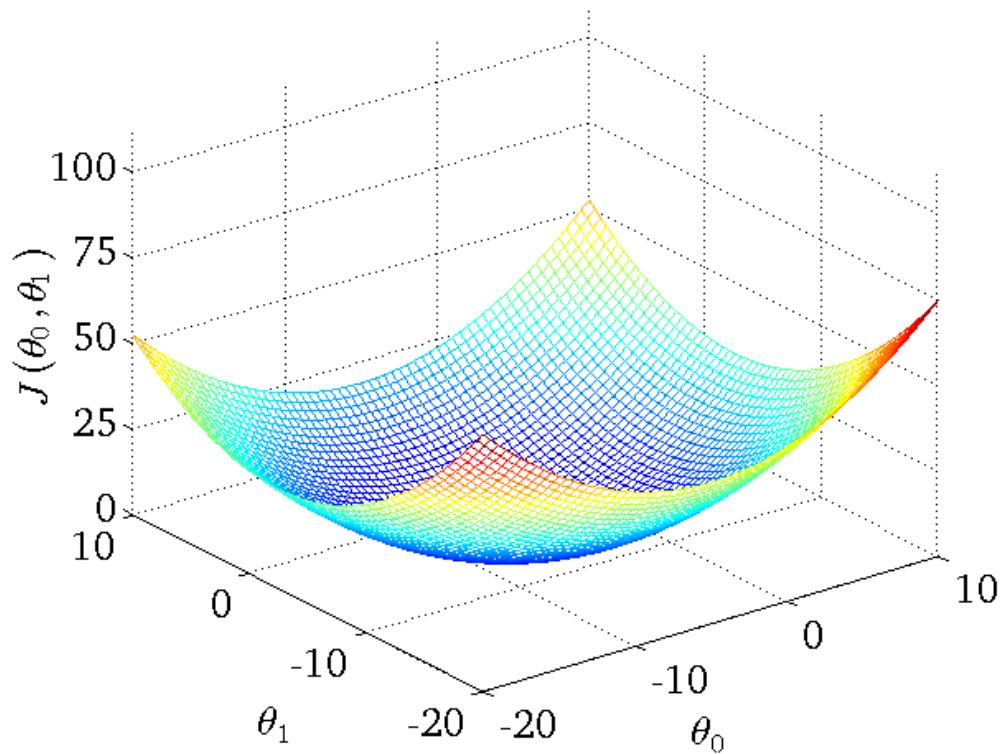


**Regression Model:**  $h_\theta(x) = \theta_0 + \theta_1 x$

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

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

**Example: a convex cost function J:**  
(Convex means bowl-shaped!)



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

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

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

### Question:

How to minimize the cost function?

### Answer:

We can use “**Gradient Descent**” algorithm to minimize the cost function and find the parameters (a generalization of the usual concept of derivative for functions of several variables).

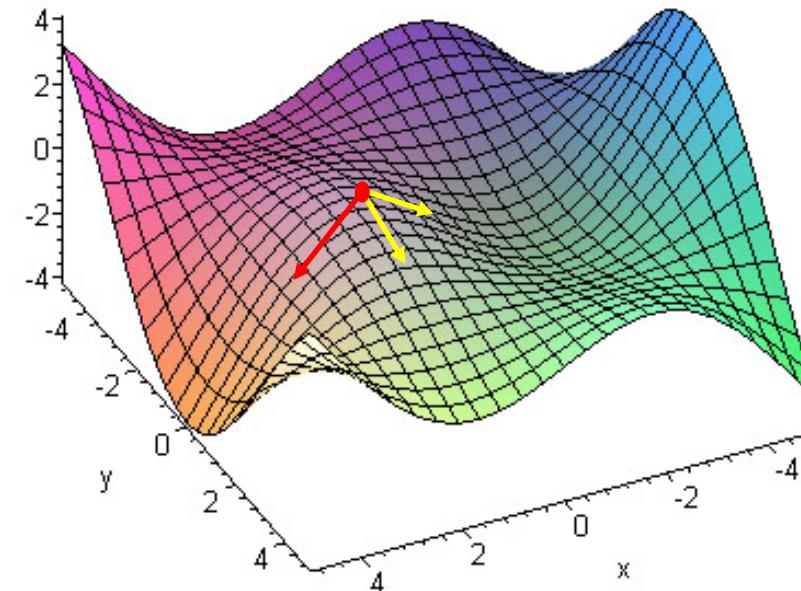




# Gradient Descent Algorithm

# Gradient Descent Algorithm\*

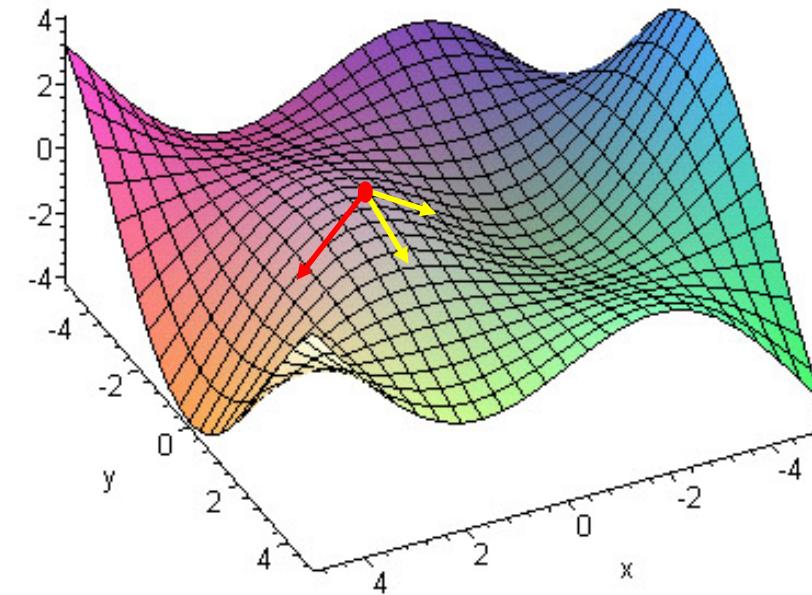
- Gradient Descent is an **iterative** optimization approach that tries to minimize a function.
- The **gradient** is a generalization of the usual concept of **derivative** for functions of **several variables**.
- The **gradient represents the slope of the tangent** of the graph of the function.
- The **negative gradient points in the direction of the greatest rate of reduction** of the function, and its **magnitude** is the **slope** of in that direction.



\* Reference: Andrew Ng, Machine Learning, Stanford University.

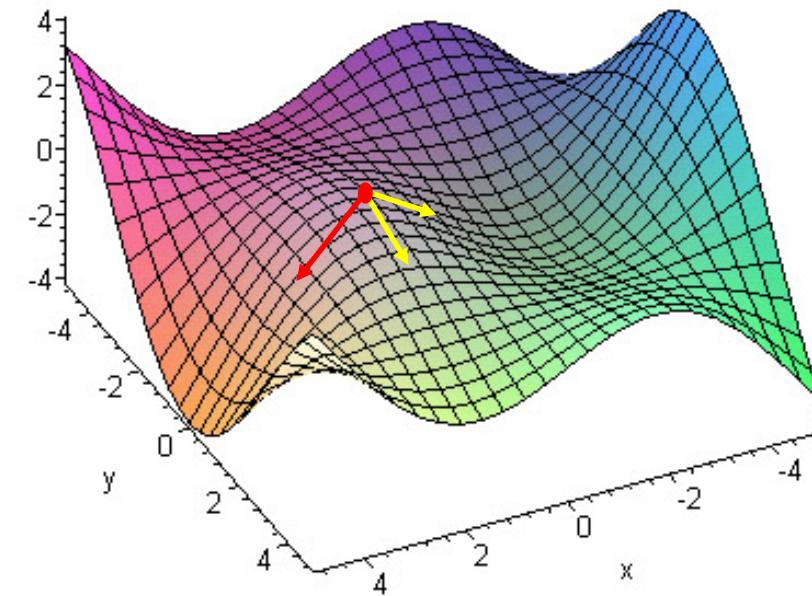
# Gradient Descent Algorithm\*

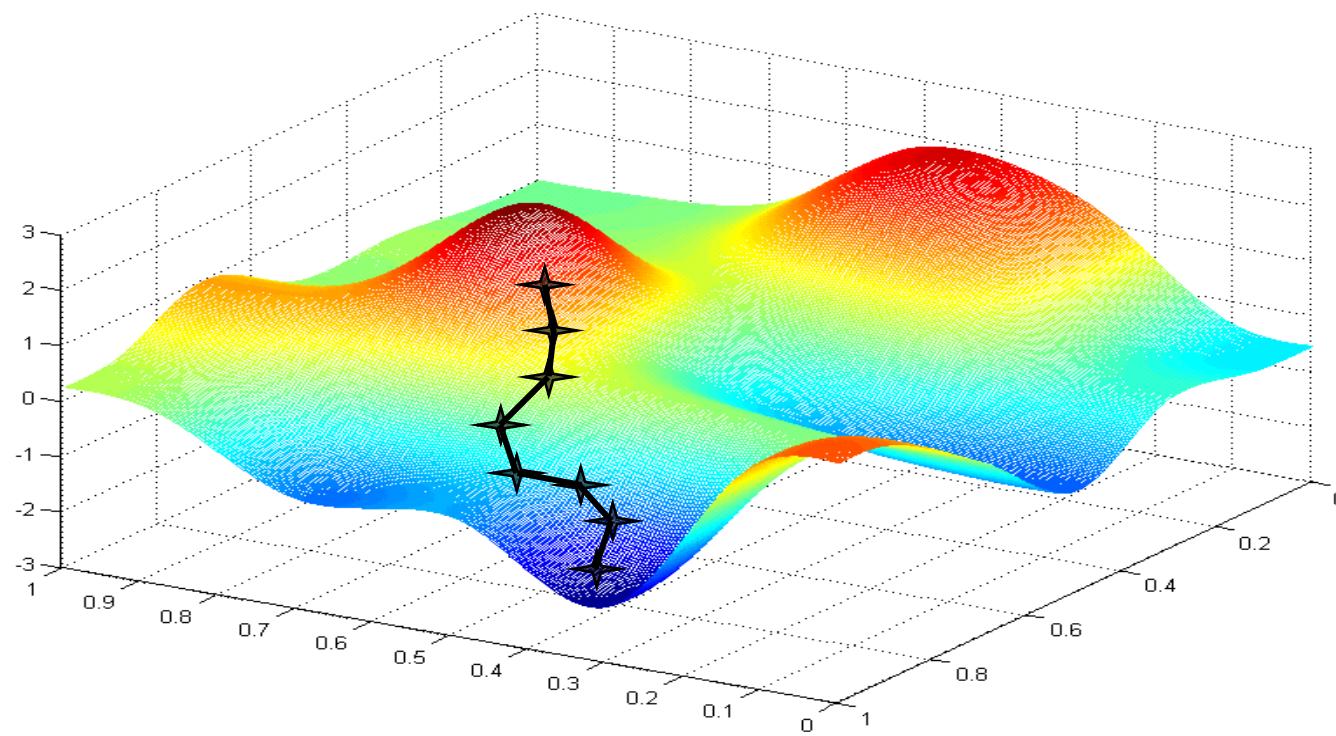
- Gradient Descent is an iterative optimization approach that tries to minimize a function.
- The negative direction of the gradient at the current point is the **Steepest Direction** at the moment.
- In other word, a differentiable function  $F$  decreases fastest if one moves in the direction of the **negative gradient** of  $F$  at point  $x$ .



# Gradient Descent Algorithm\*

- Gradient Descent is an iterative optimization approach that tries to minimize a function.
- To minimize a function, Gradient Descent starts with an initial set of parameter values, and then iteratively takes steps with a predefined rate in the **negative direction of the function gradient** at the current point (The Steepest Direction).

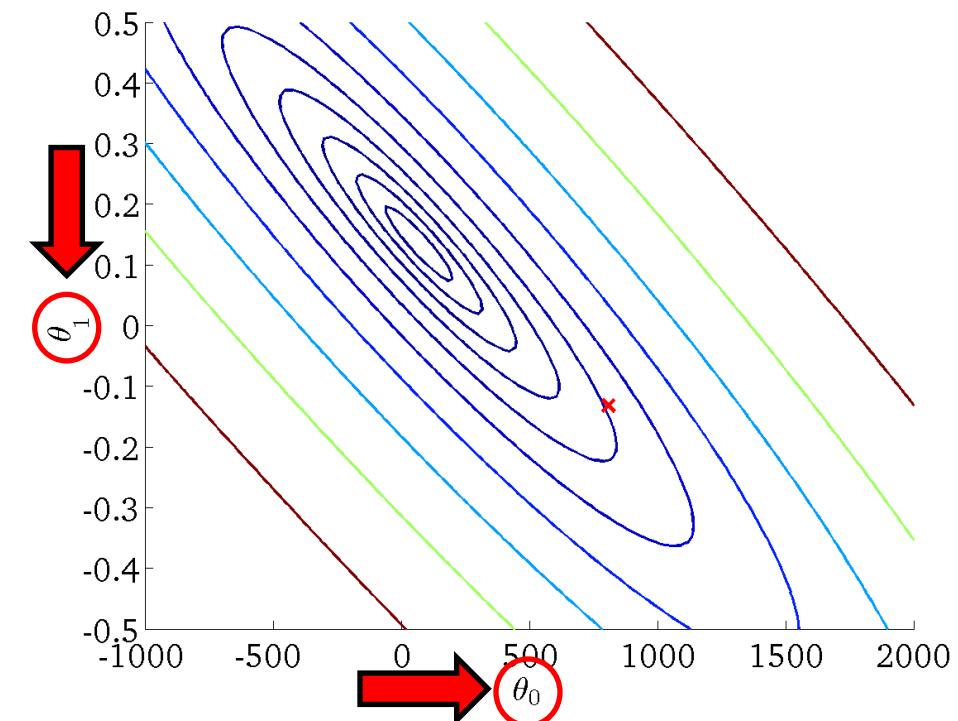
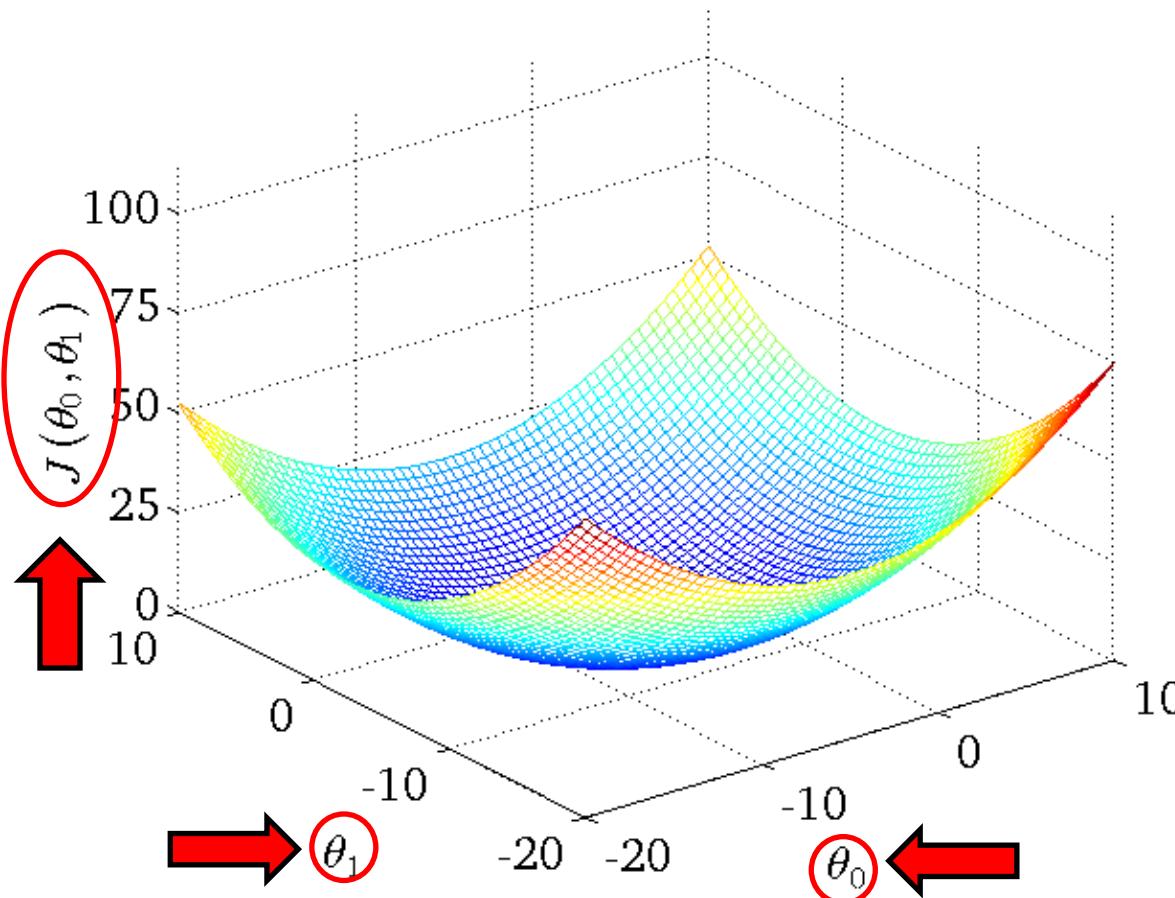




\* Reference: Andrew Ng, Machine Learning, Stanford University.



- It turns out that the **Cost Function**  $J(\theta_0, \theta_1)$  is a Convex (Bowl-Shaped) function. So, it has a global minimum! Example:



\* Reference: Andrew Ng, Machine Learning, Stanford University.



# Gradient Descent Algorithm

repeat until convergence {

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

}

Gradient (slope of tangent line)

$\alpha > 0$  is Learning Rate that controls the step size toward the steepest direction.

---

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

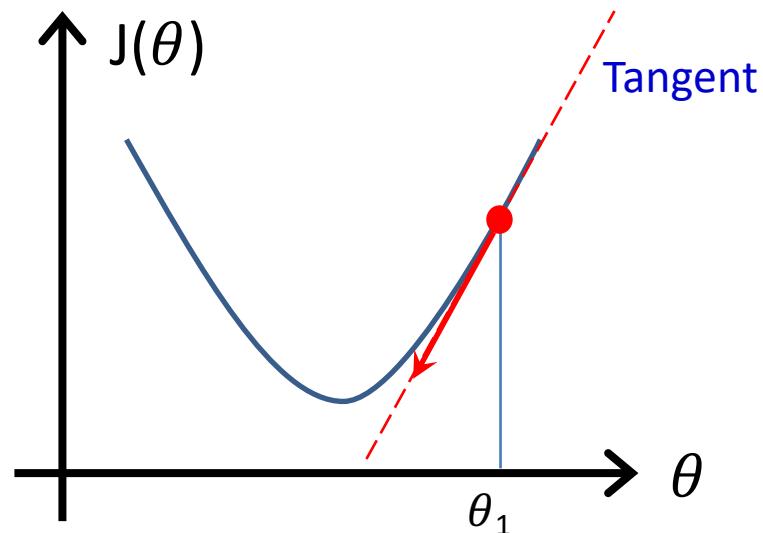
repeat until convergence {

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

}

Gradient (slope of tangent line)

$\alpha > 0$  is Learning Rate that controls the step size toward the steepest direction.



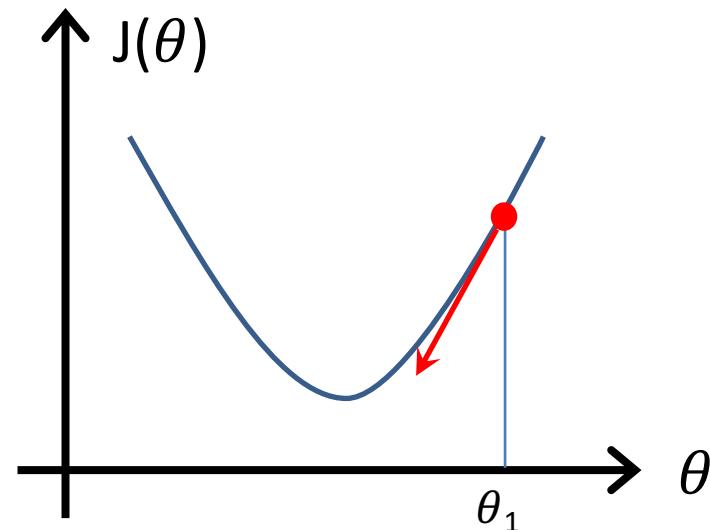
# Gradient Descent Algorithm

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**      **Gradient (slope of tangent line)**



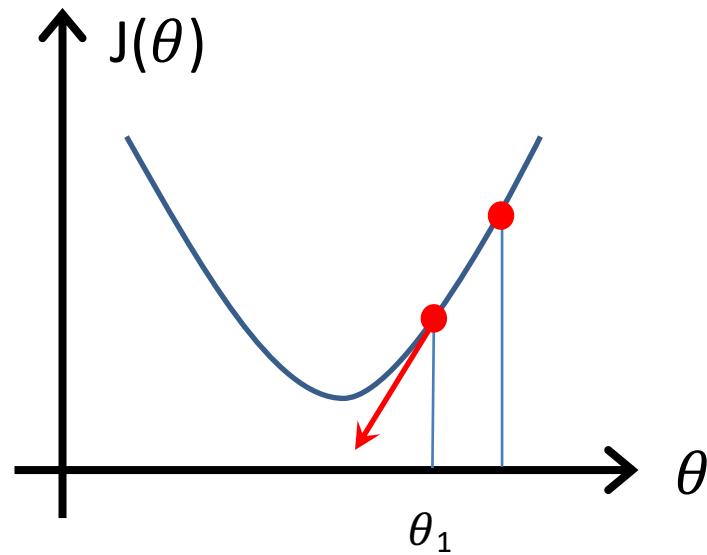
# Gradient Descent Algorithm

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**      **Gradient (slope of tangent line)**



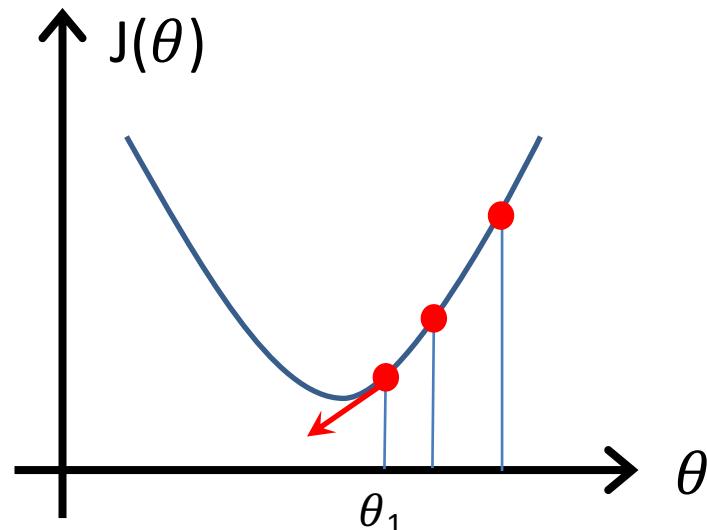
# Gradient Descent Algorithm

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      Gradient (slope of tangent line)



# Gradient Descent Algorithm

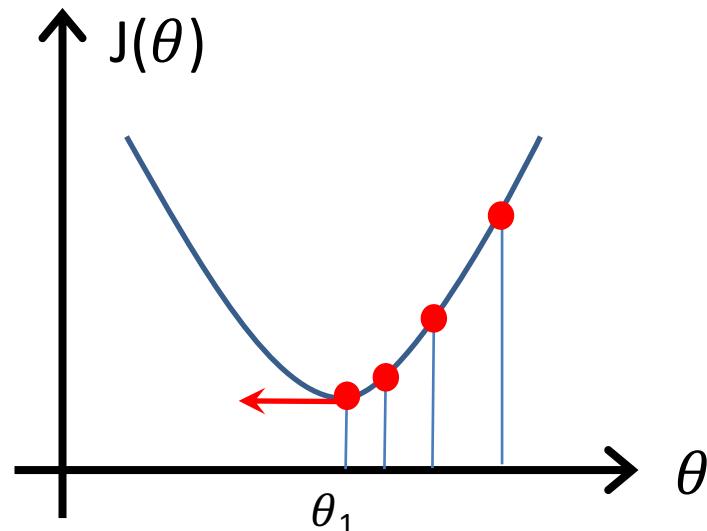
repeat until convergence {

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

}

Learning Rate

Gradient (slope of tangent line)



# Gradient Descent Algorithm

repeat until convergence {

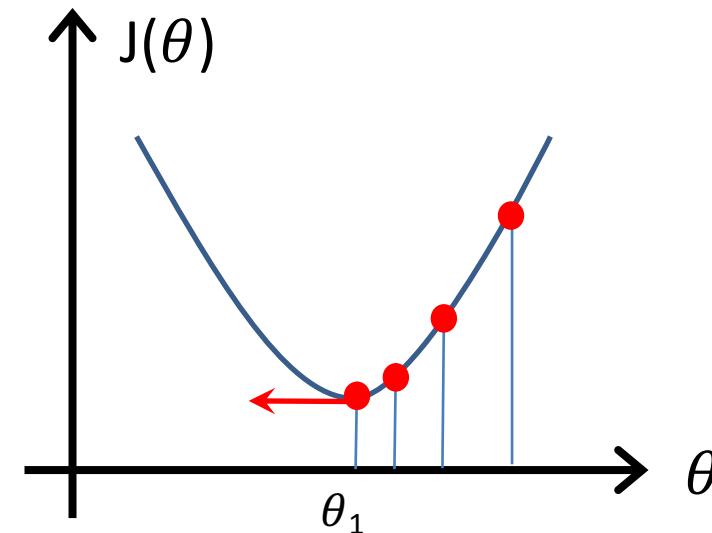
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) \quad (\text{for } j = 0 \text{ and } j = 1)$$

}

Learning Rate

Gradient (slope of tangent line)

**Note1:** At this point, the convergence achieved (the minimum found): **The slope of a horizontal line is zero, so no more move!**

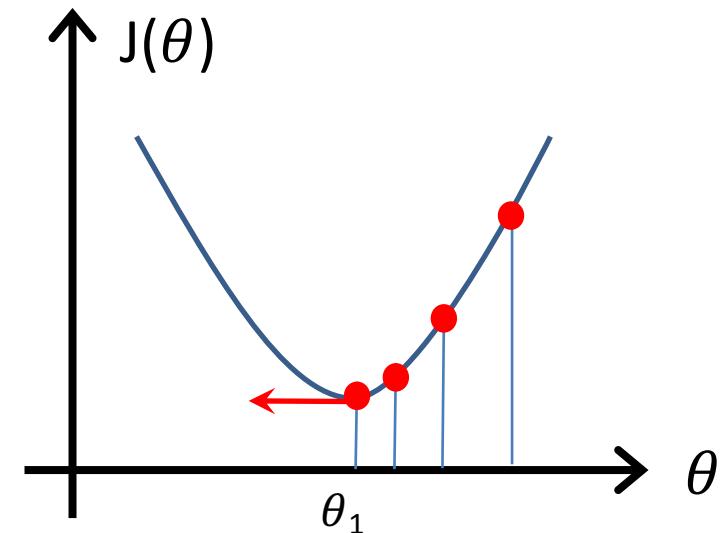


# Gradient Descent Algorithm

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

Learning Rate      Gradient (slope of tangent line)

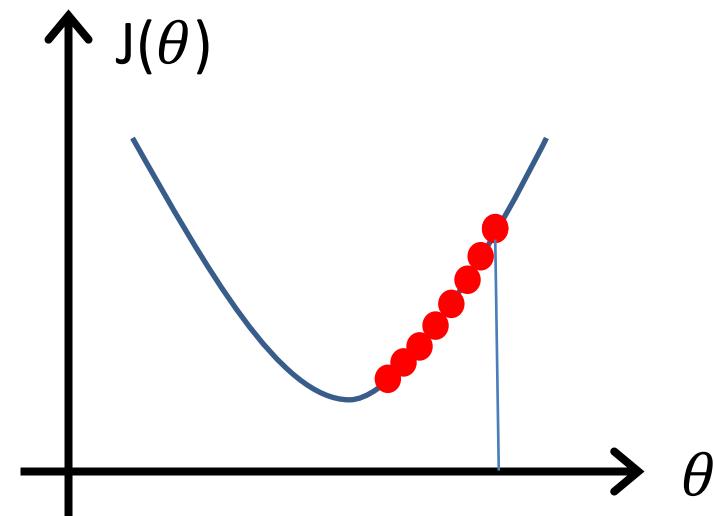
**Note2:** As we move toward a local minimum, the gradient descent **automatically takes smaller steps** in each iteration (because the slope gets smaller).



- $\alpha$  is Learning Rate that controls the step size toward the steepest direction.

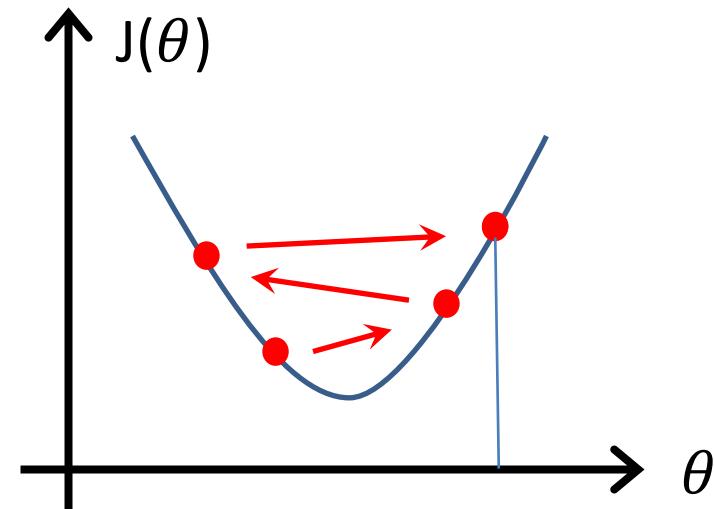
- Too Small  $\alpha$ :

Converge will be too slow.



- Too Large  $\alpha$ :

The algorithm may overshoot the minimum.  
It may fail to converge.



# Gradient Descent for Linear Regression

Specifically, for Linear Regression with one feature, let's put  $h(x)$  and  $J$  functions in the equations and calculate the derivatives to find an explicit update rule:

- **Gradient descent general rule/algorithm for two parameters:**

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

- **Linear Regression Model:**

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

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



# Gradient Descent for Linear Regression

- Gradient descent general algorithm:

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

- Linear Regression Model:

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

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

- After plugging J and calculating the Partial Derivatives:

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

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



# Gradient Descent for Linear Regression

- After plugging in and calculating the Partial Derivatives:

repeat until convergence {

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

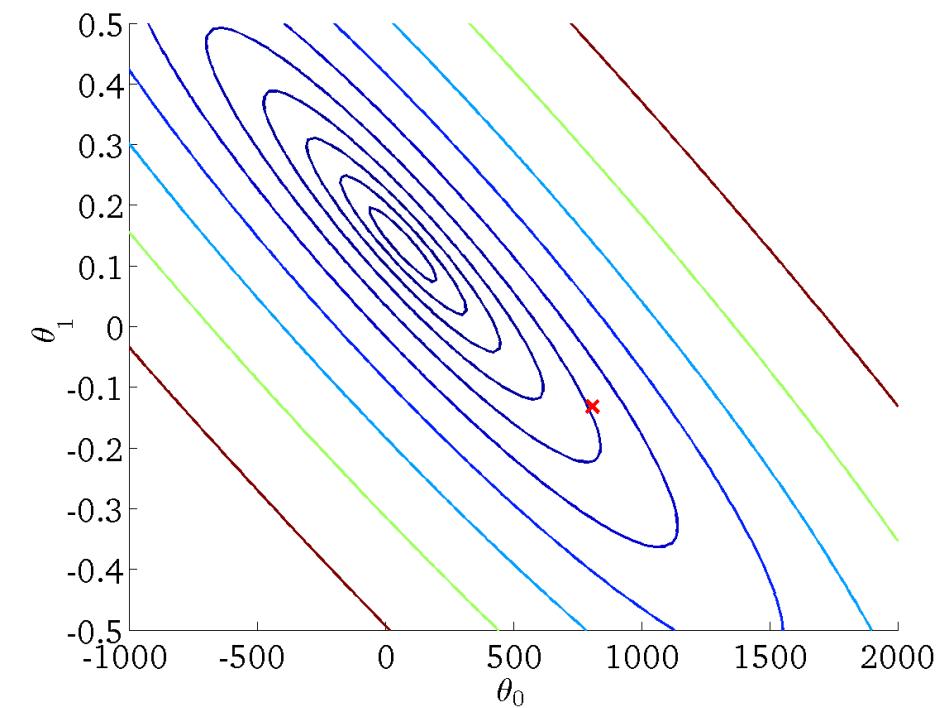
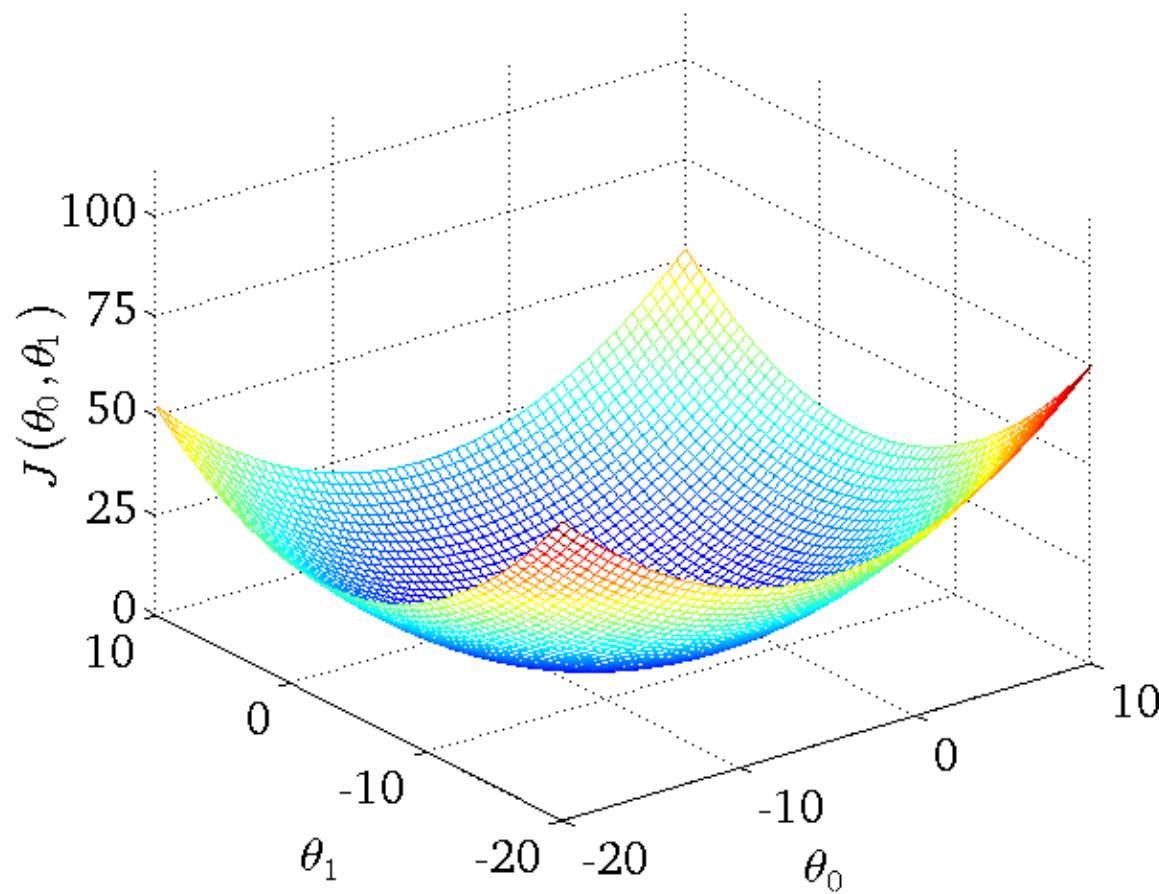
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$$

}

} update  
 $\theta_0$  and  $\theta_1$   
simultaneously



- An example of Cost Function  $J(\theta_0, \theta_1)$ :

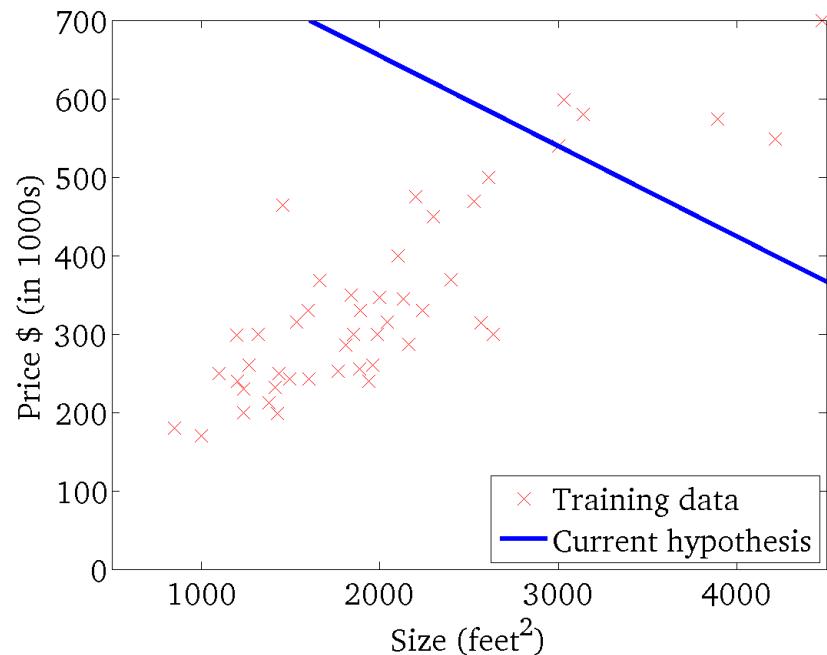


\* Reference: Andrew Ng, Machine Learning, Stanford University.



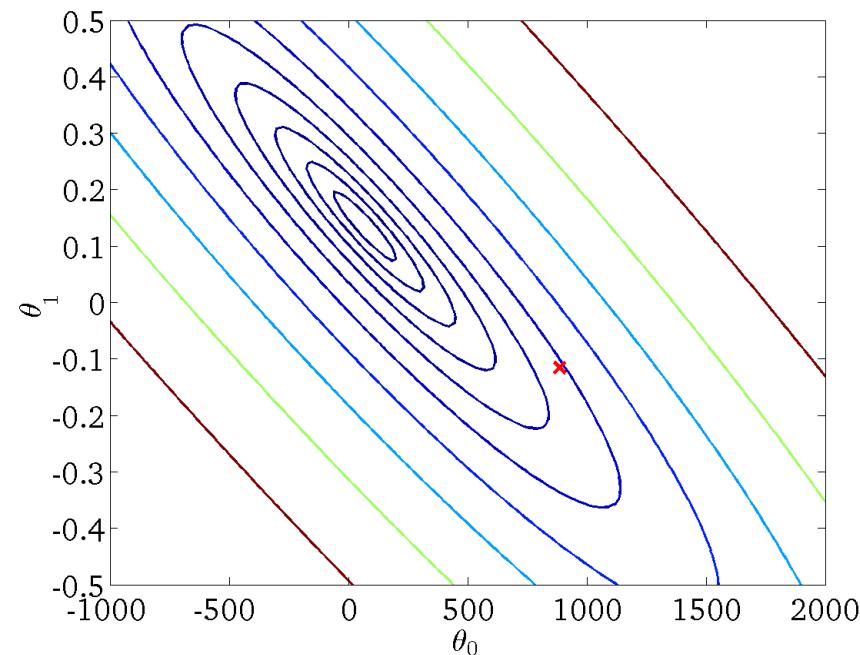
$$h_{\theta}(x)$$

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



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

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

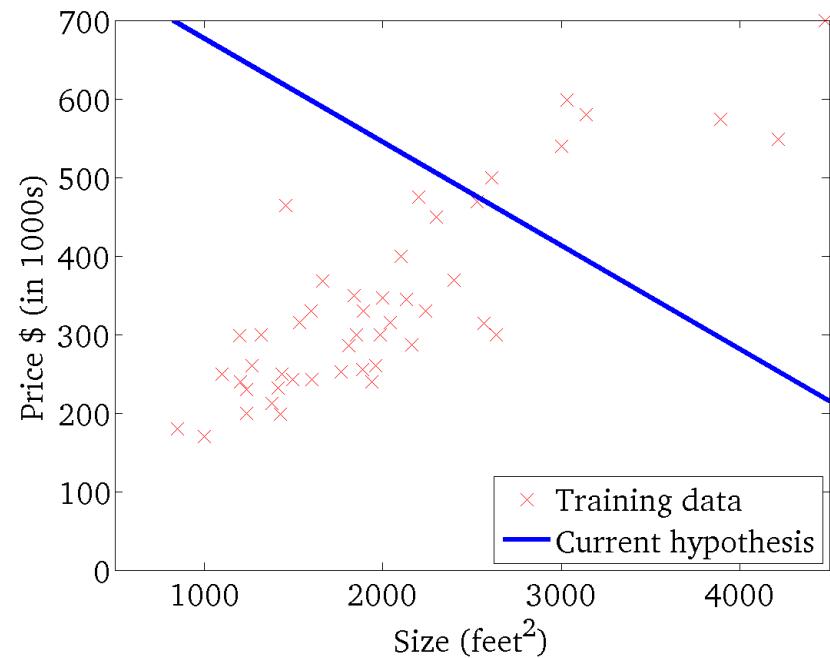


\* Example from Andrew Ng, Machine Learning, Stanford Univ.



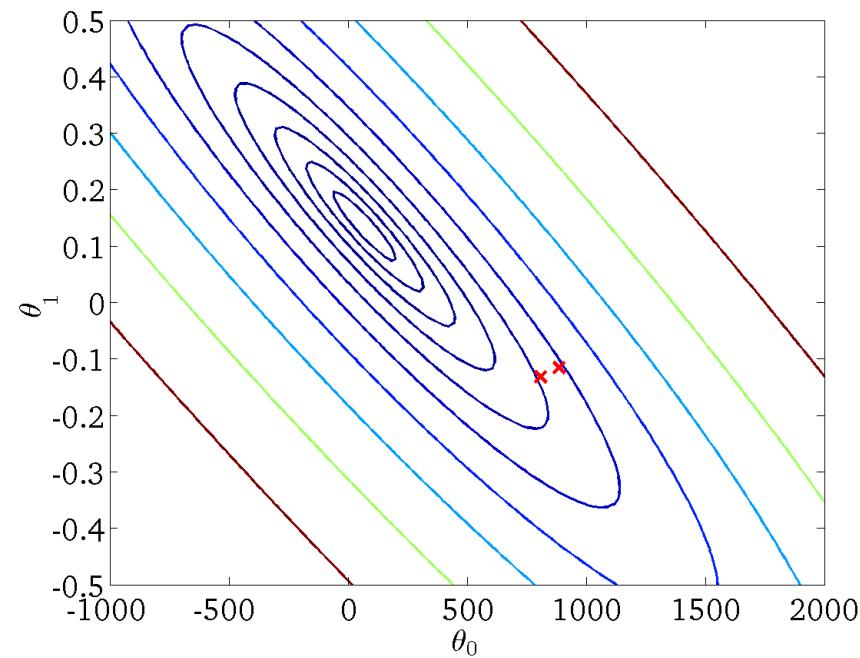
$$h_{\theta}(x)$$

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



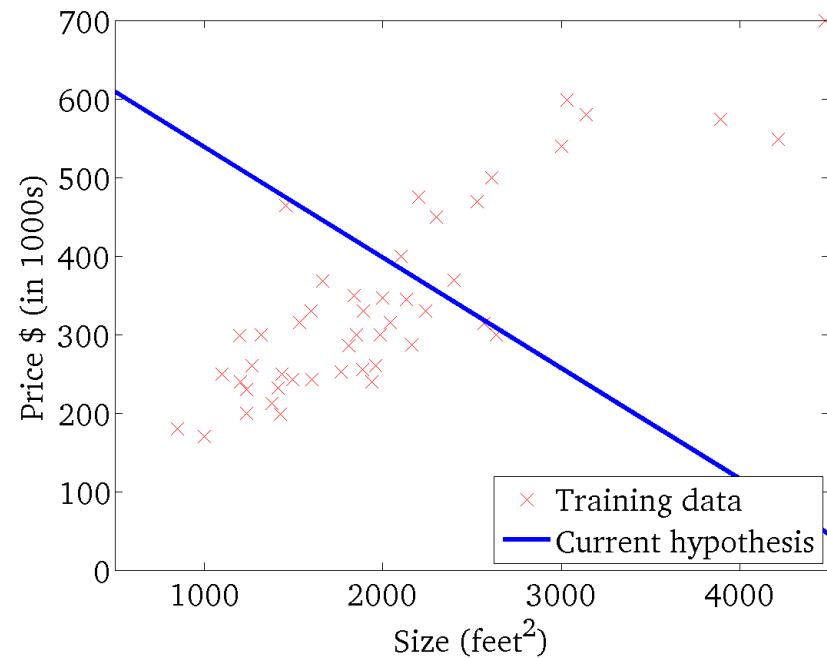
$$J(\theta_0, \theta_1)$$

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



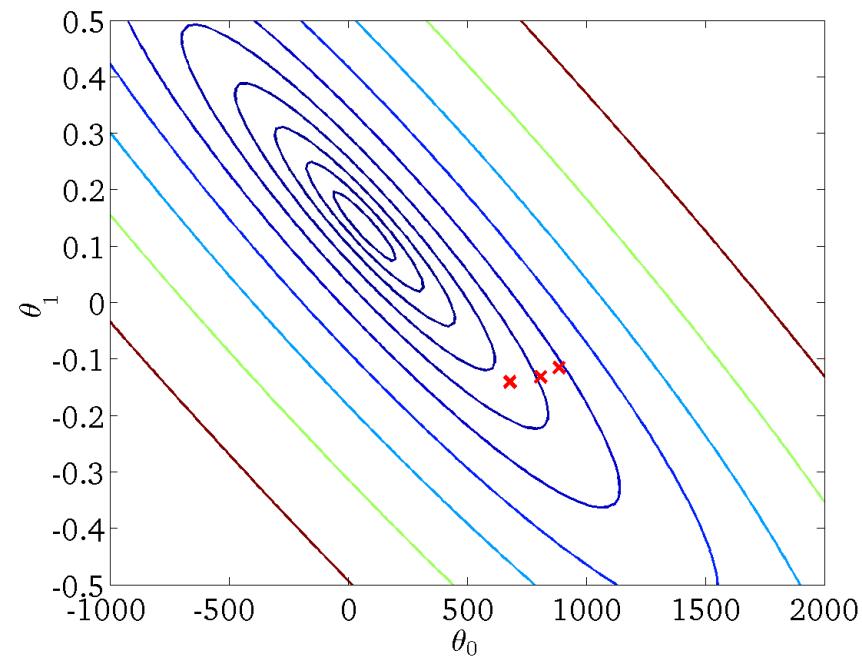
$$h_{\theta}(x)$$

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



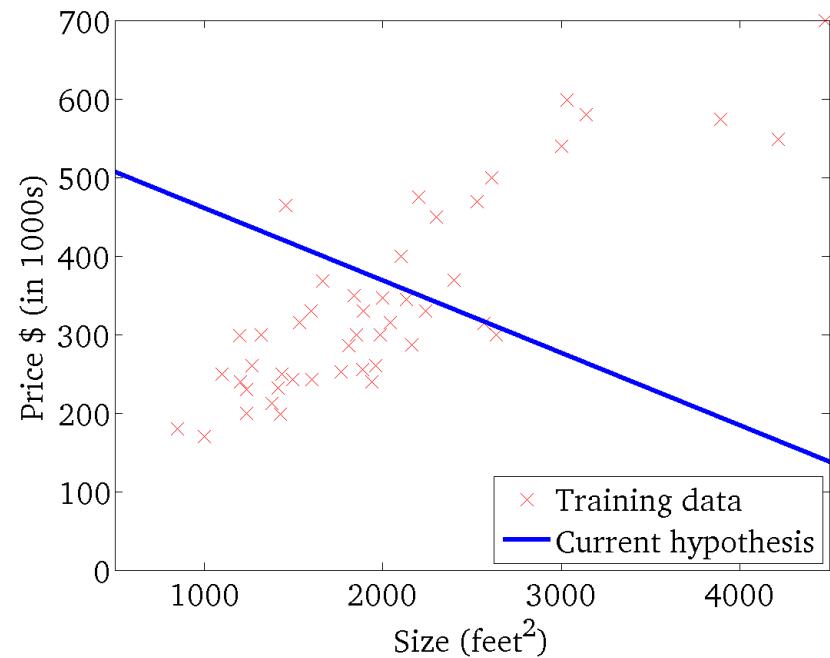
$$J(\theta_0, \theta_1)$$

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



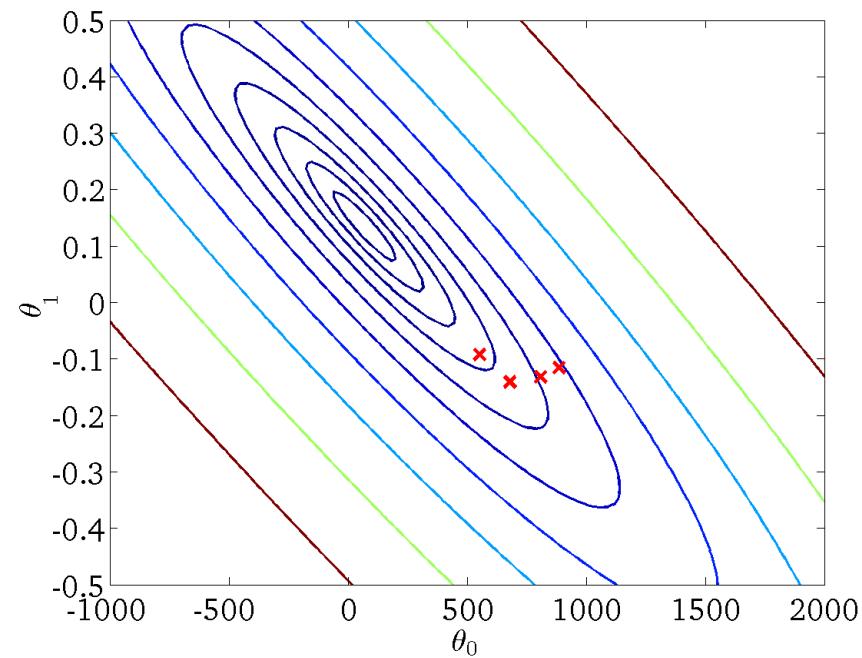
$$h_{\theta}(x)$$

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



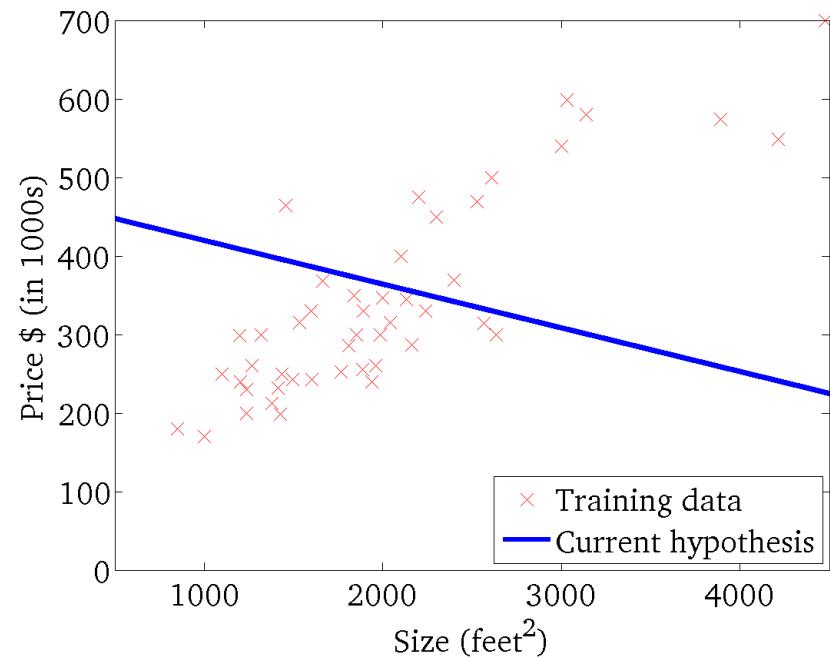
$$J(\theta_0, \theta_1)$$

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



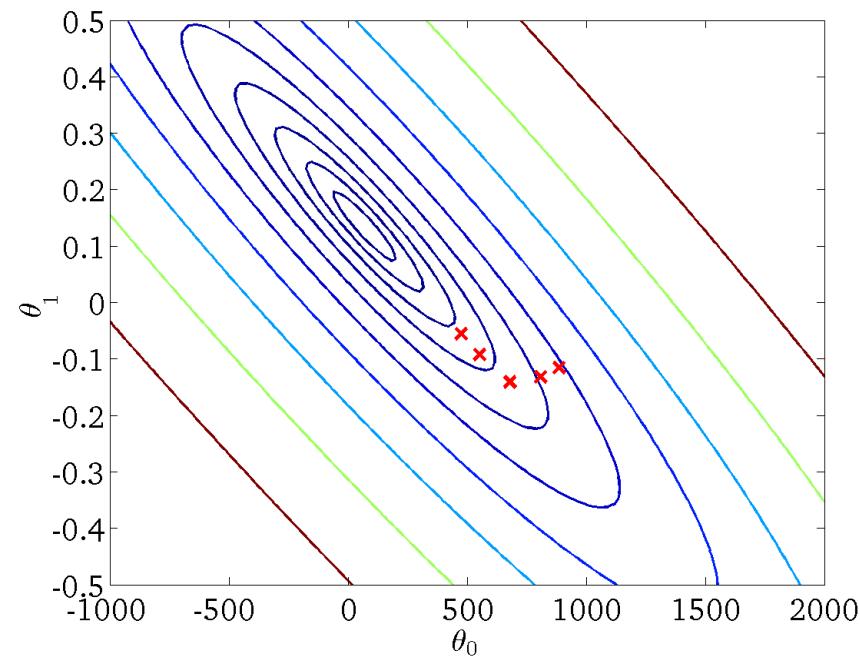
$$h_{\theta}(x)$$

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



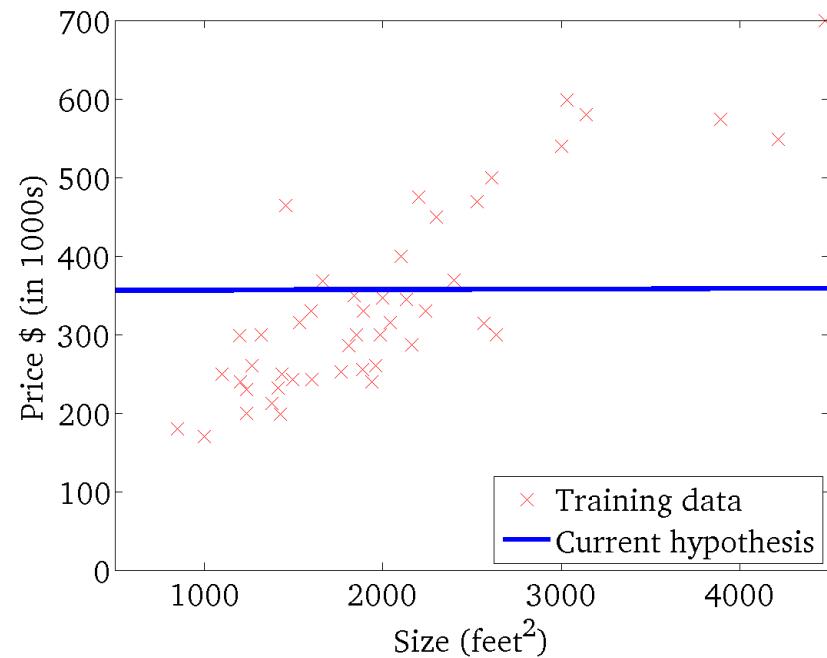
$$J(\theta_0, \theta_1)$$

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



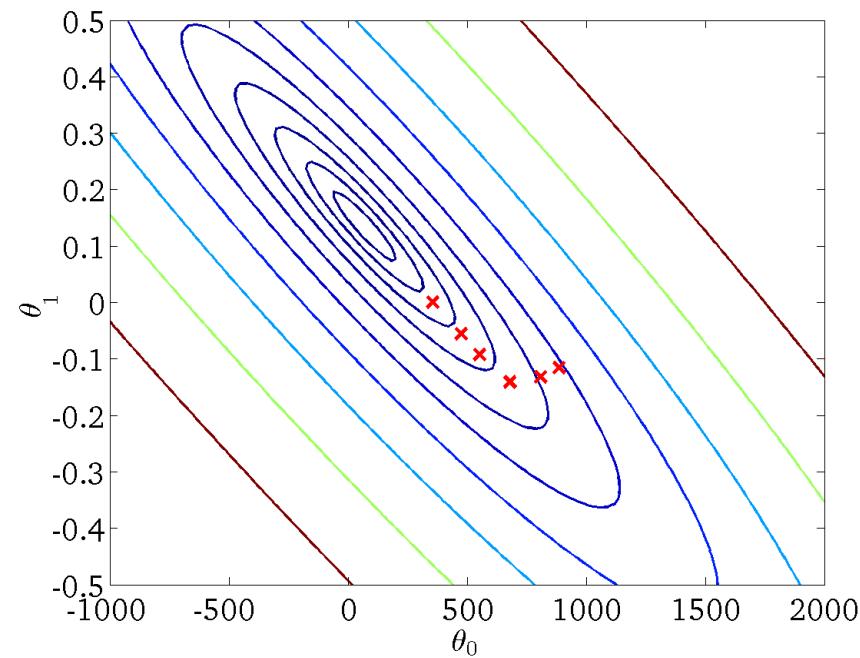
$$h_{\theta}(x)$$

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



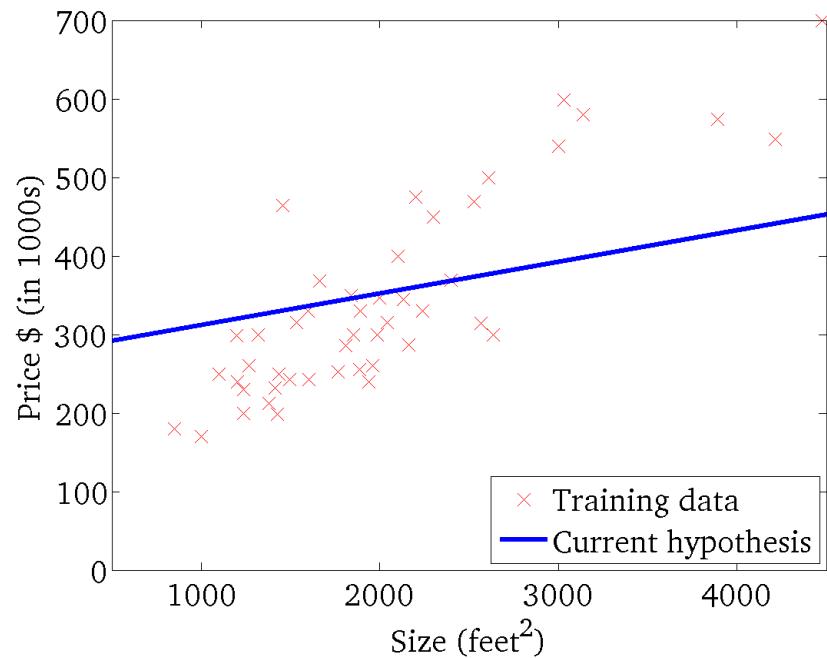
$$J(\theta_0, \theta_1)$$

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



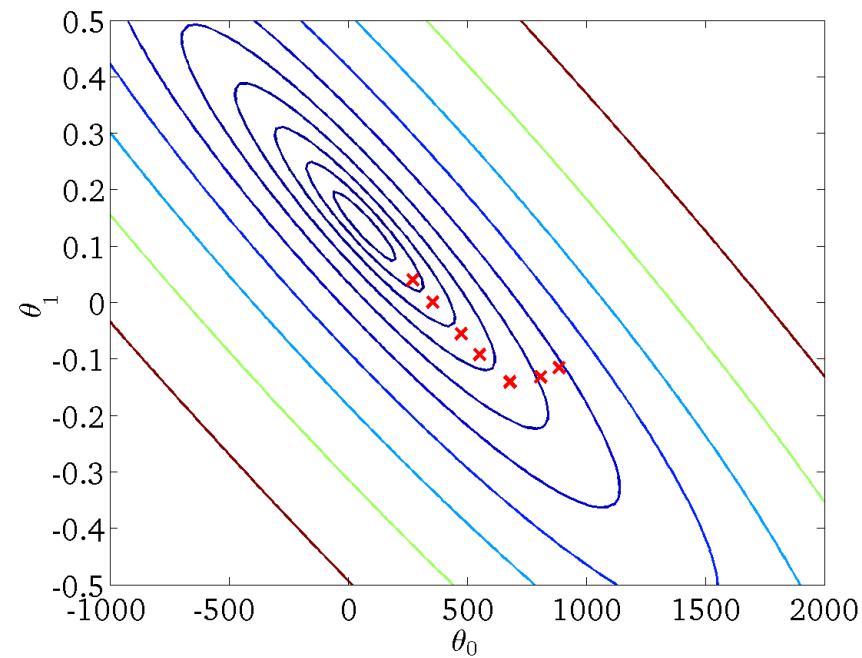
$$h_{\theta}(x)$$

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



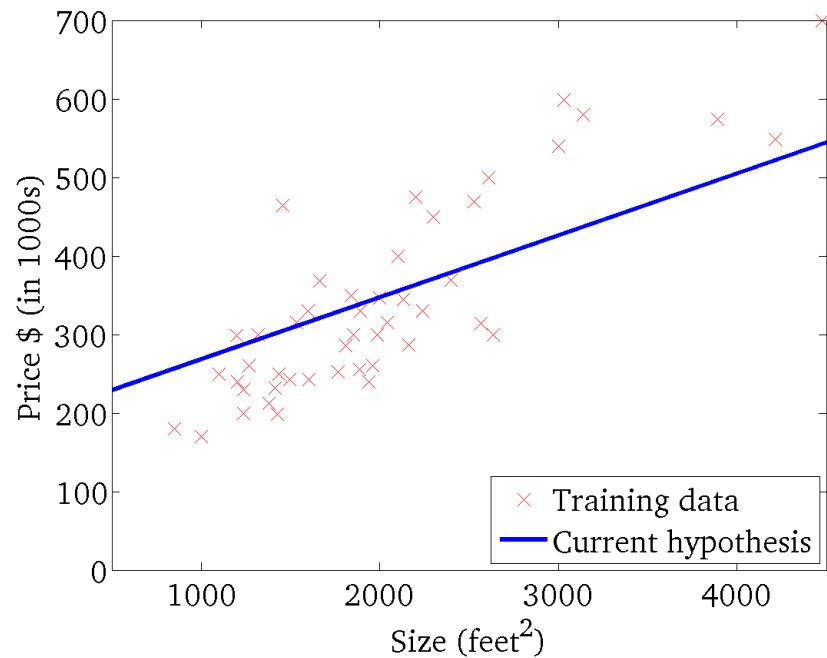
$$J(\theta_0, \theta_1)$$

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



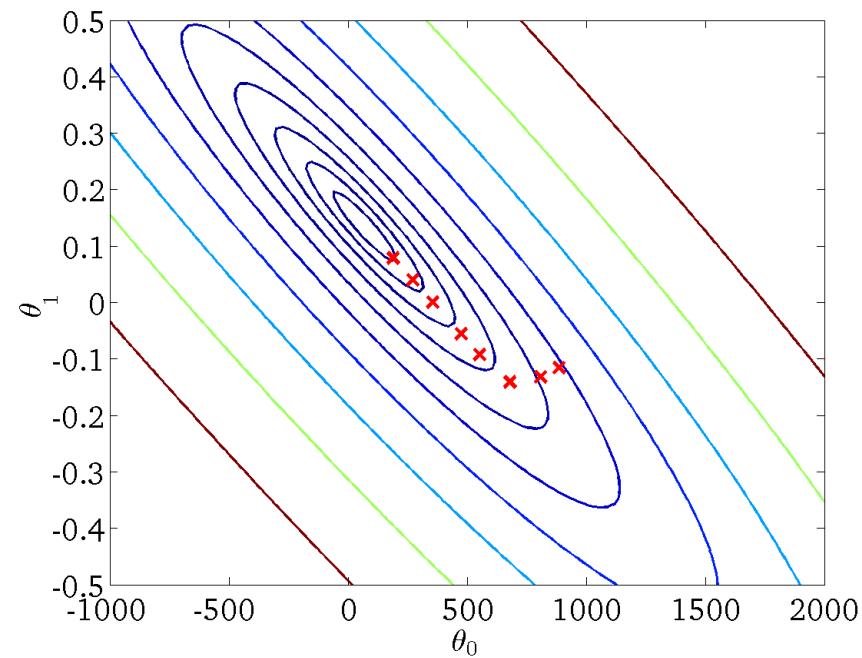
$$h_{\theta}(x)$$

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



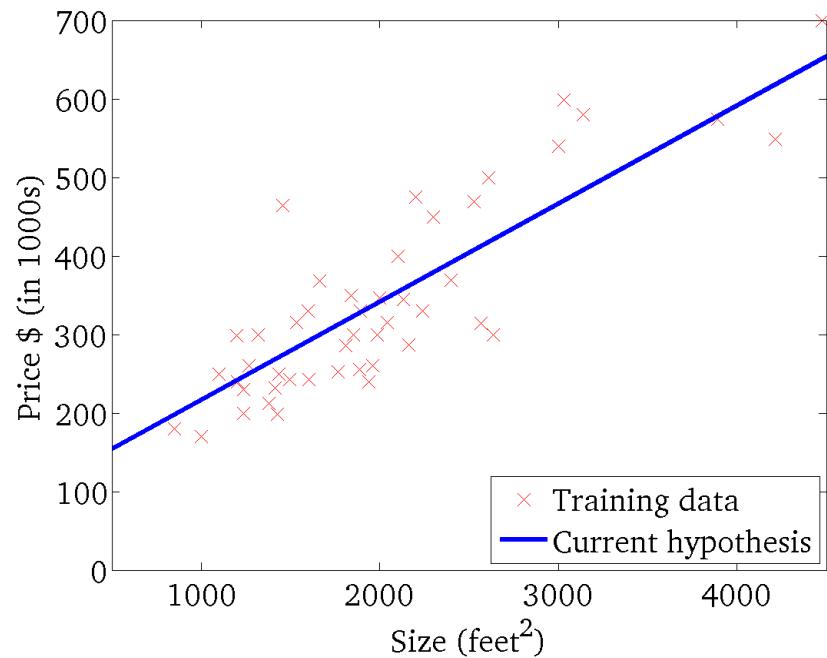
$$J(\theta_0, \theta_1)$$

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



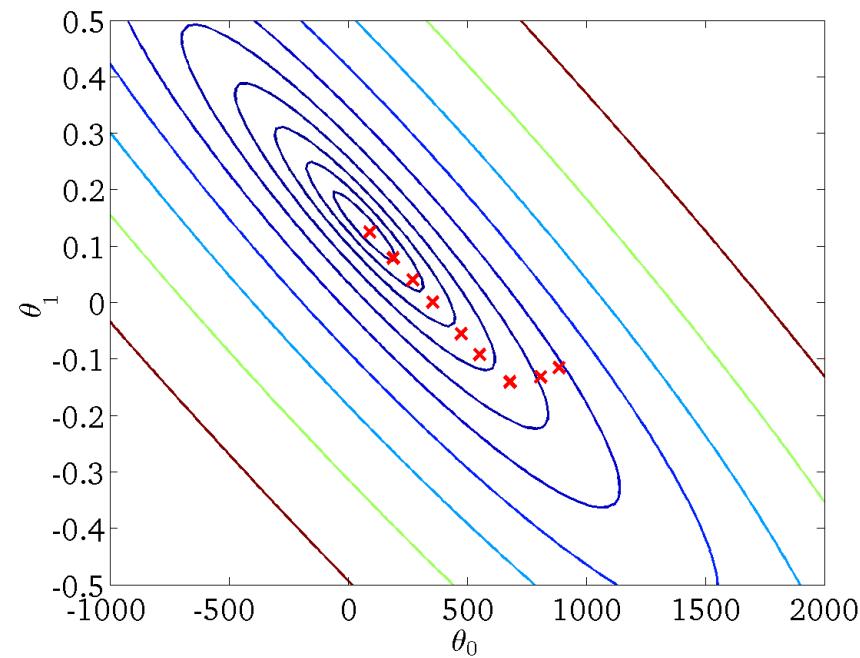
$$h_{\theta}(x)$$

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



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

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





# Linear Algebra Review: Matrices and Vectors

# Matrix

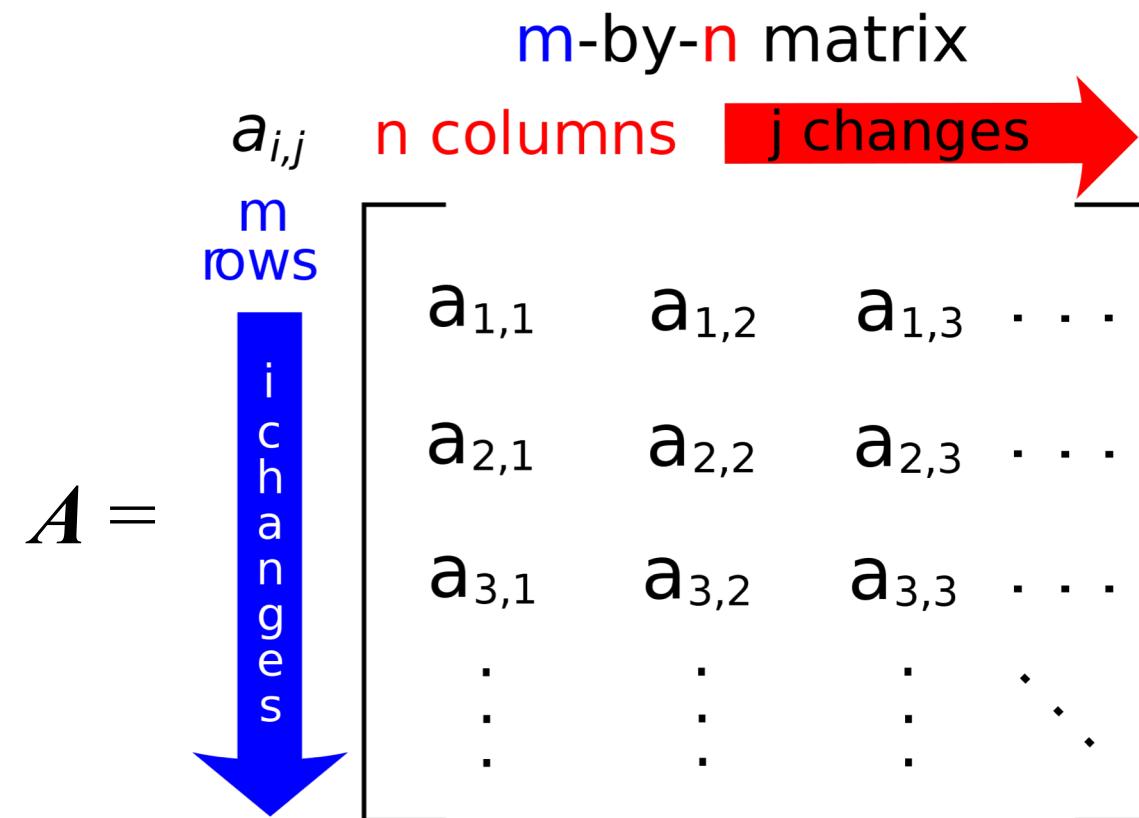
- **Matrix:** Matrix (plural matrices or matrixes) is a rectangular array (table) of numbers arranged in rows and columns.
- Dimension of matrix: (# of rows) x (# of columns)
- Example:  $M$  is 4x5

$$M = \begin{bmatrix} 2 & 3 & 6 & 78 & 0 \\ 0 & -4 & 23 & 44 & 4 \\ -34 & 4 & 78 & 8 & 2 \\ 7 & 4 & 5 & 0 & 0 \end{bmatrix}$$



# Matrix Elements

- $a_{i,j} = A[i,j]$ : element  $i,j$  in the  $i^{\text{th}}$  row and  $j^{\text{th}}$  column.



# Vector

- **Vector:** vector is a matrix with only one column ( $n \times 1$ ).  
Dimension of matrix: (# of rows)
- $a_i = v[i]$ : element  $i$  in the  $i^{\text{th}}$  row.
- **Note:** be careful of the indexing standards. In most of programming languages, the vector index starts from 0.

$$v = \begin{bmatrix} 4 \\ 12 \\ -5 \\ 7 \end{bmatrix}$$



# Notation

- We usually use a **bold** CAPITAL *Italic* letter to name a matrix.
  - E.g.  $M$ ,  $A$ , ...
- We usually use a **bold** lower-case *Italic* letter to name a vector.
  - E.g.  $v$ ,  $u$ , ...
- We usually use a non-bold *Italic* letter to name a scalar (variable).
  - E.g.  $s$ ,  $c$ , ...





# Linear Regression with multiple features

# Linear Regression with One Feature

One Feature      Target

Size in feet <sup>2</sup> (x)	Price (y)
1000	410K
1200	600K
1230	620K
1340	645K
...	...



# Linear Regression with One Feature

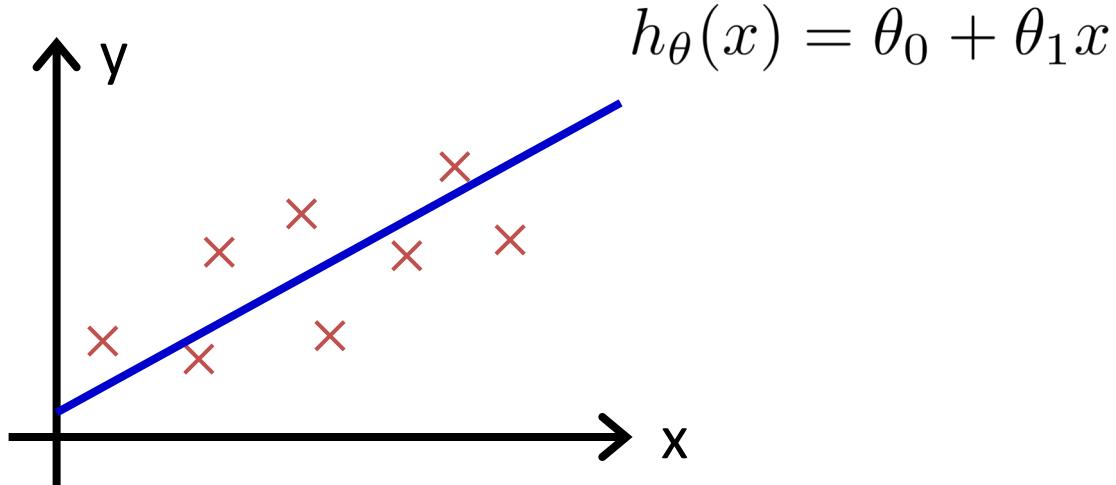
One Feature      ↴      Target

Size in feet <sup>2</sup> (x)	Price (y)
1000	410K
1200	600K
1230	620K
1340	645K
...	...

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



# Linear Regression with One Feature



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

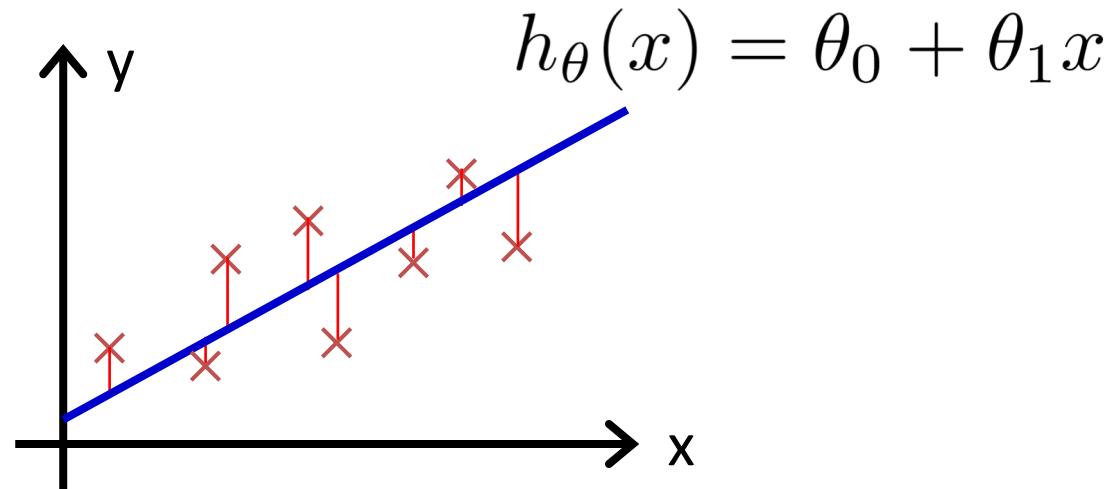


# How to Select the Parameters

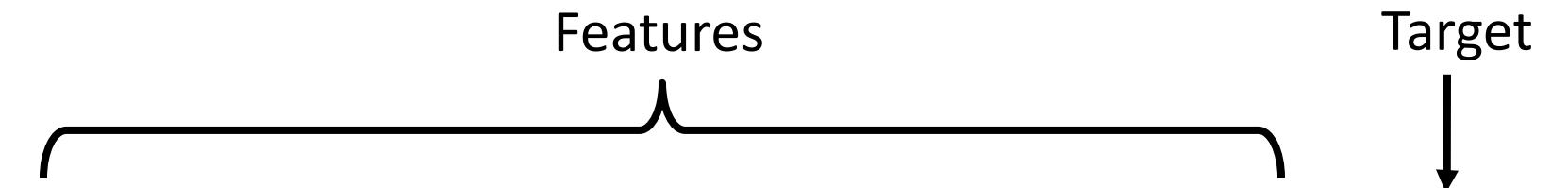
**Training Set:**  $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(m)}, y^{(m)})\}$

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

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



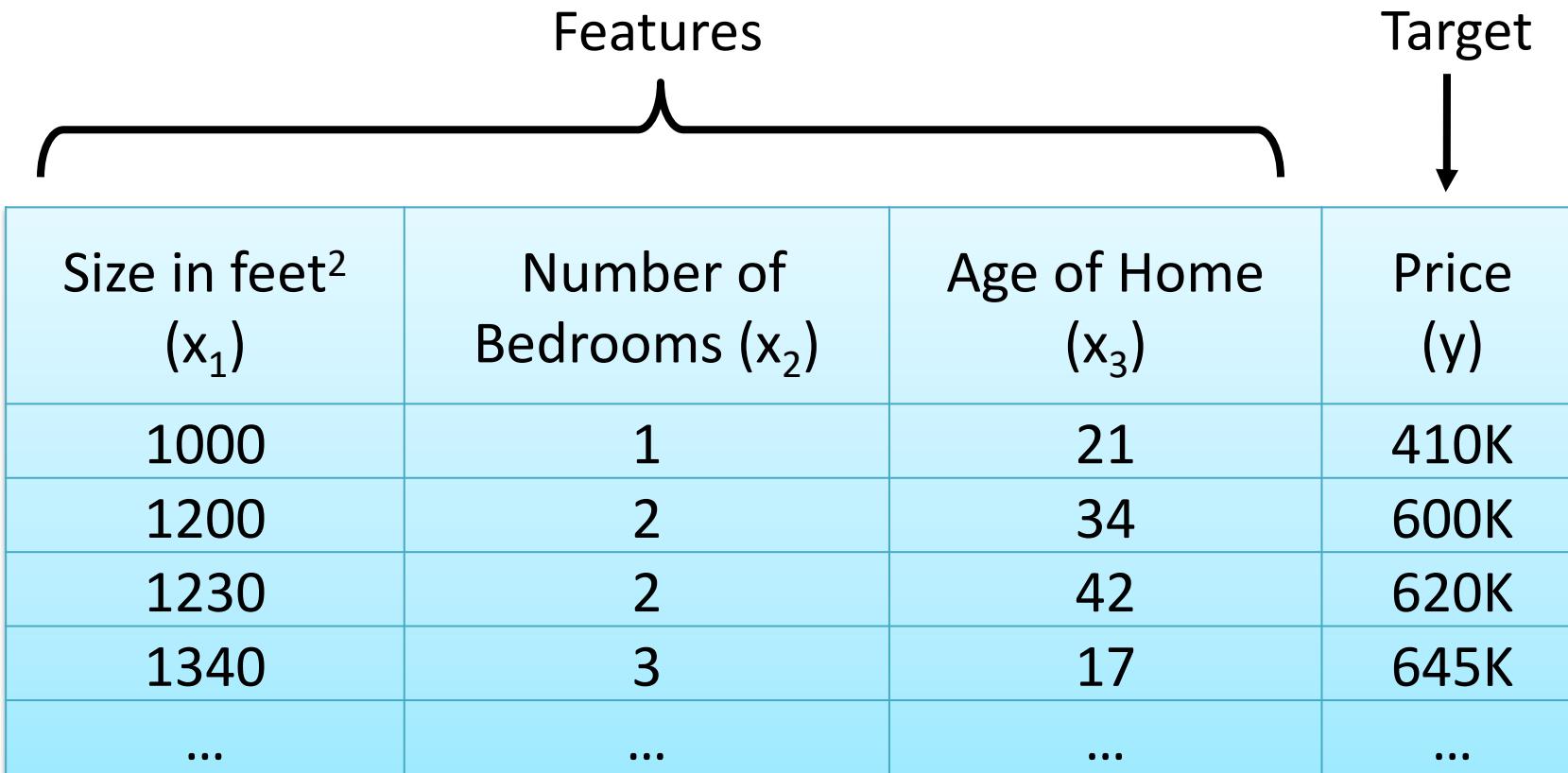
# Linear Regression with Multiple Features



Size in feet <sup>2</sup> ( $x_1$ )	Number of Bedrooms ( $x_2$ )	Age of Home ( $x_3$ )	Price ( $y$ )
1000	1	21	410K
1200	2	34	600K
1230	2	42	620K
1340	3	17	645K
...	...	...	...



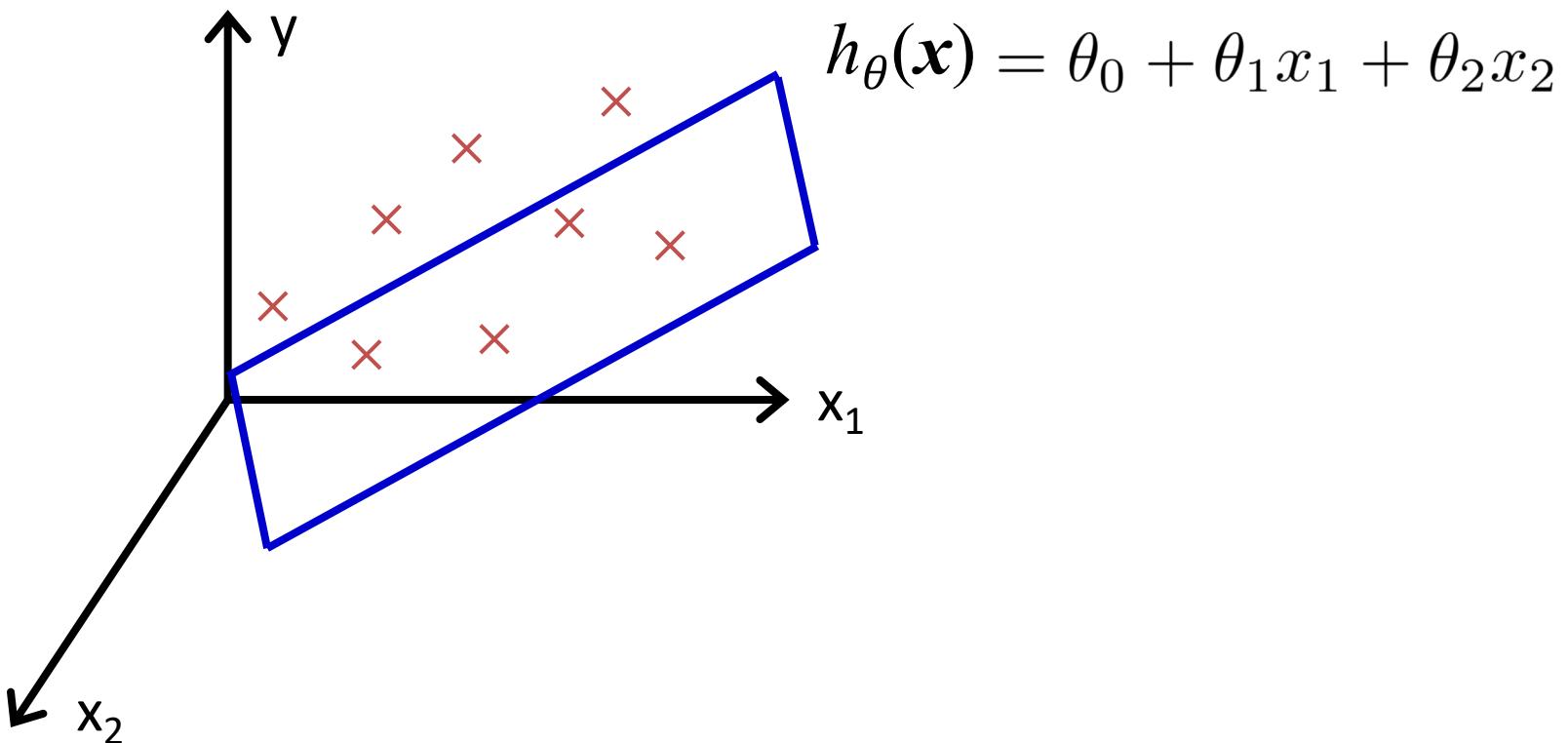
# Linear Regression with Multiple Features



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



# Example: Two Features



# Linear Regression with Multiple Features

Size in feet <sup>2</sup> ( $x_1$ )	Number of Bedrooms ( $x_2$ )	Age of Home ( $x_3$ )	Price (y)
1000	1	21	410K
1200	2	34	600K
1230	2	42	620K
1340	3	17	645K
...	...	...	...

## Notation:

$n$  = number of features

$m$  = number of training samples

$\mathbf{x}^{(i)}$  = feature vector for  $i^{th}$  training samples (the entire  $i^{th}$  row).

$x_j^{(i)}$  = value of feature  $j$  in  $i^{th}$  training samples (element  $j$  in  $i^{th}$  row) .



# Linear Regression with Multiple Features

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

For simplicity of notation, we define  $x_0 = 1$  (add a dummy feature with a constant value for all data samples). Then, we can define:

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} ; \quad x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix} \quad \rightarrow \quad h_{\theta}(x) = \theta^T x$$



# Gradient Descent for Multiple Variables\*

- Regression Model:

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

- Parameter vector & feature vectors:

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \quad x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$

- Cost function:

$$J(\theta) = J(\theta_0, \theta_1, \theta_2, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

\* Reference: Andrew Ng, Machine Learning, Stanford University.



# Gradient Descent for Multiple Variables

- Hypothesis:  $h_{\theta}(x) = \theta^T x = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$
- Cost function:  $J(\theta) = J(\theta_0, \theta_1, \theta_2, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$
- **Gradient descent:**

Repeat {

$$\begin{aligned}\theta_j &:= \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \dots, \theta_n) \\ &= \theta_j - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_j^{(i)}\end{aligned}$$

}

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



# Gradient Descent for Multiple Features

Previously (n=1):

Repeat {

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

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

(simultaneously update  $\theta_0, \theta_1$ )

}

New algorithm ( $n \geq 1$ ):

Repeat {

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

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

}

---

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

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

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

...





*Thank You!*

**Questions?**