

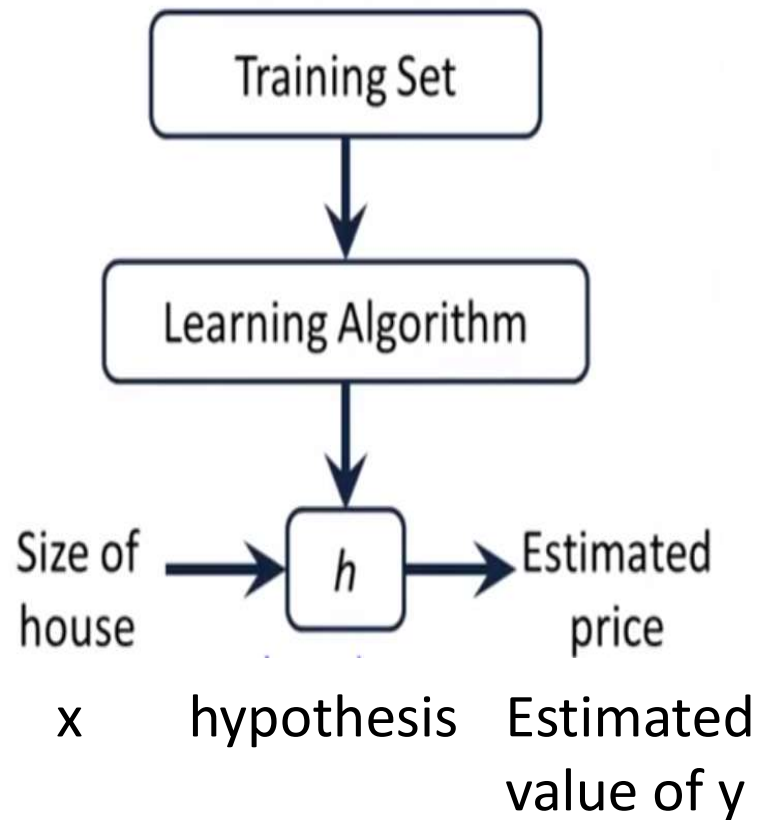
# Linear Regression

## Supervised Learning

Some slides were adapted/taken from various sources, including Prof. Andrew Ng's Coursera Lectures, Stanford University, Prof. Kilian Q. Weinberger's lectures on Machine Learning, Cornell University, Prof. Sudeshna Sarkar's Lecture on Machine Learning, IIT Kharagpur, Prof. Bing Liu's lecture, University of Illinois at Chicago (UIC), CS231n: Convolutional Neural Networks for Visual Recognition lectures, Stanford University and many more. We thankfully acknowledge them. Students are requested to use this material for their study only and **NOT** to distribute it.

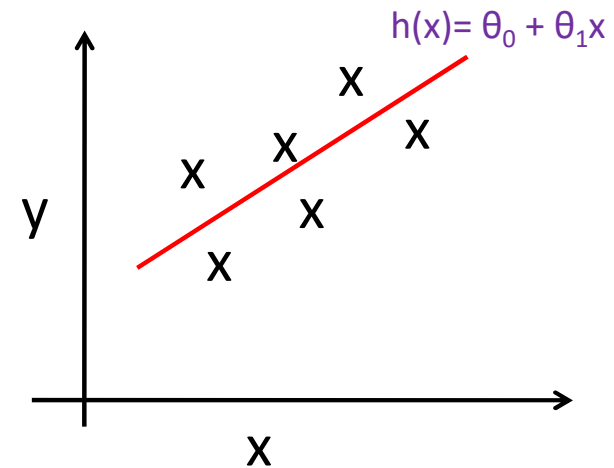
to continue...

# Supervised Learning



How do we represent  $h$

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



Univariate linear regression:  
linear regression with one  
variable

## Cost Function

Training Set	Size in feet <sup>2</sup> (x)	Price (\$) in 1000's (y)
	2104	460
	1416	232
	1534	315
	852	178
	...	...

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

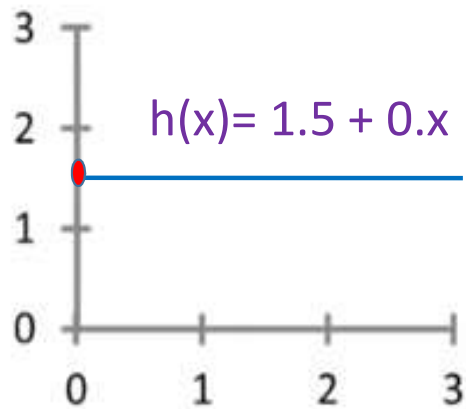
$\theta_1$ 's  $\rightarrow$  Parameters

How to choose  $\theta_1$ 's

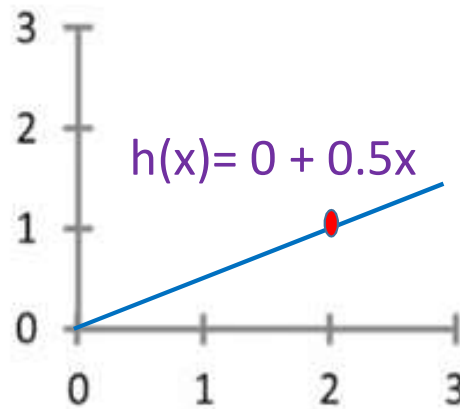
# Cost Function

Hypothesis Function:

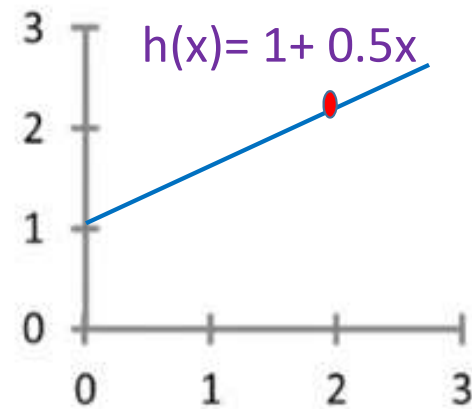
$$h_{\theta}(x) = \theta_0 + \theta_1 x$$



$$\theta_0 = 1.5$$
$$\theta_1 = 0$$



$$\theta_0 = 0$$
$$\theta_1 = 0.5$$



$$\theta_0 = 1$$
$$\theta_1 = 0.5$$

# Cost Function



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

Parameters:  $\theta_0, \theta_1$

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

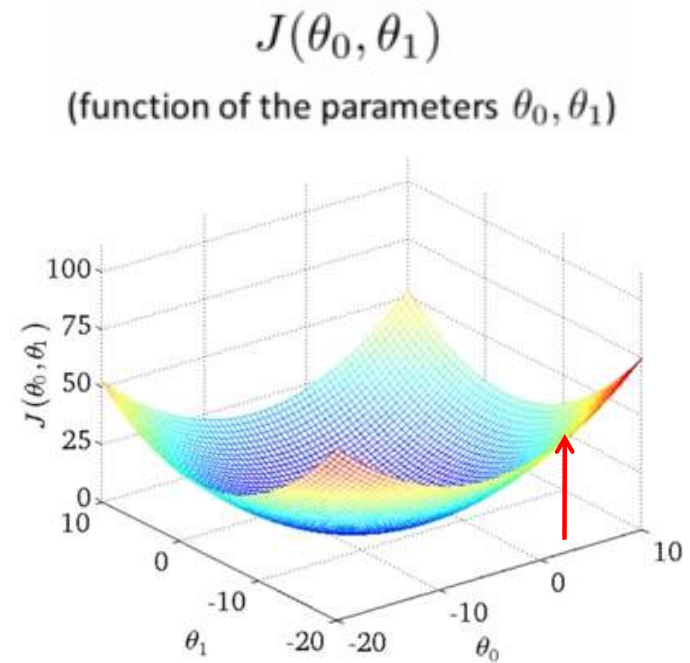
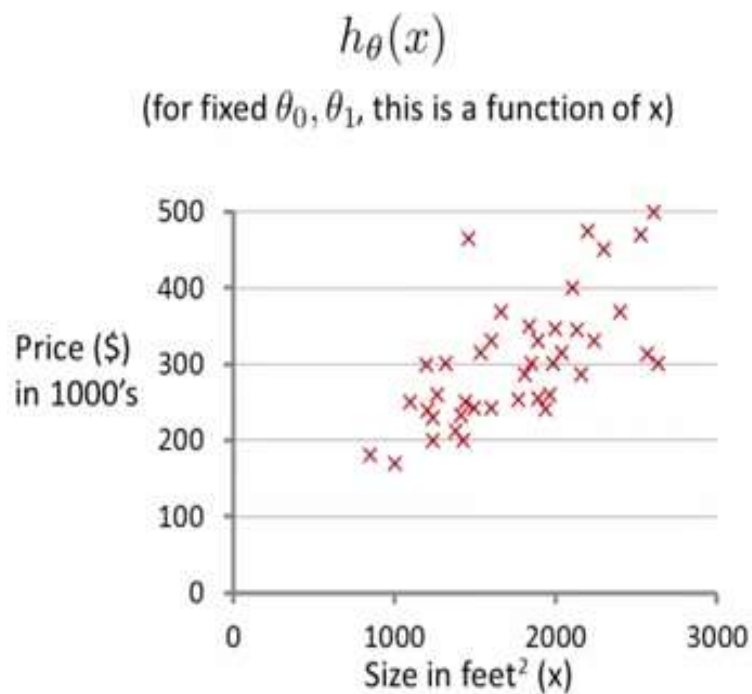
Squared error function

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

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

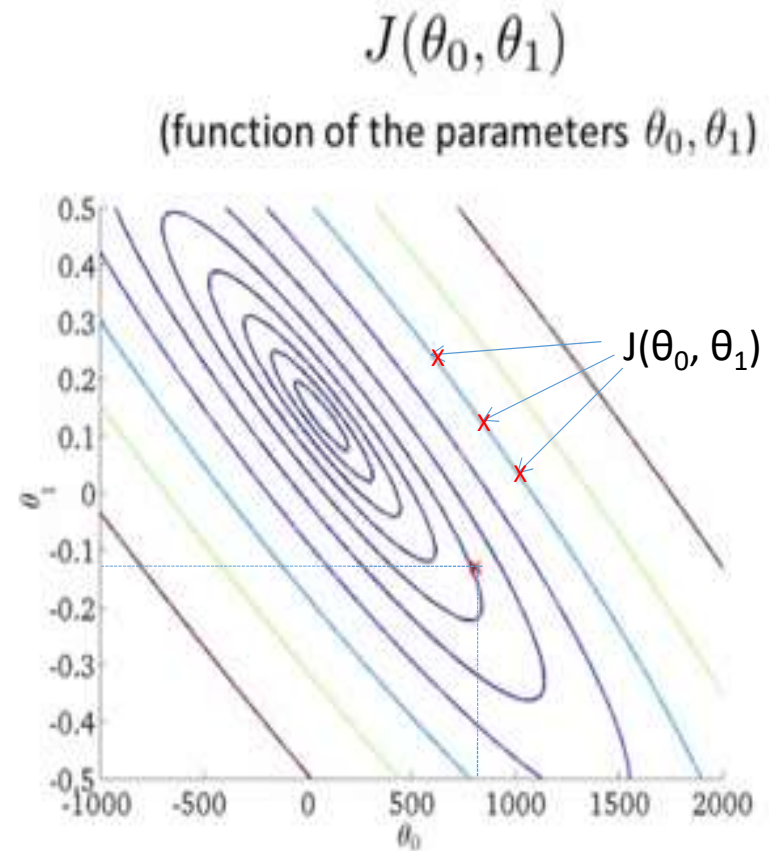
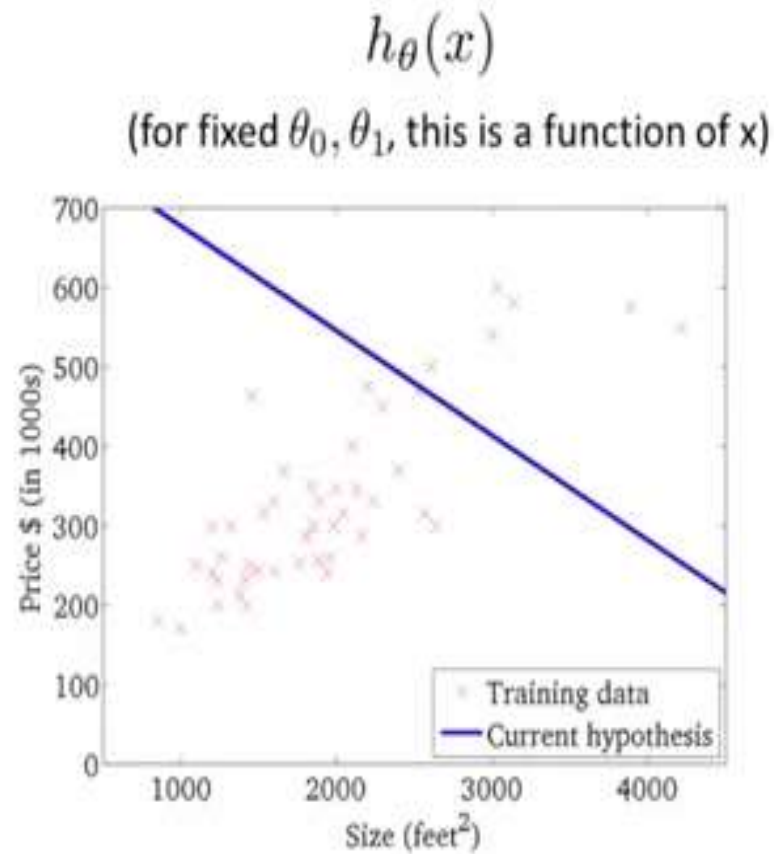
$m$  = No. of training samples

# Cost Function



$J(\theta_0, \theta_1)$  = value of the height of the surface

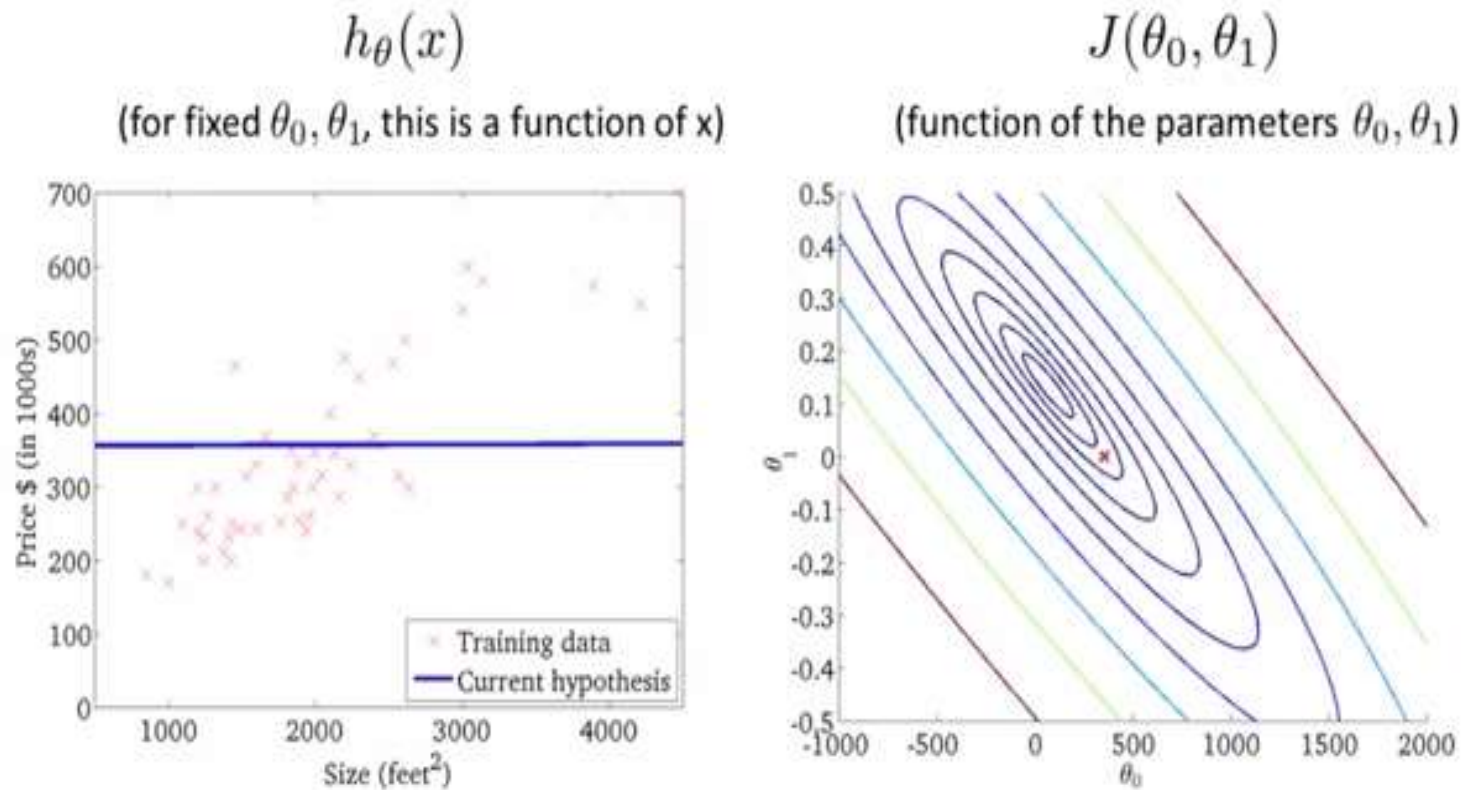
# Contour Plots / Figures



$$(\theta_0, \theta_1) = (800, -0.125)$$

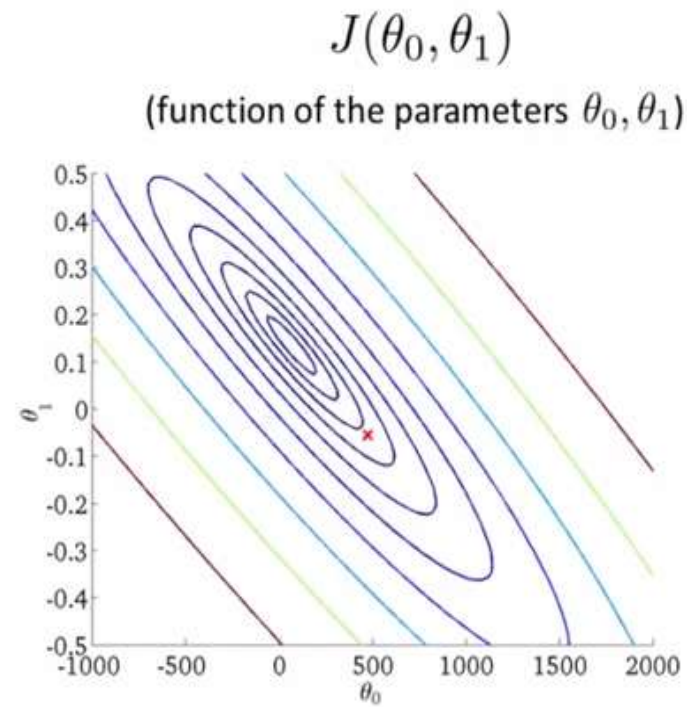
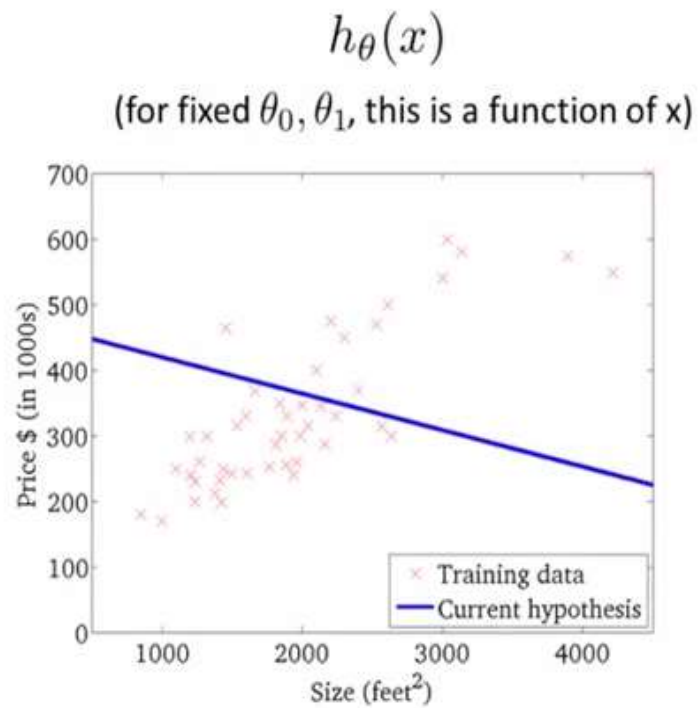


# Contour Plots / Figures



$$(\theta_0, \theta_1) = (360, 0)$$

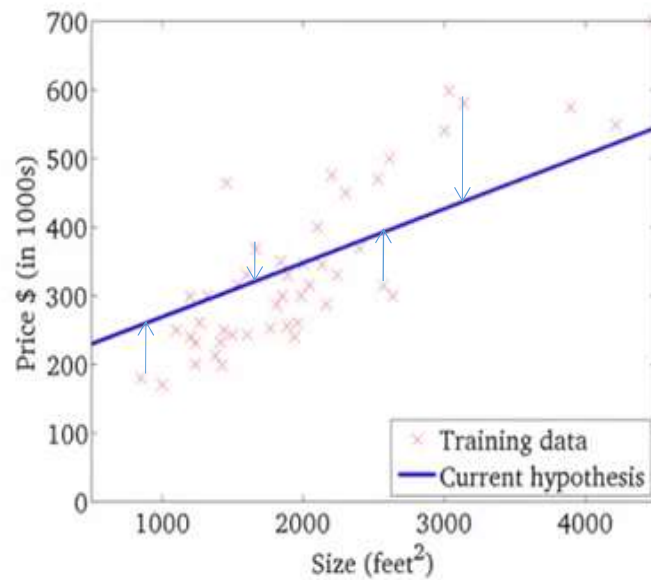
# Contour Plots / Figures



# Contour Plots / Figures

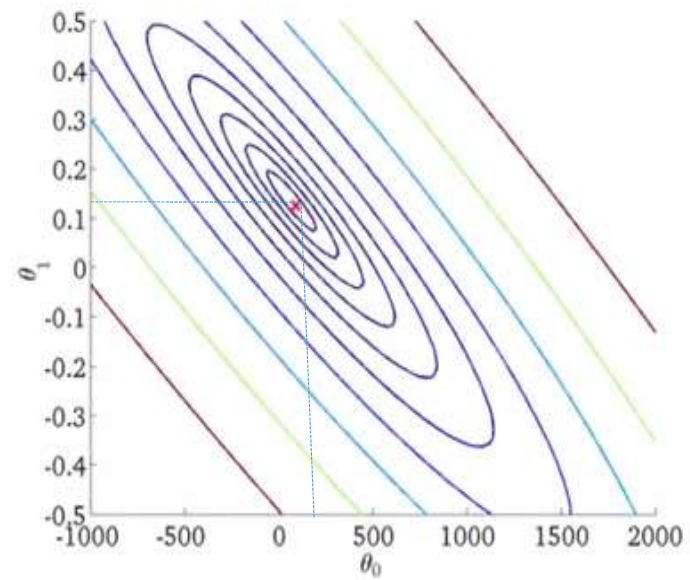
$$h_{\theta}(x)$$

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



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

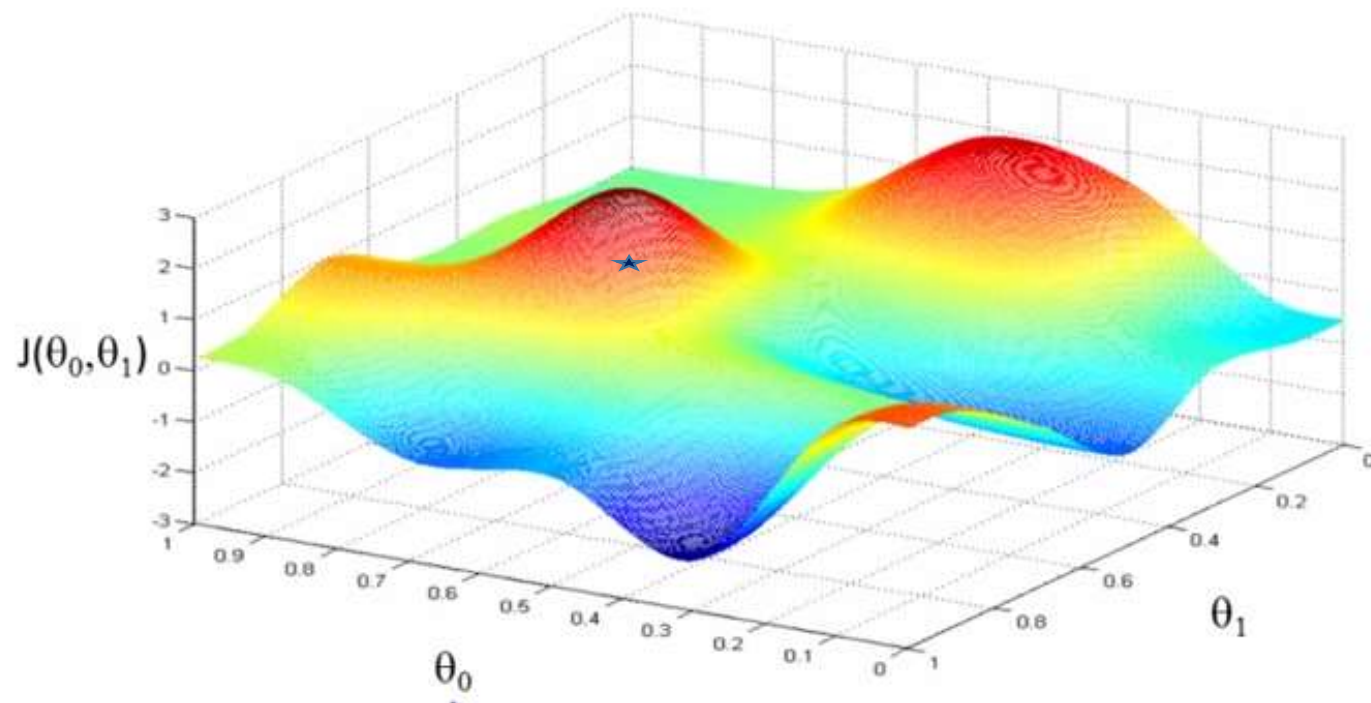
(function of the parameters  $\theta_0, \theta_1$ )



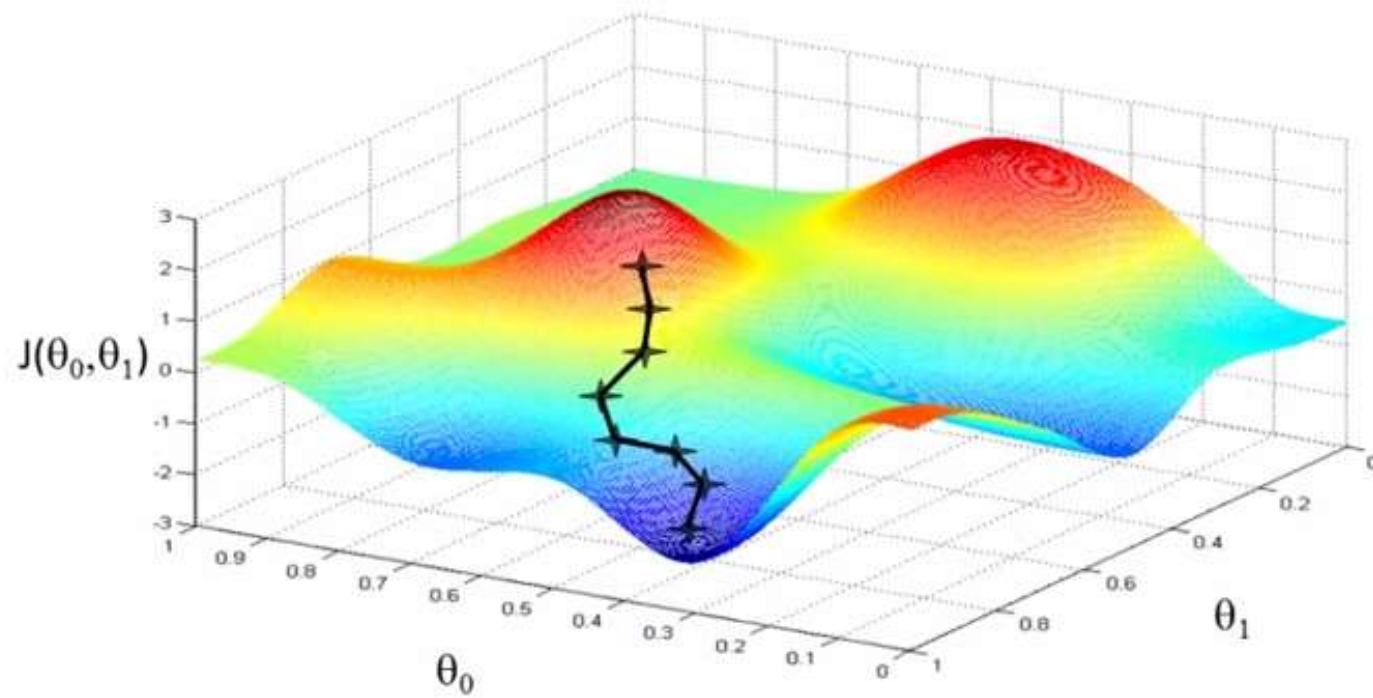
# Gradient Descent

- Let some function  $J(\theta_0, \theta_1)$
- We have to find  $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$
- Start with some  $(\theta_0, \theta_1)$  (let say  $\theta_0=0, \theta_1=0$ )
- Keep changing  $\theta_0, \theta_1$  to reduce  $J(\theta_0, \theta_1)$   
until we hopefully end up at a minimum

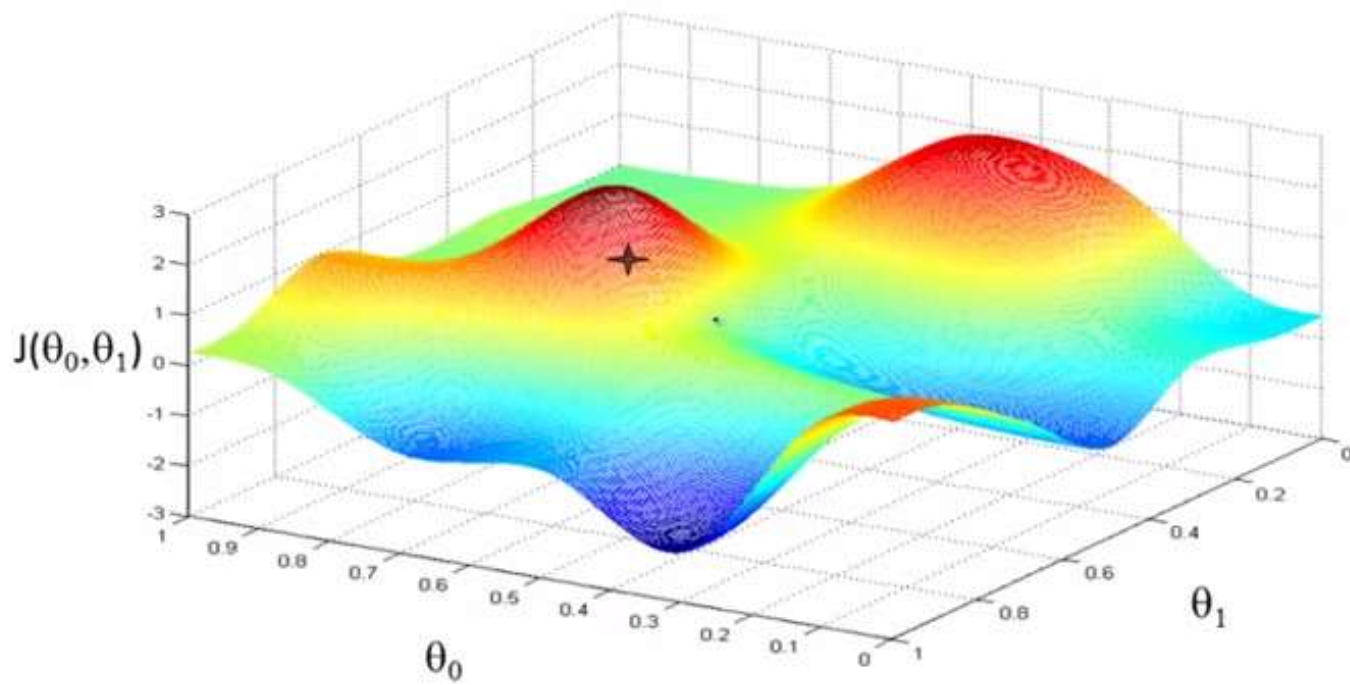
# Gradient Descent



# Gradient Descent

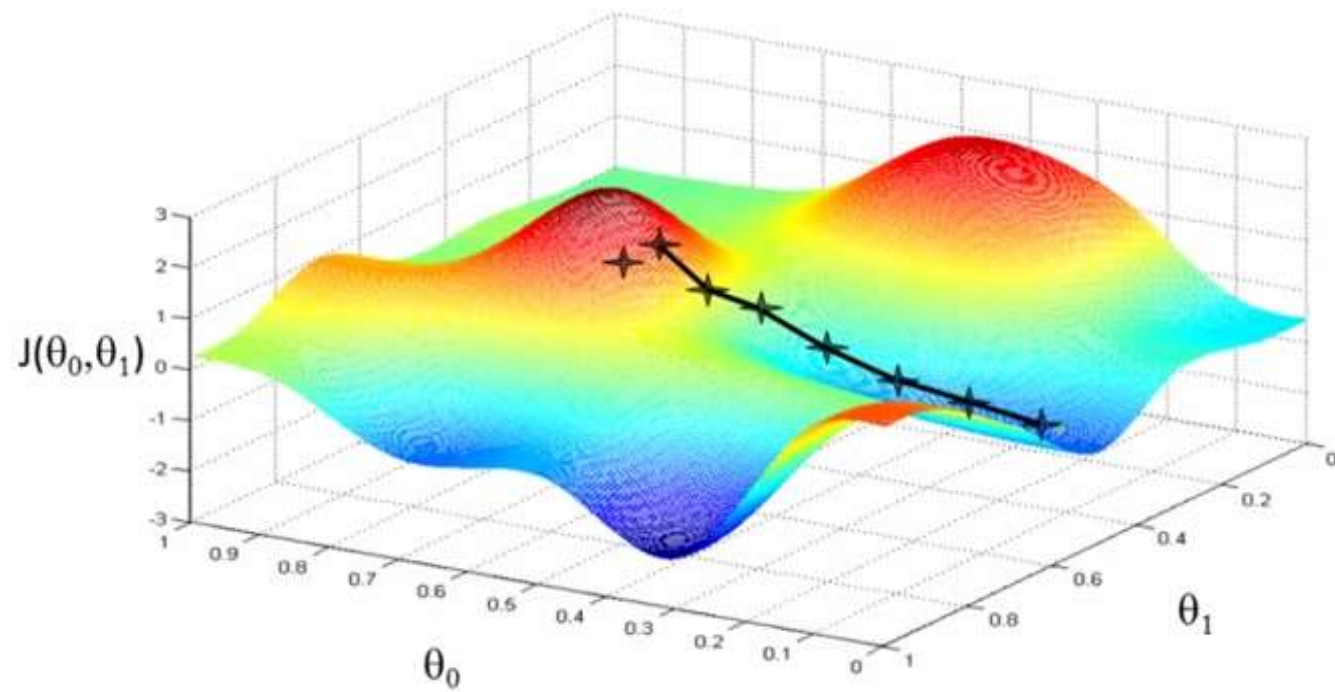


# Gradient Descent





# Gradient Descent





# 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  $j = 1$ )  
}

$\alpha$  = learning rate

Implication of  $\alpha$  = it controls how bigger steps we are taking over gradient descent

**Correct: Simultaneous update**

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

**Incorrect:**

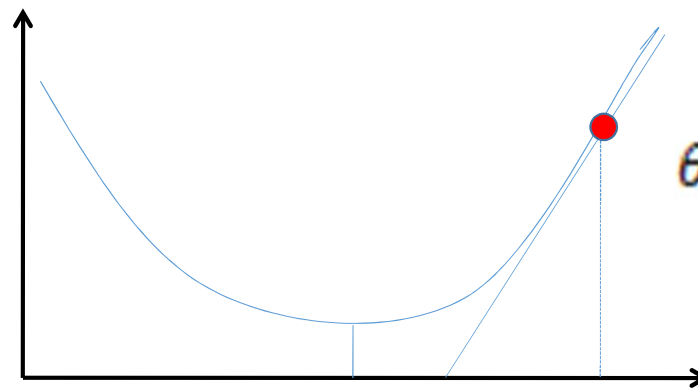
```
temp0 :=  $\theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$   
 $\theta_0 :=$  temp0  
temp1 :=  $\theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$   
 $\theta_1 :=$  temp1
```

# Gradient Descent Algorithm

- Let take a single variable
- we have to minimize  $\min_{\theta_1} J(\theta_1)$  where  $\theta_1 \in \mathbb{R}$
- So the GD algorithm becomes

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

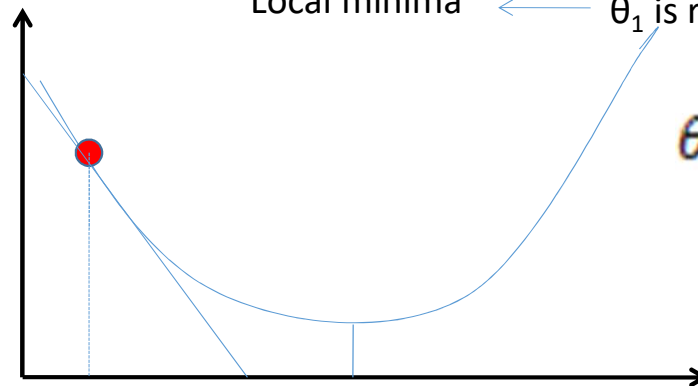
# Gradient Descent Algorithm



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

$$\geq 0$$

(slope is positive)



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

$$\leq 0$$

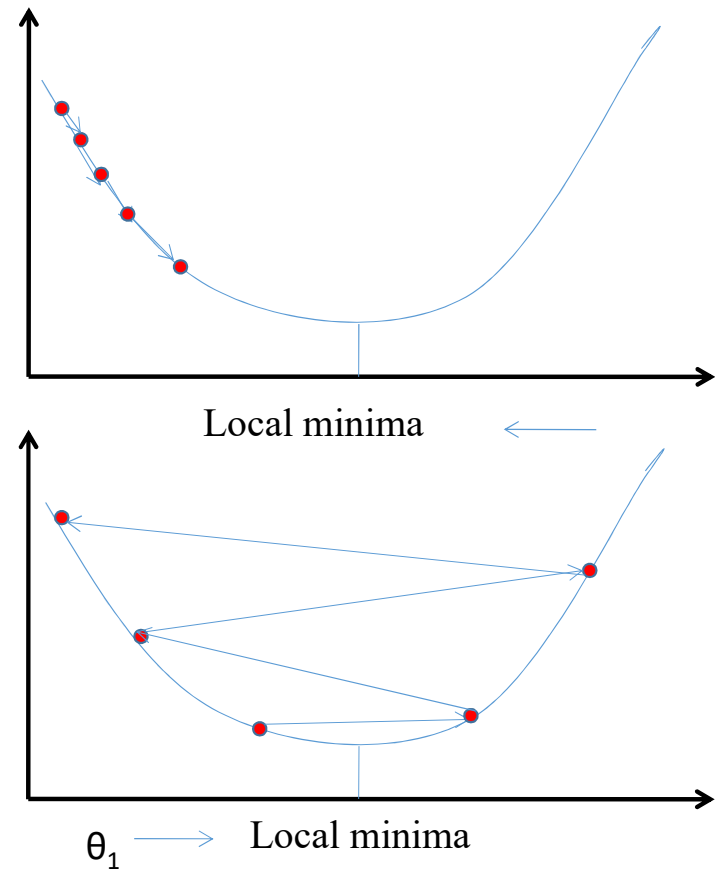
(slope is negative)

# Gradient Descent Algorithm

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

If  $\alpha$  is too small, gradient descent can be slow.

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



# Multivariate Linear Regression

Univariate Hypothesis function:

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

Multivariate Hypothesis function:

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

$$X = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \mathbb{R}^{n+1} \quad \theta^T x = [\theta_0 \quad \theta_1 \quad \cdots \quad \theta_n] \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

where  $x_0 = 1$

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

# Multivariate Gradient Descent

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

Parameters:  $\theta_0, \theta_1, \dots, \theta_n$   $\rightarrow$   $\Theta$  :  $n+1$  dimensional vector

Cost function:

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

$J(\Theta)$

Gradient descent:

Repeat {

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

}

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

$J(\Theta)$

# Multivariate Gradient Descent

## Gradient Descent

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  $\theta_0, \theta_1$ )

}

New algorithm ( $n \geq 1$ ):

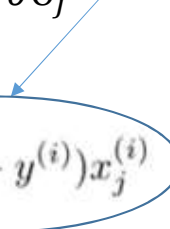
Repeat {

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

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

}

$$\begin{aligned} \theta_0 &:= \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_0^{(i)} \\ \theta_1 &:= \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_1^{(i)} \\ \theta_2 &:= \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)}) x_2^{(i)} \end{aligned}$$

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


# Feature Scaling

## Feature Scaling

Idea: Make sure features are on a similar scale.

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

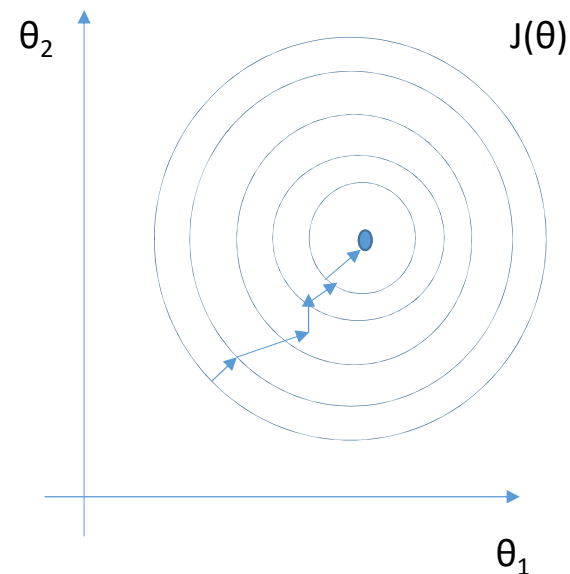
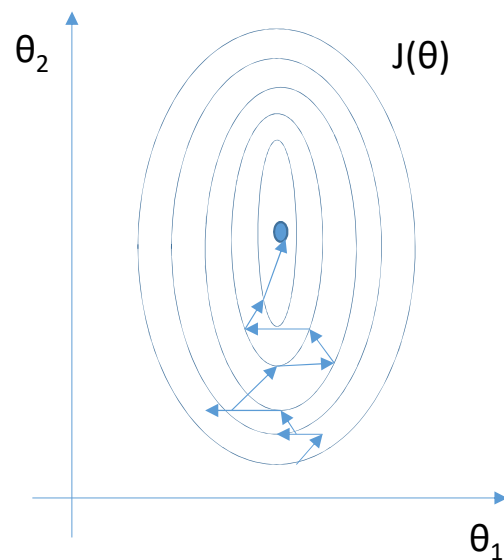
$x_2 = \text{number of bedrooms (1-5)}$

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

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

$$0 \leq x_1 \leq 1$$

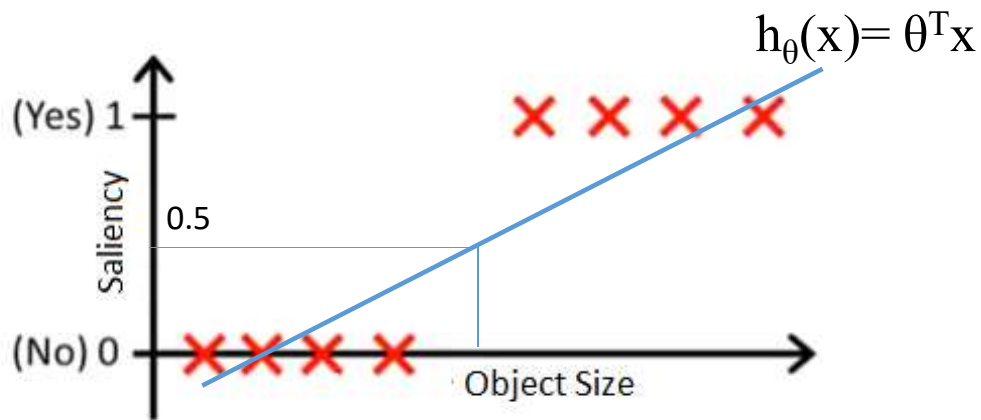
$$0 \leq x_2 \leq 1$$



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



# Logistic Regression: Classification



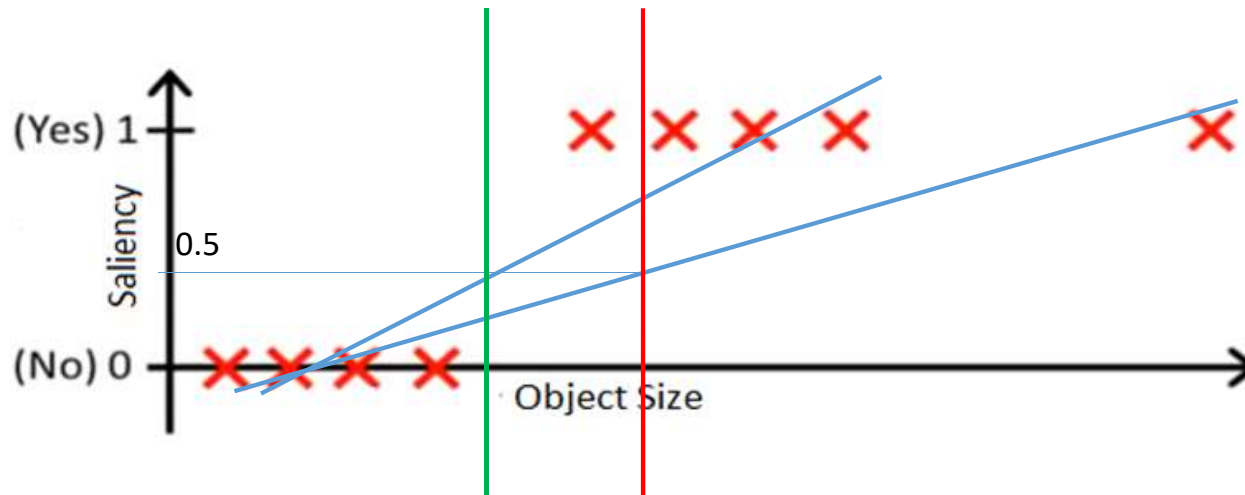
$$h_{\theta}(x) = \theta^T x$$

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”

# Logistic Regression



Linear regression for classification problem is not always good

Classification:  $y = 0$  or  $1$

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

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

# Logistic Regression Model

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

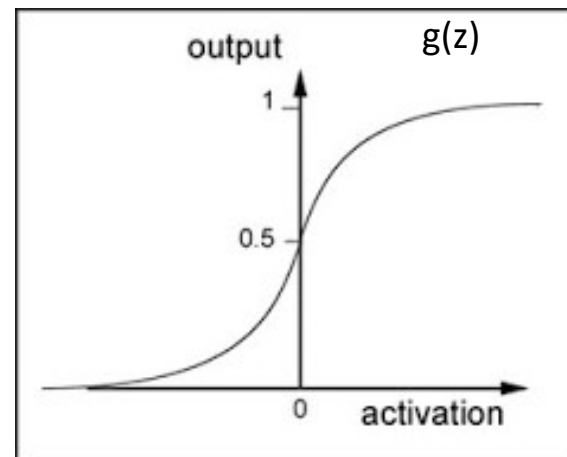
Linear Regression:  $h_{\theta}(x) = \theta^T x$

Logistic Regression:

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

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

Sigmoid Function or Logistic function



# Hypothesis Representation

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

Example: if  $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{Object Size} \end{bmatrix}$

$$h_{\theta}(x) = 0.7$$

There is 70% chance that the object is salient

$$h_{\theta}(x) = p(y=1|x, \Theta)$$

i.e. “probability that  $y=1$ , given  $x$ , parameterized by  $\Theta$ ”

$$p(y=0|x; \Theta) + p(y=1|x; \Theta) = 1$$

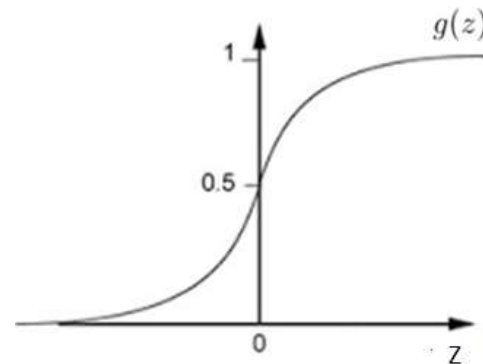
$$p(y=0|x; \Theta) = 1 - p(y=1|x; \Theta)$$

# Decision Boundary

## Logistic regression

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

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



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

i.e.  $\theta^T x \geq 0$

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

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

i.e.  $\theta^T x < 0$

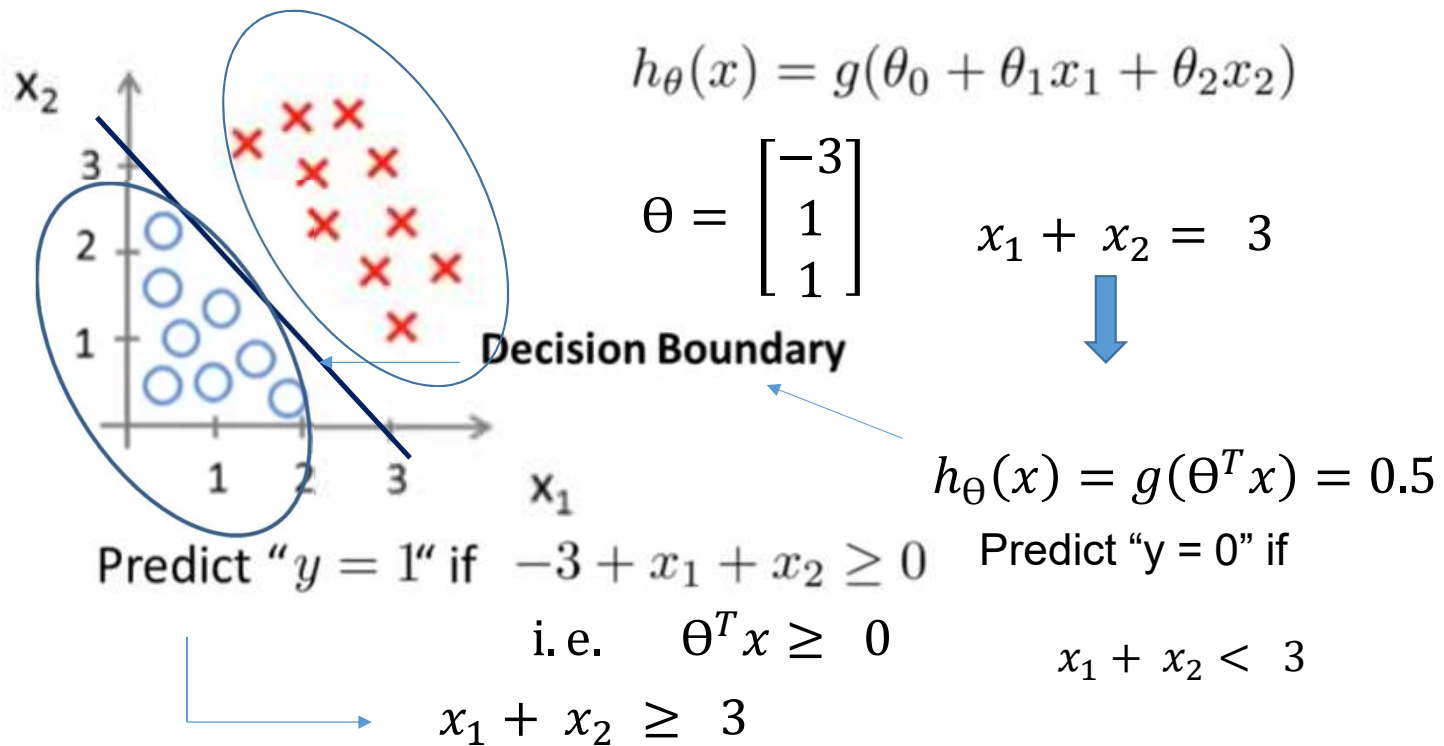
$g(z) \geq 0.5$  when  $z \geq 0$

$$h_{\theta}(x) = g(\theta^T x) \geq 0.5$$

whenever  $\theta^T x \geq 0$

$z$

# Decision Boundary



Decision boundary is a property of hypothesis function **NOT of a data set**

# Non-Linear Decision Boundary

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

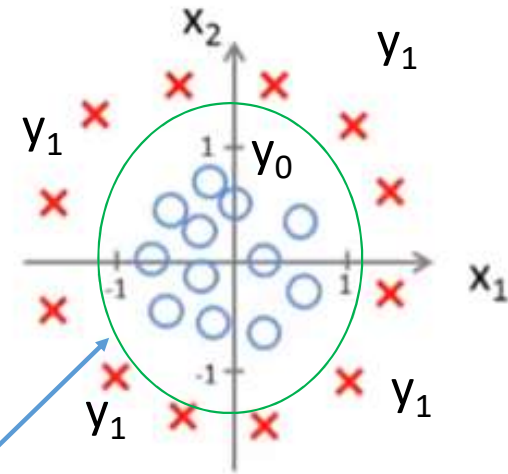
Let  $\theta^T = [-1 \ 0 \ 0 \ 1 \ 1]$

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

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

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

Decision Boundary



Again, decision boundary is a property of hypothesis function **NOT of a data set**

# Cost Function

- Optimization objective of the cost function

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

m examples  $x \in \begin{bmatrix} x_0 \\ x_1 \\ \dots \\ x_n \end{bmatrix} \quad x_0 = 1, y \in \{0, 1\}$

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

How to choose parameters  $\theta$  ?



# Cost Function

## Cost function

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

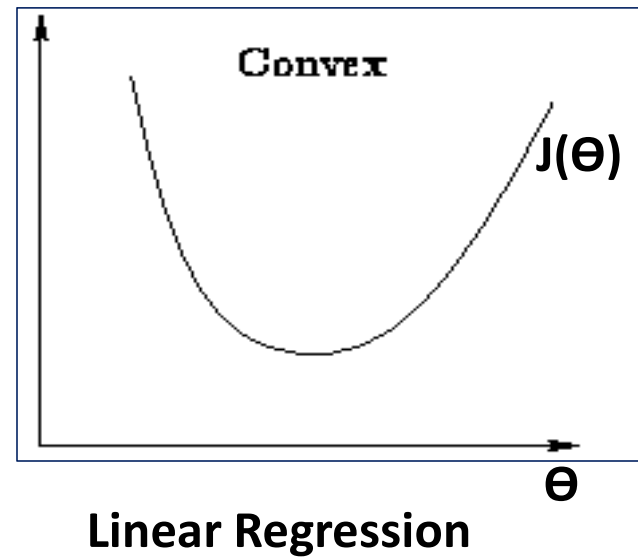
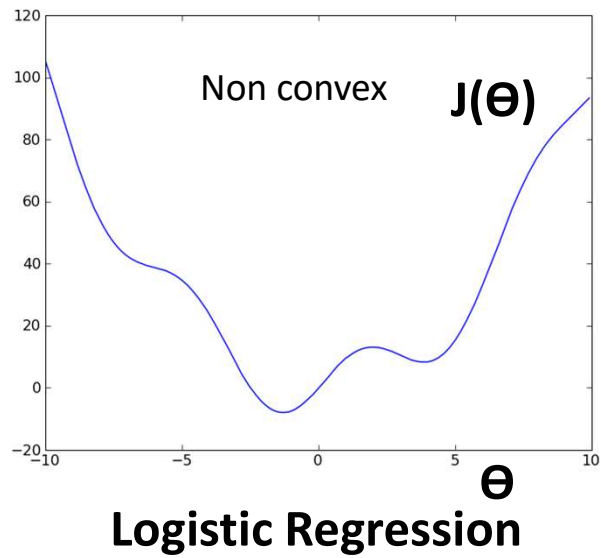
Let,  $\text{Cost}(h_{\theta}(x^{(i)}), y^{(i)}) = \frac{1}{2} (h_{\theta}(x^{(i)}) - y^{(i)})^2$

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

where, for logistic regression

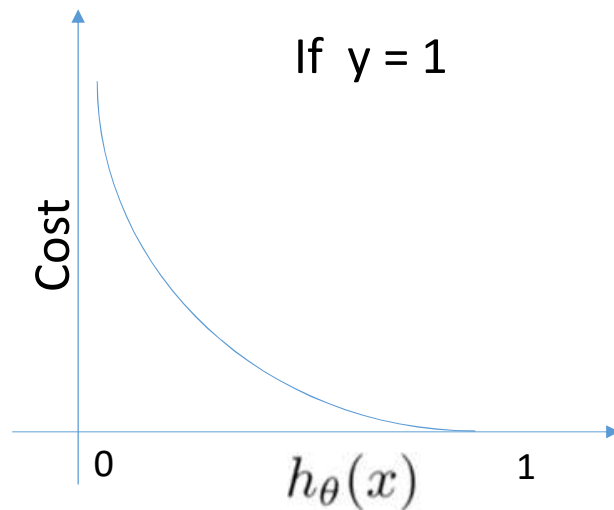
$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$

# Cost Function



# Cost Function: Logistic Regression

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



Cost = 0 if  $y = 1, h_{\theta}(x) = 1$

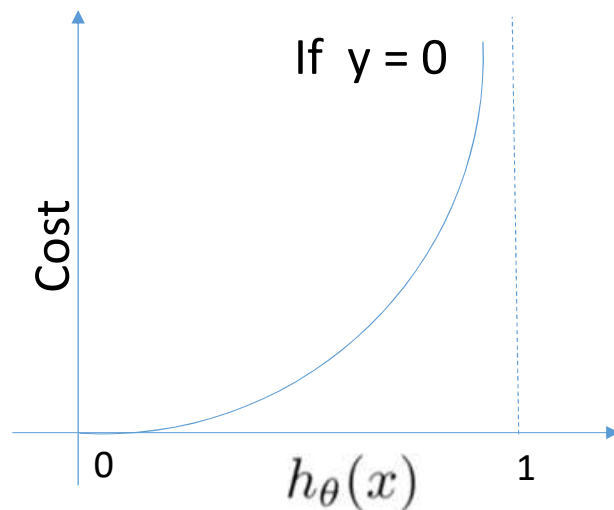
But as  $h_{\theta}(x) \rightarrow 0$

$\text{Cost} \rightarrow \infty$

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.

# Cost Function: Logistic Regression

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



Cost = 0 if  $y=0$ ,  $h_{\theta}(x) = 0$

But as  $h_{\theta}(x) \rightarrow 1$

Cost  $\rightarrow \infty$

Captures intuition that if  $h_{\theta}(x) = 1$ ,  
(predict  $P(y=0|x; \Theta) = 1$ ), but  $y = 0$ ,  
We will penalize learning algorithm  
by a very large cost.

It can be shown that the overall cost function is **convex function and local optimum free**. But details of such convexity analysis is beyond of the scope of this course.

# Cost Function: Logistic Regression

## Logistic regression cost function

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

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

Note:  $y = 0$  or  $1$  always

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

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

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

# Cost Function: Logistic Regression

## Logistic regression cost function

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

## Principle of Maximum Likelihood Estimation

To fit parameters  $\theta$ :      To make a prediction given new  $x$ :

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

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

and get  $\Theta$

For  $p(y=1|x; \Theta)$

How to minimize  $J(\Theta)$  ?

# Cost Function and Gradient Descent

## 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 {

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

}

(simultaneously update all  $\theta_j$ )

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

# Cost Function and Gradient Descent

## 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 {

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

(simultaneously update all  $\theta_j$ )

}

For Linear Regression:

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

For Logistic Regression:

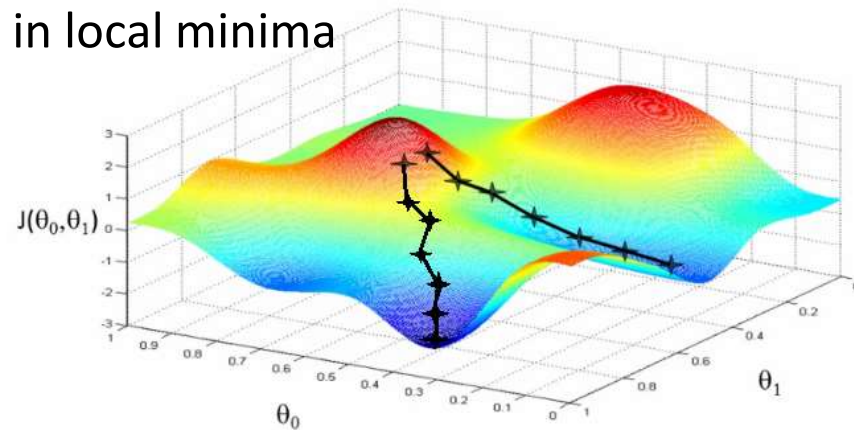
$$h_{\theta}(x) = \frac{1}{1 + e^{-\theta^T x}}$$

Algorithm looks identical to linear regression!



# Gradient descent optimization

- Problems:
  - Choosing step size
    - too small  $\rightarrow$  convergence is slow and inefficient
    - too large  $\rightarrow$  may not converge
  - Can get stuck on “flat” areas of function
  - Easily trapped in local minima



# Gradient Descent

- Gradient Descent is a popular optimization technique in Machine Learning and Deep Learning, and it can be used with most, if not all, of the learning algorithms.
- A gradient is the slope of a function. It measures the degree of change of a variable in response to the changes of another variable.
- Mathematically, Gradient Descent is a convex function whose output is the partial derivative of a set of parameters of its inputs. The greater the gradient, the steeper the slope.
- Starting from an initial value, Gradient Descent is run iteratively to find the optimal values of the parameters to find the minimum possible value of the given cost function.

# Types of Gradient Descent

- Typically, there are three types of Gradient Descent:
  - Batch Gradient Descent
  - Stochastic Gradient Descent
  - Mini-batch Gradient Descent

# Stochastic Gradient Descent (SGD)

- The word ‘*stochastic*’ means a system or a process that is linked with a random probability.
- Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration.
- In Gradient Descent, there is a term called “batch” which denotes the total number of samples from a dataset that is used for calculating the gradient for each iteration.
- In typical Gradient Descent optimization, like Batch Gradient Descent, the batch is taken to be the whole dataset.
- Although, using the whole dataset is really useful for getting to the minima in a less noisy and less random manner, but the problem arises when our datasets gets big.
- Suppose, you have a million samples in your dataset, so if you use a typical Gradient Descent optimization technique, you will have to use all of the one million samples for completing one iteration while performing the Gradient Descent, and it has to be done for every iteration until the minima is reached. Hence, it becomes computationally very expensive to perform.

# Stochastic Gradient Descent

- This problem is solved by Stochastic Gradient Descent.
- In SGD, it uses only a single sample, i.e., a batch size of one, to perform each iteration.
- The sample is randomly shuffled and selected for performing the iteration.

*Normal Gradient Descent*

$$\theta_j = \theta_j - \alpha \sum_{i=1}^m (\hat{y}^i - y^i) x_j^i$$

*Stochastic Gradient Descent*

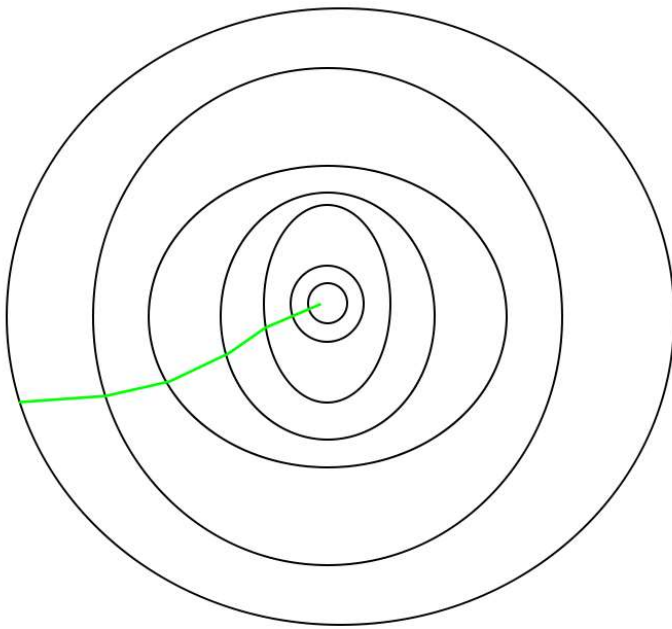
*for  $i$  in range ( $m$ ):*

$$\theta_j = \theta_j - \alpha (\hat{y}^i - y^i) x_j^i$$

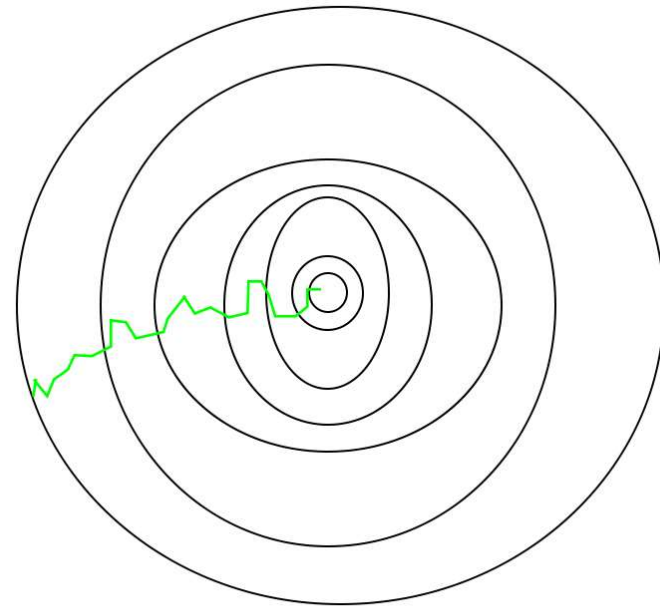
## Stochastic Gradient Descent vs Typical Gradient Descent

- So, in SGD, we find out the gradient of the cost function of a single example at each iteration instead of the sum of the gradient of the cost function of all the examples. Less expensive than normal GD.
- In SGD, since only one sample from the dataset is chosen at random for each iteration, the path taken by the algorithm to reach the minima is usually noisier than your typical Gradient Descent algorithm.
- But that doesn't matter all that much because the path taken by the algorithm does not matter, as long as we reach the minima and with significantly shorter training time.

# Stochastic Gradient Descent vs Typical Gradient Descent



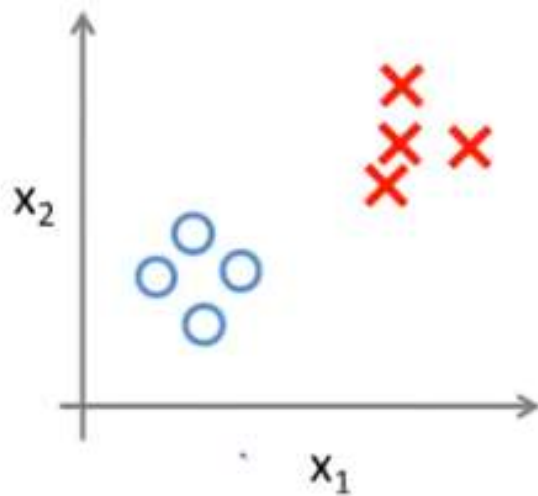
Path taken by Typical Gradient Descent



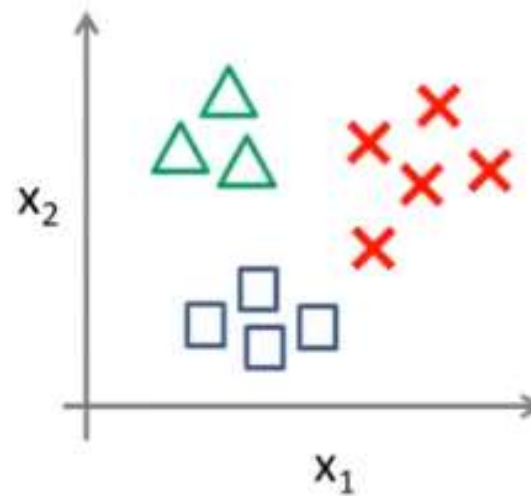
Path taken by Stochastic Gradient Descent

# Multi Class Classification

Binary classification:



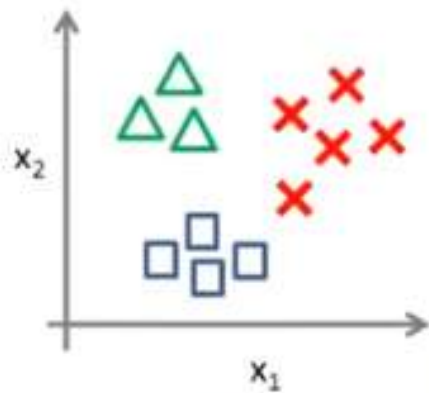
Multi-class classification:








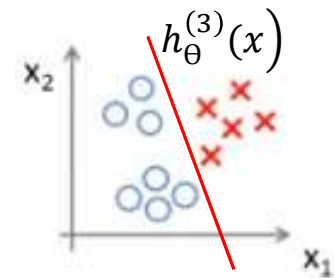
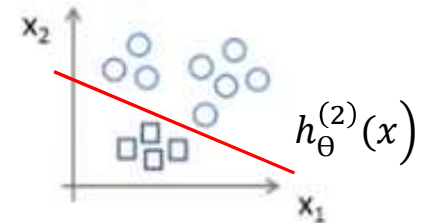
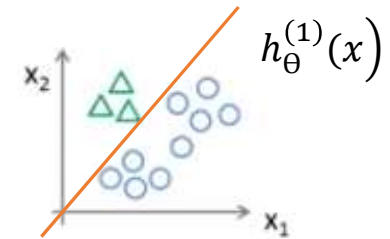
# One vs. All (One vs. Rest)

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



Class 1:   
Class 2:   
Class 3: 

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



## One vs. All (One vs. Rest)

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

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

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

# Stochastic gradient descent

Stochastic (definition):

1. involving a random variable
2. involving chance or probability; probabilistic

# Stochastic gradient descent

- Application to training a machine learning model:
  1. Choose one sample from training set
  2. Calculate loss function for that single sample
  3. Calculate gradient from loss function
  4. Update model parameters a single step based on gradient and learning rate
  5. Repeat from 1) until stopping criterion is satisfied
- Typically entire training set is processed multiple times before stopping.
- Order in which samples are processed can be fixed or random.