Lecture 3
Introduction to optimization: Gradient descent

GEOL 4397: Data analytics and machine learning for geoscientists

Jiajia Sun, Ph.D.

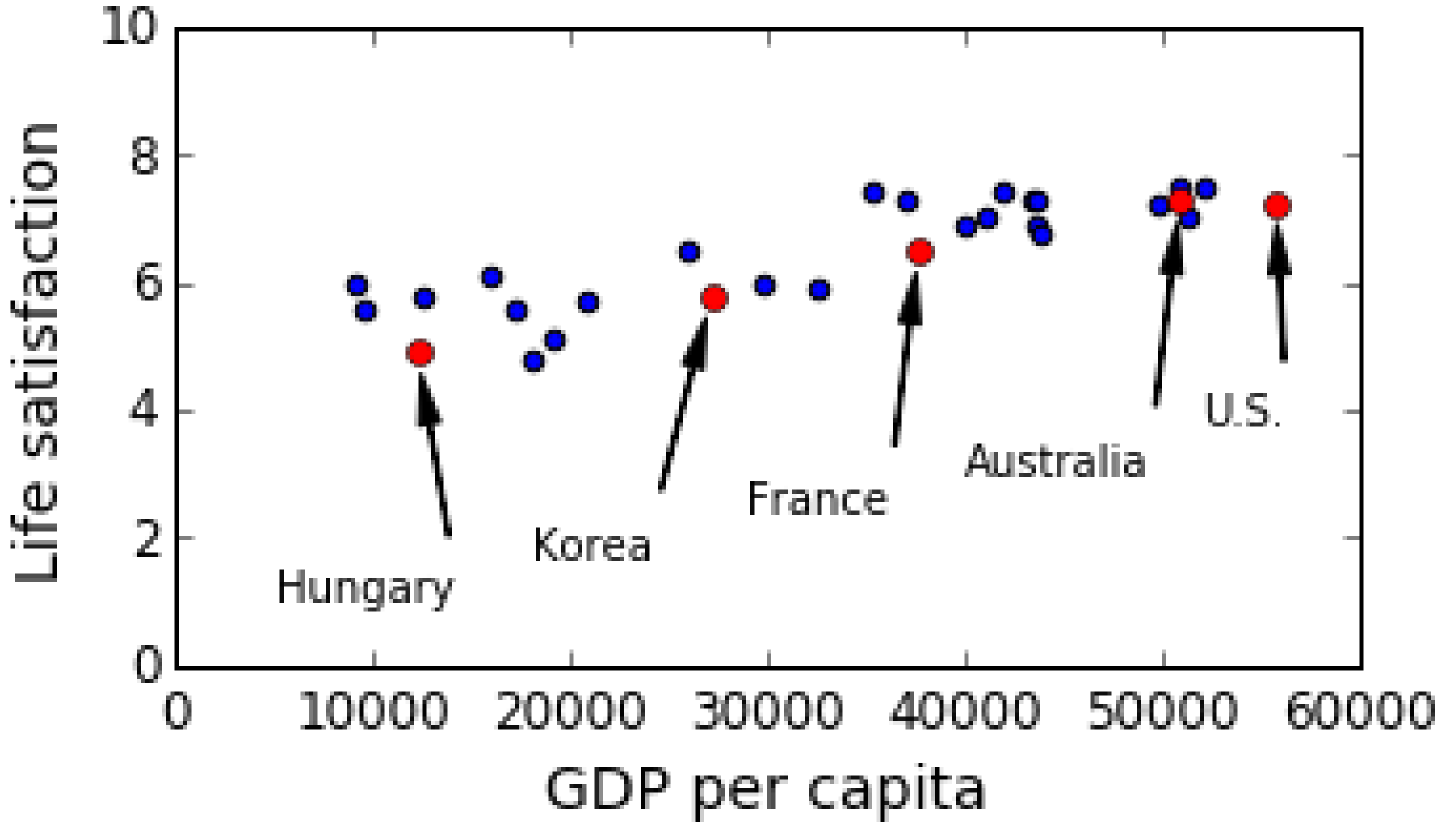
Jan. 24th, 2019

UNIVERSITY of HOUSTON
YOU ARE THE PRIDE
EARTH AND ATMOSPHERIC SCIENCES

Today’s agenda

- Motivation
- Concept: gradient
- Gradient descent
  - Stochastic gradient descent
  - Mini-batch gradient descent
- Learning rate

Motivation



Each point represents one training example/instance.

Figure from Aurelien Geron's ML book, page 19

General approach to learning/training

defining a cost function

&

minimizing it

Minimization

- Cost function measures how bad a candidate model is

$$\min \; J(\theta_0, \theta_1) = \sum_{i=1}^{M} (\theta_0 \; + \; \theta_1 x^{(i)} \; - y^{(i)})^{\,2}$$

- Learning/training = Minimization = Optimization

- Optimization: finding optimal parameter values that minimize a cost function

Matrix-vector form

$$J(\theta) = \sum_{i=1}^{M} (\boldsymbol{\theta}^T \boldsymbol{x}^{(i)} - y^{(i)})^{\,2}$$

$$J(\theta) = ||\boldsymbol{X\theta} - \boldsymbol{y}||^2$$

$$\boldsymbol{X} = \begin{bmatrix} (\boldsymbol{x}^{(1)})^T \\ \ldots \\ (\boldsymbol{x}^{(M)})^T \end{bmatrix} \qquad \boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ \ldots \\ y^{(M)} \end{bmatrix}$$

Mx(N+1) Mx1

Analytical solution

- Minimize:

$$J(\theta) = \|\boldsymbol{X\theta} - \boldsymbol{y}\|^2$$

$$\widetilde{\boldsymbol{\theta}} = \left(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}\right)^{-\mathbf{1}}(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y})$$

Analytical solution

- Minimize:

$$J(\theta) = \|\boldsymbol{X\theta} - \boldsymbol{y}\|^2$$

- Normal equation method

$$\widetilde{\boldsymbol{\theta}} = \left(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}\right)^{-\mathbf{1}}(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y})$$

Analytical solution

- Minimize:

$$\color{red}{J(\theta)} = \color{blue}{\|\boldsymbol{X\theta} - \boldsymbol{y}\|^2}$$

- Normal equation method

$$\color{red}{\widetilde{\boldsymbol{\theta}}} = \color{blue}{\left(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X}\right)^{-\mathbf{1}}(\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{y})}$$

```
theta = np.matmul(np.linalg.inv(np.matmul(X.T,X)), np.matmul(X.T,y))
```

To derive normal equation (optional)

- http://www.programming-techniques.com/2013/12/gradient-descent-versus-normal-equation.html

- http://cs229.stanford.edu/notes/cs229-notes1.pdf

  page 8-11

Problem with normal equation method

$$\tilde{\boldsymbol{\theta}} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}(\boldsymbol{X}^T\boldsymbol{y})$$

Problem with normal equation method

$$\tilde{\boldsymbol{\theta}} = \left(\boxed{\boldsymbol{X}^T\boldsymbol{X}}\right)^{-1}\left(\boldsymbol{X}^T\boldsymbol{y}\right)$$

(N+1)X(N+1)

Computational cost

- increases linearly with M (# of instances)

Problem with normal equation method

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

(N+1)X(N+1)

Computational cost

- increases linearly with M (# of instances)
- Increases cubically* with N (# of features)

*Strictly speaking, computational complexity is $O(N^{2.4})$ to $O(N^3)$. If we double the number of features, the computation time increase by $2^{2.4} = 5.3$ to $2^3 = 8$ times

Problem with normal equation method

$$\tilde{\boldsymbol{\theta}} = \left(\boxed{\boldsymbol{X}^T\boldsymbol{X}}\right)^{-1}\left(\boldsymbol{X}^T\boldsymbol{y}\right)$$

(N+1)X(N+1)

Computational cost

- increases linearly with M (# of instances)
- Increases cubically* with N (# of features)
- For big data, computationally very expensive

*Strictly speaking, computational complexity is $O(N^{2.4})$ to $O(N^3)$. If we double the number of features, the computation time increase by $2^{2.4} = 5.3$ to $2^3 = 8$ times

Problem with normal equation method

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

(N+1)X(N+1)

Computational cost

- increases linearly with M (# of instances)
- Increases cubically* with N (# of features)
- For big data, computationally very expensive

Think about the life satisfaction problem, we only used one feature, i.e., GDP per capita. What other features could we use?

*Strictly speaking, computational complexity is $O(N^{2.4})$ to $O(N^3)$. If we double the number of features, the computation time increase by $2^{2.4} = 5.3$ to $2^3 = 8$ times

Problem with normal equation method

$$\tilde{\boldsymbol{\theta}} = \left(\boxed{\boldsymbol{X}^T\boldsymbol{X}}\right)^{-1}(\boldsymbol{X}^T\boldsymbol{y})$$

(N+1)X(N+1)

Computational cost

- increases linearly with M (# of instances)
- Increases cubically* with N (# of features)
- For big data, computationally very expensive

For nonlinear optimization, normal equation does not even exist.

*Strictly speaking, computational complexity is $O(N^{2.4})$ to $O(N^3)$. If we double the number of features, the computation time increase by $2^{2.4} = 5.3$ to $2^3 = 8$ times

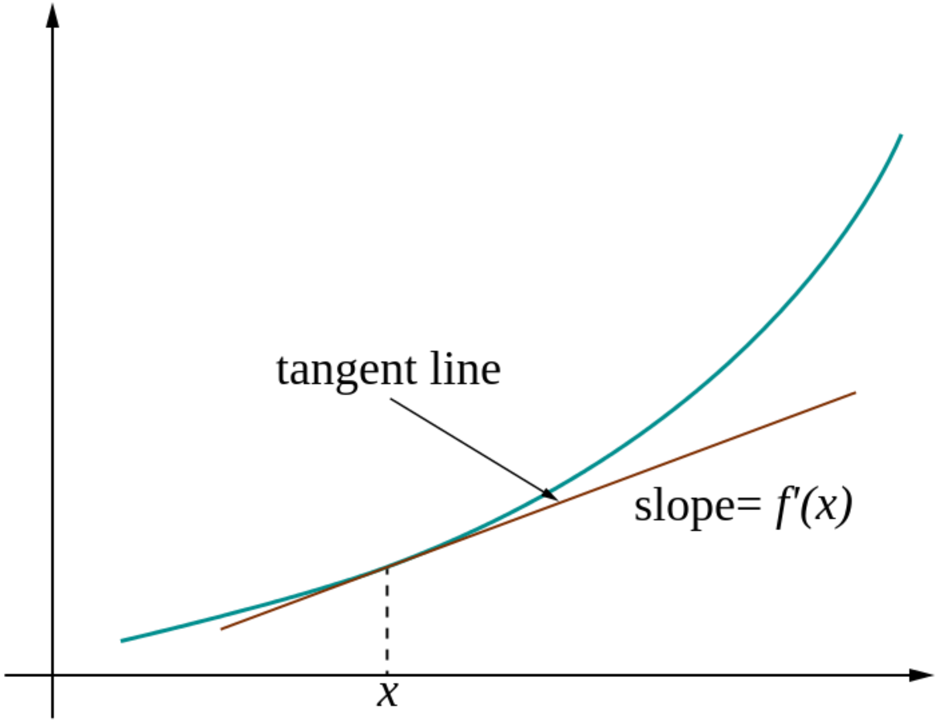
Gradient descent

- Computationally less demanding
- Generally applicable to both linear and nonlinear optimization*

*so long as gradient can be calculated.

What is gradient?

- Let us first recall what is derivative.



Picture taken from https://en.wikipedia.org/wiki/Derivative

From derivative to gradient

- Let us consider a function $f(x, y)$
- Two partial derivatives

$$\frac{\partial f}{\partial x}$$

$$\frac{\partial f}{\partial y}$$

Gradient

- Gradient of a function $f(x, y)$ is defined as

$$\nabla f(x, y) = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \\ \dfrac{\partial f}{\partial y} \end{bmatrix}$$

More on gradient

- It is a vector
- Therefore, it has direction and magnitude
- Its direction points in the direction of the greatest rate of increase (i.e., <u>direction of maximum increase</u>) of the function
- Its magnitude is the slope of the graph of the function (i.e., the rate of increase) in that direction

Put gradient in context

- Imagine you are standing on a hillside. Look all around you, and find the direction of steepest ascent.

Put gradient in context

- Imagine you are standing on a hillside. Look all around you, and find the direction of steepest ascent.
- That is the direction of the gradient.

Put gradient in context

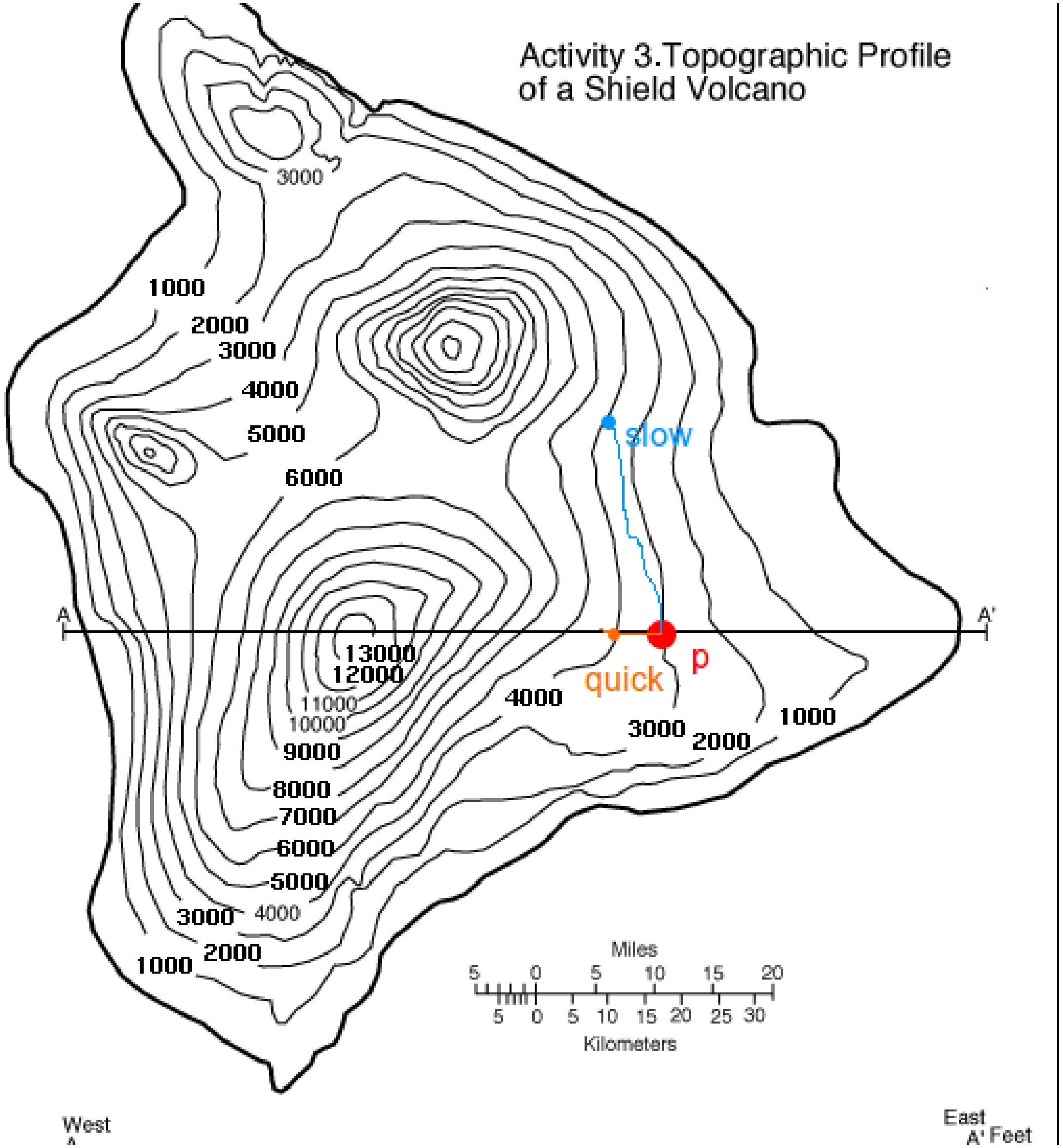
- Imagine you are standing on a hillside. Look all around you, and find the direction of steepest ascent.
- That is the direction of the gradient.
- Now measure the slope in that direction (rise over run)

Put gradient in context

- Imagine you are standing on a hillside. Look all around you, and find the direction of steepest ascent.
- That is the direction of the gradient.
- Now measure the slope in that direction (rise over run)
- That is the magnitude of the gradient.
- Here, the function is the height of hill (as a function of positions).

Understanding gradient

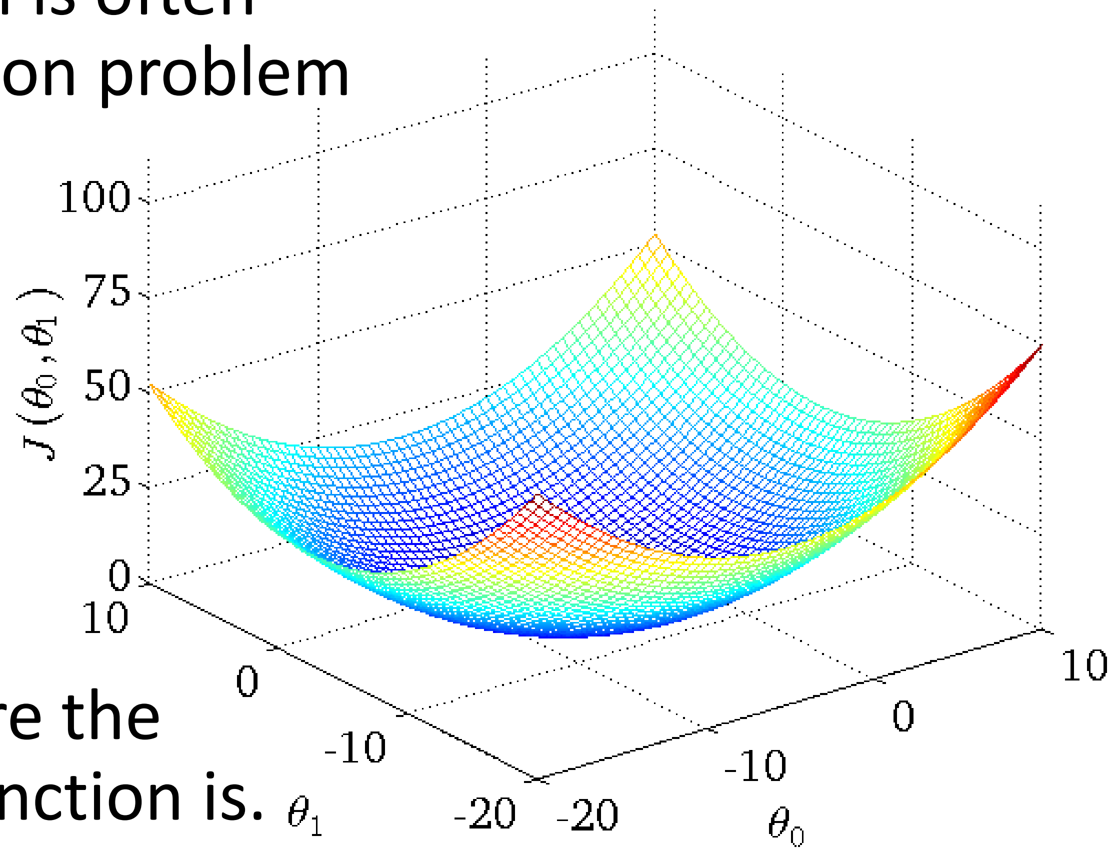
- Consider the topography as a 2D function $f(x, y)$
- The gradient direction tells you the fastest way up



Picture taken from https://mathoverflow.net/questions/1977/why-is-the-gradient-normal

Gradient in the context of optimization

- Optimization problem is often posed as a minimization problem



- We want to find where the minimum of a cost function is.

Picture taken from Andrew Ng's Machine Learning class on Coursera.org

Gradient descent algorithm

- Given initial values $\boldsymbol{\theta}^{(0)} = \left[\theta_0^{(0)}, \theta_1^{(0)}\right]$
- While (not convergence):

$$\boldsymbol{\theta}^{(j)} = \boldsymbol{\theta}^{(j-1)} - \alpha \nabla J(\boldsymbol{\theta}^{(j-1)})$$

Gradient descent algorithm

- Given initial values $\boldsymbol{\theta}^{(0)} = \left[\theta_0^{(0)}, \theta_1^{(0)}\right]$
- While (not convergence):

$$\theta_0^{(j)} = \theta_0^{(j-1)} - \alpha \frac{\partial}{\partial \theta_0} \mathrm{J}(\theta_0^{(j-1)}, \theta_1^{(j-1)})$$
$$\theta_1^{(j)} = \theta_1^{(j-1)} - \alpha \frac{\partial}{\partial \theta_1} \mathrm{J}(\theta_0^{(j-1)}, \theta_1^{(j-1)})$$

Gradient descent algorithm for linear regression

- Given initial values $\boldsymbol{\theta}^{(0)} = \left[\theta_0^{(0)}, \theta_1^{(0)}\right]$
- While (not convergence):

$$\theta_0^{(j)} = \theta_0^{(j-1)} - \alpha \frac{\partial}{\partial \theta_0} \mathrm{J}(\theta_0^{(j-1)}, \theta_1^{(j-1)})$$

$$\theta_1^{(j)} = \theta_1^{(j-1)} - \alpha \frac{\partial}{\partial \theta_1} \mathrm{J}(\theta_0^{(j-1)}, \theta_1^{(j-1)})$$

$$\min \quad J(\theta_0, \theta_1) = \sum_{i=1}^{M} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2$$

Gradient descent algorithm for linear regression

- Given initial values $\boldsymbol{\theta}^{(0)} = \left[\theta_0^{(0)}, \theta_1^{(0)}\right]$
- While (not convergence):

$$\theta_0^{(j)} = \theta_0^{(j-1)} - \alpha \frac{\partial}{\partial \theta_0} \mathrm{J}(\theta_0^{(j-1)}, \theta_1^{(j-1)})$$

$$\theta_1^{(j)} = \theta_1^{(j-1)} - \alpha \frac{\partial}{\partial \theta_1} \mathrm{J}(\theta_0^{(j-1)}, \theta_1^{(j-1)})$$

$$\min \quad J(\theta_0, \theta_1) = \frac{1}{2M} \sum_{i=1}^{M} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2$$

Gradient descent algorithm for linear regression

- Given initial values $\boldsymbol{\theta}^{(0)} = \left[\theta_0^{(0)}, \theta_1^{(0)}\right]$
- While (not convergence):

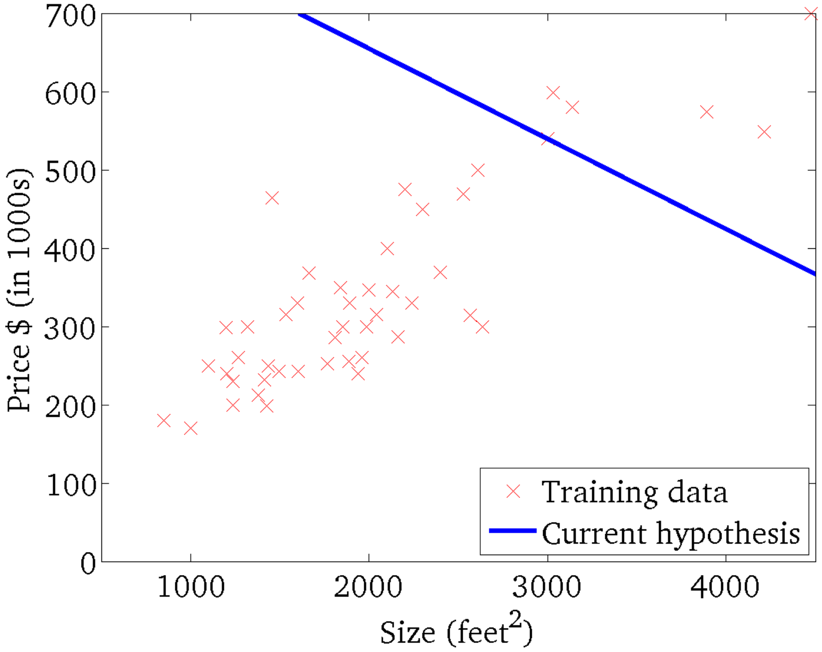
$$\theta_0^{(j)} = \theta_0^{(j-1)} - \alpha \frac{1}{M} \sum_{i=1}^{M} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})$$

$$\theta_1^{(j)} = \theta_1^{(j-1)} - \alpha \frac{1}{M} \sum_{i=1}^{M} (\theta_0 + \theta_1 x^{(i)} - y^{(i)}) x^{(i)}$$

Slides 36-44 are from Andrew Ng’s machine learning course on coursera.org

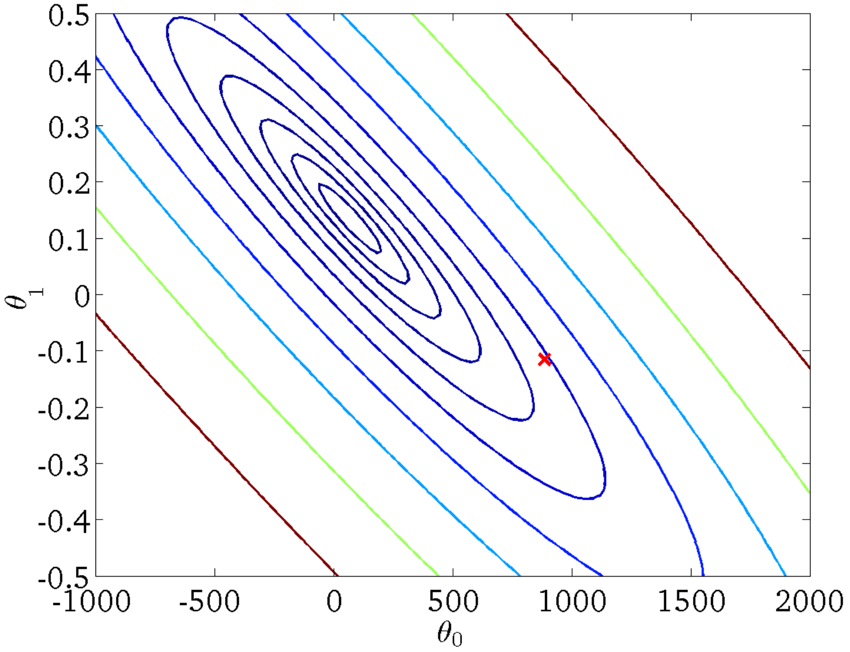
$h_\theta(x)$

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



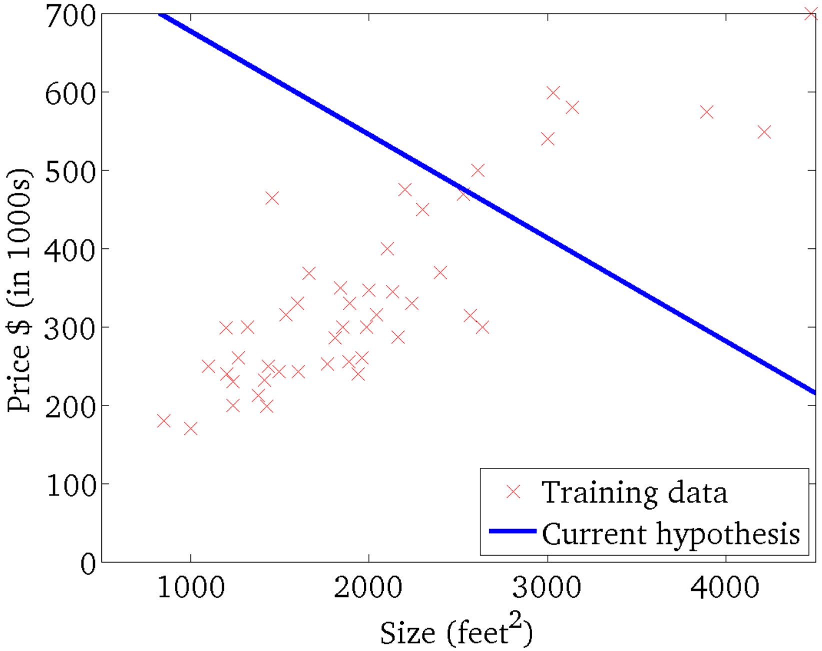
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



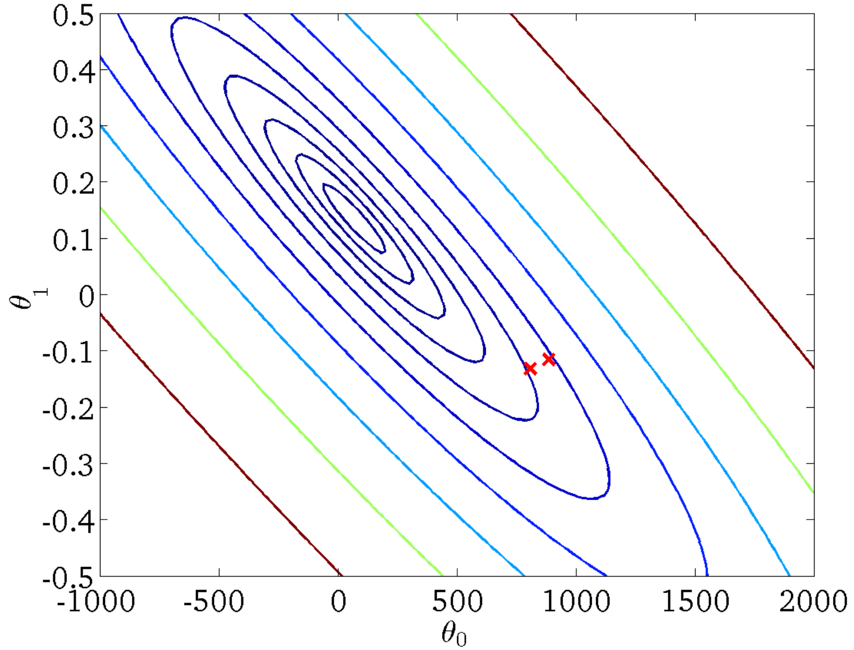
$$h_\theta(x)$$

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



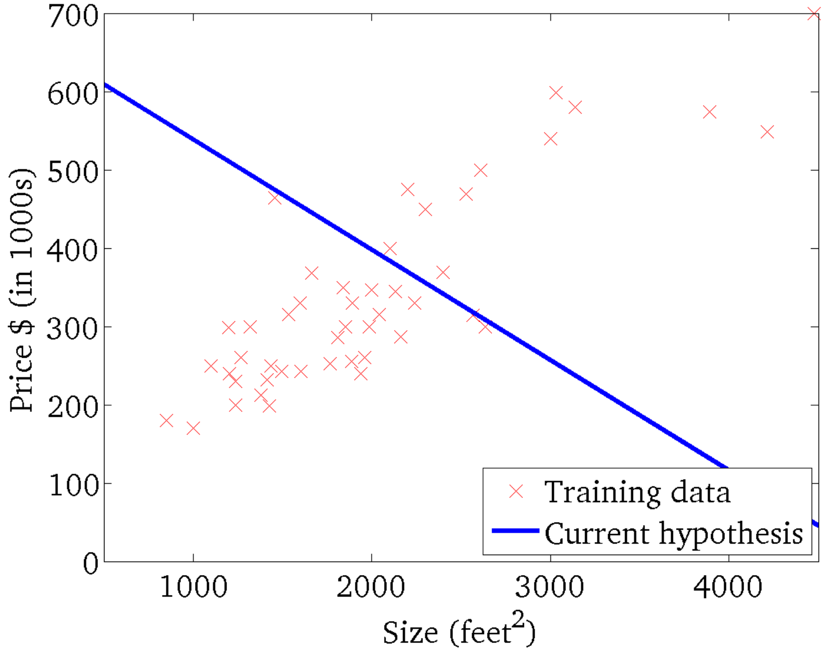
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



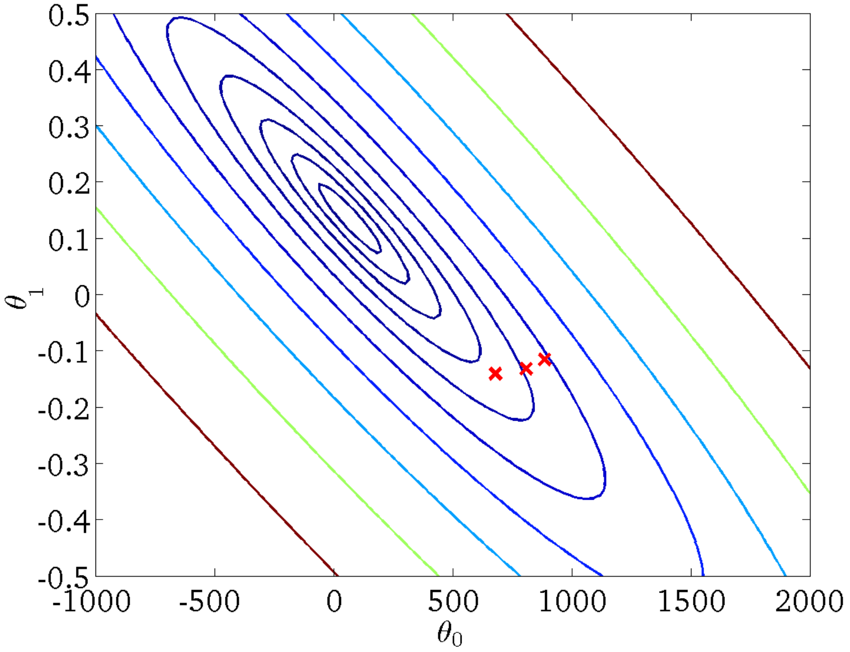
$$h_\theta(x)$$

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



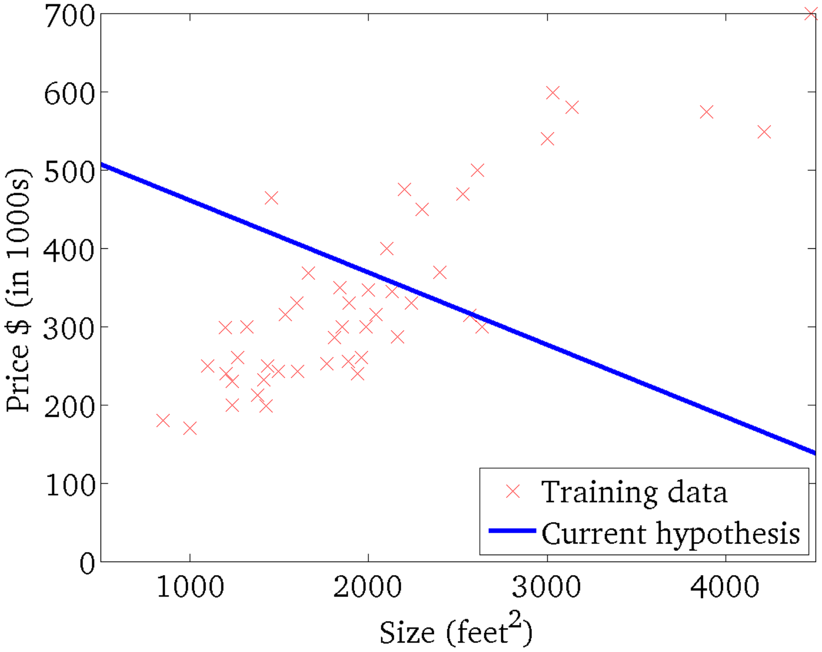
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



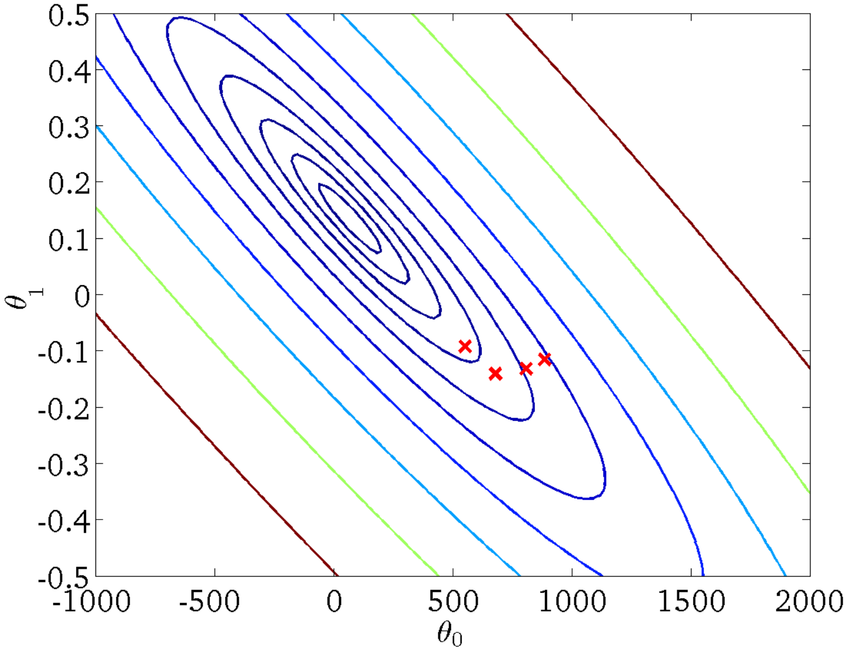
$$h_\theta(x)$$

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



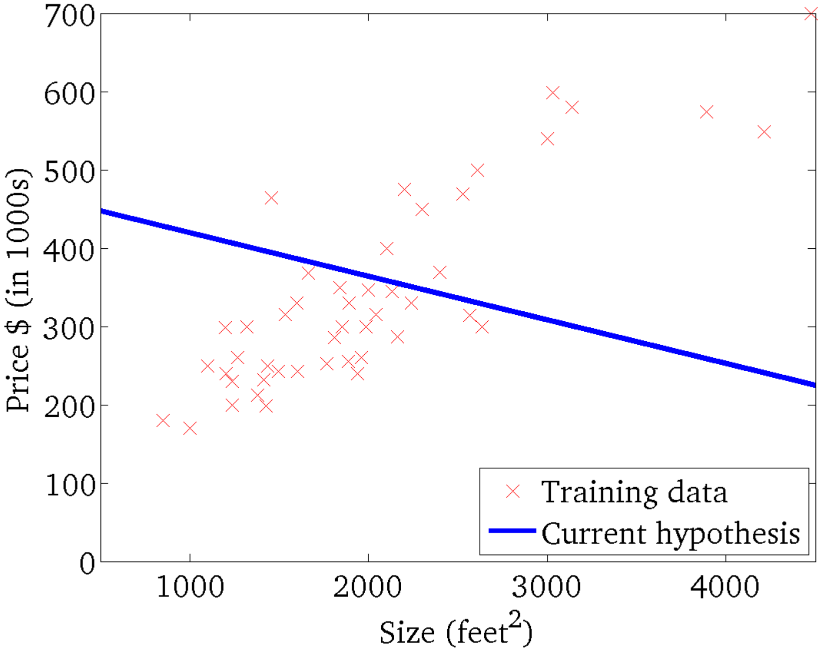
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



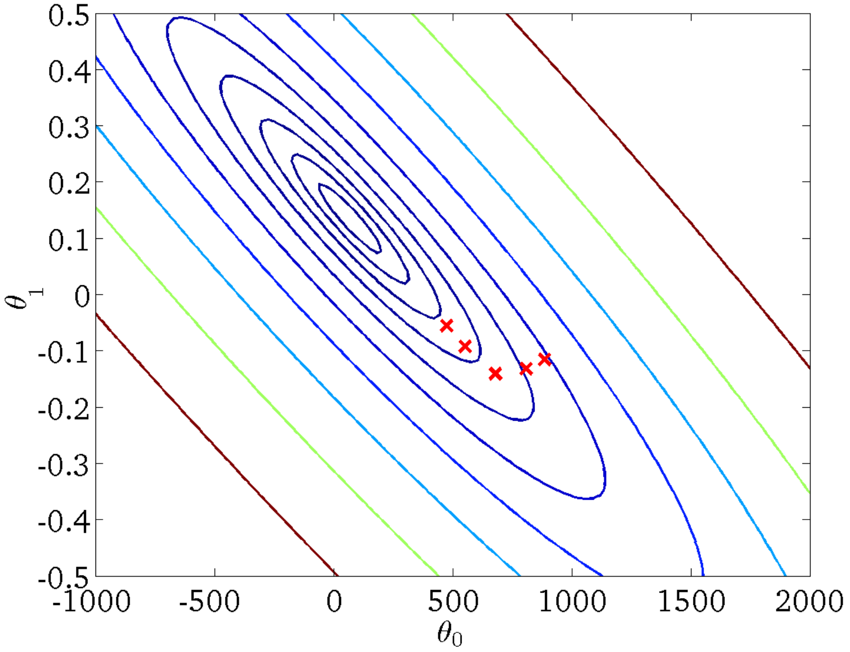
$$h_\theta(x)$$

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



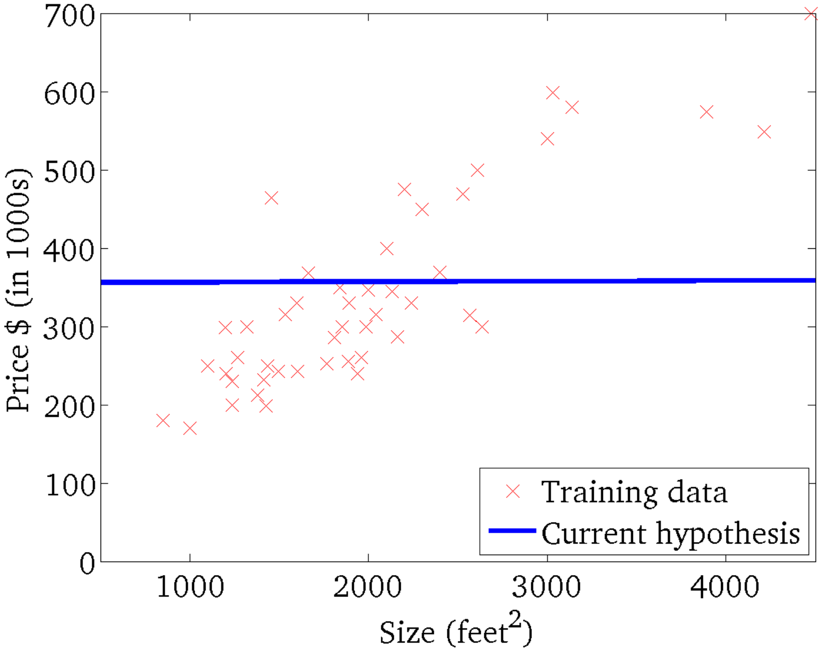
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



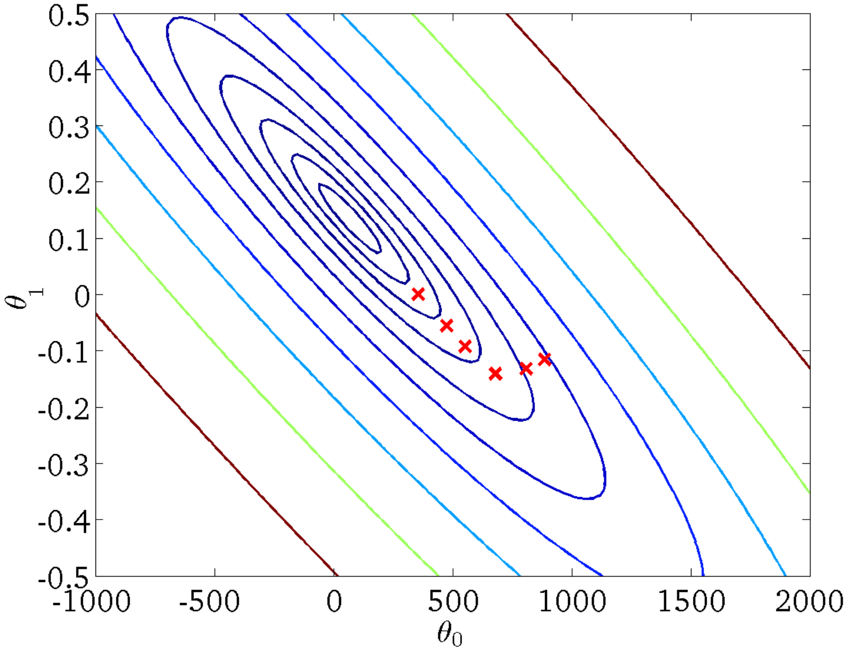
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

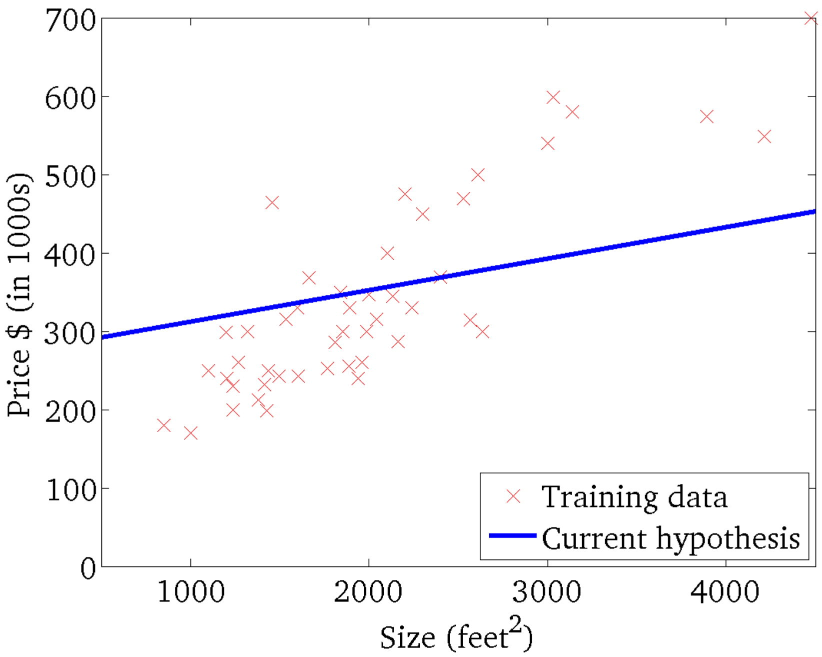
(function of the parameters θ_0, θ_1)



Andrew Ng

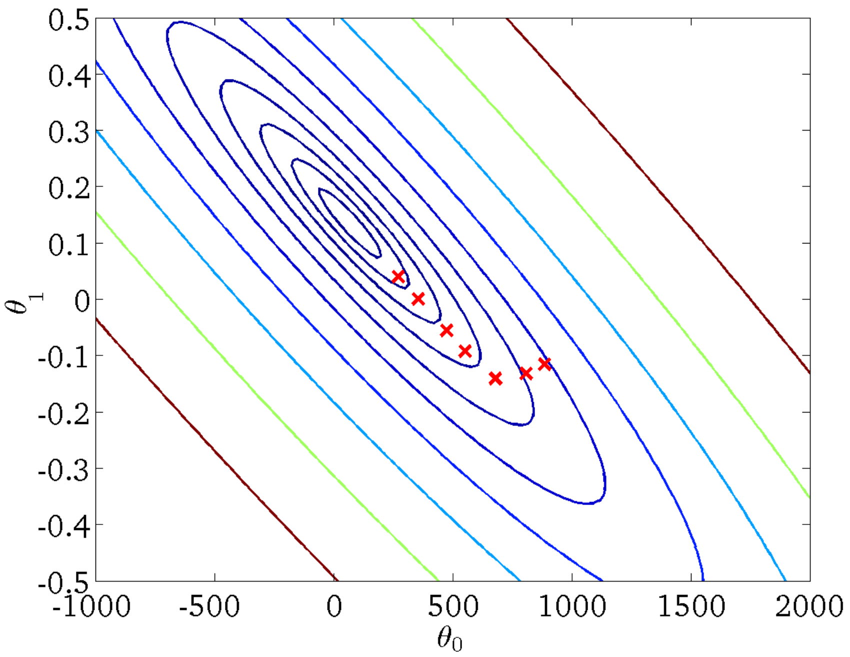
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

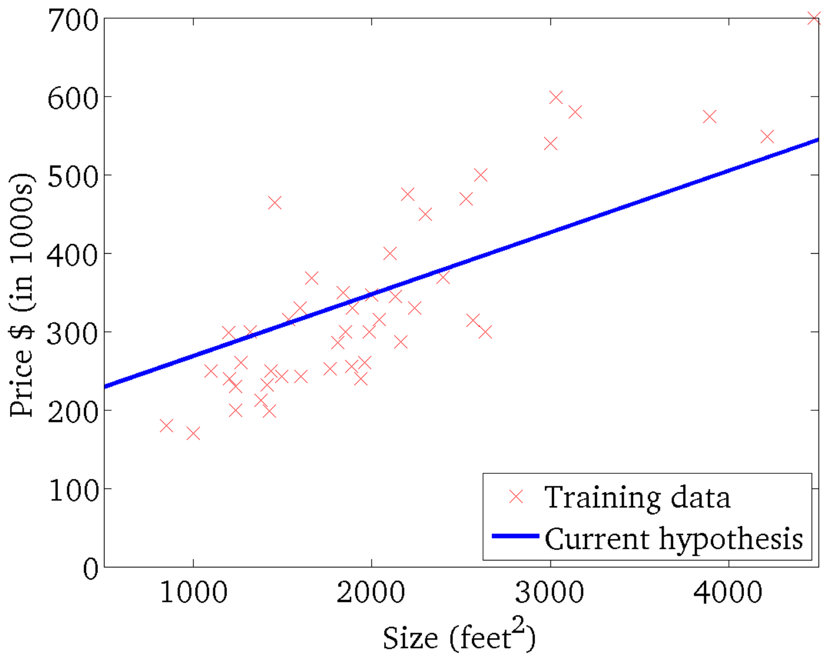
(function of the parameters θ_0, θ_1)



Andrew Ng

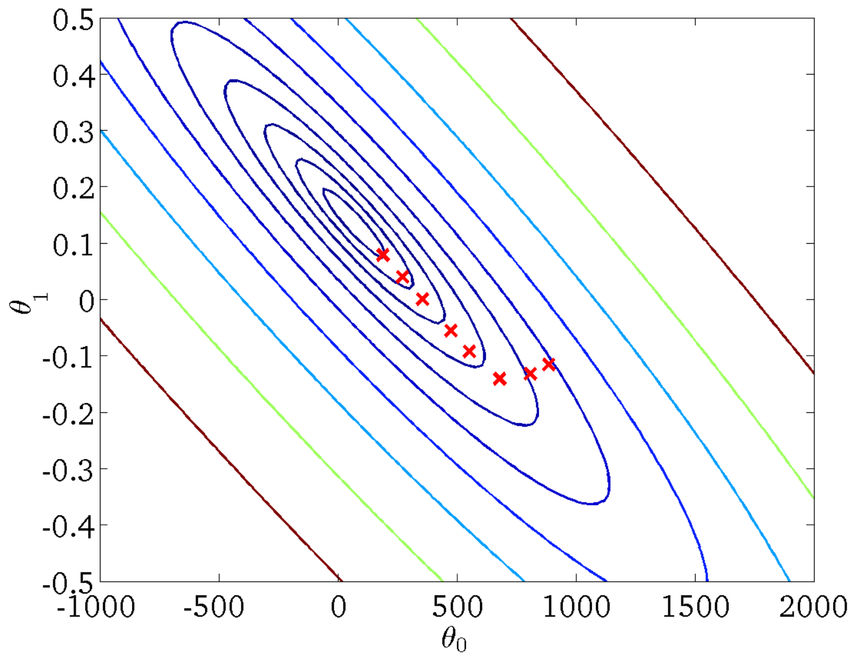
$h_\theta(x)$

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



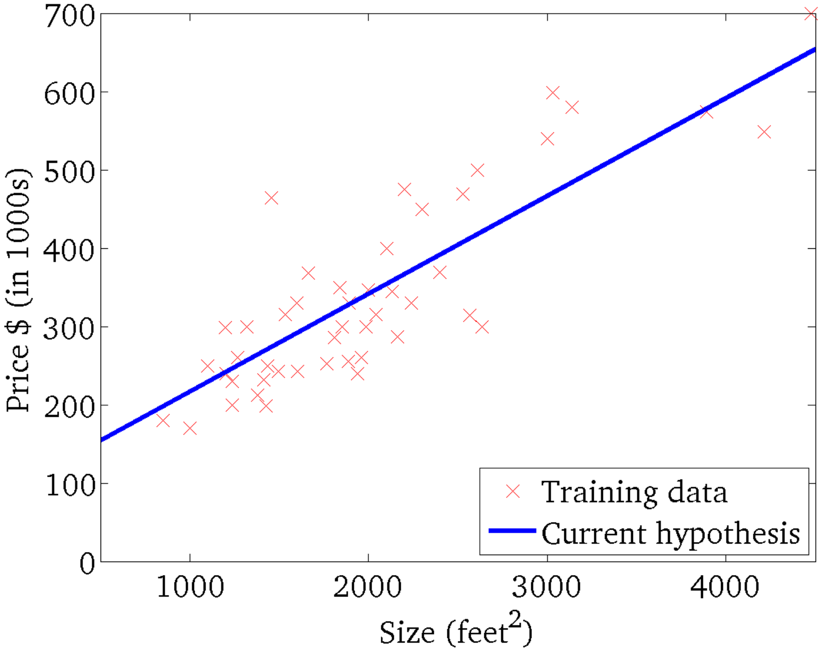
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



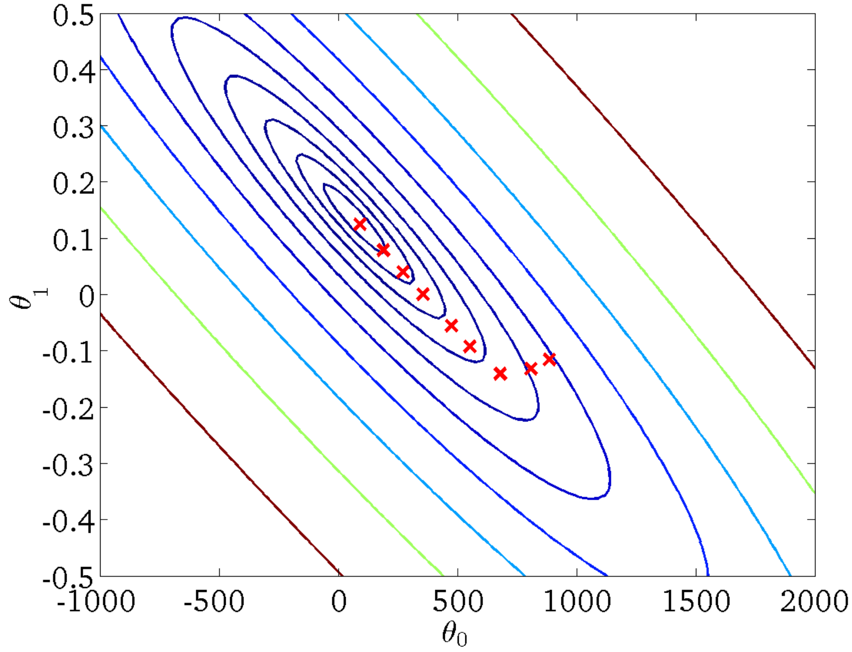
$$h_\theta(x)$$

(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)



Andrew Ng

Gradient descent algorithm

- Given initial values $\boldsymbol{\theta}^{(0)} = \left[\theta_0^{(0)}, \theta_1^{(0)}\right]$
- While (not convergence):

$$\theta_0^{(j)} = \theta_0^{(j-1)} - \alpha \frac{1}{M} \sum_{i=1}^{M} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})$$

$$\theta_1^{(j)} = \theta_1^{(j-1)} - \alpha \frac{1}{M} \sum_{i=1}^{M} (\theta_0 + \theta_1 x^{(i)} - y^{(i)}) x^{(i)}$$

Observation

- Each step of gradient descent uses ALL the training examples.

Batch gradient descent

- Each step of gradient descent uses ALL the training examples.

Problem with batch gradient descent

- When the number of training data is huge, say, M = 300,000,000, batch gradient descent becomes very slow.

Problem with batch gradient descent

- When the number of training data is huge, say, M = 300,000,000, <u>batch gradient descent becomes very slow.</u>

In 2017,

67 million Instagram posts uploaded each day!

656 million Tweets were generated each day!

4.3 billion Facebook messages posted daily!

https://blog.microfocus.com/how-much-data-is-created-on-the-internet-each-day/

Gradient descent algorithm

- Given initial values $\boldsymbol{\theta}^{(0)} = \left[\theta_0^{(0)}, \theta_1^{(0)}\right]$
- While (not convergence):

$$\theta_0^{(j)} = \theta_0^{(j-1)} - \alpha \frac{1}{M} \sum_{i=1}^{M} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})$$

$$\theta_1^{(j)} = \theta_1^{(j-1)} - \alpha \frac{1}{M} \sum_{i=1}^{M} (\theta_0 + \theta_1 x^{(i)} - y^{(i)}) x^{(i)}$$

- Given initial values $\boldsymbol{\theta}^{(0)} = \left[\theta_0^{(0)}, \theta_1^{(0)}\right]$

$$\theta_0 = \theta_0 - \alpha \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)$$

$$\theta_1 = \theta_1 - \alpha(\theta_0 + \theta_1 x^{(i)} - y^{(i)})\, x^{(i)}$$

- Given initial values $\boldsymbol{\theta}^{(0)} = \left[\theta_0^{(0)}, \theta_1^{(0)}\right]$

randomly shuffle indices [0,1,2, ..., M-1]

For i in shuffled indices {

$$\theta_0 = \theta_0 - \alpha \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)$$

$$\theta_1 = \theta_1 - \alpha(\theta_0 + \theta_1 x^{(i)} - y^{(i)})\, x^{(i)}$$

}

- Given initial values $\boldsymbol{\theta}^{(0)} = \left[\theta_0^{(0)}, \theta_1^{(0)}\right]$

one pass
one epoch

randomly shuffle indices [0,1,2, …, M-1]

For i in shuffled indices {

$$\theta_0 = \theta_0 - \alpha \left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)$$

$$\theta_1 = \theta_1 - \alpha(\theta_0 + \theta_1 x^{(i)} - y^{(i)})\, x^{(i)}$$

}

Stochastic gradient descent

- Given initial values $\boldsymbol{\theta}^{(0)} = \left[\theta_0^{(0)}, \theta_1^{(0)}\right]$

  Repeat {

  randomly shuffle indices [0,1,2, …, M-1]

  For i in shuffled indices {

  $$\theta_0 = \theta_0 - \alpha\left(\theta_0 + \theta_1 x^{(i)} - y^{(i)}\right)$$

  $$\theta_1 = \theta_1 - \alpha(\theta_0 + \theta_1 x^{(i)} - y^{(i)})\, x^{(i)}$$

  }

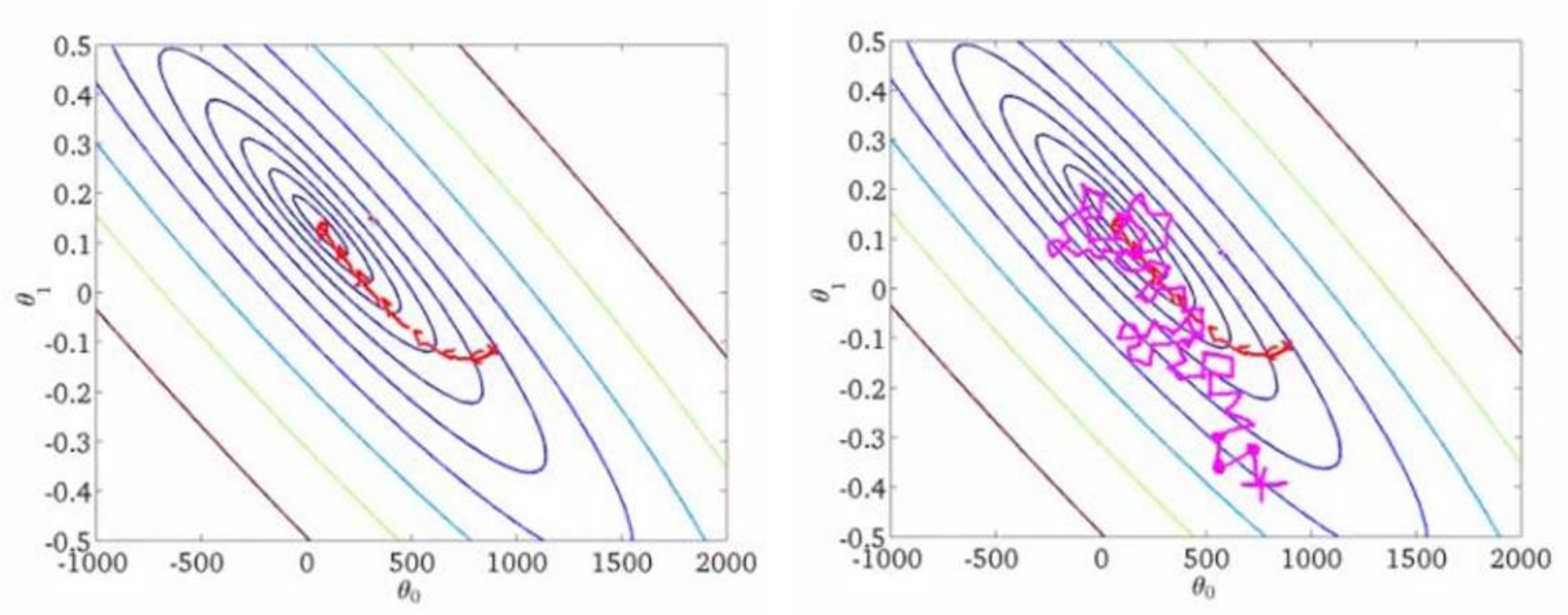
  }

one pass
one epoch

SGD vs BGD

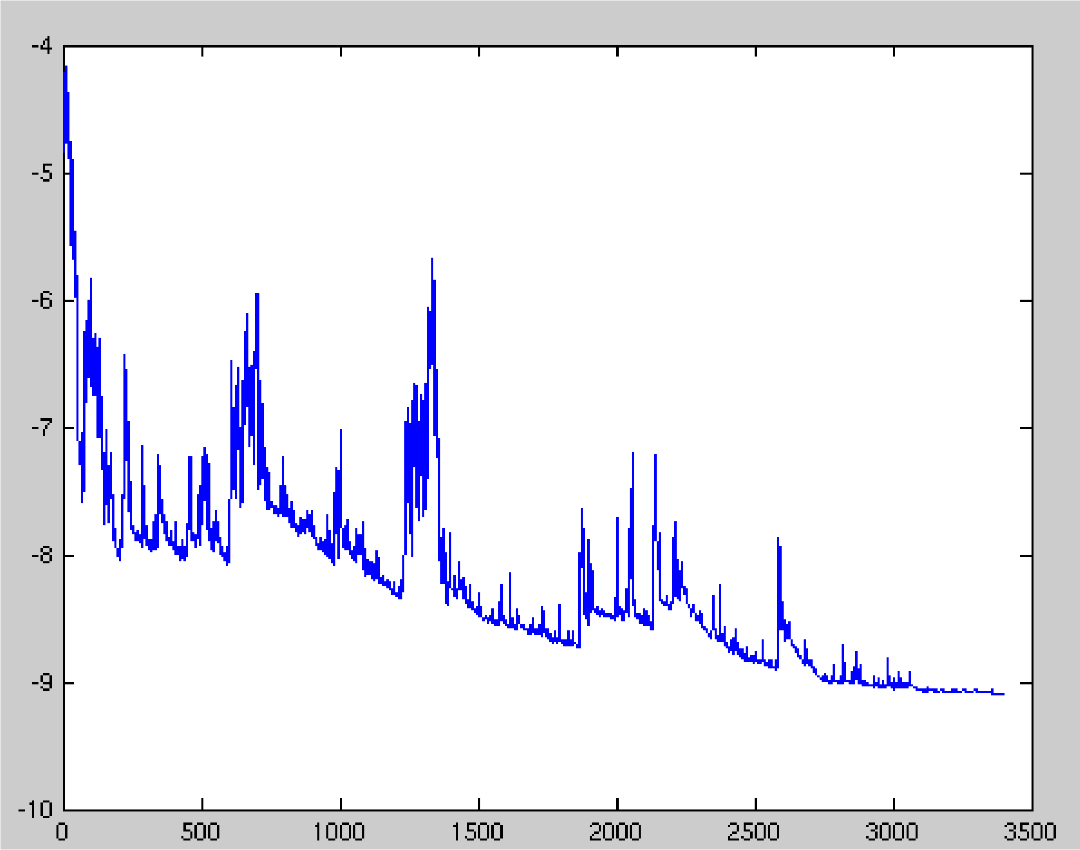
- SGD much faster
- Search path very irregular
- Cost function bounces up and down, decreasing only on average
- Over time, it ends up close to minimum, but never settles down.

SGD vs BGD



Picture taken from
https://www.cs.cmu.edu/~yuxiangw/docs/SSGD.pdf

SGD cost function



https://upload.wikimedia.org/wikipedia/commons/f/f3/Stogra.png

Mini-batch gradient descent

- Mini-batch uses a small number ($1 < \# < M$) of training examples to update model parameters

Batch vs stochastic vs Mini-batch

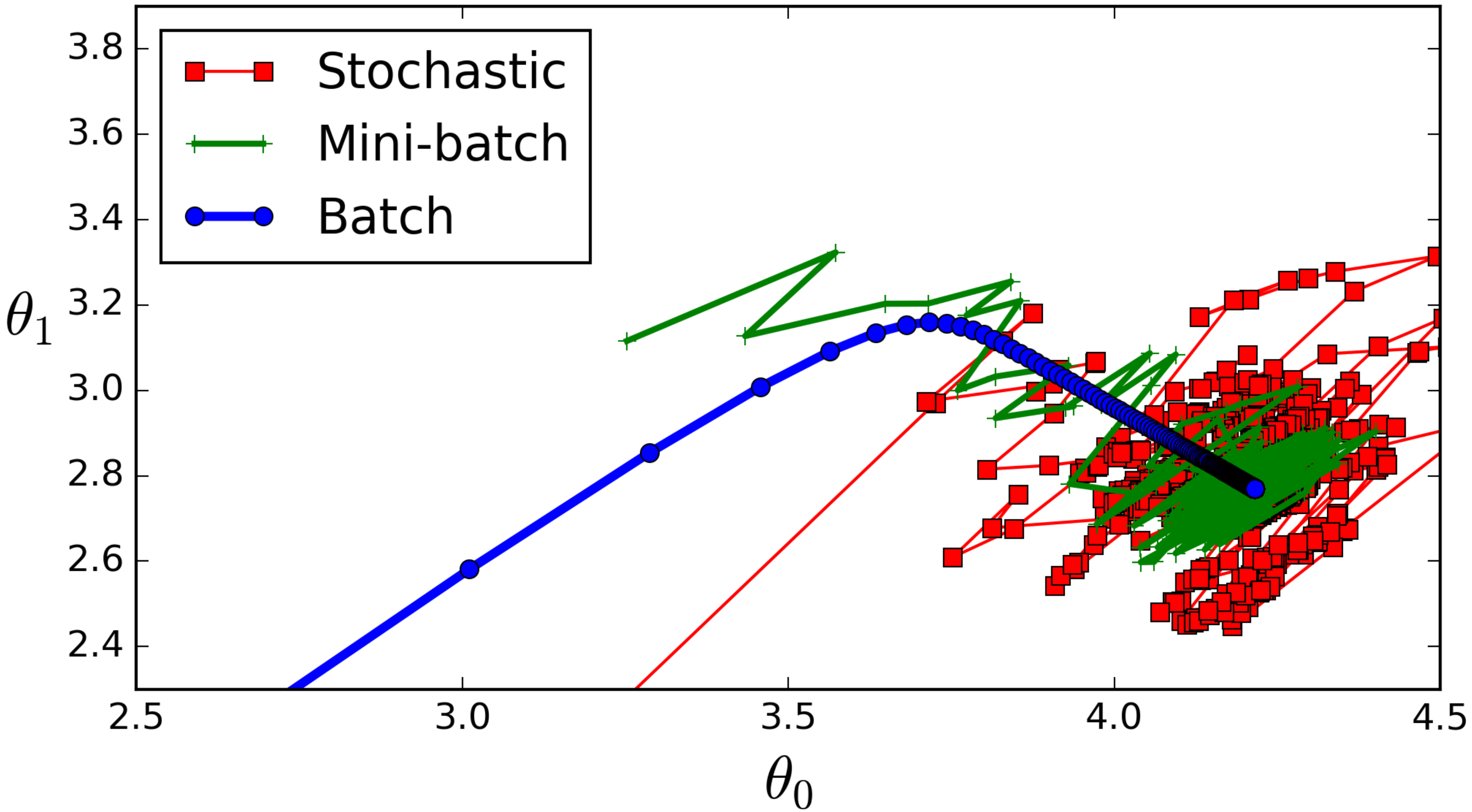
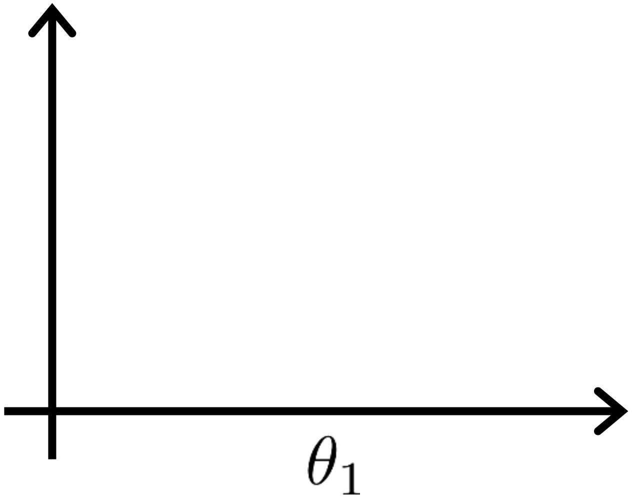


Figure from Aurelien Geron's ML book, page 120

Learning rate

$$\theta_1^{(j)} = \theta_1^{(j-1)} - \alpha \frac{\partial}{\partial \theta_1} \mathrm{J}(\theta_1^{(j-1)})$$

If α is too small, gradient descent becomes very slow.



Learning rate

$$\theta_1^{(j)} = \theta_1^{(j-1)} - \alpha \frac{\partial}{\partial \theta_1} \mathrm{J}(\theta_1^{(j-1)})$$

If α is too large, you might overshoot the minimum. It may fail to converge, sometimes even diverge.