CS 60050
Machine Learning

Linear Regression

Some slides taken from course materials of Andrew Ng

Dataset of living area and price of houses in a city

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

This is a training set.

How can we learn to predict the prices of houses of other sizes in the city, as a function of their living area?

Dataset of living area and price of houses in a city

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

Example of supervised learning problem.

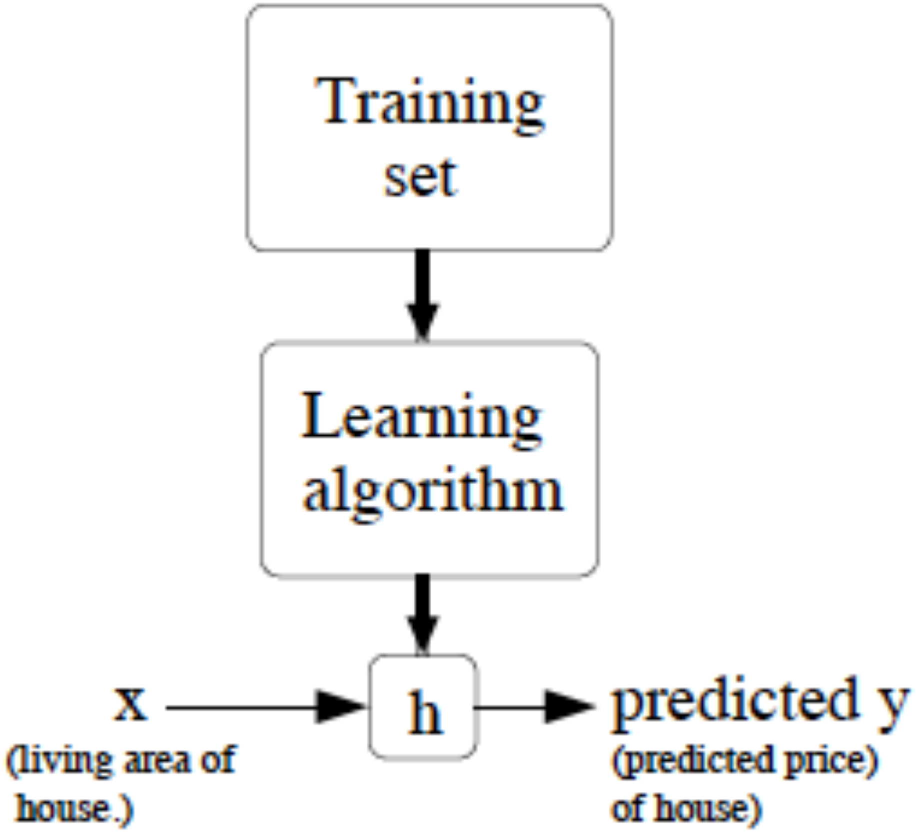
When the target variable we are trying to predict is continuous, regression problem.

Dataset of living area and price of houses in a city

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

m = number of **training examples**
x's = input variables / features
y's = output variables / "target" variables
(x,y) - single training example
(x^i, y^j) - specific example (i^{th} training example)
i is an index to training set

How to use the training set?



Learn a function h(x), so that h(x) is a good predictor for the corresponding value of y

h: hypothesis function

How to represent hypothesis h?

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

θ_i are **parameters**
- θ_0 is zero condition
- θ_1 is gradient

θ: vector of all the parameters

We assume y is a linear function of x
Univariate linear regression
How to learn the values of the parameters?

Digression:
Multivariate linear regression

Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
$\vdots$	$\vdots$	$\vdots$

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

How to represent hypothesis h?

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

θ_i are **parameters**

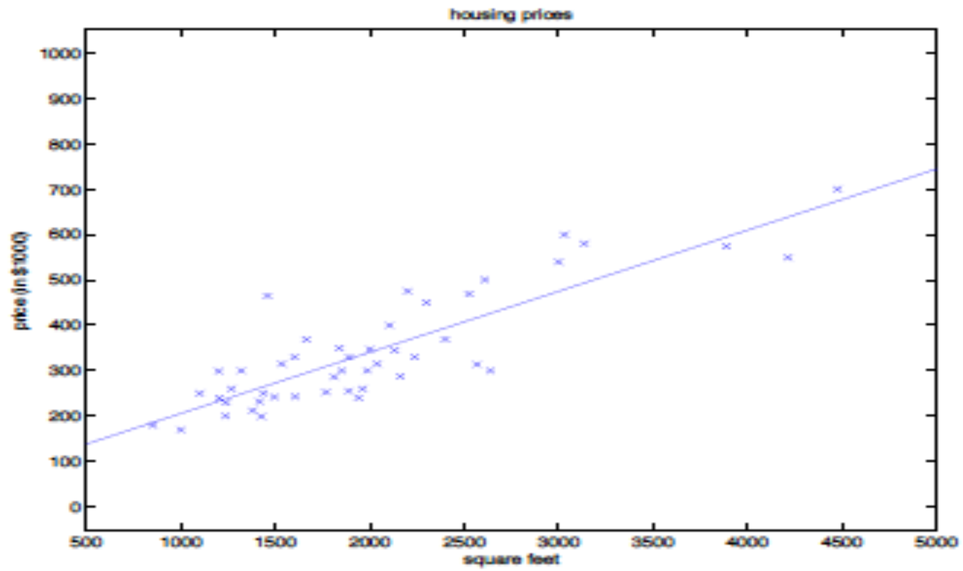
- θ_0 is zero condition
- θ_1 is gradient

We assume y is a linear function of x

Univariate linear regression

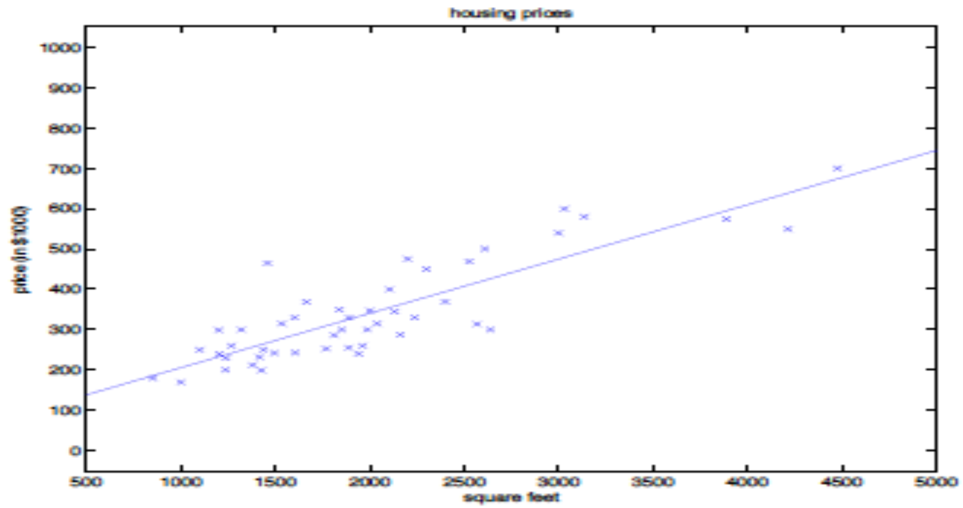
How to learn the values of the parameters θ_i?

Intuition of hypothesis function



- We are attempting to fit a straight line to the data in the training set
- Values of the parameters decide the equation of the straight line
- Which is the best straight line to fit the data?

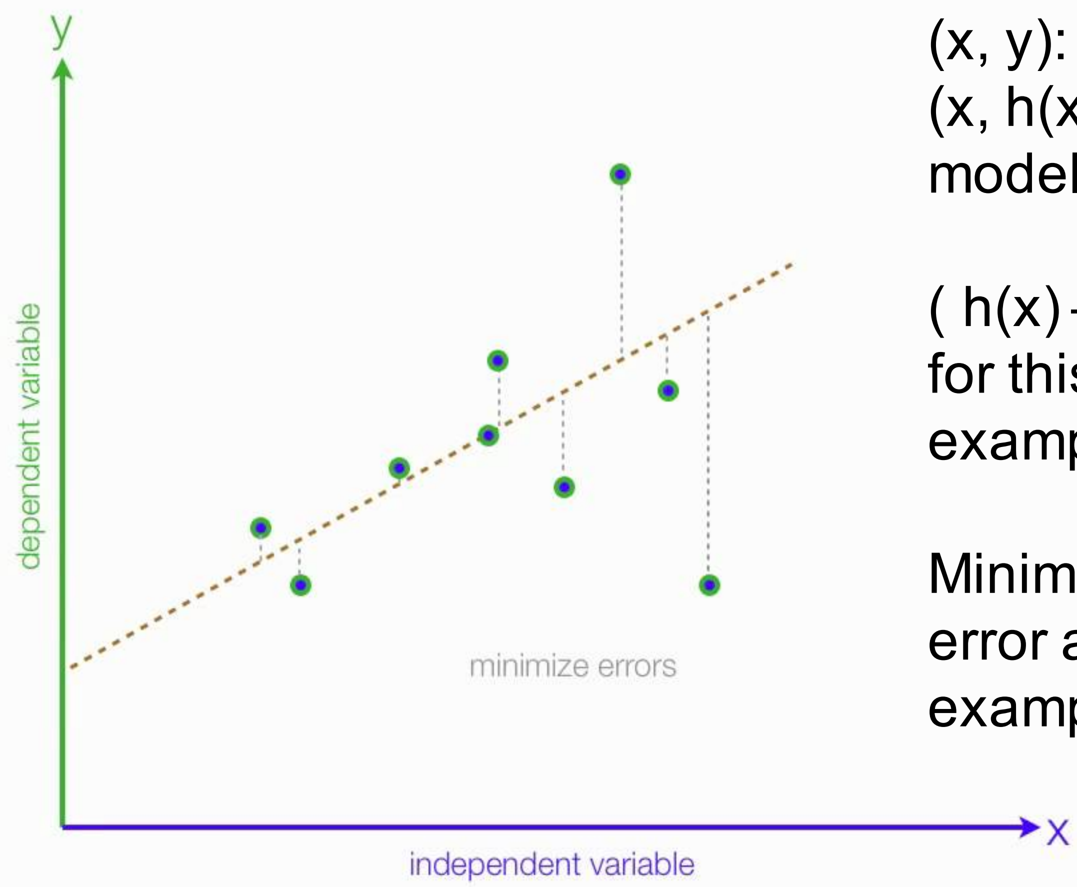
Intuition of hypothesis function



- Which is the best straight line to fit the data?
- How to learn the values of the parameters θ_i?

- Choose the parameters such that the prediction is close to the actual y-value for the training examples

How good is the prediction given by the straight line?



(x, y): a training example
(x, h(x)): prediction of the model

(h(x) – y): prediction error for this particular training example

Minimize the prediction error across all training examples

Cost function

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

- Measure of how close the predictions are to the actual y-values
- Average over all the m training instances

- Squared error cost function $J(\theta)$
- Choose parameters θ so that $J(\theta)$ is minimized

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

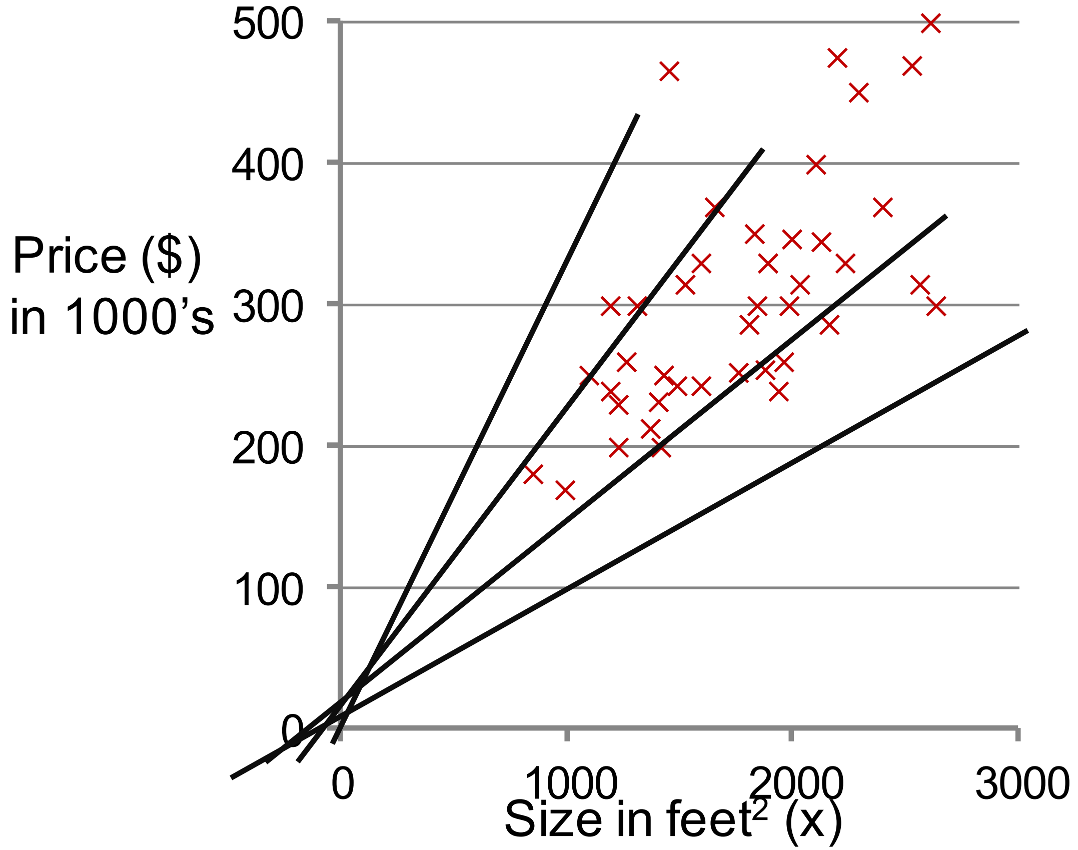
Parameters: θ_0, θ_1

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

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

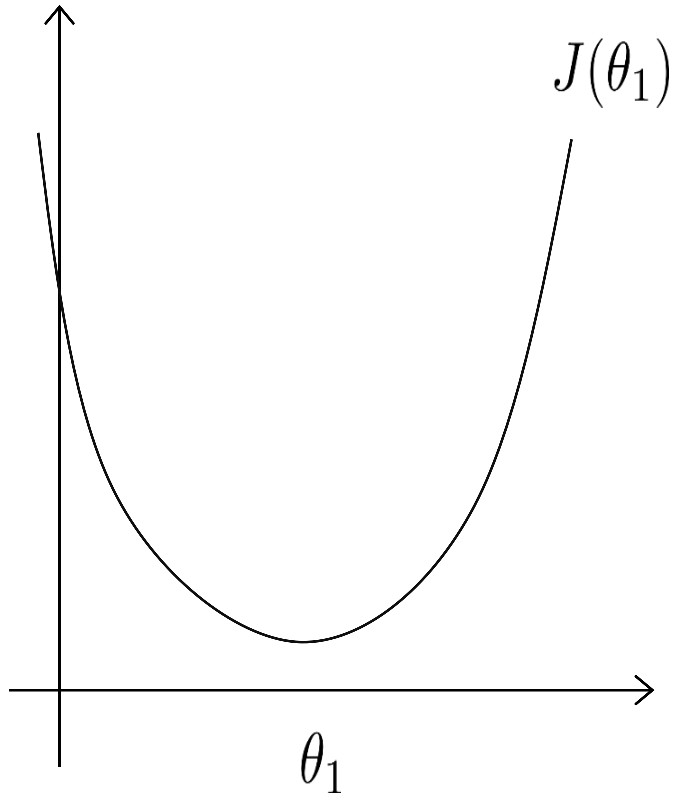
$$h_\theta(x)$$

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



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

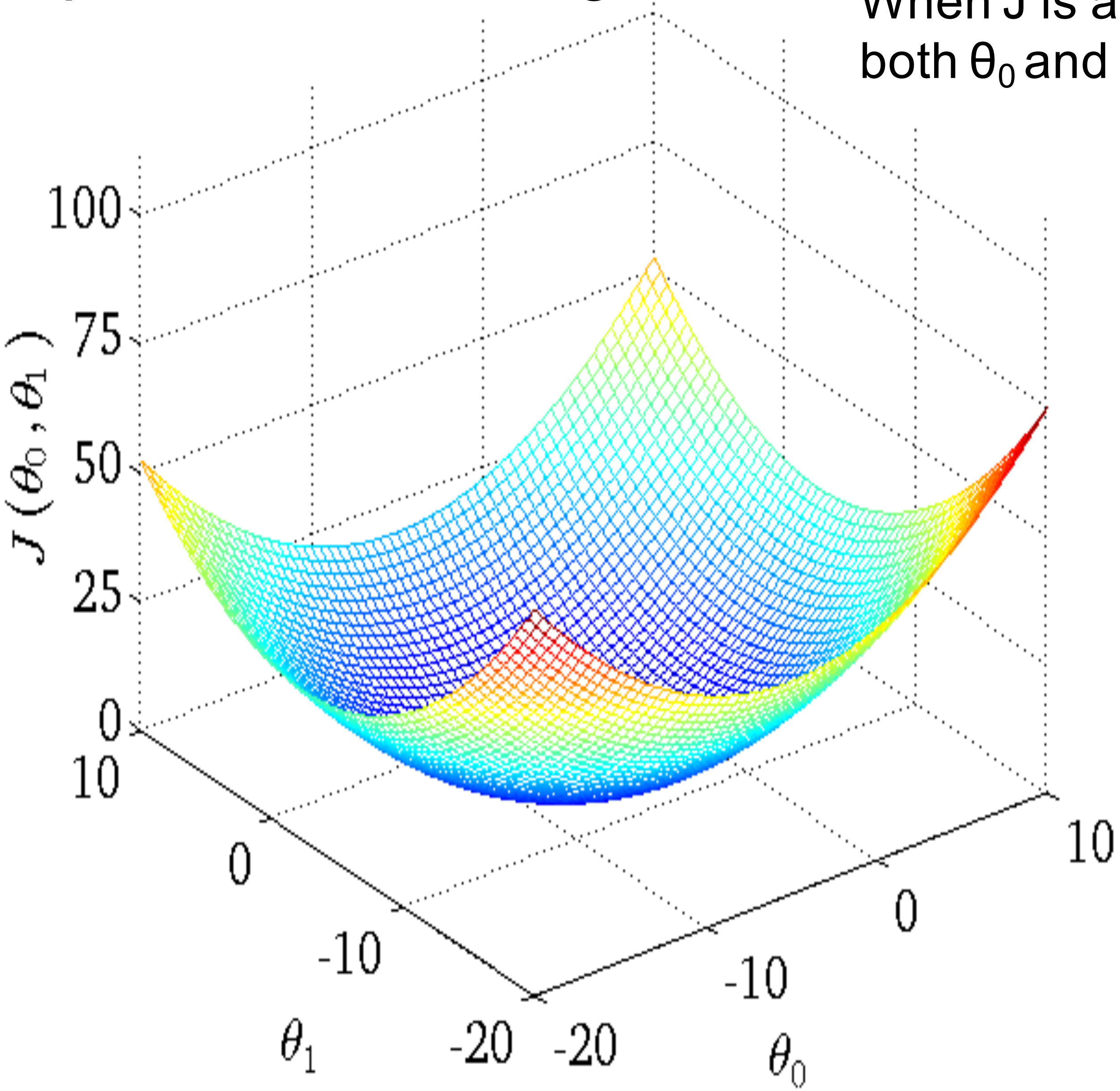
(function of the parameters θ_0, θ_1)



For simplicity, assume Θ_0 is a constant

Contour plot or Contour figure

When J is a function of both θ_0 and θ_1



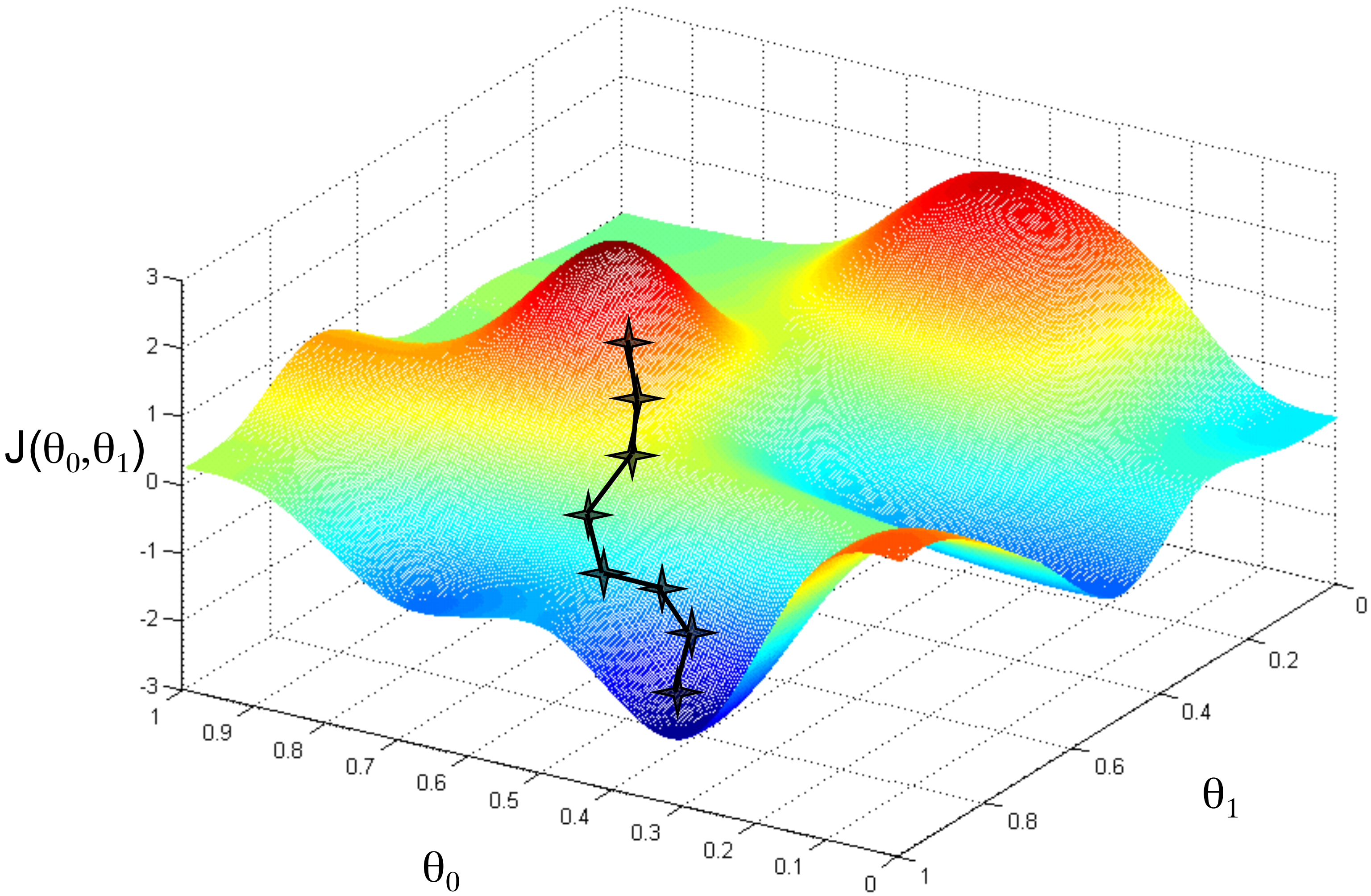
Minimizing a function

- For now, let us consider some arbitrary function (not necessarily a cost function)
- Analytical minimization not scalable to complex functions of hundreds of parameters
- Algorithm called gradient descent
  - Efficient and scalable to thousands of parameters
  - Used in many applications of minimizing functions

Have some function $J(\theta_0, \theta_1)$

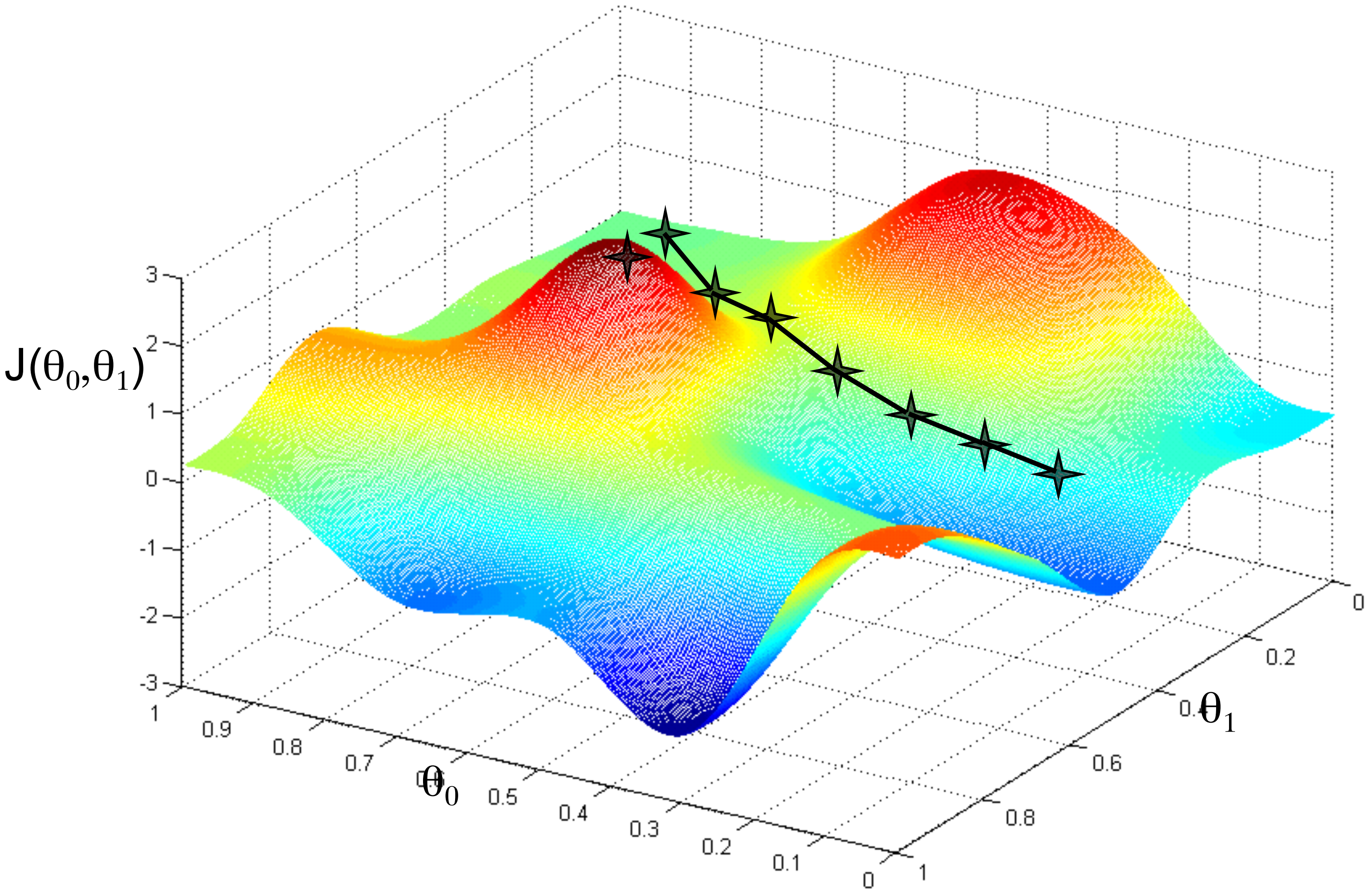
Want $\min_{\theta_0, \theta_1} J(\theta_0, \theta_1)$

- **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 a minimum
- Iterative method, similar to Newton-Raphson method for solving equations



$J(\theta_0,\theta_1)$
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
θ_0
0
0.2
0.4
0.6
0.8
1
θ_1

If the function has multiple local minima, where one starts can decide which minimum is reached



Gradient descent algorithm

$$\text{repeat until convergence } \{$$

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

$$\}$$

α is the learning rate – more on this later

Gradient descent algorithm

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

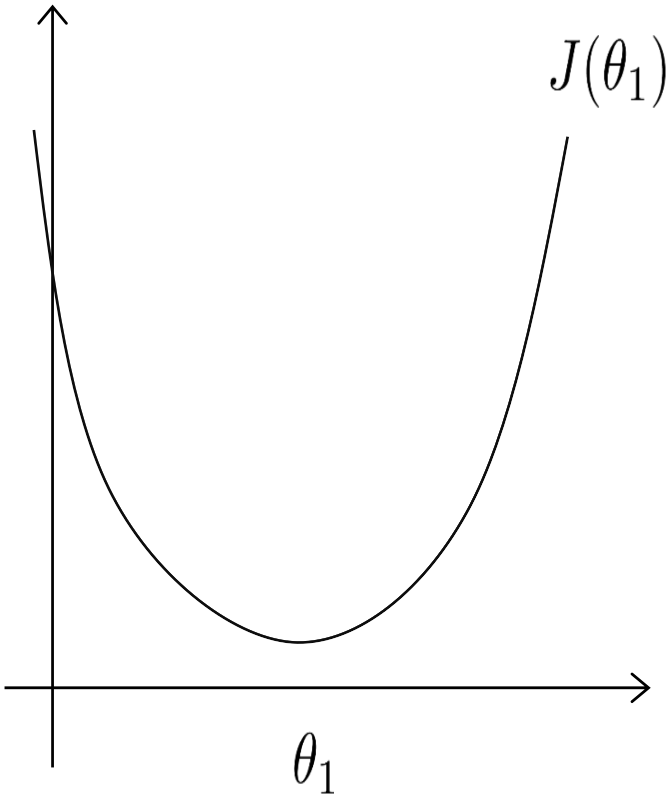
Correct: Simultaneous update

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

Incorrect:

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

For simplicity, let us first consider a function of a single variable



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

If the derivative is positive, reduce value of θ_1

If the derivative is negative, increase value of θ_1

The learning rate

- Do we need to change learning rate over time?
  - No, Gradient descent can converge to a local minimum, even with the learning rate α fixed
  - Step size adjusted automatically
- But, value needs to be chosen judiciously
  - If α is too small, gradient descent can be slow to converge
  - If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.

Gradient descent for univariate 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 for univariate linear regression

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

“Batch” Gradient Descent

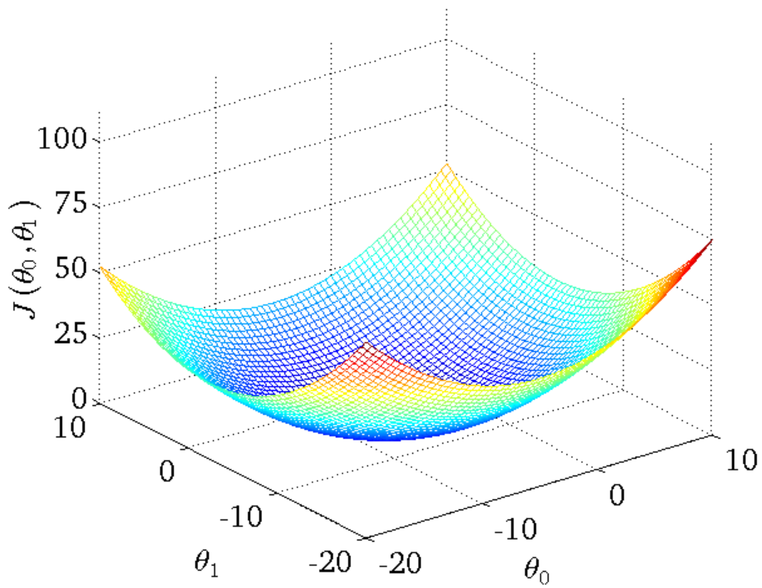
“Batch”: Each step of gradient descent uses all the training examples.

There are other variations like “stochastic gradient descent” (used in learning over huge datasets)

What about multiple local minima?

- The cost function in linear regression is always a convex function – always has a single global minimum

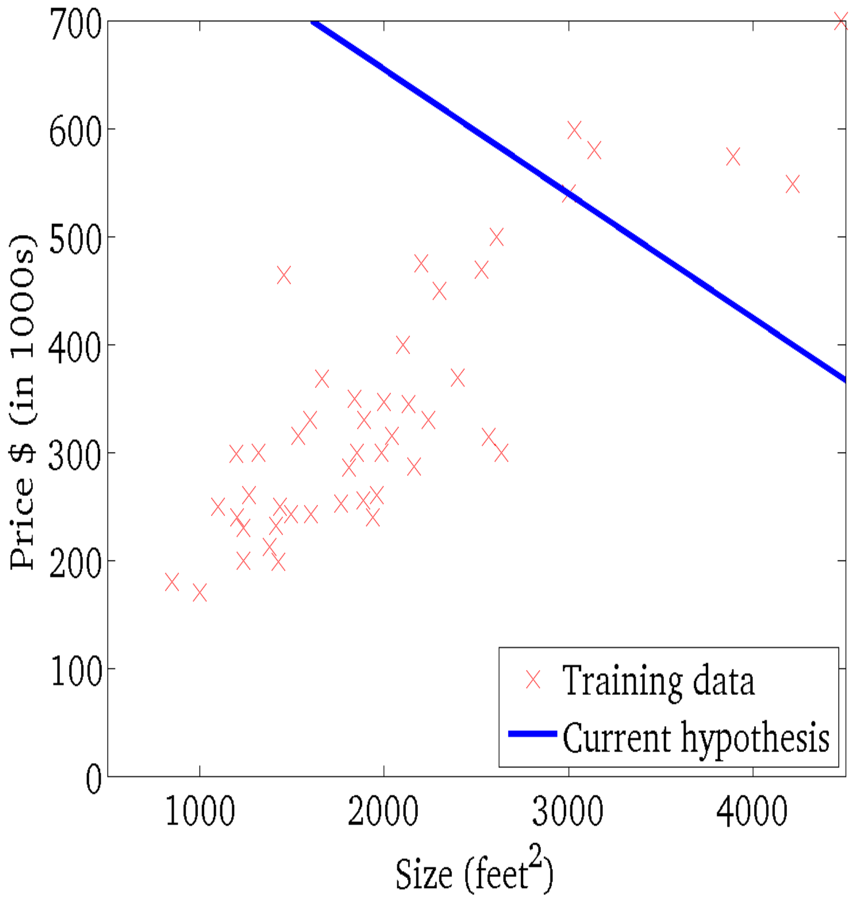
- So, gradient descent will always converge



Gradient descent in action

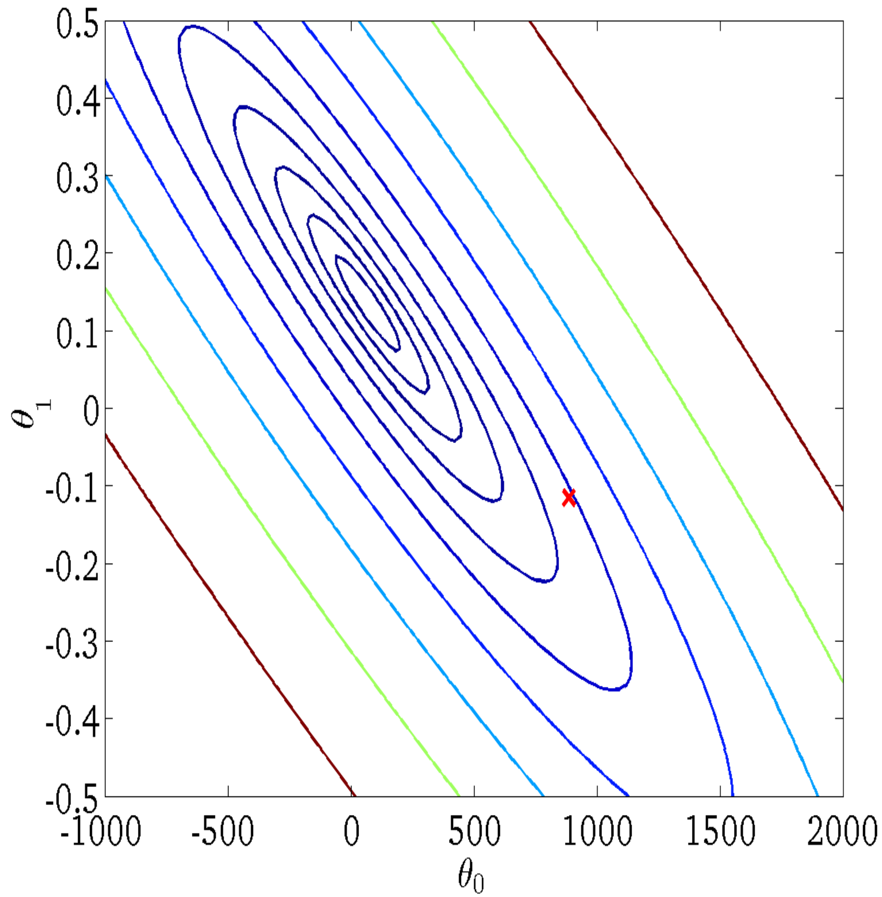
$$h_\theta(x)$$

(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)

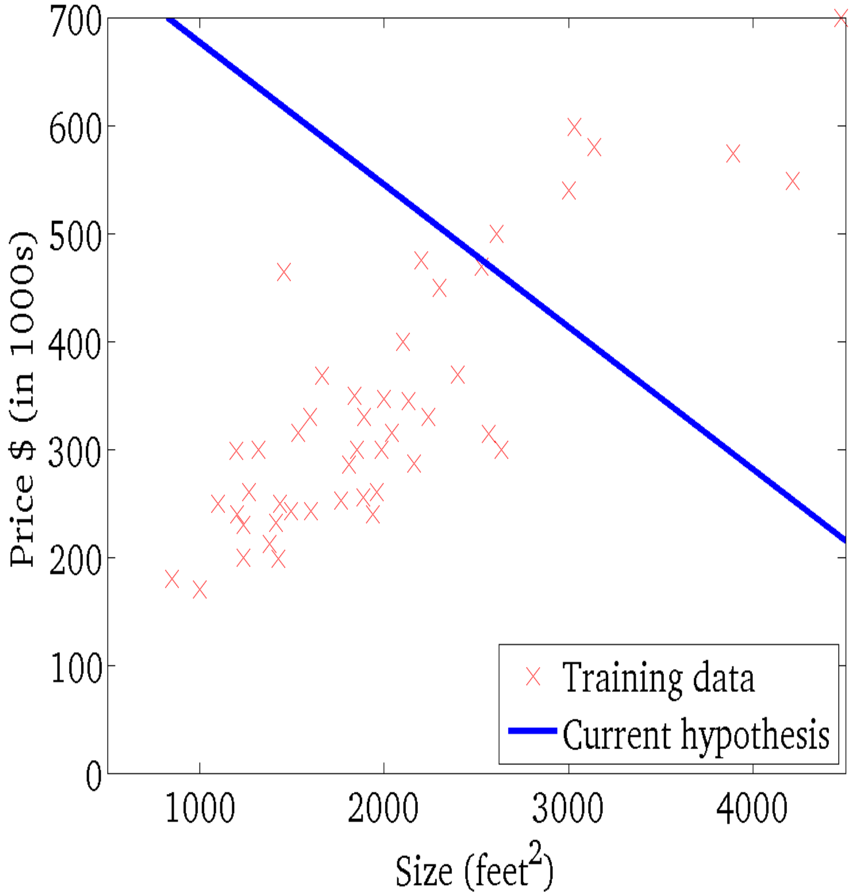


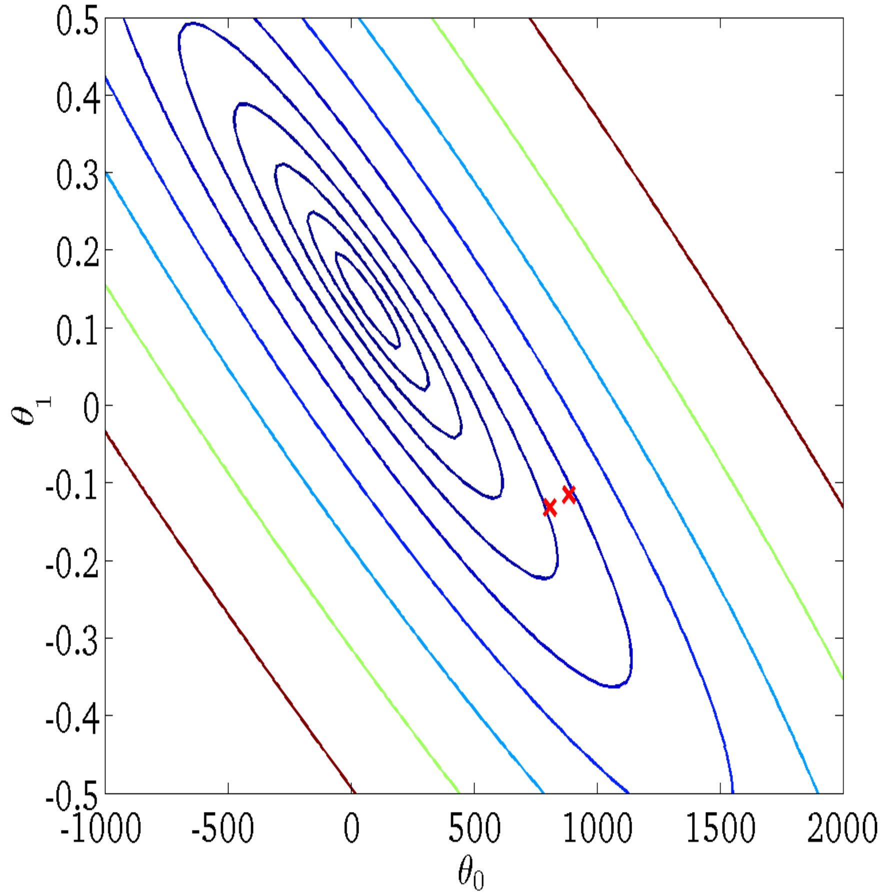
$h_\theta(x)$

(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)



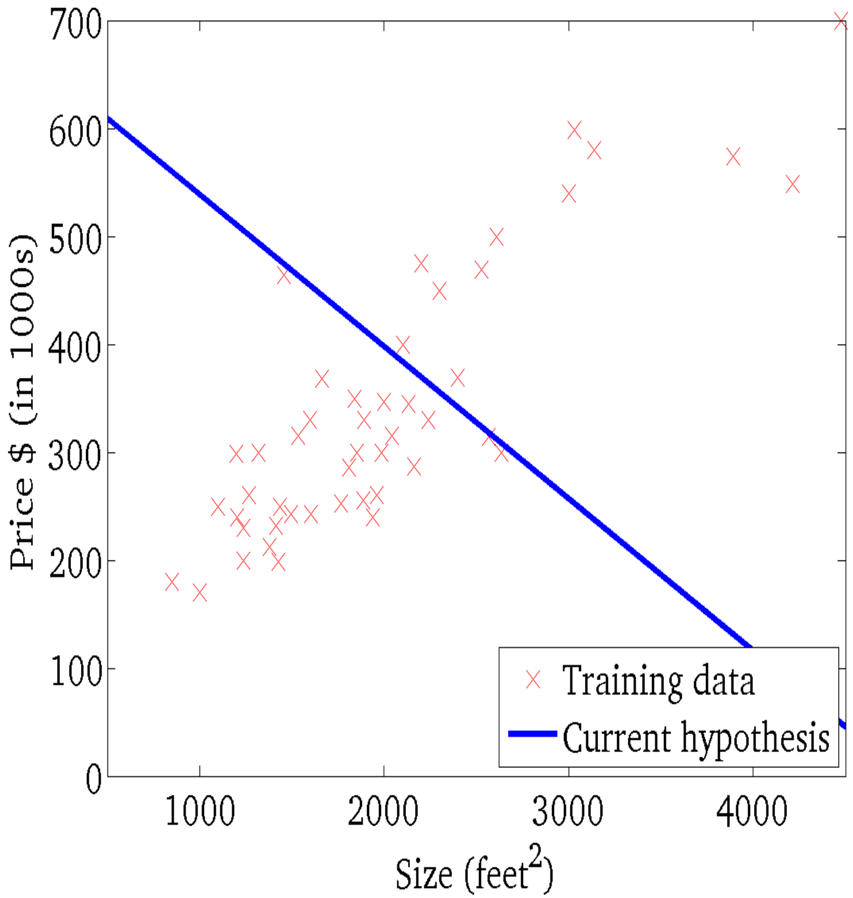


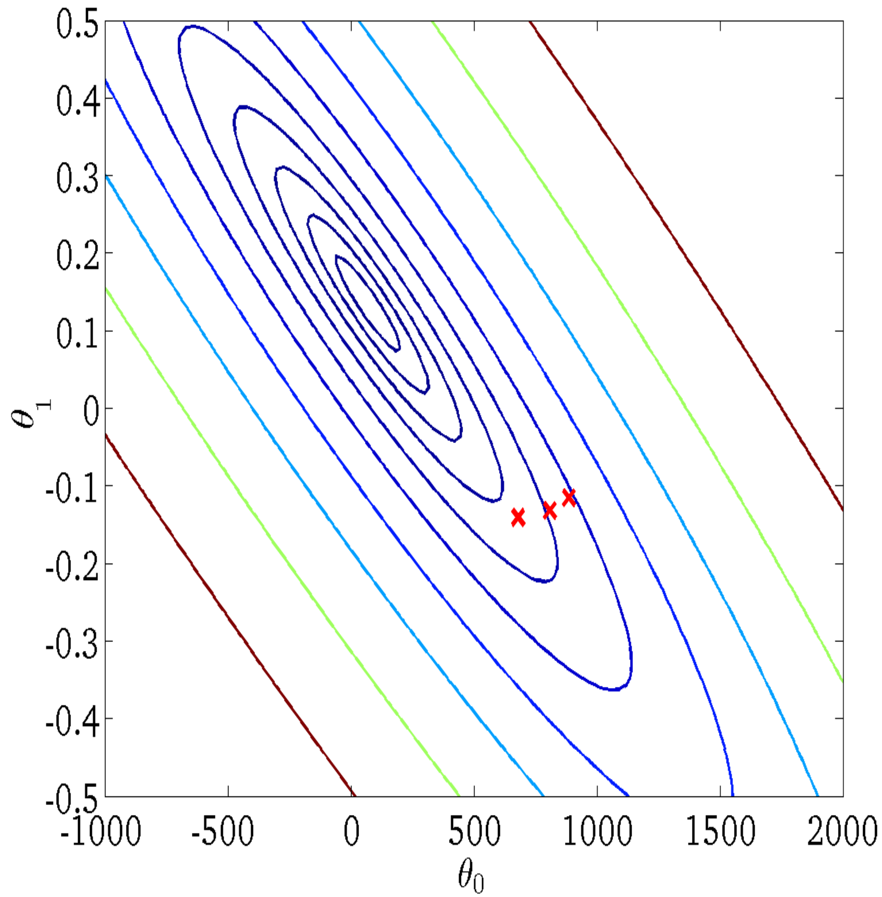
$h_\theta(x)$

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

$J(\theta_0, \theta_1)$

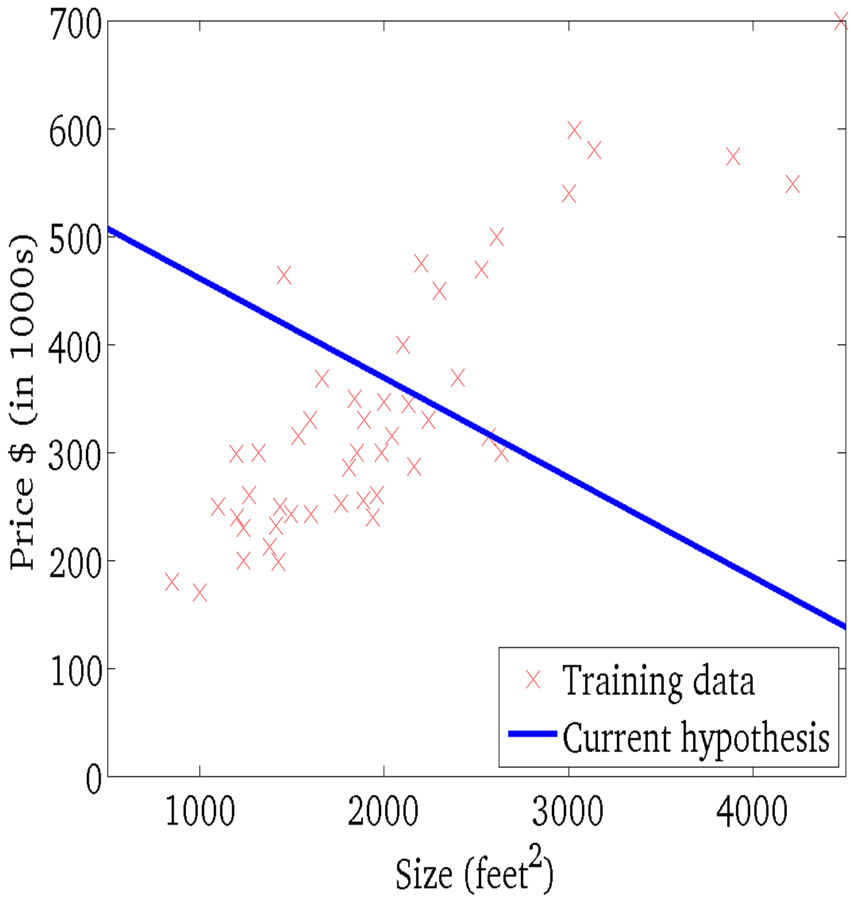
(function of the parameters θ_0, θ_1)





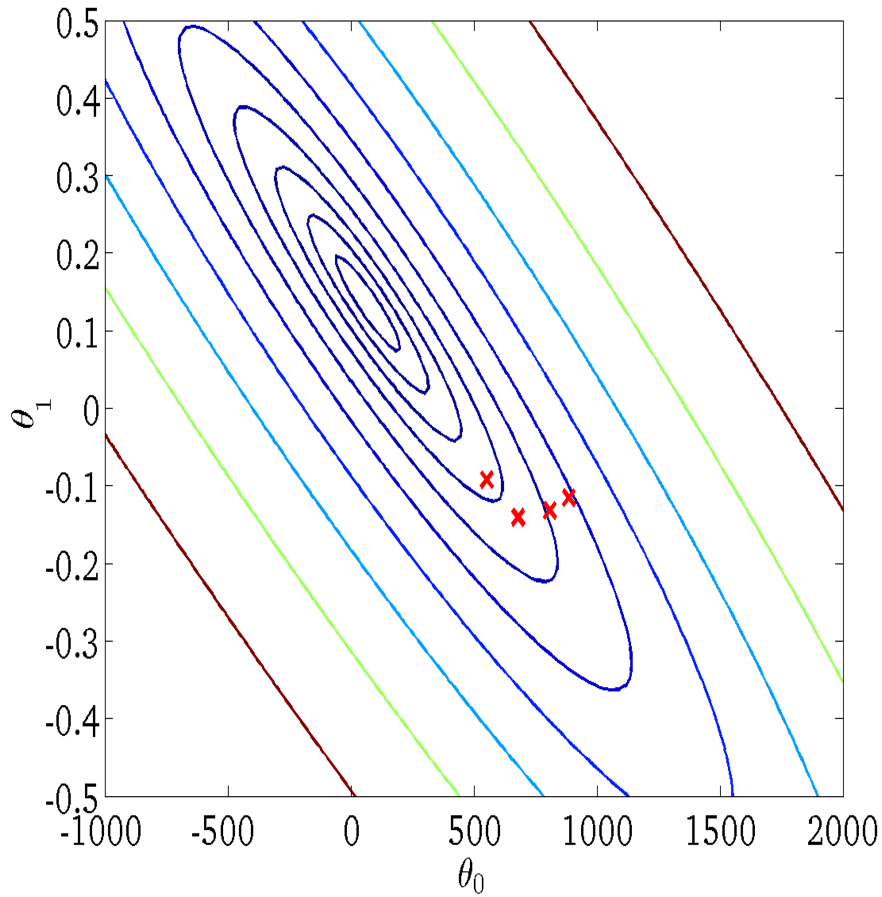
$$h_\theta(x)$$

(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)

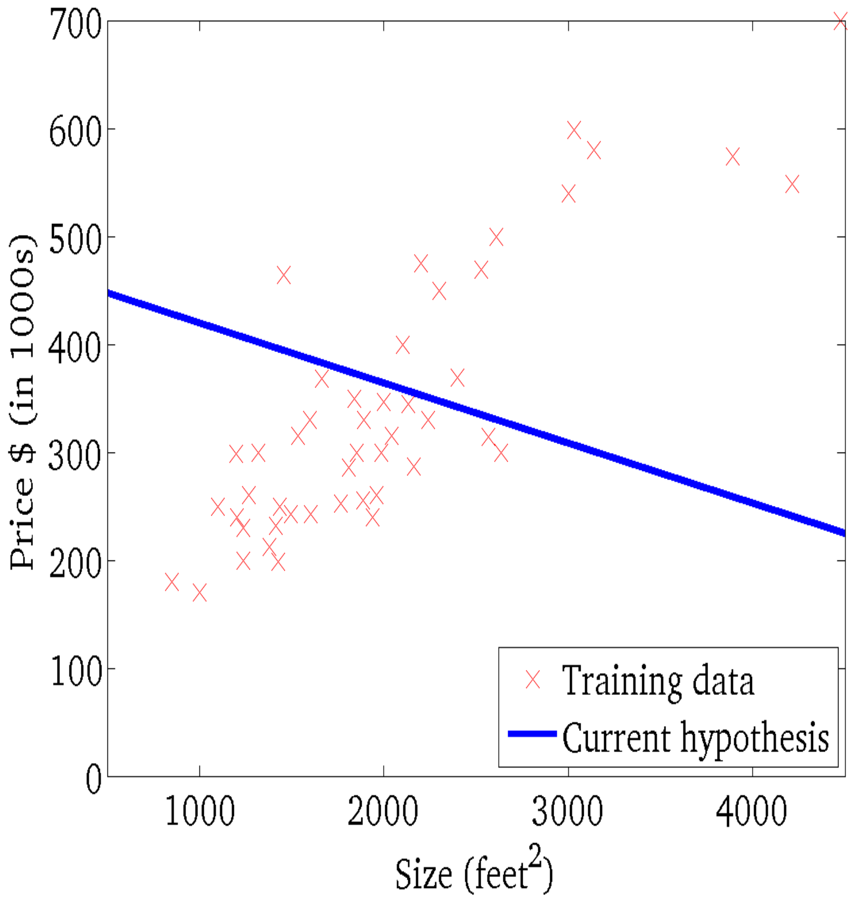


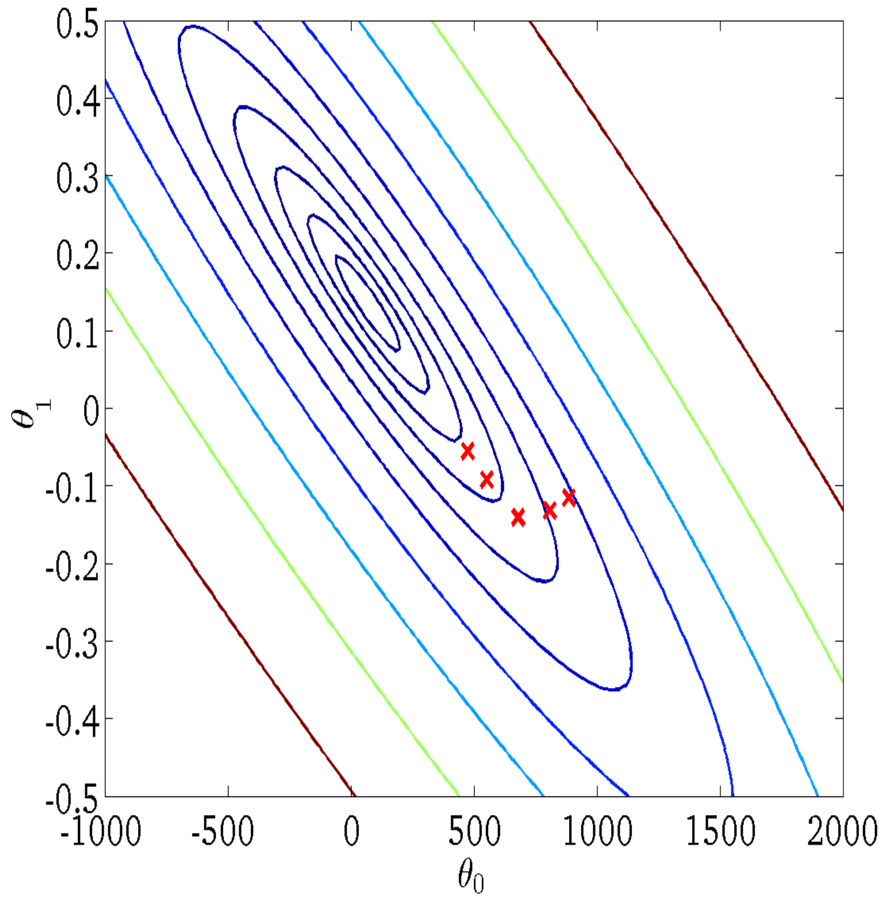
$h_\theta(x)$

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

$J(\theta_0, \theta_1)$

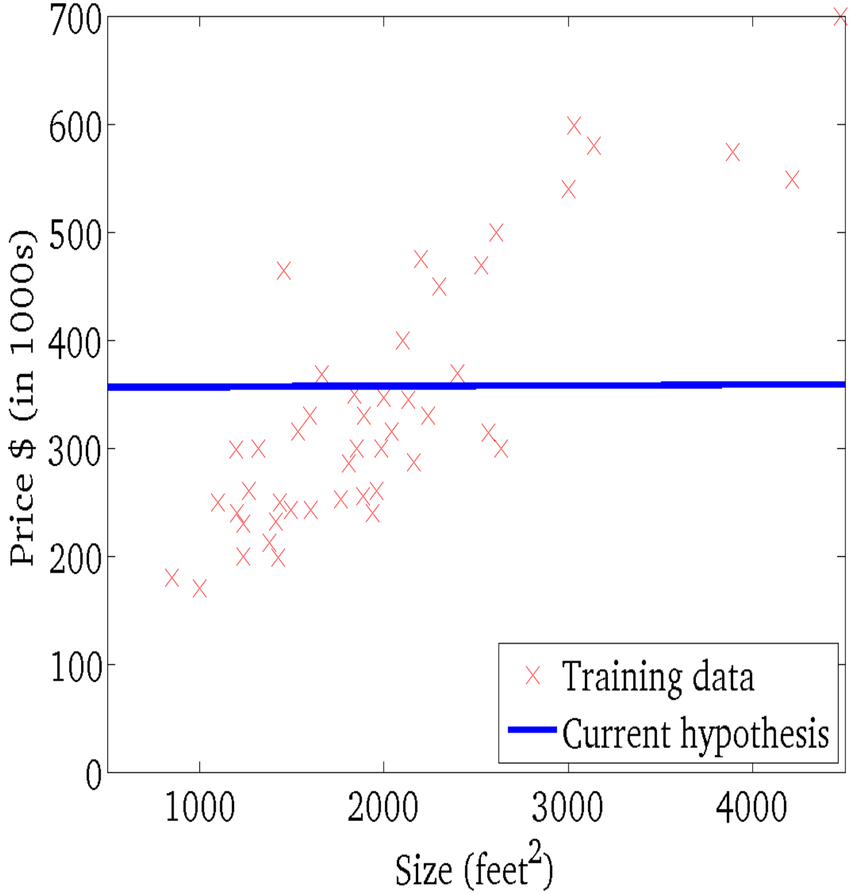
(function of the parameters θ_0, θ_1)





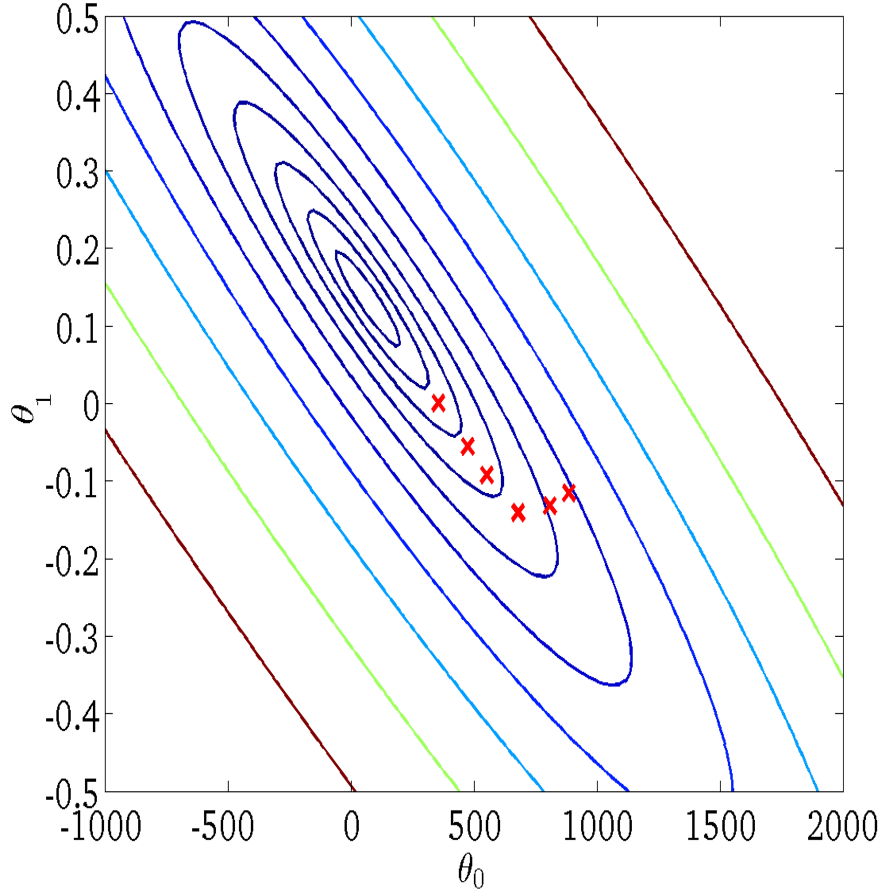
$$h_\theta(x)$$

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



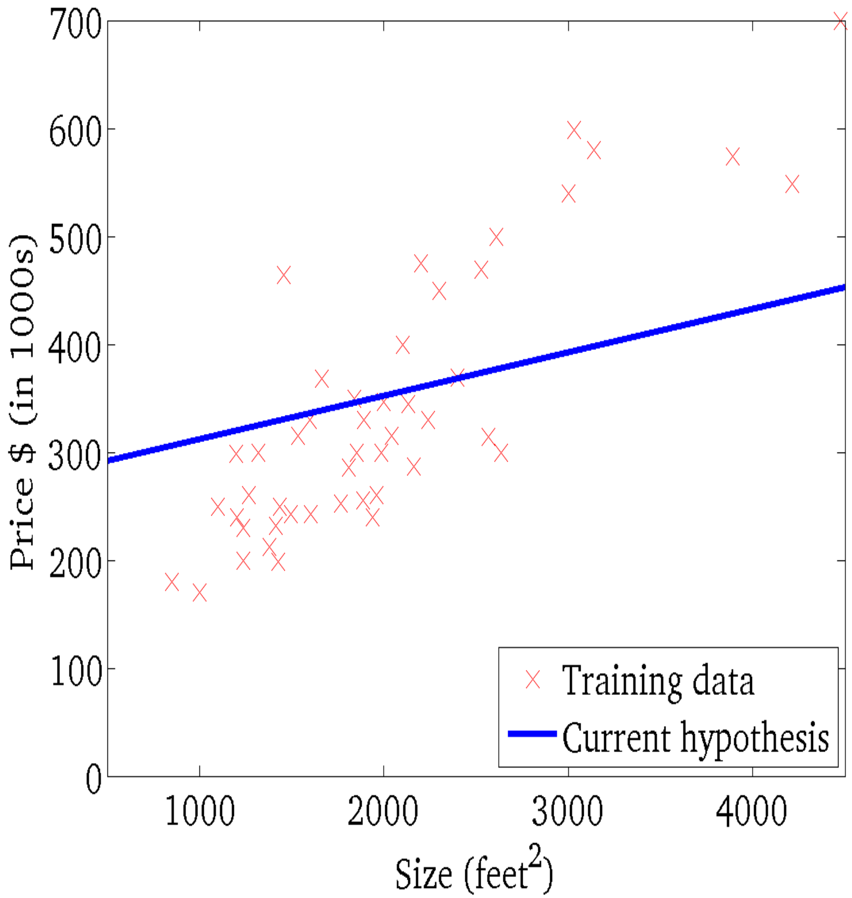
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



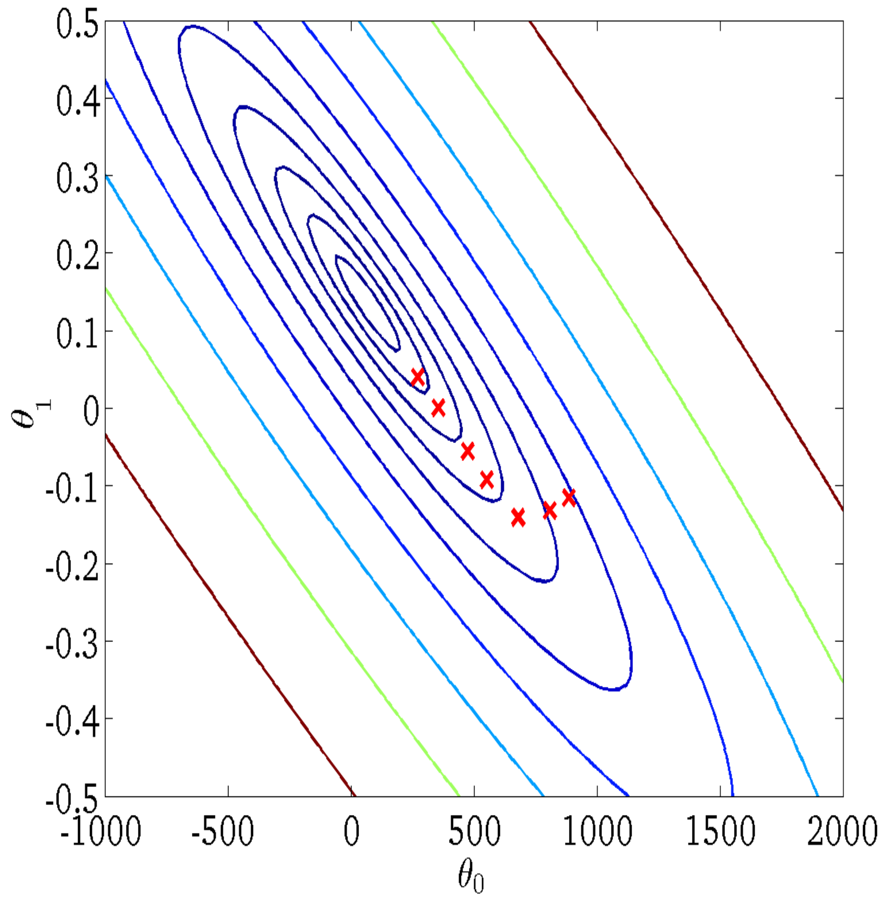
$$h_\theta(x)$$

(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)

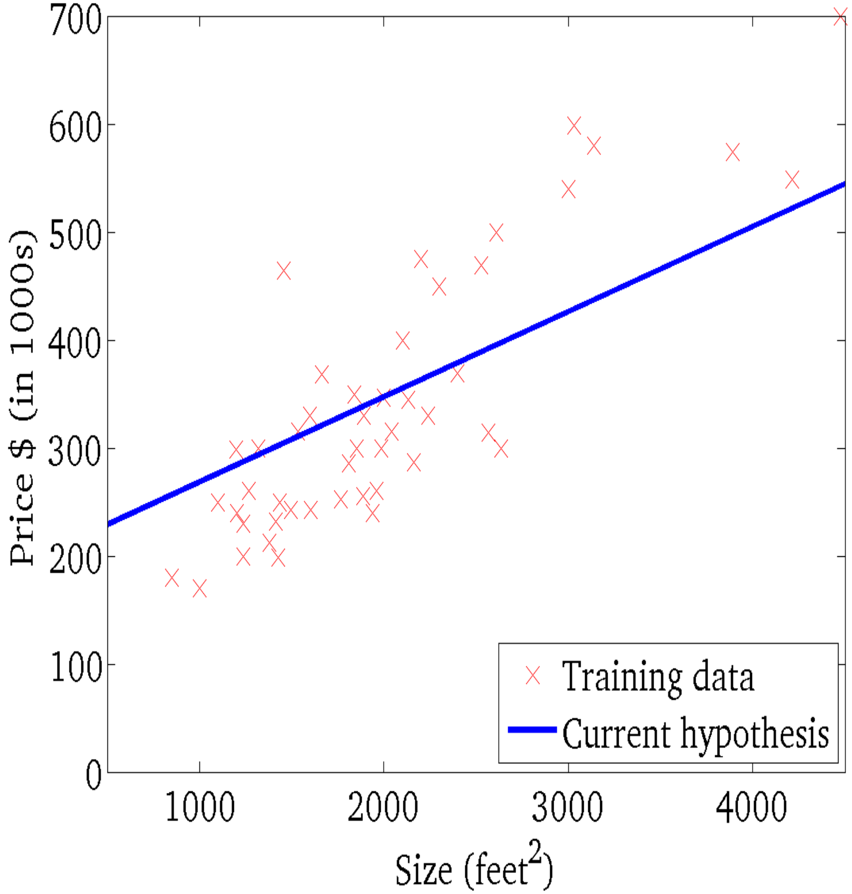


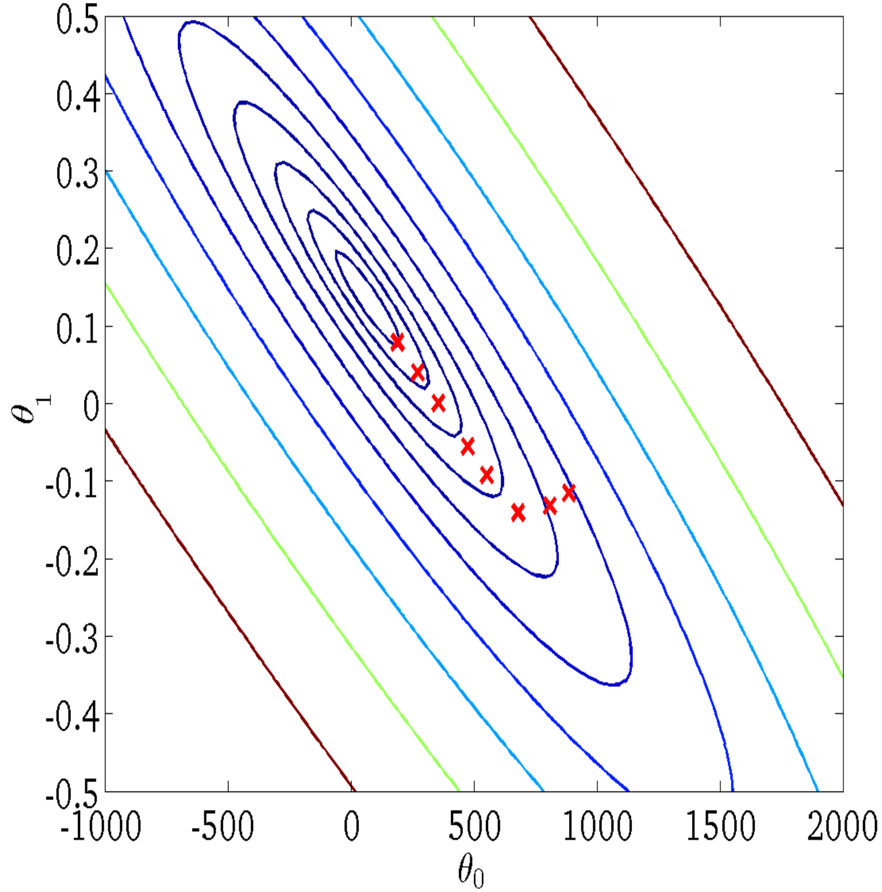
$h_\theta(x)$

(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)



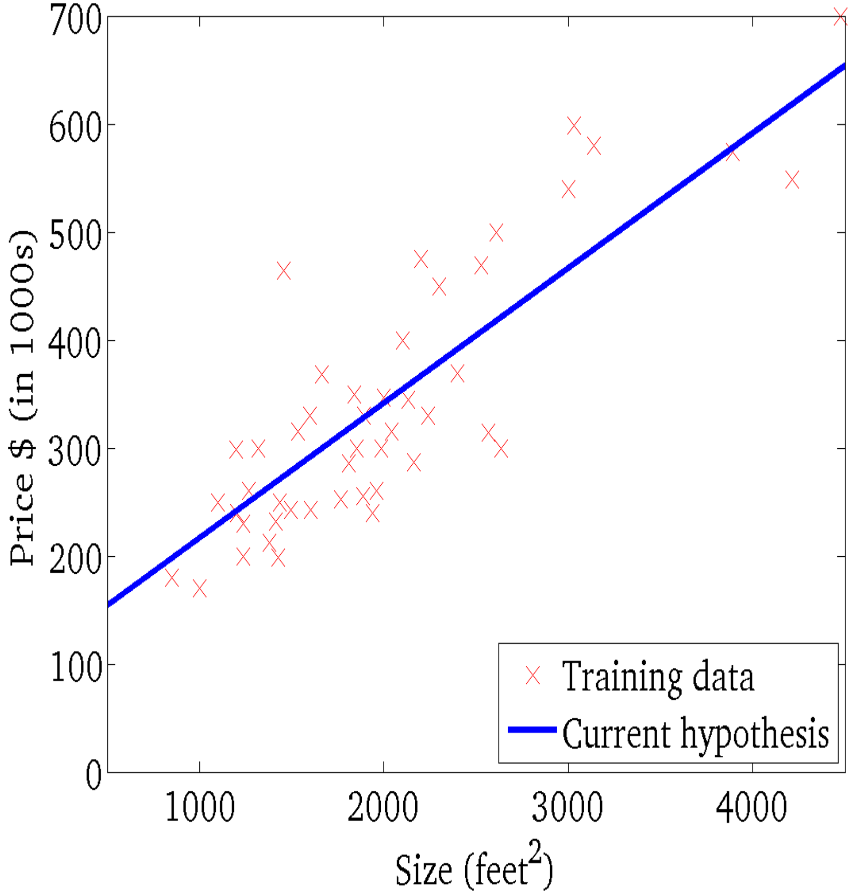


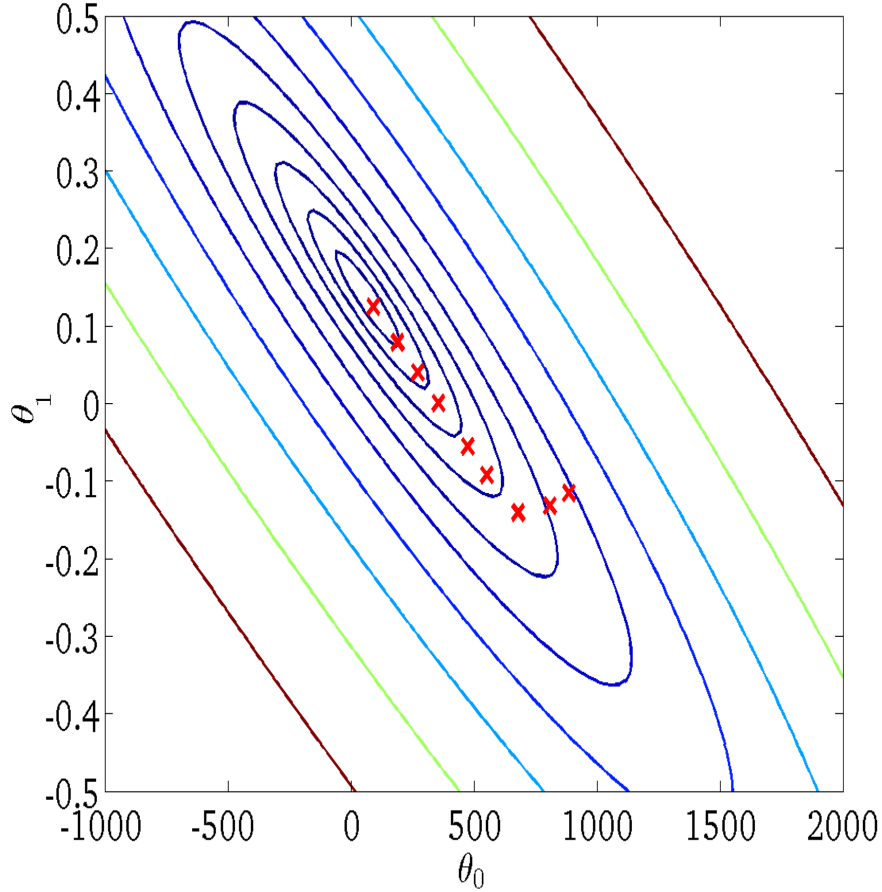
$$h_\theta(x)$$

(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)





Linear Regression for multiple variables

Multiple features (variables).

Size ($feet^2$)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

Multiple features (variables).

Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

Notation:

n = number of features. m = number of training examples

$x^{(i)}$ = input (features) of i^{th} training example.

$x_j^{(i)}$ = value of feature j in i^{th} training example.

Hypothesis:

For univariate linear regression:

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

For multi-variate linear regression:

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

For convenience of notation, define $x_0 = 1$.

Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Parameters: $\theta_0, \theta_1, \ldots, \theta_n$

Cost function:

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

Gradient descent:

Repeat $\{$

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

$\}$ (simultaneously update for every $j = 0, \ldots, n$)

Gradient Descent

Previously (n=1):

Repeat {

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

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

(simultaneously update θ_0, θ_1)

}

New algorithm $(n \geq 1)$:

Repeat {

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

simultaneously update θ_j for $j = 0, \ldots, n$

}

...

Gradient Descent

Previously (n=1):

Repeat $\{$

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

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

(simultaneously update θ_0, θ_1)

$\}$

New algorithm $(n \geq 1)$:

Repeat $\{$

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

simultaneously update θ_j for $j = 0, \ldots, n$

$\}$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_2^{(i)}$$

$\ldots$

Practical aspects of applying gradient descent

Feature Scaling
Idea: Make sure features are on a similar scale.

E.g. x_1 = size (0-2000 feet^2)
x_2 = number of bedrooms (1-5)

Normalization wrt the maximum value:

$$x_1 = \frac{\text{size (feet}^2\text{)}}{2000}$$

$$x_2 = \frac{\text{number of bedrooms}}{5}$$

Feature Scaling

Idea: Make sure features are on a similar scale.

E.g. x_1 = size (0-2000 feet^2)

x_2 = number of bedrooms (1-5)

Mean normalization:

Replace x_i with $x_i - \mu_i$ to make features have approximately zero mean (Do not apply to $x_0 = 1$).

Other types of normalization:

$$x_1 = \frac{size - 1000}{2000}$$

$$x_2 = \frac{\#bedrooms - 2}{5}$$

$$-0.5 \leq x_1 \leq 0.5, -0.5 \leq x_2 \leq 0.5$$

Is gradient descent working properly?

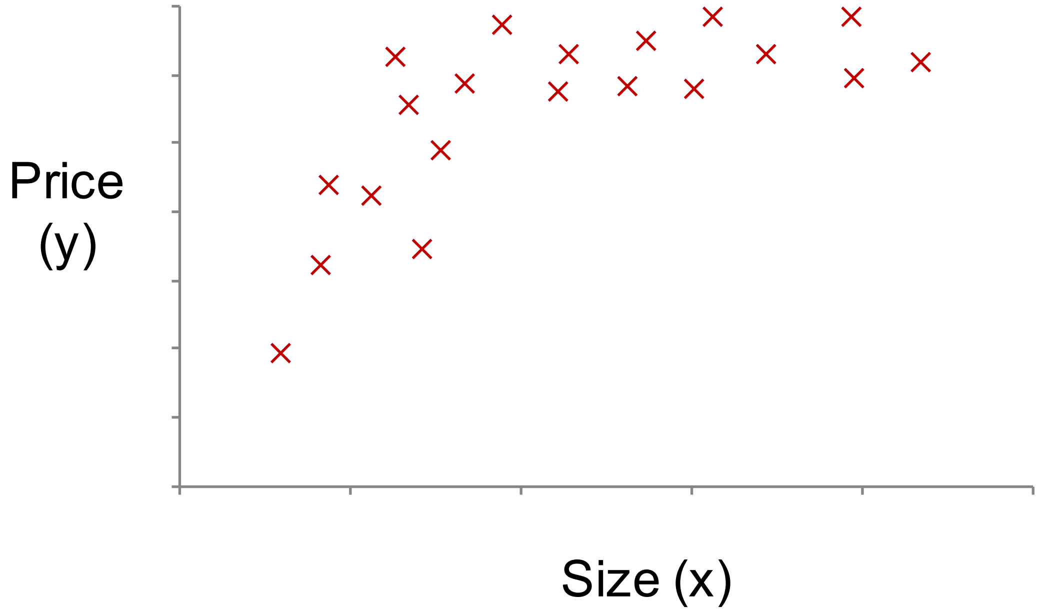
- Plot how $J(\theta)$ changes with every iteration of gradient descent
- For sufficiently small learning rate, $J(\theta)$ should decrease with every iteration
- If not, learning rate needs to be reduced
- However, too small learning rate means slow convergence

When to end gradient descent?

- Example convergence test:
- Declare convergence if $J(\theta)$ decreases by less than 0.001 in an iteration (assuming $J(\theta)$ is decreasing in every iteration)

Polynomial Regression for multiple variables

Choice of features



$$h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2(size)^2$$

$$h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2\sqrt{(size)}$$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$
$$= \theta_0 + \theta_1(size) + \theta_2(size)^2 + \theta_3(size)^3$$