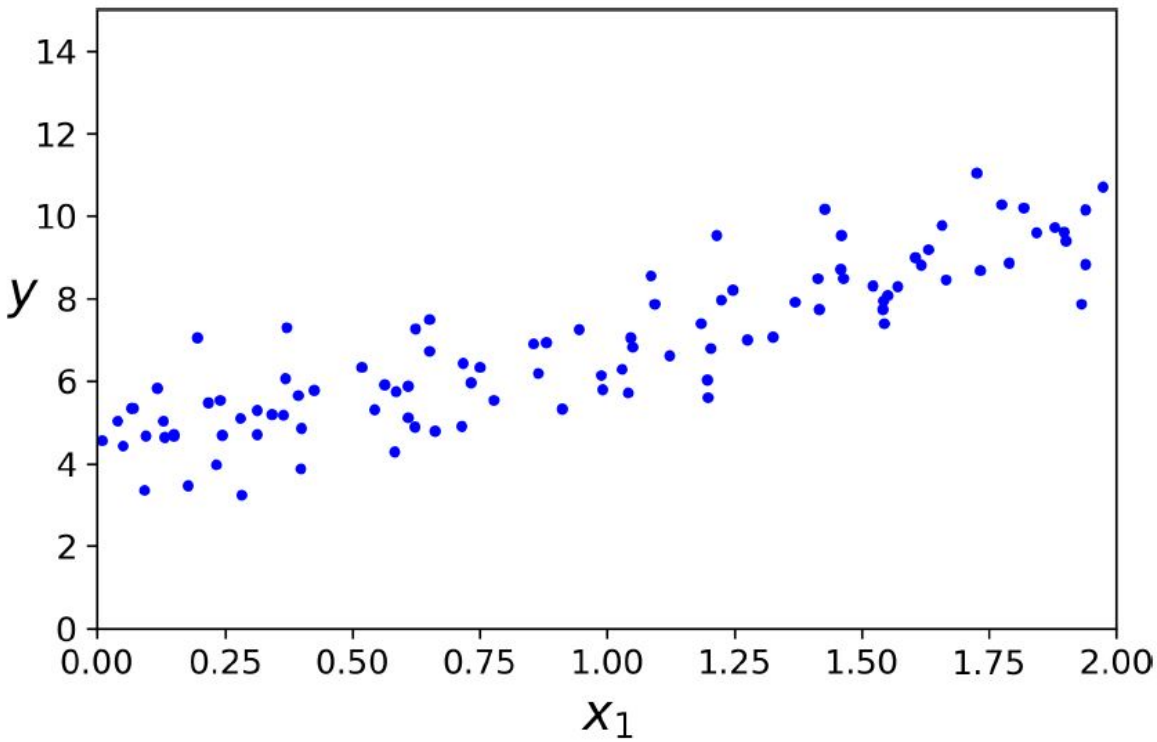
Linear Regression with one variable
AI4003 - Applied Machine Learning

Dr. Mohsin Kamal

Department of Electrical Engineering
National University of Computer and Emerging Sciences, Lahore, Pakistan

Randomly generated linear dataset



Supervised Learning
Given the "right answer" for each example in the data.

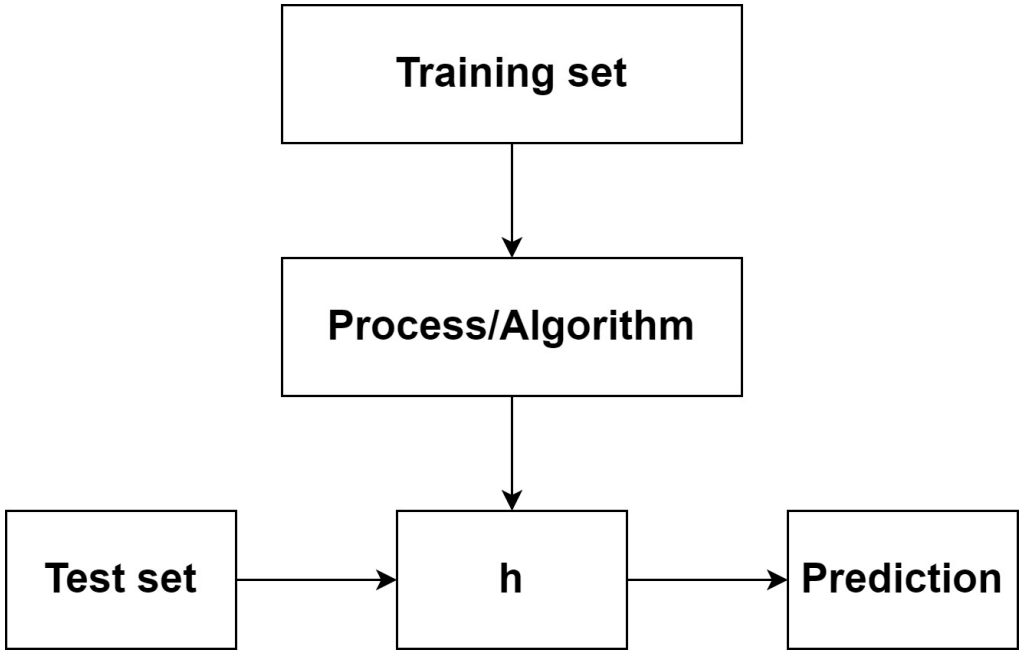
Regression Problem
Predict real-valued output

Training set of housing prices

Area in *feet*^{2} **(*x*)**	**Price (*y*)**
100	100,000
150	140,000
200	250,000
220	290,000
...	...

Notation:

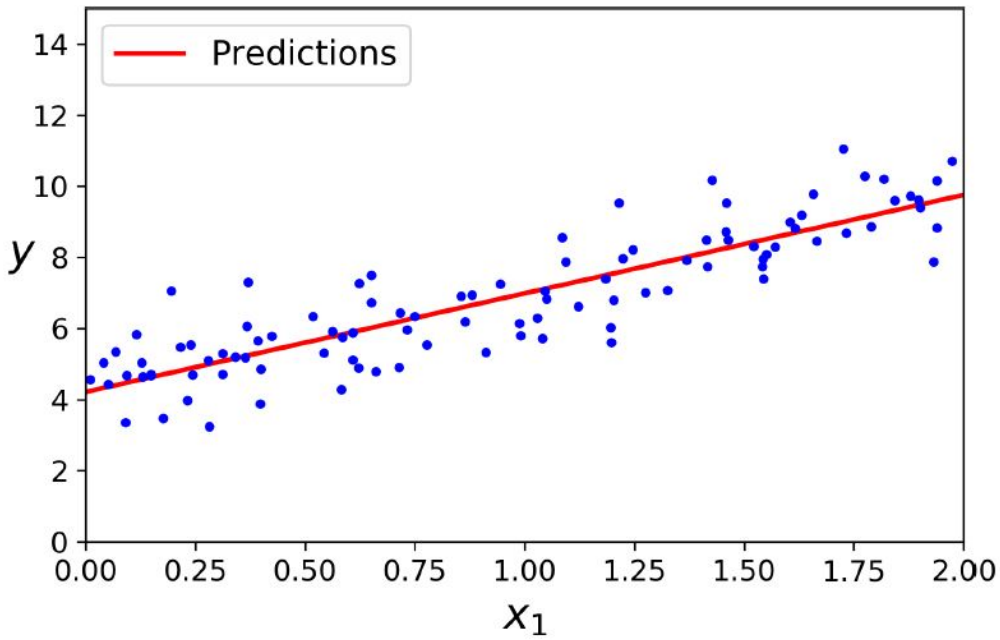
- m = Number of training examples
- x's = "input" variable / features
- y's = "output" variable / "target" variable



How do we represent h?

What to do?

- Training a model means setting its parameters so that the model best fits the training set.
- To train a Linear Regression model, you need to find the value of θ that minimizes the Mean Squared Error.



Cost function: It is a function that measures the performance of a Machine Learning model for given data.

Table 1 : Training set

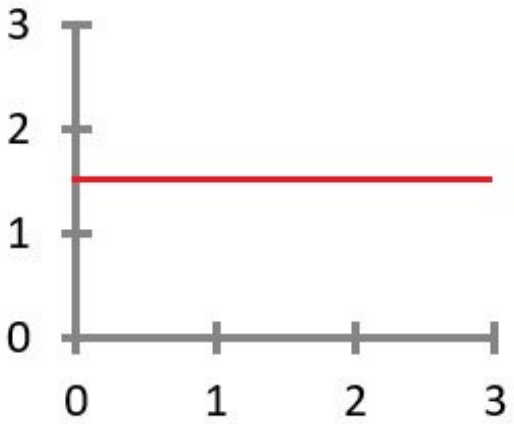
Area in $feet^2$ **(**x**)**	**Price (**y**)**
100	100,000
150	140,000
200	250,000
220	290,000
...	...

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

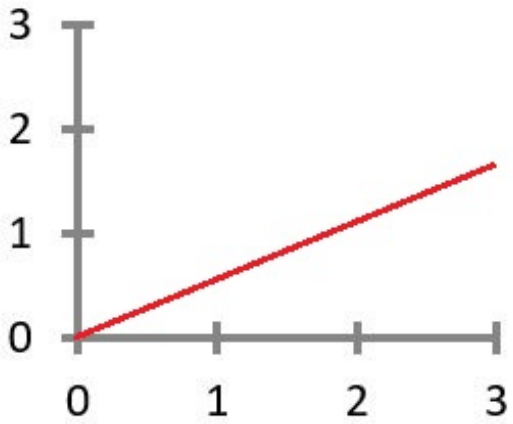
θ's: Parameters

How to choose θ's?

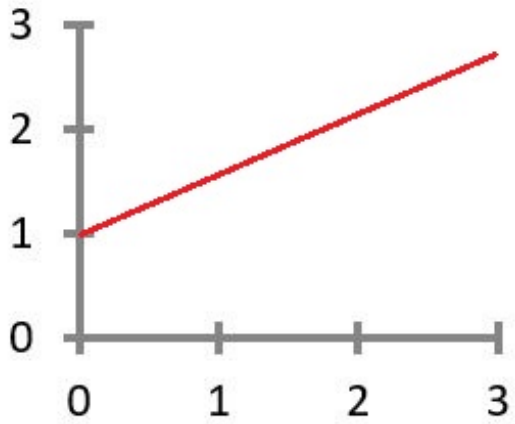
$$h_\theta(x) = \theta_0 + \theta_1 x$$



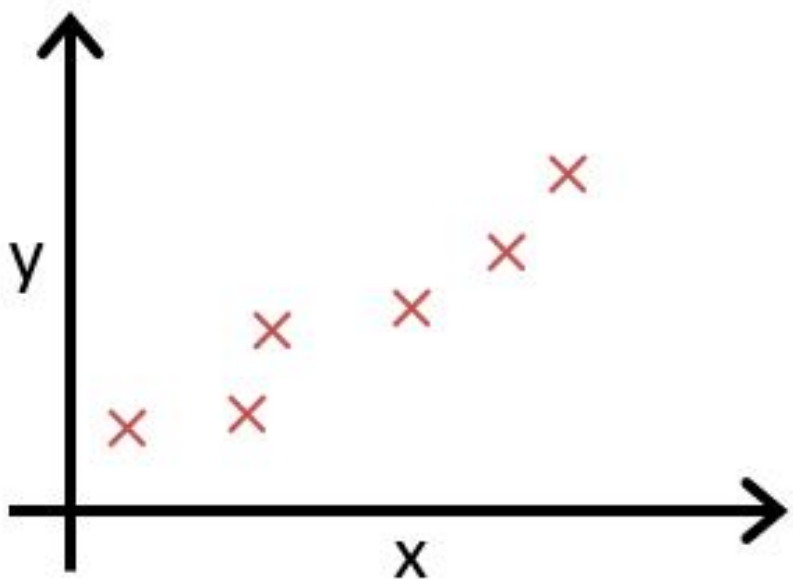
$\theta_0 = 1.5$
$\theta_1 = 0$



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



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



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

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

Parameters: θ_0, θ_1

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

Goal: $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

Simplified

$h_\theta(x) = \theta_1 x$ when $\theta_0 = 0$

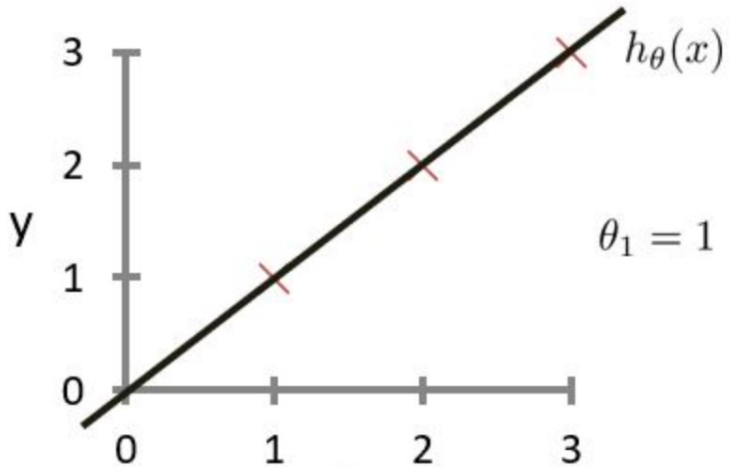
θ_1

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

$\min_{\theta_1} J(\theta_1)$

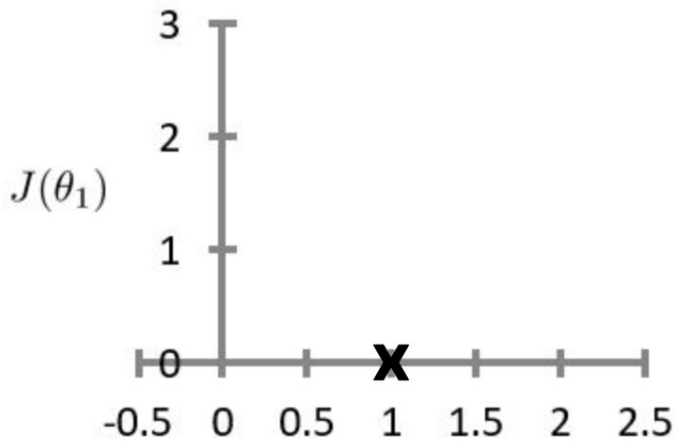
$h_\theta(x)$

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



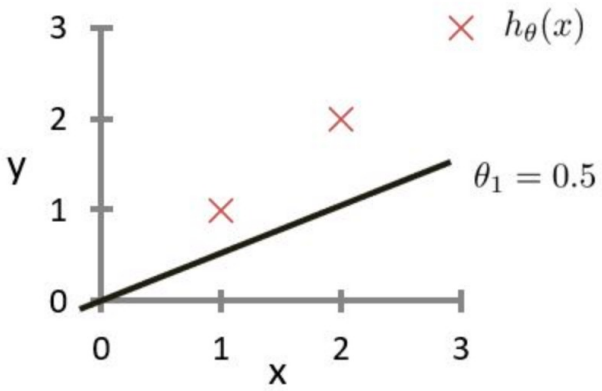
$J(\theta_1)$

(function of the parameter θ_1)



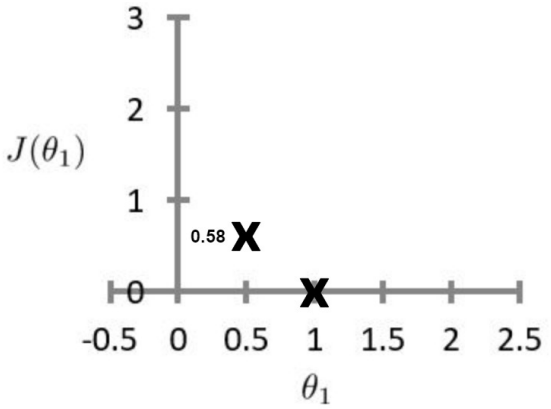
$h_\theta(x)$

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



$J(\theta_1)$

(function of the parameter θ_1)

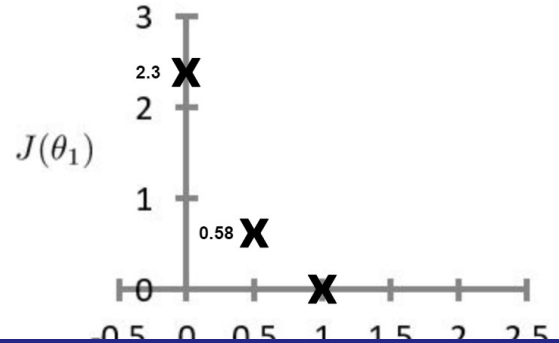


$h_\theta(x)$

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

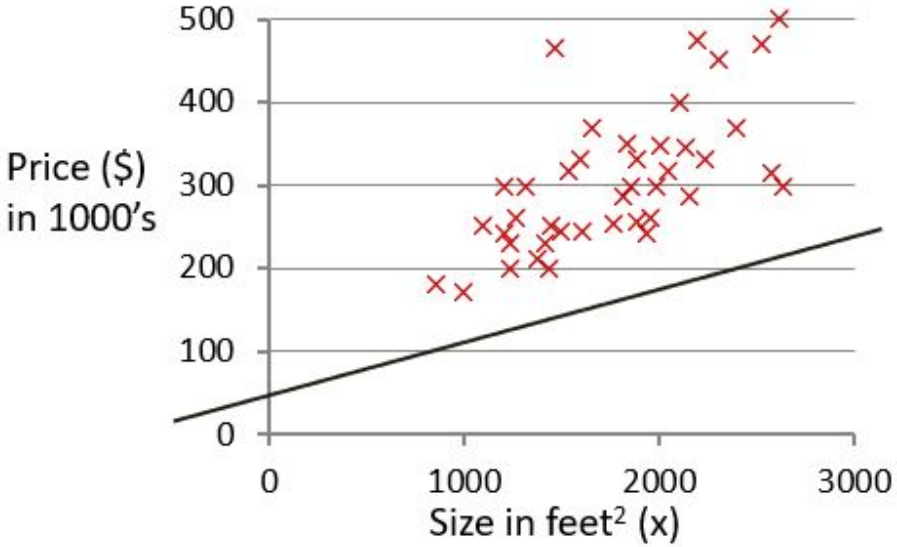
$J(\theta_1)$

(function of the parameter θ_1)



$h_\theta(x)$

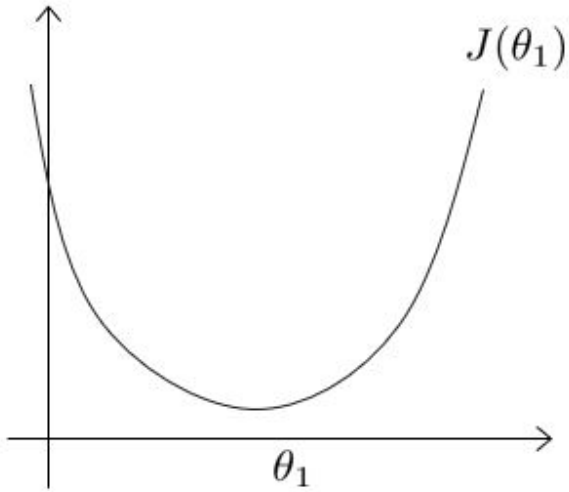
(for fixed θ_0, θ_1, this is a function of x)



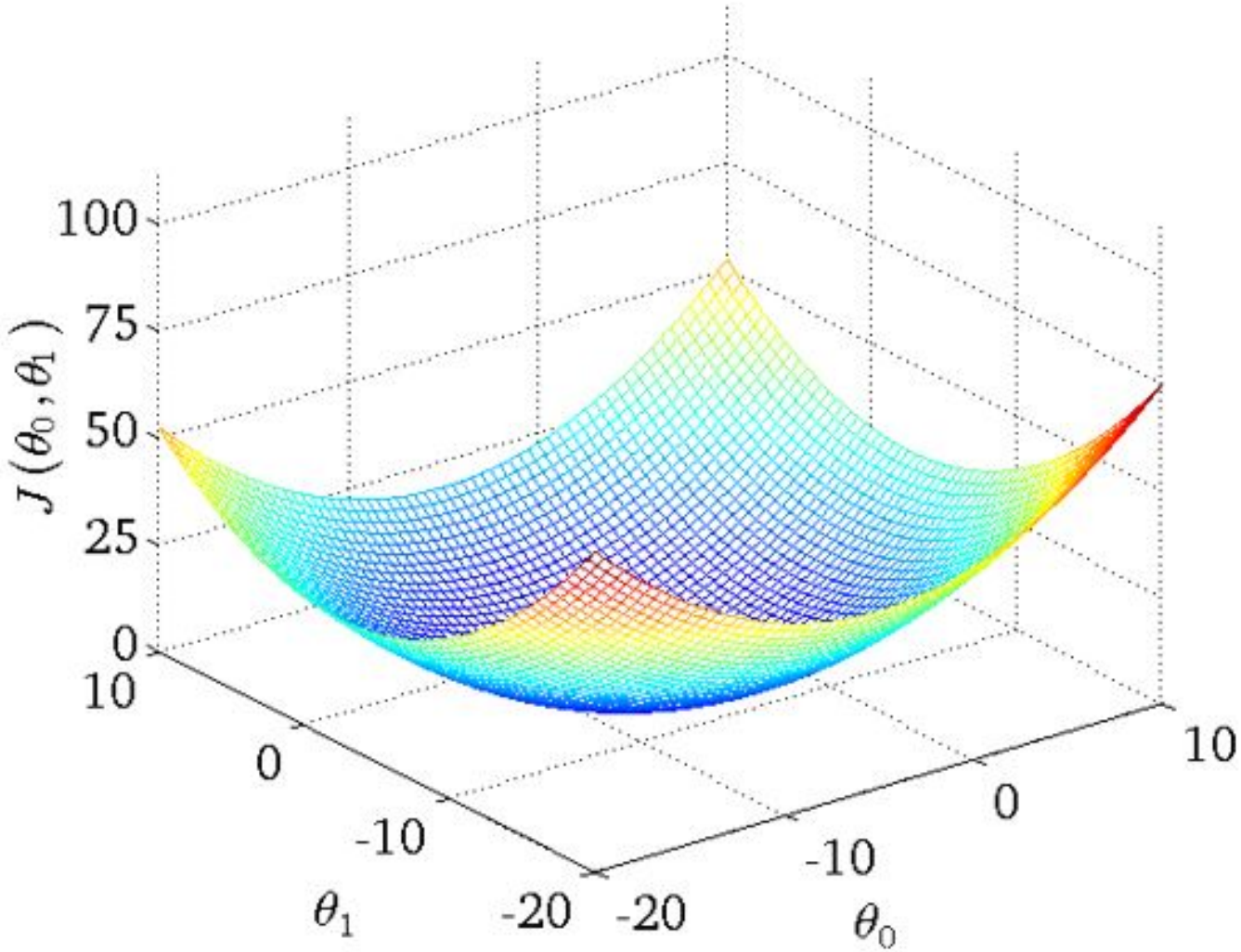
$h_\theta(x) = 50 + 0.06x$

$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

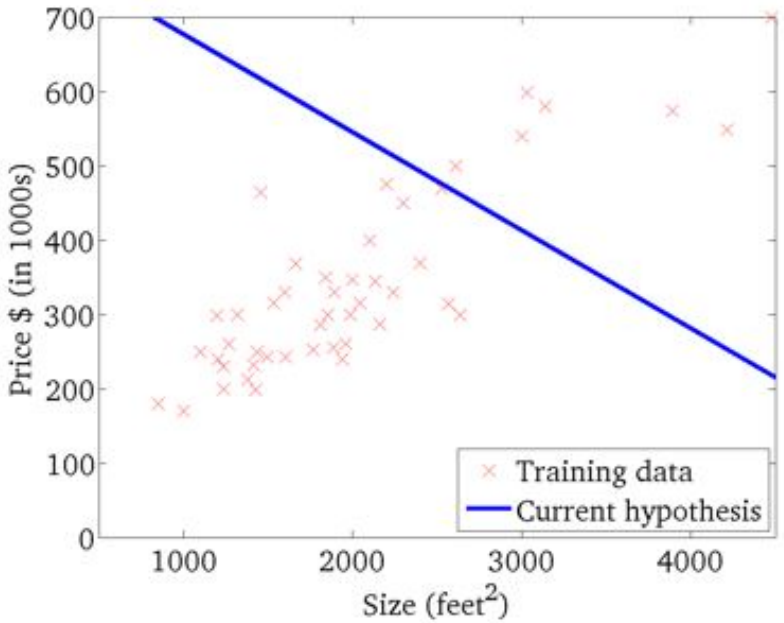


Fortunately, the MSE cost function for a Linear Regression model happens to be a convex function, which means that if you pick any two points on the curve, the line segment joining them never crosses the curve. This implies that there are no local minima, just one global minimum.


100
75
50
25
0
$J(\theta_0, \theta_1)$
10
0
-10
-20
θ_1
-20
-10
0
10
θ_0

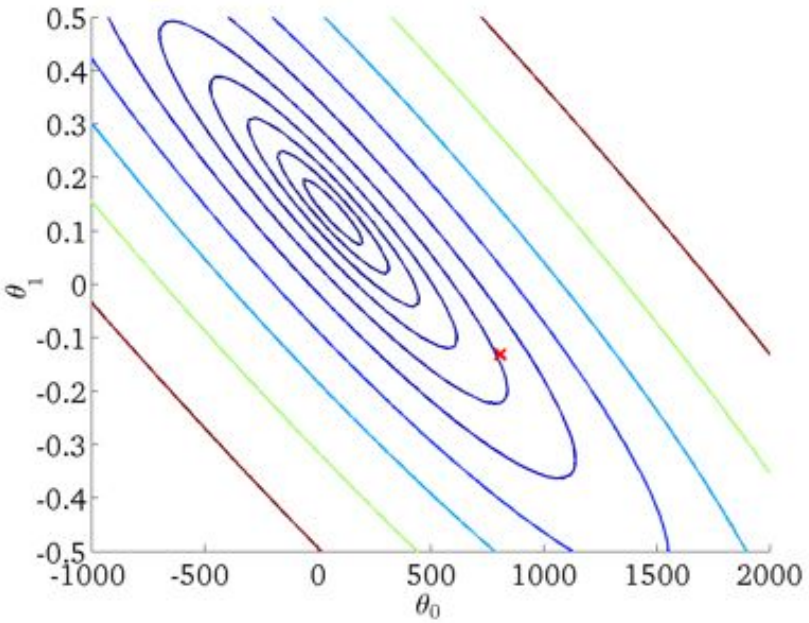
$h_\theta(x)$

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



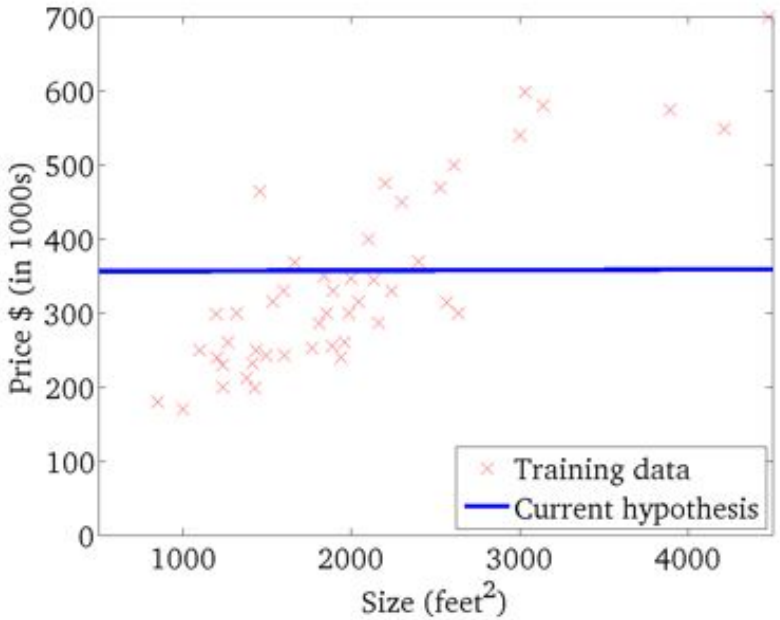
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



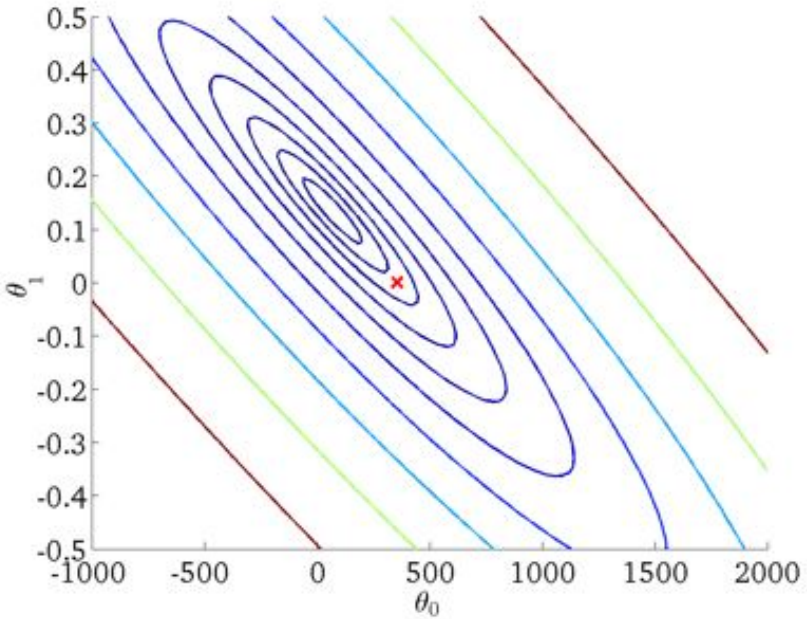
$h_\theta(x)$

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



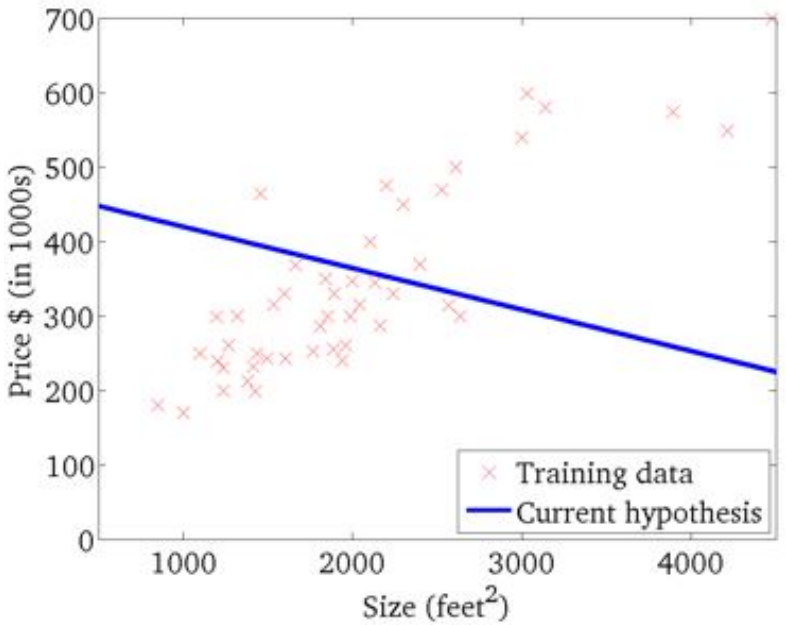
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



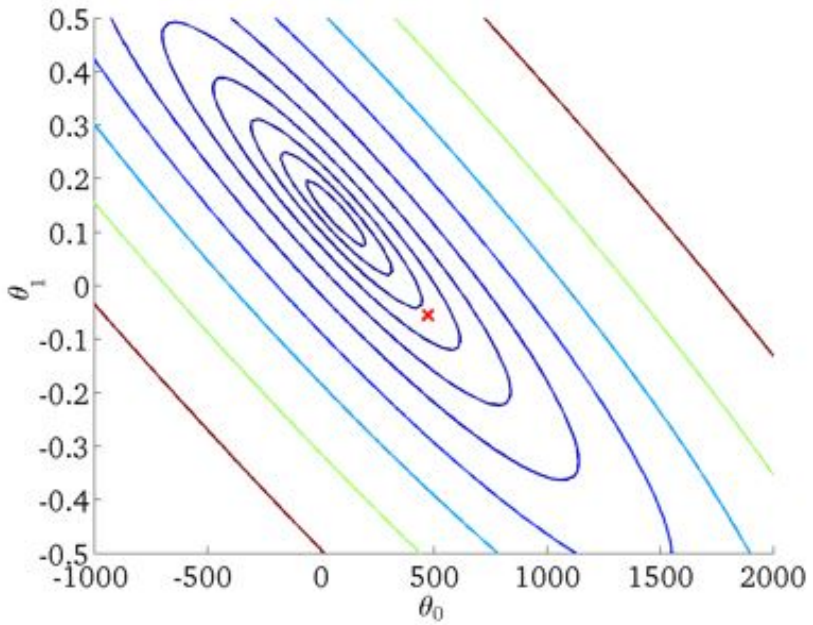
$h_\theta(x)$

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



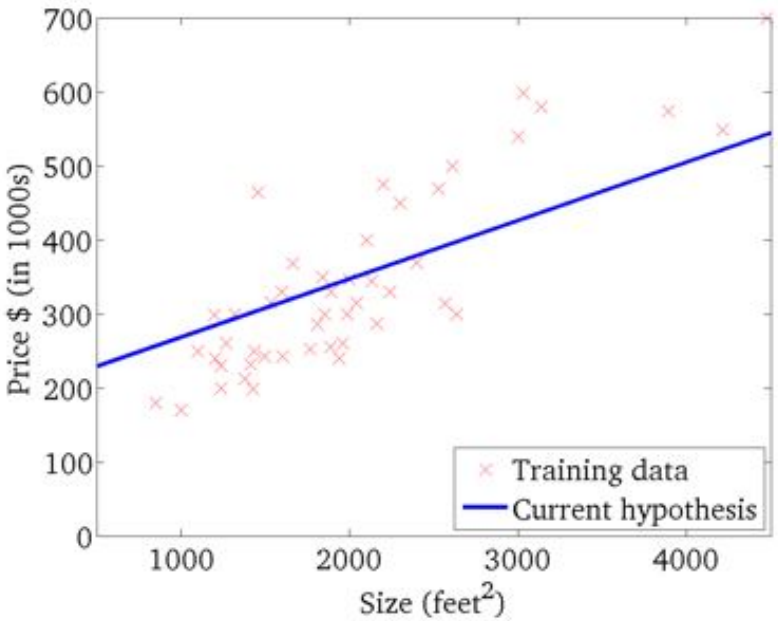
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



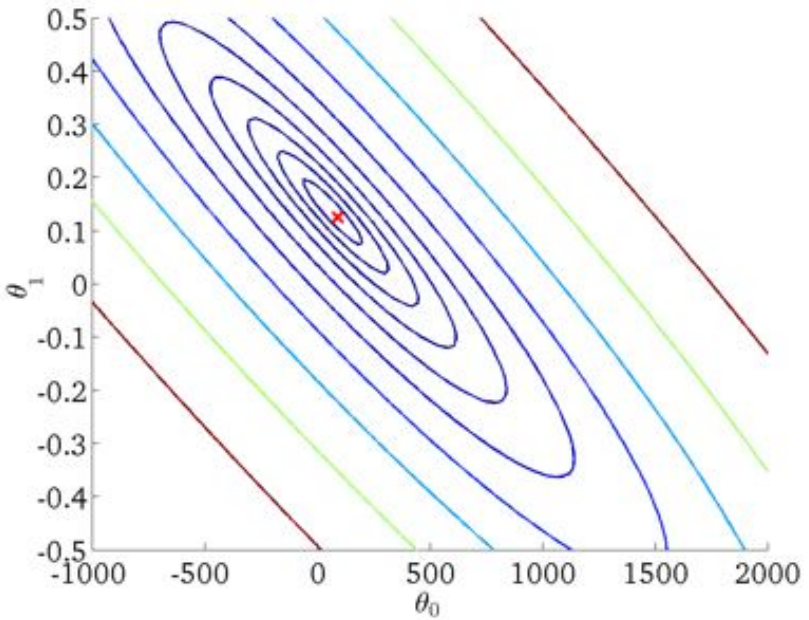
$h_\theta(x)$

(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)



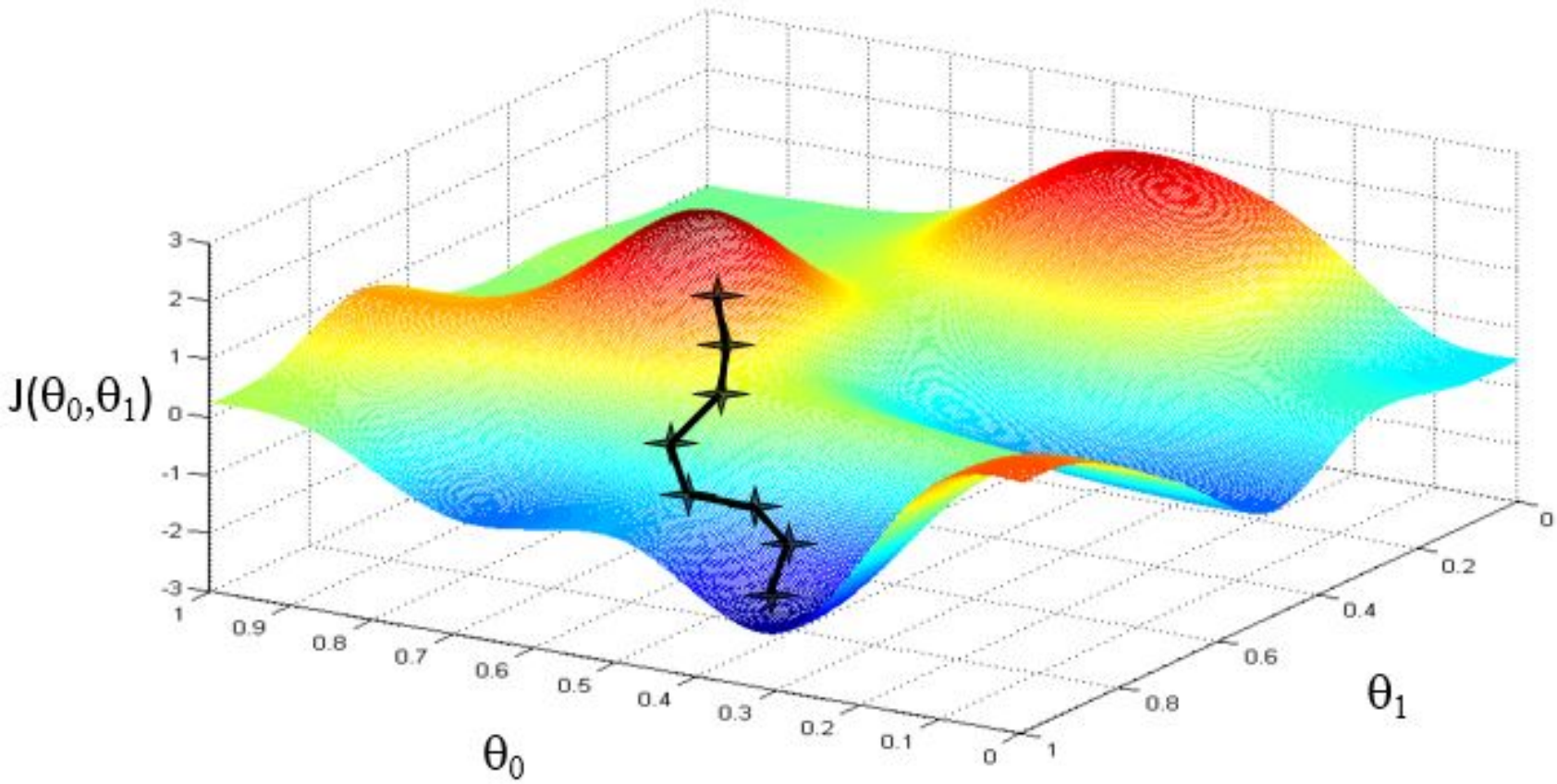
Gradient Descent is a very generic optimization algorithm capable of finding optimal solutions to a wide range of problems. The general idea of Gradient Descent is to tweak parameters iteratively in order to minimize a cost function.

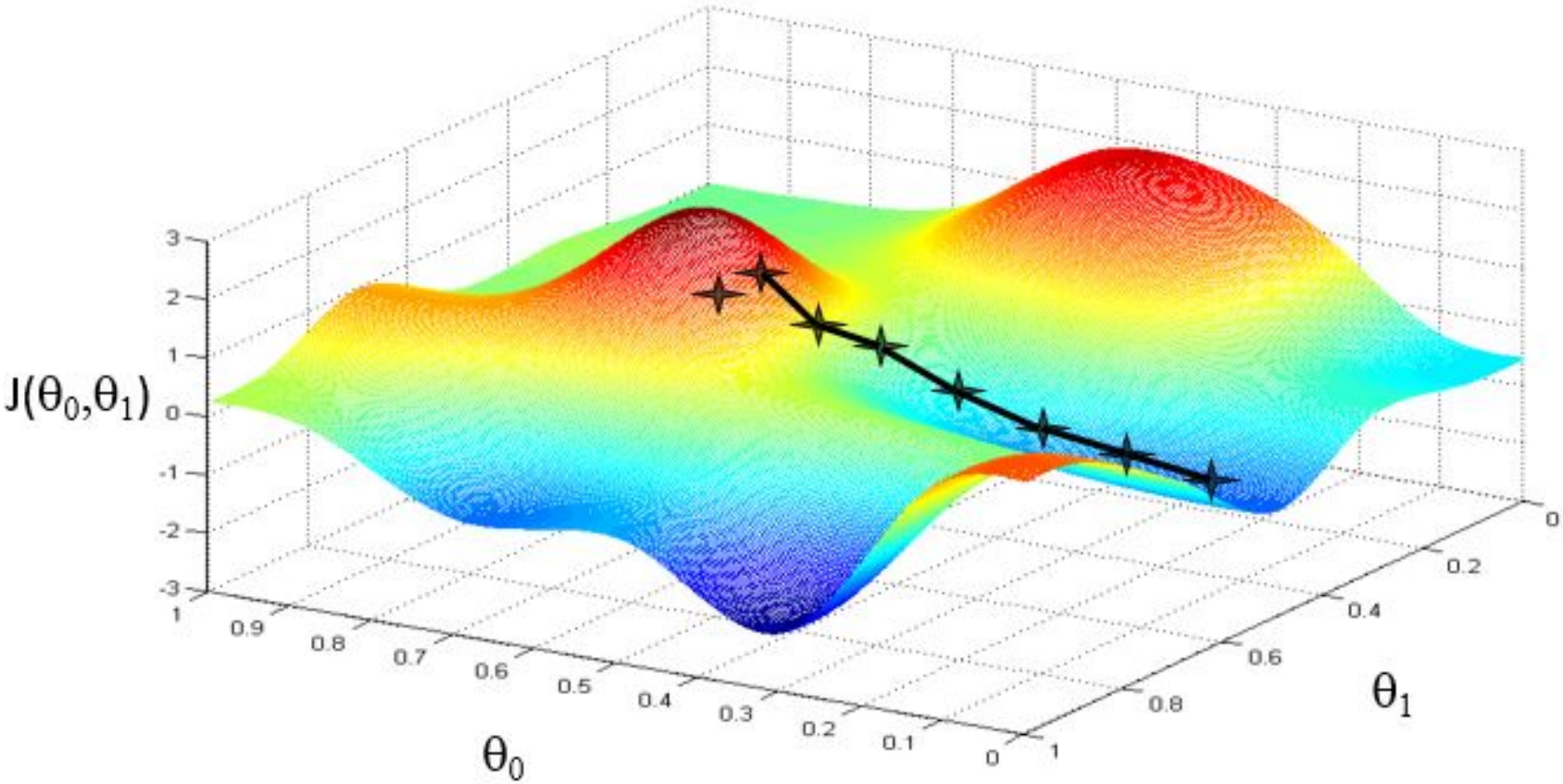
- Have some function $J(\theta_0, \theta_1)$
- Goal: $\min_{\theta_0,\theta_1} J(\theta_0, \theta_1)$

Outline:

- Start with some θ_0, θ_1
- Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we hopefully end up at the minimum

Summary: You start by filling θ with random values (this is called random initialization), and then you improve it gradually, taking one baby step at a time, each step attempting to decrease the cost function (e.g., the MSE), until the algorithm converges to a minimum


J(θ0,θ1)
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ0
0
0.2
0.4
0.6
0.8
1
θ1


J(θ0,θ1)
3
2
1
0
-1
-2
-3
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
θ0
0
0.2
0.4
0.6
0.8
1
θ1

Gradient descent Algorithms

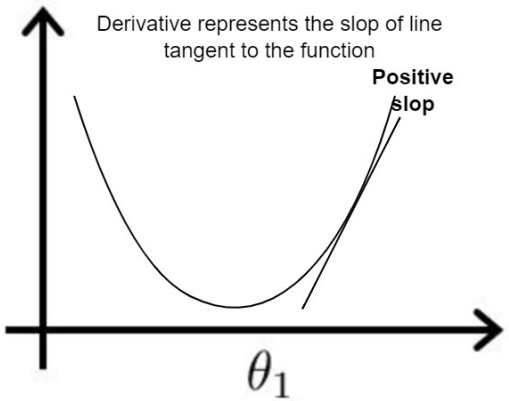
- Batch gradient descent
- Stochastic gradient descent
- Mini-batch gradient descent

Batch gradient descent

repeat until convergence:

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

Correct: Simultaneous update	**Incorrect**
$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$	$\text{temp0} := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$
$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$	$\theta_0 := \text{temp0}$
$\theta_0 := \text{temp0}$	$\text{temp1} := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$
$\theta_1 := \text{temp1}$	$\theta_1 := \text{temp1}$



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

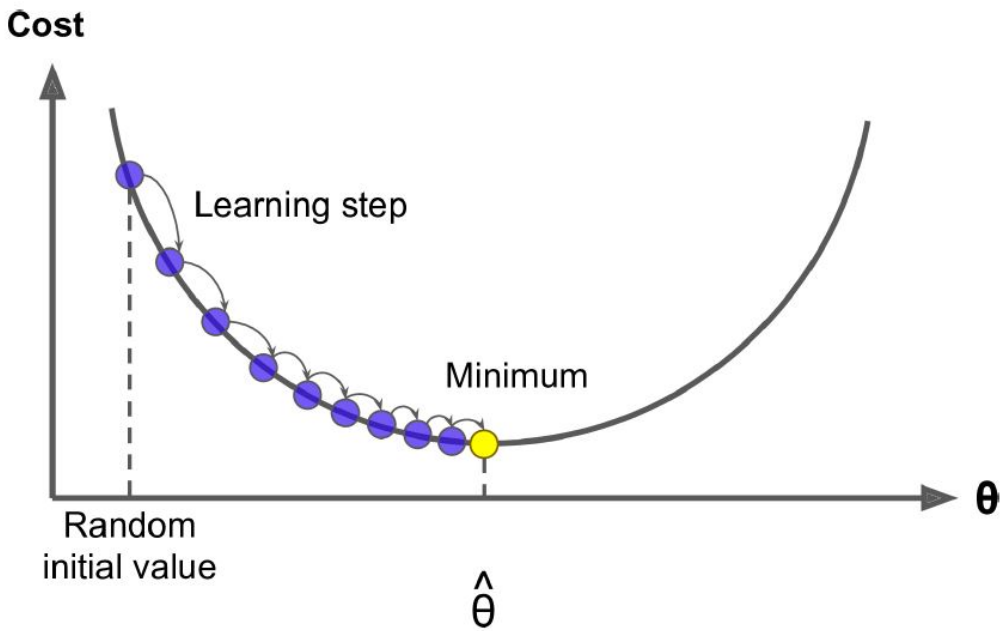
- If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.

Batch gradient descent

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

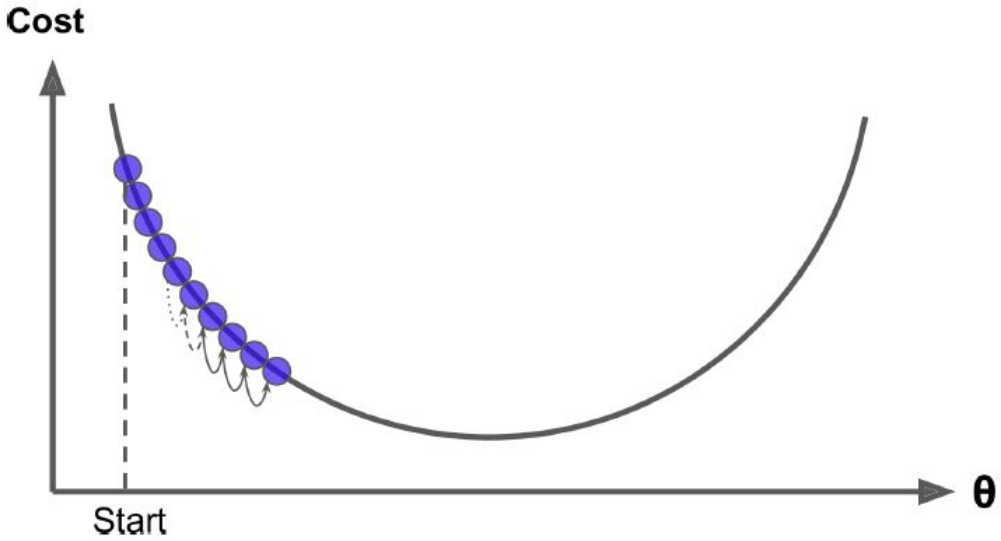
$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

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



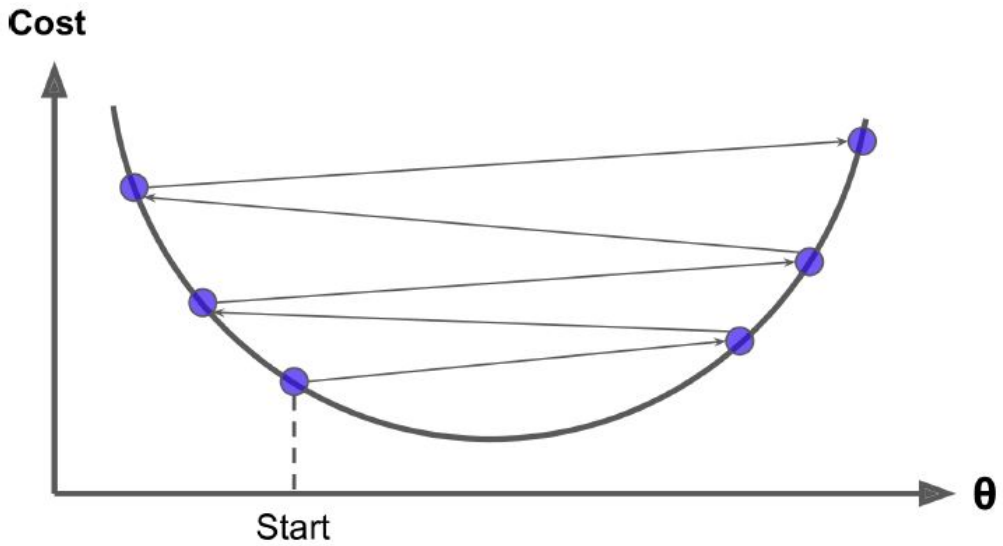
Batch gradient descent

If the **learning rate is too small**, then the algorithm will have to go through many iterations to converge, which will take a long time.



Batch gradient descent

- If the **learning rate is too high**, you might jump across the valley and end up on the other side, possibly even higher up than you were before.
- This might make the algorithm diverge, with larger and larger values, failing to find a good solution.



Batch gradient descent

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

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

$$j = 0 : \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})$$

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

Gradient descent algorithm becomes:

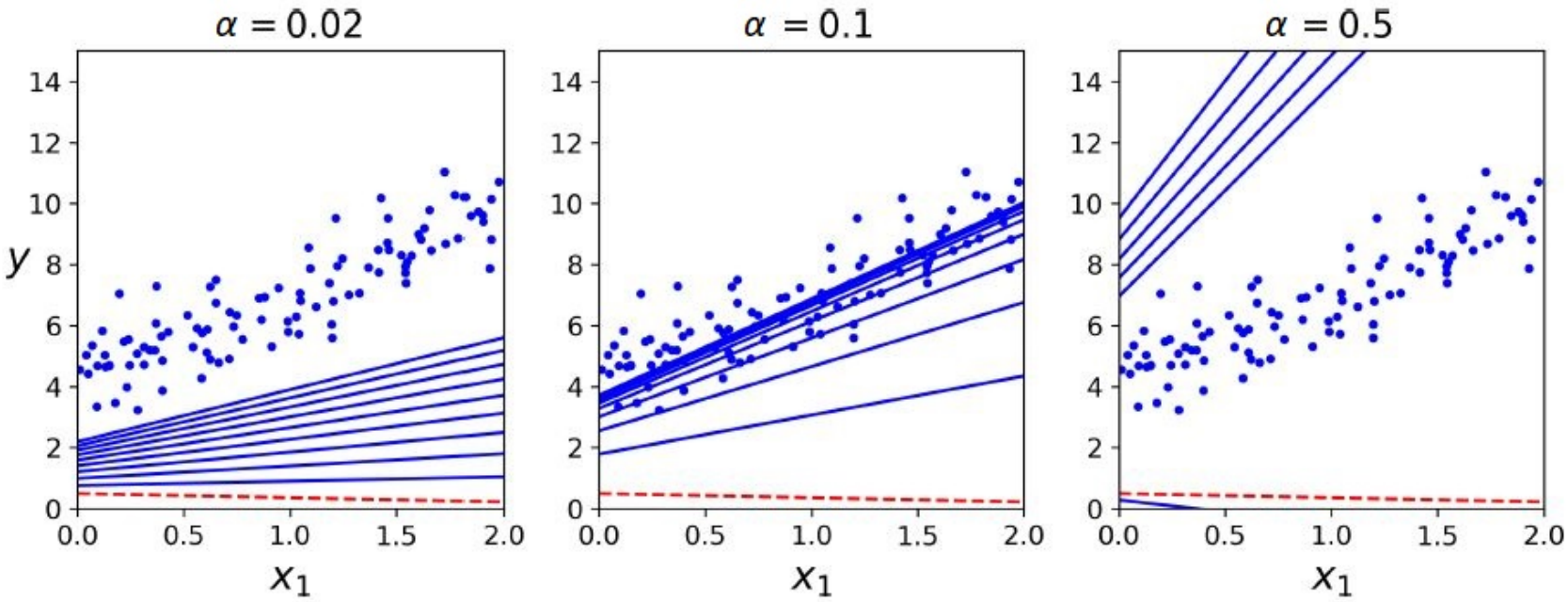
Repeat until convergence {

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

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

}

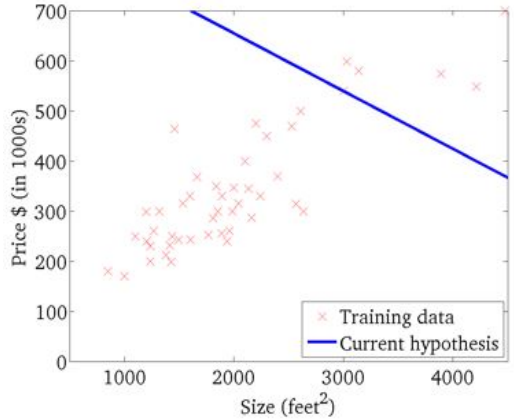
Batch gradient descent



Batch gradient descent

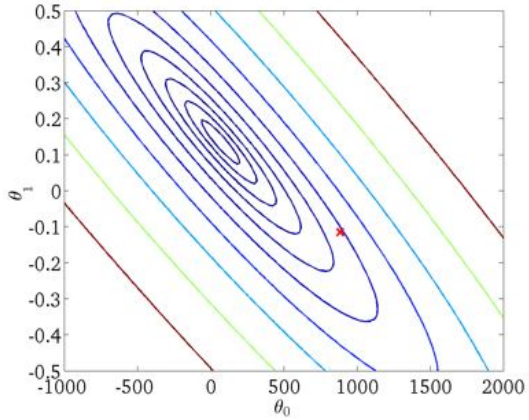
$h_\theta(x)$

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



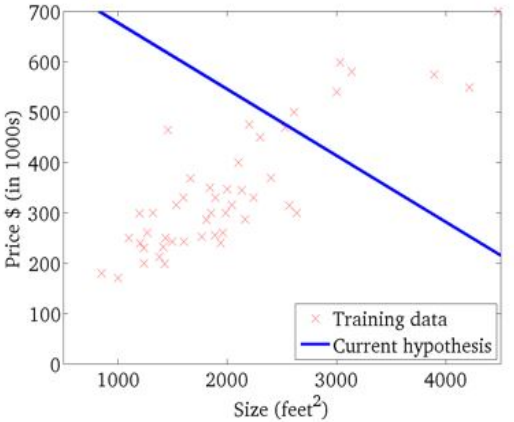
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



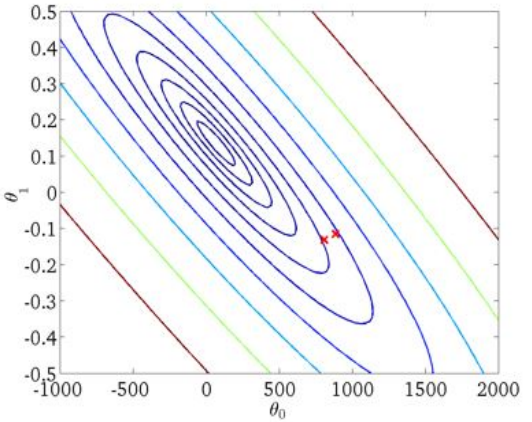
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

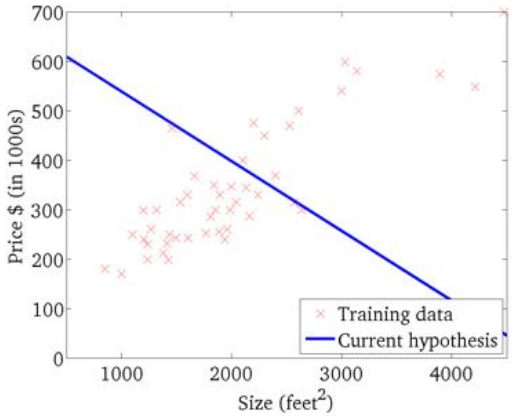
(function of the parameters θ_0, θ_1)



Batch gradient descent

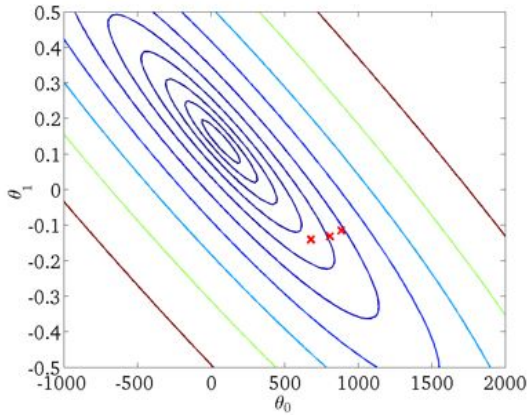
$h_\theta(x)$

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



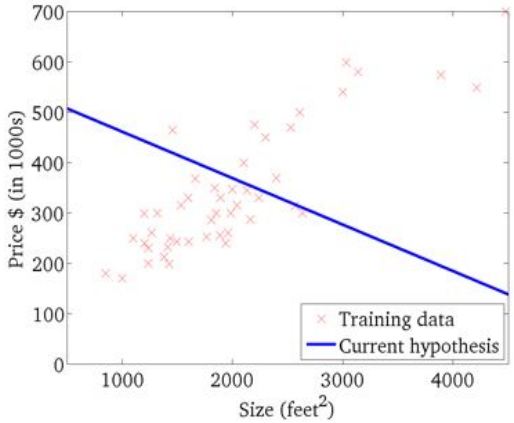
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



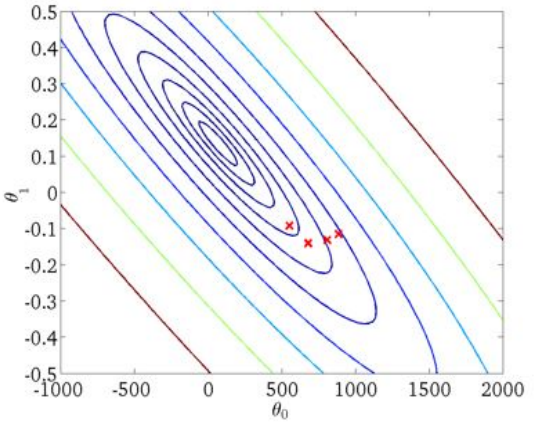
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

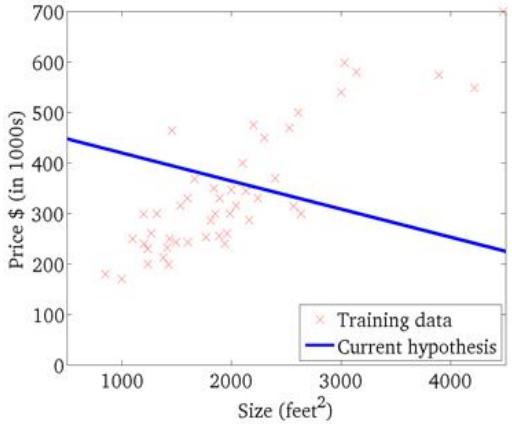
(function of the parameters θ_0, θ_1)



Batch gradient descent

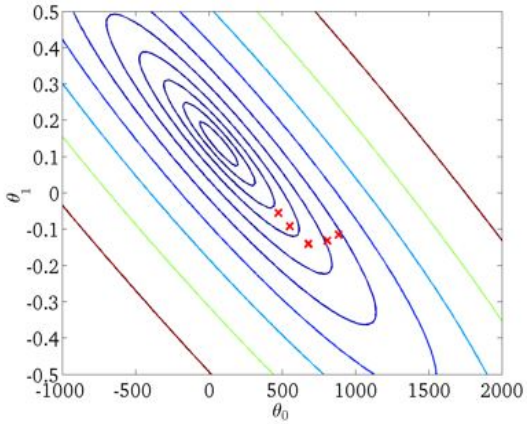
$h_\theta(x)$

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



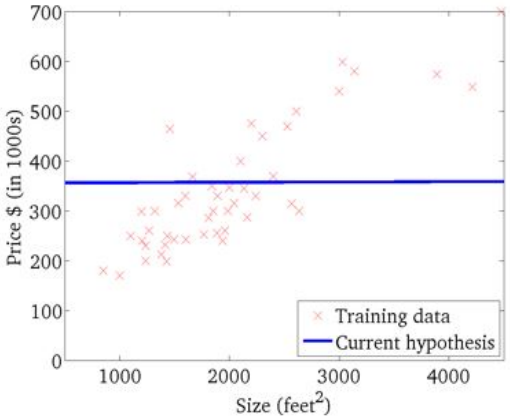
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



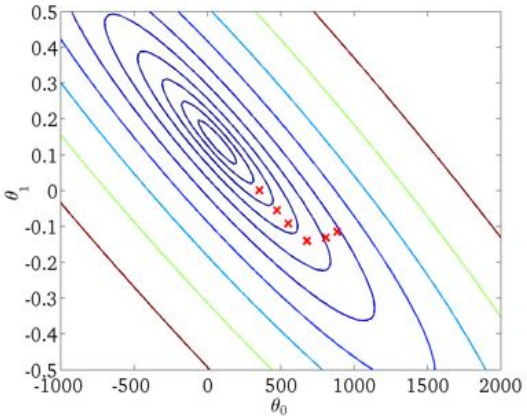
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

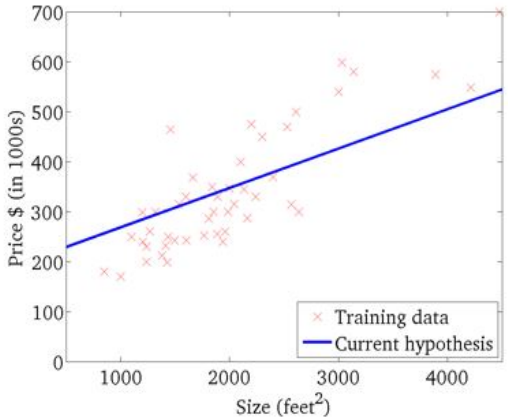
(function of the parameters θ_0, θ_1)



Batch gradient descent

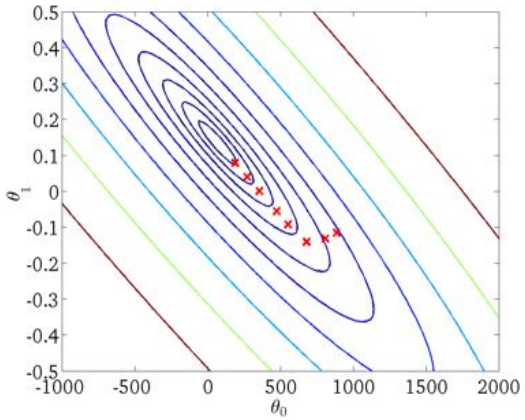
$h_\theta(x)$

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



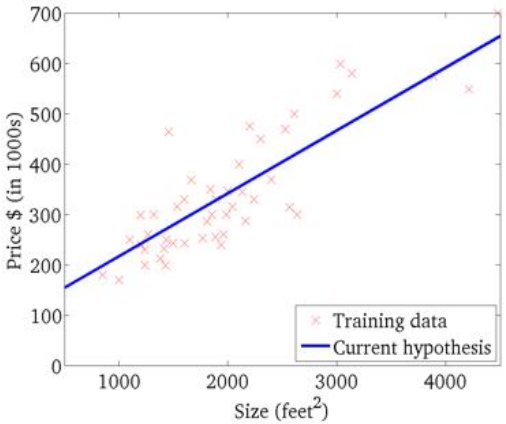
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



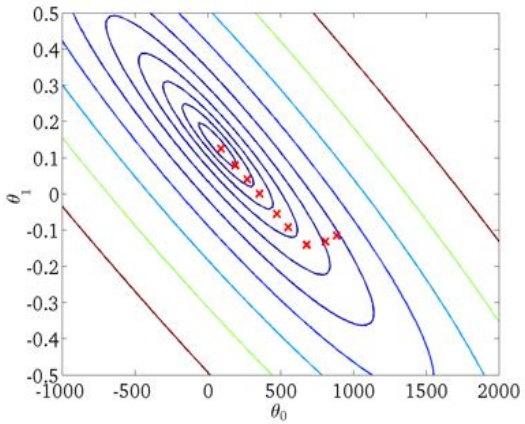
$h_\theta(x)$

(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)



Batch gradient descent

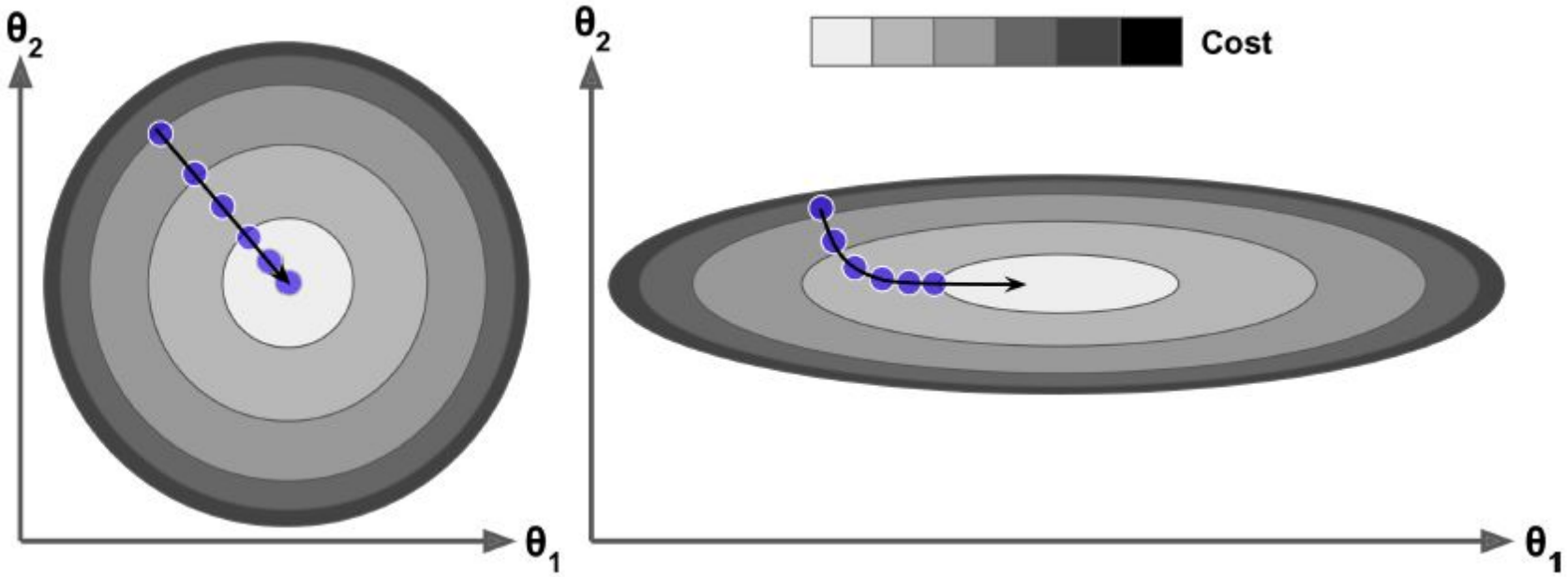


Figure 1 : Gradient descent with and without feature scaling

Stochastic gradient descent

- The main problem with Batch Gradient Descent is the fact that it uses the whole training set to compute the gradients at every step, which makes it very slow when the training set is large.
- Stochastic Gradient Descent just picks a random instance in the training set at every step and computes the gradients based only on that single instance.
- This makes the algorithm much faster since it has very little data to manipulate at every iteration.

Stochastic gradient descent

- On the other hand, due to its stochastic (i.e., random) nature, this algorithm is much less regular than Batch Gradient Descent: instead of gently decreasing until it reaches the minimum, the cost function will bounce up and down, decreasing only on average.
- Over time it will end up very close to the minimum, but once it gets there it will continue to bounce around, never settling down. So once the algorithm stops, the final parameter values are good, but not optimal.
- When the cost function is very irregular, this can actually help the algorithm jump out of local minima, so Stochastic Gradient Descent has a better chance of finding the global minimum than Batch Gradient Descent does.

Stochastic gradient descent

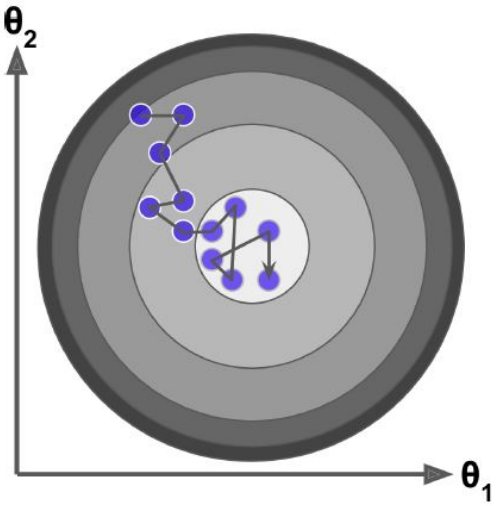


Figure 2 : Stochastic Gradient Descent

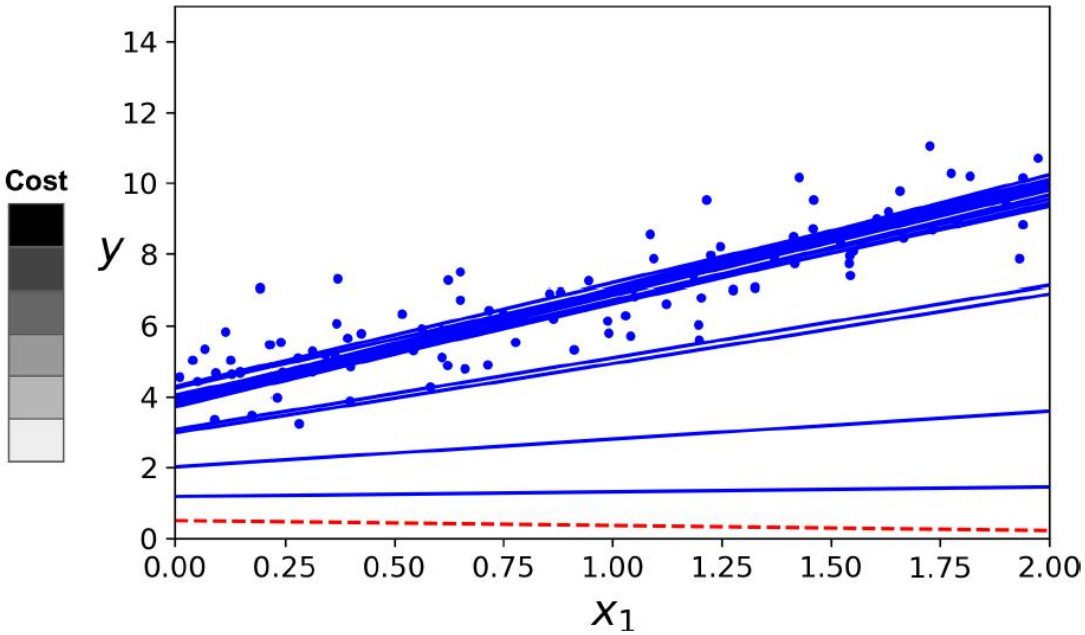


Figure 3 : Stochastic Gradient Descent first 20 steps

Mini-batch gradient descent

- It is quite simple to understand once you know Batch and Stochastic Gradient Descent: at each step, instead of computing the gradients based on the full training set (as in Batch GD) or based on just one instance (as in Stochastic GD), Minibatch GD computes the gradients on small random sets of instances called *minibatches*.
- The algorithm's progress in parameter space is less irregular than with SGD, especially with fairly large mini-batches.
- As a result, Mini-batch GD will end up walking around a bit closer to the minimum than SGD.

Mini-batch gradient descent

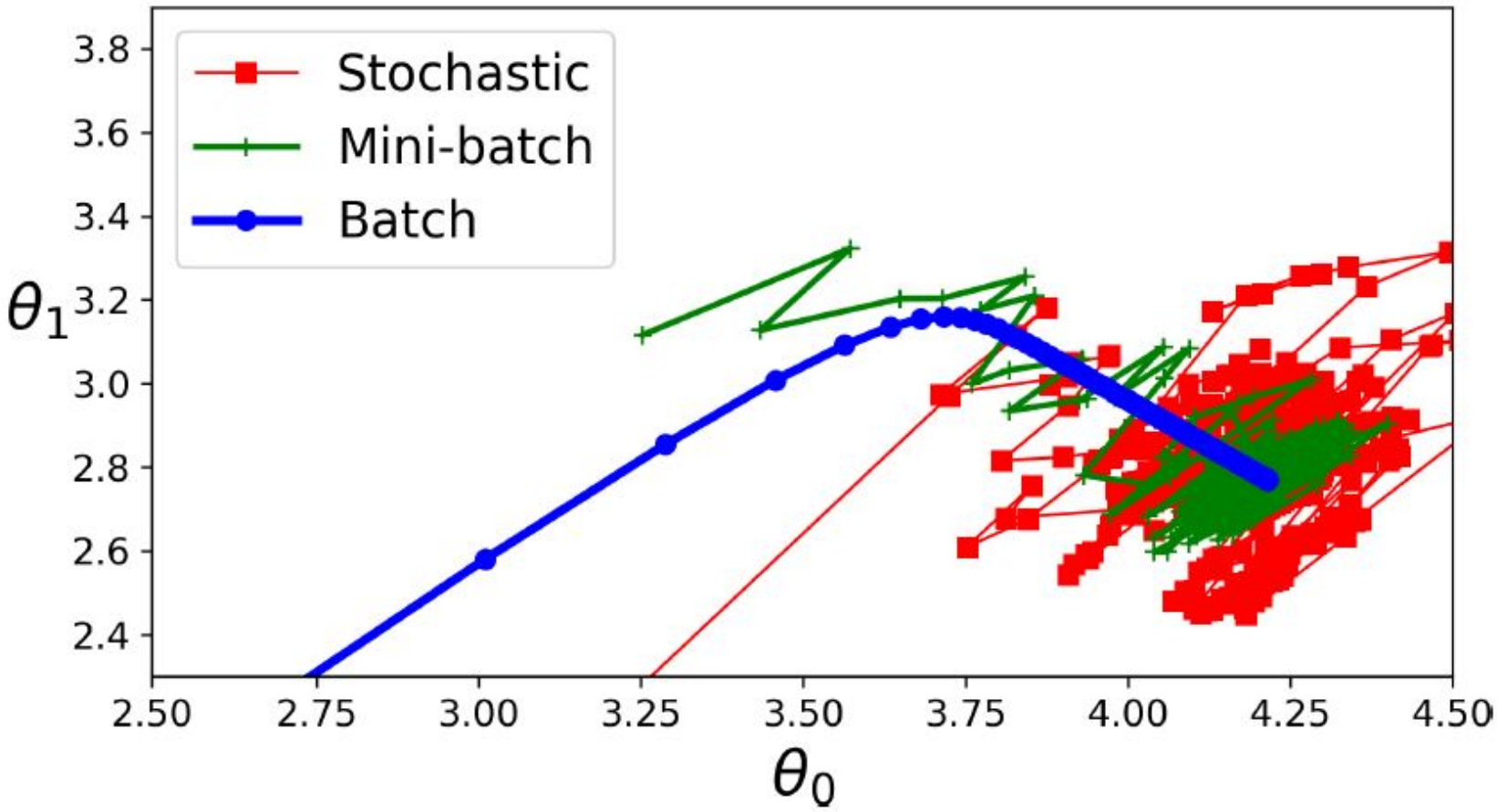


Figure 4 : Gradient Descent paths in parameter space