Applied Machine Learning

Lecture 2
One variable (simple) linear regression

Ekarat Rattagan, Ph.D.

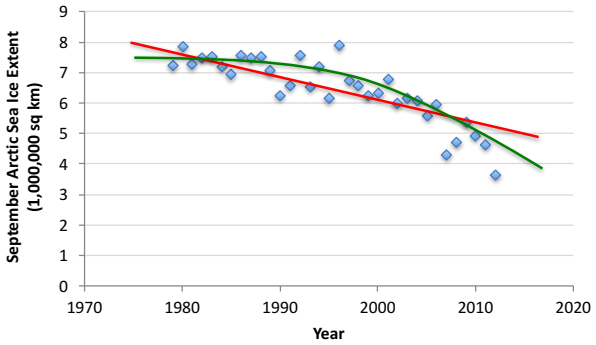
Outline

2.1 Linear regression
2.2 Learning algorithm
2.3 Iterative method: Gradient descent
2.4 Gradient descent for linear regression

2.1 Linear regression

Supervised Learning: Regression

- Given $X = \{(x^1, y^1), (x^2, y^2), \ldots, (x^m, y^m)\}$
- Learn a function $f(x)$ to predict y given x
  - y is real-valued == regression



Data from G. Witt. Journal of Statistics Education, Volume 21, Number 1 (2013)

26

Notation:

m = Number of training examples

x's = "input" or "independent" variables or features

y's = "output", "target", or "dependent" variable

Function approximation

Problem Setting:

- Set of possible instances $X = \{(x^1, y^1), (x^2, y^2), \ldots, (x^m, y^m)\}$
- Unknown target function $f : X \rightarrow Y$
- Set of function hypotheses $H = \{ h \mid h : X \rightarrow Y \}$

Input:

superscript: i^{th} training example

- Training examples $\{<x^{(i)}, y^{(i)}>\}$ of unknown target function f

Output:

- Hypothesis $h \in H$ that best approximates target function f

Credit: Tom M. Mitchell

2.2 Learning algorithm

Learning algorithm

- **Objective: to find the best target function** $h \in H$

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

(Model representation, L1, P40)

- **We need to specify a performance measure**
  - Cost function

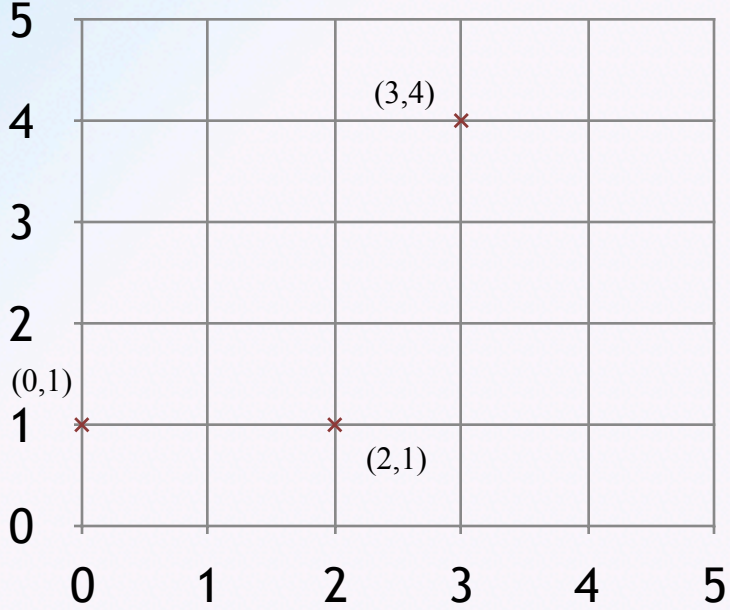
"A computer program is said to learn from experience E with respect to some task T and some **performance measure P,** if its performance on T, as measured by P, improves with experience E."

What is cost function ?

- A function that measures the **performance of a Machine Learning model** for given data.
- It quantifies the error between **estimated and true values**.
- It is used for **parameter estimation**.

Cost function (Sum of Squared Errors)

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



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

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

To find θ_0, θ_1 so that the gap between $h_\theta(x)$ and y is minimized.

Closed-form solution

- The Normal equation

$$\theta = (X^T X)^{-1} X^T y$$

Code: appendix

Normal equation

- **Limitation**
  - Non-invertible or Singular Matrix (det = 0)
  - Some features are redundant

- **Solution**
  - The Moore-Penrose Pseudo-inverse
    - Singular Value Decomposition (SVD)

https://hadrienj.github.io/posts/Deep-Learning-Book-Series-2.9-The-Moore-Penrose-Pseudoinverse/

Normal equation

- **Limitation**
  - Computational complexity $O(n^3)$
    - Slow when the number of features grows large (e.g., 100,000), to inverse matrix

- **Solution**
  - Another approach: iterative method

2.3 Iterative method: (Batch) Gradient descent

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

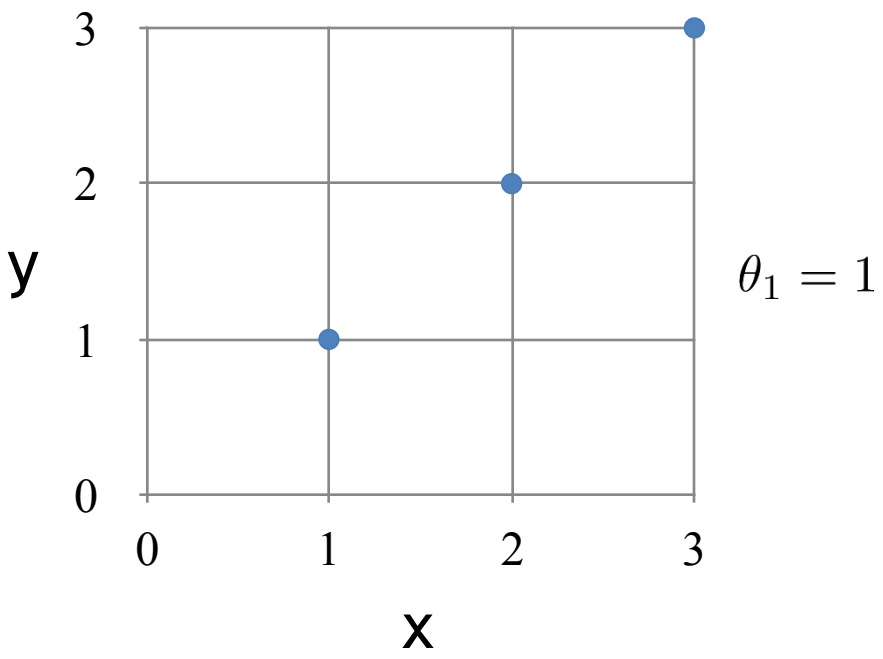
Parameters: θ_0, θ_1

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

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

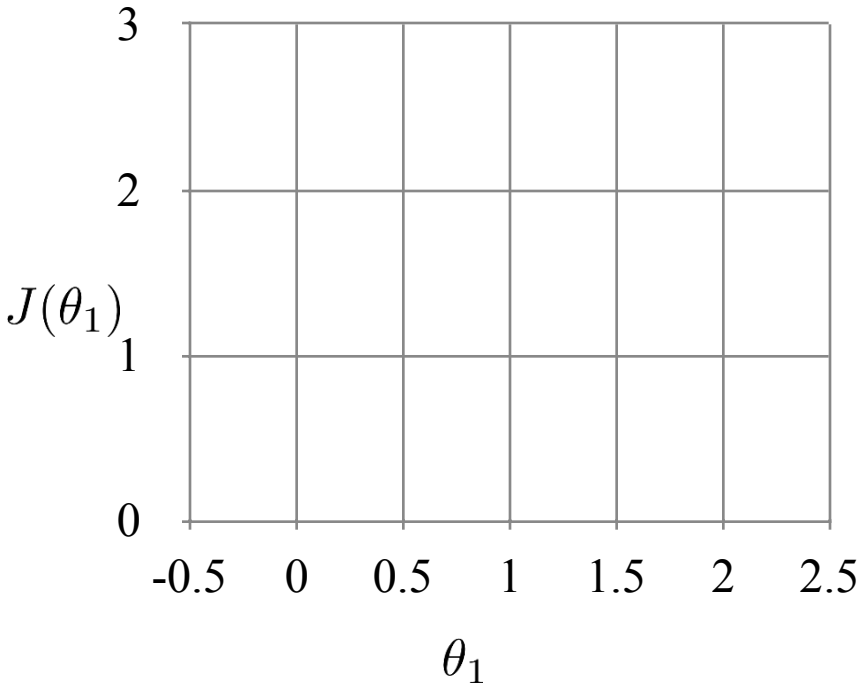
$h_\theta(x)$

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



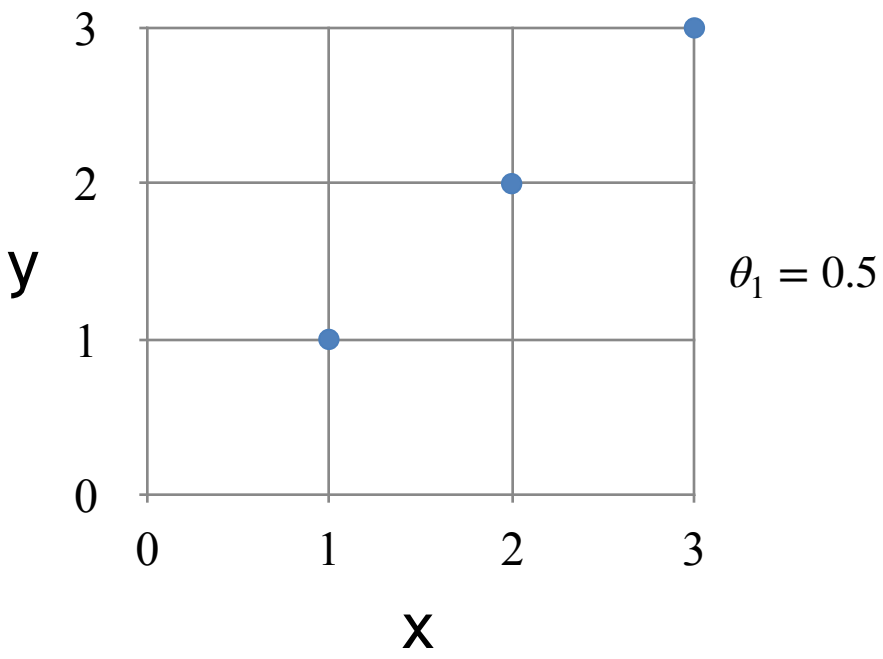
$J(\theta_1)$

(function of the parameter θ_1)



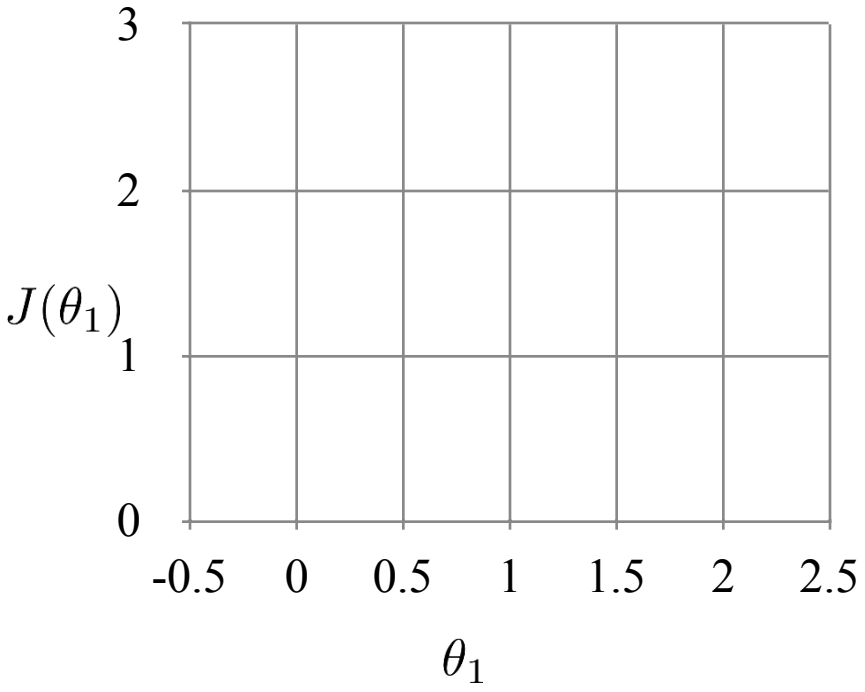
$h_\theta(x)$

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



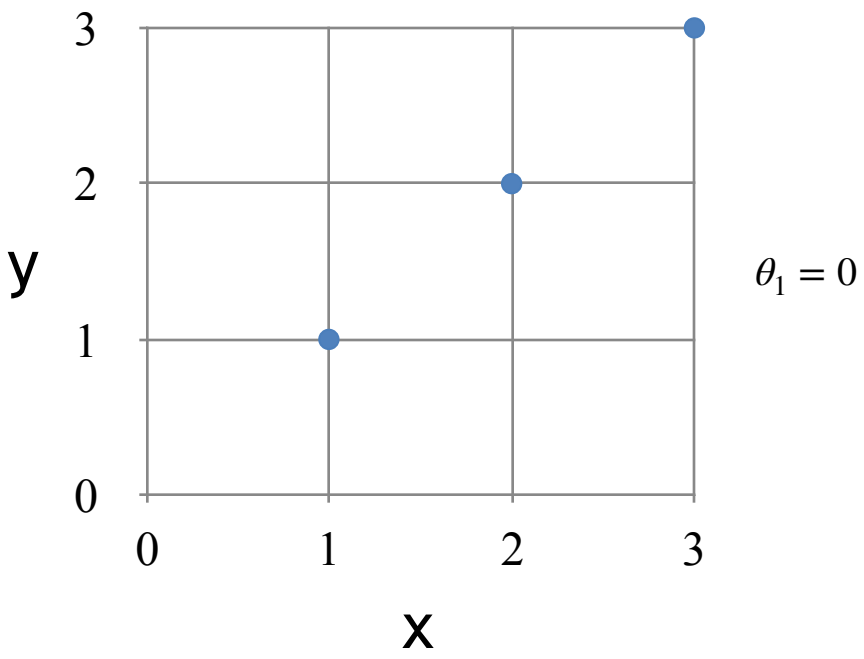
$J(\theta_1)$

(function of the parameter θ_1)



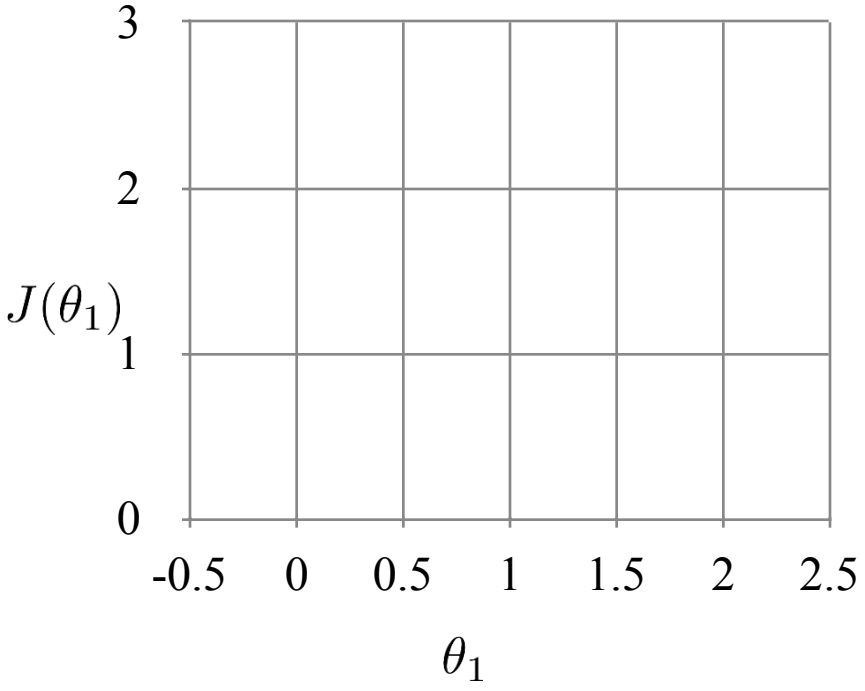
$h_\theta(x)$

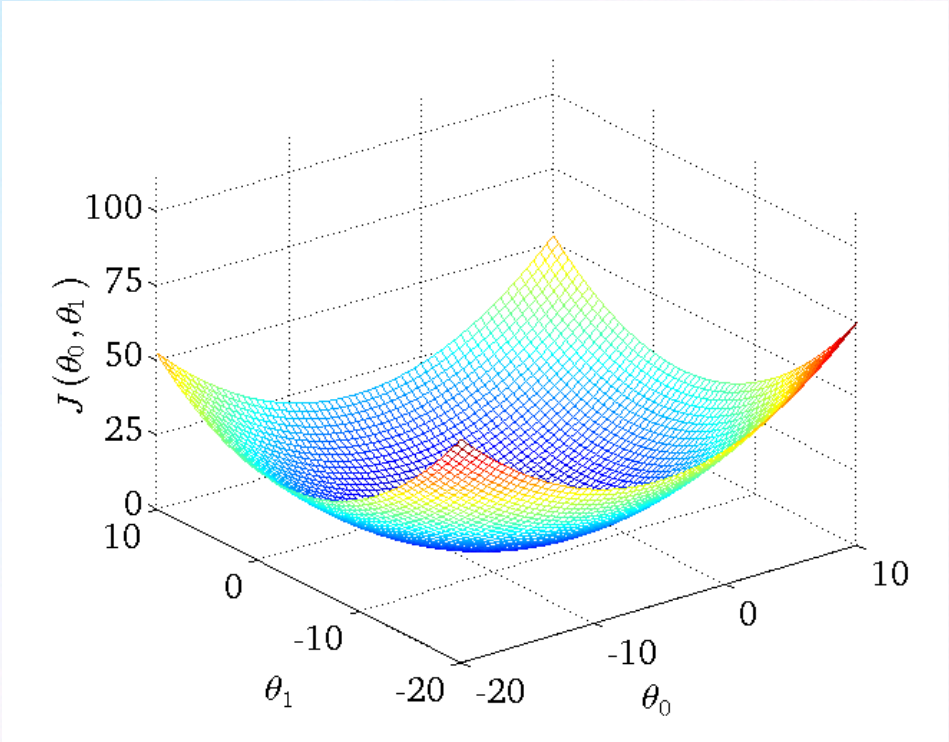
(for fixed θ_1 , this is a function of x)



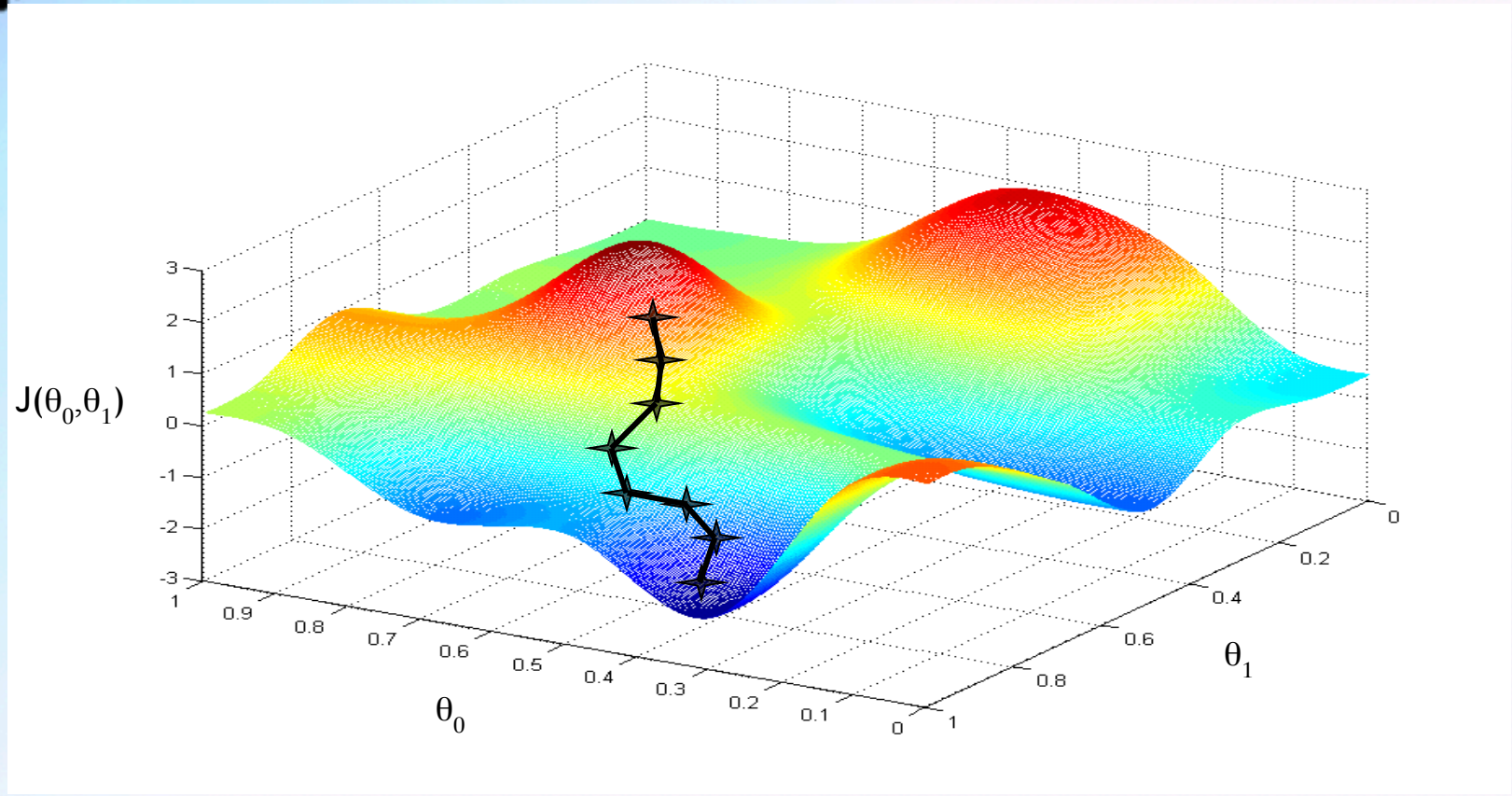
$J(\theta_1)$

(function of the parameter θ_1)





Credit: Andrew NG



Credit: Andrew NG

Gradient descent

Gradient descent is a way to minimize an objective function $J(\theta)$ parameterized by a model's parameters $\theta \in \mathbb{R}^d$ by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_\theta J(\theta)$ w.r.t. to the parameters. The learning rate α determines the size of the steps we take to reach a (local) minimum. In other words, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley.[5]

reference: https://arxiv.org/pdf/1609.04747.pdf

http://sites.science.oregonstate.edu/math/home/programs/undergrad/CalculusQuestStudyGuides/vcalc/grad/grad.html

Gradient descent

repeat until convergence {

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

}

Simultaneous update

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

Declare convergence

Declare convergence if $J(\theta)$ decreases by less than, e.g., 10^{-3} in one iteration.

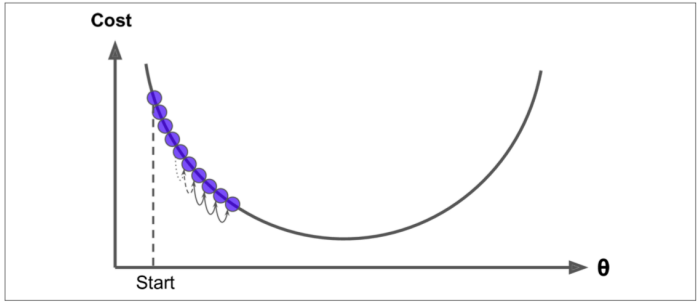


Figure 4-4. Learning rate too small

If α is too small, gradient descent can be slow.

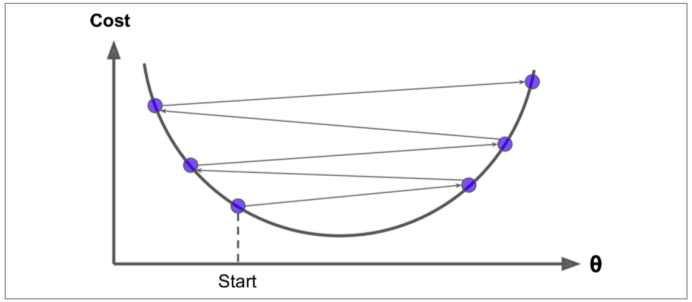


Figure 4-5. Learning rate too large

If α is too large, gradient descent can overreach the minimum.
It may fail to converge, or even diverge.

Figures from “Hands-on Machine Learning with Scikit-Learn, Keras, & TensorFlow, …”

2.4 Gradient descent for linear regression

Gradient descent algorithm

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

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 algorithm

repeat until convergence {

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

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

}

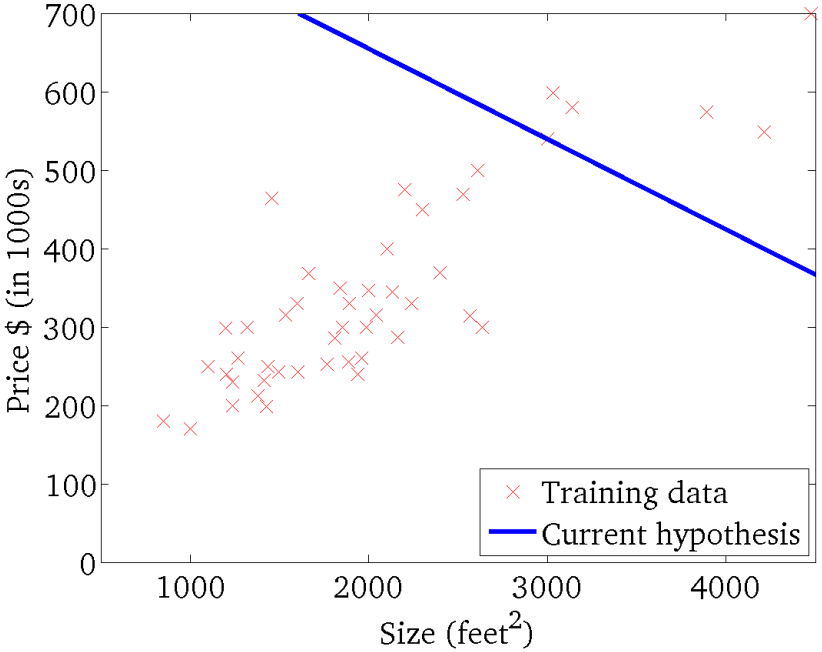
How to get these two equations?

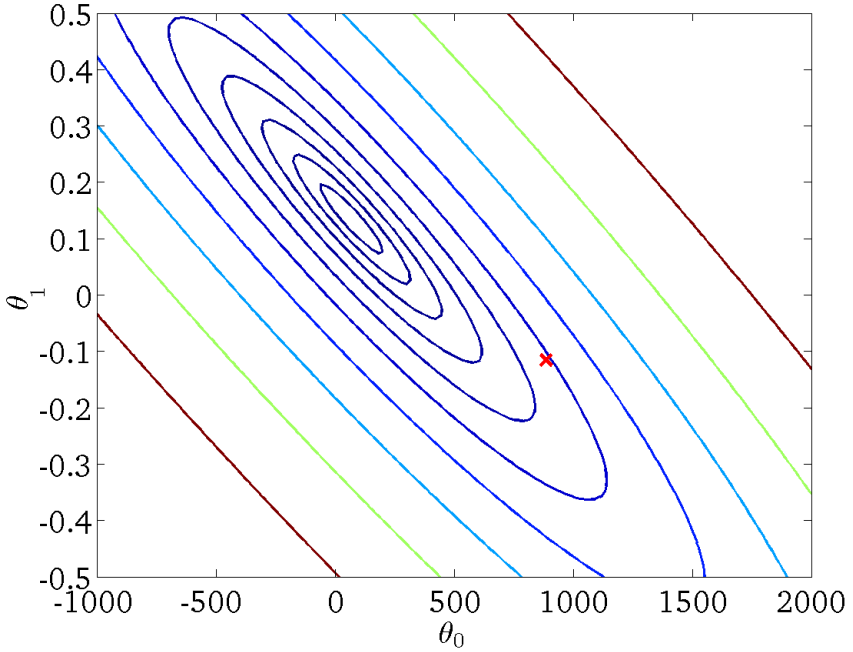
$h_\theta(x)$

(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)





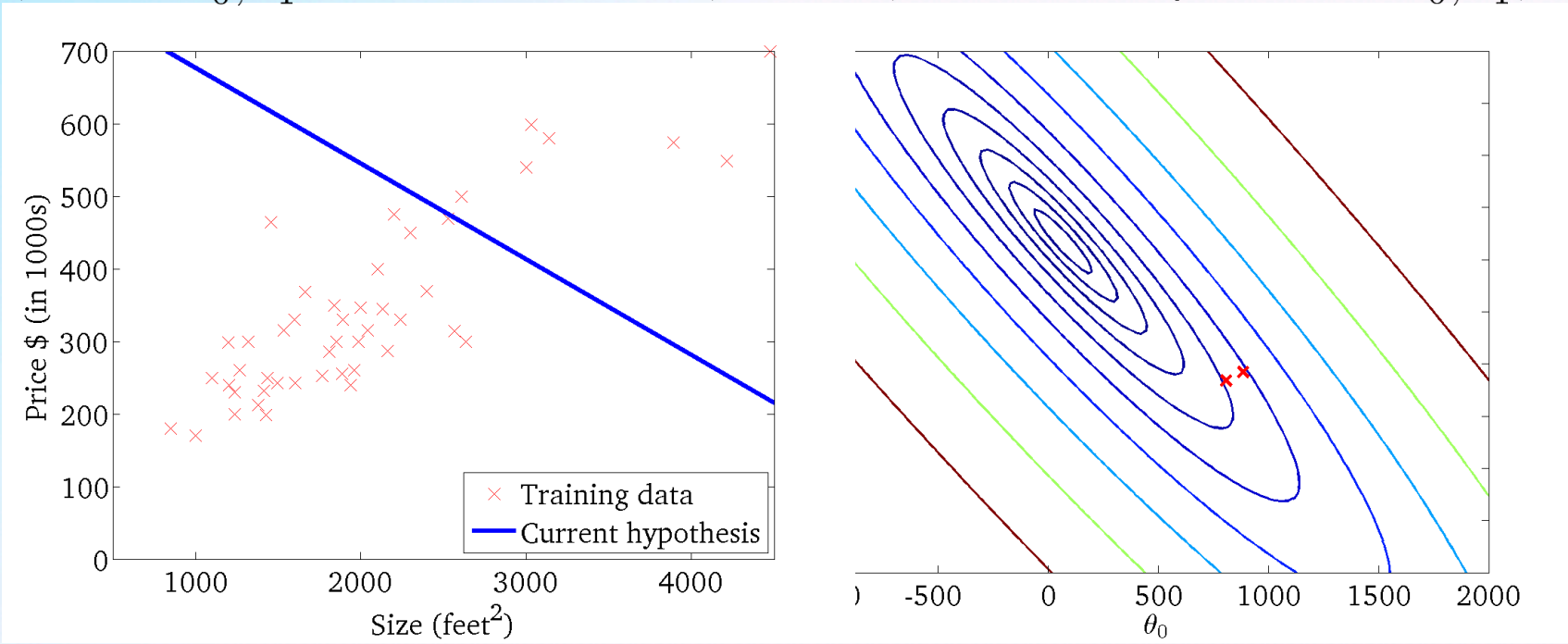
Credit: Andrew NG

$h_\theta(x)$

(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)



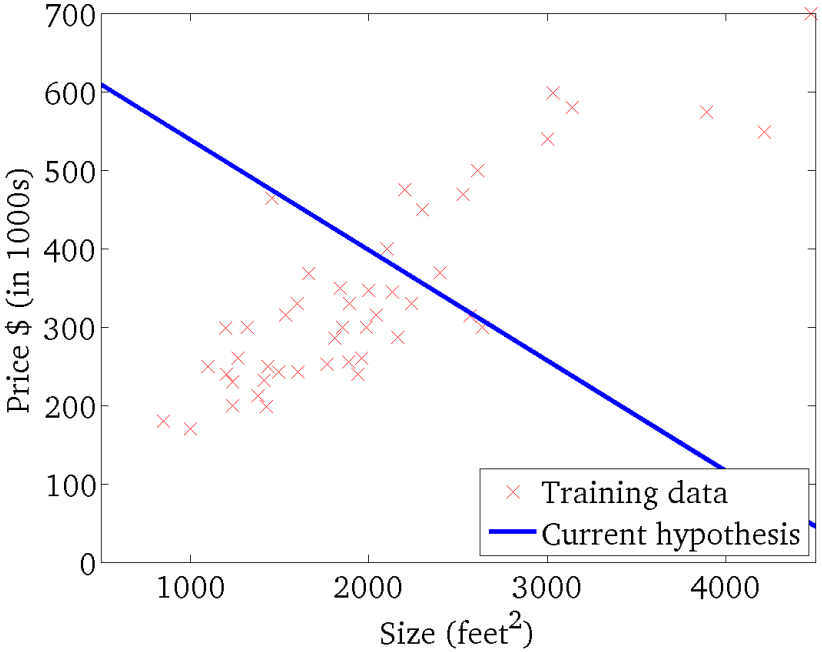
Credit: Andrew NG

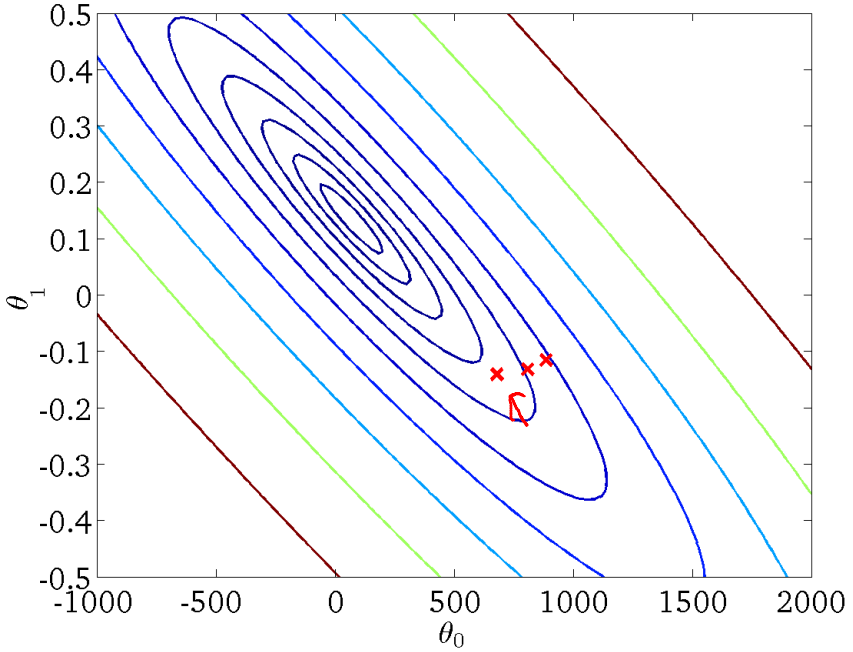
$h_\theta(x)$

(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)





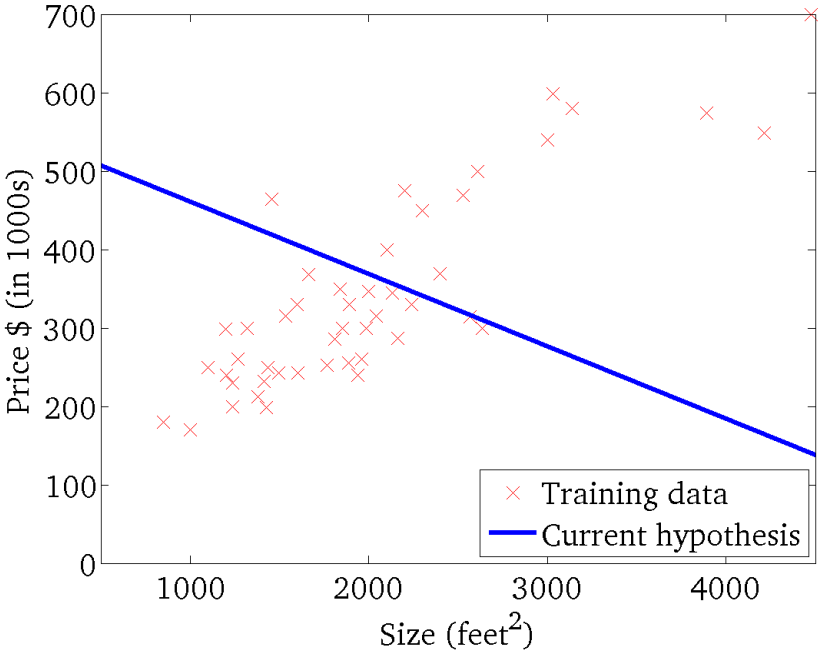
Credit: Andrew NG

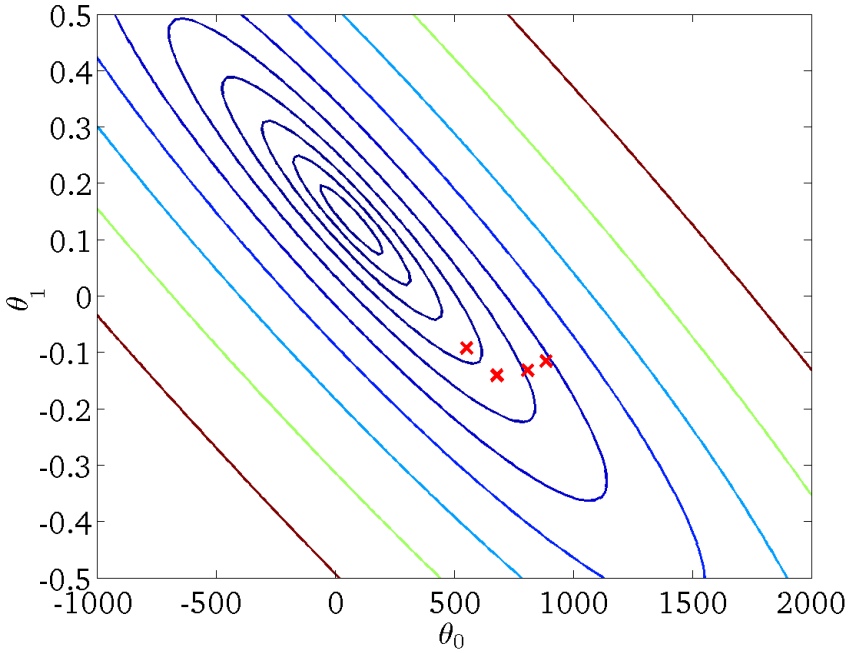
$h_\theta(x)$

(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)





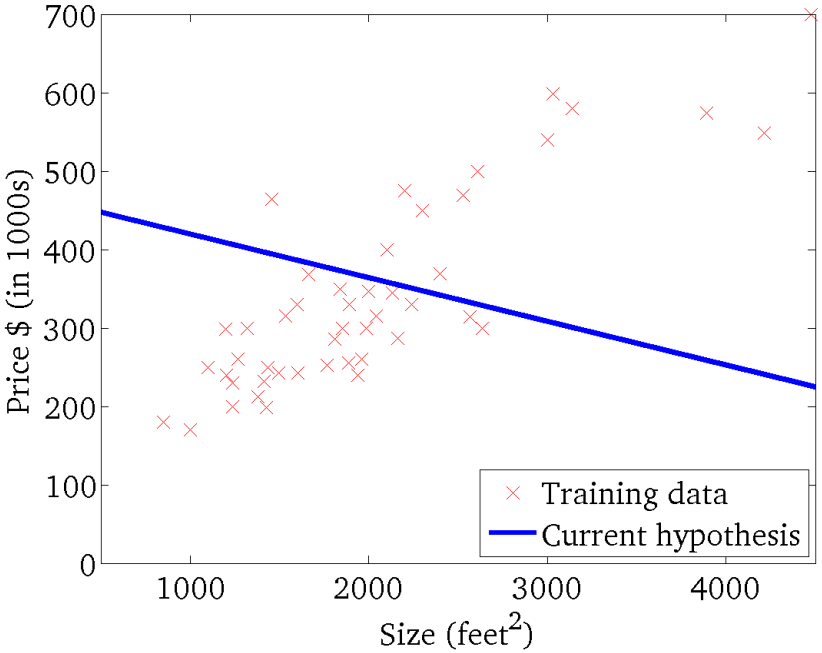
Credit: Andrew NG

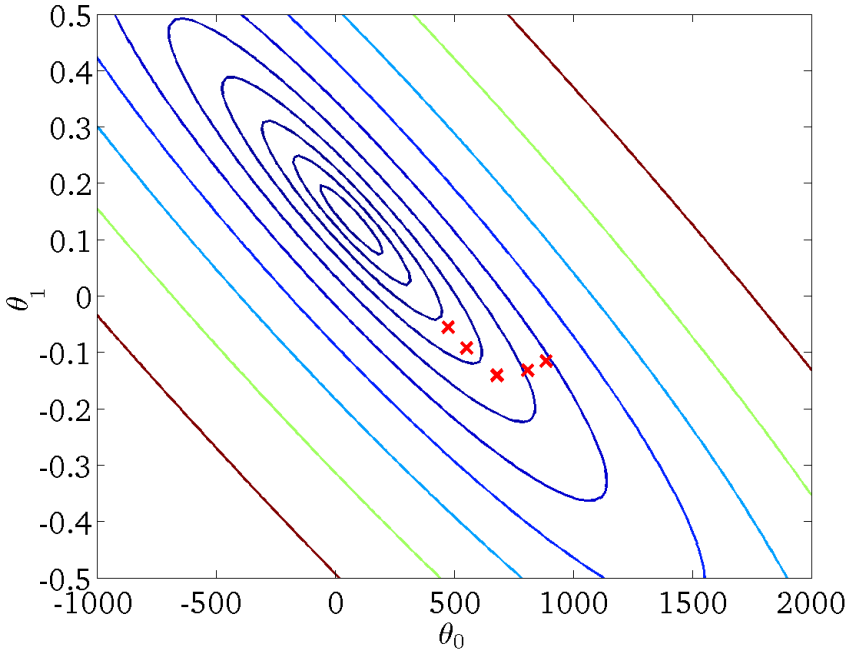
$h_\theta(x)$

(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)





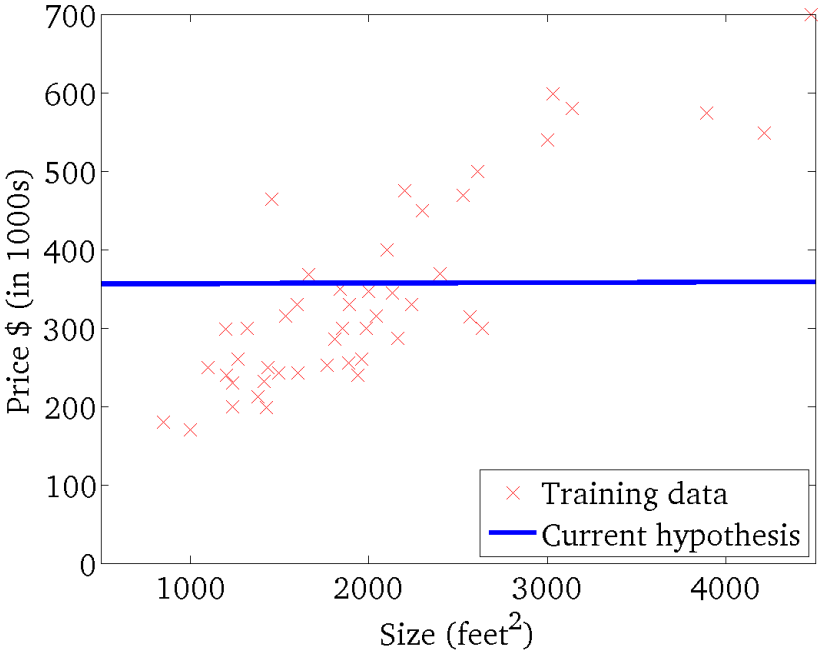
Credit: Andrew NG

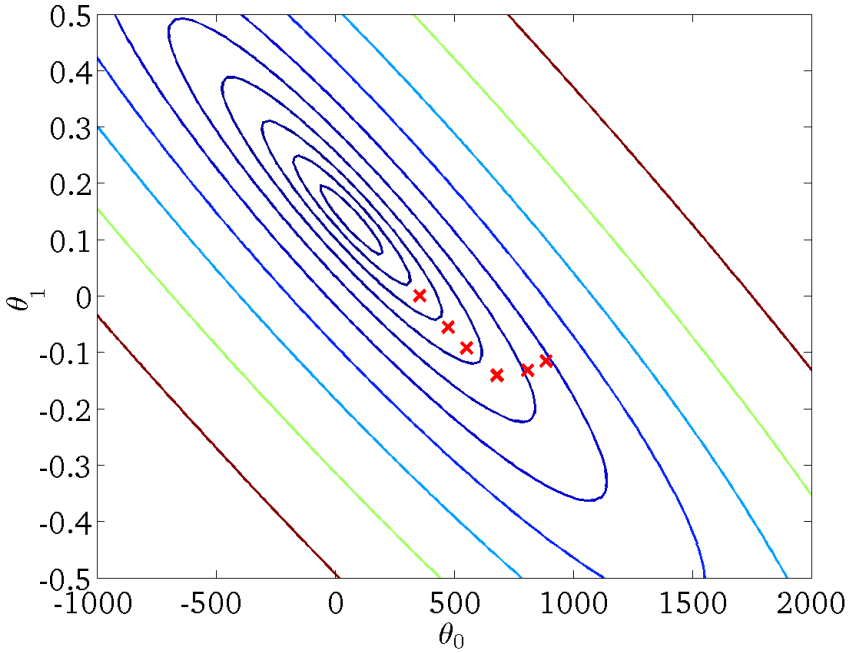
$h_\theta(x)$

(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)





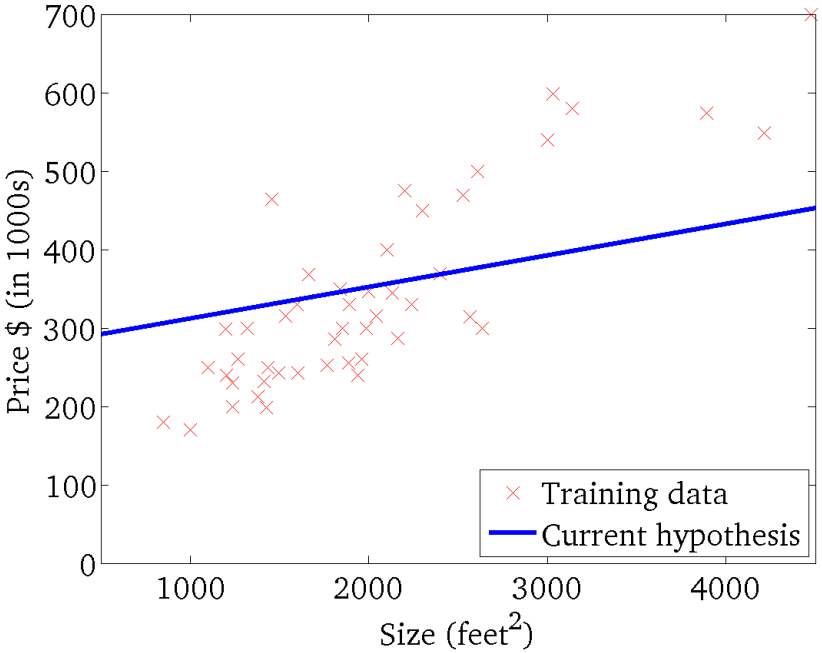
Credit: Andrew NG

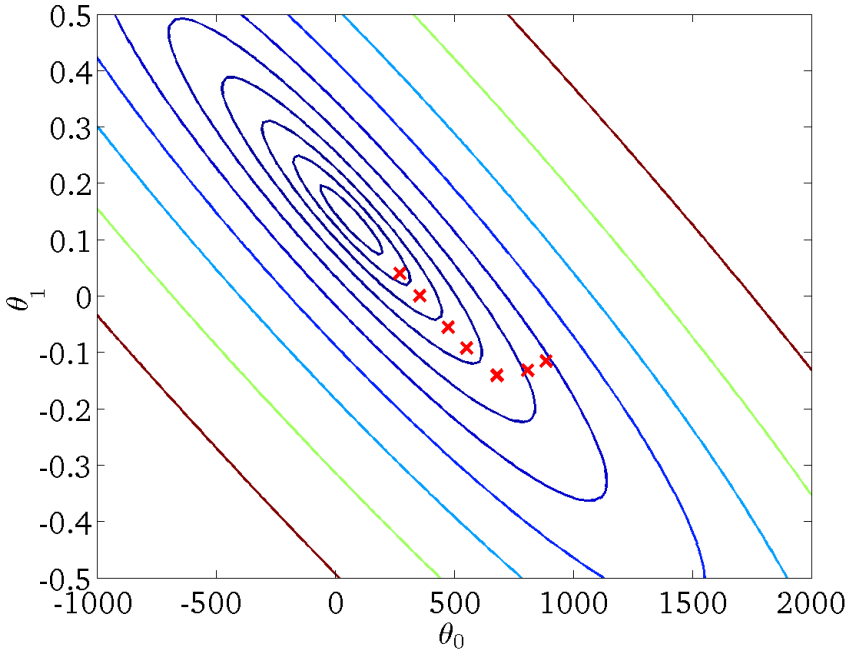
$h_\theta(x)$

(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)





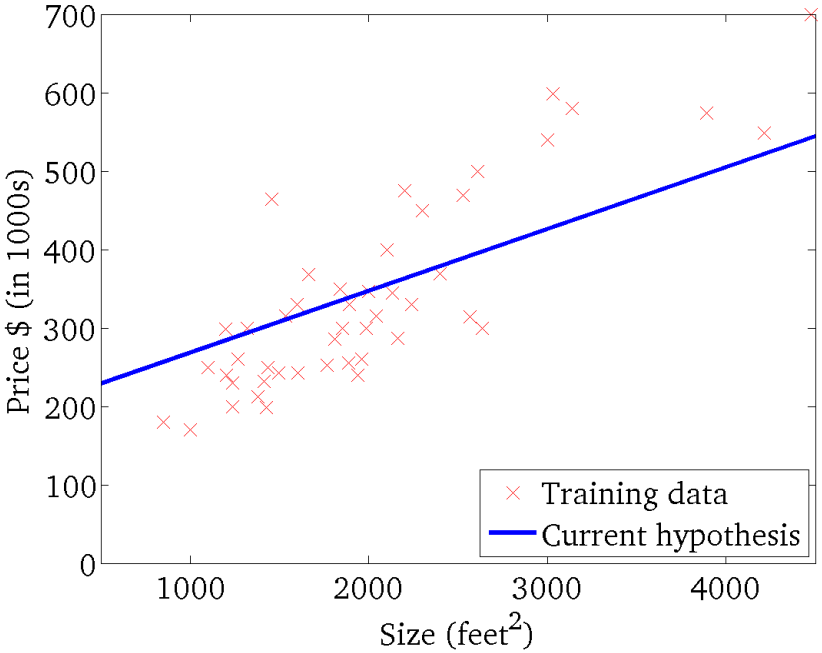
Credit: Andrew NG

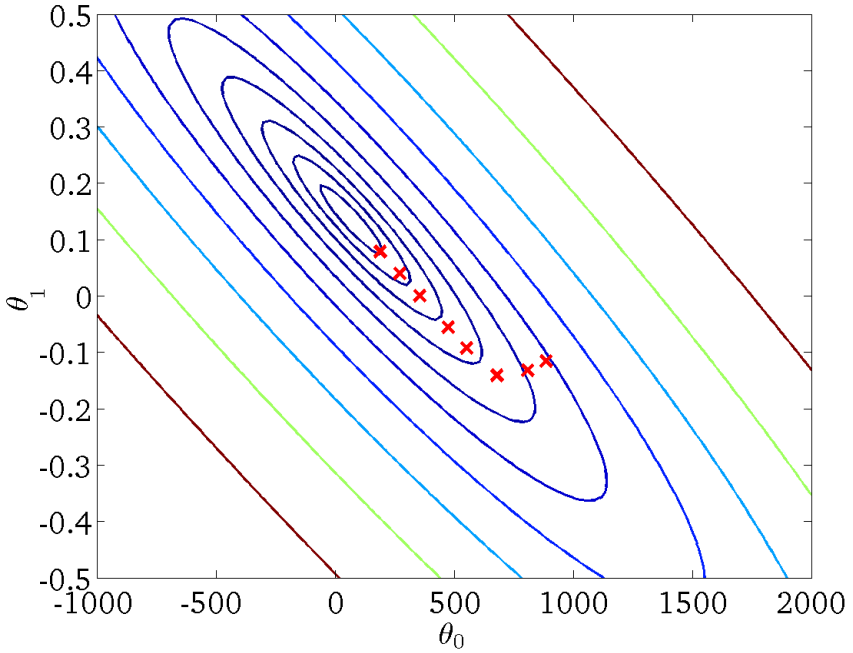
$h_\theta(x)$

(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)





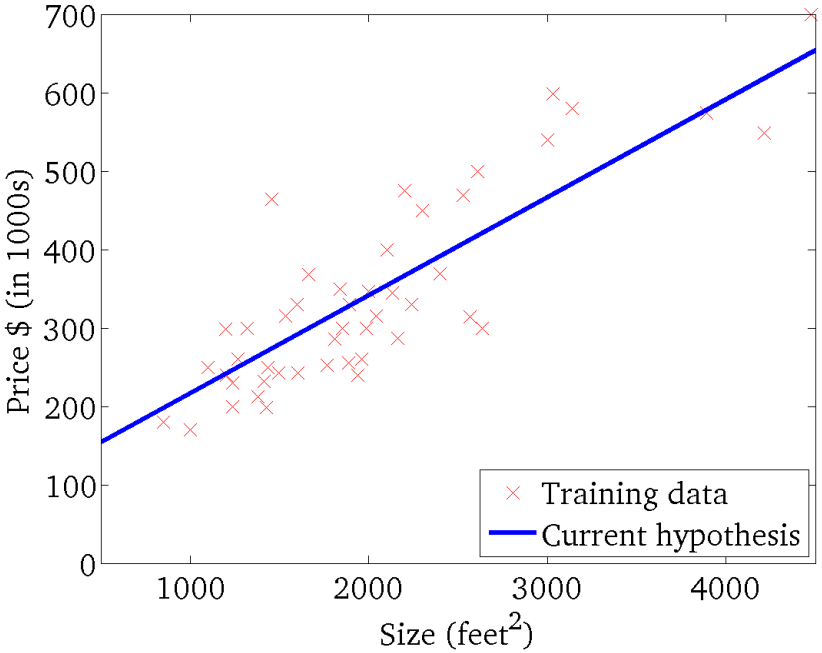
Credit: Andrew NG

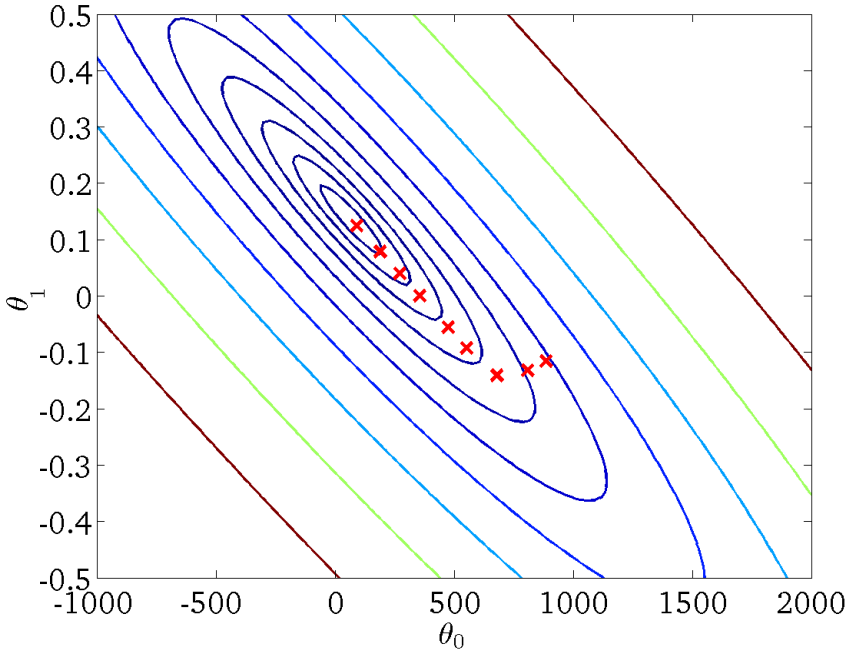
$h_\theta(x)$

(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)





Credit: Andrew NG

Gradient Descent

- **Limitation**
  - Need to choose the learning rate
  - Can be very slow as large datasets may not fit in the memory

Gradient Descent VS Normal Equation

S.NO.	GRADIENT DESCENT	NORMAL EQUATION
1.	In gradient descent we need to choose learning rate.	In normal equation , no need to choose learning rate.
2.	It is an iterative algorithm.	It is analytical approach.
3.	Gradient descent works well with large number of features.	Normal equation works well with small number of features.
4.	Feature scaling can be used.	No need for feature scaling.
5.	No need to handle non-invertibility case.	If (X^TX) is non-invertible , use pseudo-inverse
6.	Algorithm complexity is O(n^2). n is the number of features.	Algorithm complexity is O(n^3). n is the number of features.

https://www.geeksforgeeks.org/difference-between-gradient-descent-and-normal-equation/