Introduction to Data Science

(Lecture 11)

Dr. Mohammad Pourhomayoun

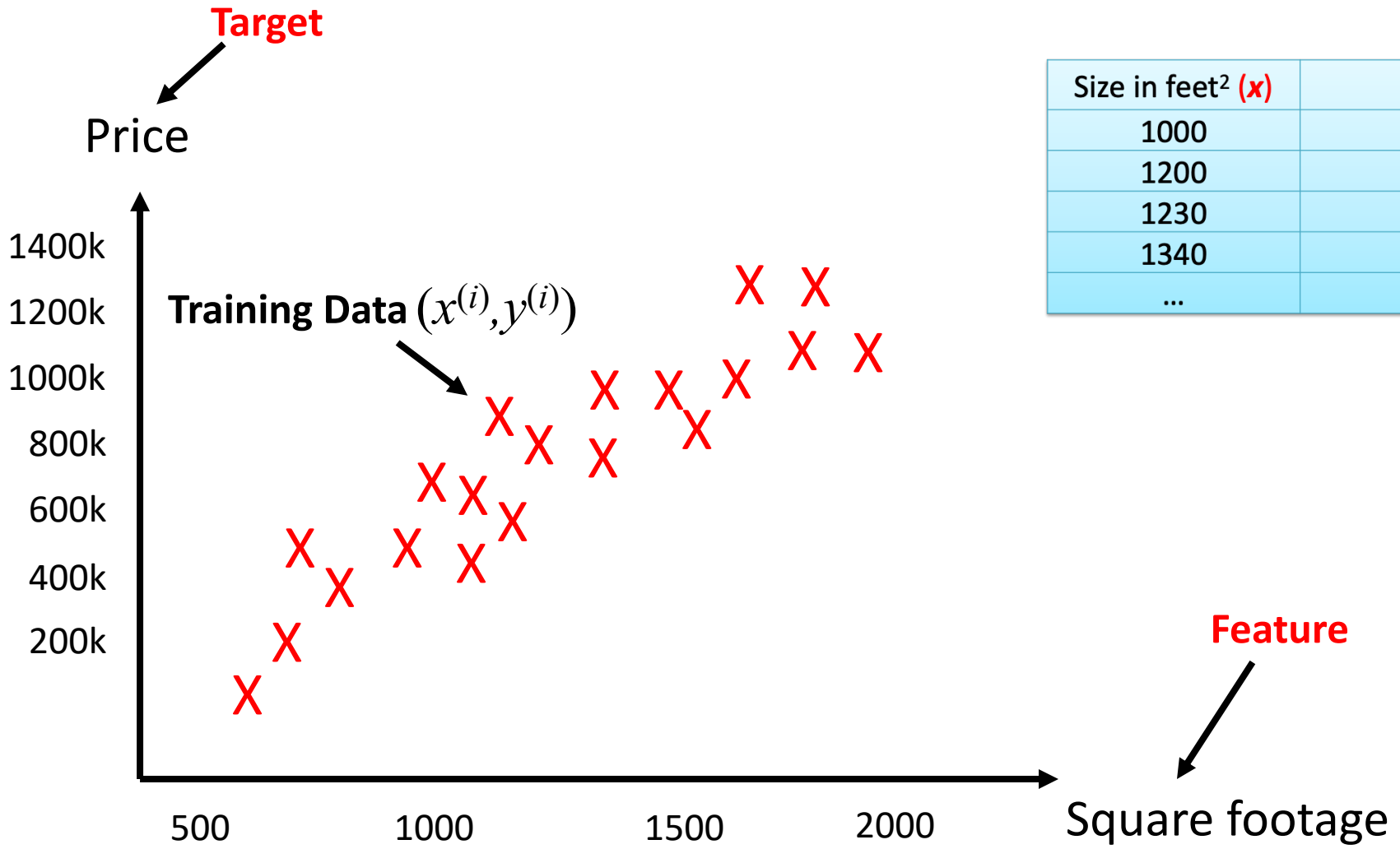
Assistant Professor

Computer Science Department

California State University, Los Angeles

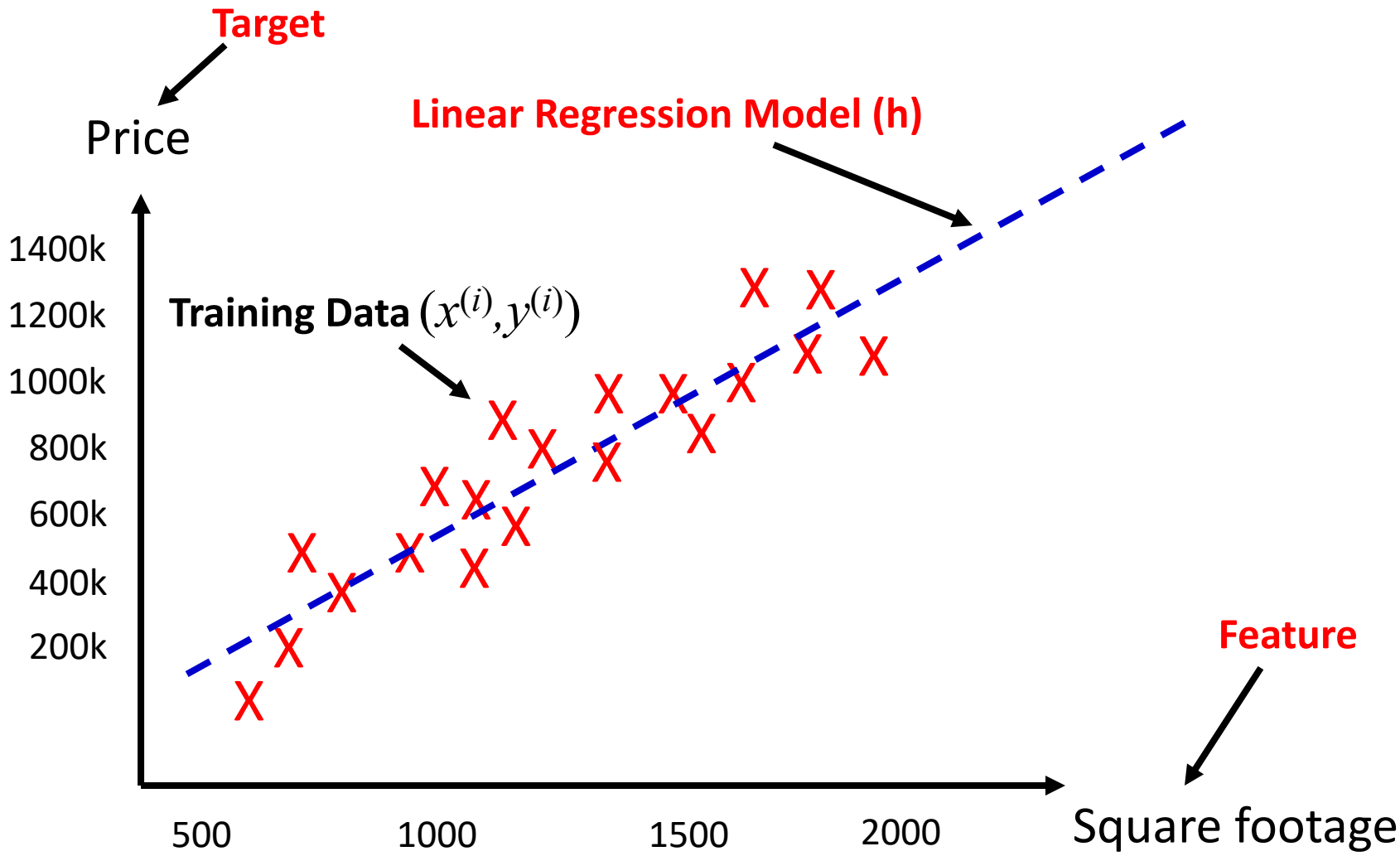
Continue: Gradient Descent for Linear Regression

Review: Regression Example: Housing Price

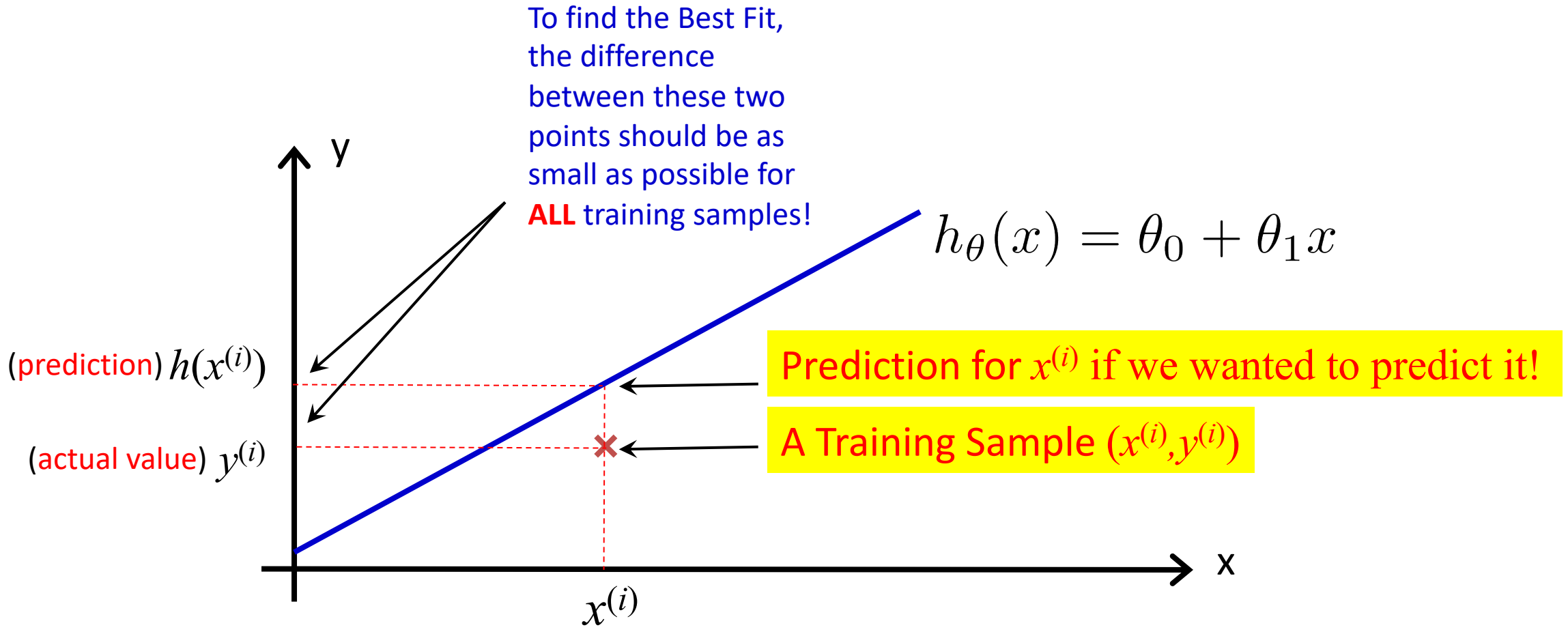


Size in feet² (*x*)	Price (*y*)
1000	410K
1200	600K
1230	620K
1340	645K
…	…

Regression Example: Housing Price



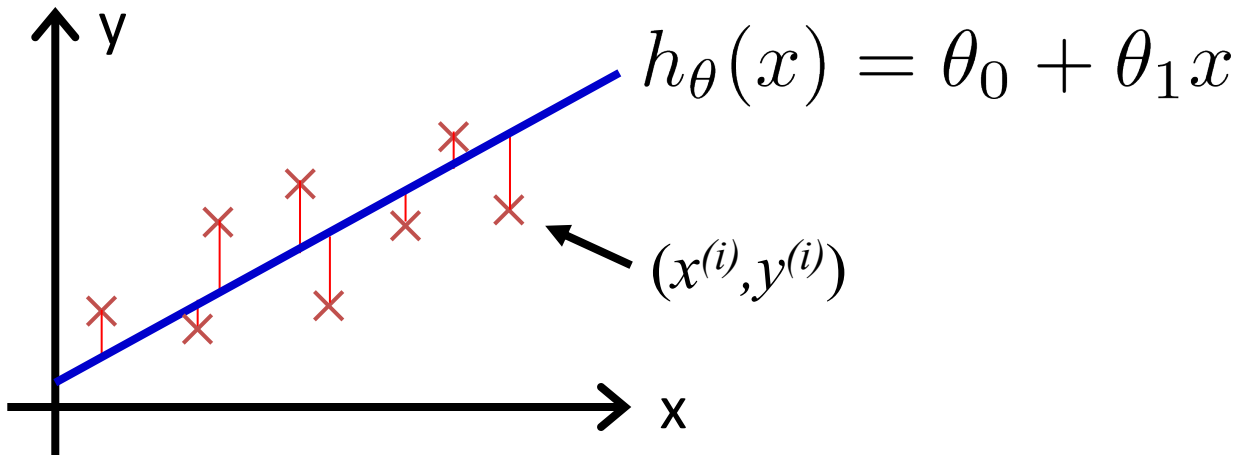
- Thus, we just need to minimize $(h(x^{(i)}) - y^{(i)})^2$ for **every** training sample *i*.
- *Question: Why squared?*



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

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

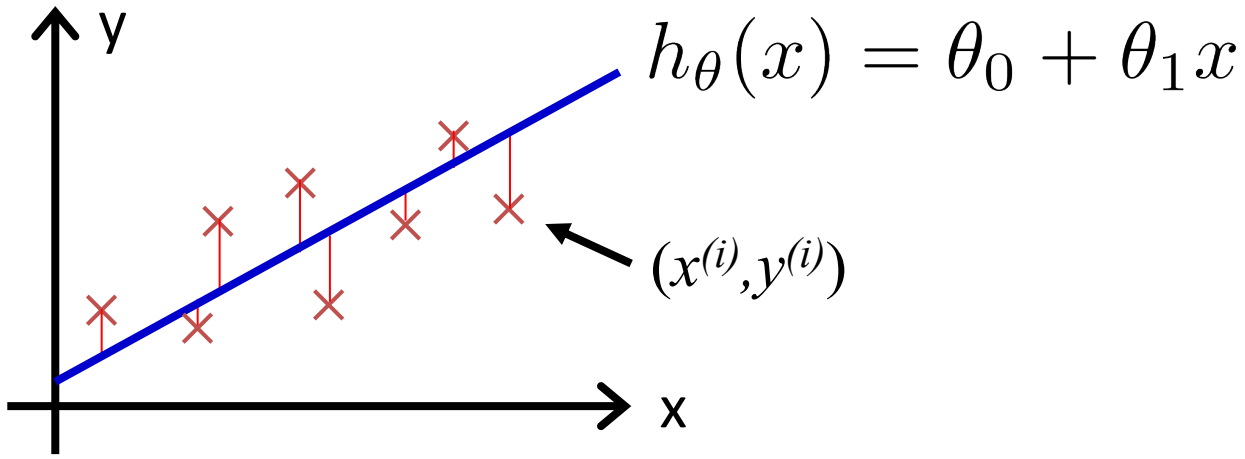


Training Set: $\{(x^{(1)},y^{(1)}), (x^{(2)},y^{(2)}), \ldots, (x^{(m)},y^{(m)})\}$

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

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

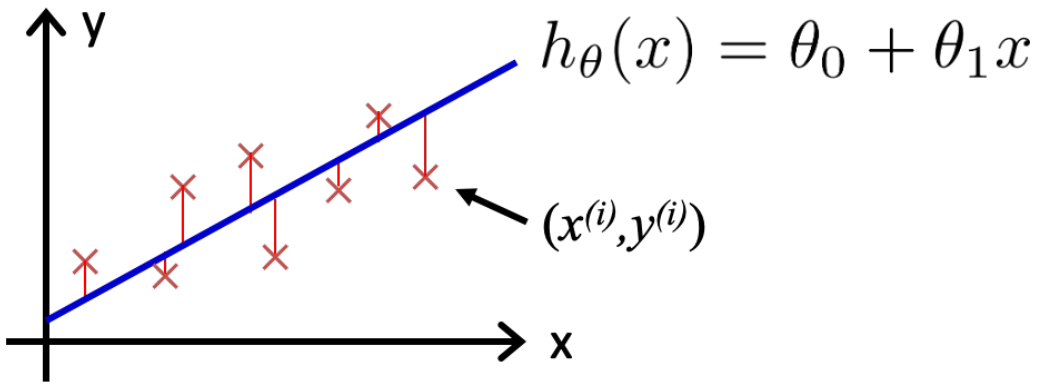
$$\underset{\theta_0,\theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$$

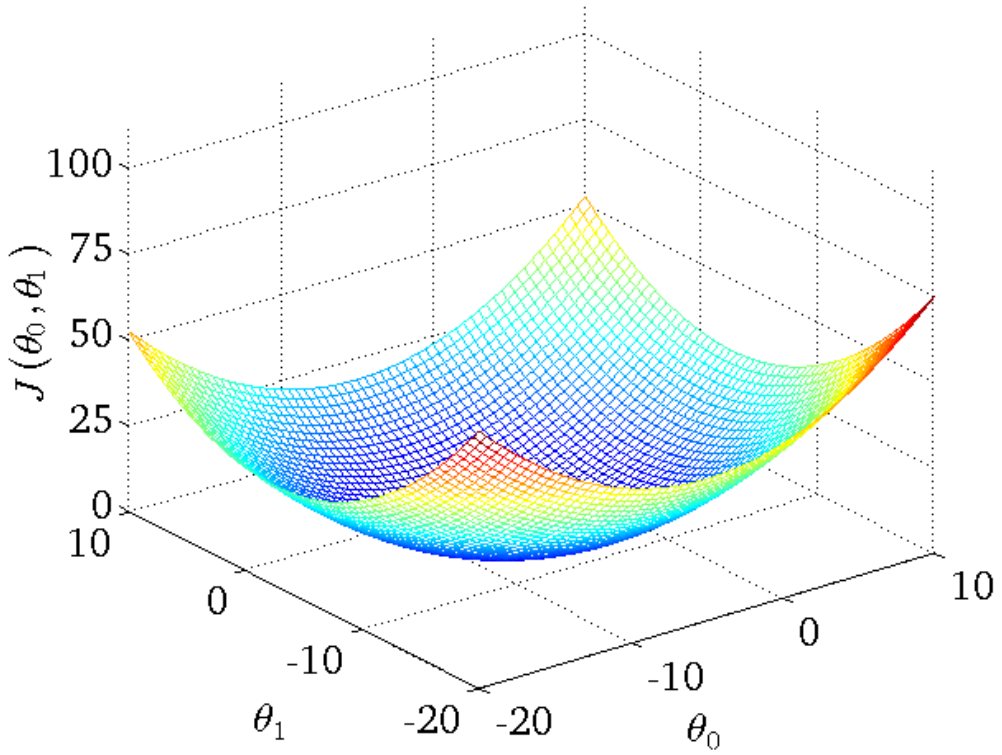


Regression Model: $h_\theta(x) = \theta_0 + \theta_1 x$

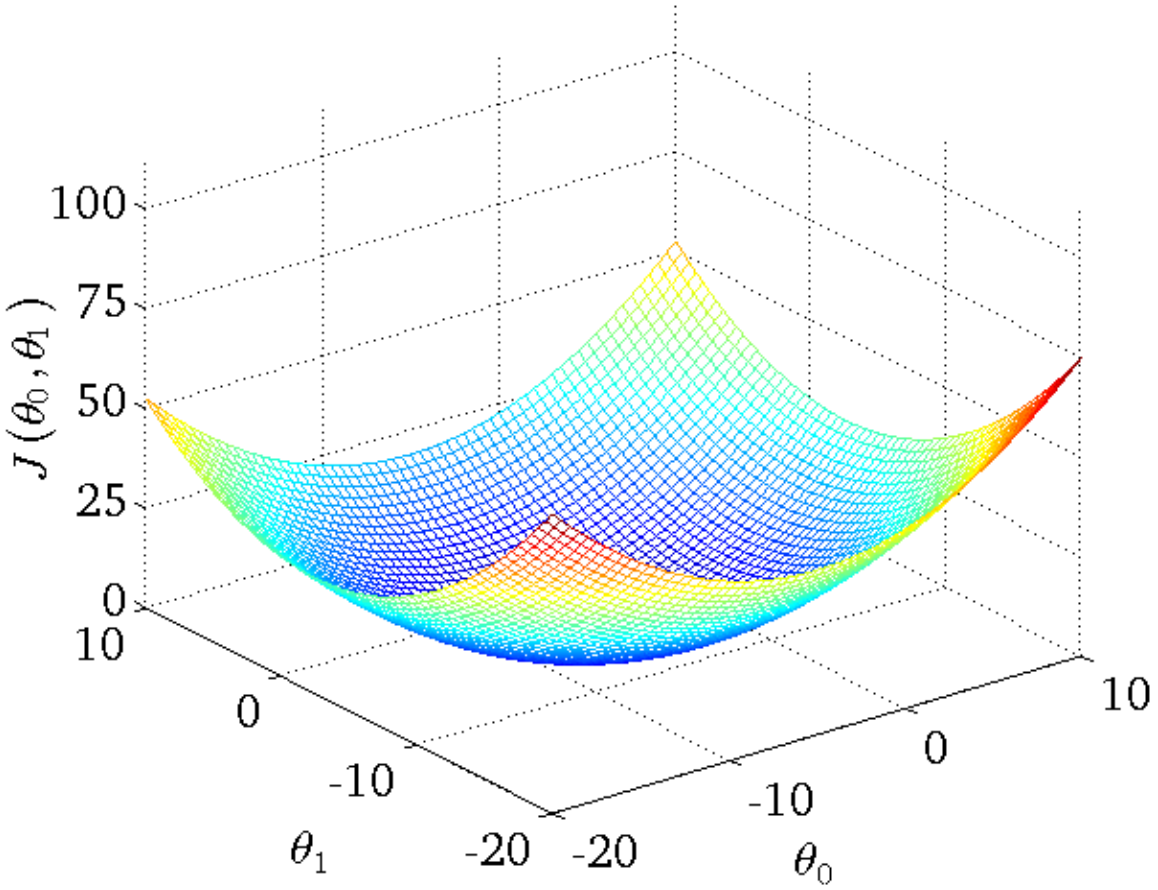
Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$

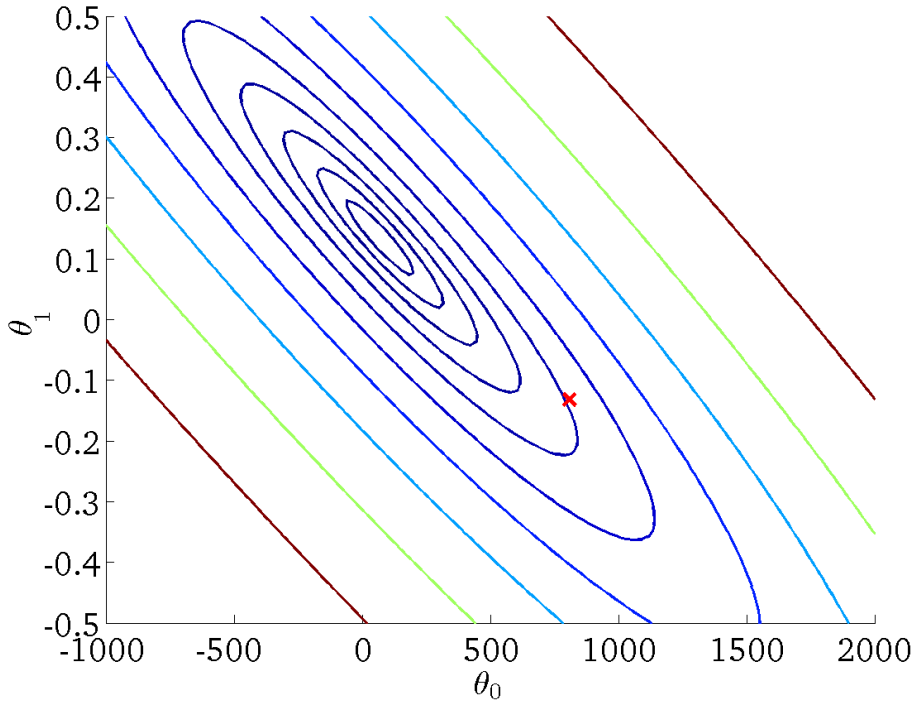
Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$





- 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:

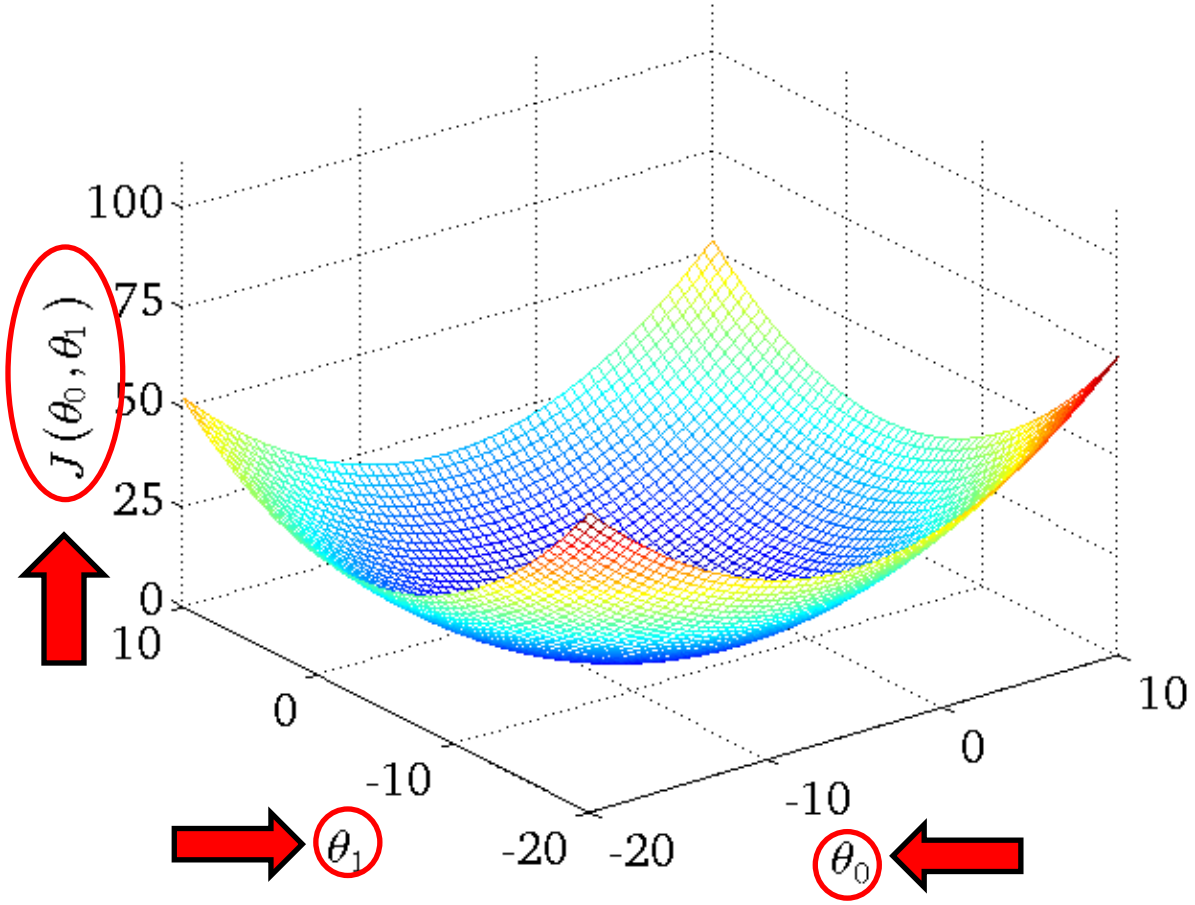


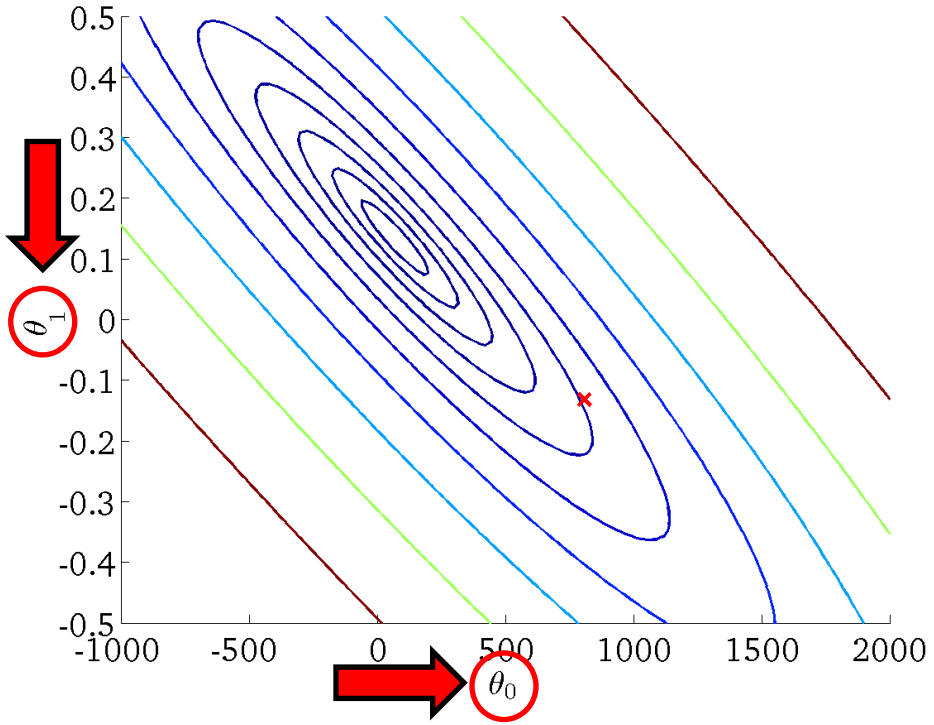


Contour: The projection of the J surface on the $(\theta_0 - \theta_1)$ plane (on the floor)

* 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.

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

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

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

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

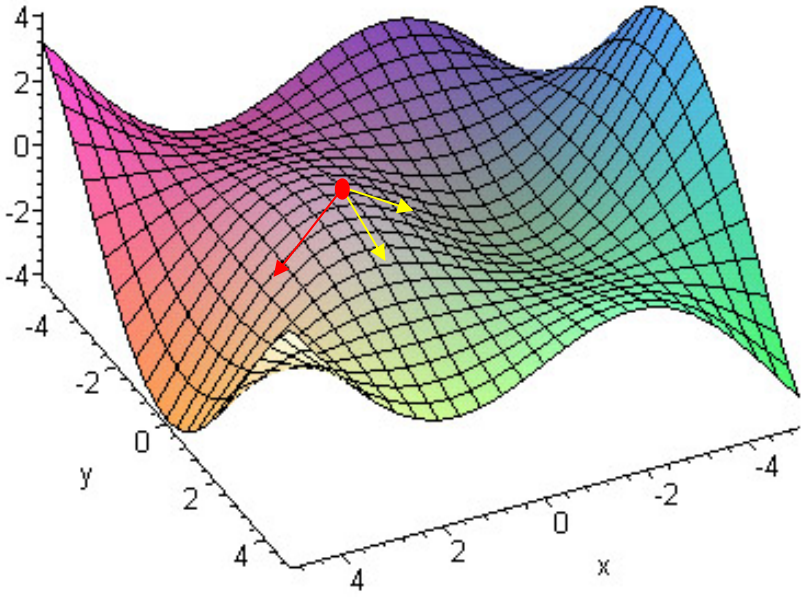
Gradient Descent Algorithm

- 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 it that direction.



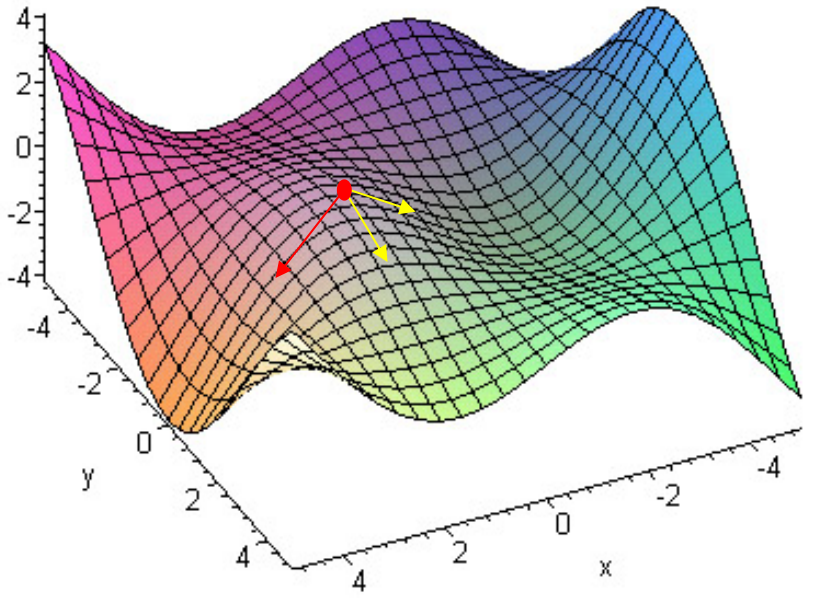
* Figure Ref: Andrew Ng, Machine Learning, Stanford University.

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.

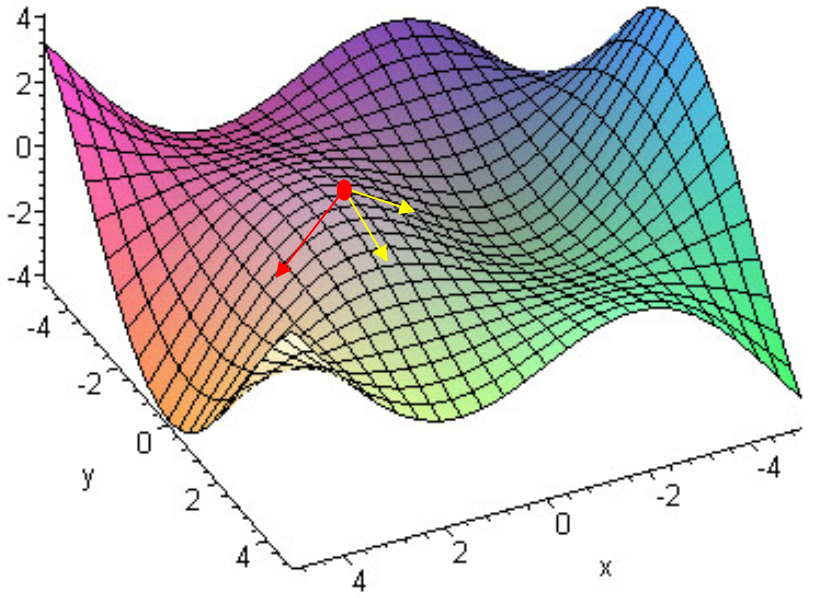
- In other word, a differentiable function ***F*** **decreases fastest** if one moves in the direction of the **negative gradient** of ***F*** at point ***x***.



Gradient Descent Algorithm

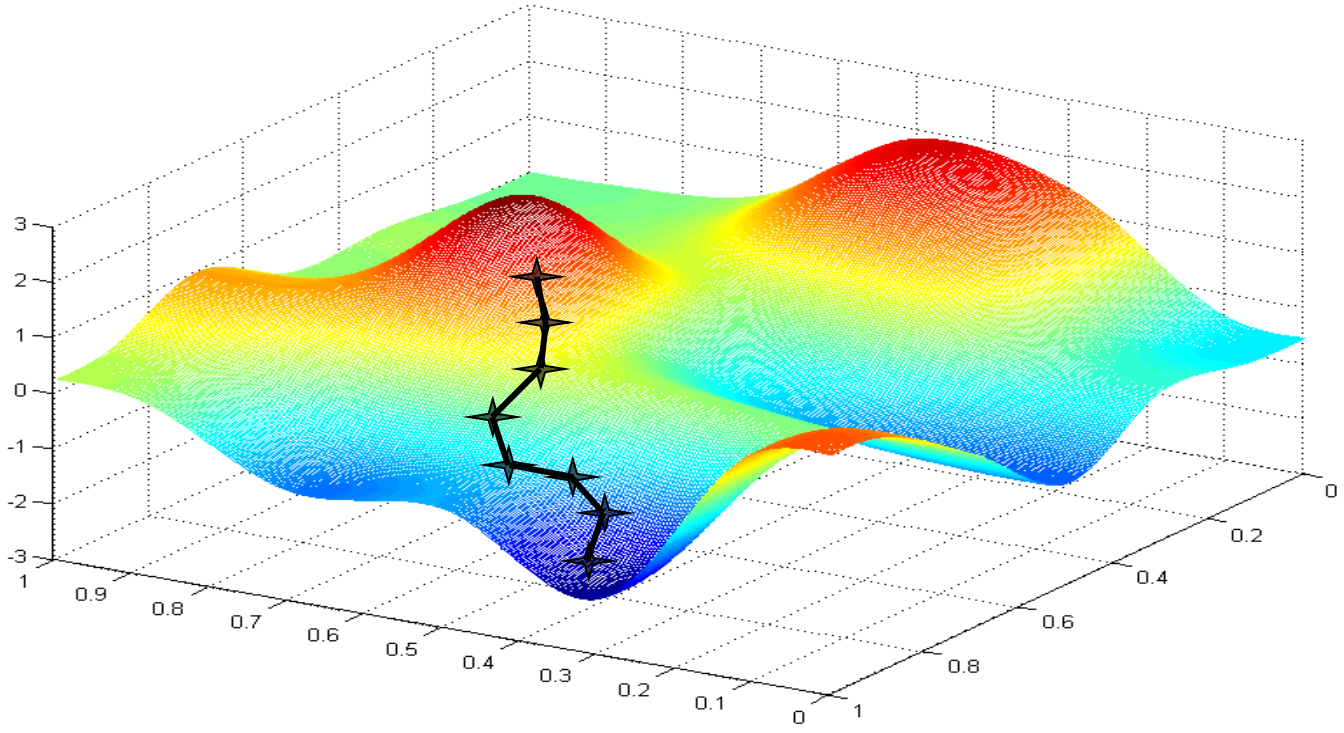
- Gradient Descent is an iterative optimization approach that tries to minimize a function.

- 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).



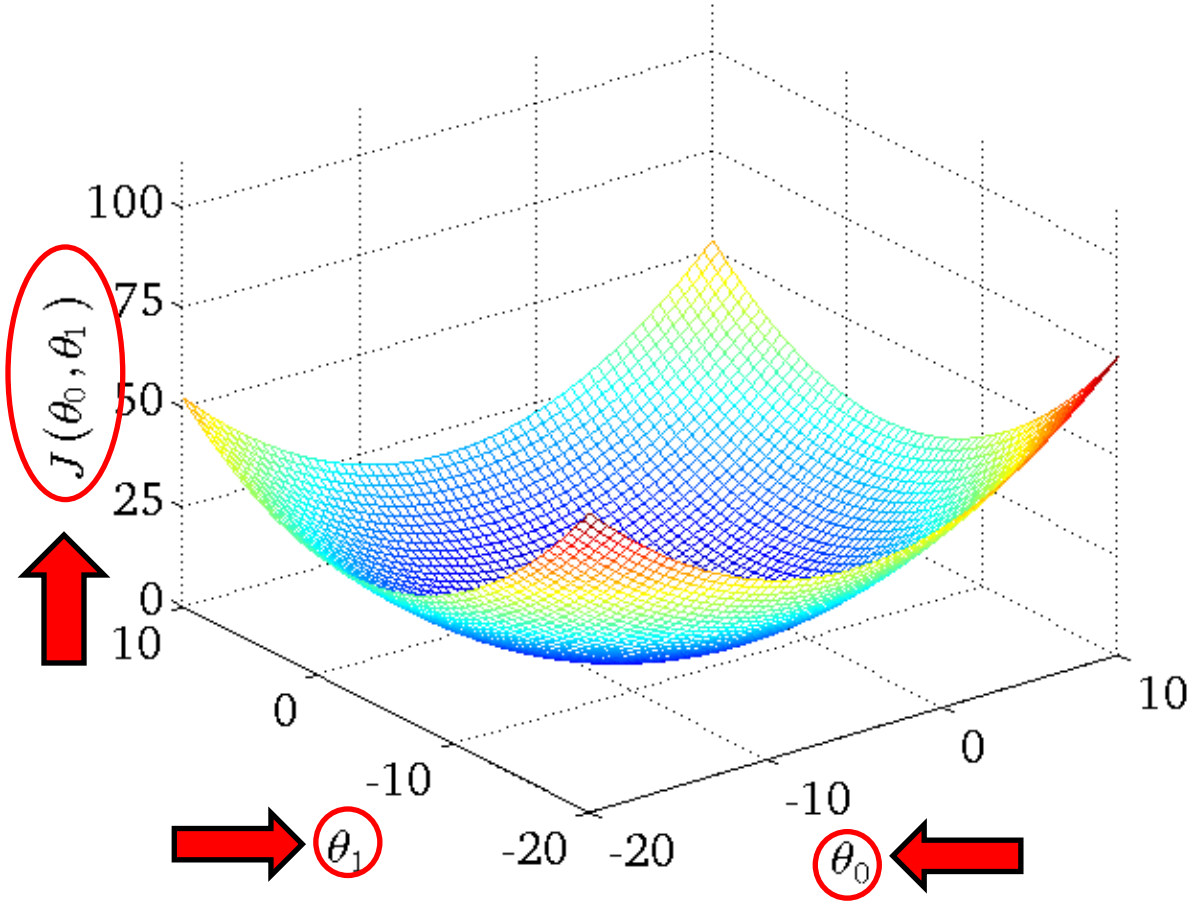
- Coming down the mountain on the most efficient and fastest path!

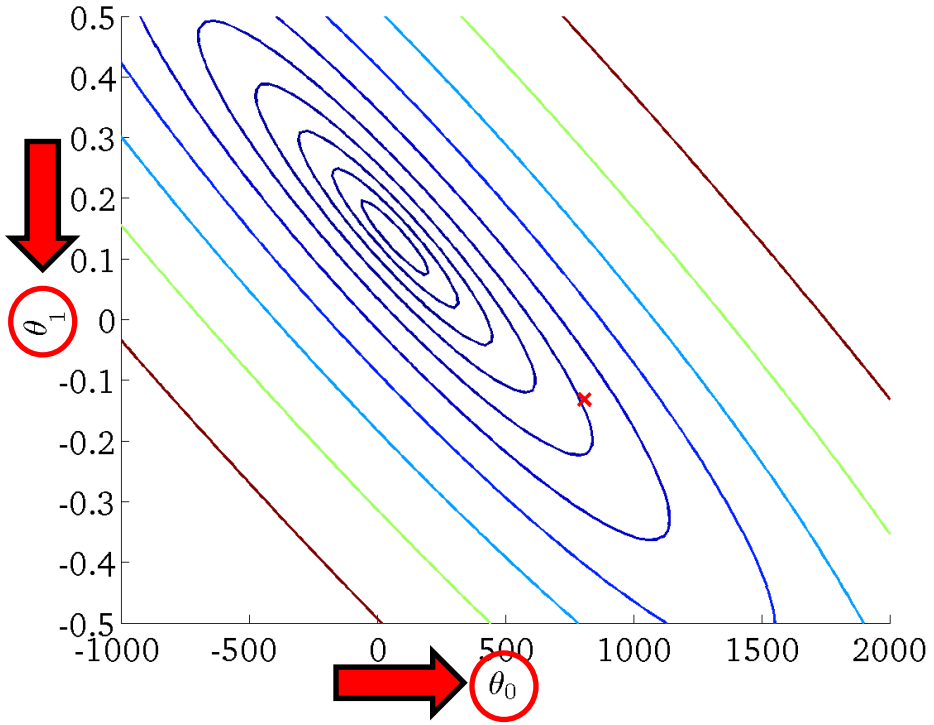
- Gradient Descent iteratively takes steps with a predefined rate in the **negative direction of the function gradient** at the current point (The Steepest Direction).



* Figure Ref: 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

$$\text{repeat until convergence } \{$$

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

$$\}$$

Gradient (slope of tangent line)

α >0 is Learning Rate that controls the step size toward the steepest direction.

This is what taking one step means

Gradient Descent Algorithm

repeat until convergence {

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

}

Gradient (slope of tangent line)

α >0 is Learning Rate that controls the step size toward the steepest direction.

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

Gradient Descent Algorithm

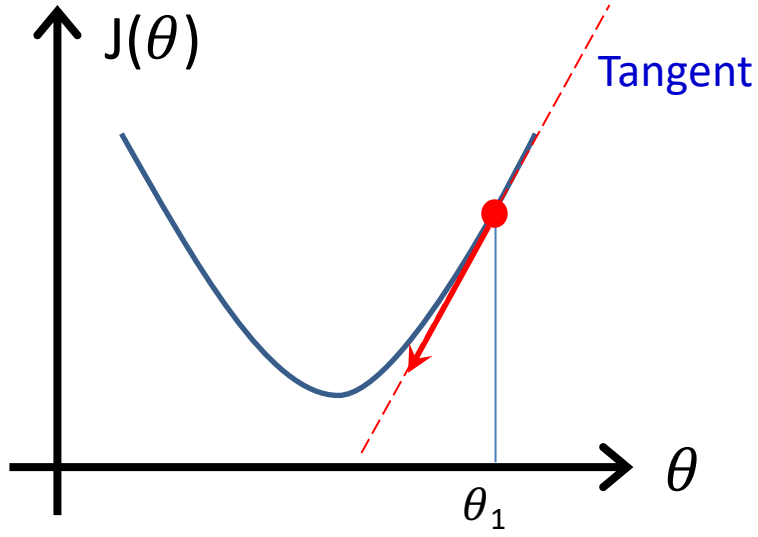
repeat until convergence {

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

}

Gradient (slope of tangent line)

α >0 is Learning Rate that controls the step size toward the steepest direction.



Gradient Descent Algorithm

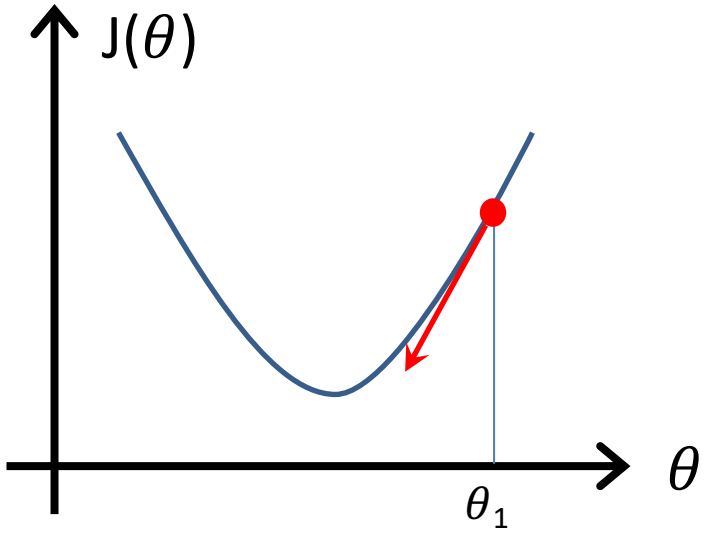
repeat until convergence {

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

}

Learning Rate

Gradient (slope of tangent line)



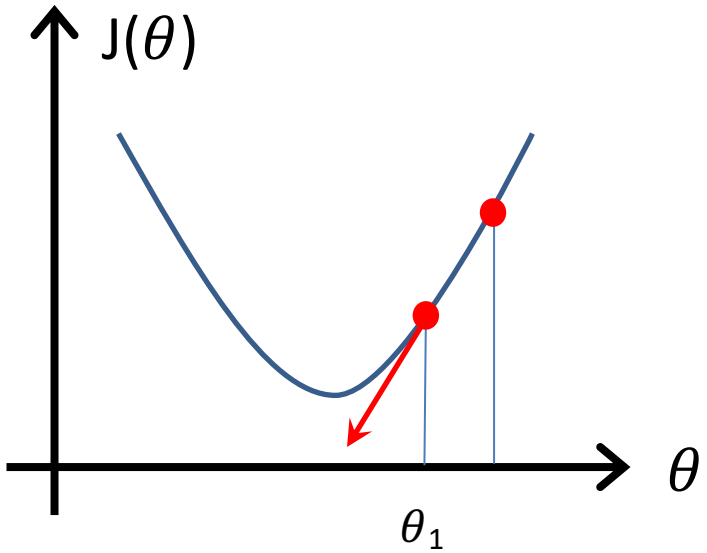
Gradient Descent Algorithm

repeat until convergence {

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

}

Learning Rate Gradient (slope of tangent line)



Gradient Descent Algorithm

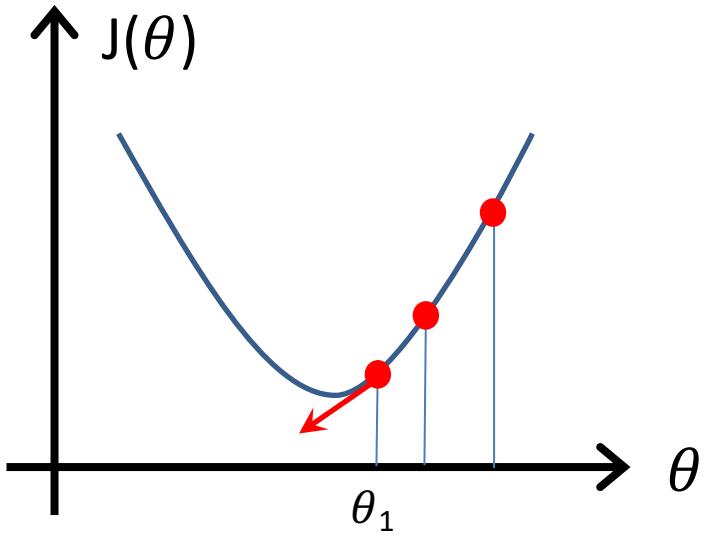
repeat until convergence {

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

}

Learning Rate

Gradient (slope of tangent line)



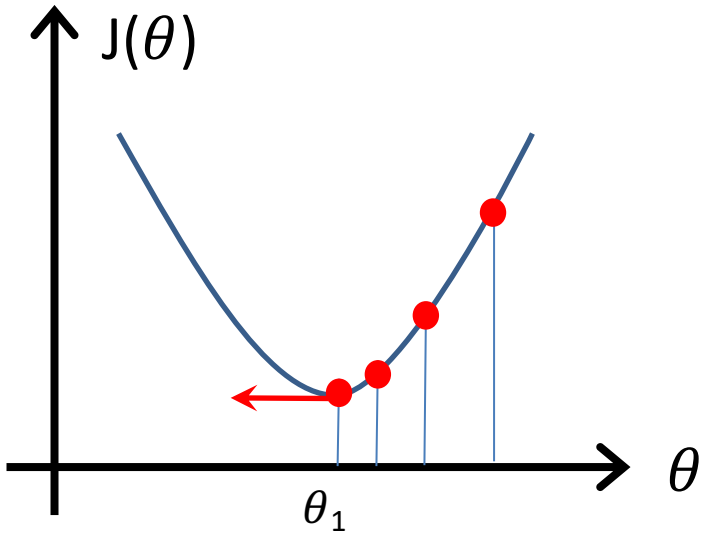
Gradient Descent Algorithm

repeat until convergence {

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

}

Learning Rate Gradient (slope of tangent line)



Gradient Descent Algorithm

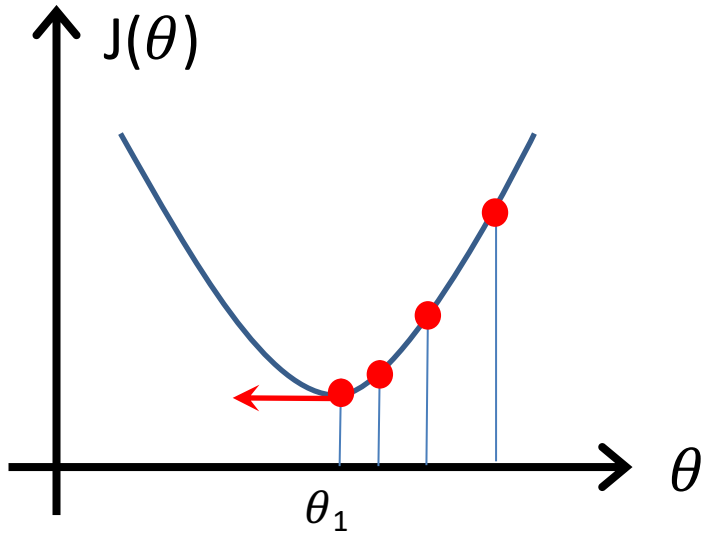
$$\text{repeat until convergence } \{$$

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

$$\}$$

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

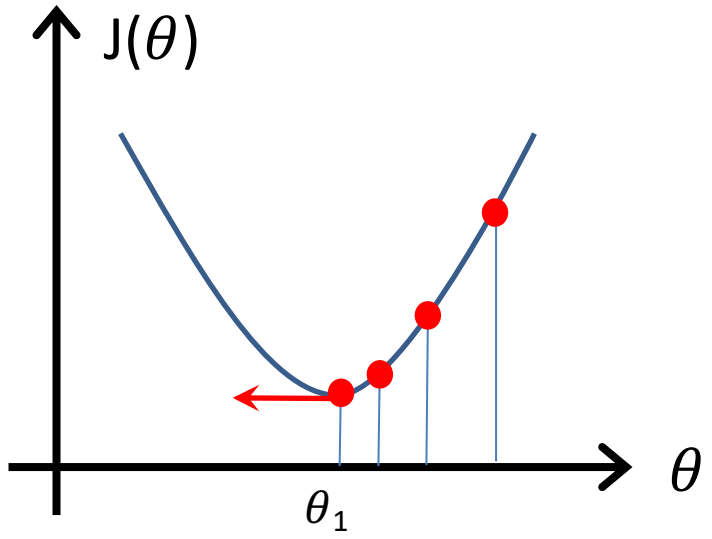
$$\text{repeat until convergence } \{$$

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

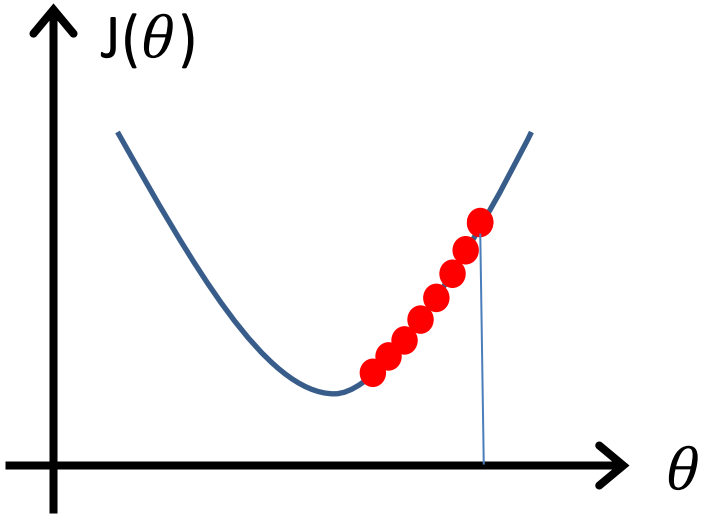
$$\}$$

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).



- α is Learning Rate that controls the step size toward the steepest direction.

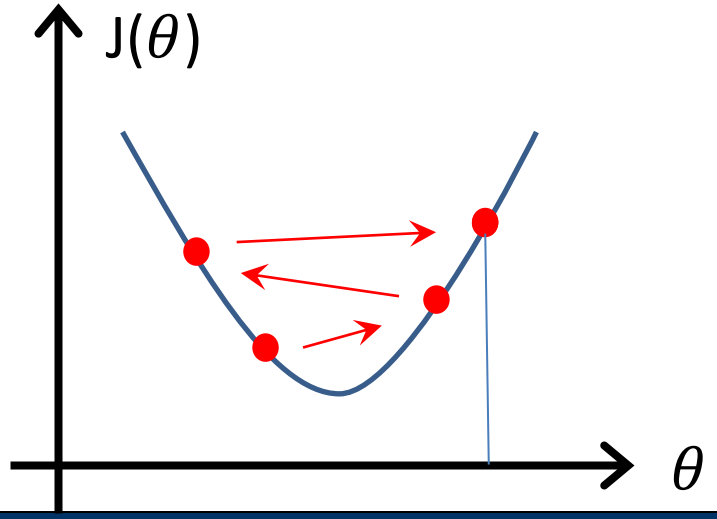


- Too Small α:

Converge will be too slow.

- Too Large α:

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 $\mathrm{h}(x)$ and J functions in the equations and calculate the derivatives to find an explicit update rule:

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

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

- **Linear Regression Model:**

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

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

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:

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

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

- After plugging J and calculating the Partial Derivatives:

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

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

Gradient Descent for Linear Regression

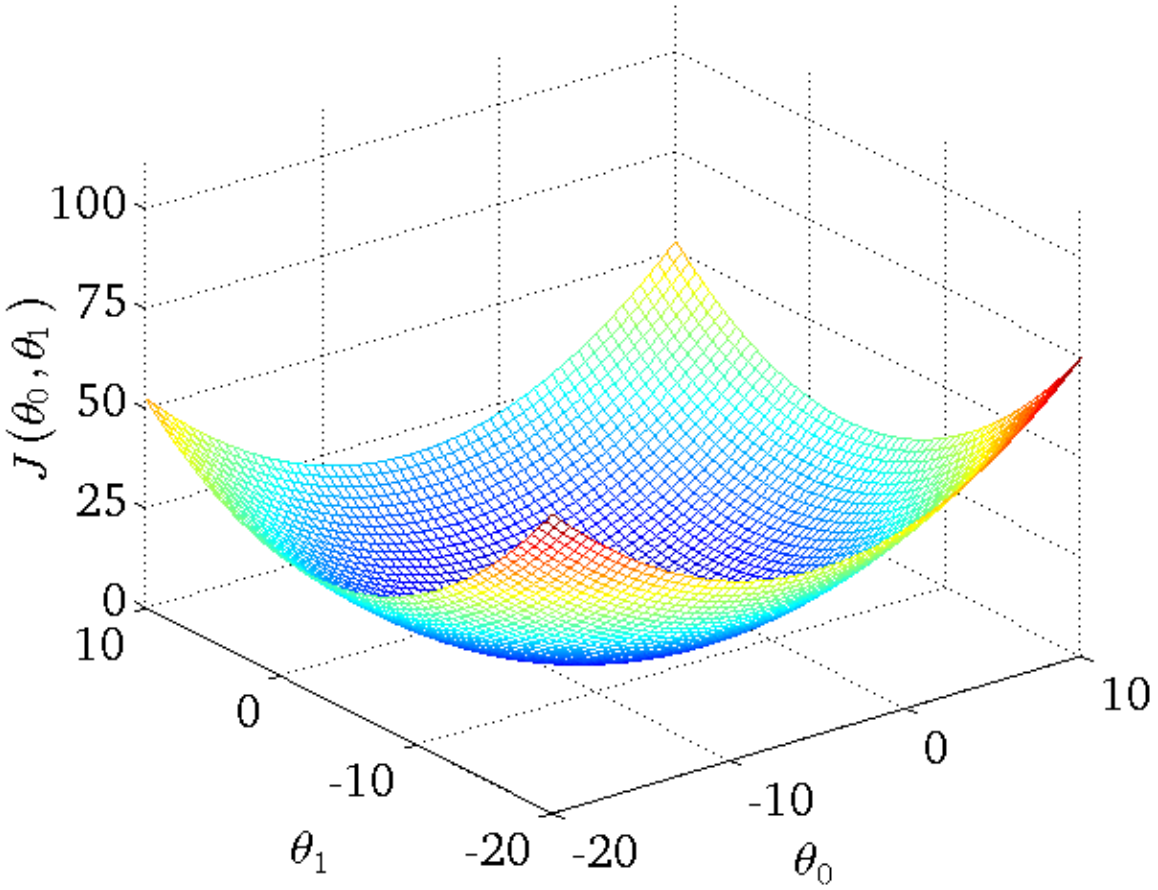
- After plugging in and calculating the Partial Derivatives:

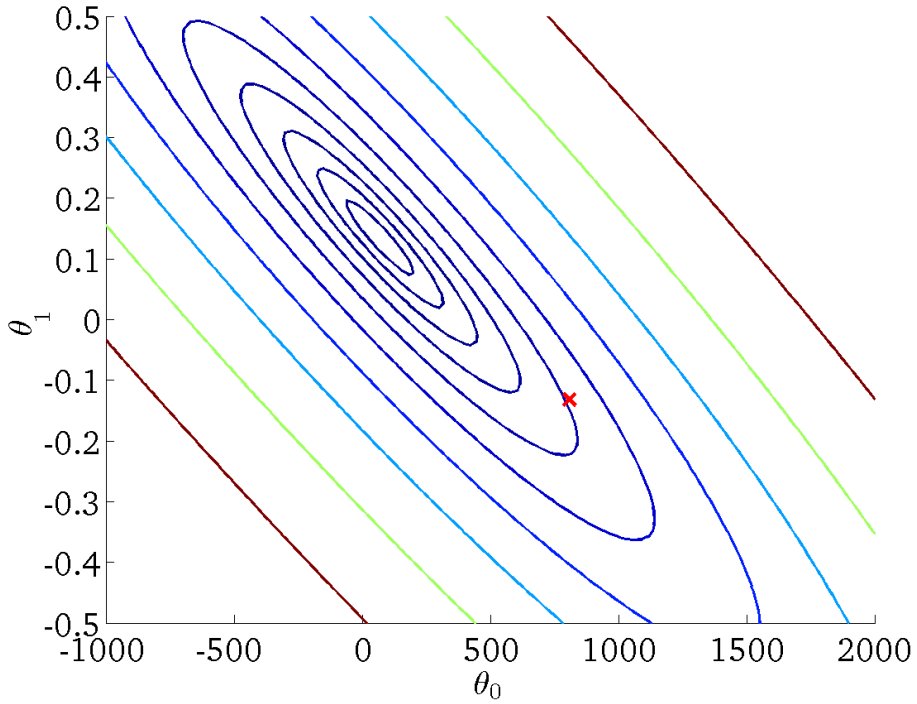
$$\text{repeat until convergence } \{$$

$$\left.\begin{aligned} \theta_0 &:= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \\ \theta_1 &:= \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)} \end{aligned}\right\} \text{update } \theta_0 \text{ and } \theta_1 \text{ simultaneously}$$

$$\}$$

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





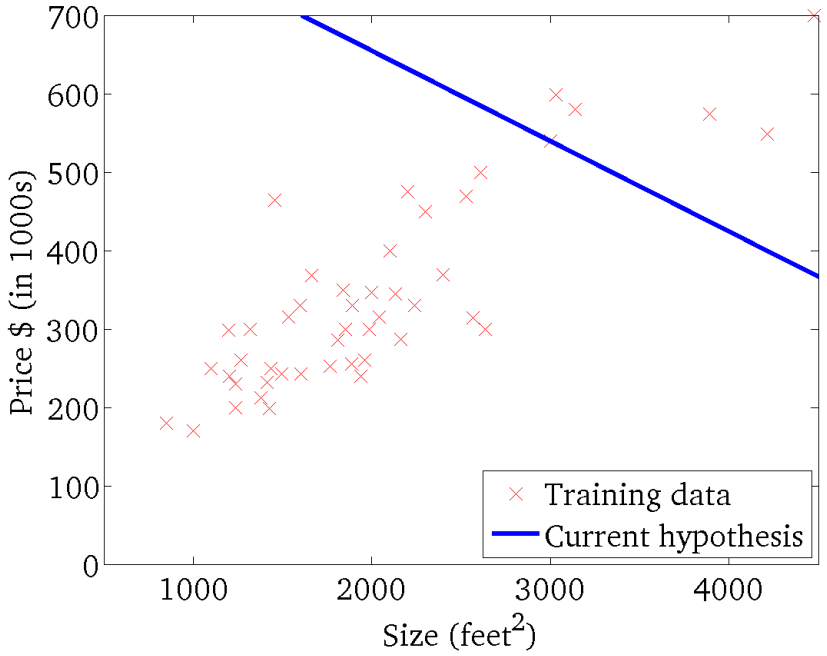
* Reference: Andrew Ng, Machine Learning, Stanford University.

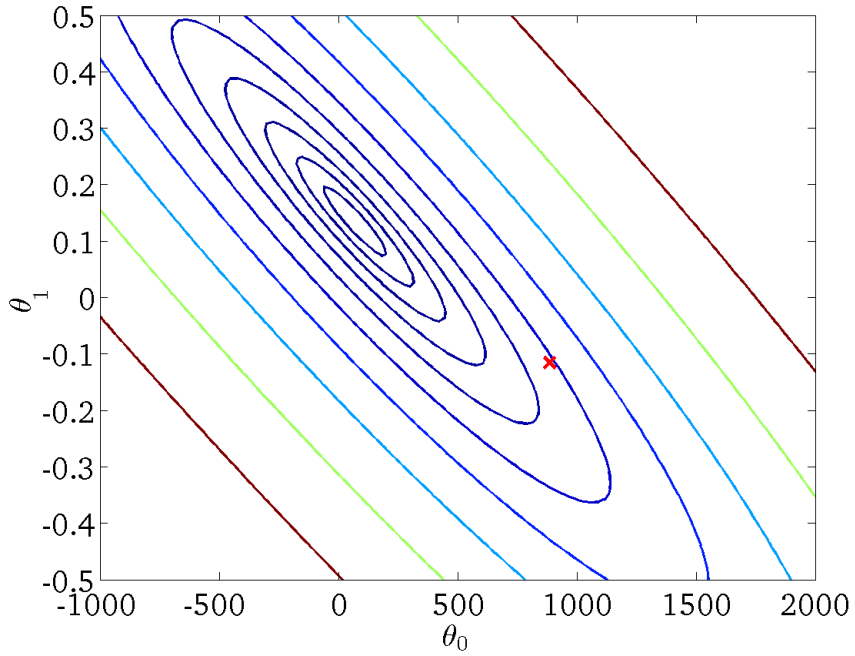
$$h_\theta(x) = \theta_0 + \theta_1 x$$

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

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

(function of the parameters θ_0, θ_1)





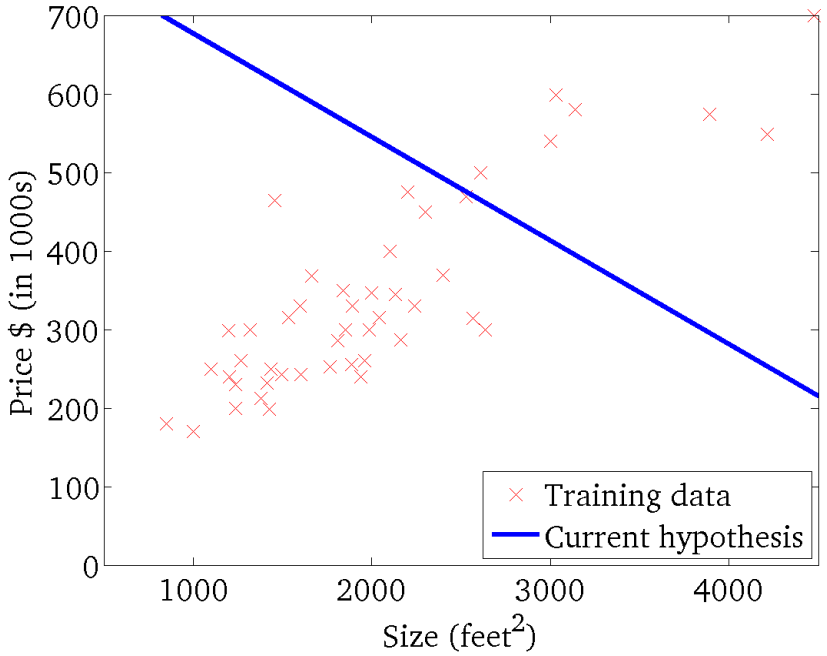
* Example from Andrew Ng, Machine Learning, Stanford Univ.

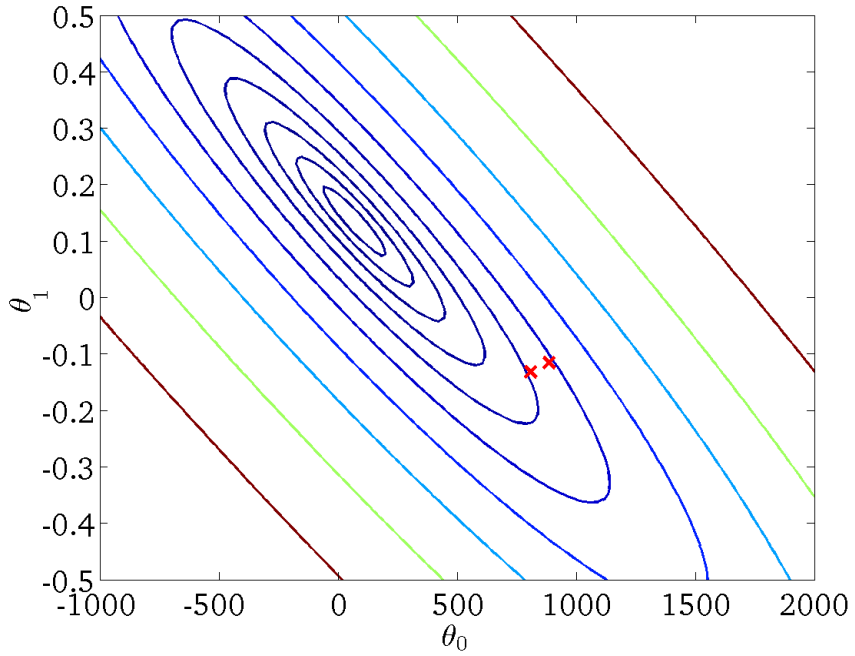
$$h_\theta(x) = \theta_0 + \theta_1 x$$

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

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

(function of the parameters θ_0, θ_1)



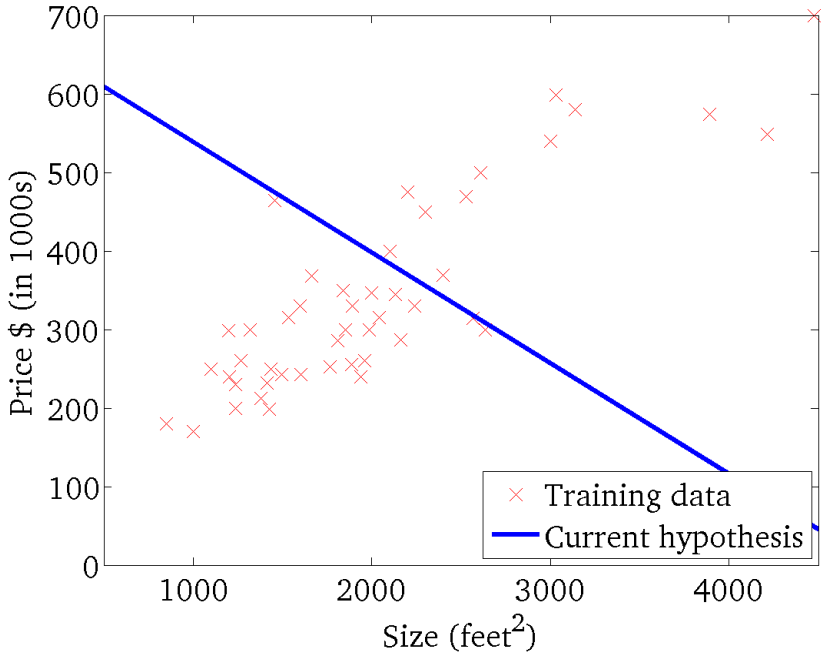


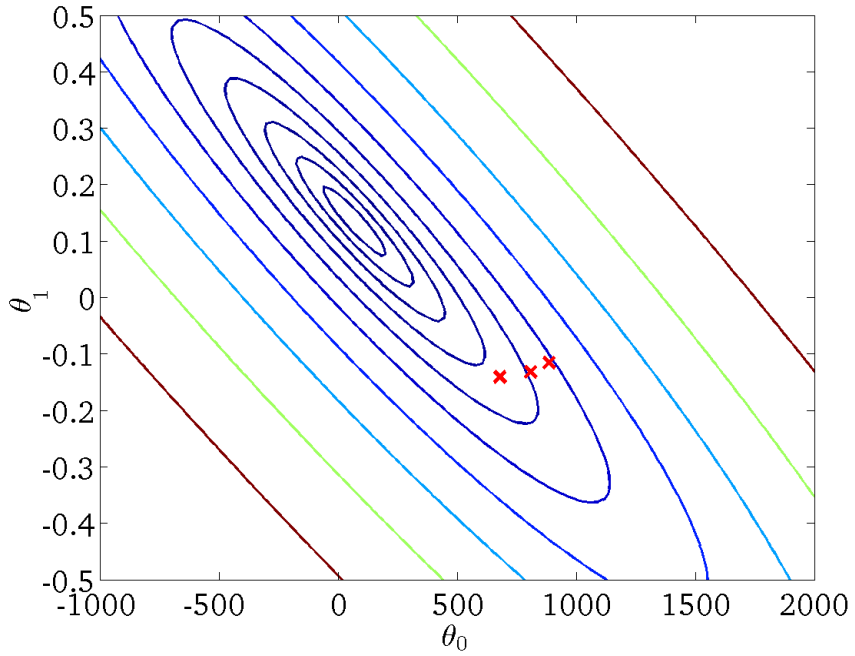
$$h_\theta(x) = \theta_0 + \theta_1 x$$

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

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

(function of the parameters θ_0, θ_1)



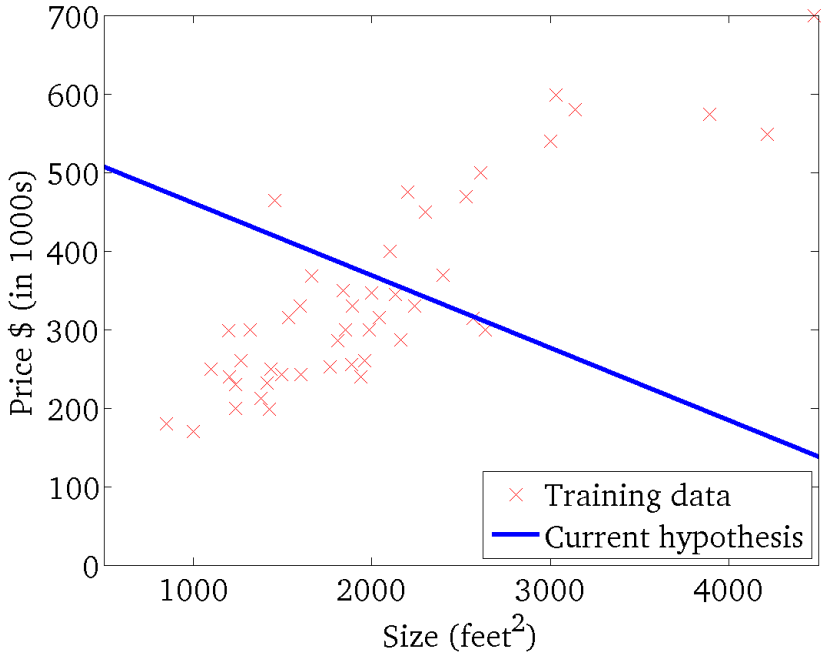


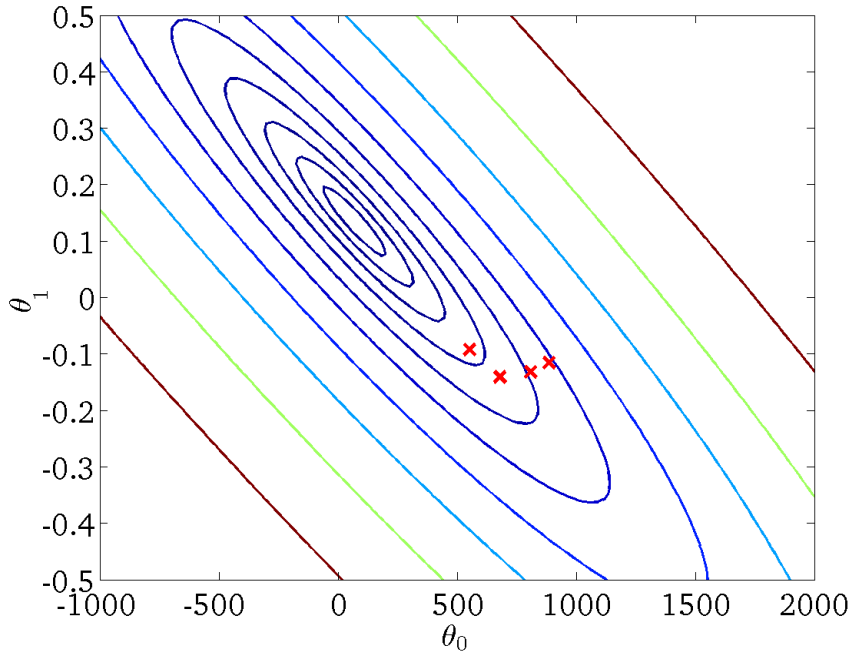
$$h_\theta(x) = \theta_0 + \theta_1 x$$

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

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

(function of the parameters θ_0, θ_1)



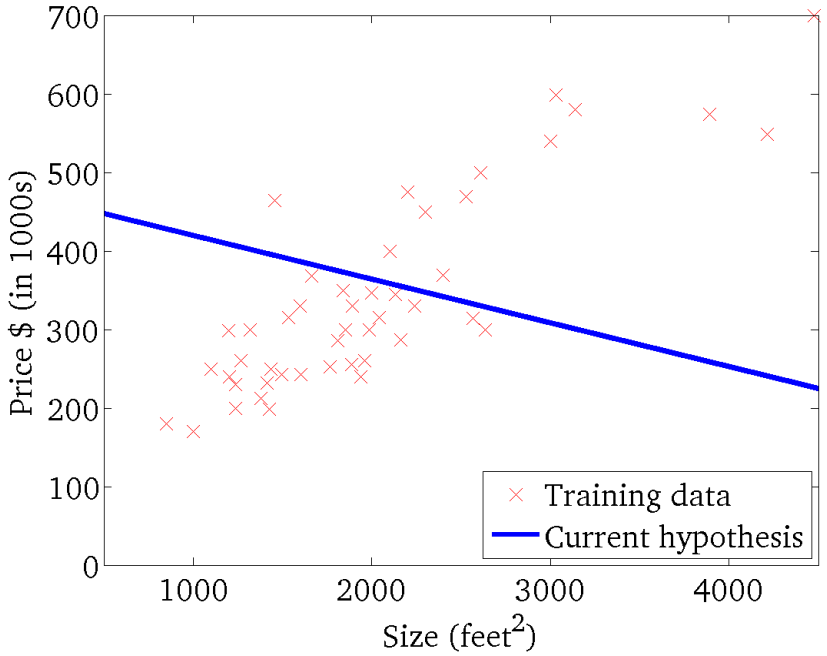


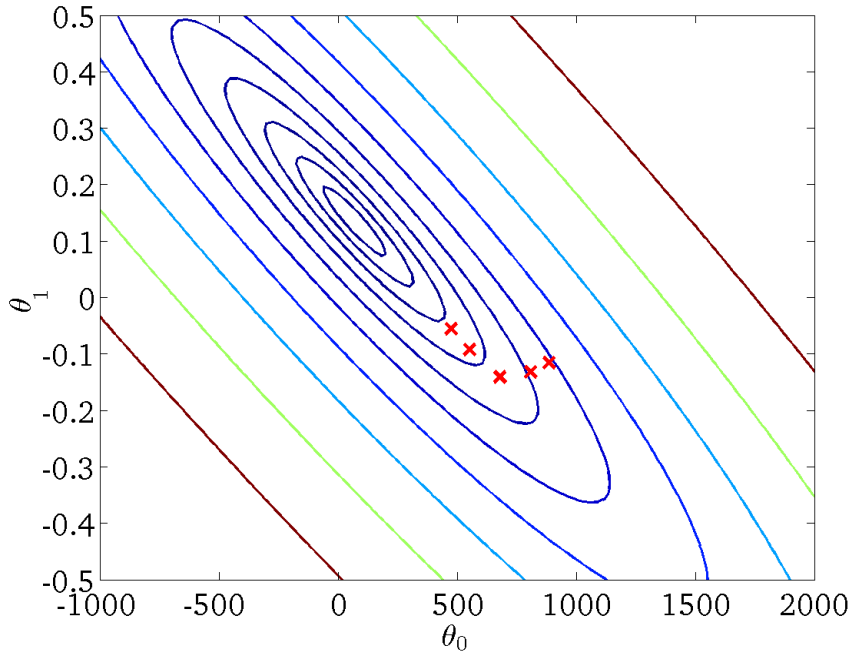
$$h_\theta(x) = \theta_0 + \theta_1 x$$

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

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

(function of the parameters θ_0, θ_1)



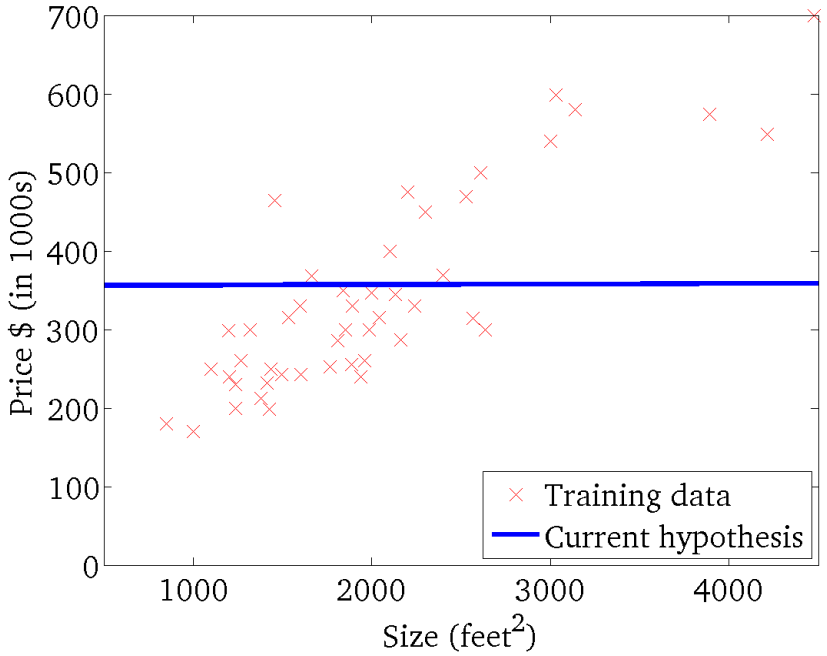


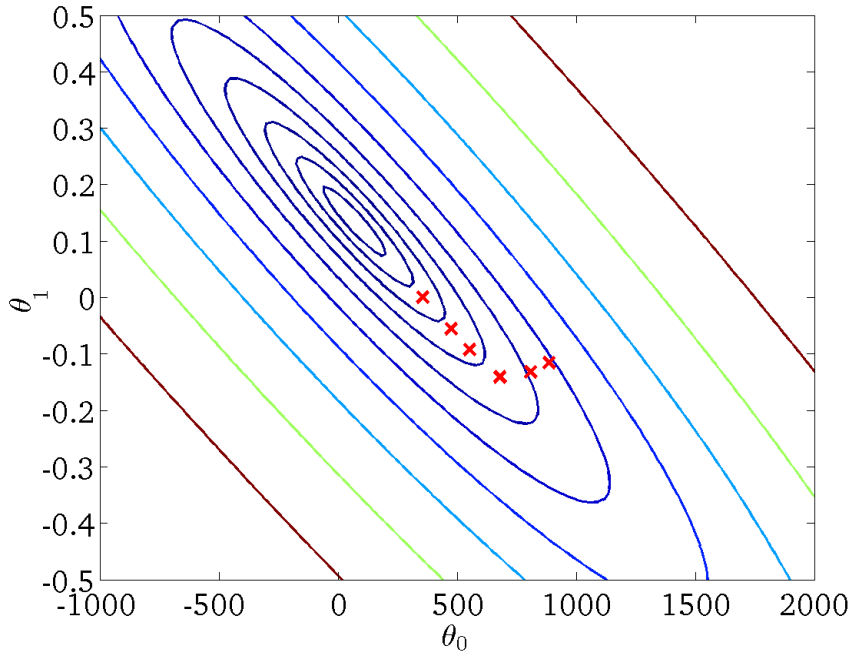
$$h_\theta(x) = \theta_0 + \theta_1 x$$

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

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

(function of the parameters θ_0, θ_1)



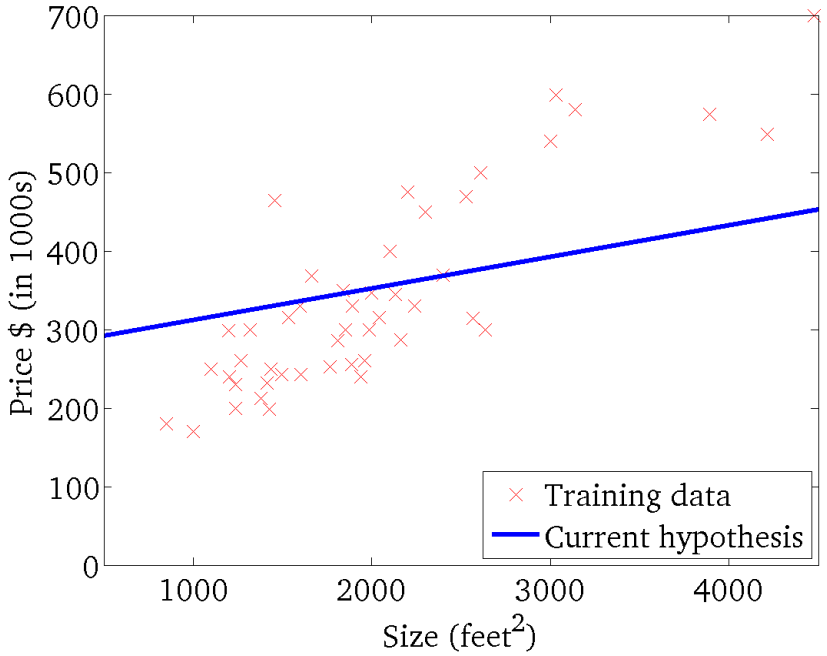


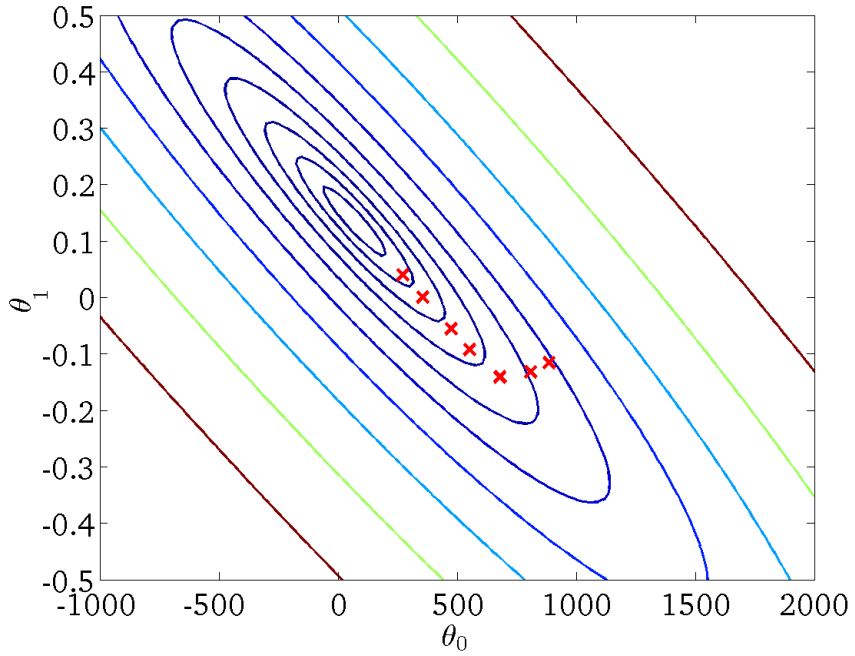
$$h_\theta(x) = \theta_0 + \theta_1 x$$

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

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

(function of the parameters θ_0, θ_1)



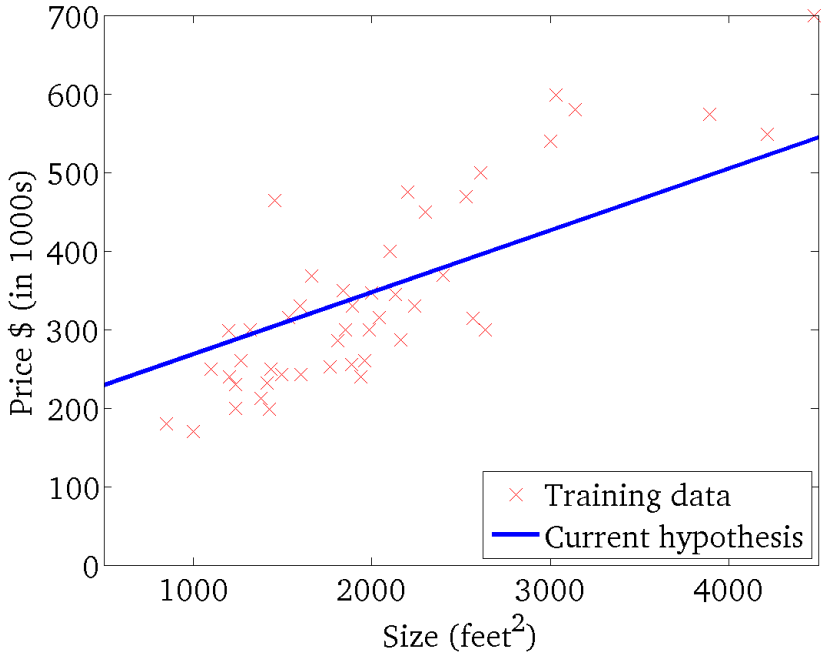


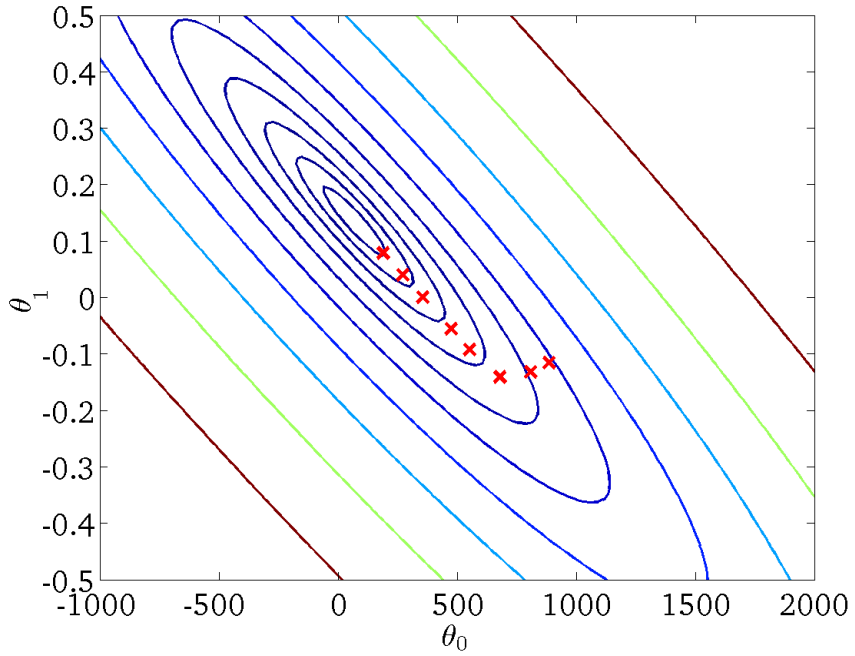
$$h_\theta(x) = \theta_0 + \theta_1 x$$

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

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

(function of the parameters θ_0, θ_1)



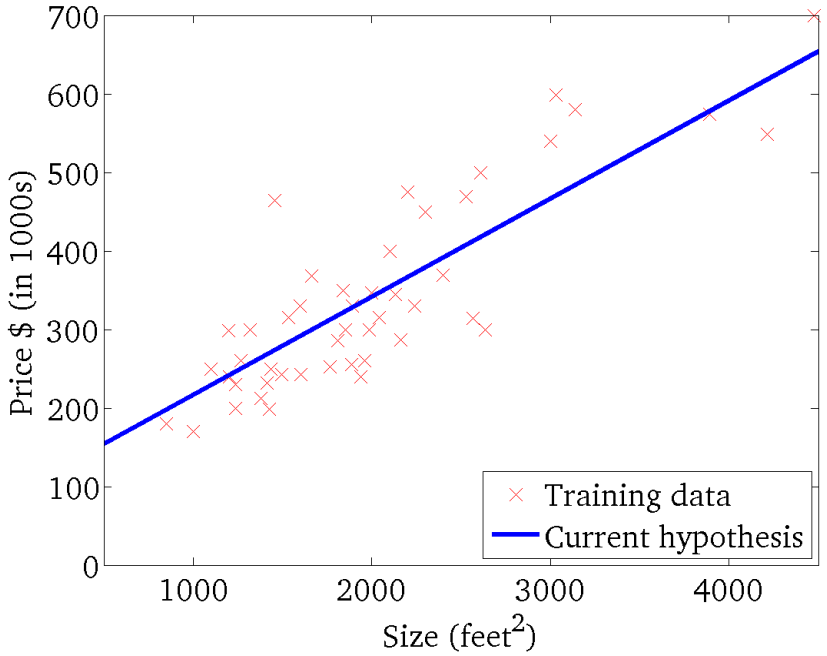


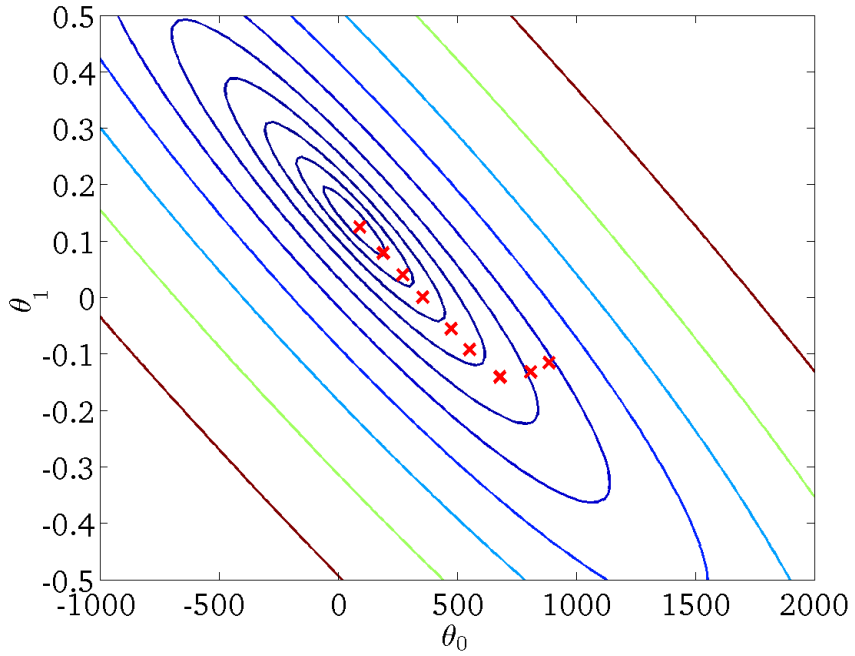
$$h_\theta(x) = \theta_0 + \theta_1 x$$

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

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

(function of the parameters θ_0, θ_1)





Review:
Matrices and Vectors

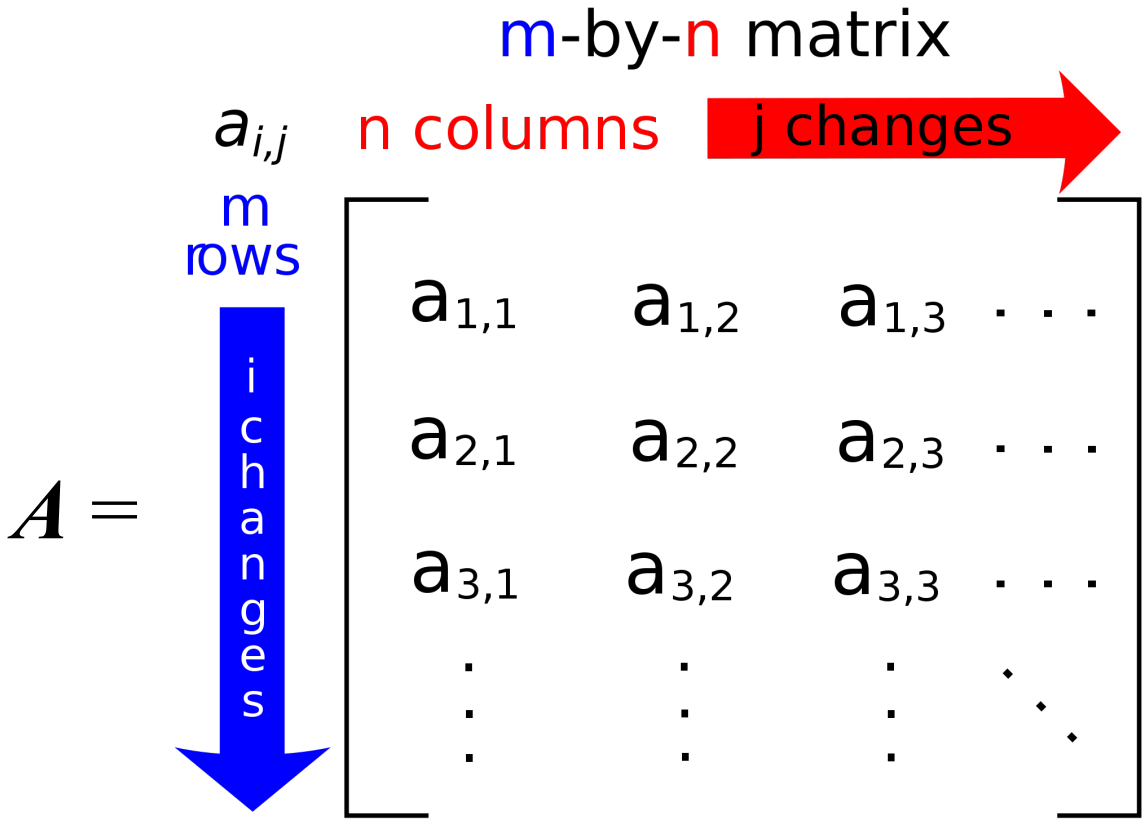
Matrix

- **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

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

Matrix Elements

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



Vector

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

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

Notation

- We usually use a **bold** CAPITAL *Italic* letter to name a matrix.
  - E.g. $\boldsymbol{M}, \boldsymbol{A}, \ldots$

- We usually use a **bold** lower-case *Italic* letter to name a vector.
  - E.g. $\boldsymbol{v}, \boldsymbol{u}, \ldots$

- We usually use a non-bold *Italic* letter to name a scalar (variable).
  - E.g. $s, c, \ldots$

Linear Regression with multiple features

Linear Regression with One Feature

One Feature → Target

Size in feet^2 (x)	Price (y)
1000	410K
1200	600K
1230	620K
1340	645K
…	…

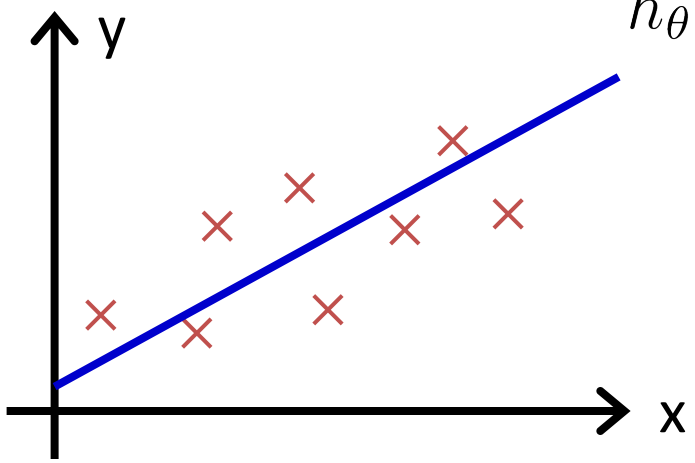
Linear Regression with One Feature

One Feature → Target

Size in feet² (x)	Price (y)
1000	410K
1200	600K
1230	620K
1340	645K
…	…

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

Linear Regression with One Feature



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

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

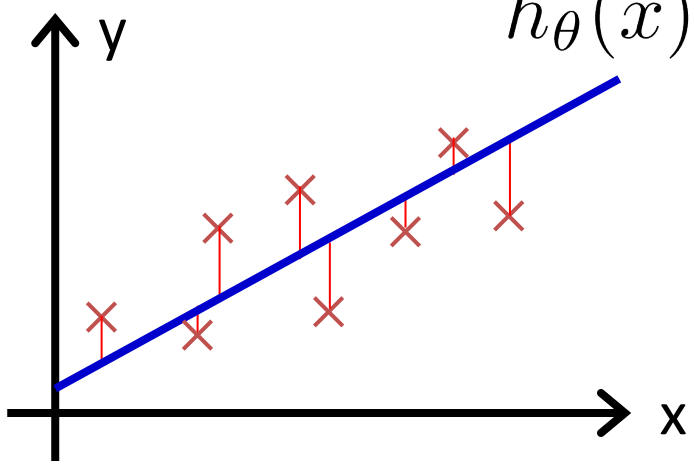
How to Select the Parameters

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

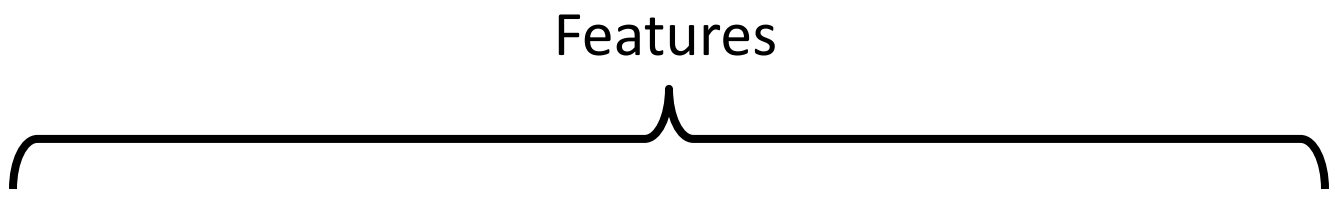
Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

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

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

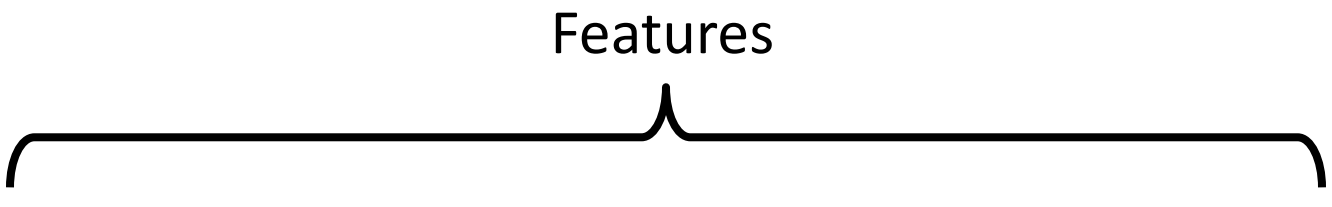


Linear Regression with Multiple Features



Size in feet^2 (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
…	…	…	…

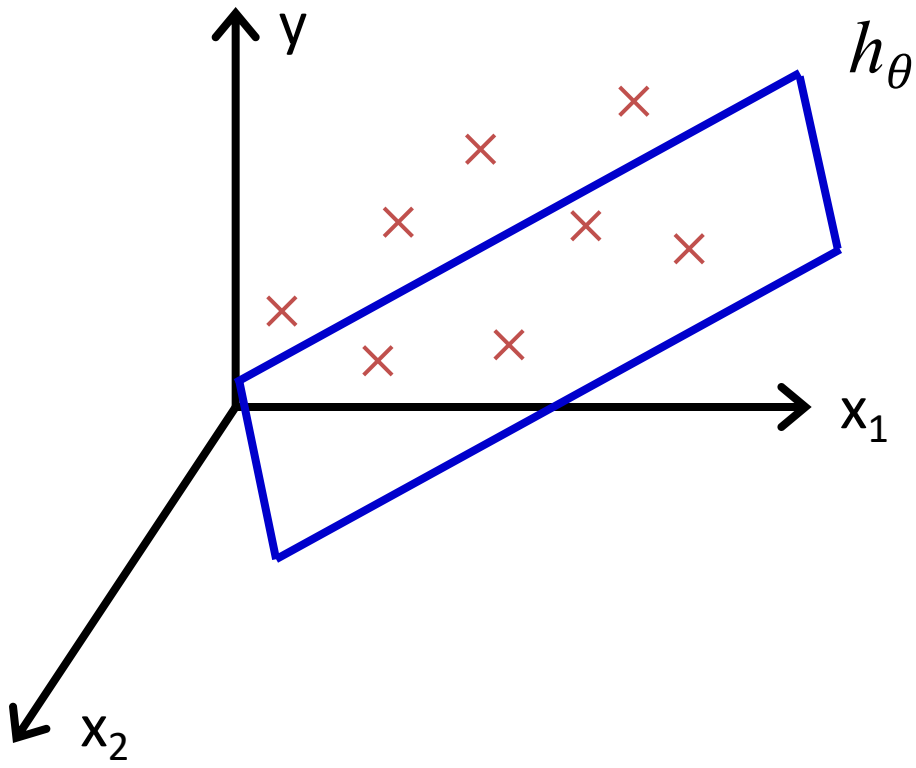
Linear Regression with Multiple Features



Size in feet^2 (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
…	…	…	…

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

Example: Two Features



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

Linear Regression with Multiple Features

Size in feet^2 (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

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

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

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:

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

Gradient Descent for Multiple Variables*

- Regression Model:

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

- Parameter vector & feature vectors:

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

- **Cost function:**

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

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

Gradient Descent for Multiple Variables

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

Repeat $\{$

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

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

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

Gradient Descent for Multiple Features

Previously (n=1):

Repeat $\{$

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

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

(simultaneously update θ_0, θ_1)

$\}$

New algorithm $(n \geq 1)$:

Repeat $\{$

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

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

$\}$

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

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

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

...

Important tip: Implementing Gradient Descent in Practice

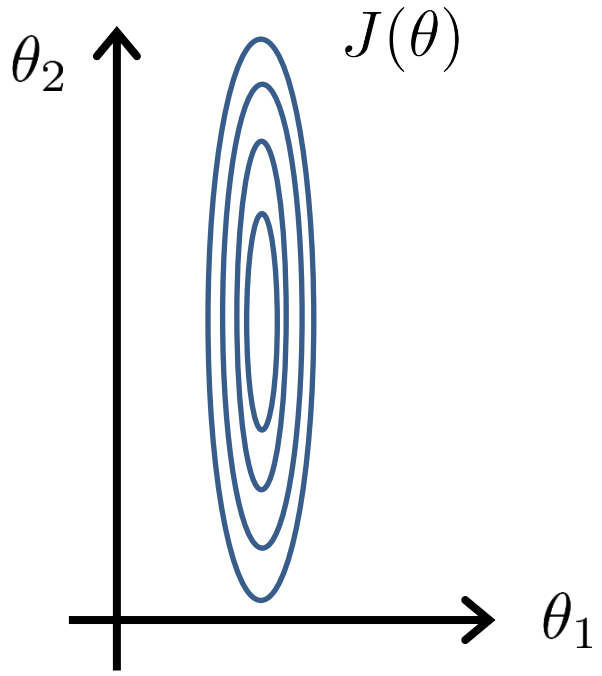
- Make sure to **Normalize** data samples to have all features on a similar scale.

E.g. x_1 = size (0-4000 sqr feet^2).

x_2 = number of bedrooms (1-5).

=> cost contour will be in the form of very tall and skinny ellipses.

=> gradient descent will have a very hard time to find the minimum (convergence will be very slow).



Example

- Predicting value of a house based on **size of the house** and **number of bedrooms**!

Feature1 = size (0-4000 sqr feet^2).

Feature2 = number of bedrooms (1-5).

An easy way to scale features:

F1_Scaled = size / max(size)

F2_Scaled = number of bedrooms / max(number of bedrooms)

(0 <= F1_Scaled <= 1)

(0 <= F2_Scaled <= 1)

Example

- Predicting value of a house based on **size of the house** and **number of bedrooms**!

- Before Scaling:

	Number of Bedrooms	Size (sqr footage)
1	1	1300
2	3	2400
3	3	2270
4	2	1450
5	2	1400
6	4	2900

Example

- Predicting value of a house based on **size of the house** and **number of bedrooms**!

- After Scaling:

	Number of Bedrooms	Size (sqr footage)
1	1/4	1300/2900
2	3/4	2400/2900
3	3/4	2270/2900
4	2/4	1450/2900
5	2/4	1400/2900
6	4/4	2900/2900

Implementing Gradient Descent in Practice

- Make sure to **Normalize** data samples to have all features on a similar scale.

E.g. x_1 = size (0-3000 sqr feet^2).

x_2 = number of bedrooms (1-5).

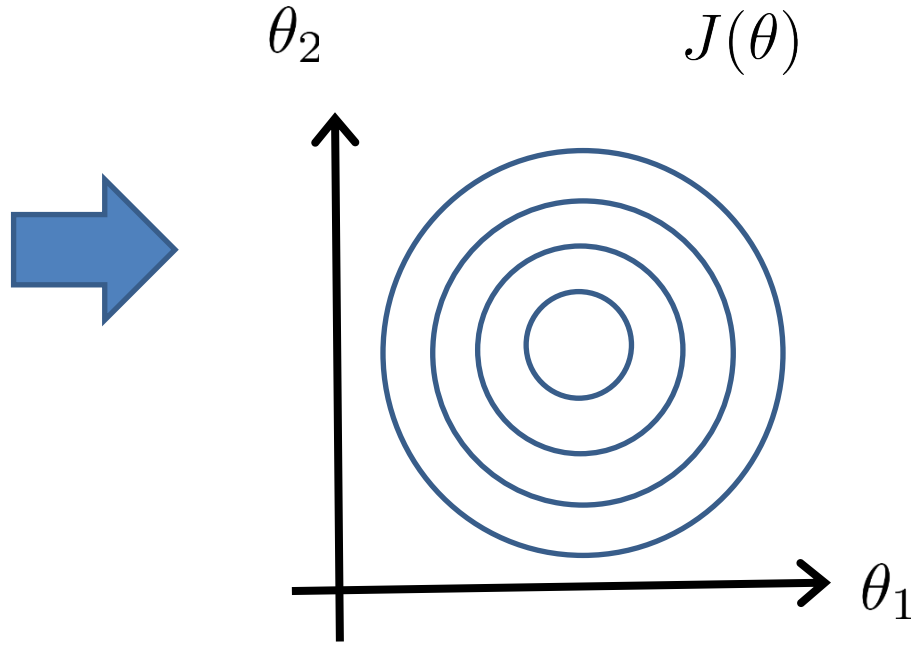
An easy way to scale features:

F1_Scaled = size / max(size)

F2_Scaled = number of bedrooms / max(number of bedrooms)

(0 <= F1_Scaled <= 1)

(0 <= F2_Scaled <= 1)



Implementing Gradient Descent in Practice

- A more general approach for normalization is to make each feature **zero mean** with **unit standard deviation** over all data samples (for each feature column). In other word, we should normalize **each column of the feature table**.
- Compute the mean and standard deviation for each feature (x_{nd} is the dth feature of nth data sample):

$$mean(x_d) = \bar{x}_d = \frac{1}{N}\sum_{n=1}^{N} x_{nd} \qquad std(x_d) = s_d = \sqrt{\frac{1}{N-1}\sum_{n=1}^{N}(x_{nd} - \bar{x}_d)^2}$$

- Scale the feature accordingly

$$x_{nd} \leftarrow \left(\frac{x_{nd} - \bar{x}_d}{s_d}\right)$$

Thank You!