Machine Learning (ML) with Python

Linear Model

Dr. Aeshah Alsughayyir

Collage of Computer Science and Engineering

Taibah University

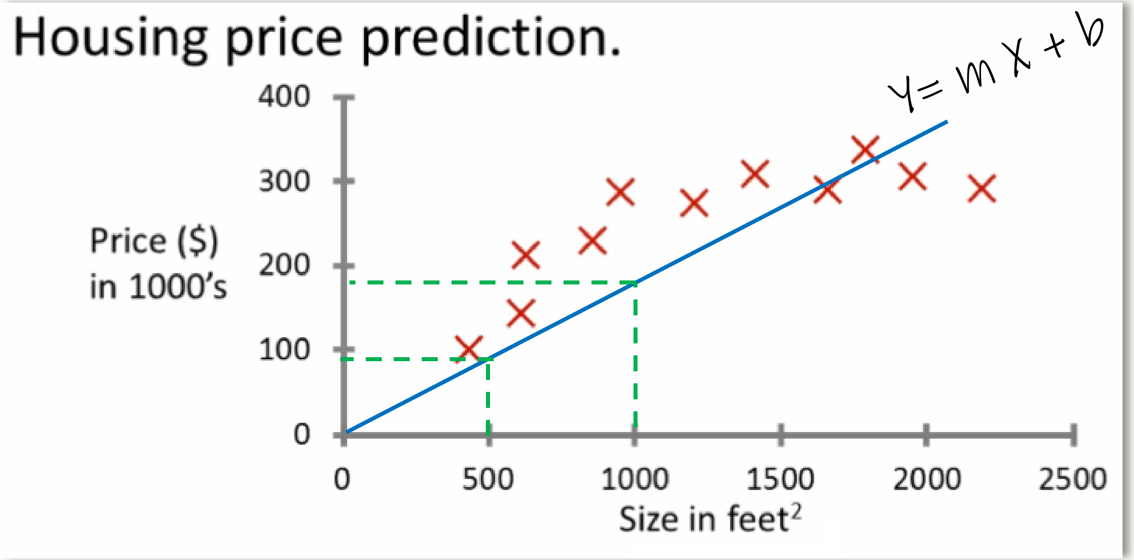
2021-2022

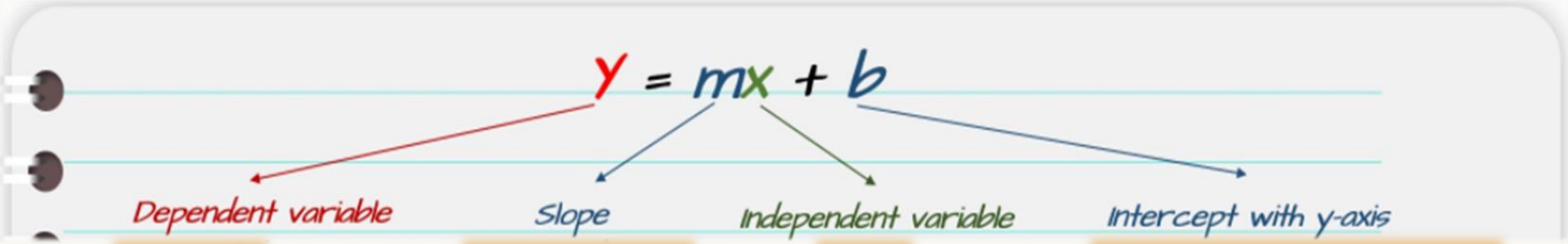
Linear Model

- Model Representation and Hypothesis
- Cost Function of Linear Regression
- Gradient Descent

Motivation Example

Living area ($feet^2$)	Price (1000$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮





Price $ | Price/Size | Size | **Zero** in the above case

$$Slop(m) = \frac{Price}{Size} = \frac{y2 - y1}{x2 - x1} = \frac{190 - 90}{1000 - 500} = 0.2$$

Model Representation and Hypothesis

Notations

- m: Number of training examples
- x's: Input variables / features
- y's: Output / Target variables
- (x, y): One training example
- $(x^{(i)}, y^{(i)})$: i^{th} training example

x's $\quad$ y's

Living area (feet^2)	Price (1000\$s)
2104	400
1600	330
2400	369
1416	232
3000	540
⋮	⋮

(x,y)

m

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

Model Representation and Hypothesis (Cont.)

- If we could find the equation of line **y = mx+b** that we use to fit the data represented by the blue inclined line (see the motivation example), then we can easily find the model that can predict the housing prices for any given area.

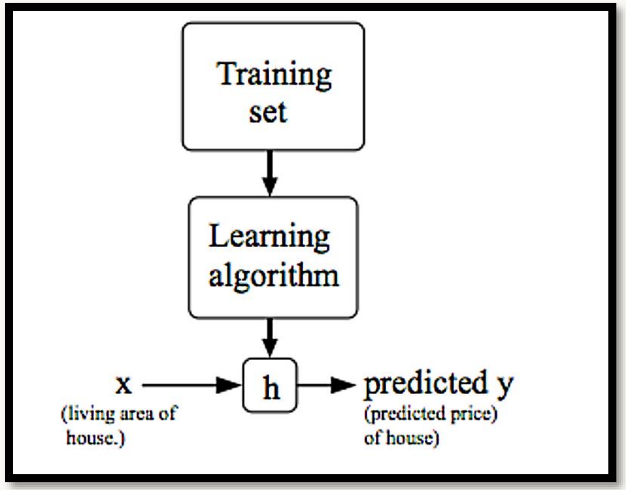
  For example: **What would be the best-estimated price for area 2450 feet square?**

- In machine learning lingo, function **y = mx+b** is also called a **hypothesis function** where ***m*** and ***b*** can be represented by $\boldsymbol{theta_0}$ and $\boldsymbol{theta_1}$ respectively. $\boldsymbol{theta_0}$ is called a ***bias term or intercept,*** and $theta_0$, $theta_1$,.. are also called ***coefficient or weights***.

The job of the learning algorithm would be to produce a hypothesis (***h*),** such that it takes the size of house as input and predicts its price

Model Representation and Hypothesis (Cont.)

Hypothesis is the function that is to be learnt by the learning algorithm by the training process for making the **predictions** about the **unseen** data.



- For example, for **Linear Regression of One Variable** or **Univariate Linear Regression**, the hypothesis, ***h*** is given by:

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

- Where
  - $h_\theta(x)$ is the hypothesis function, also denoted as $h(x)$ sometimes
  - x is the independent variable
  - θ_0 and θ_1 are the parameters of the linear regression that need to be learnt

Different values for these parameters will give different hypothesis function based on the values of slope and intercepts.

Cost Function of Linear Regression

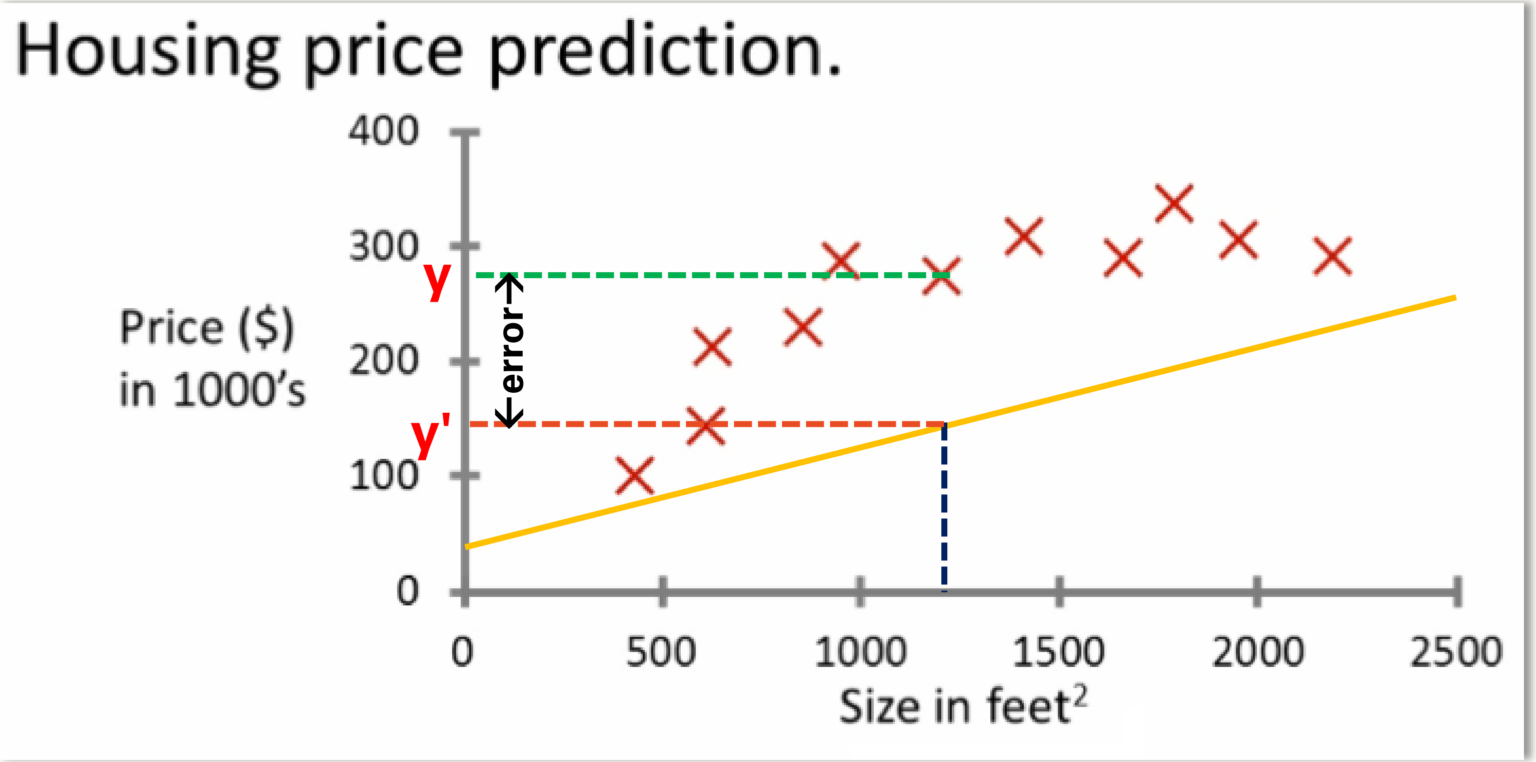
- Once the parameter values, i.e., **intercept** and $\boldsymbol{theta_1}$ are randomly initialized:
  - the hypothesis function is ready for prediction,
  - then the **error** (|**predicted value** – **actual value**|) is calculated to check whether the randomly initialized parameter is giving the right prediction or not.
  - Repeat{
    - If (the error is too high) then
      - the algorithm updates the parameters with new values

    }

The algorithm continues this process **until the error is <u>minimized</u>**

To **minimize the error,** we have a special algorithm called *Gradient Descent* but before that, we are going to understand <u>what **Cost Function** is and how it works?</u>

Cost Function of Linear Regression (Cont.)



$$cost = \frac{1}{m}\sum_{i=1}^{m}(y' - y)$$

Mean Square Error

$$MSE = \frac{1}{2m}\sum_{i=1}^{m}(y' - y)^2$$

Root Mean Square Error

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y' - y)^2}$$

Mean Absolute Error

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|y' - y|$$

Cost Function of Linear Regression (Cont.)

So, **cost function** is defined as follows,

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

Predicted by ***h*** — Actual true values

And **learning objective is to minimize the cost function** i.e.

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

Cost Function of Linear Regression (Cont.)

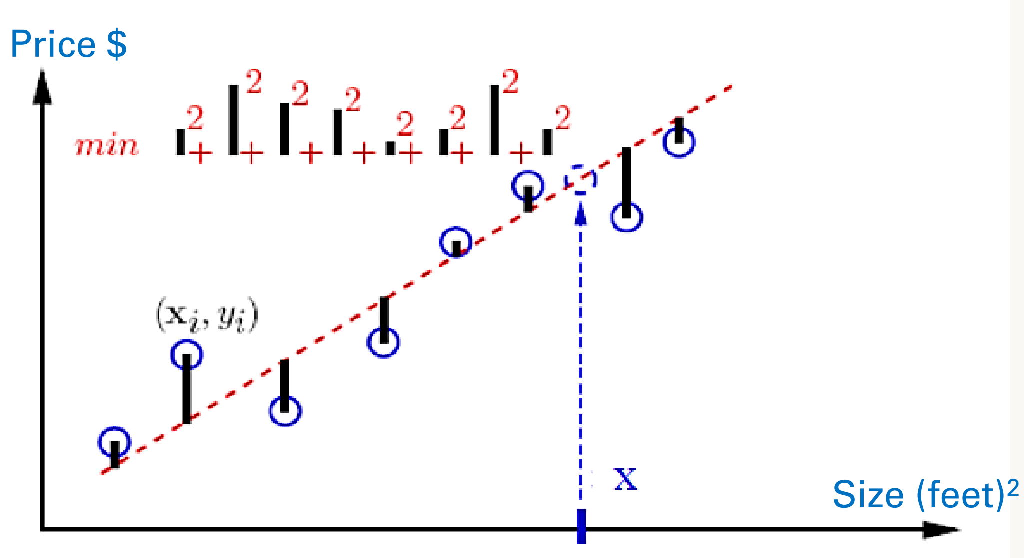
- **Linear regression** finds the parameters *(weights)* that minimize the *mean squared error* between predictions and the true regression targets *(y)*, on the training set.
  - i.e., choose θ_0 and θ_1 so that $h(x)$ is close to y for the training examples (x, y).

- The *mean squared error* is the sum of the squared differences between the predictions and the true values.

- This can be mathematically represented as,

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

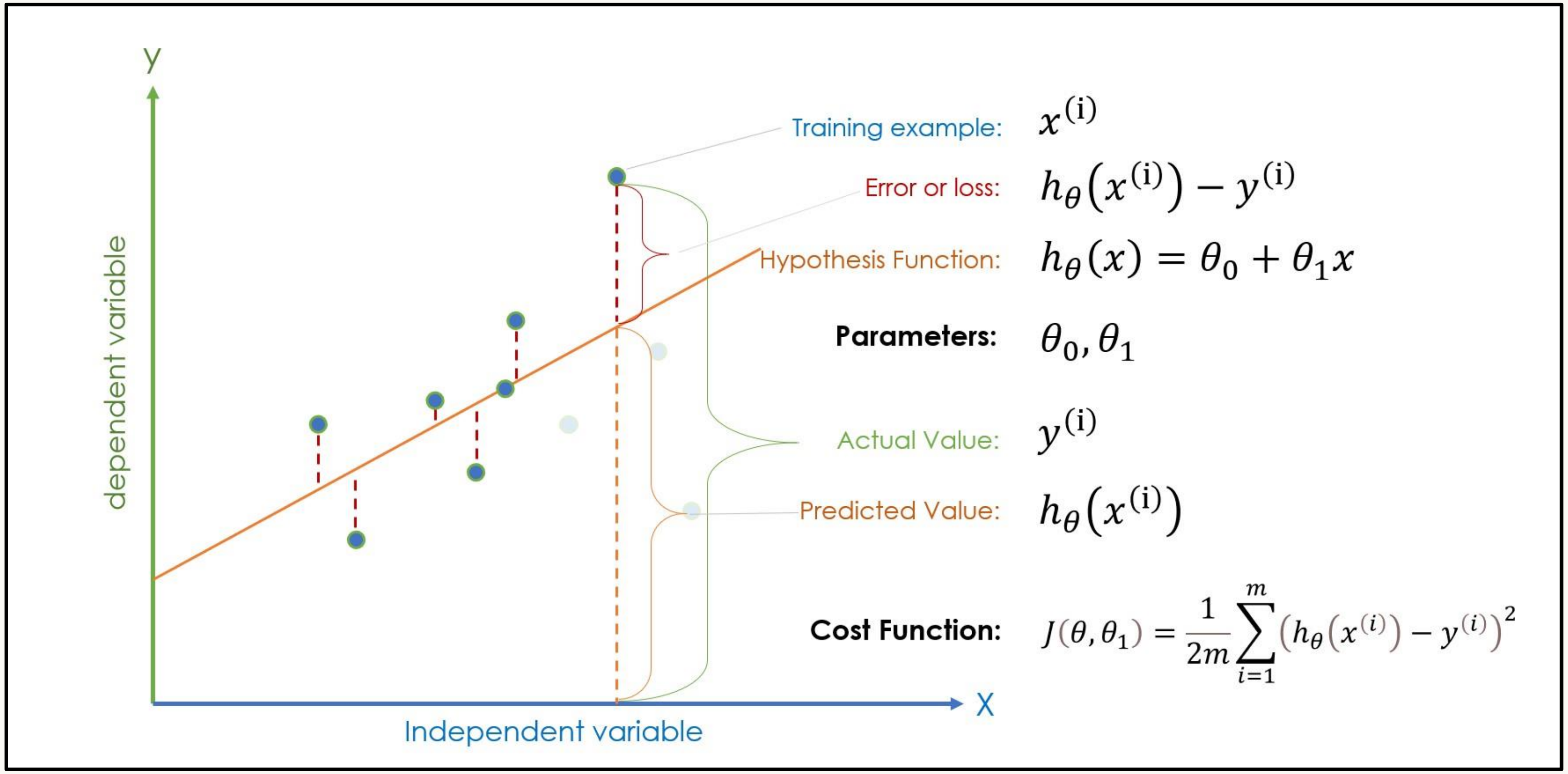
- Where
  - $h_\theta(x^{(i)}) = \theta_0 + \theta_1 \, x^{(i)}$
  - $(x^{(i)}, y^{(i)})$ is the i^{th} training data
  - m is the number of training example



Fit model by *minimizing mean of squared errors*

Cost Function of Linear Regression (Cont.)

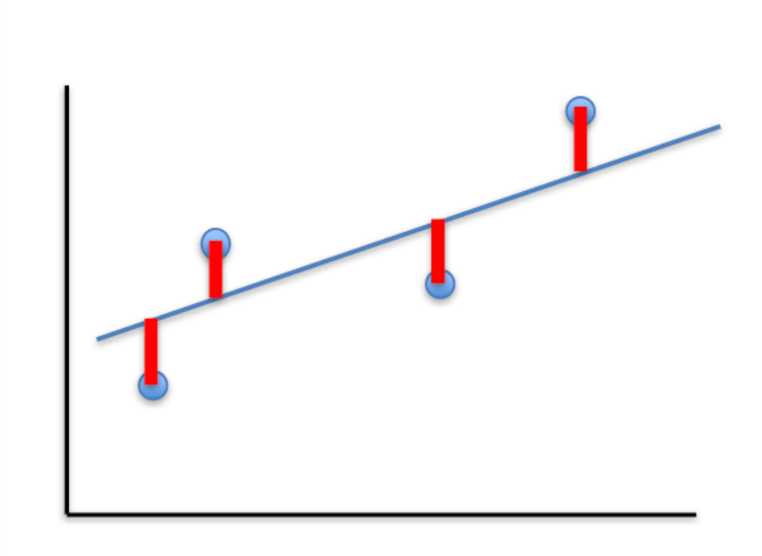
Terminologies:



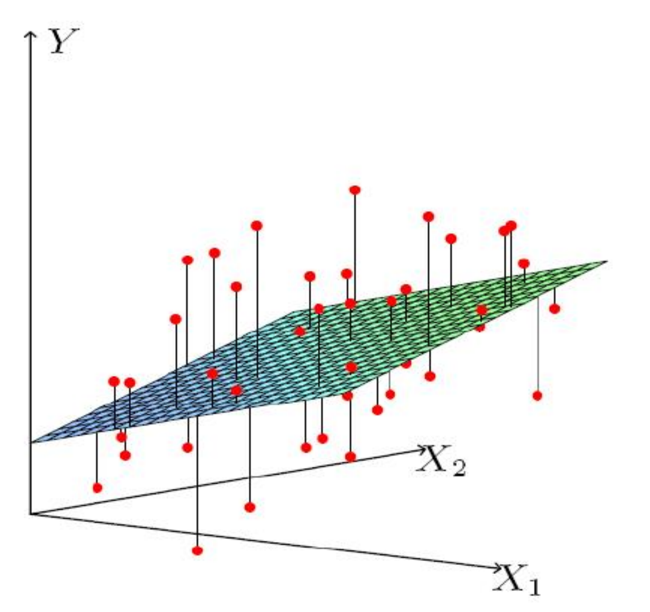
Note**: (i) in the equation represents the ith training example, not the power.**

Cost Function of Linear Regression (Cont.)

One Feature



two Features



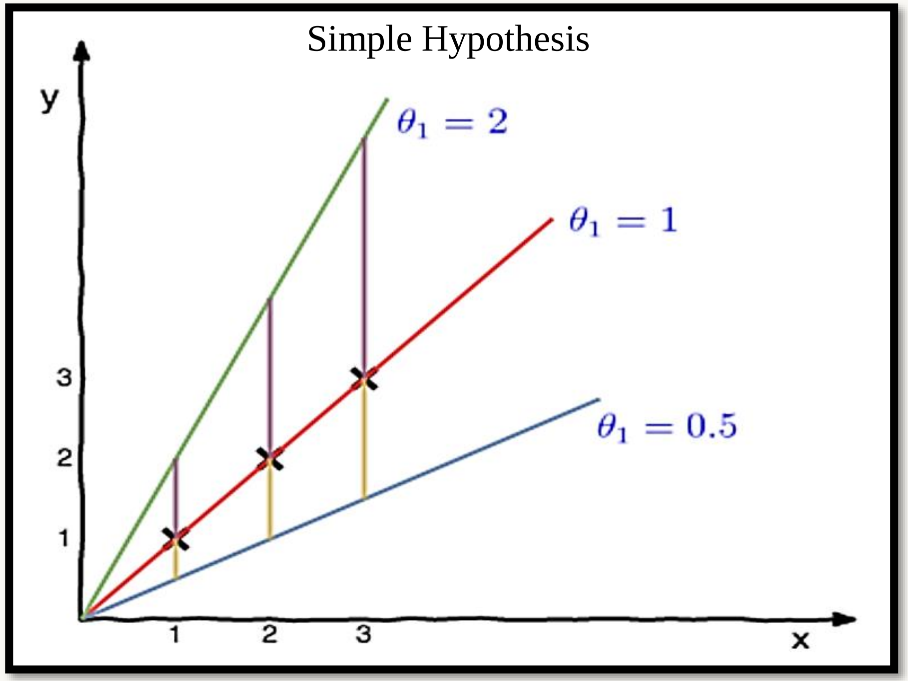
$$minimize_{\theta_0,\theta_1} \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

Cost function visualization

- Consider a simple case of hypothesis **by setting** $\theta_0=0$, then h becomes : $h_\theta(x)=\theta_1 x$
- Each value of θ_1 corresponds to a different hypothesis as it is the **slope** of the line.

which corresponds to different lines passing through the **origin** as shown in plots below as **y-intercept** i.e., θ_0 is nulled out.

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



Calculating each value of θ_1 corresponds to a different hypothesizes:

At θ_1=2, $J(2) = \frac{1}{2*3}(1^2 + 2^2 + 3^2) = \frac{14}{6} = 2.33$

At θ_1=1, $J(1) = \frac{1}{2*3}(0^2 + 0^2 + 0^2) = 0$

At θ_1=0.5, $J(0.5) = \frac{1}{2*3}(0.5^2 + 1^2 + 1.5^2) = 0.58$

Cost function visualization (Cont.)

What is the optimal value of θ_1 that minimizes $J(\theta_1)$?

It is clear that best value for $\boldsymbol{\theta_1 = 1}$ as $\boldsymbol{J(\theta_1) = 0}$, which is the minimum.

How to find the best value for θ_1 ?

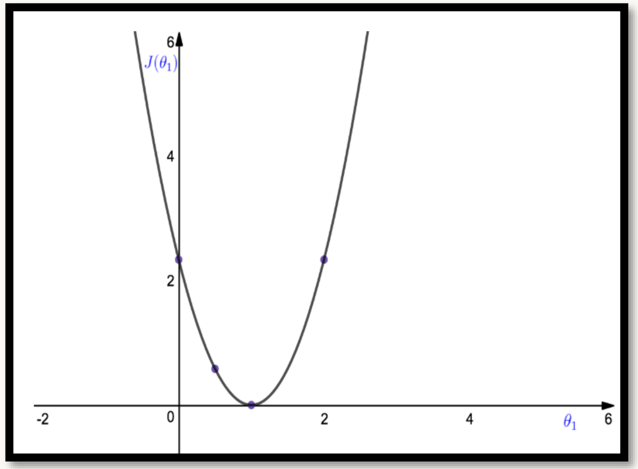
Plotting ?? Not practical specially in high dimensions!

<u>The solution :</u>

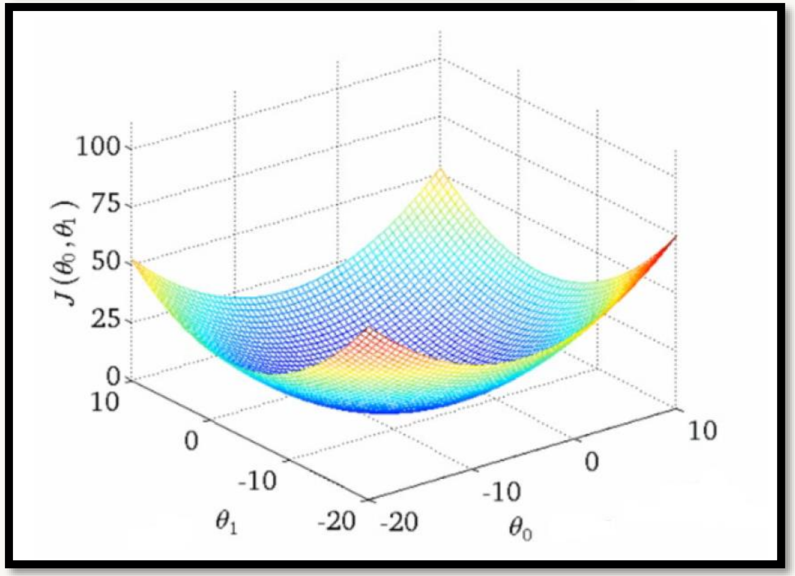
1. <u>Analytical solution</u>: Not applicable for large datasets
2. <u>Numerical solution</u>: e.g., Gradient descent.

Cost function visualization (Cont.)

- What if we plot our cost function with respect to the value of θ_1 :
  - θ_1 vs $J(\theta_1)$

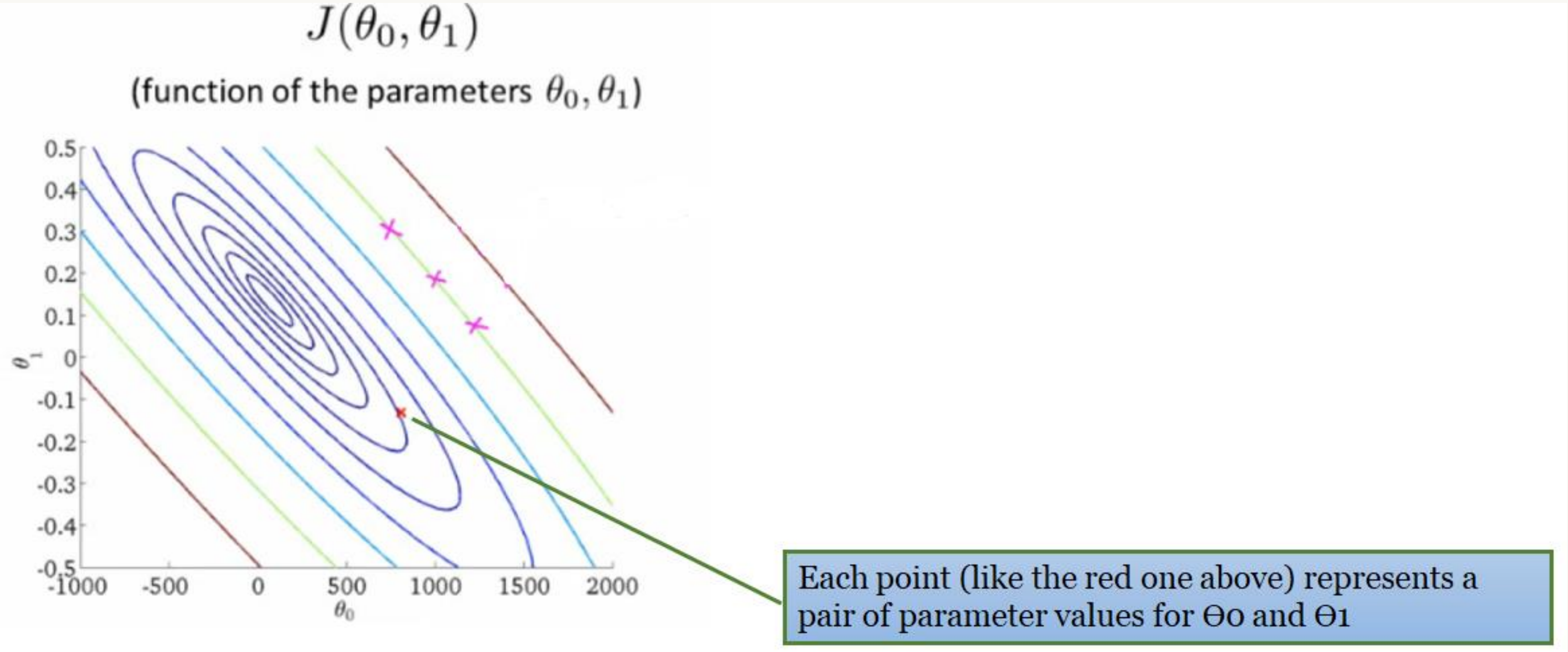


- What if we plot our cost function with respect to the values of both θ_1 and θ_0:
  - *Plot becomes a bit more complicated*
  - *Generates a 3D surface plot where axis are:*
    - $X = \theta_1$
    - $Z = \theta_0$
    - $Y = J(\theta_0, \theta_1)$



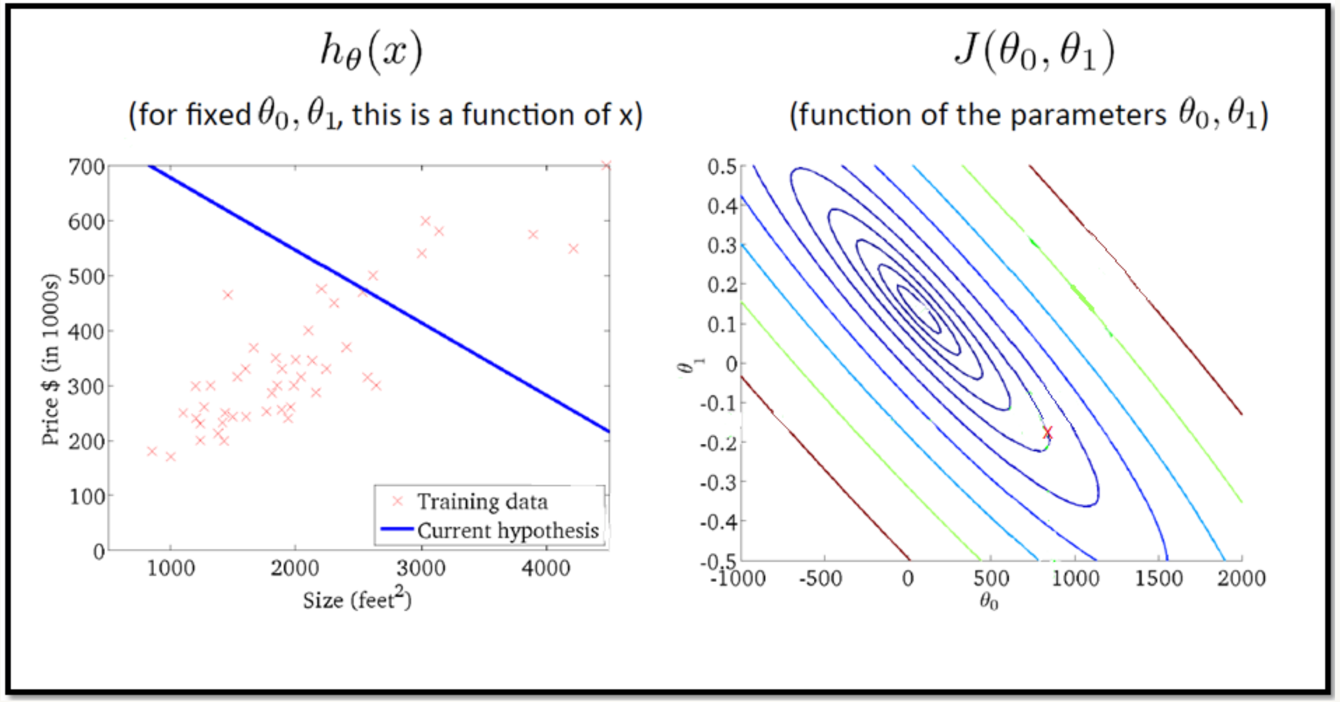
Cost function visualization (Cont.)

- The following is a contour plot *of the cost function* where:
  - Ellipses in different color
  - Each color shows the same value of $J(\theta_0, \theta_1)$
  - Imagine a bowl-shape function coming out of the screen, so the middle is the concentric circle.



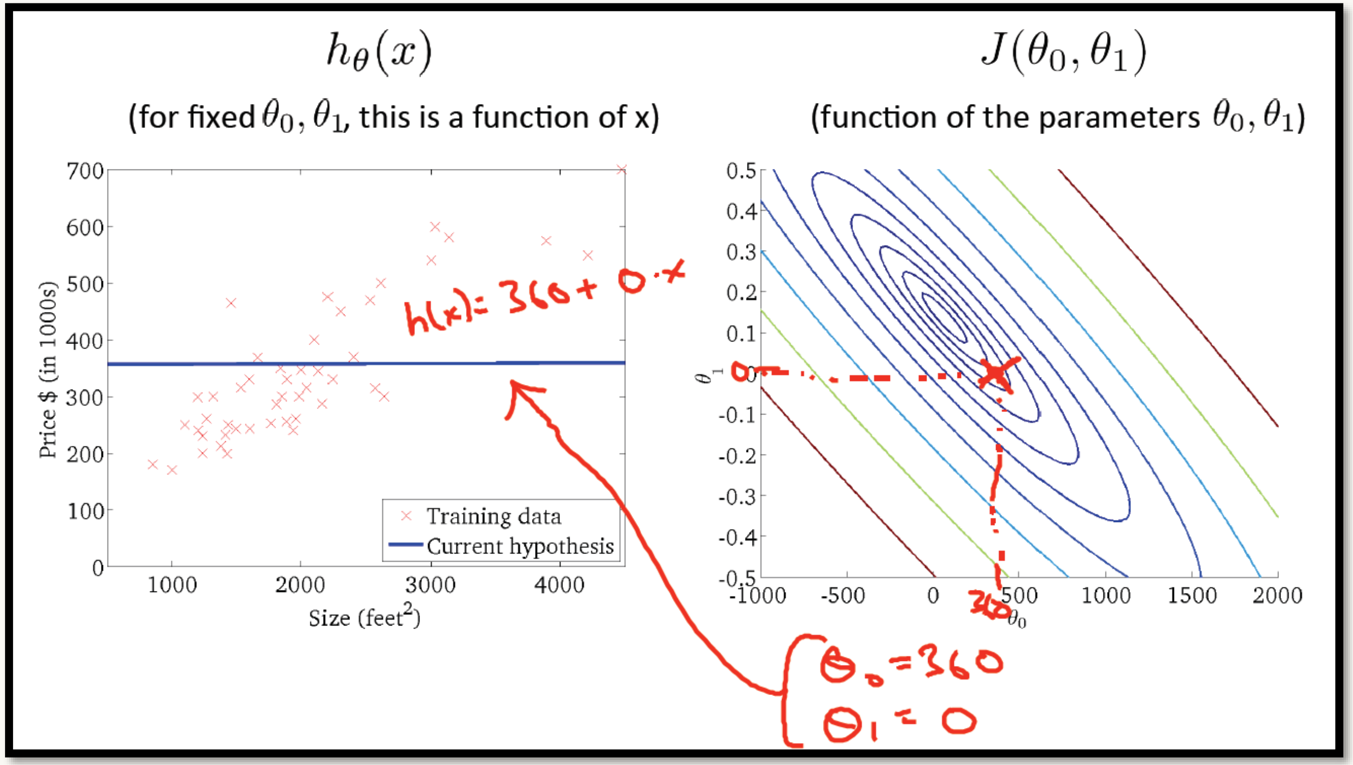
Cost function visualization (Cont.)

- Our example here put the values at
  - θ_0 = ~800
  - θ_1 = ~-0.15
- Not a good fit
  - i.e. these parameters give a value on our contour plot far from the center



Cost function visualization (Cont.)

- If we have
  - θ_0 = ~360
  - θ_1 = 0
  - This gives a better hypothesis, but still not great - not in the center of the countour plot

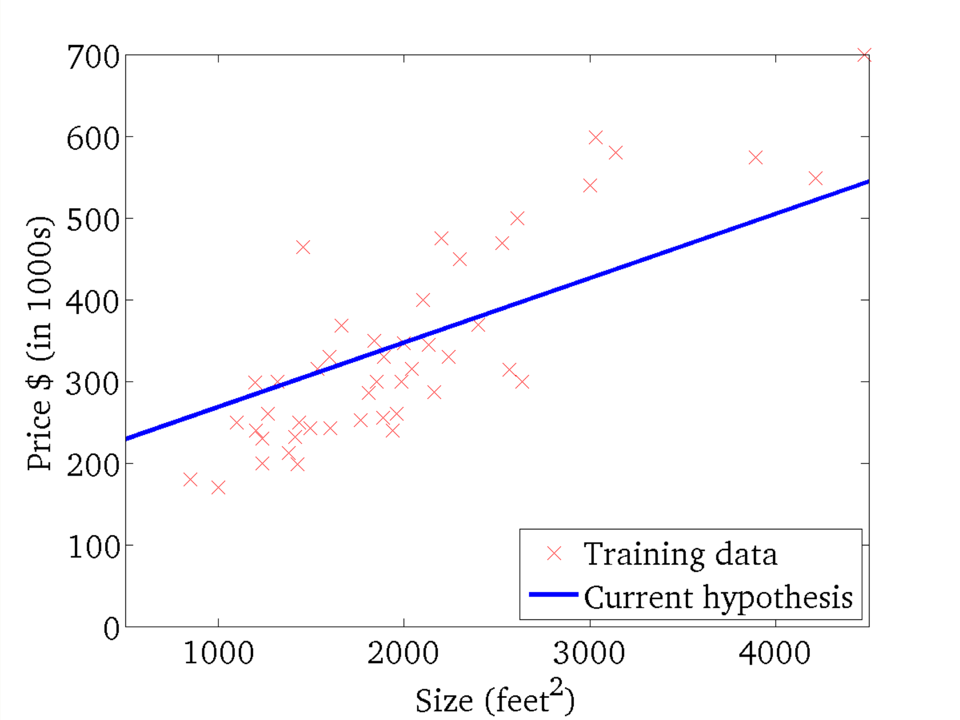


Cost function visualization (Cont.)

The best fit…

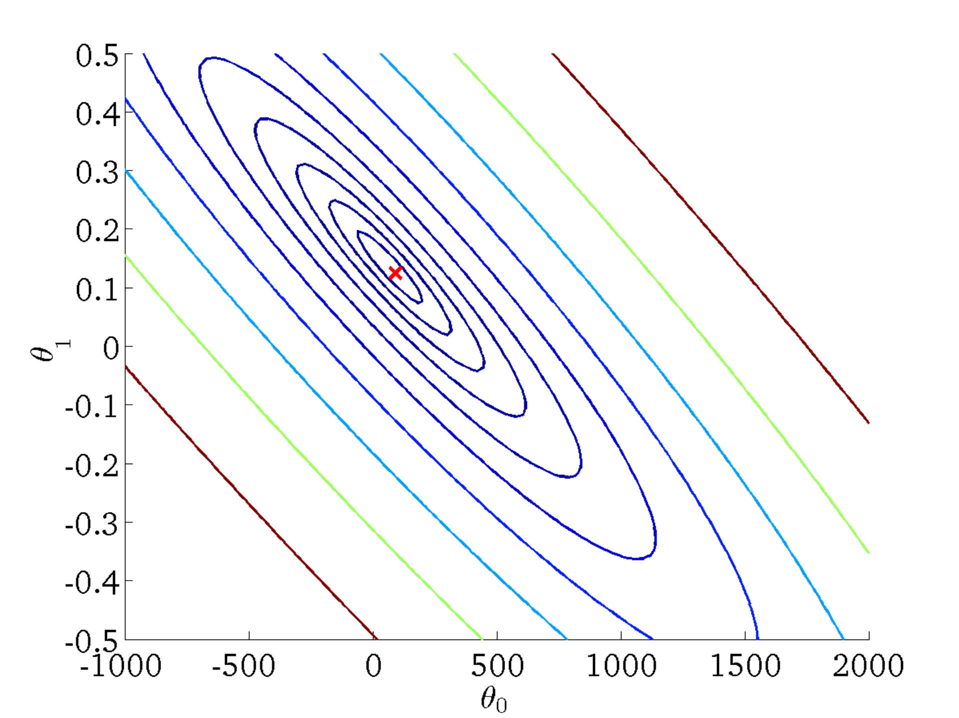
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



Next…

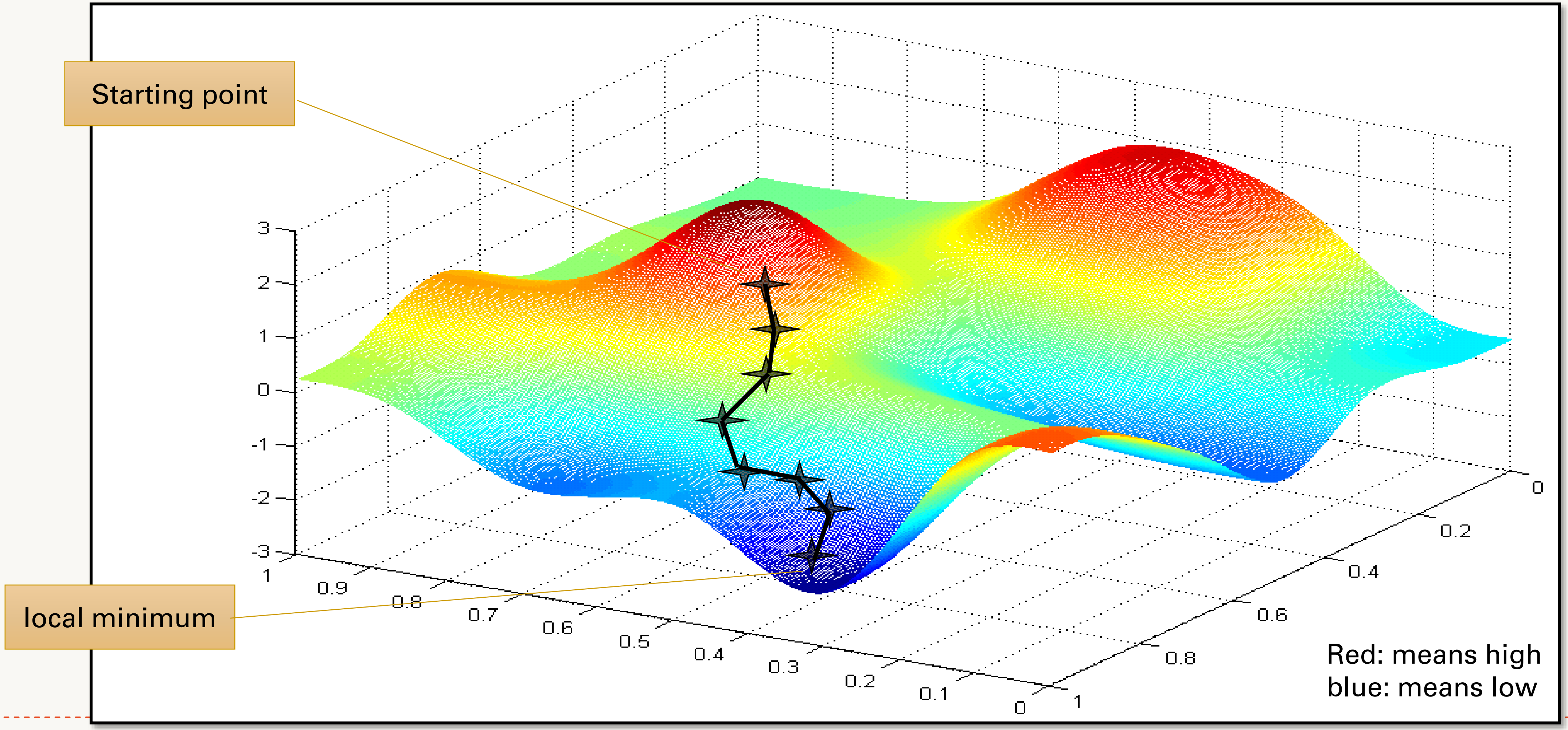
Gradient descent

Motivation


To reach the lowest point
*We should move the ball **step** by **step** in the gradient direction*

The idea of gradient descent.

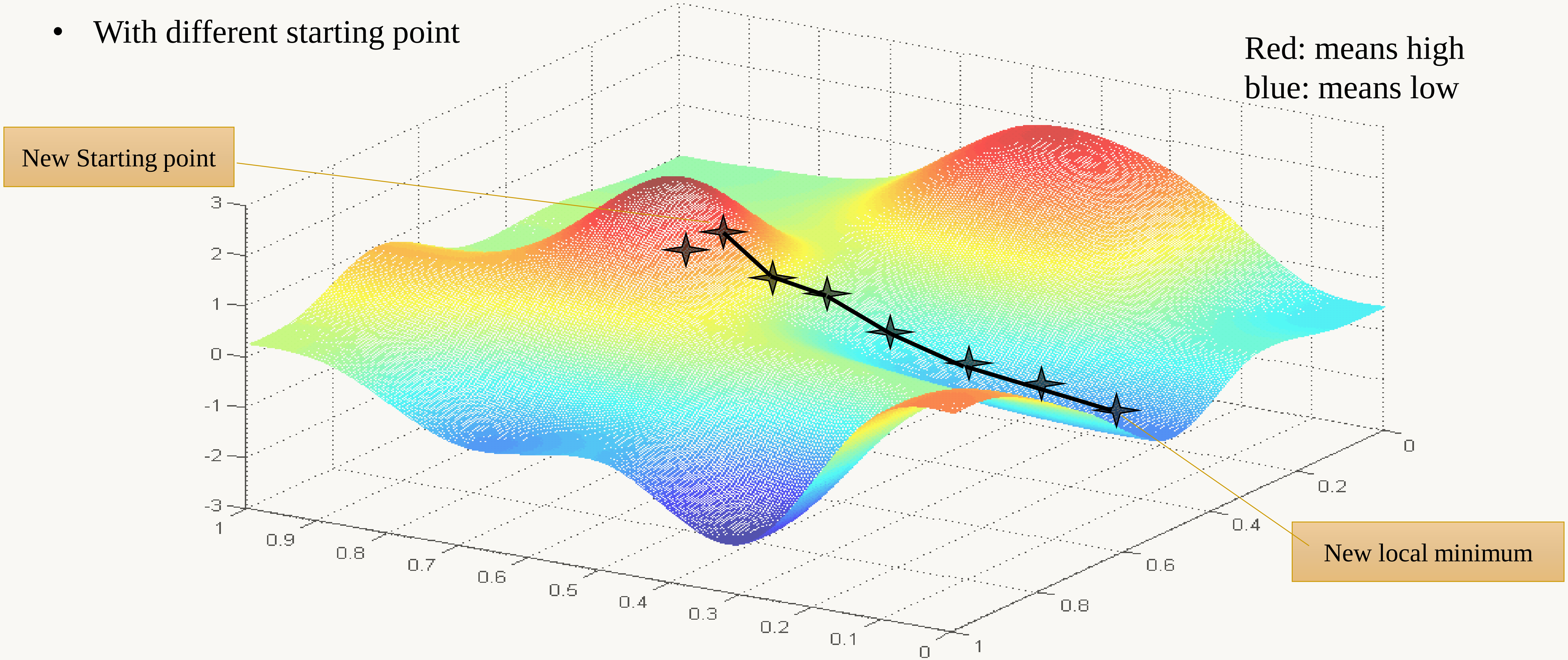
- Imagine that this is a landscape of grassy park, and you want to go to the lowest point in the park as rapidly as possible …



The idea of gradient descent (Cont.)

- With different starting point

Red: means high
blue: means low



What is Gradient Descent?

- Have some function $J(\theta_0, \theta_1)$
- The objective is $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

Gradient descent is an **iterative optimization algorithm** for finding the minimum of a function.

There are **two basic steps** involved in this algorithm:

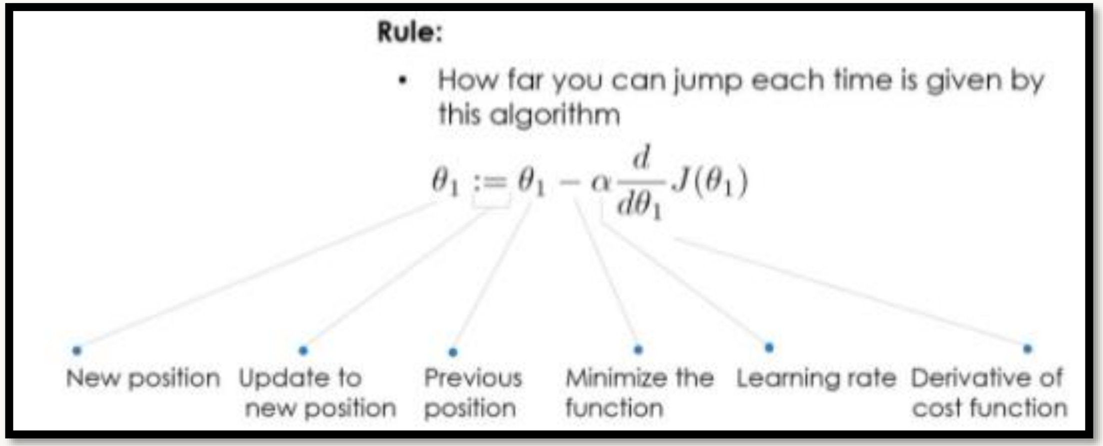
- Start with some random values of θ_0, θ_1 .
- Keep changing the values of θ_0, θ_1 in order to reduce $J(\theta_0, \theta_1)$, until we hopefully end up at a minimum.

Gradient Descent (Cont.)

Why do we need a Gradient Descent?

In short **to minimize the cost function**, But How? **Let's see**

- The *cost function* only works when it knows the parameters' values
  - Note: in the previous example we manually choose the parameters' value each time
- During the algorithmic calculation, once the parameters' values are randomly initialized it's the *gradient descent* who must decide what params value to choose.
- In the next iteration, in order to minimize the error, it's the gradient descent who decide by how much to increase or decrease the params values.



Gradient Descent (Cont.)

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

- Where
  - := is the assignment operator
  - α is the **learning rate** which basically defines how big the steps are during the descent
  - $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ is the **partial derivative** term
  - j = 0, 1 represents the **feature index number**

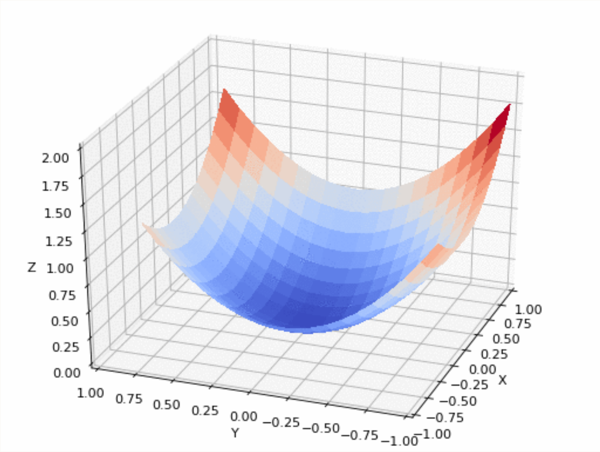
Also the parameters should be **updated simulatenously**, i.e. ,

$$temp_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$temp_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$\theta_0 := temp_0$$

$$\theta_1 := temp_1$$



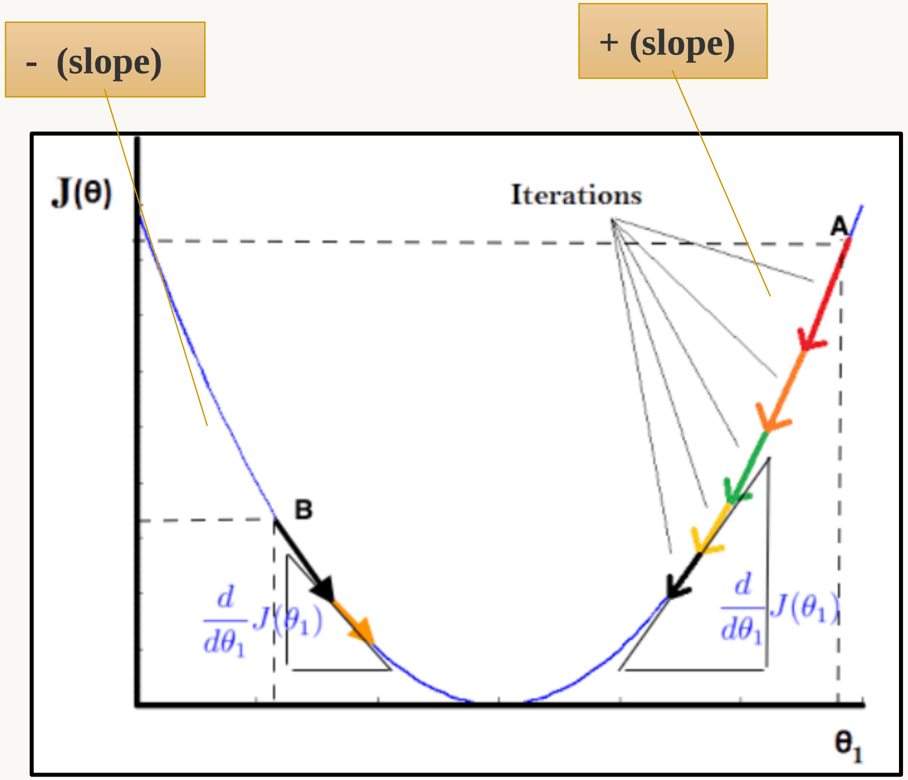
Understanding Gradient Descent

Consider a simpler cost function $J(\theta_1)$ and the objective: $\min_{\theta_1} J(\theta_1)$

Repeat{

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

}



- If the slope is **positive** and since α is a positive real number, then overall term $-\alpha \frac{d}{d\theta_1} J(\theta_1)$ would be negative. This means the value of θ_1 will be decreased in magnitude as shown by the arrows.

- If the initialization was at point B, then the **slope** of the line **would be negative** and then ***update term would be positive*** leading to increase in the value of θ_1 as shown in the plot.

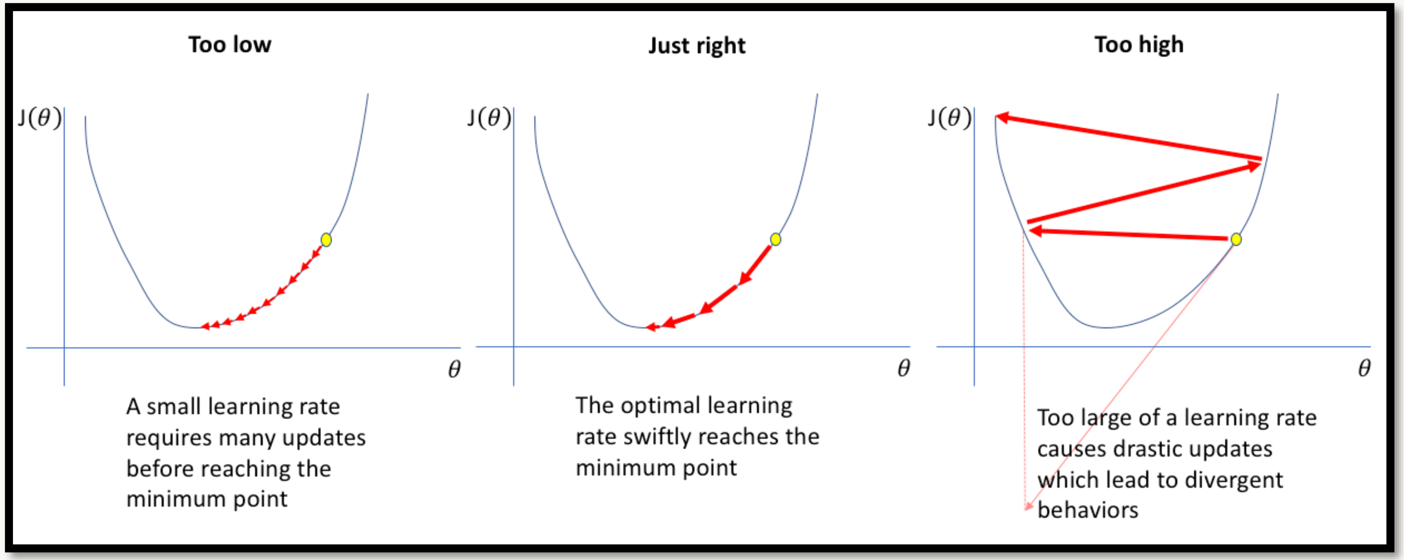
So, no matter where the θ_1 is initialized, the algorithm ensures that parameter is updated in the **right direction** towards the minima, **given proper chosen value for α** .

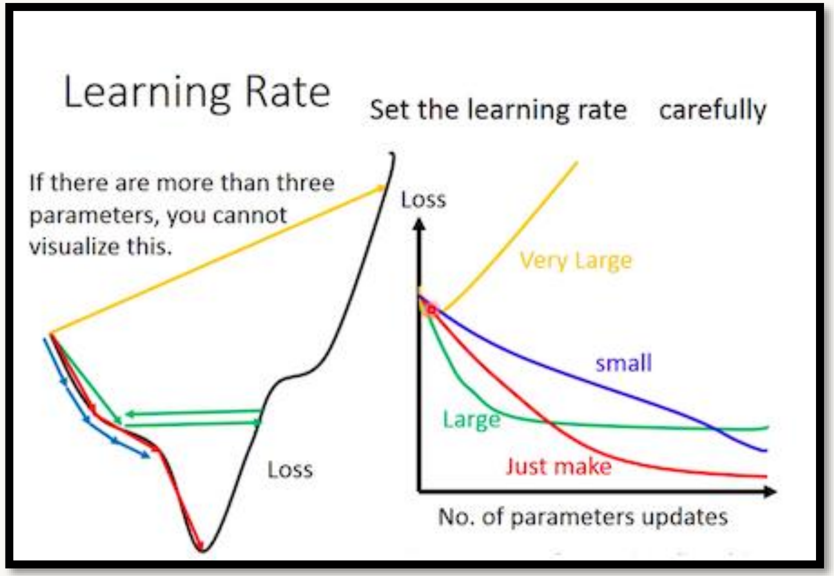
Understanding Gradient Descent(Cont.)

How can we choose the learning Rate: α?

Understanding Gradient Descent(Cont.)

Choosing learning rate:





- The **yellow** plot shows the **divergence** of the algorithm when the learning rate is really high wherein the learning steps overshoot.
- The **green** plot shows the case where learning rate is not as large as the previous case but is high enough that the steps keep **oscillating** at a point which is not the minima.
- The **red** plot would be the **optimum curve** for the cost drop as it drops steeply initially and then saturates very close to the optimum value.
- The **blue** plot is the least value of αα and **converges very slowly** as the steps taken by the algorithm during update steps are very small.

Understanding Gradient Descent(Cont.)

Practical tips:

- Gradient Descent Stopping condition

Automatic Convergence Test: Gradient descent can be considered to be converged if the drop in cost function is not more than a preset threshold say $\mathbf{10^{-3}}$.

- How to choose the learning rate?

In order to choose optimum value of (α), run the algorithm with different values like: ***1, 0.3, 0.1, 0.03, 0.01, etc.,*** and plot the learning curve to understand whether the value should be increased or decreased.

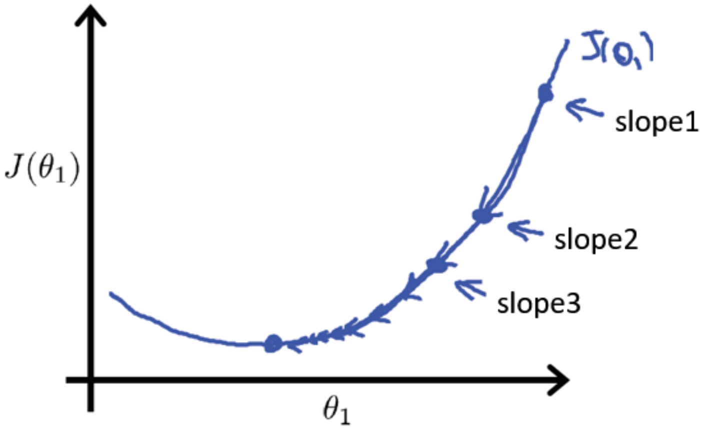
Understanding Gradient Descent(Cont.)

- **Is it required to decrease the value of α by time?**

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

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

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



Slope1> slope2> slope3> …..

By time the slope is getting smaller, consequently the value of the step is automatically is getting smaller by time, so no need to change the value of alpha.

Applying Gradient Decent To The Linear Model

Applying Gradient Descent

Model against dataset

the housing price problem

	area	price
1		
2		
...		
m		

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

	area	age	price
1			
2			
...			
m			

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

n

	area	....	#Floors	price
1				
2				
...				
m				

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

Applying Gradient Descent (Cont.)

Linear Regression Model

1

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

2

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

Gradient descent algorithm

3

repeat until convergence {

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

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

}

In order to apply gradient descent, the derivative term $\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ needs to be calculated.

Applying Gradient Descent (Cont.)

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

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

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

$$= \begin{cases} \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)}) & for\ j=0 \\ \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)})-y^{(i)})x^{(i)} & for\ j=1 \end{cases}$$

$$\text{repeat until convergence} \begin{cases} \theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}-y^{(i)})) \\ \theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h(x^{(i)}-y^{(i)}))\cdot x^{(i)} \end{cases}$$

Applying Gradient Descent (Cont.)

Gradient descent algorithm

repeat until convergence {

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

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

}

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

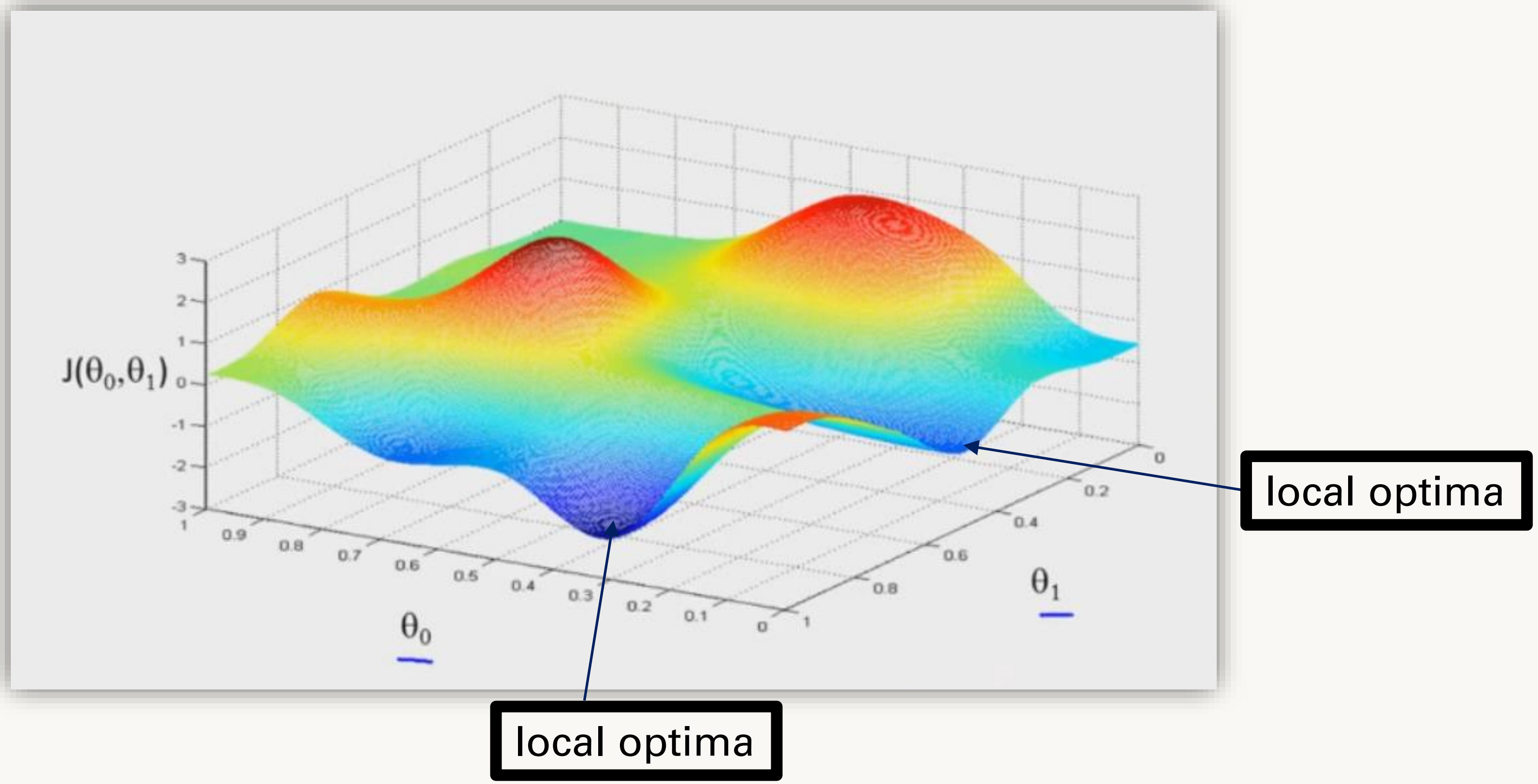
$\frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$

update θ_0 and θ_1 simultaneously

Applying Gradient Descent (Cont.)

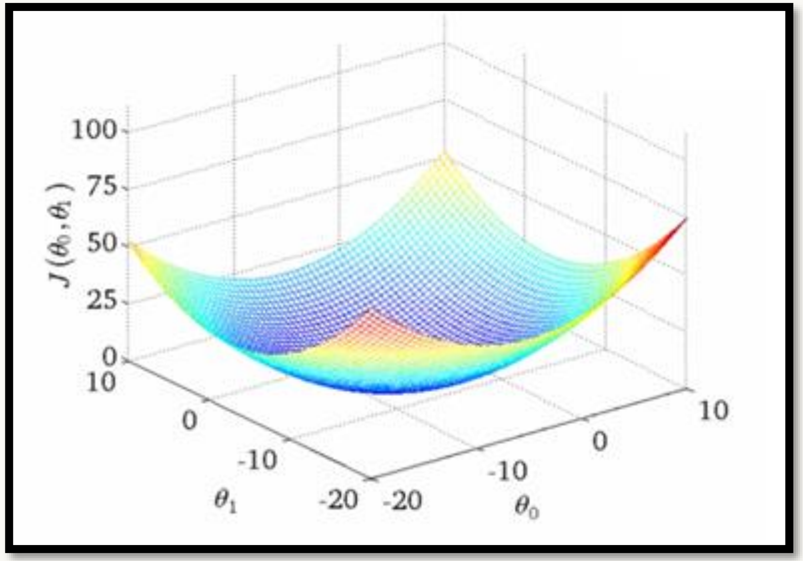
Risk of meeting different local optimum

The problem of existing more than one local optima, may make the gradient decent converge to the non global optima.



Applying Gradient Descent (Cont.)

- Cost/error function is always convex:

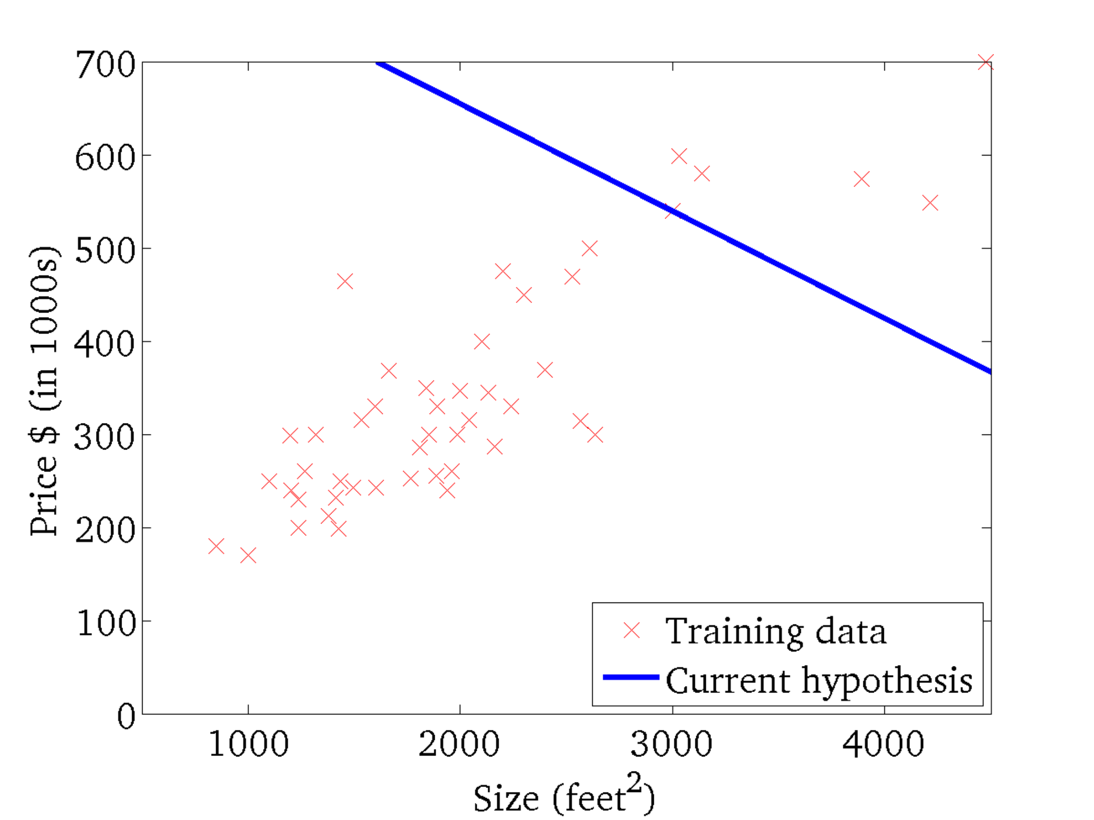


- The linear regression cost function is always a **convex function** - always has a single minimum
  - Bowl shaped
  - One global optima
    - So gradient descent will always converge to global optima

Applying Gradient Descent (Cont.)

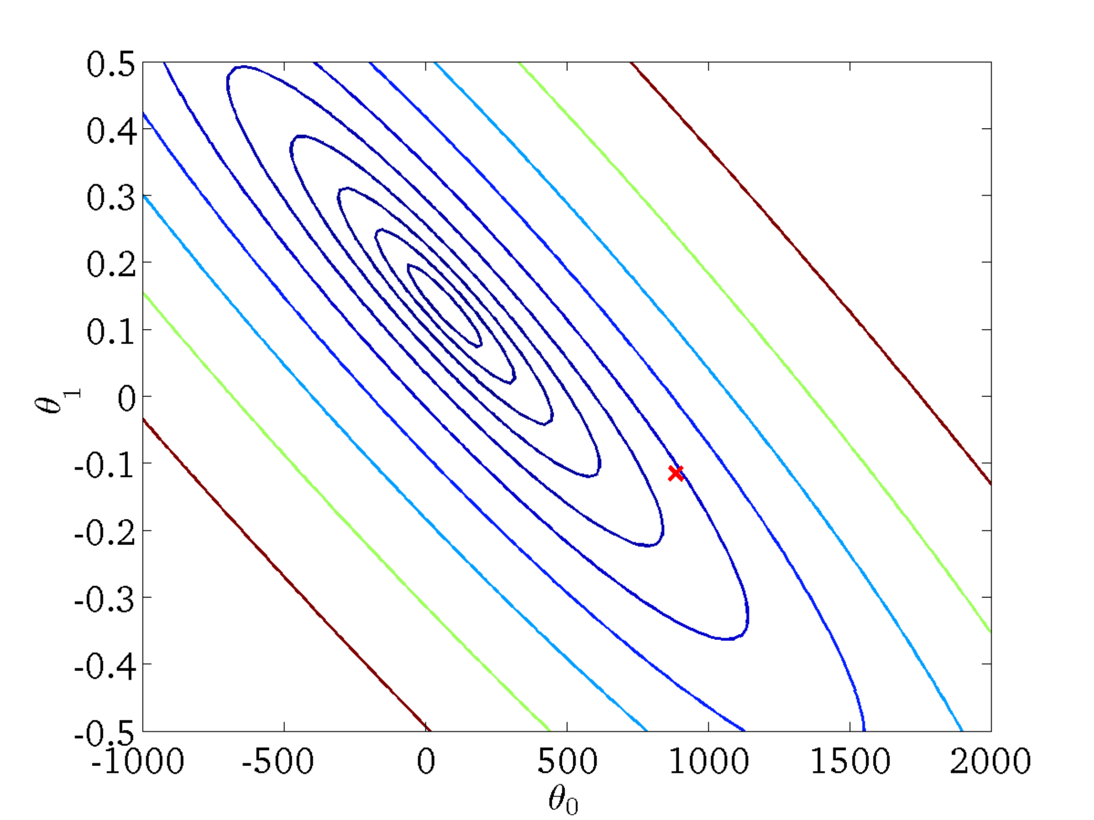
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

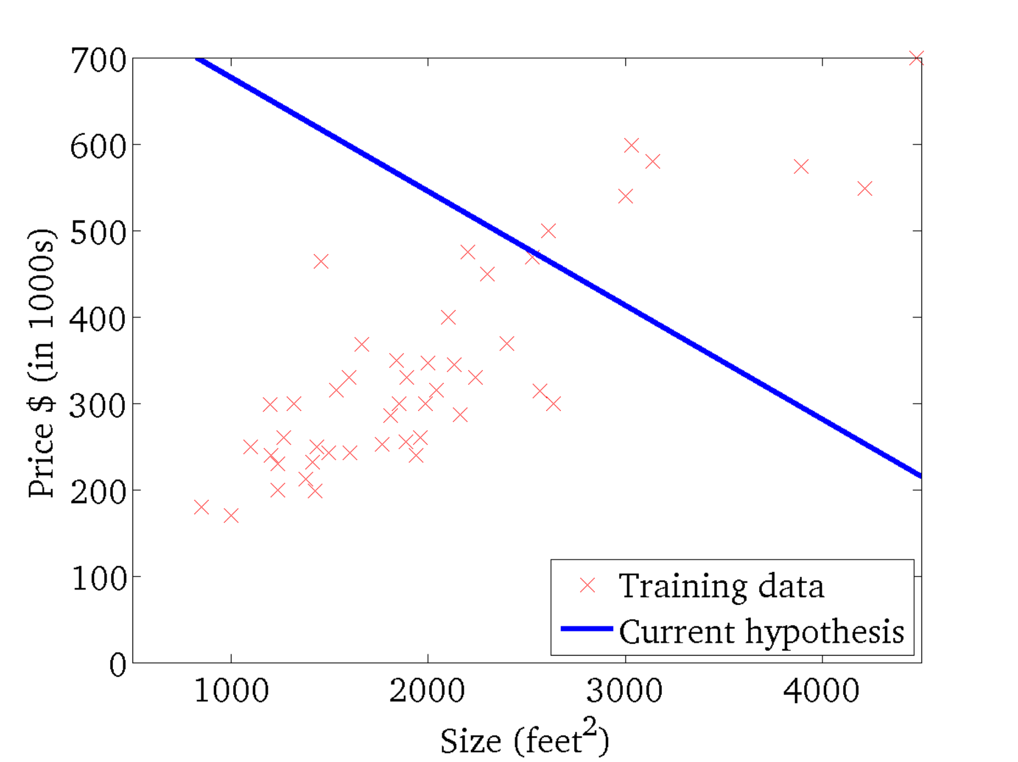
(function of the parameters θ_0, θ_1)



Applying Gradient Descent (Cont.)

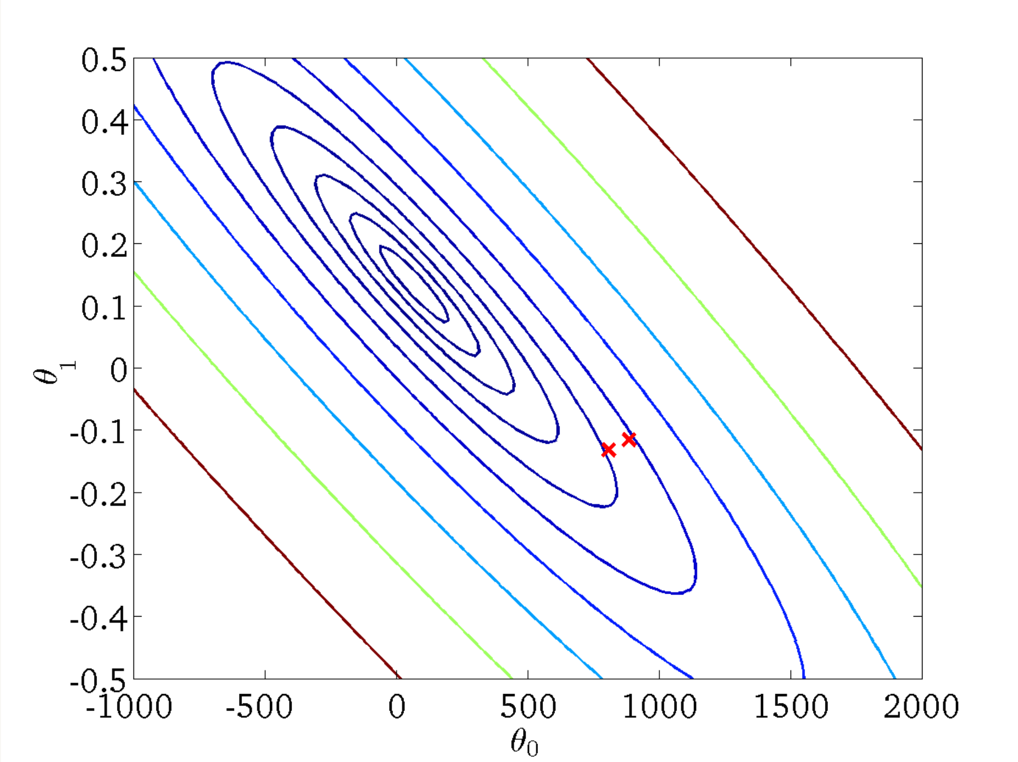
$$h_\theta(x)$$

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



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

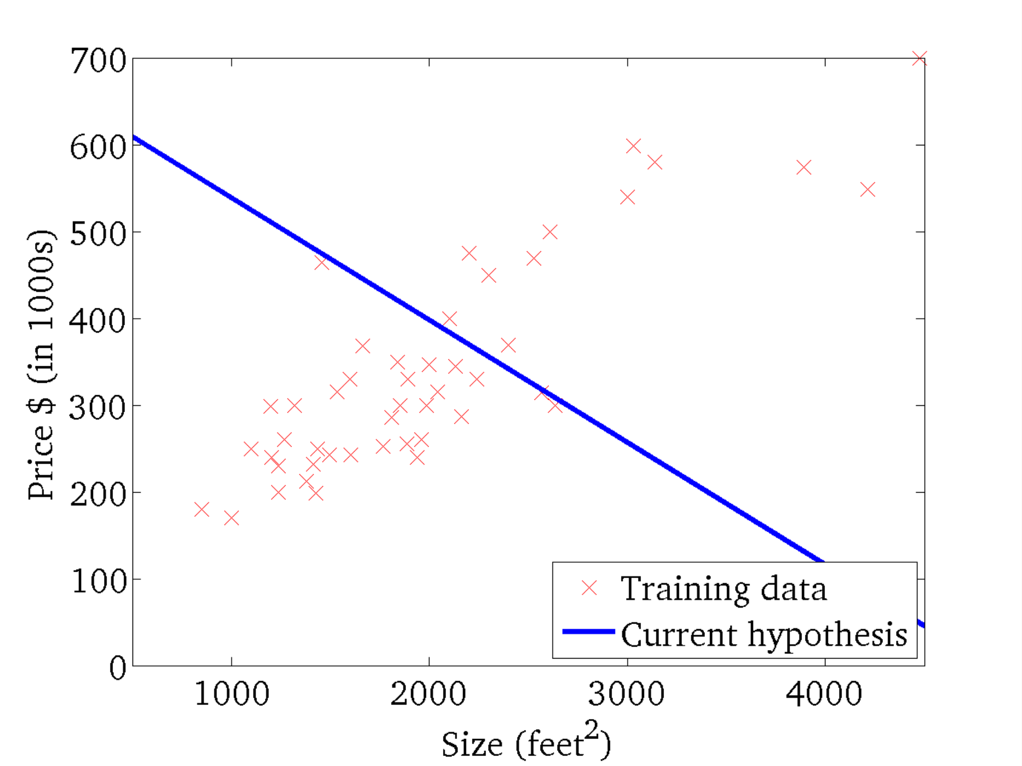
(function of the parameters θ_0, θ_1)



Applying Gradient Descent (Cont.)

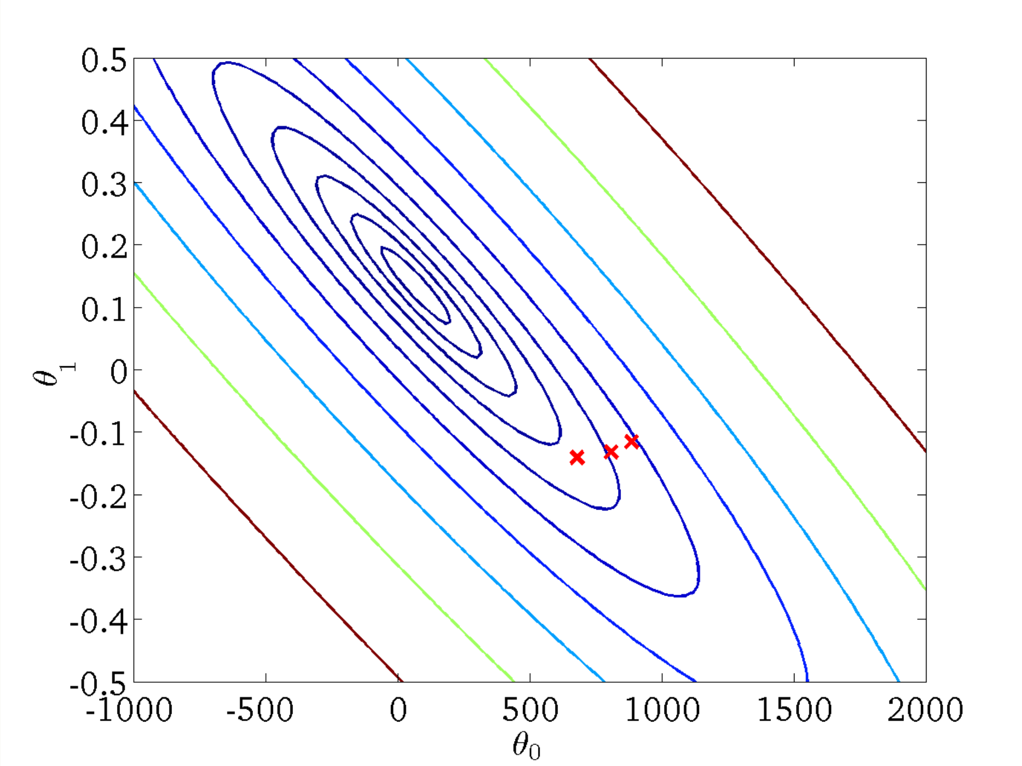
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

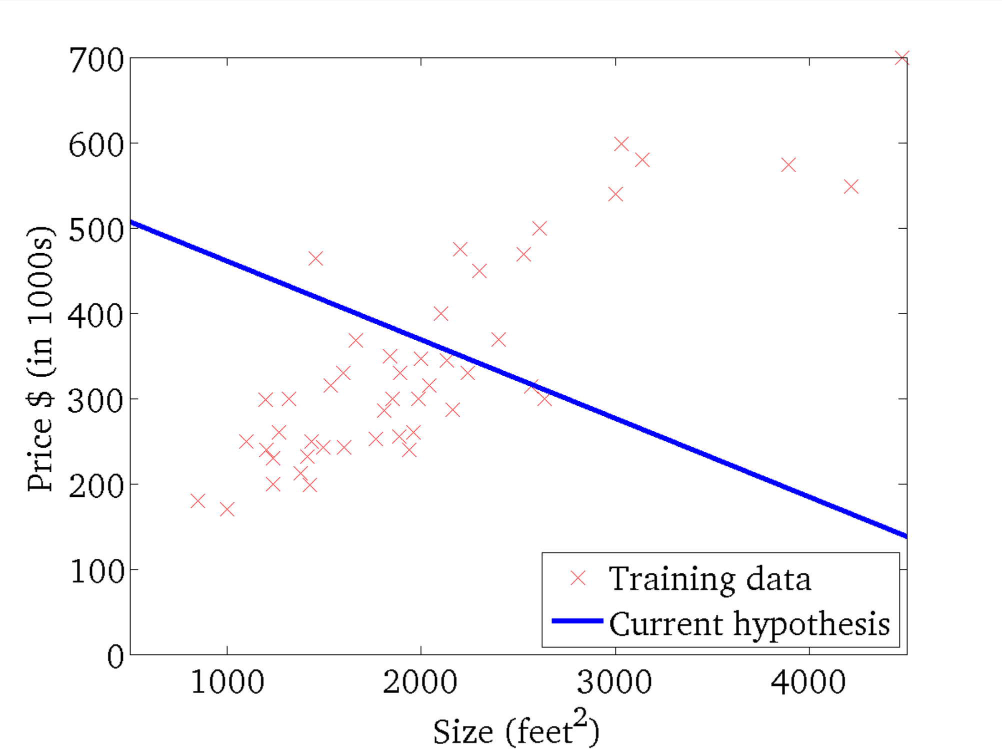
(function of the parameters θ_0, θ_1)



Applying Gradient Descent (Cont.)

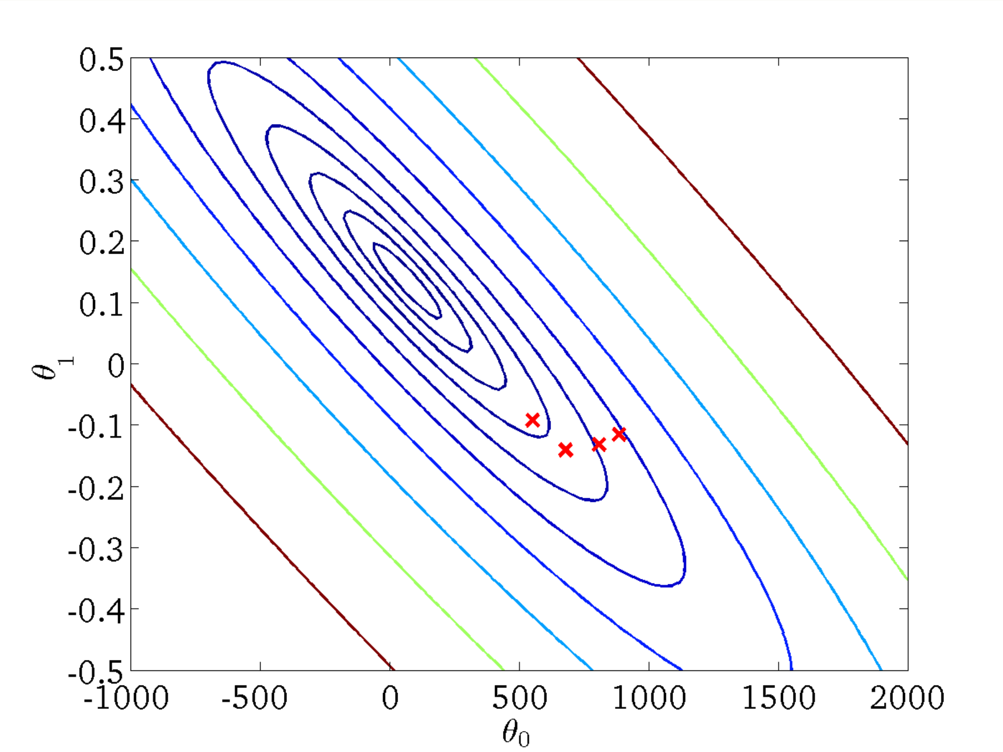
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

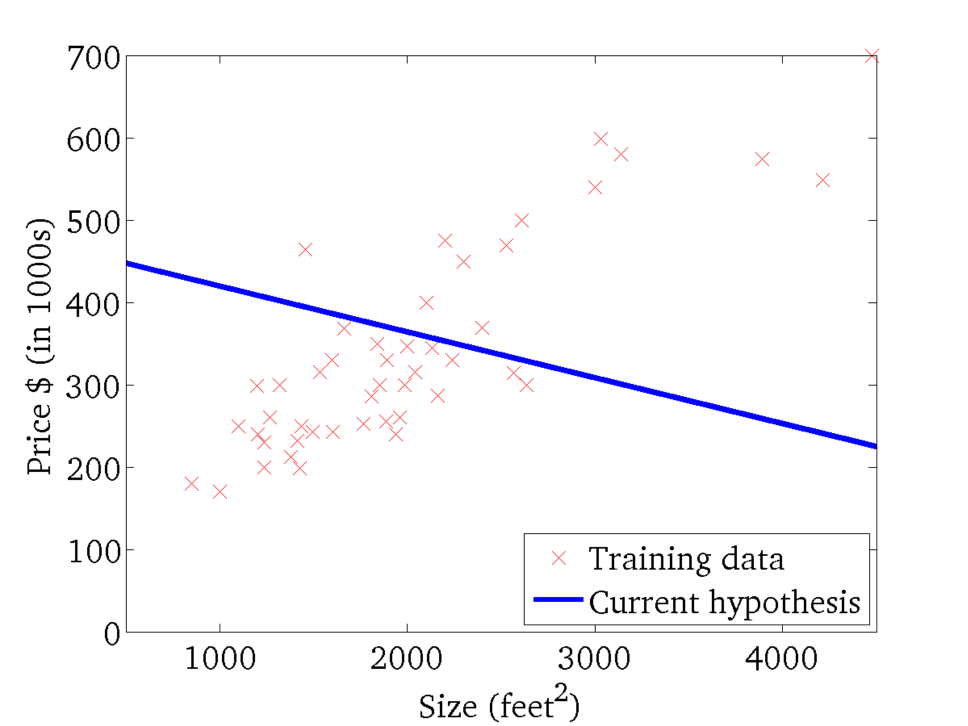
(function of the parameters θ_0, θ_1)



Applying Gradient Descent (Cont.)

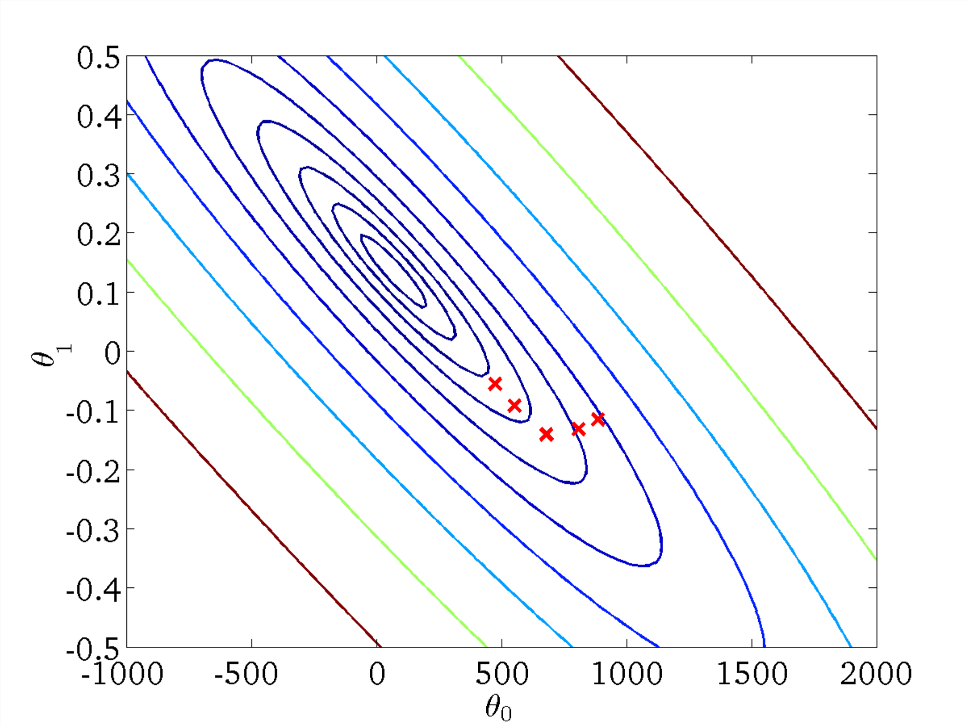
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

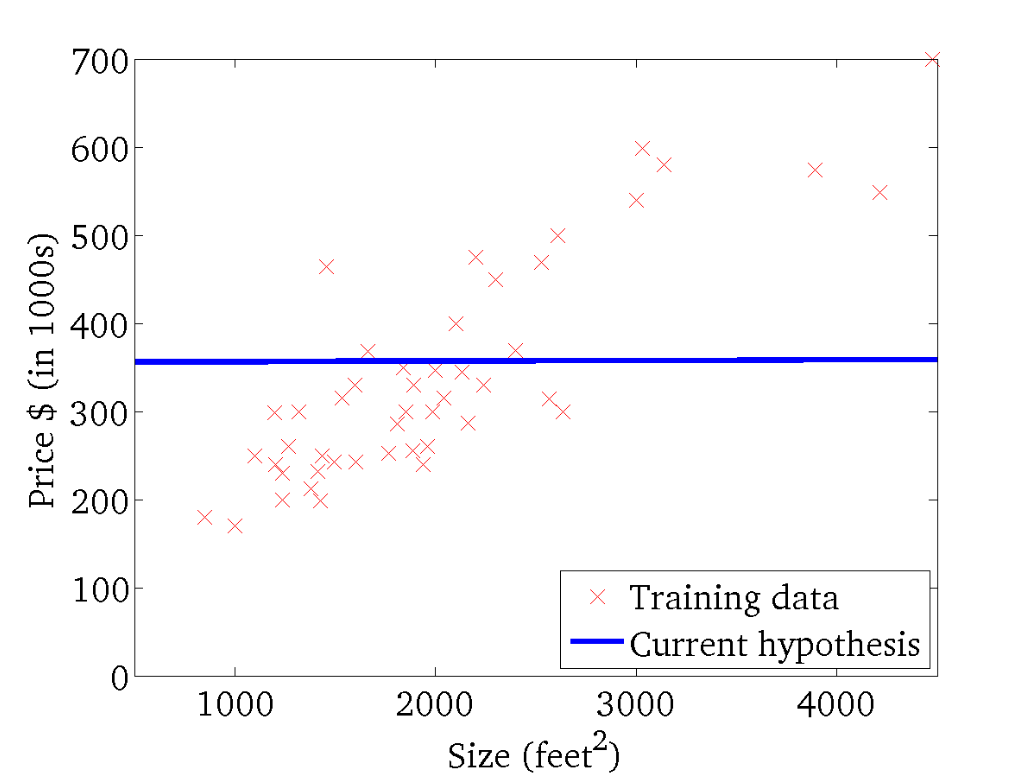
(function of the parameters θ_0, θ_1)



Applying Gradient Descent (Cont.)

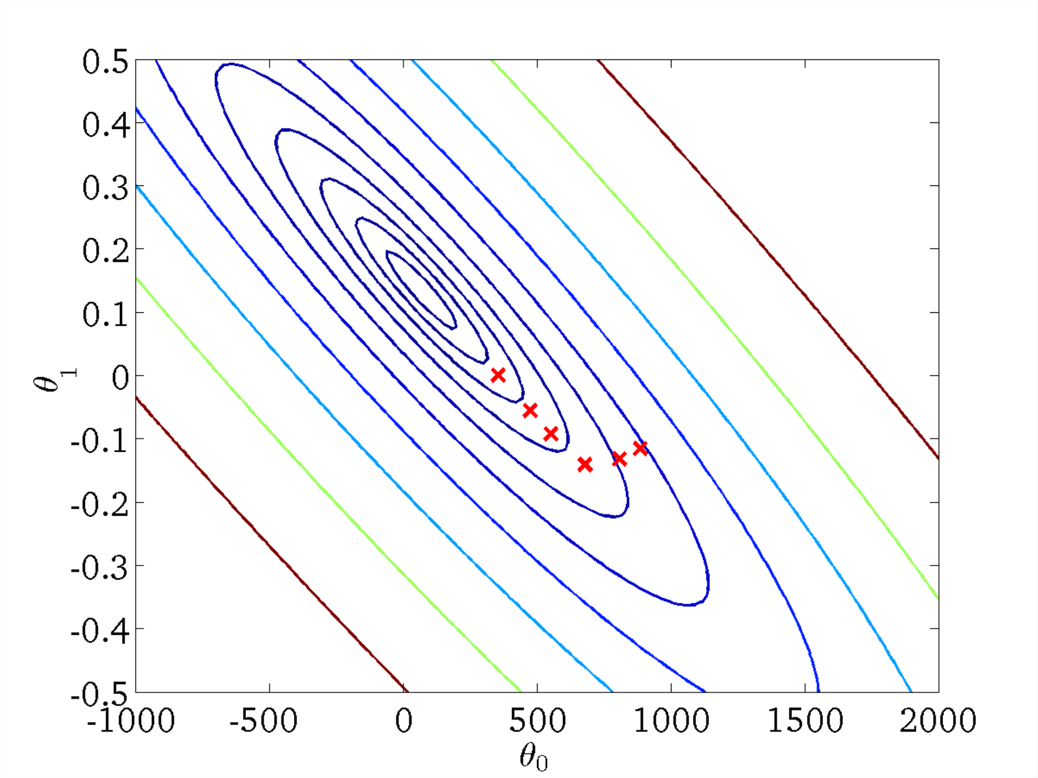
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

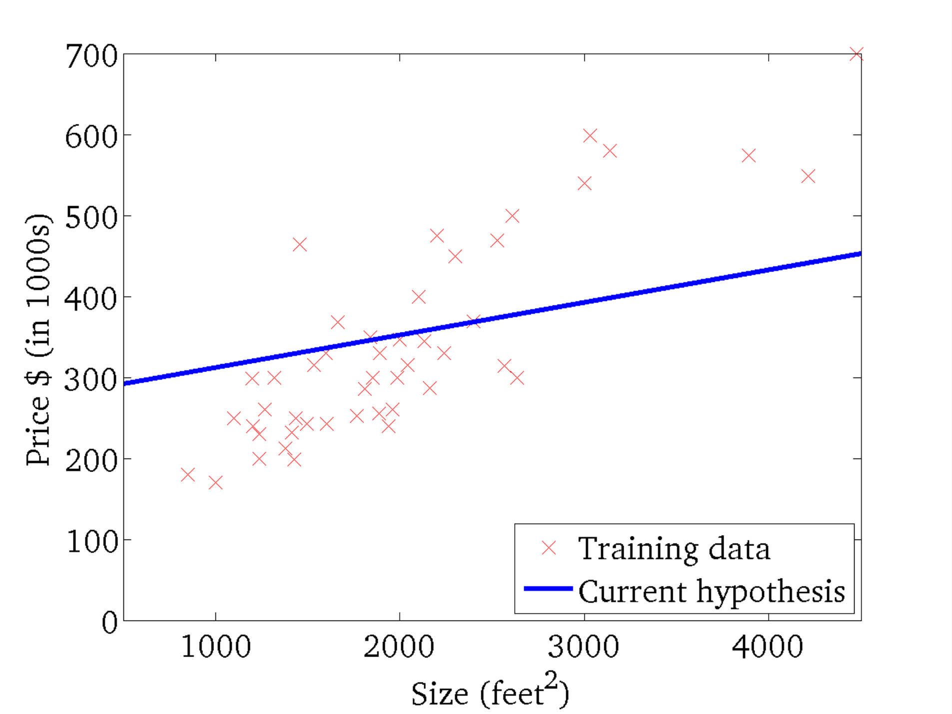
(function of the parameters θ_0, θ_1)



Applying Gradient Descent (Cont.)

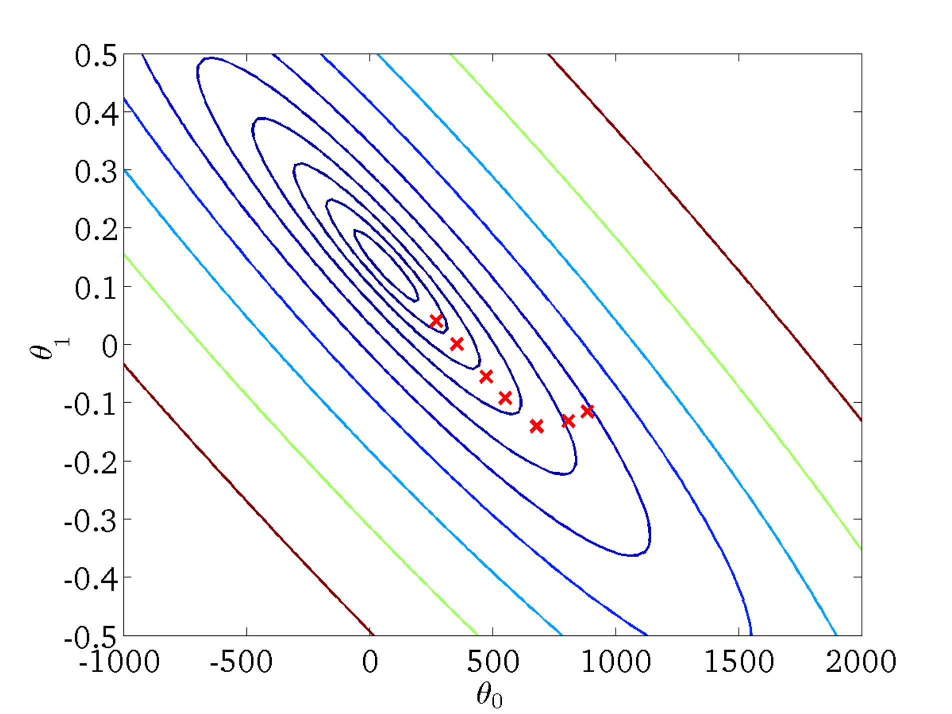
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

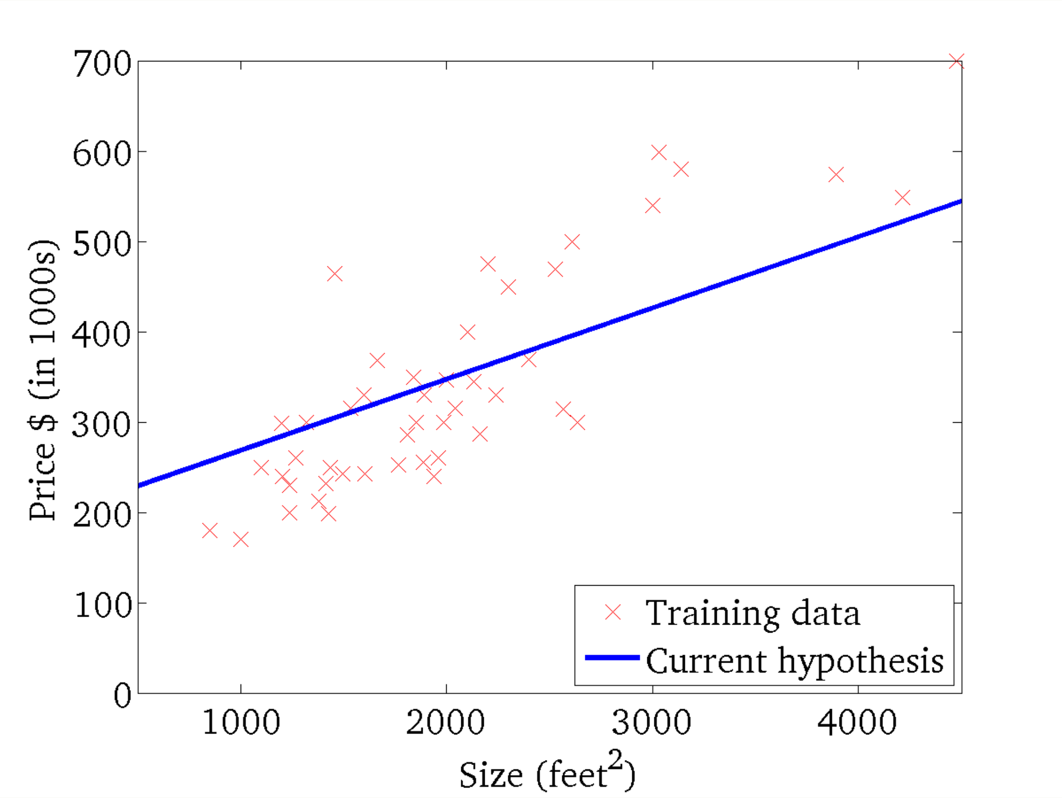
(function of the parameters θ_0, θ_1)



Applying Gradient Descent (Cont.)

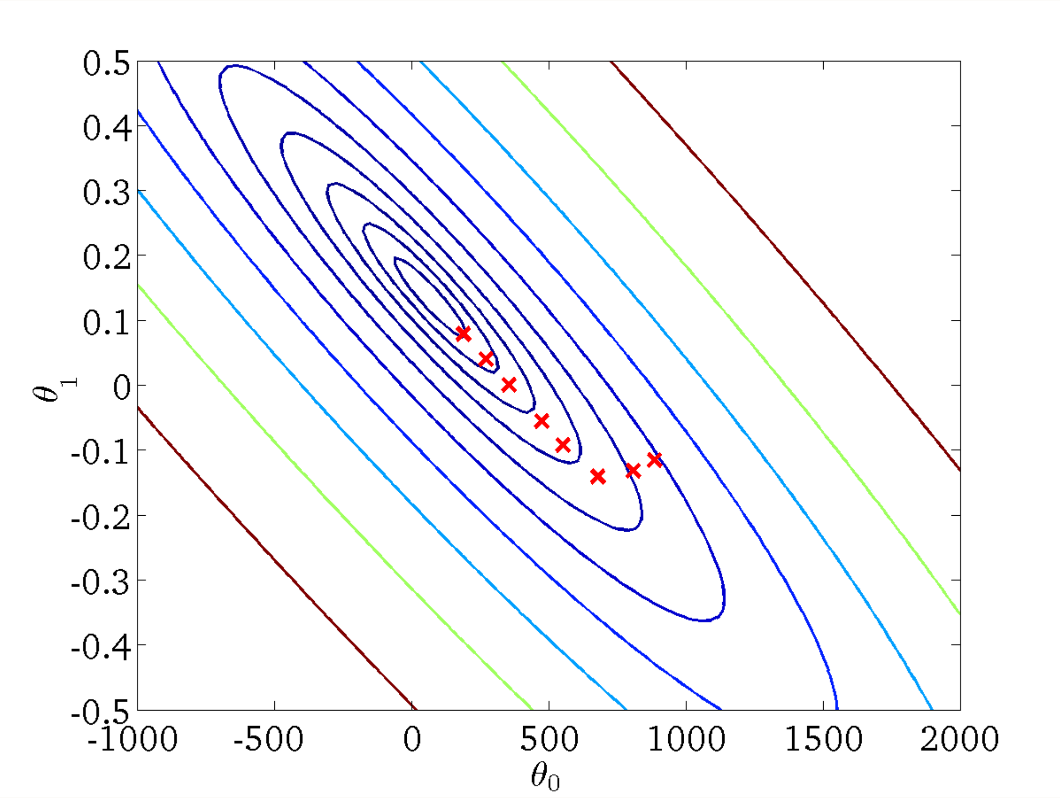
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



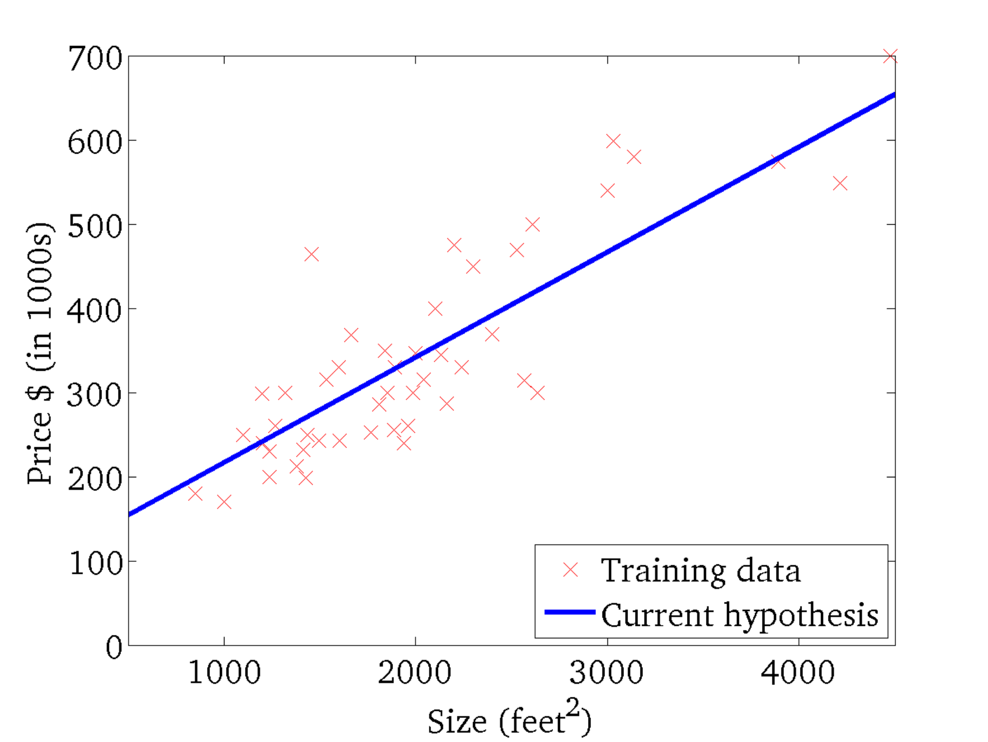
Applying Gradient Descent (Cont.)

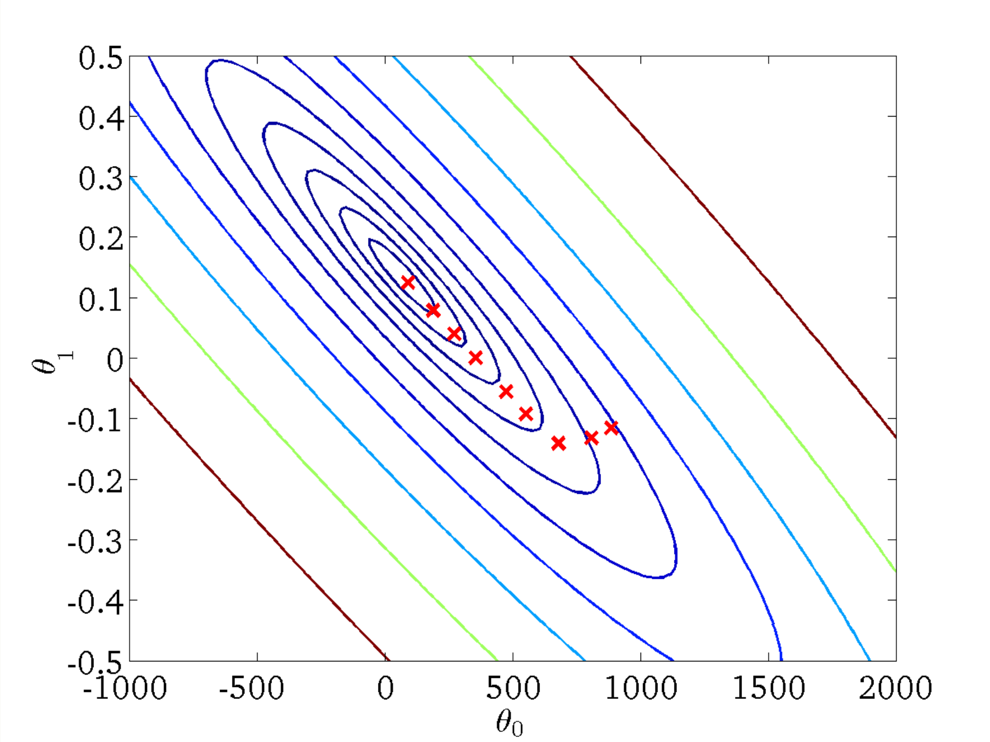
$h_\theta(x)$

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

$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

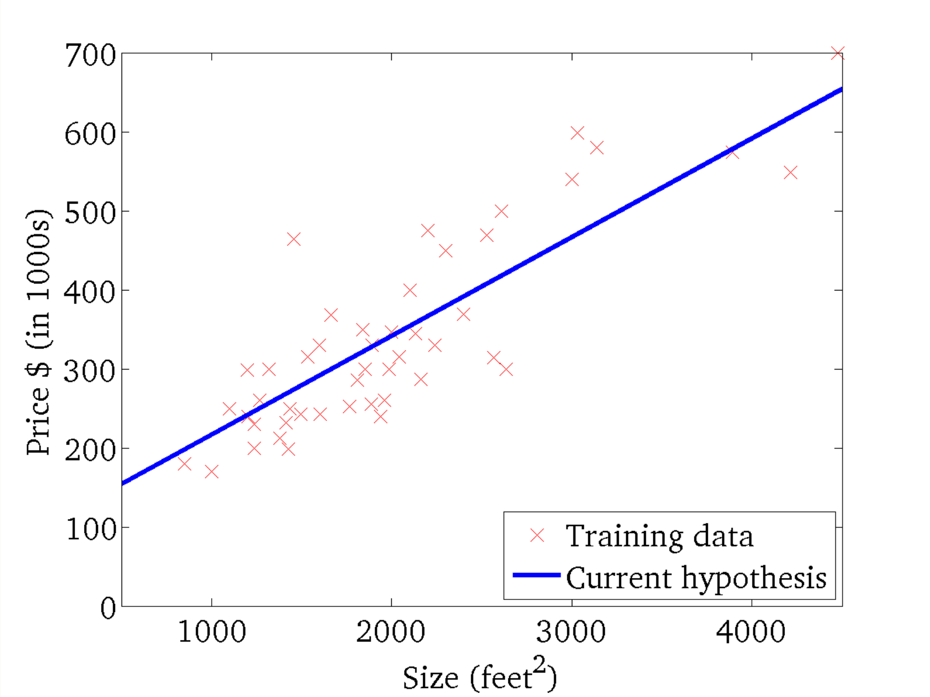




Applying Gradient Descent (Cont.)

$$h_\theta(x)$$

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



Now you can do prediction, given a house size!

Batch Gradient Descent

- The gradient descent technique that uses all the training step is called **Batch Gradient Descent**. This is basically the calculation of the derivative term over all the training examples as it can be seen it the equation above.

Gradient descent algorithm

repeat until convergence {

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

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

}

Note:
The equation of linear regression can also be solved using **Normal Equations** method, but it poses a disadvantage that it does not scale very well on larger data while gradient descent does.

More about Gradient Descent

- There are plenty of optimizers

  - Adagrad
  - Adadelta
  - Adam
  - Conjugate Gradients
  - BFGS
  - Momentum
  - Nesterov Momentum
  - Newton's Method
  - RMSProp
  - SGD

Summary

- With the cost function the learning algorithm determine whither the current parameters' values are the best or not (by calculating the error).

- Using the gradient descent algorithm is mainly to automate the process of updating the values of the parameters towards the global minimum.

- There are **two basic steps** involved in gradient descent algorithm:
  - ❑ Start with some random value of θ_0, θ_1.
  - ❑ Keep updating the value of θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until minimum is reached.

Any Question?

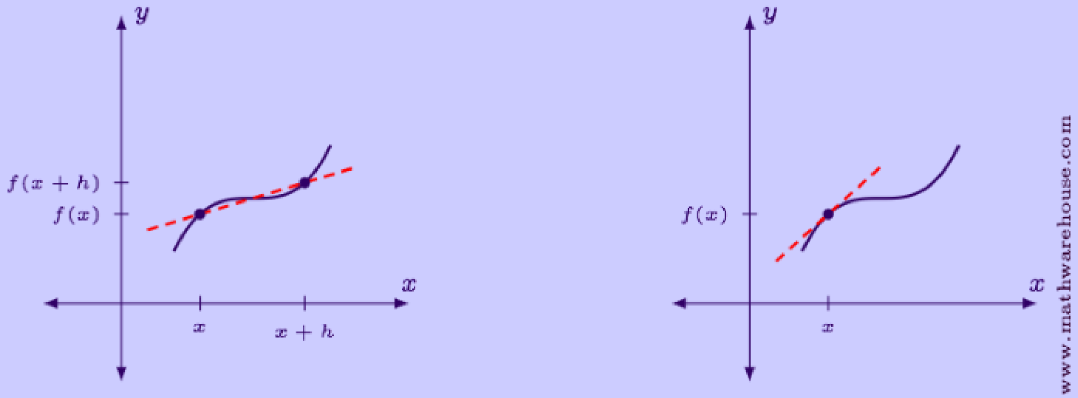
Appendix

The Derivative as the Slope of a Tangent Line

Recall that the definition of the derivative is

$$\lim_{h \to 0} \frac{f(x+h) - f(x)}{(x+h) - x}.$$

Without the limit, this fraction computes the slope of the line connecting two points on the function (see the left-hand graph below).



The only thing the limit does is to move the two points closer to each other until they are right on top of each other. But the fundamental calculation is still a slope. So the end result is the slope of the line that is tangent to the curve at the point $(x, f(x))$.