CS395: Selected CS1 (Introduction to Machine Learning)

Associate Prof. Wessam El-Behaidy

Fall 2021

References:

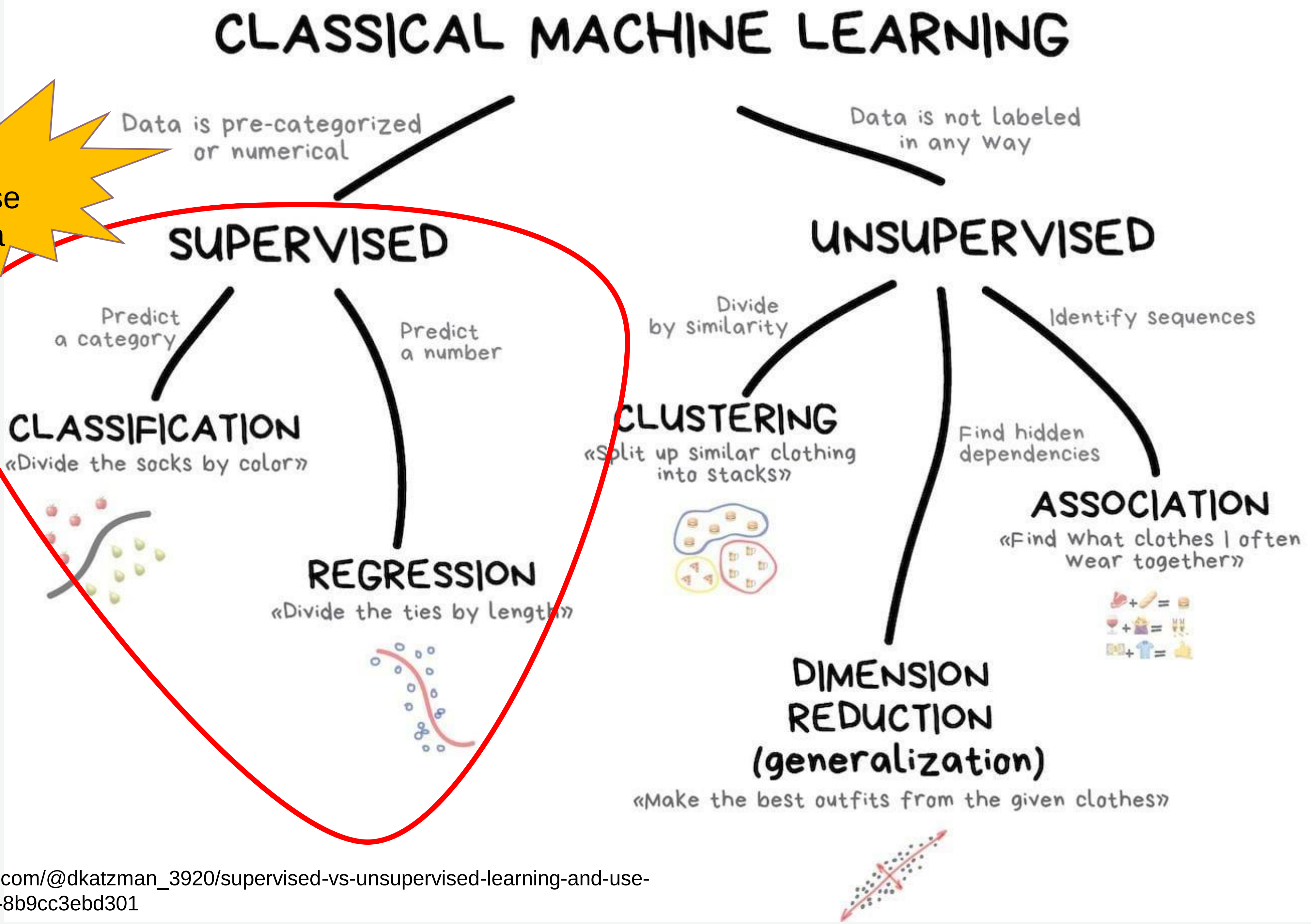
https://www.coursera.org/learn/machine-learning (Andrew Ng)

Machine learning A to Z: **Kirill Eremenko ©superdatascience**

Grading Distribution (Update)

- Project: (15%)
  - Code “github”, pitching video, presentation.
- Midterm exam (15%)
- Quiz (10%)
- Final exam (60%)

Our course area



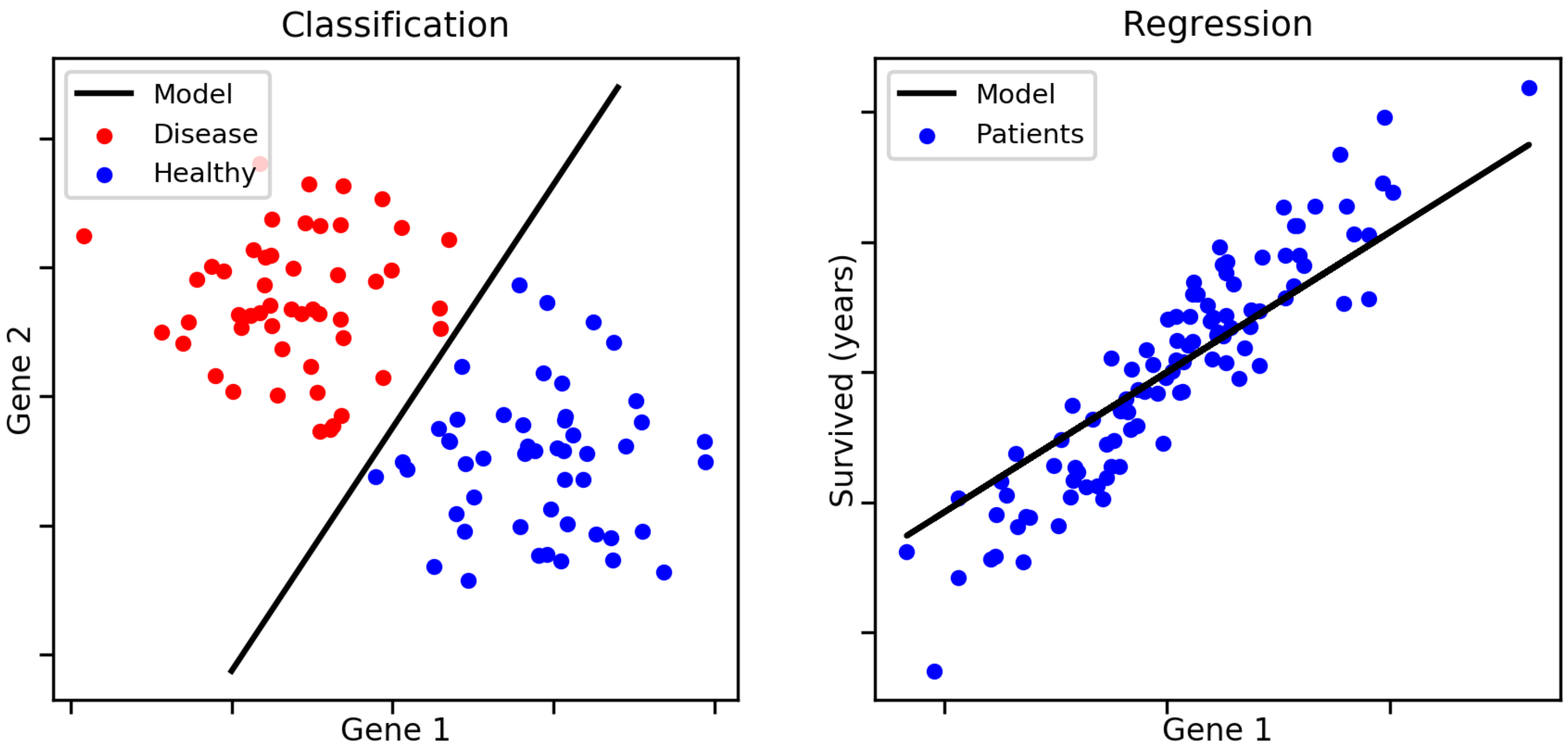
https://medium.com/@dkatzman_3920/supervised-vs-unsupervised-learning-and-use-cases-for-each-8b9cc3ebd301

Supervised Learning

- **In supervised learning**, we are given an (**labeled**/**annotated**) dataset that we already know what our correct output should look like (**true label**), having the idea that there is a relationship between the input and the output.
- Supervised learning problems are categorized into "regression" and "classification" problems.

Supervised Learning

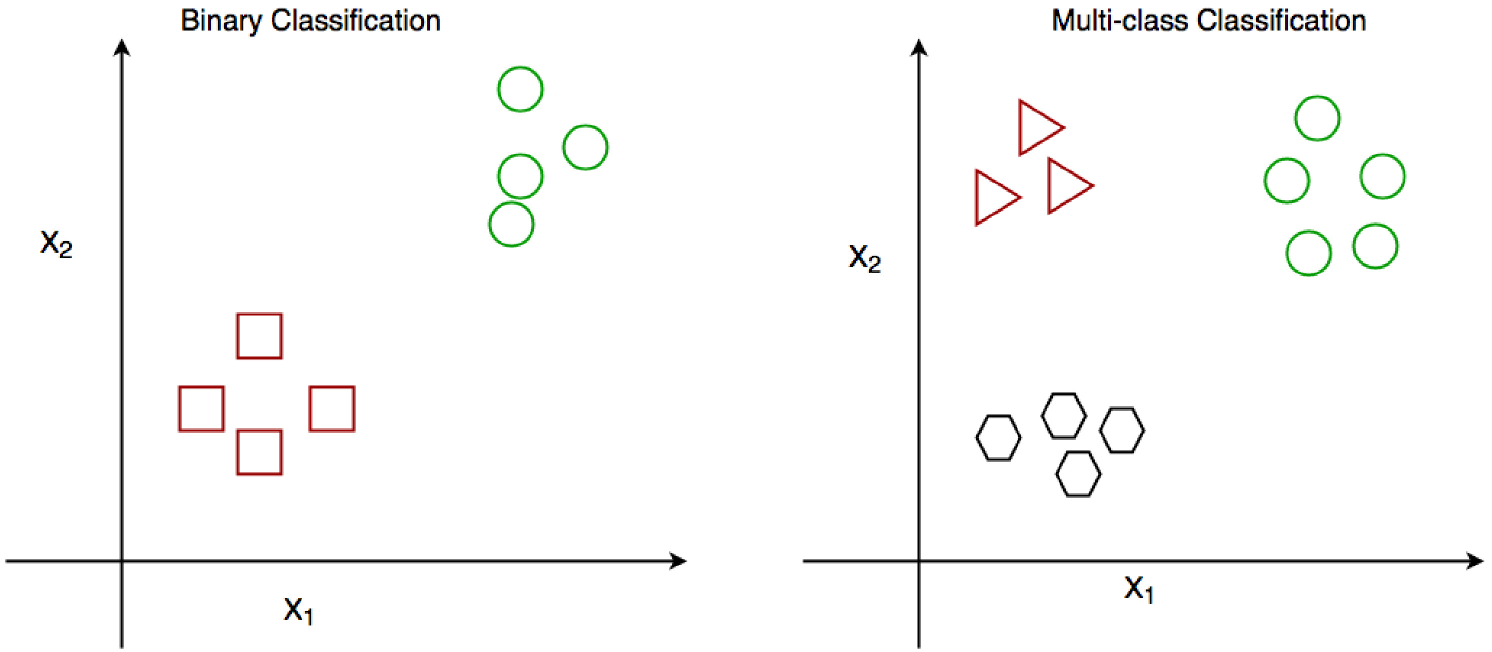
- In a **regression problem**, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function.
- In a **classification problem**, we are instead trying to predict results in a discrete output.



https://aldro61.github.io/microbiome-summer-school-2017/sections/basics/

Supervised Learning

- **Classification problem** can be binary or multi-class. **Binary classification** has only 2 outputs, whereas **multi-class** has more than 2 outputs (3 or more).



https://www.geeksforgeeks.org/ml-classification-vs-regression/

Regression or Classification ?

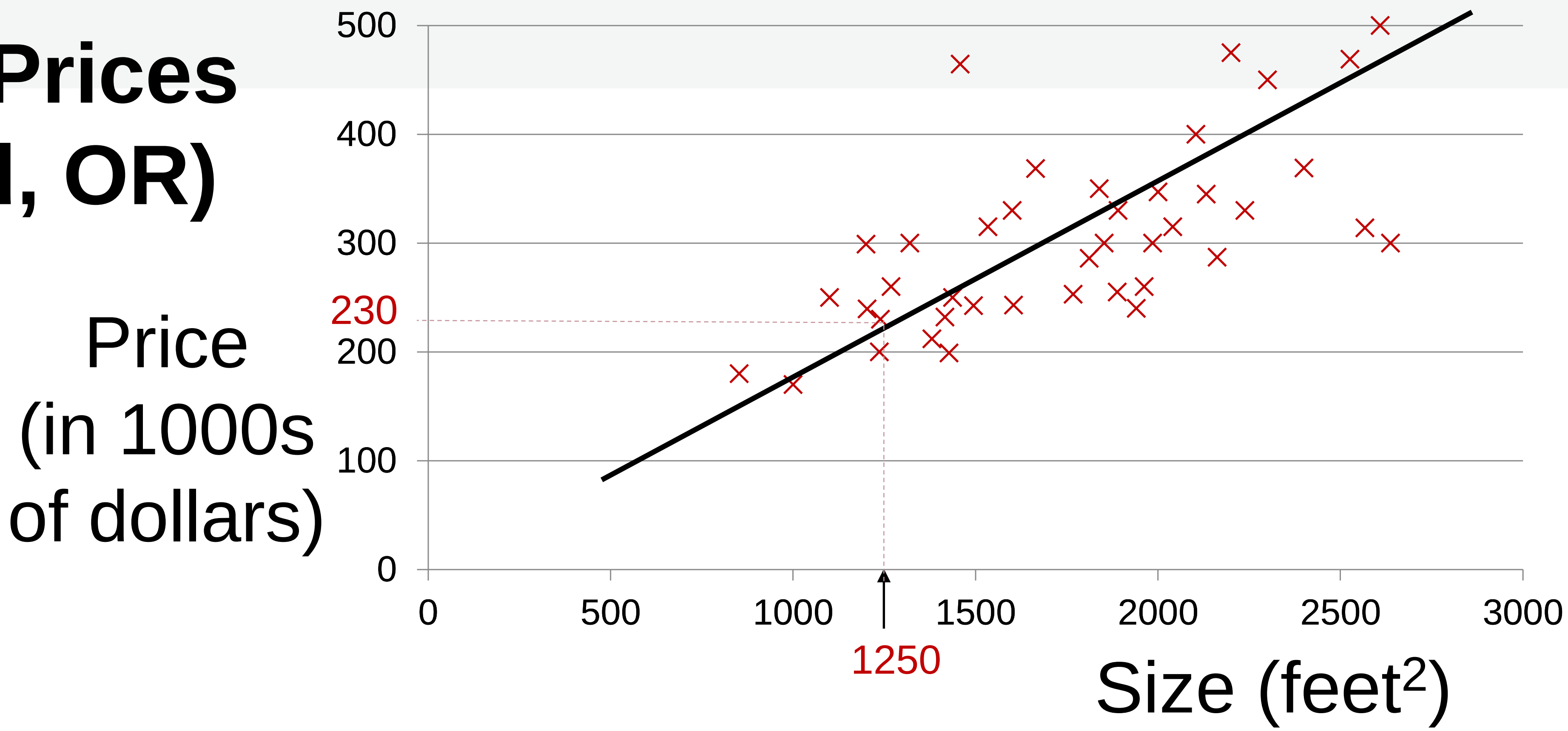
- Given data about the size of houses on the real estate market, try to predict their price. R
- Given a picture of Male/Female, We have to predict his/her age on the basis of given picture. R
- Given a picture of Male/Female, We have to predict Whether She is of High school, College, Graduate age. C
- Banks have to decide whether or not to give a loan to someone on the basis of his credit history. C

Housing price data example

- Suppose we have a dataset giving the living areas and prices of **47** houses from Portland, Oregon:

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

Housing Prices (Portland, OR)



<u>Supervised Learning</u>

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

<u>Regression Problem</u>

Predict real-valued output

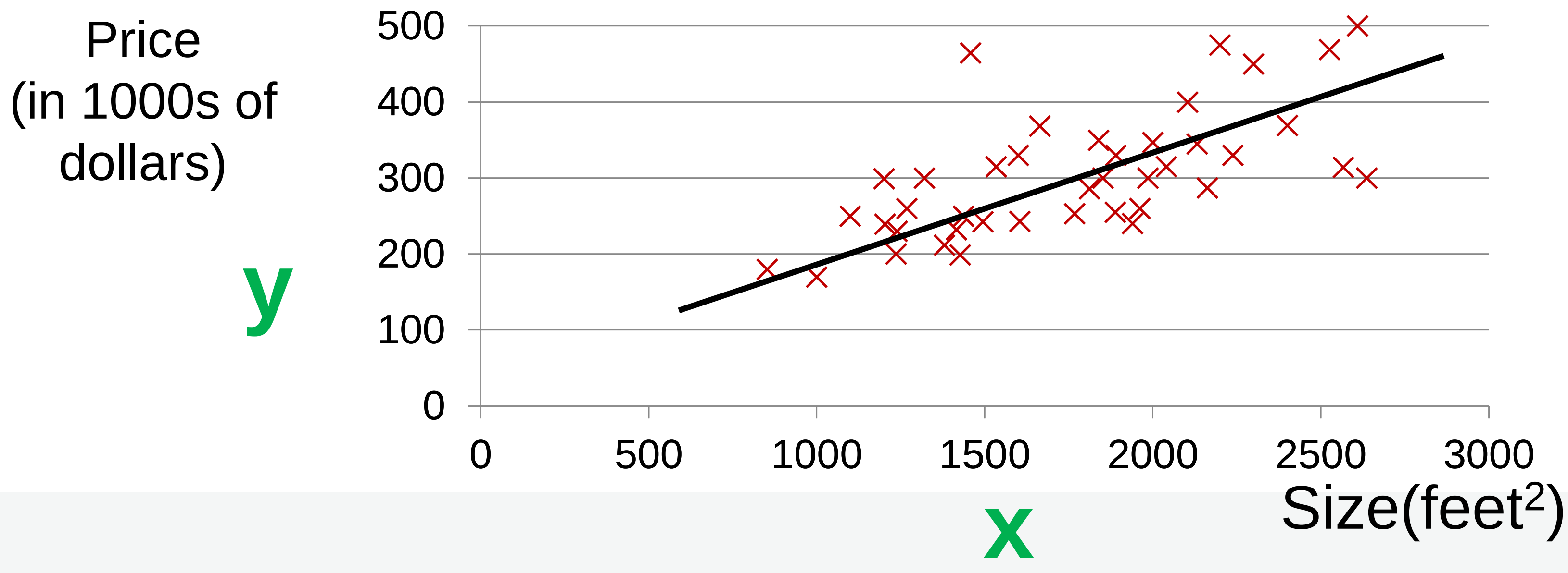
Linear Regression with One Variable

- **Model Representation**

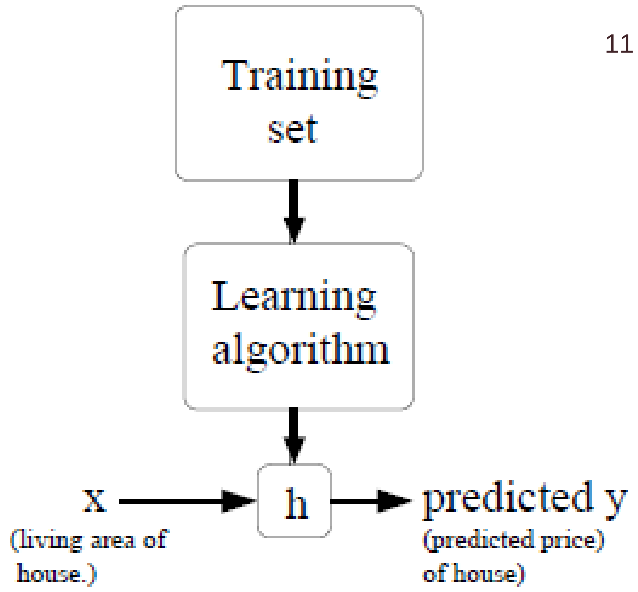
- *regression problems*, we are taking input variables and trying to fit the output onto a ***continuous*** expected result function.

- Linear regression with one variable is also known as "**univariate** linear regression.“

- We want to predict a **single output** value **y** from a **single input** value **x**.



The Hypothesis Function



(hypothesis)

- Our **hypothesis function** has the general form:

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

Parameters/ weights

  - like the equation of a straight line
  - we are trying to create a function called h_θ that is trying to map our input data (the x's) to our output data (the y's).

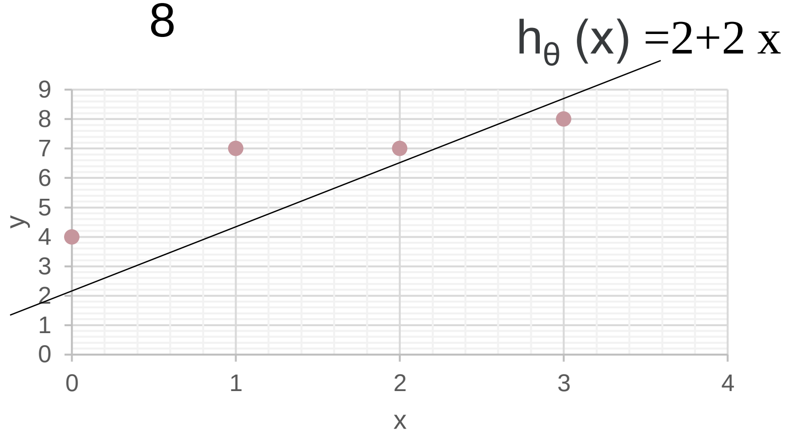
Notation:

x's = "input" variable / features
y's = "output" variable / "target" variable

Example

input **x**	output **y**
0	4
1	7
2	7
3	8

- Suppose we have the following set of training data:



- Now we can make a random guess about our h_θ

  For example: θ_0, θ_1 =2

- The hypothesis function becomes h_θ (x) =2+2 x
- So for input of x=1 to our hypothesis $h_\theta(1)$, $\hat{y} = 4$. This is off by 3. Note that we will be trying out various values θ_0, θ_1
  - to try to find values which provide the best possible "**fit**" or the most representative "**straight line**" through the data points mapped on the x-y plane.

Cost Function

- We can measure the accuracy of our hypothesis function by using a **cost function**.
- This takes an average of all the results of the hypothesis with inputs from x's compared to the actual output y's.

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

Squared error function

Notation
m = Number of training examples
$(x^{(i)}, y^{(i)})$ = i^{th} *training example*

Objective function (Our goal) : $minimize_{\theta_0,\theta_1} J(\theta_0, \theta_1)$

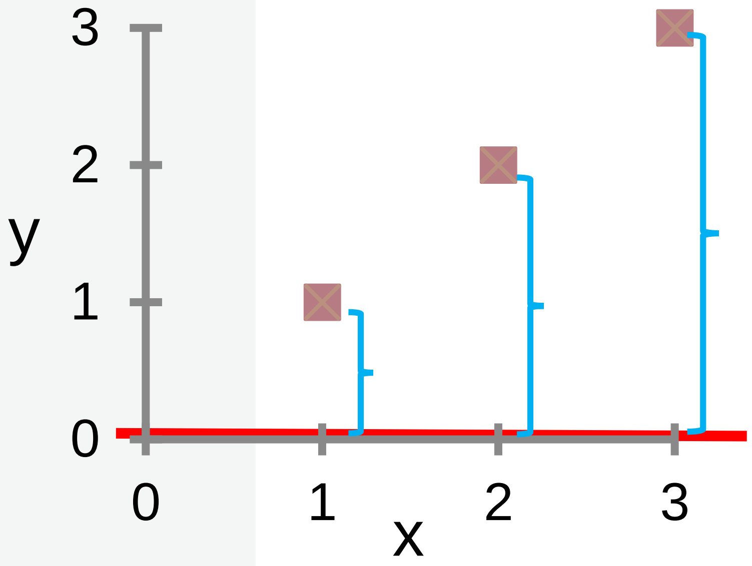
Choose θ_0, θ_1 so that $h_\theta(x)$ is close to output y for our training examples (x,y)

input x	output y
1	1
2	2
3	3

Simplified Example $h_\theta = \theta_1 x$

$h_\theta(x)$

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



$\theta_1 = 0$
$h_\theta = 0$

Cost function J(θ_1)= $\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$

Where m (# of samples)= 3

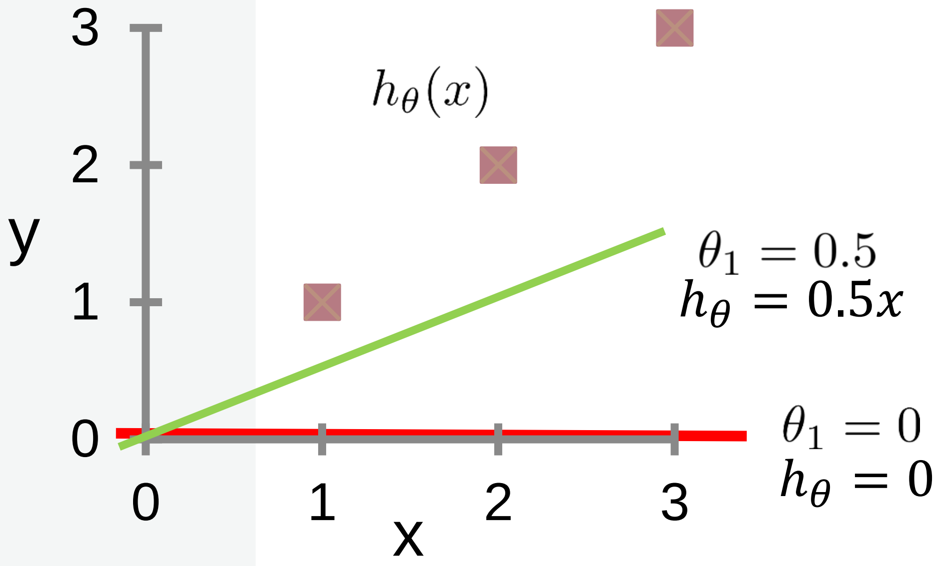
J(0)= $\frac{1}{6}$ $((0-1)^2 + (0-2)^2 + (0-3)^2)$ =$\frac{1}{6}$ (1+4+9) ≈ 2.3

input x	output y
1	1
2	2
3	3

Simplified Example $h_\theta = \theta_1 x$

$h_\theta(x)$

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



Cost function J(θ_1)= $\frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$

Where m (# of samples)= 3

J(0)= $\frac{1}{6}$ $((0-1)^2 + (0-2)^2 + (0-3)^2)$ =$\frac{1}{6}$ (1+4+9) ≈ 2.3

J(0.5)= $\frac{1}{6}$ $((0.5-1)^2 + (1-2)^2 + (1.5-3)^2)$

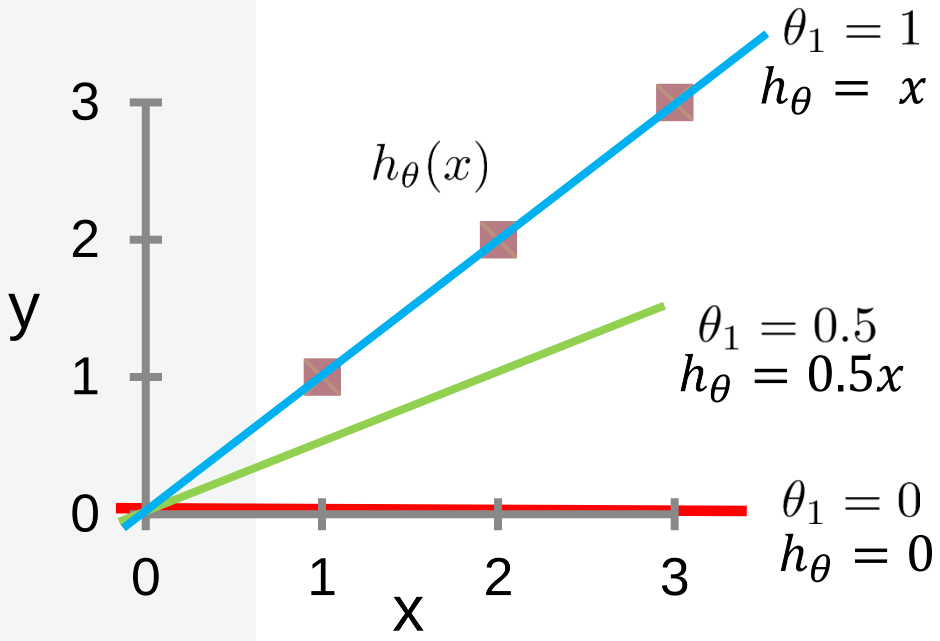
= $\frac{1}{6}$ (0.25+1+2.25) ≈ 0.58

input x	output y
1	1
2	2
3	3

Simplified hypothesis $\boldsymbol{h_\theta = \theta_1 x}$

$h_\theta(x)$

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



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

Where m (# of samples)= 3

$J(0) = \frac{1}{6}\,((0-1)^2 + (0-2)^2 + (0-3)^2) = \frac{1}{6}\,(1+4+9) \approx 2.3$

$J(0.5) = \frac{1}{6}\,((0.5-1)^2 + (1-2)^2 + (1.5-3)^2)$

$= \frac{1}{6}\,(0.25+1+2.25) \approx 0.58$

$J(1) = \frac{1}{6}\,((1-1)^2 + (2-2)^2 + (3-3)^2)$

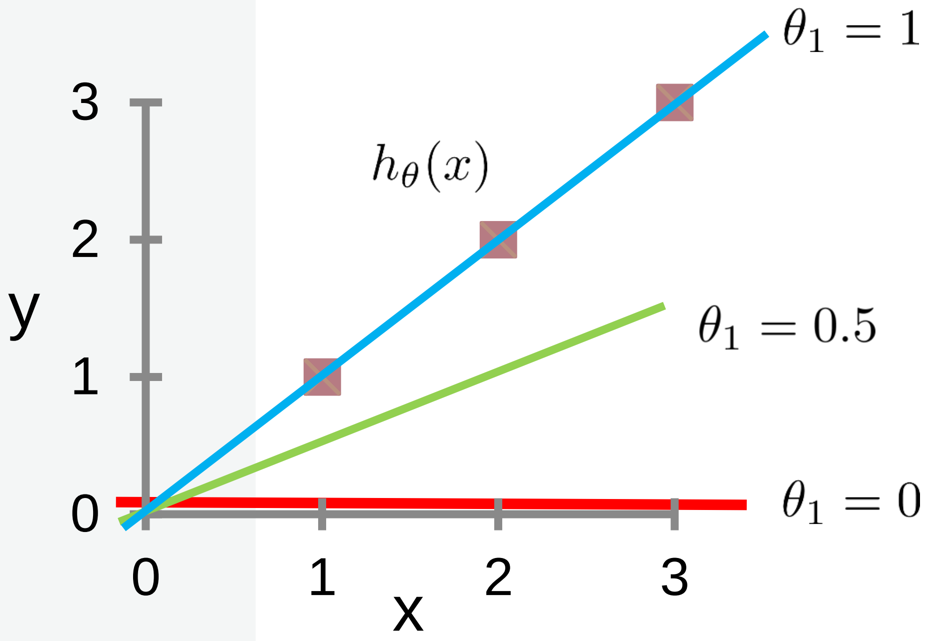
$= \frac{1}{6}\,(0+0+0) = 0$

Simplified Example $h_\theta(x) = \theta_1 x$

J(0) ≈ 2.3
J(0.5) ≈ 0.58
J(1)=0

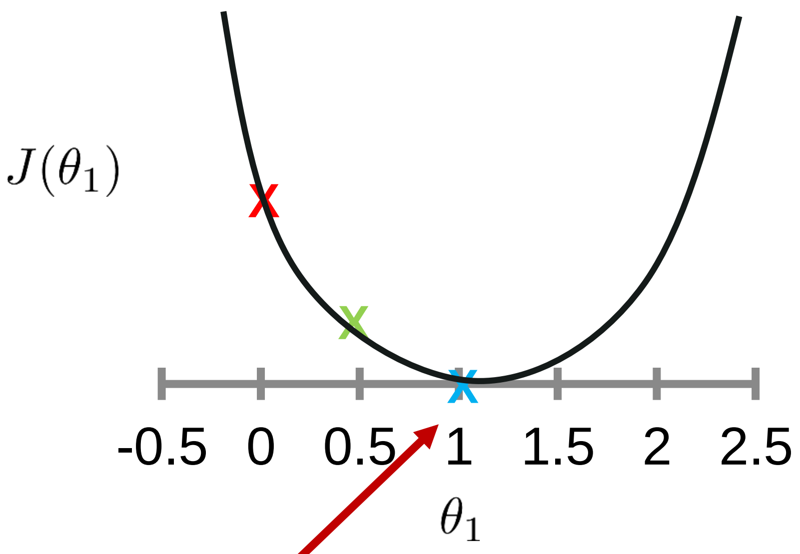
$h_\theta(x)$

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



$J(\theta_1)$

(function of the parameter θ_1)



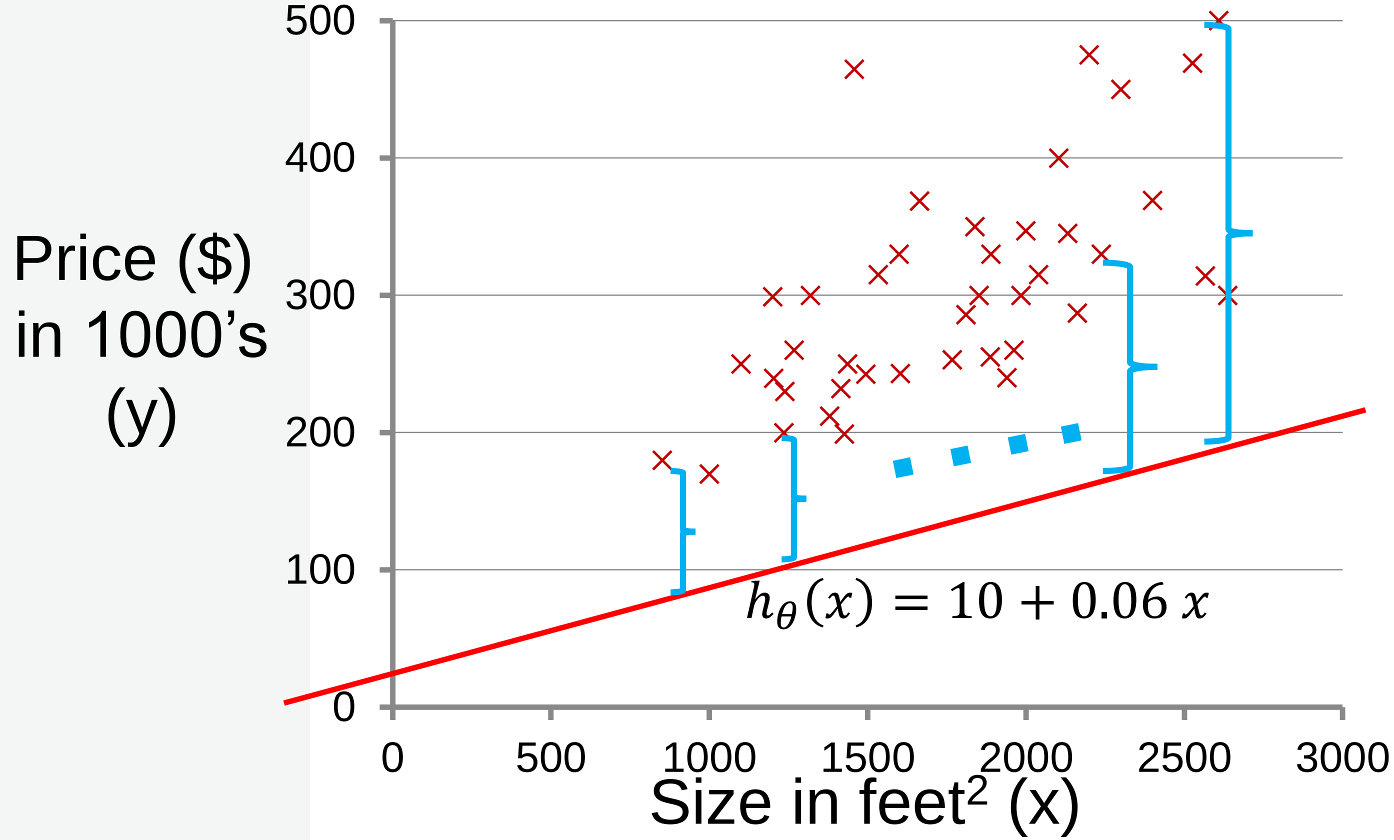
Minimum cost = Best fit line (our objective)

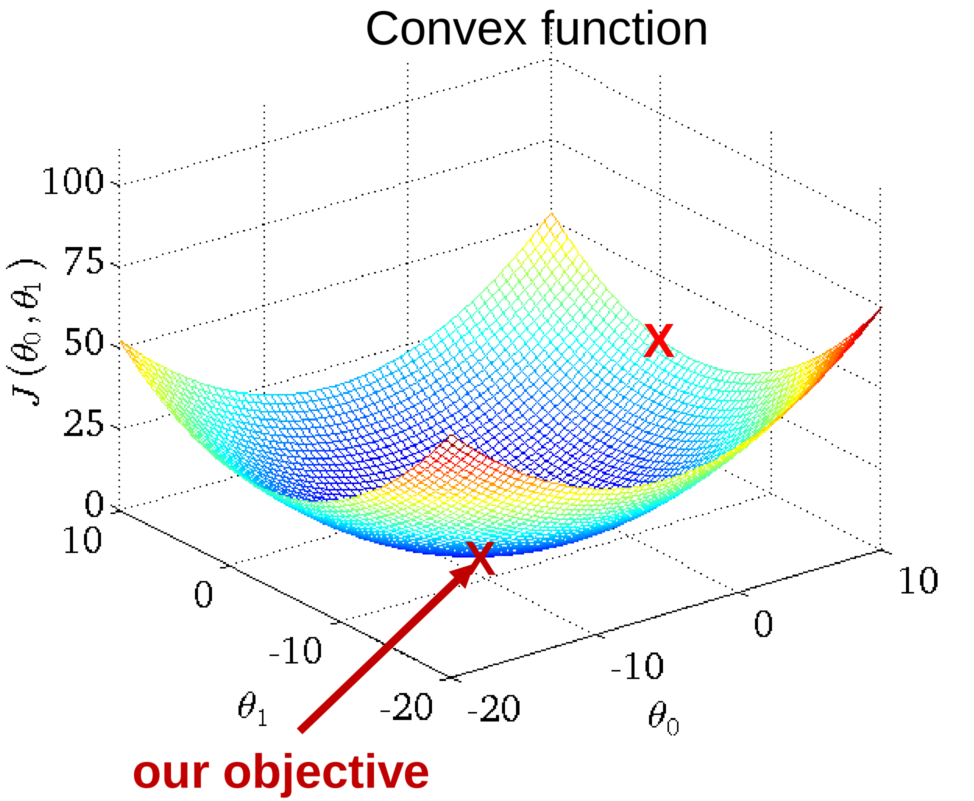
Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

$J(\theta_0, \theta_1)$

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

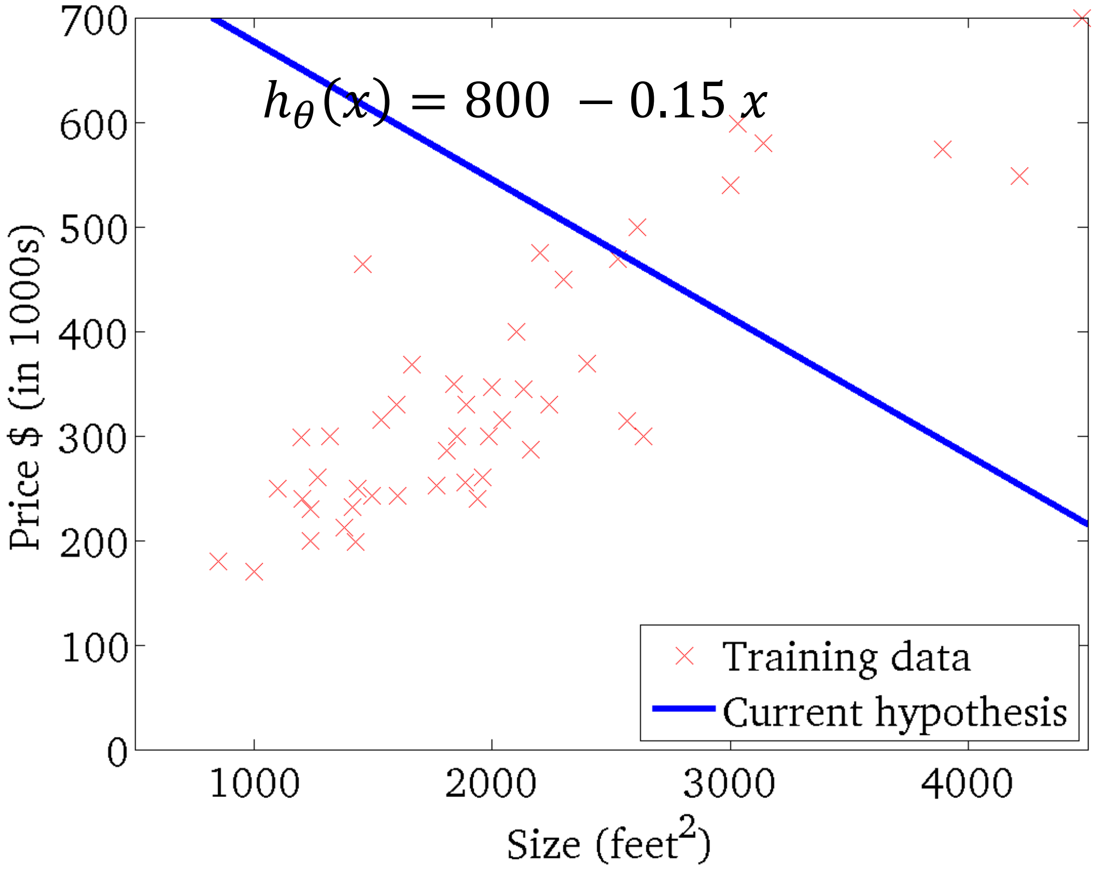
(function of the parameters θ_0, θ_1)





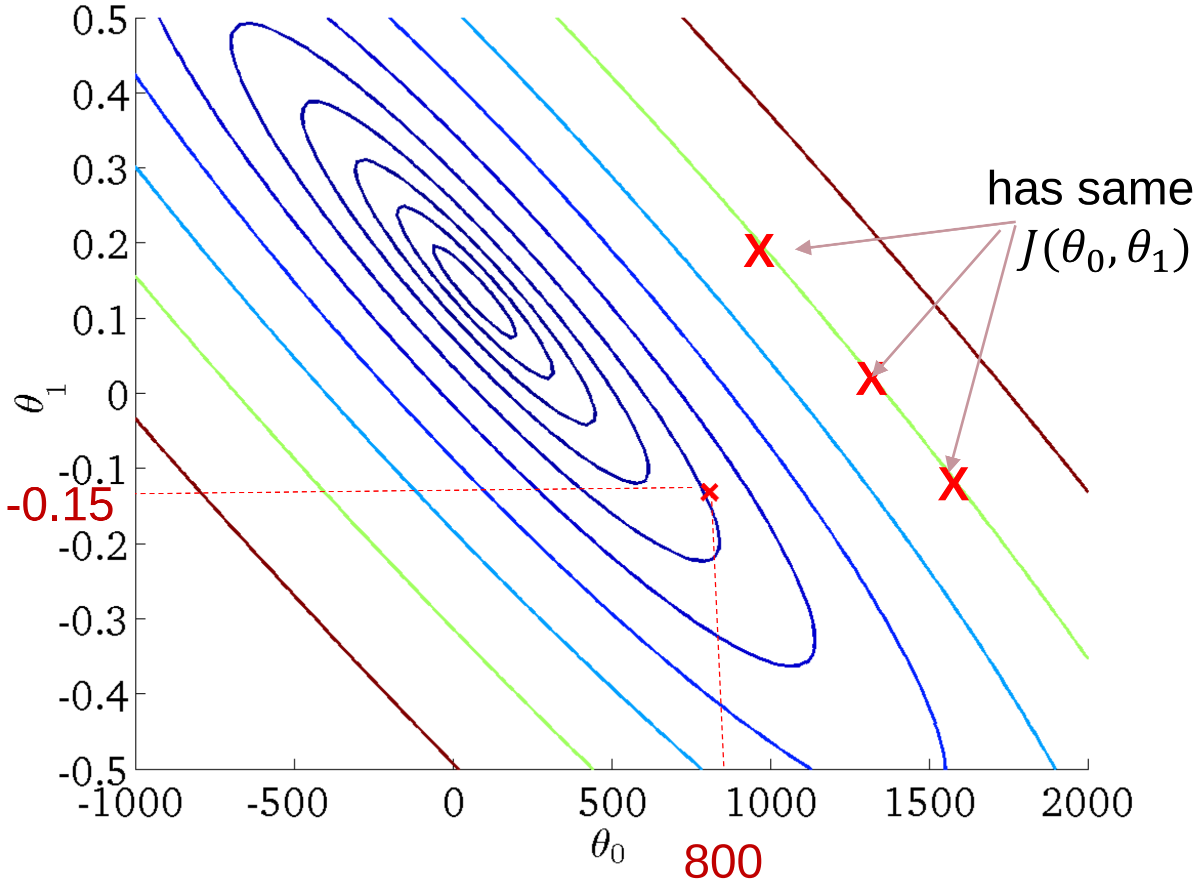
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

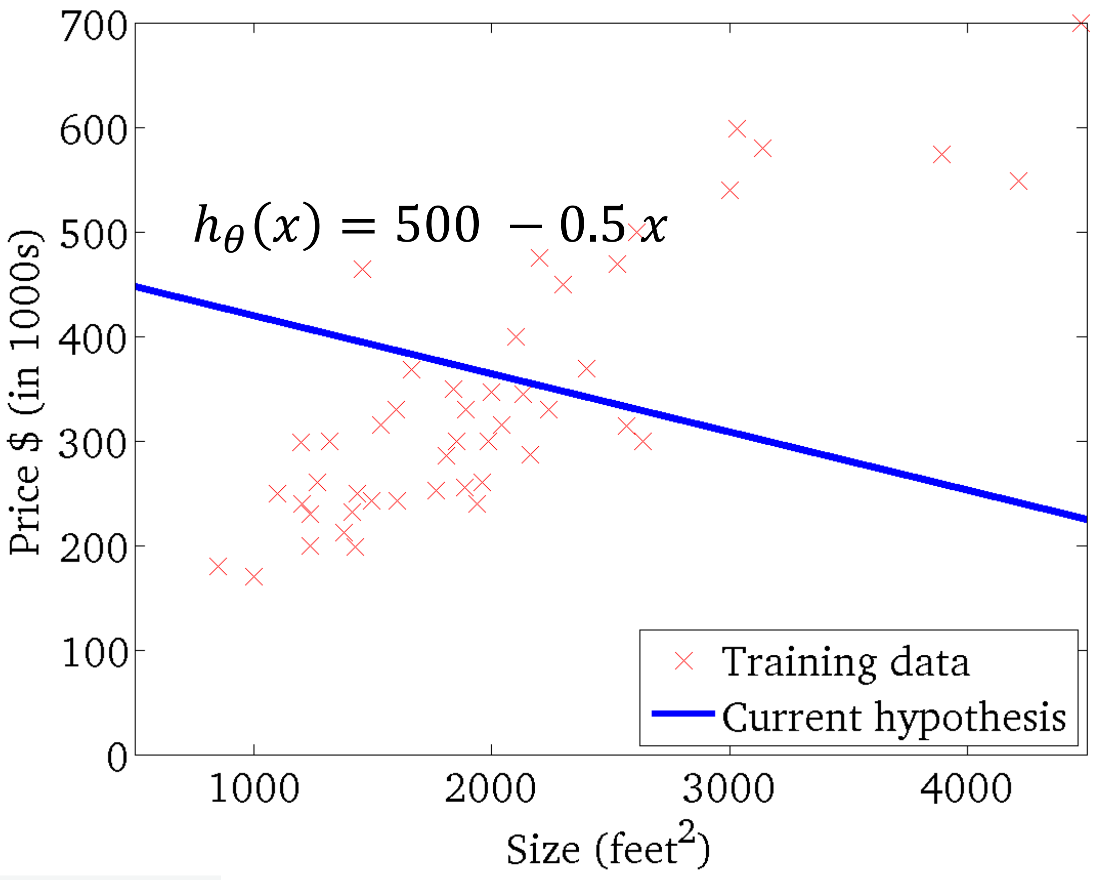
(function of the parameters θ_0, θ_1)

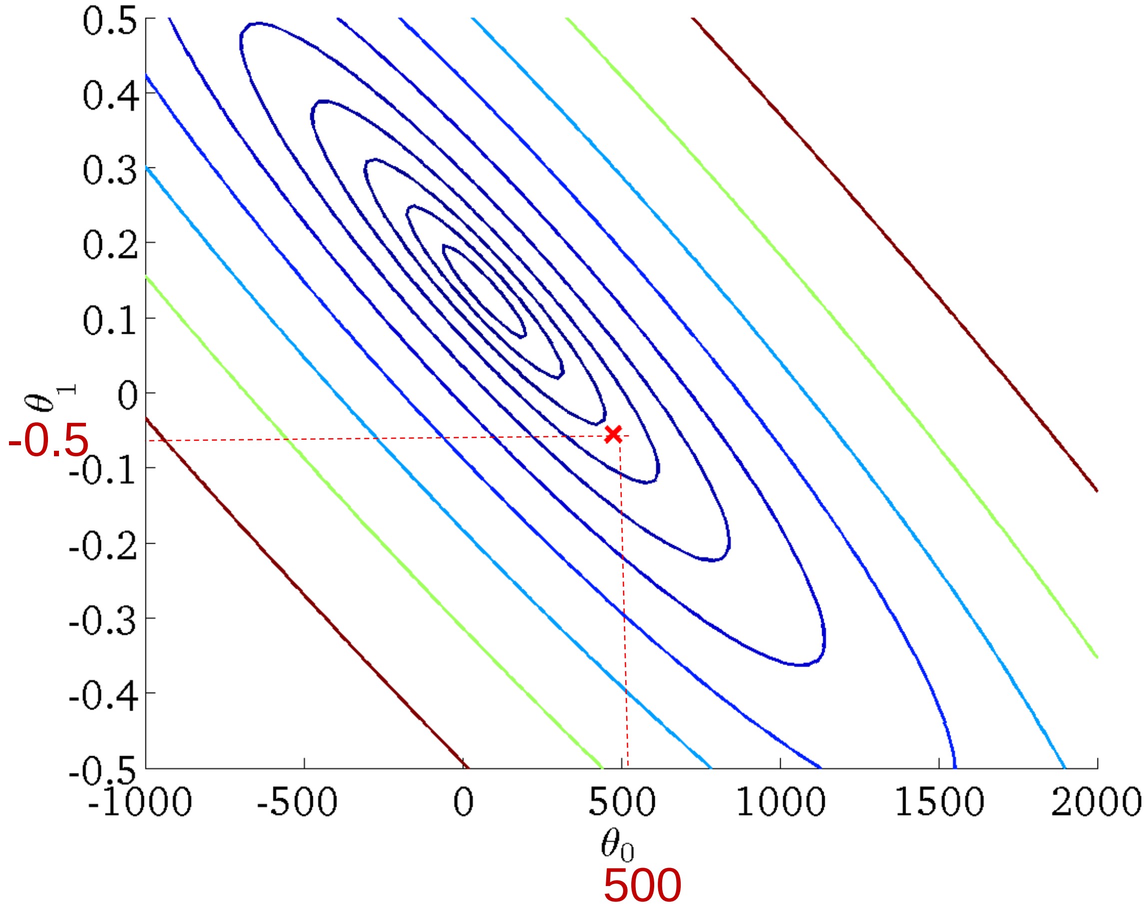


“Contour plots” or “Contour figures”

$h_\theta(x)$ $J(\theta_0, \theta_1)$

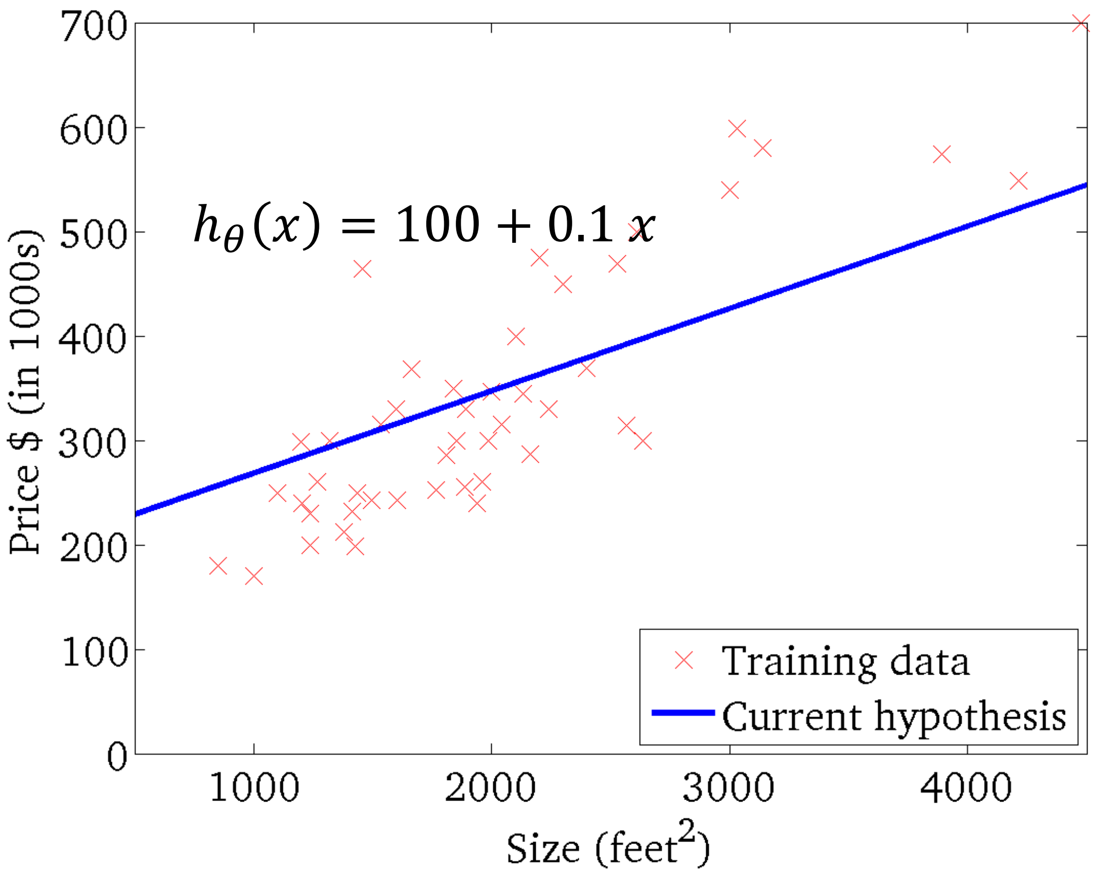
(for fixed θ_0, θ_1 , this is a function of x) (function of the parameters θ_0, θ_1)





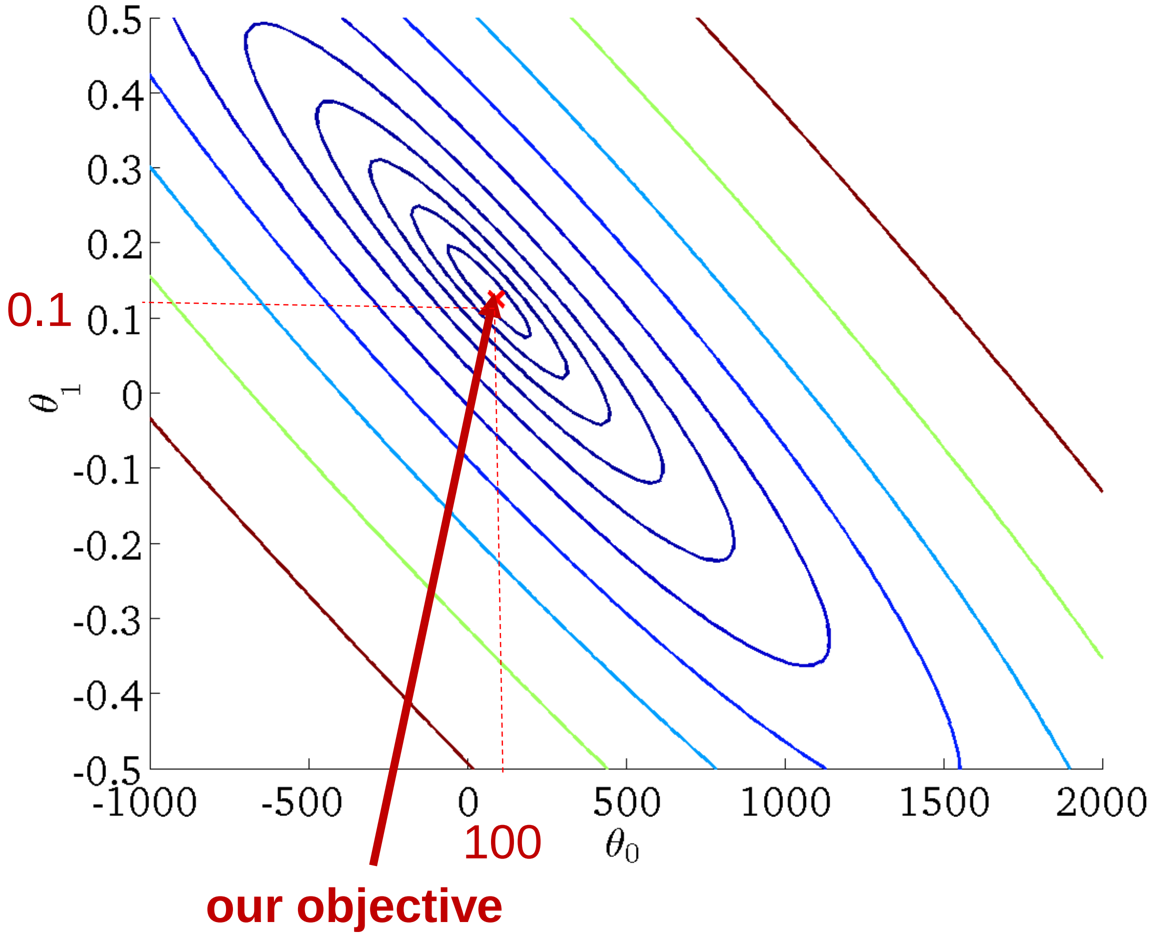
$h_\theta(x)$

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

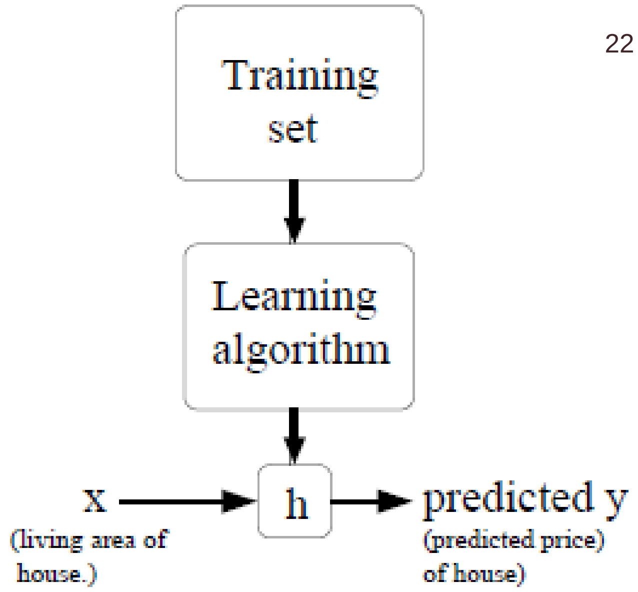


$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



Quick Summary until now



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

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

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

Our strategy until now: Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at a minimum!!

Andrew Ng

Optimization

Optimization is the process of finding the set of parameters/weights θ_0, θ_1 that minimize the **cost function**.

Strategy 1: A first very bad idea solution: RANDOM SEARCH

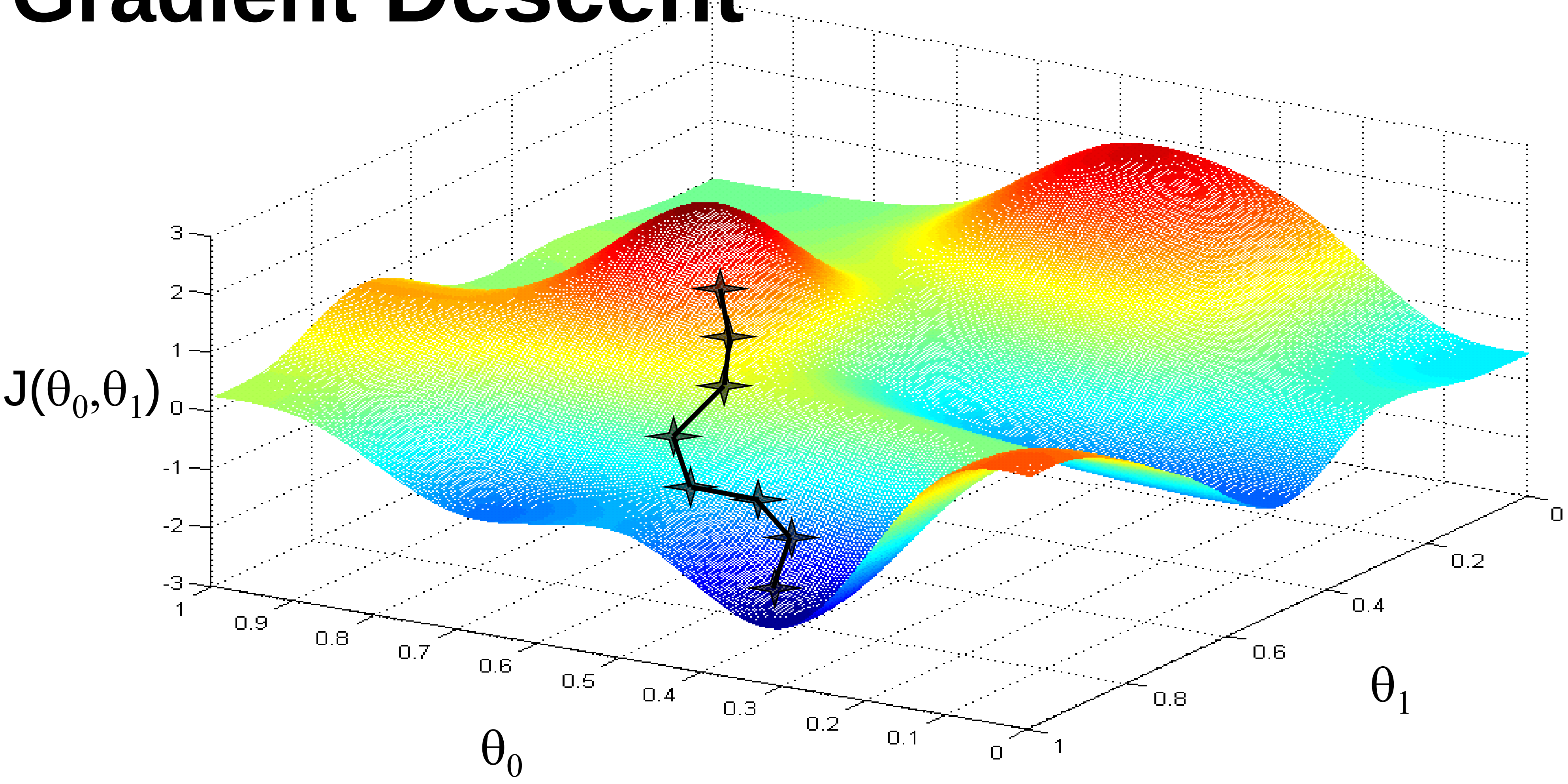
- is to simply try out many different random parameters and keep track of what works best. (iterative refinement.)
- start with random weights and iteratively refine them over time to get lower cost

Optimization

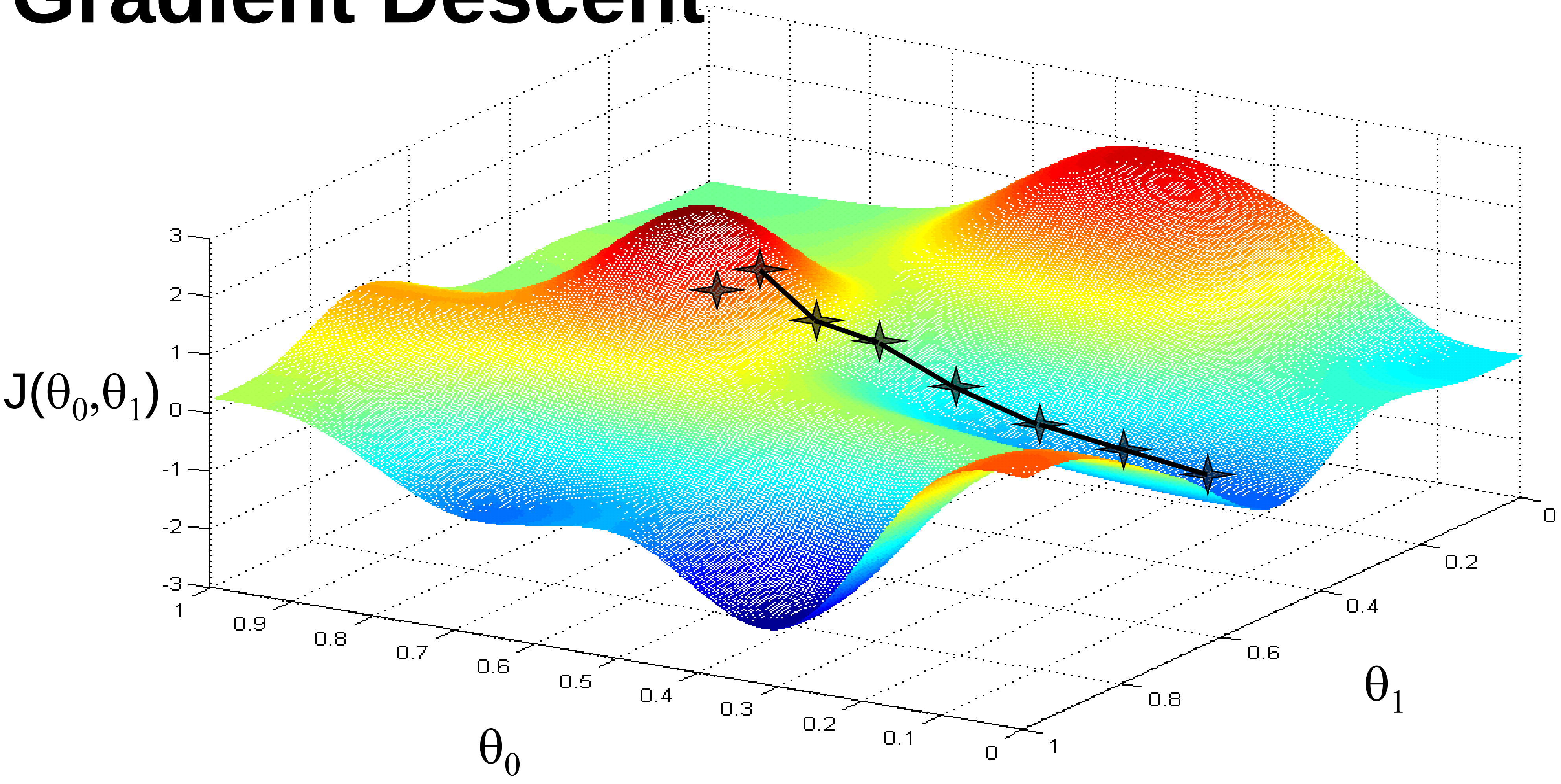
Strategy 2: Following the Gradient

- Compute the best direction along which we should change our parameter (weight) vector that is mathematically guaranteed to be the direction of the **steepest descend.**
- This direction will be related to the **gradient** of the cost function.

Gradient Descent

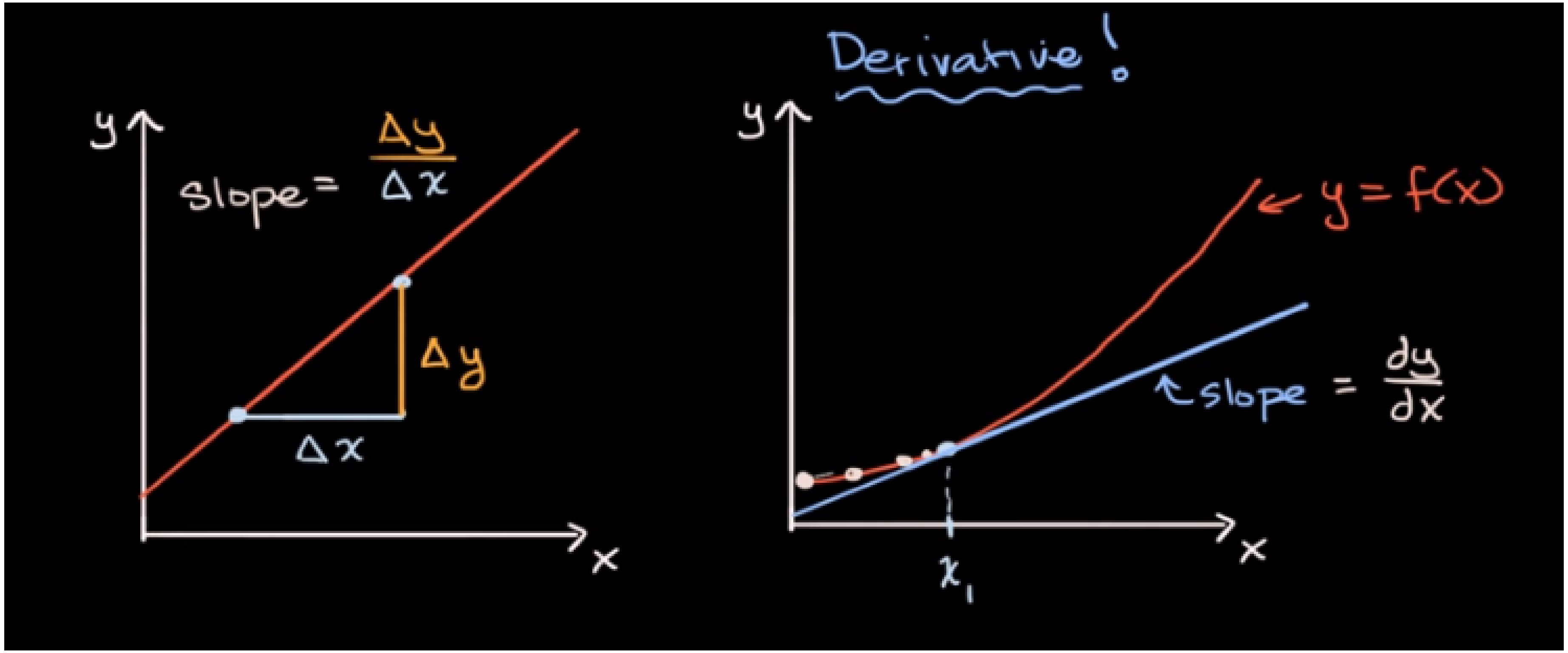


Gradient Descent



Recap: Slope & derivatives

- In one-dimensional functions, the **slope** is the instantaneous rate of change of the function at any point you might be interested in.

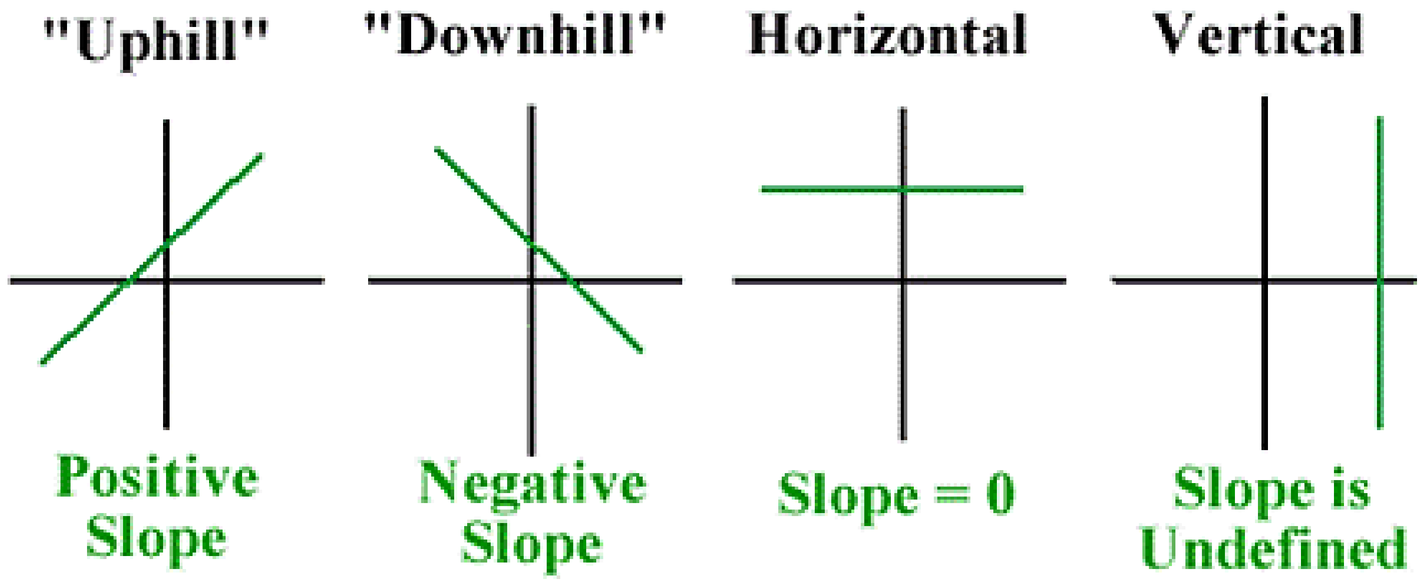


The derivative tells us the slope of a function at any point.

Recap: Derivatives (Slope of tangent line)

Derivative of function = Slope of tangent line at any point

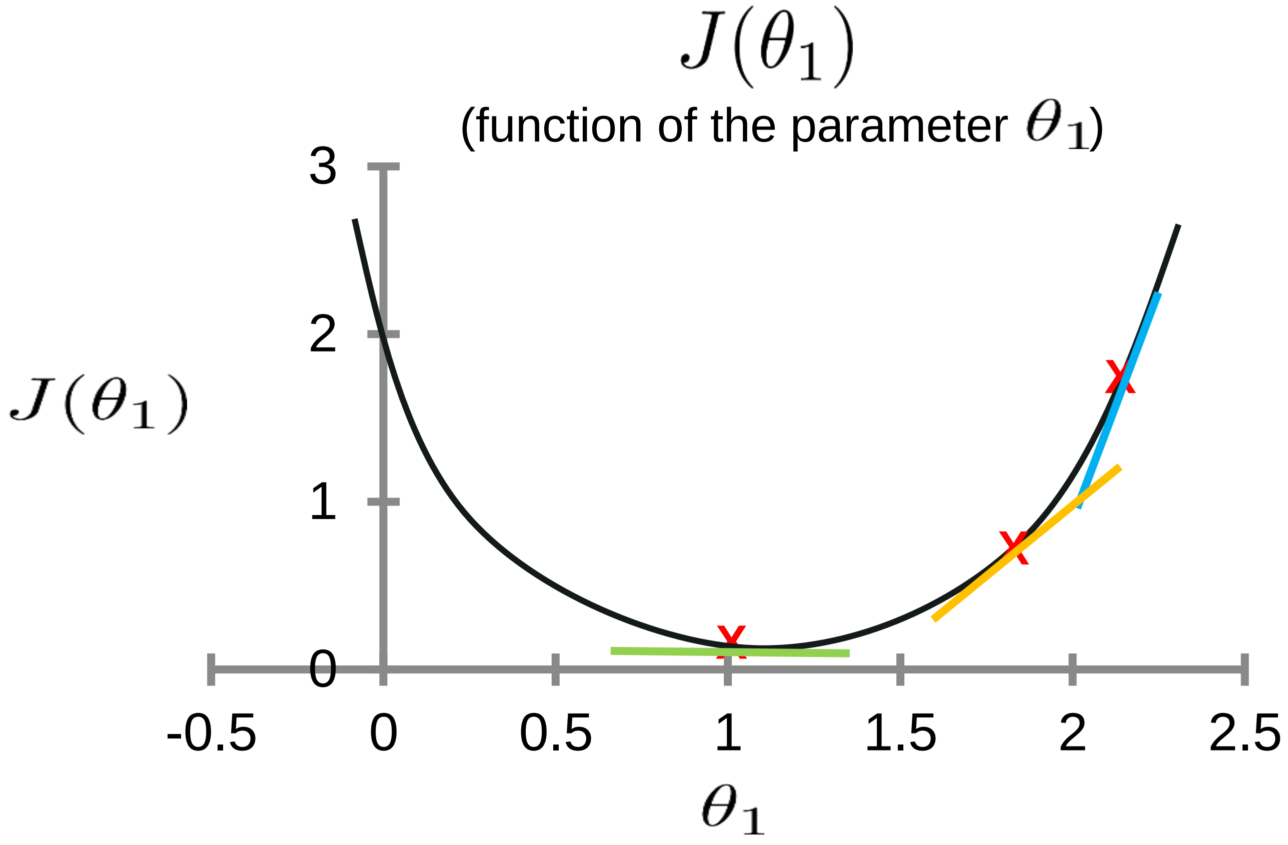
- Slopes of tangent line is a **scalar value**, it can be positive or negative.
  A **positive slope** moves upward on a graph from left to right.
  A **negative slope** moves downward on a graph from left to right.



https://sites.google.com/site/cmp3extrapractice/thinking-with-mathematical-models/slope-and-writing-linear-equations

Gradient Descent

- The way we do this is by taking the derivative (the tangential line to a function) of our cost function.
- The **slope of the tangent** is the derivative at that point and it will give us a direction to move towards.
- We will know that we have succeeded when our cost function is at the very bottom of the pits in our graph, i.e. when its value is the minimum.



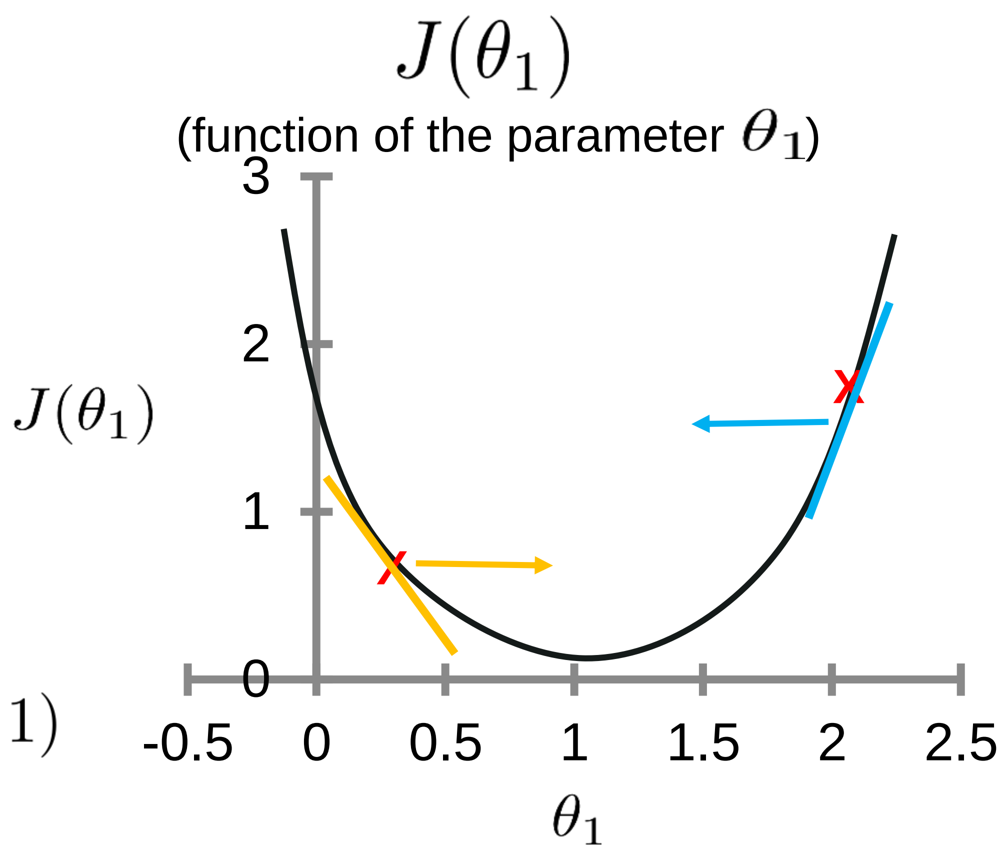
Gradient Descent

- We make steps down the cost function in the direction with the steepest descent, and the size of each step is determined by the parameter **α**, which is called the **learning rate.**
- The gradient descent algorithm is:
- Repeat until **convergence** {

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

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

Learning rate (step size)



Positive slope (positive number)→ Θ will decrease
Negative slope (negative number)→ Θ will increase

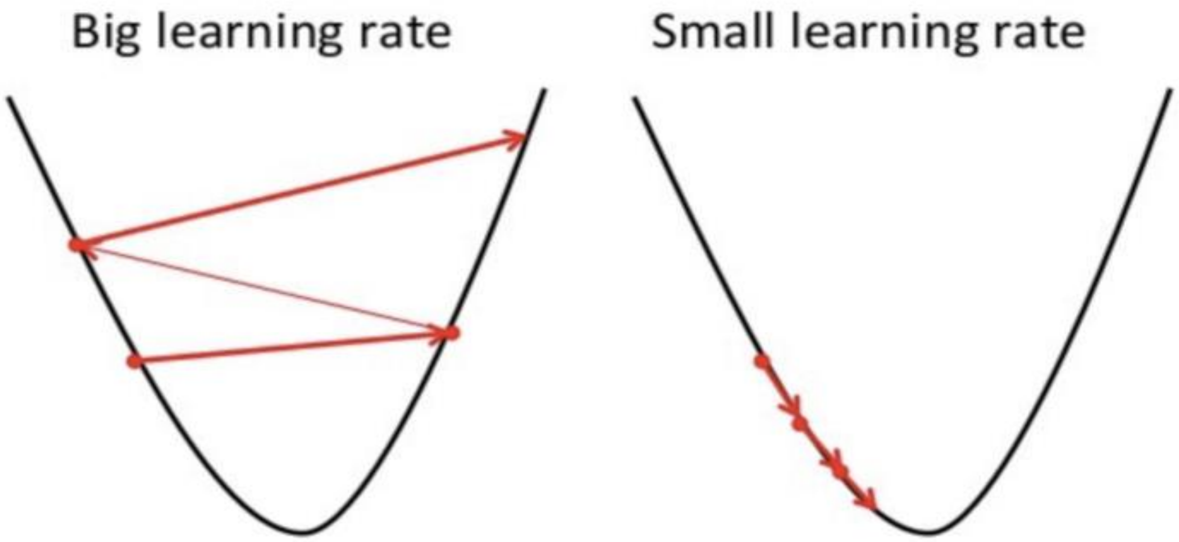
Learning Rate

- The gradient tells us the direction, but it does not tell us how far along this direction we should step.

- The **learning rate (step size)** determines how big the step would be on each iteration. It determines how fast or slow we will move towards the optimal weights.

Learning Rate

- If **learning rate is large**, it may fail to converge and overshoot the minimum.
- If **learning rate is very small**, it would take long time to converge and become computationally expensive.
- <u>The most commonly used rates are</u> :
- *0.001, 0.003, 0.01 (default), 0.03, 0.1, 0.3*



Two ways to compute the gradient

- There are two ways to compute the gradient:

1) **Numerical gradient:** A slow, approximate but easy way to implement. Approximate (since we have to pick a small value of *h*, while the true gradient is defined as the limit as *h* goes to zero), and that it is very computationally expensive to compute

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

2) **Analytic gradient:** A fast, exact but more error-prone way that requires **calculus**. It allows us to derive a direct formula for the gradient (no approximations) that is also very fast to compute.

Always use analytic gradient, but check implementation with numerical gradient. This is called **a gradient check**.

Gradient Descent for Linear Regression

- When specifically applied to the case of **linear regression**, a new form of the gradient descent equation can be derived. We can substitute our actual cost function and our actual hypothesis function and modify the equation to:

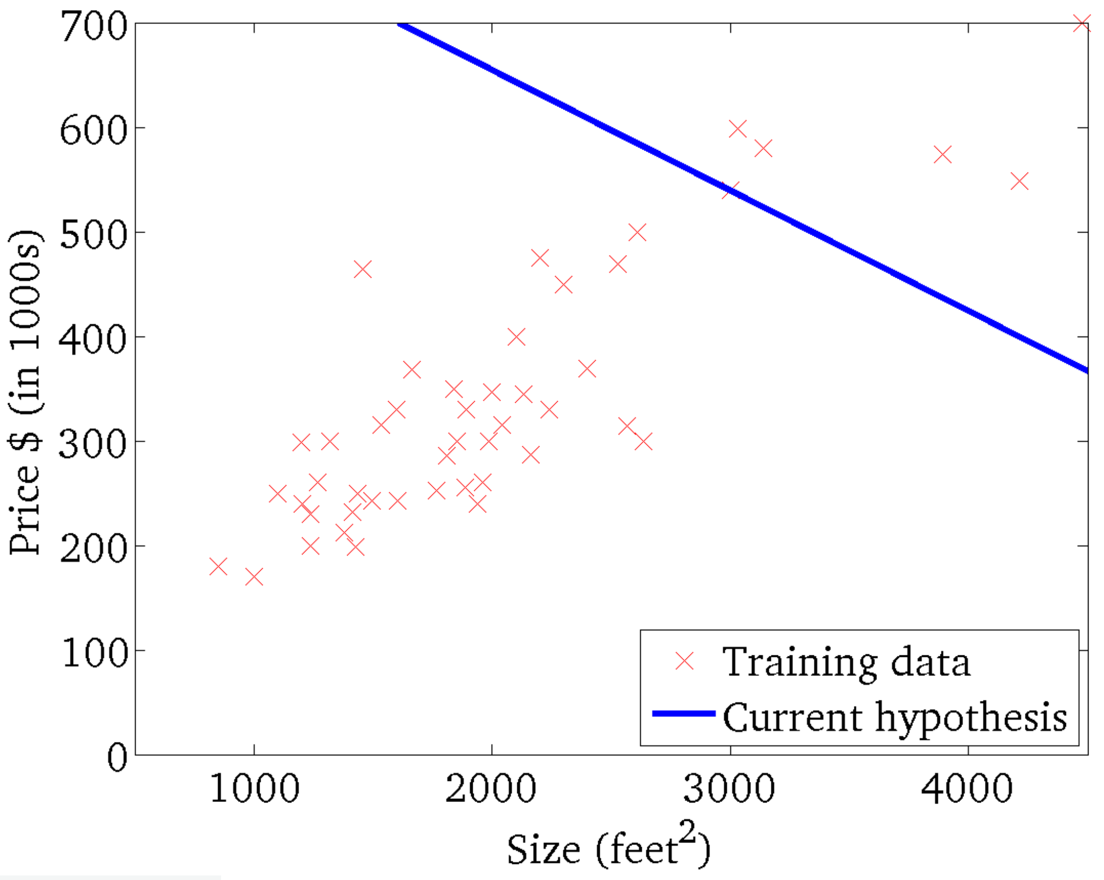
Start by initializing the parameters θ_0, θ_1 randomly

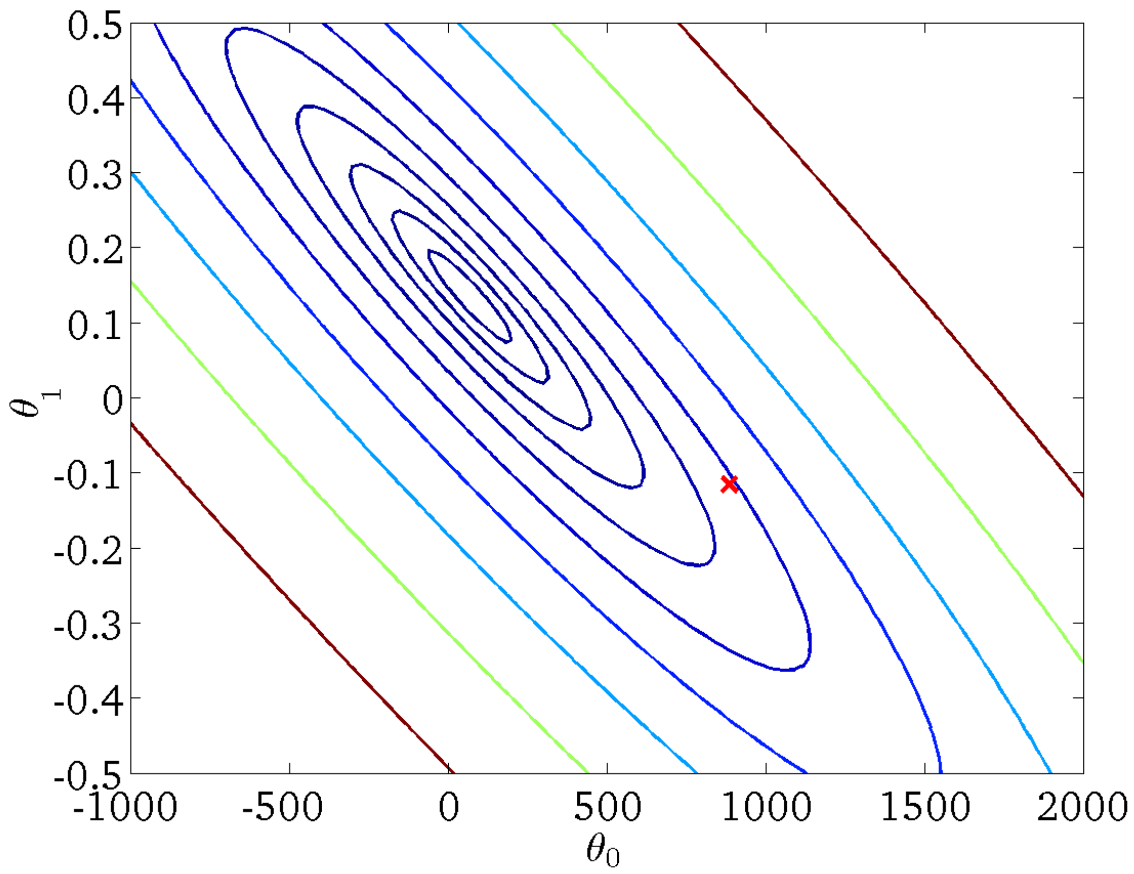
Should be done simultaneously

$$
\begin{aligned}
&\text{repeat until convergence } \{ \\
&\quad \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \qquad \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1) \\
&\quad \theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) \cdot x^{(i)} \qquad \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1) \\
&\}
\end{aligned}
$$

$h_\theta(x)$ $J(\theta_0, \theta_1)$

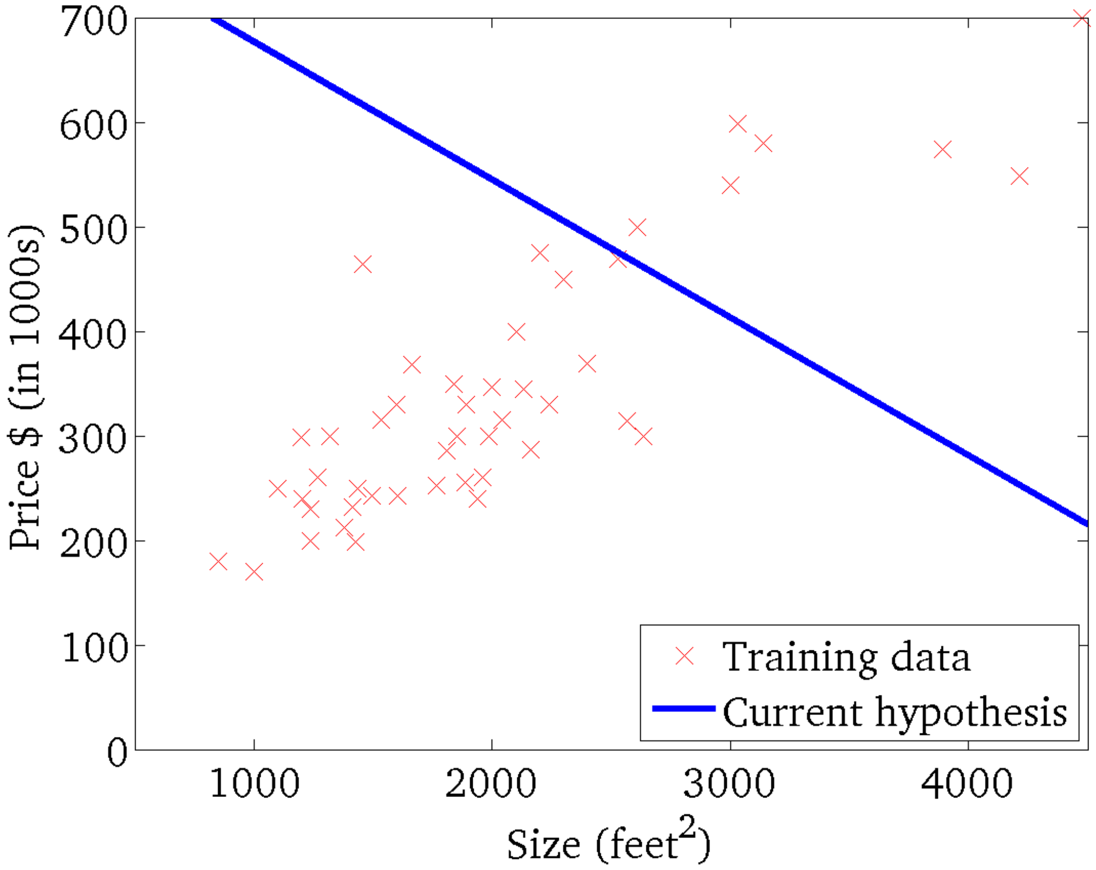
(for fixed θ_0, θ_1 , this is a function of x) (function of the parameters θ_0, θ_1)

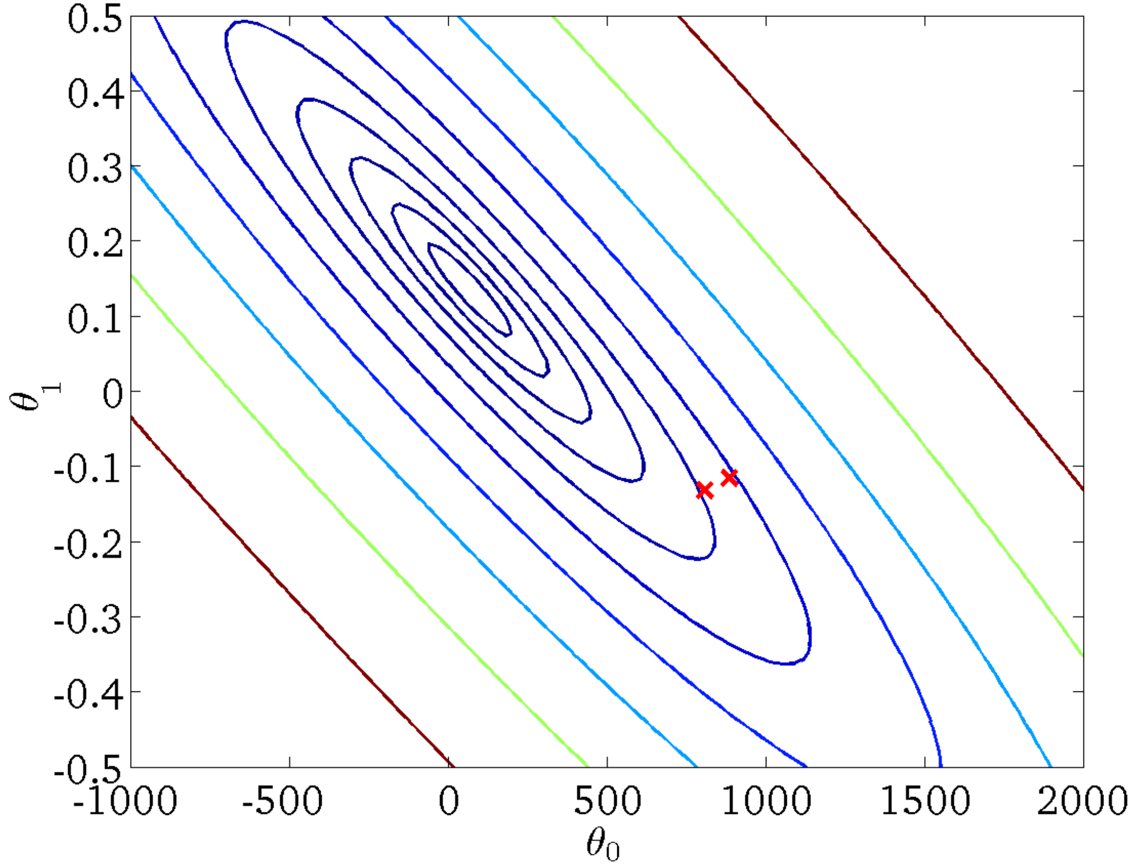




$h_\theta(x)$ (for fixed θ_0, θ_1, this is a function of x)

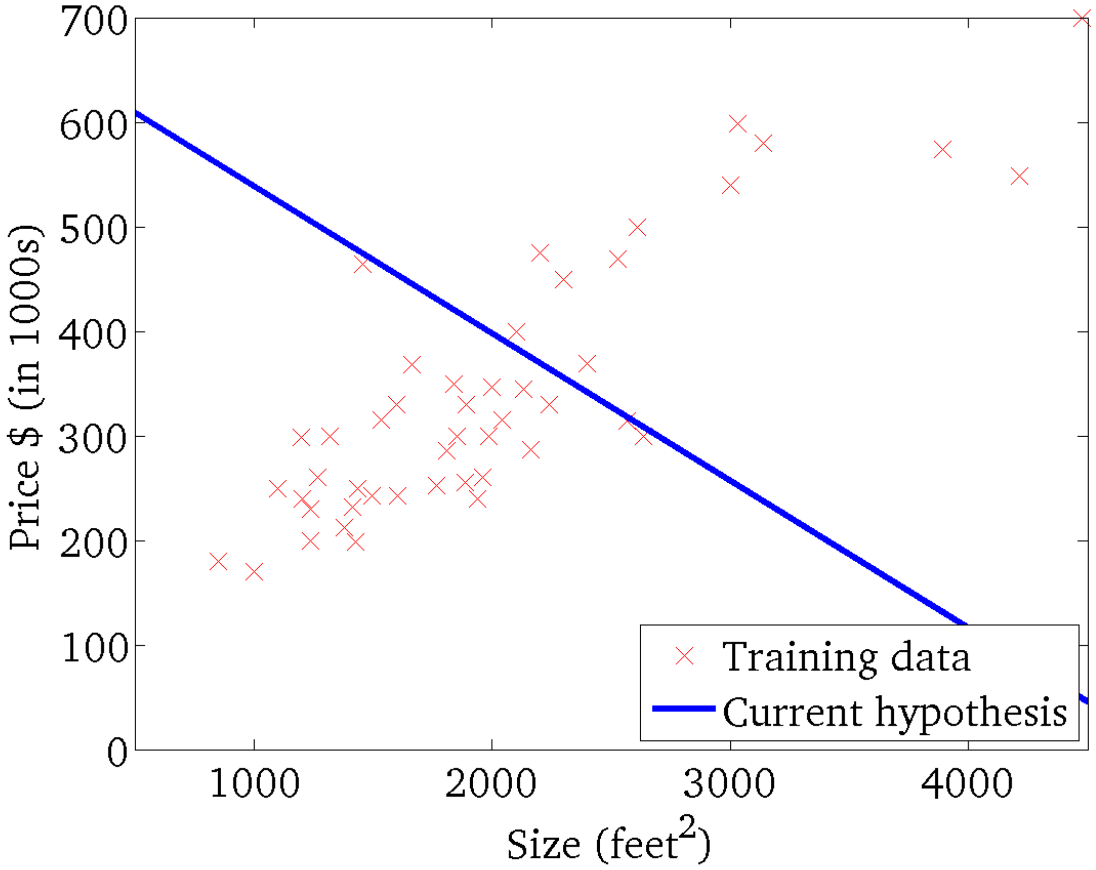
$J(\theta_0, \theta_1)$ (function of the parameters θ_0, θ_1)





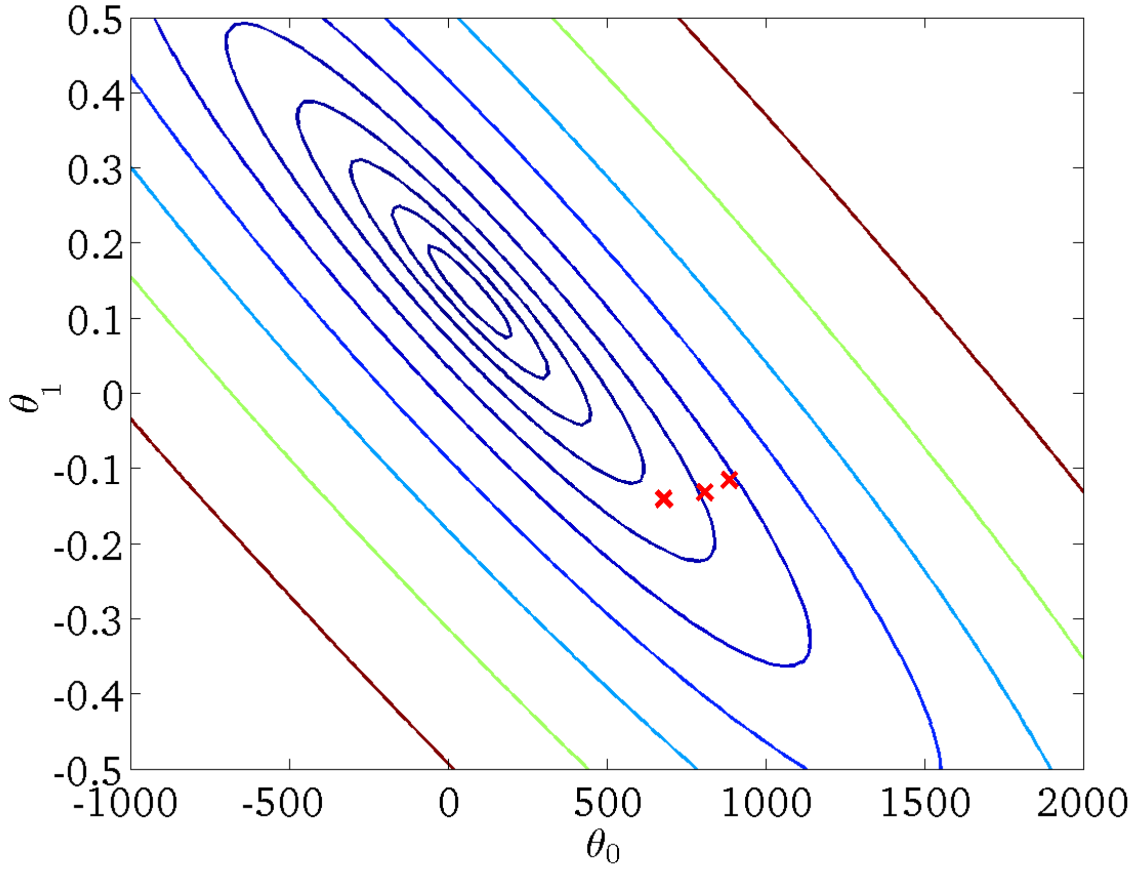
$h_\theta(x)$

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



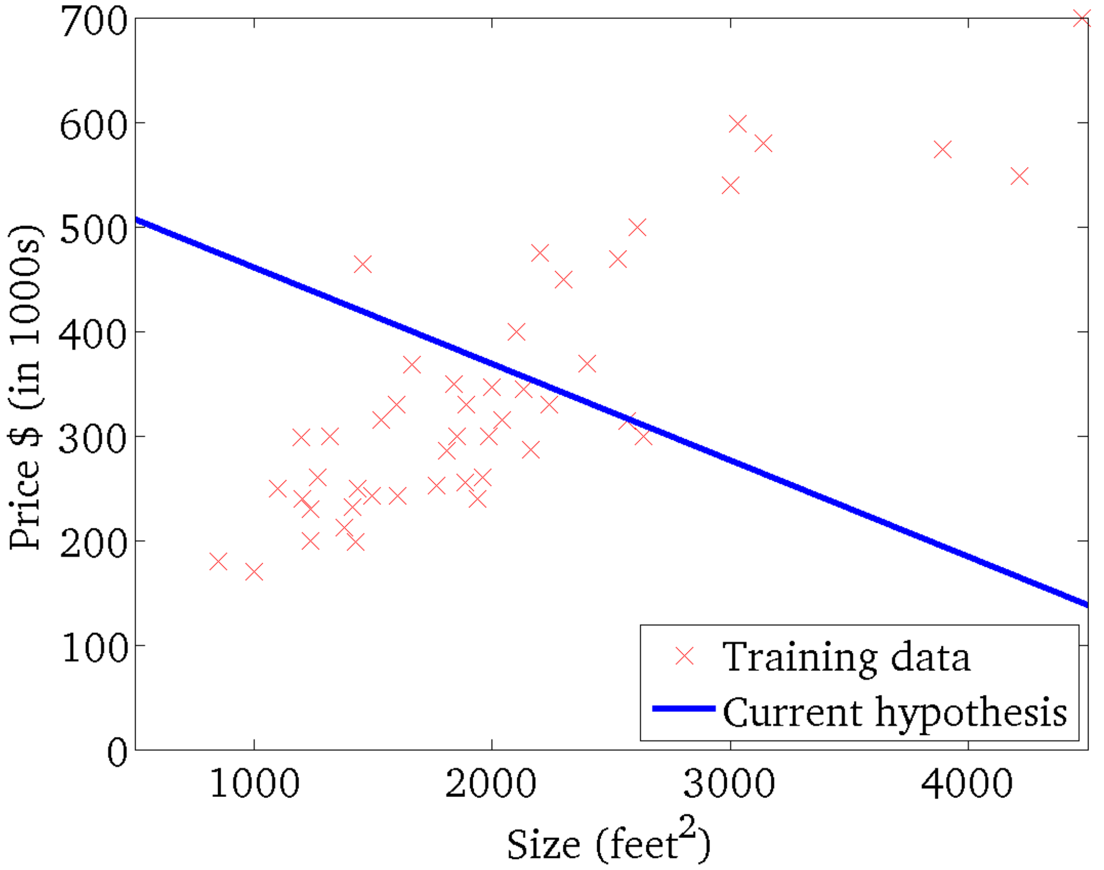
$J(\theta_0, \theta_1)$

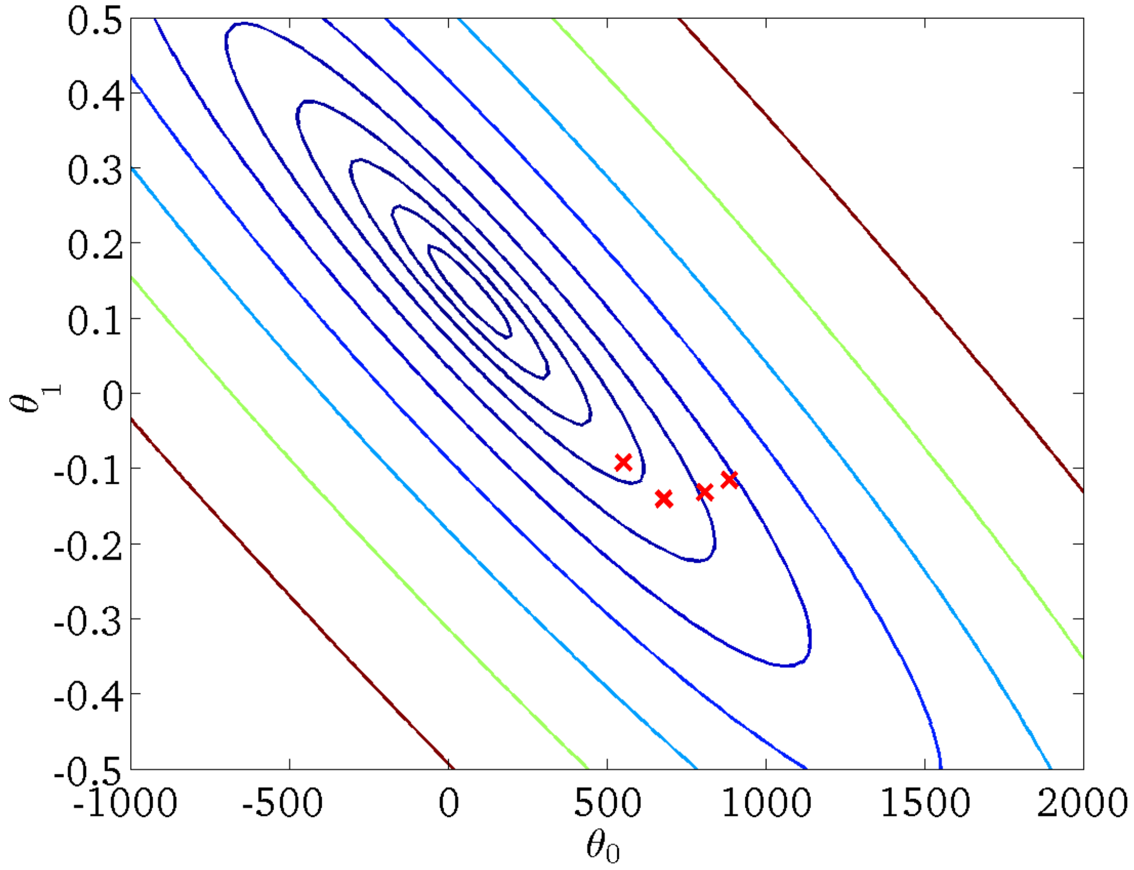
(function of the parameters θ_0, θ_1)



$h_\theta(x)$ $J(\theta_0, \theta_1)$

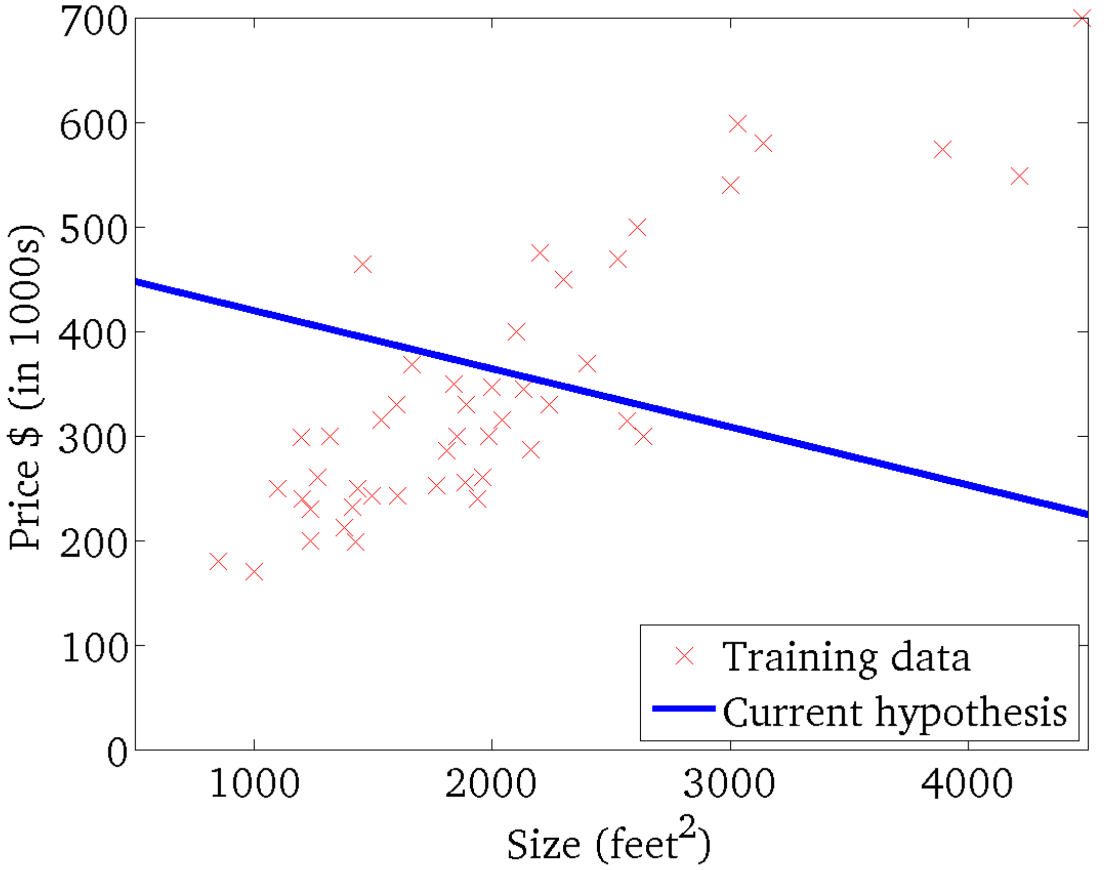
(for fixed θ_0, θ_1 , this is a function of x) (function of the parameters θ_0, θ_1)





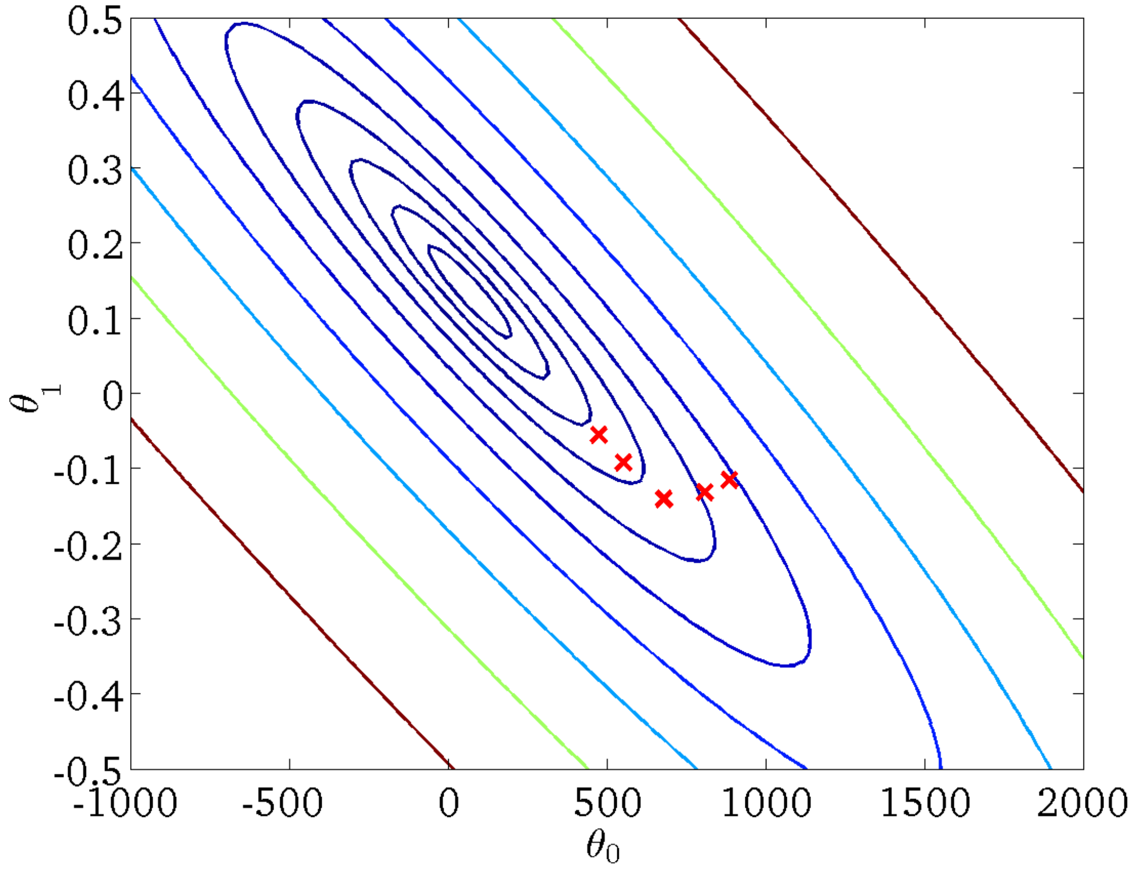
$h_\theta(x)$

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



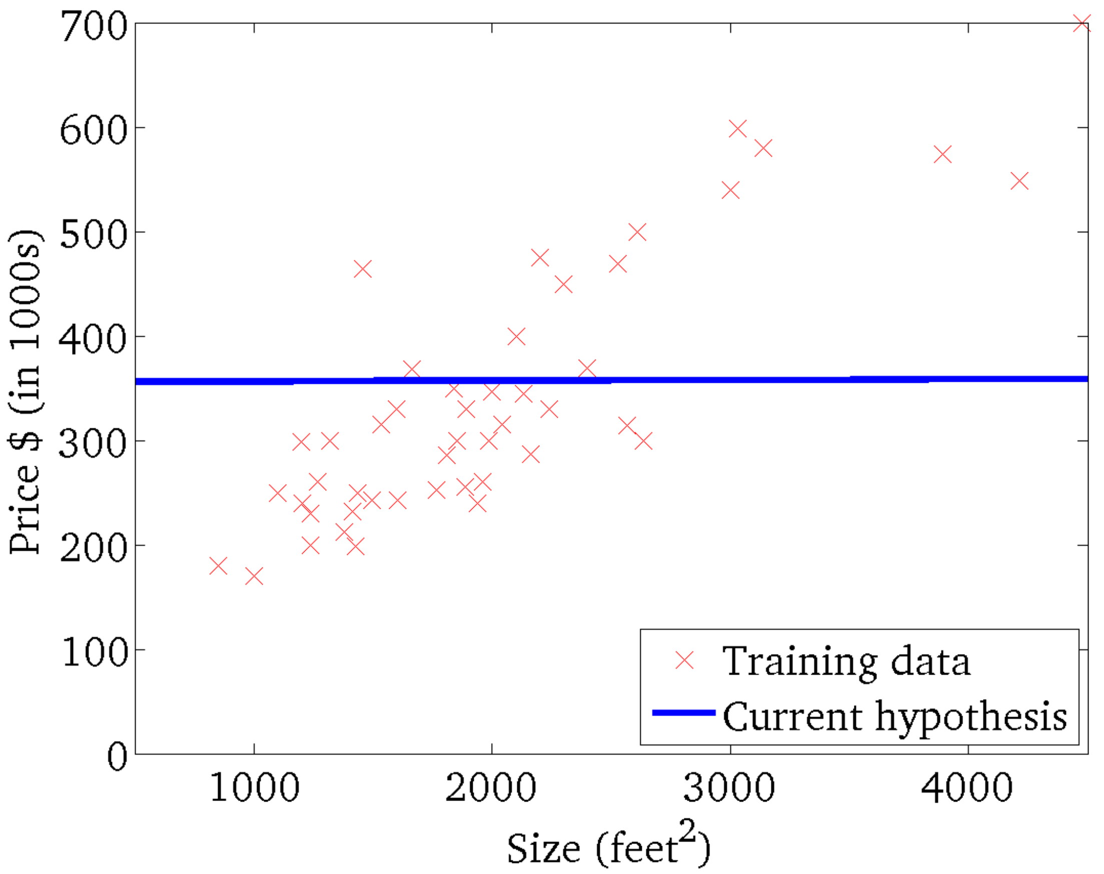
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



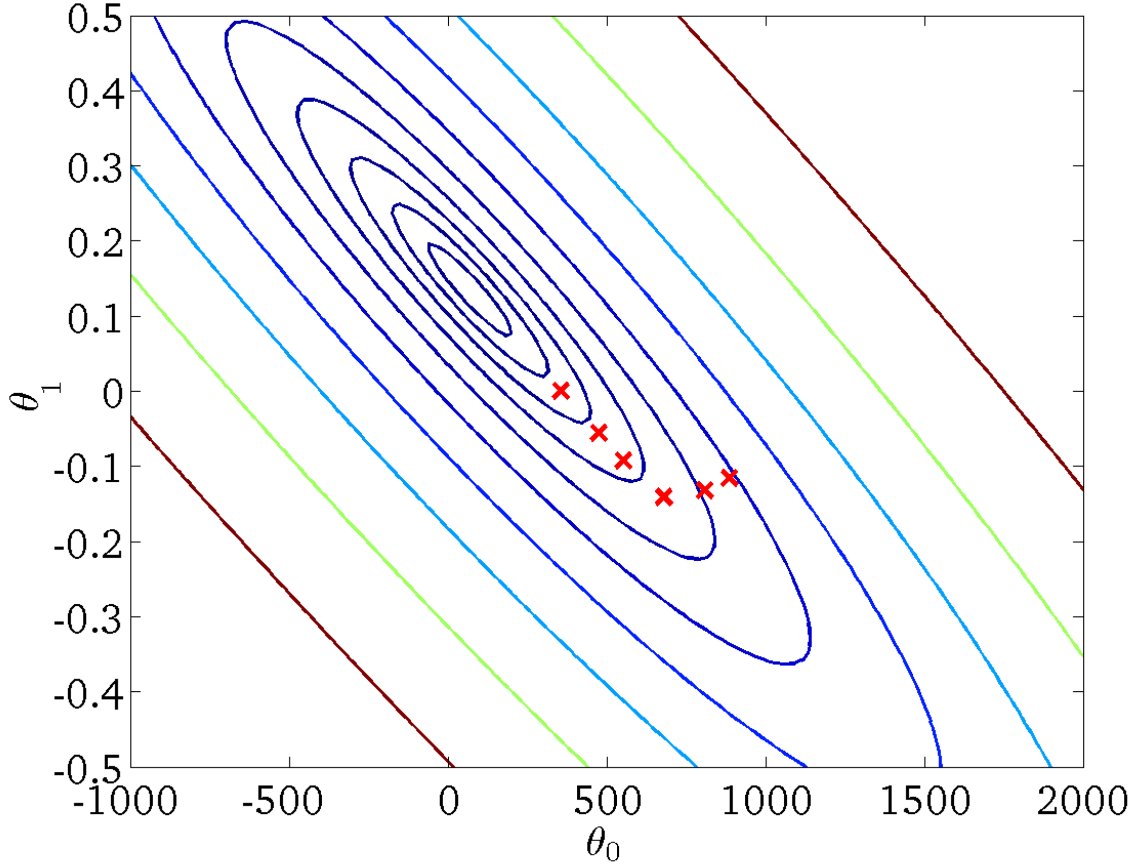
$h_\theta(x)$

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



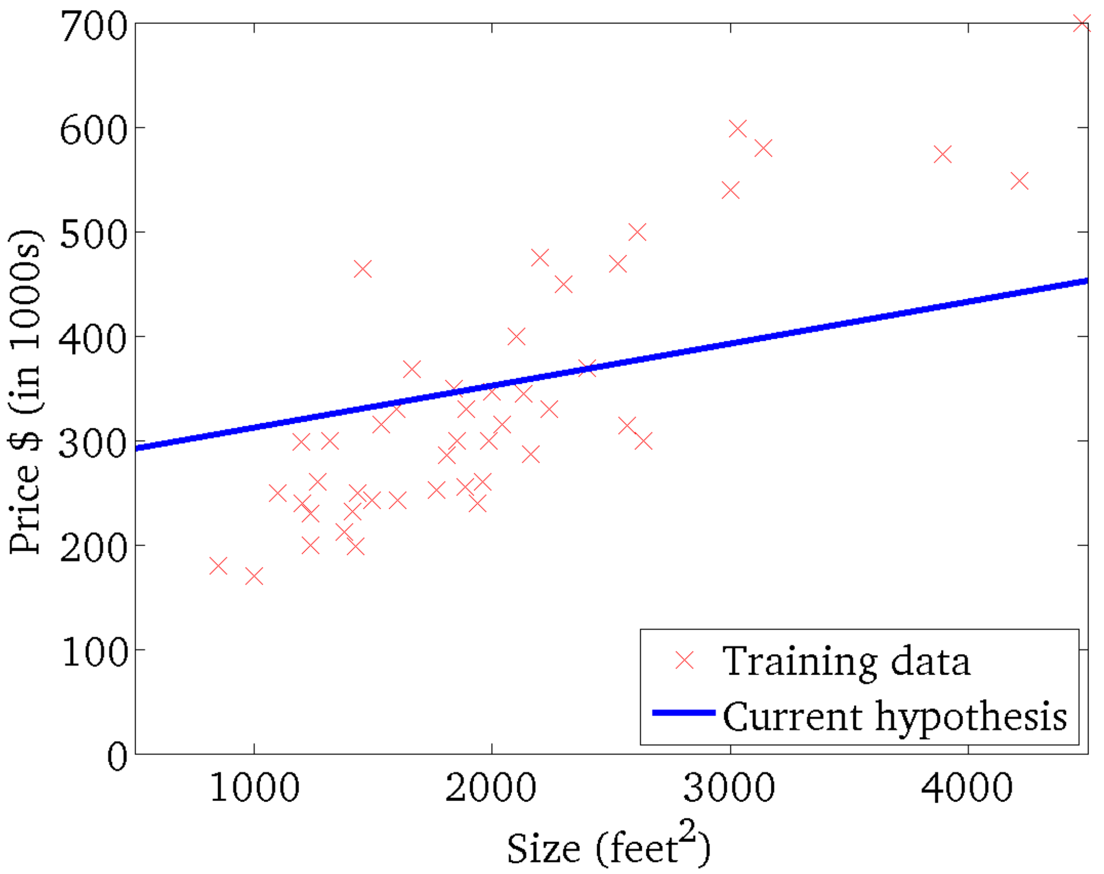
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



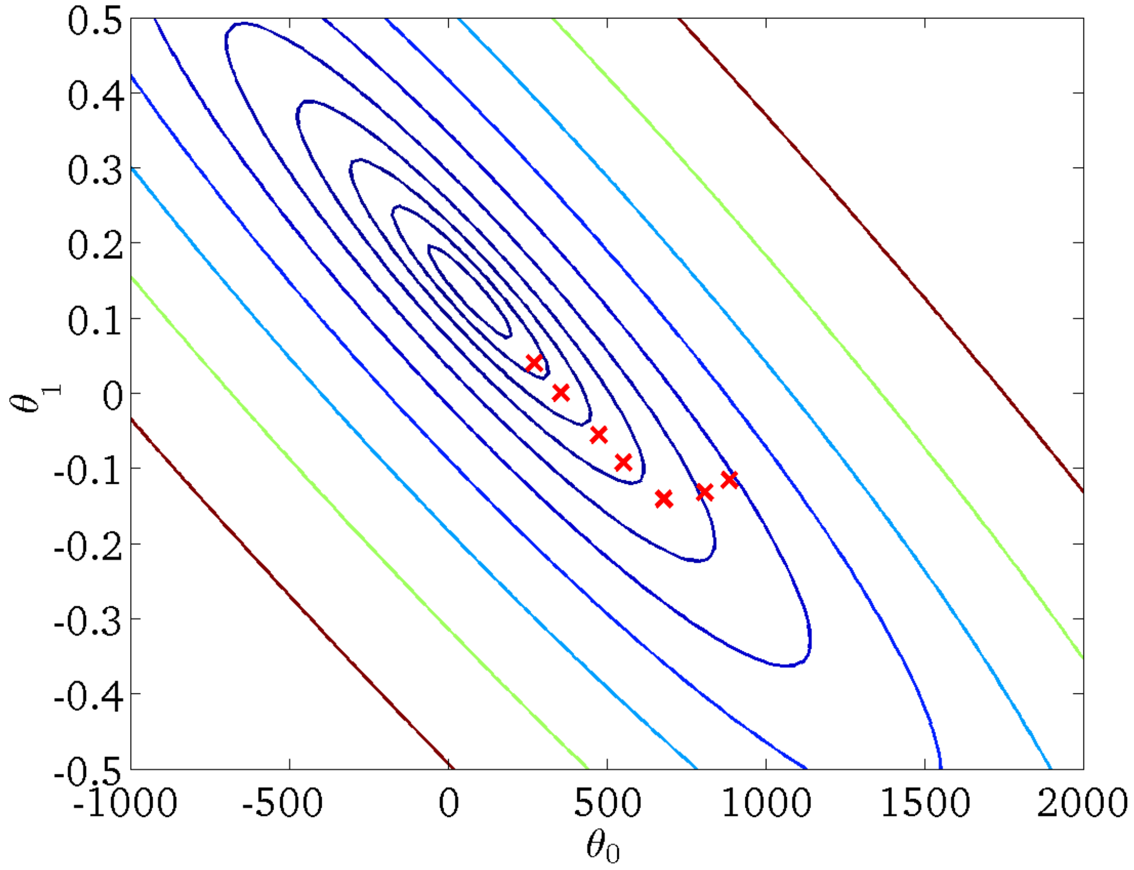
$h_\theta(x)$

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



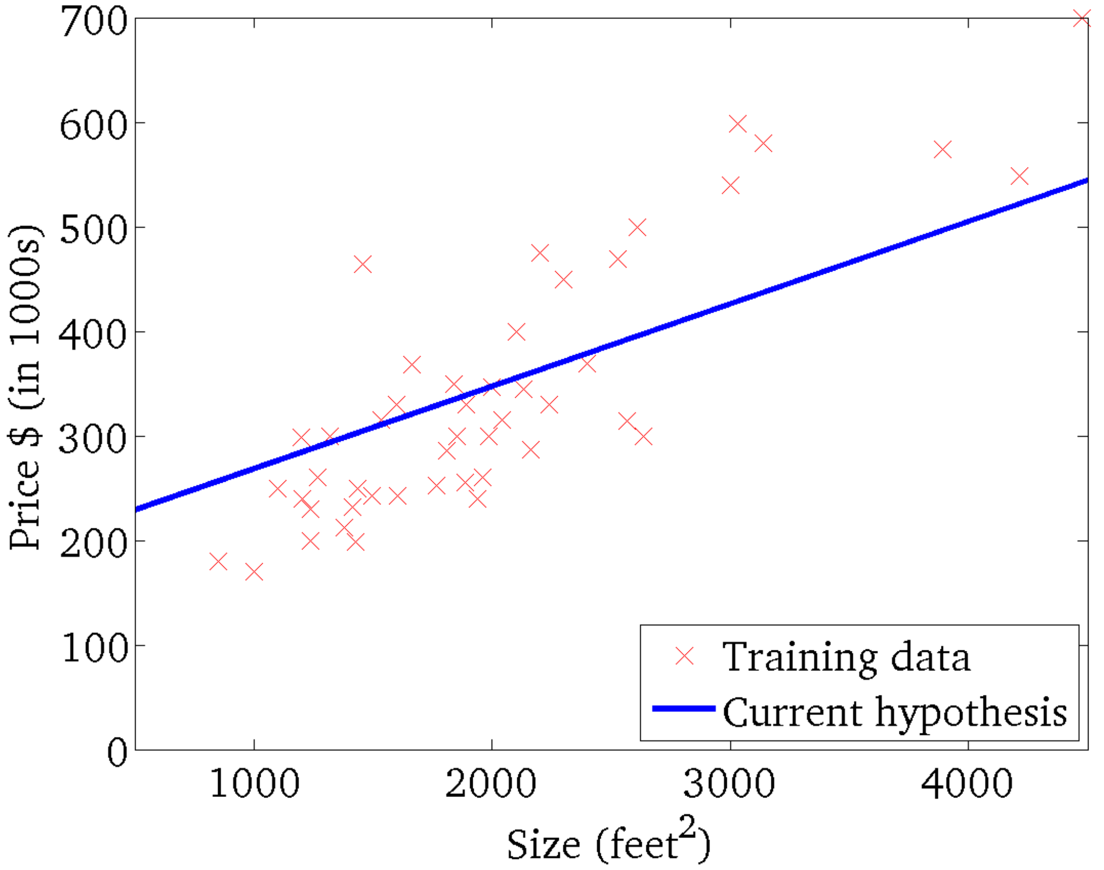
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



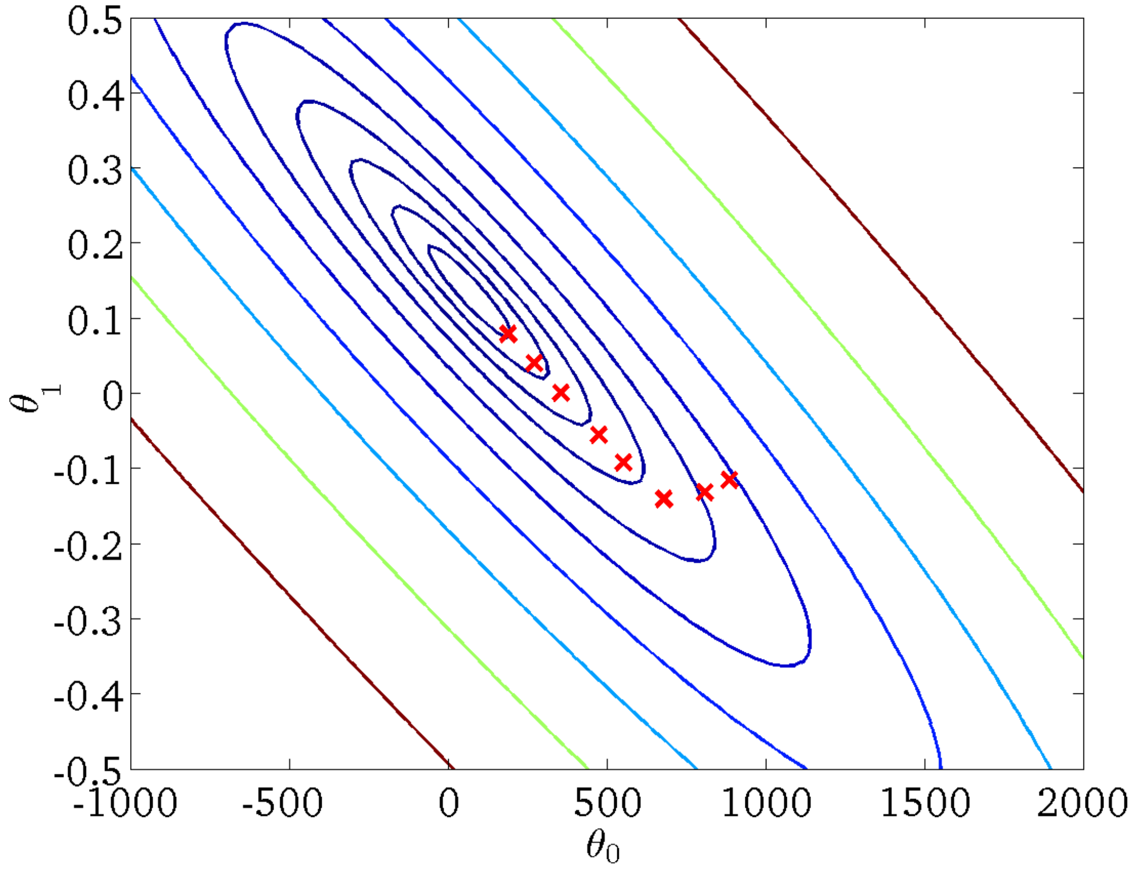
$h_\theta(x)$

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



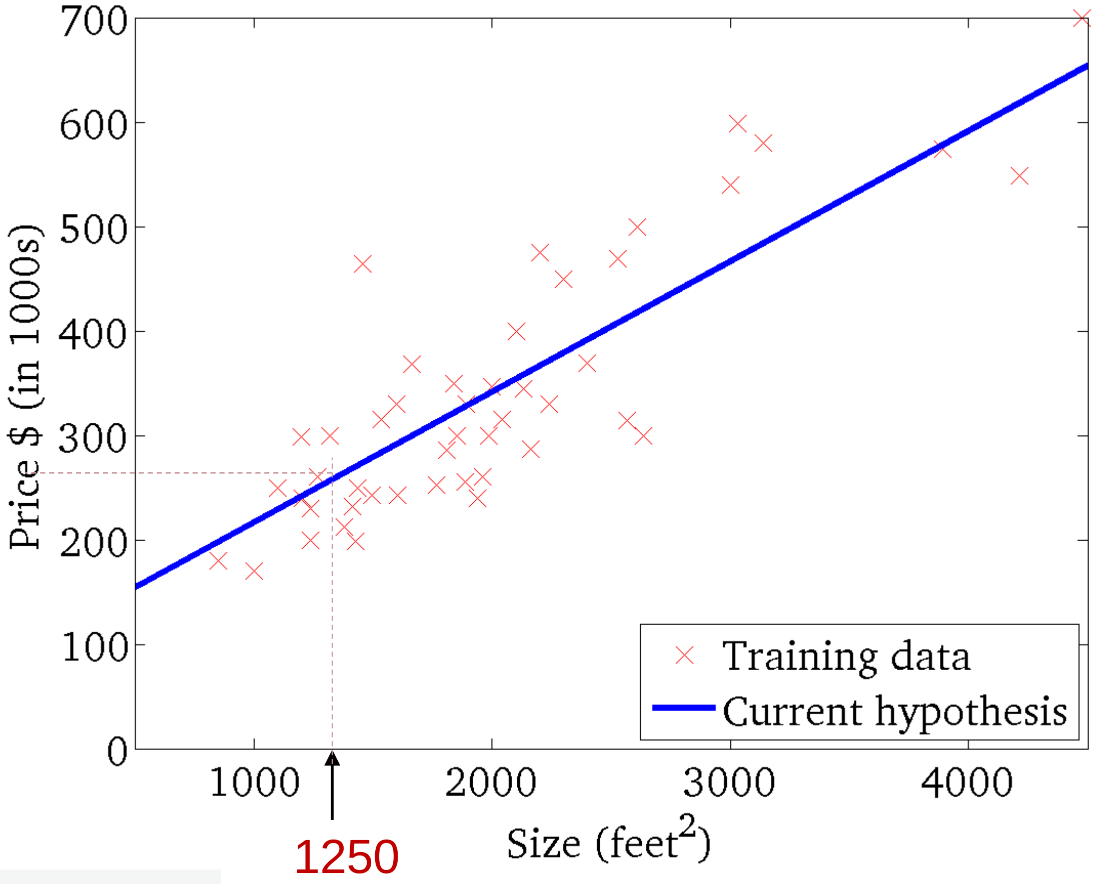
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



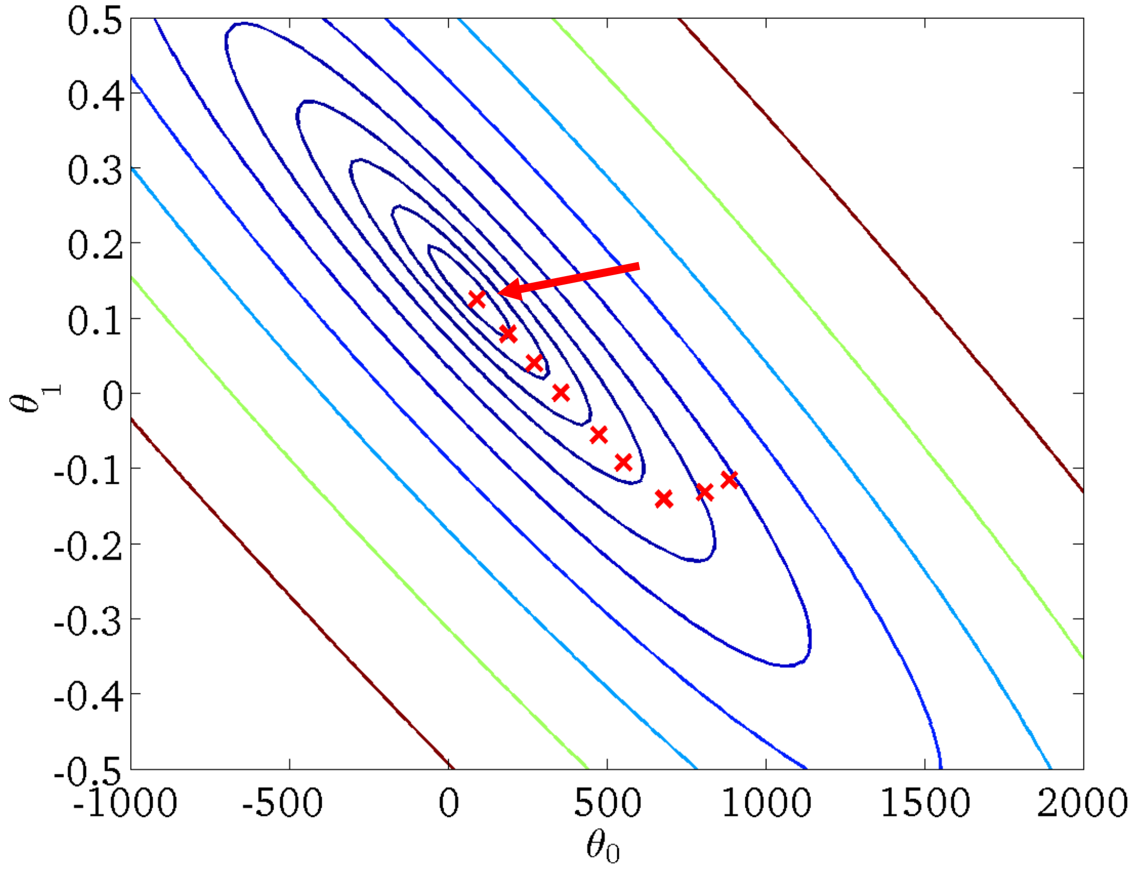
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



Gradient Descent variants

- There are three variants of gradient descent based on the amount of data used to calculate the gradient:

1. Batch gradient descent
2. Stochastic gradient descent
3. Mini-batch gradient descent

Batch Gradient Descent

- Batch Gradient Descent, aka **Vanilla gradient descent**, calculates the error for each observation in the dataset but performs an update only after all observations have been evaluated.

- One cycle through the entire training dataset is called a training epoch. Therefore, it is often said that batch gradient descent performs model updates at the end of each training epoch.

- Batch gradient descent is not often used, because it represents a huge consumption of computational resources, as the entire dataset needs to remain in memory.

Stochastic Gradient Descent (SGD)

- Stochastic gradient descent, often abbreviated **SGD**, is a variation of the gradient descent algorithm that calculates the error and updates the model for each example in the training dataset.

- SGD is usually faster than batch gradient descent, but its frequent updates cause a higher variance in the error rate, that can sometimes jump around instead of decreasing.

- The noisy update process can allow the model to **avoid local minima** (e.g. premature convergence).

Mini-Batch Gradient Descent

- Mini-batch gradient descent seeks to find a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent.

- It is the *most common implementation* of gradient descent used in the field of deep learning.

- It splits the training dataset into **small batches** that are used to calculate model error and update model coefficients.

- **“Batch size” commonly used as power of 2: 32, 64, 128, 256, and so on.**

Thanks