SUPERVISED LEARNING

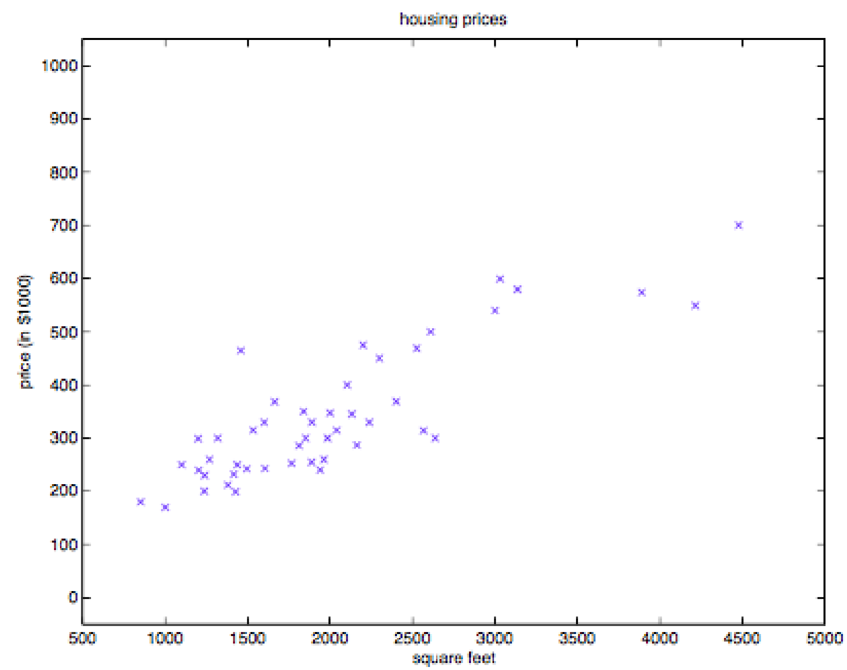
Supervised Learning

- **Example:** Given a dataset of living areas and prices of houses in a given area.

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

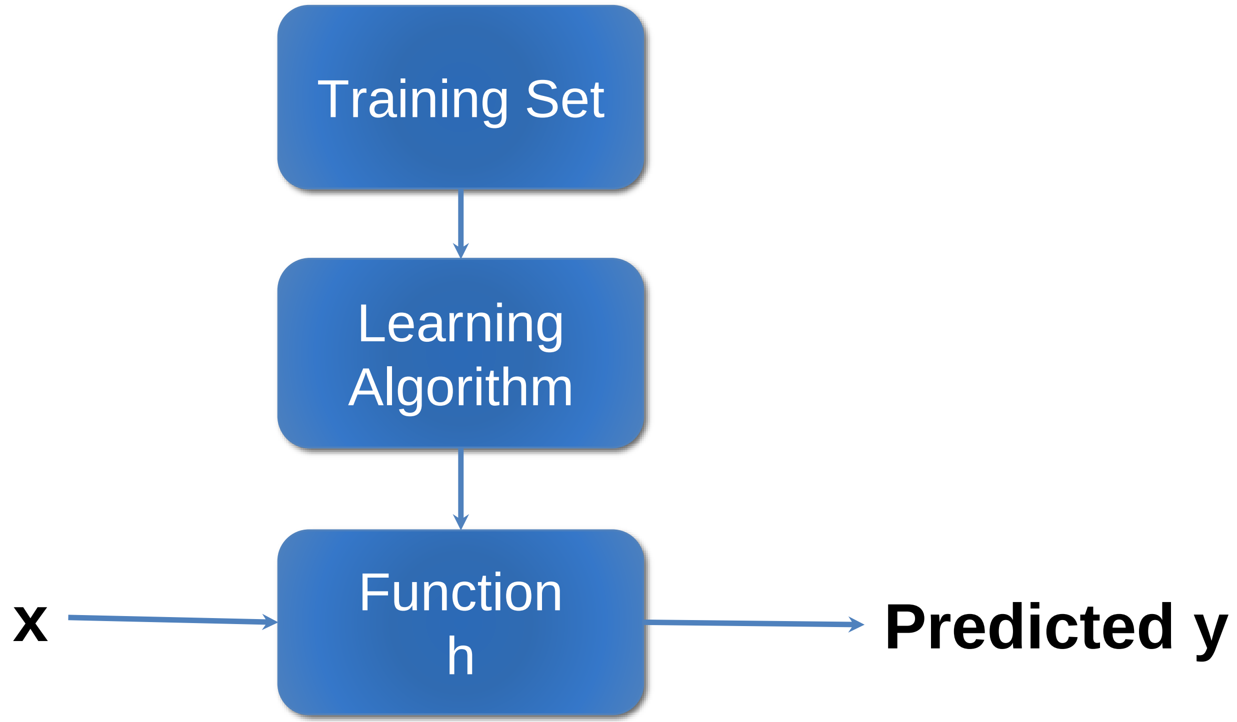
"input" variables or features => $x^{(i)}$

"output" variables or target => $y^{(i)}$



- How can we learn to predict the prices of other houses in that same area, as a function of the size of the living area.
  - i.e. Can we learn a function h, such that h(x) is a good predictor for the corresponding value of y.

Supervised Learning



When trying to predict a **continuous variable**, that's called a **regression problem.**

When trying to predict only small number of **discrete values**, we call it a **classification problem**.

Outline

- Regression
  - Linear regression

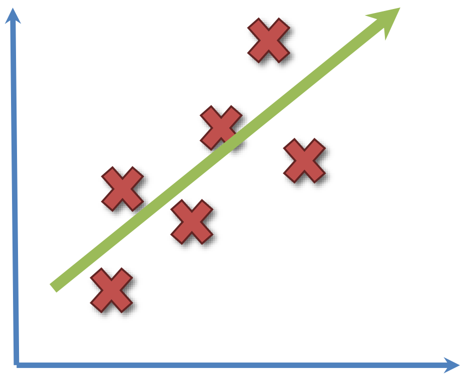
- Classification
  - Logistic Regression

Linear Regression

- **<u>Problem</u>**: Need to find a function **h**, that approximates the target variable **y** as a linear function of the features $\mathbf{x_i}$.

$$\mathbf{h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2}$$

  - Assume training data has two features $\mathbf{x_1}$ and $\mathbf{x_2}$ (or input variables).
  - The $\boldsymbol{\theta}_i$ are the parameters (or weights) of the linear function.



<u>Idea:</u> choose $\boldsymbol{\theta_0, \theta_1}$**, and** $\boldsymbol{\theta_2}$ so that h(x) is close to y for our training examples

Objective Function for Linear Regression

- **Problem**: Need to find a function h, that approximates the target variable y as a linear function of the features x_i .

$$\mathbf{h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2}$$

- So, given the training data, how to pick the parameters θ_i ?
  - Choose θ_i such that h(x) closely approximates y. i.e. choose θ_i that <u>minimize the cost function.</u>

m – number of training samples

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

J(**θ**) is called squared error cost function

Summary of the problem

- **Find a function:**

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

- **Parameters/Weights:** $\theta_0, \theta_1, \theta_2$

- **Cost Function:**

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

- **Goal:**

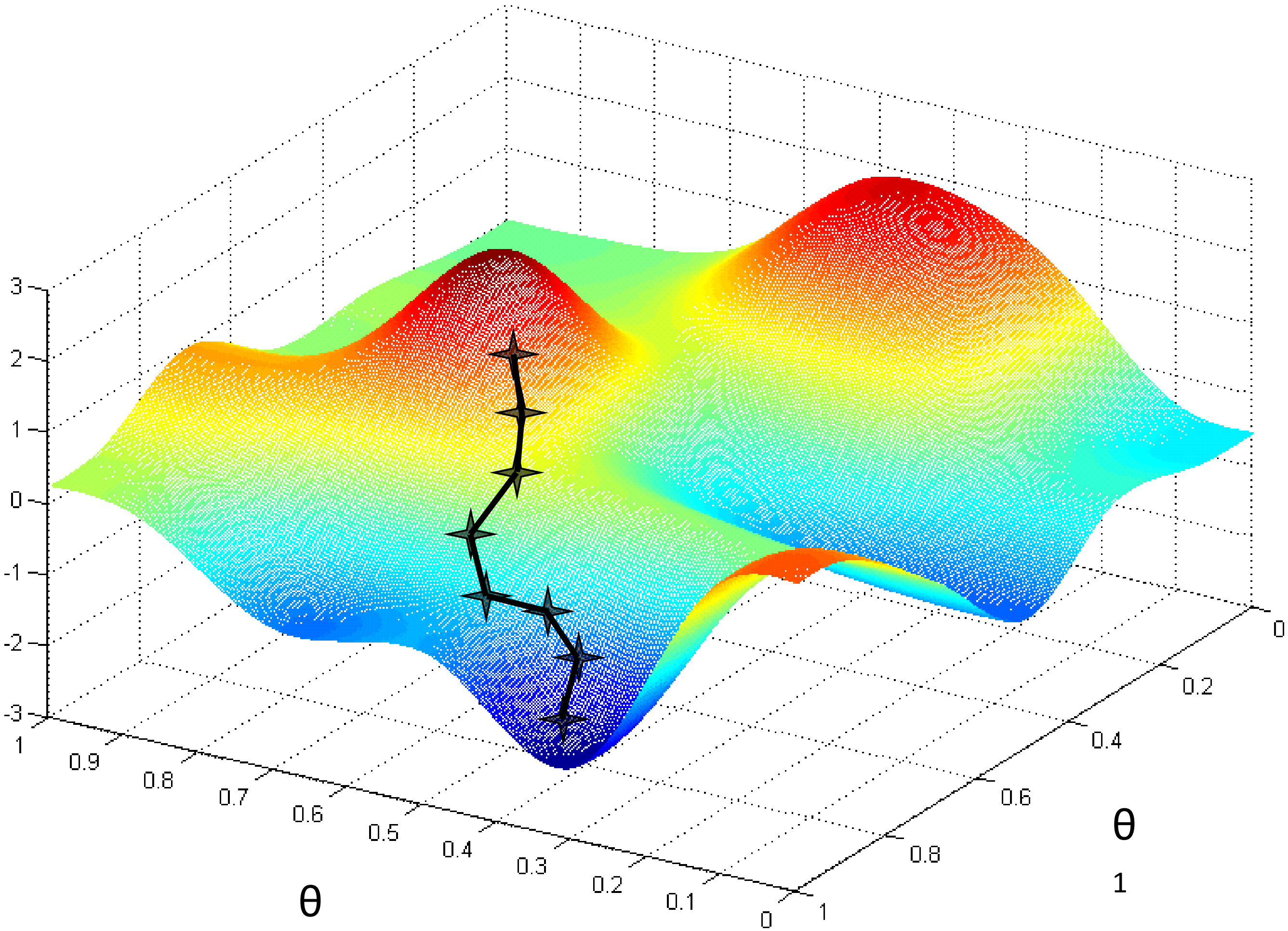
  $$\min_{\theta_0,\theta_1,\theta_2} J(\theta_0,\theta_1,\theta_2)$$

Gradient Descent Algorithm

- **Idea:** We want to choose θ that minimize $J(\theta)$, so we can start with an initial guess for θ and repeatedly change θ to make $J(\theta)$ smaller until we converge.

- We have some function $J(\theta_0, \theta_1)$ that we want to minimize.

- Outline:
  - Start with some value for θ_0, θ_1
  - Keep changing θ_0, θ_1 to reduce $J(\theta_0, \theta_1)$ until we end up at a minimum.

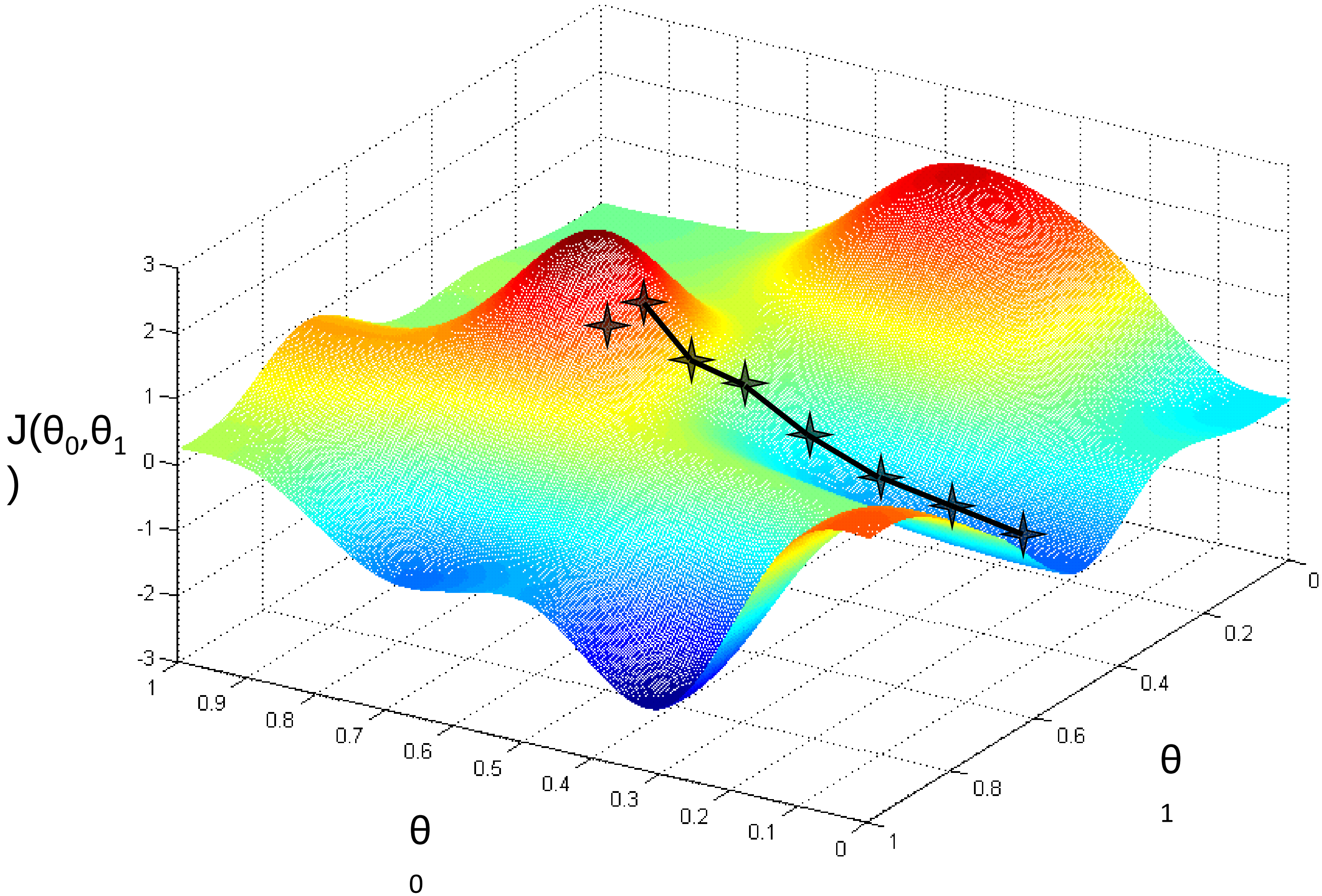


$J(\theta_0, \theta_1)$

Cost function

θ_0

θ_1


J(θ0,θ1)
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
0
0.2
0.4
0.6
0.8
1

Gradient Descent Algorithm

repeat until convergence {

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

}

Learning rate

Correct: Simultaneous update

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

Incorrect:

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

Gradient Descent Algorithm

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

<u>Learning Rate</u>
How large of a step to take as we update θ_j

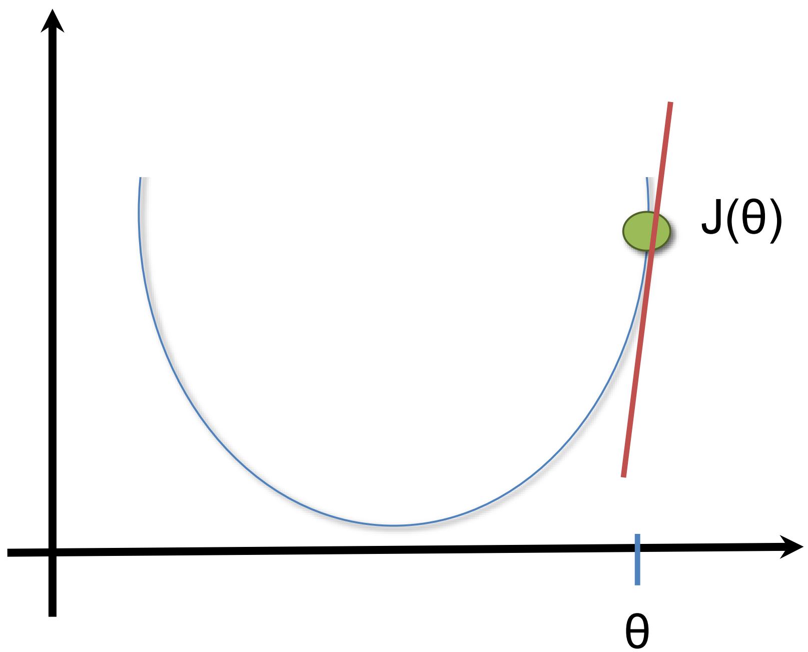
<u>Derivative</u>

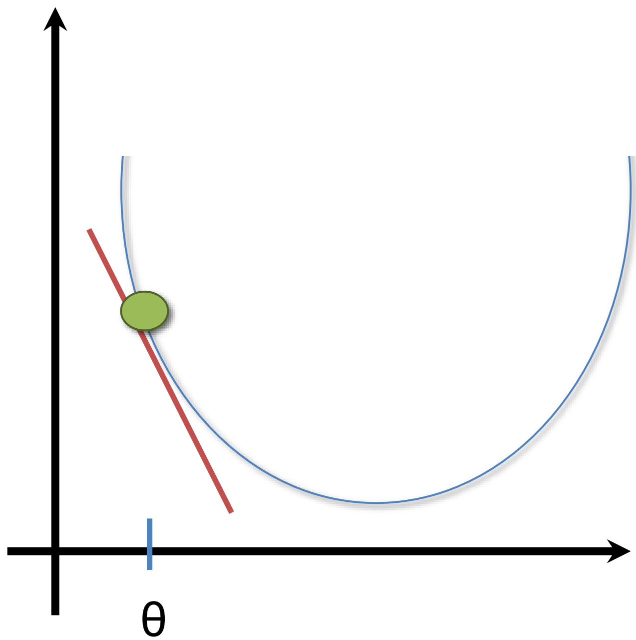
$$\theta_i = \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta_i)$$

$$\theta_i = \theta_i - \alpha (pos\#)$$

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

$$\theta_i = \theta_i - \alpha (neg\#)$$



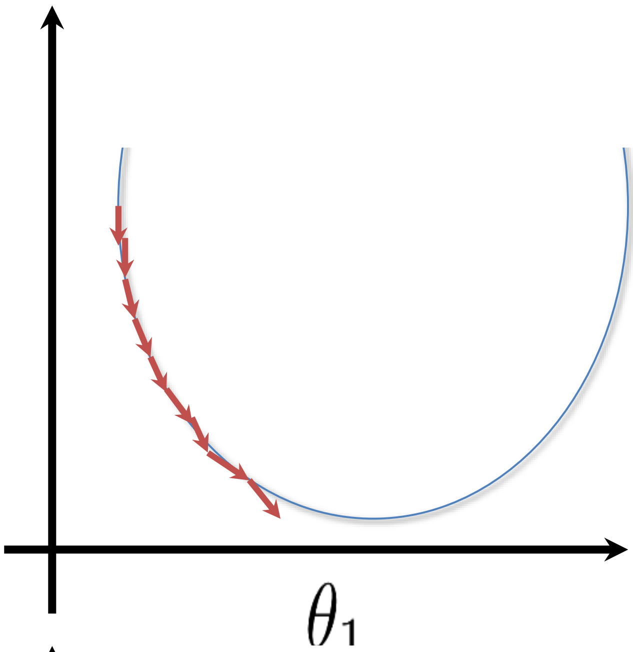


This line has a positive slope

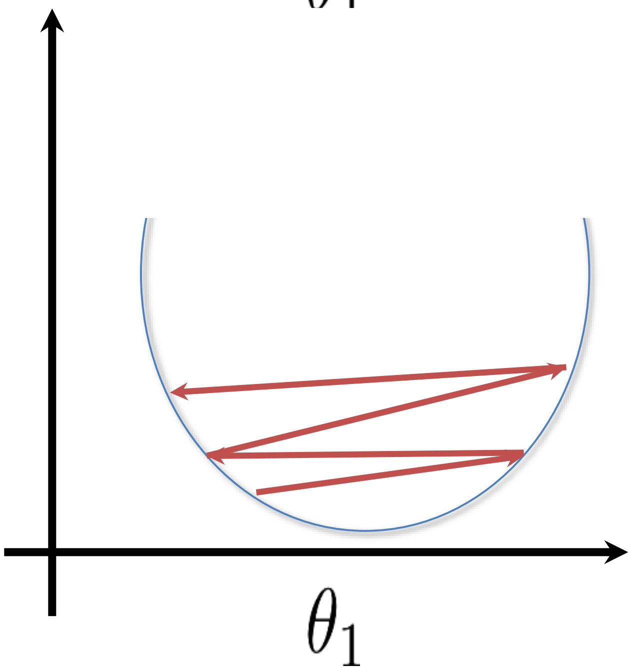
This line has a negative slope

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

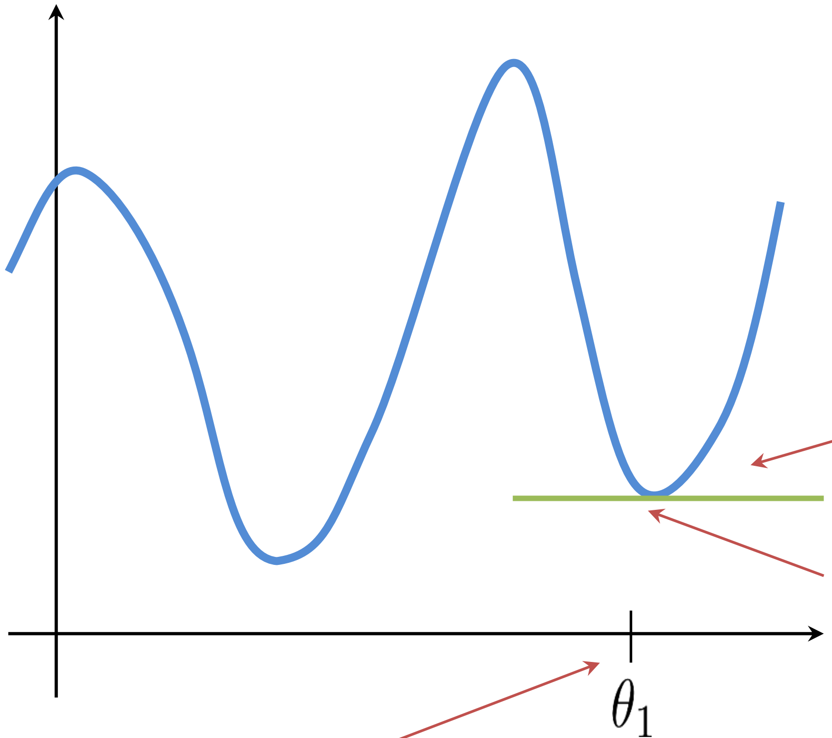
If α is too small, gradient descent can be slow.



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



What if you already start at the local minimum?



Slope of gradient (the derivative) = 0

θ_1 at local optima

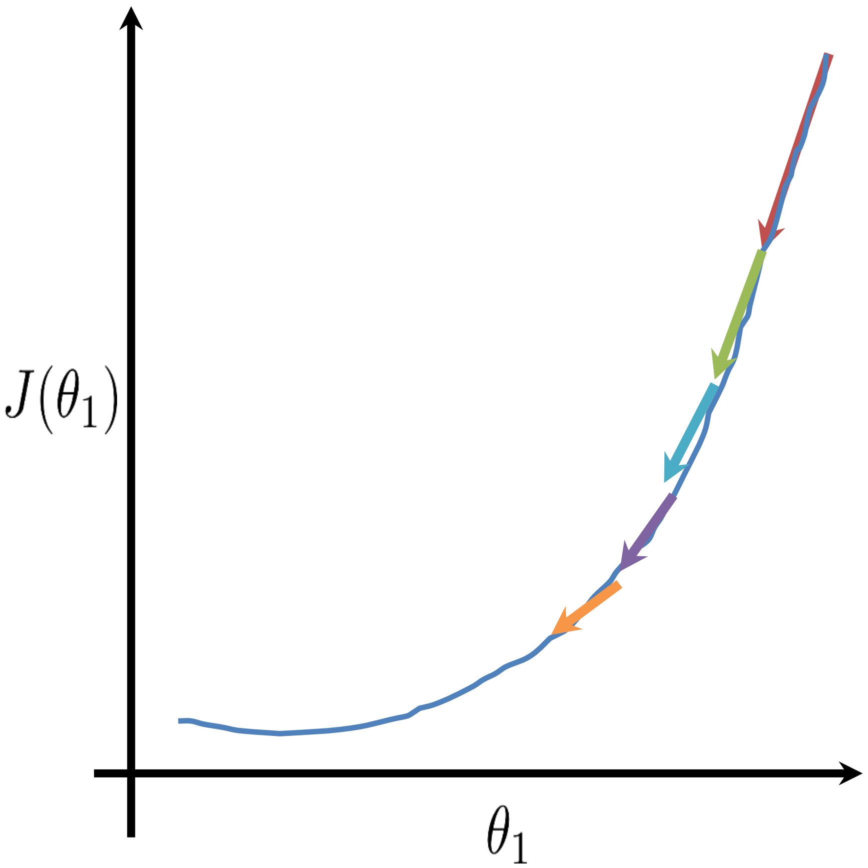
Current value of θ_1

$$\theta_1 := \theta_1 - \alpha \boxed{\frac{d}{d\theta_1} J(\theta_1)} = 0$$

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

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

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



Gradient descent algorithm

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

Linear Regression Model

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

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

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

$$\frac{\partial}{\partial\theta_j}J(\theta_0,\theta_1)=\frac{\partial}{\partial\theta_j}\frac{1}{2m}\sum_{i=1}^{m}(\theta_0+\theta_1x^i-y^i)^2$$

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

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

Gradient Descent Algorithm

$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

repeat until convergence {

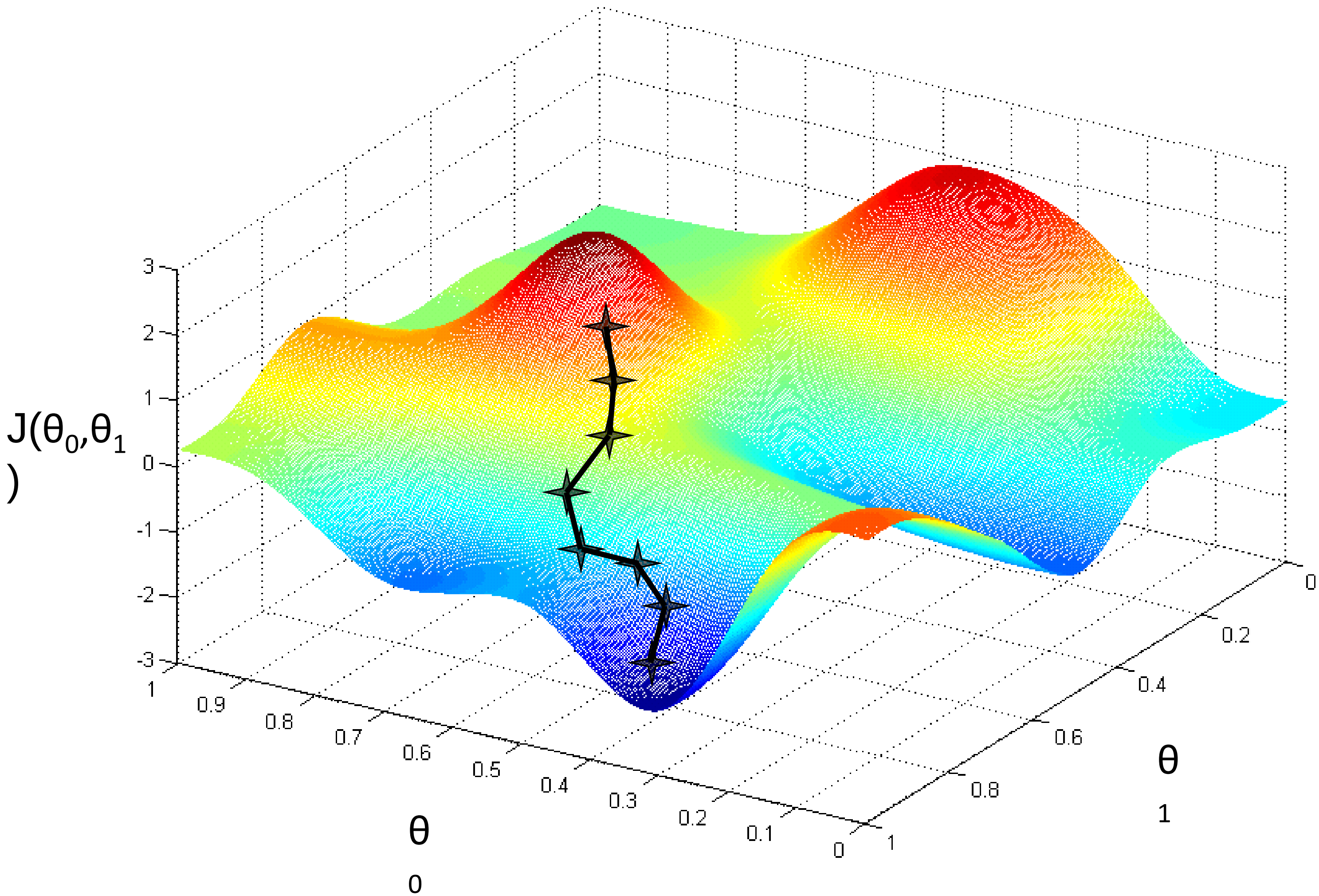
$$\theta_0 := \theta_0 - \alpha \boxed{\frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)}$$

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

}

update θ_0 and θ_1 simultaneously

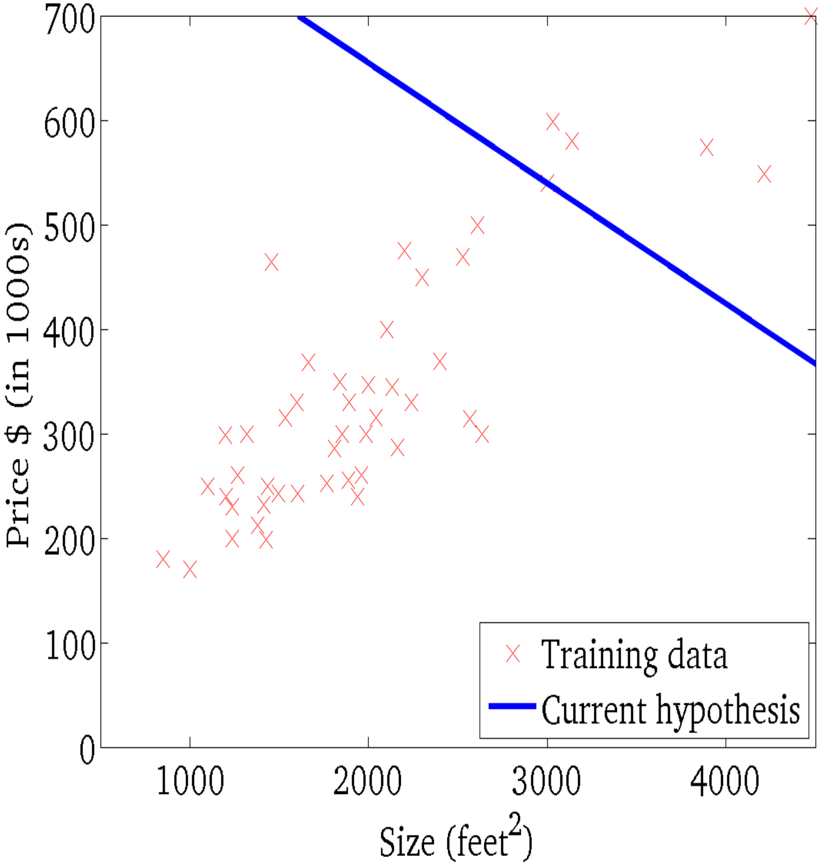
$$\frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$



J(θ0,θ1)
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
0
0.2
0.4
0.6
0.8
1

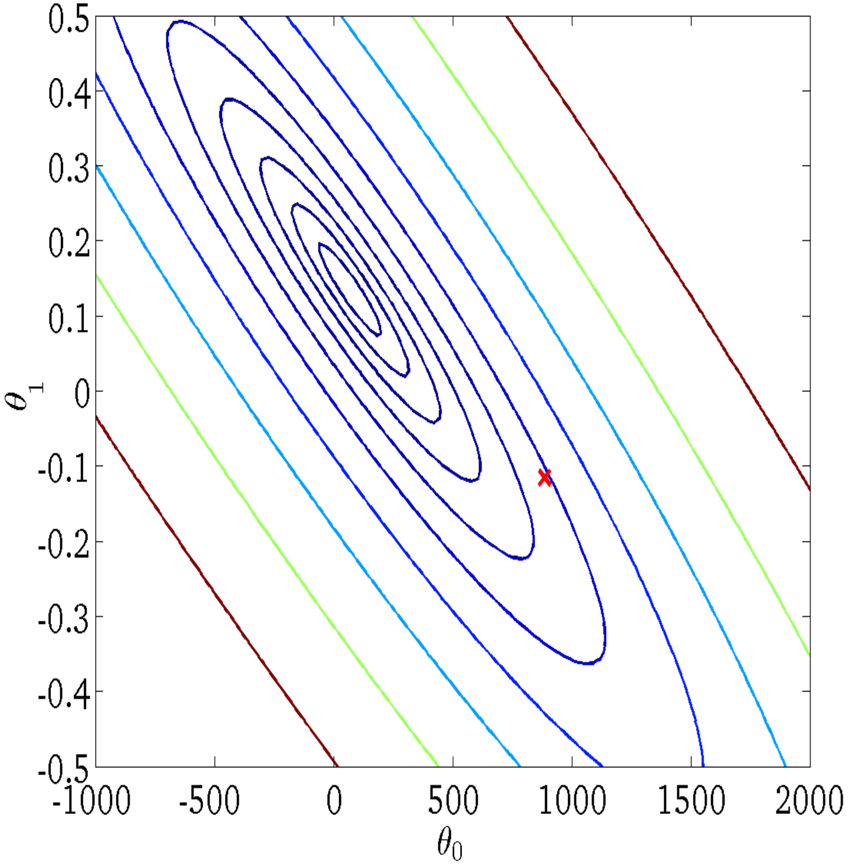
$$h_\theta(x)$$

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



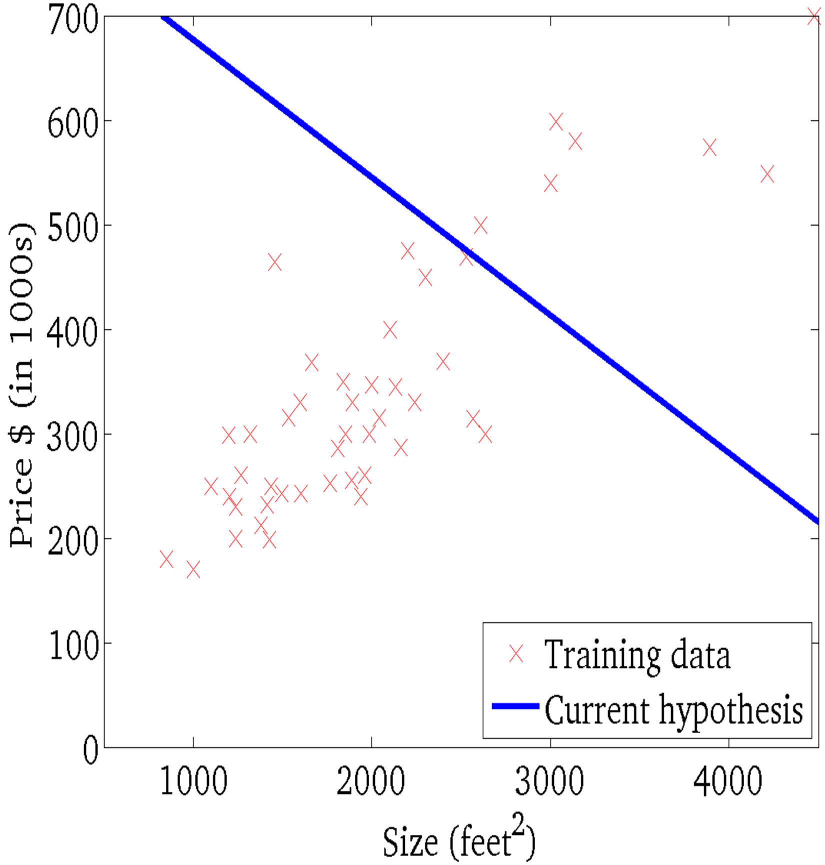
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



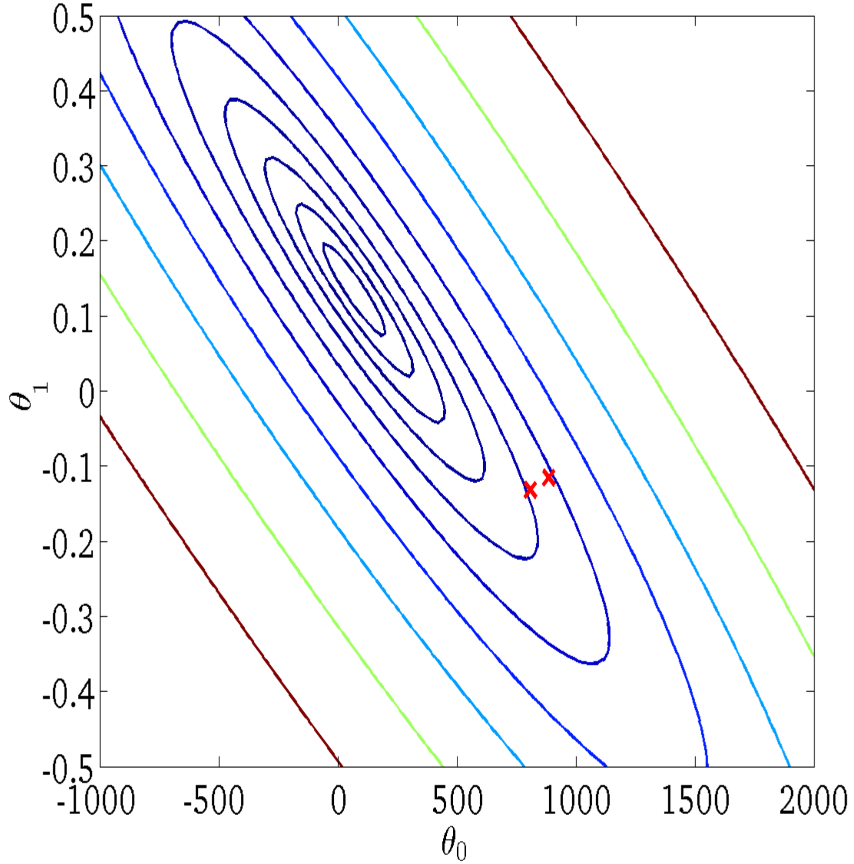
$$h_\theta(x)$$

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



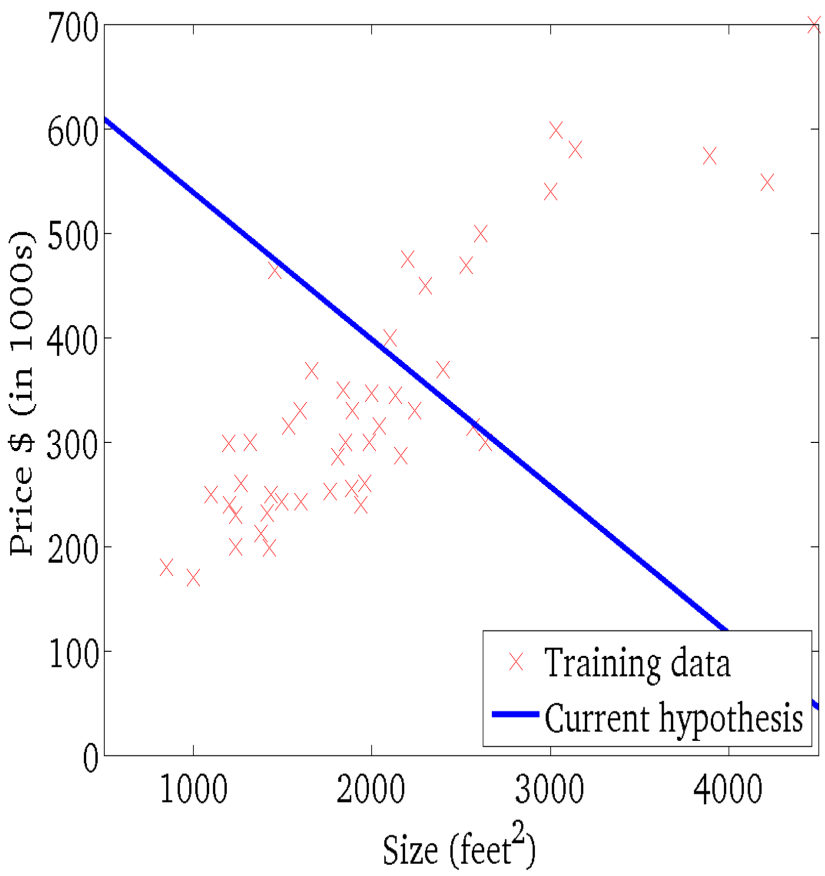
$$J(\theta_0, \theta_1)$$

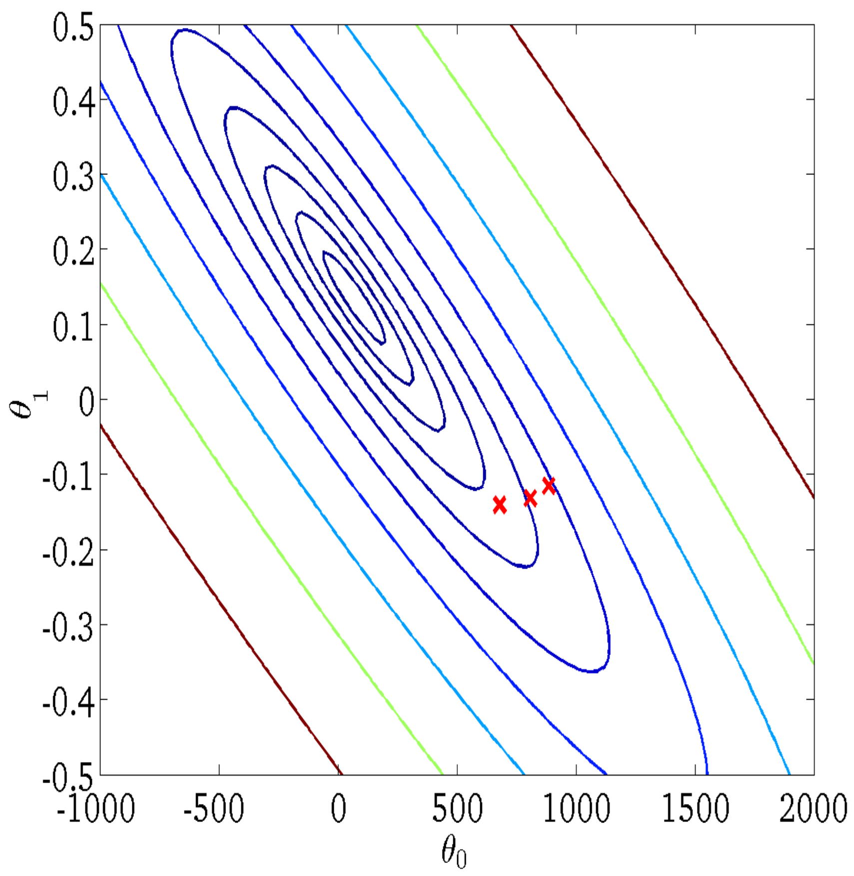
(function of the parameters θ_0, θ_1)



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

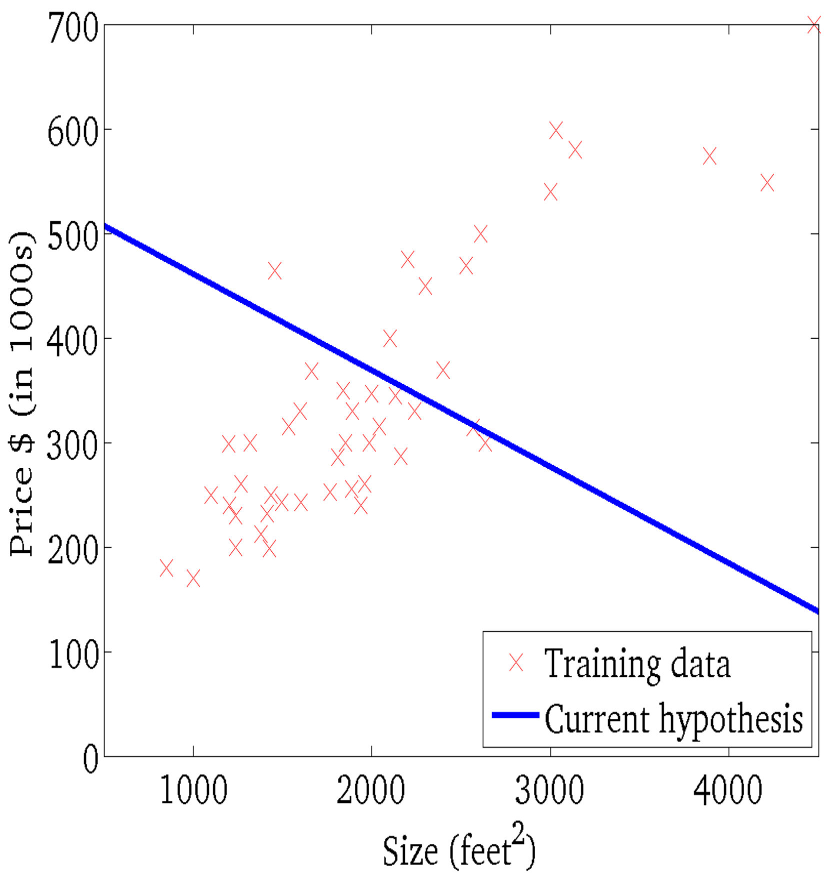
$J(\theta_0, \theta_1)$ (function of the parameters θ_0, θ_1)

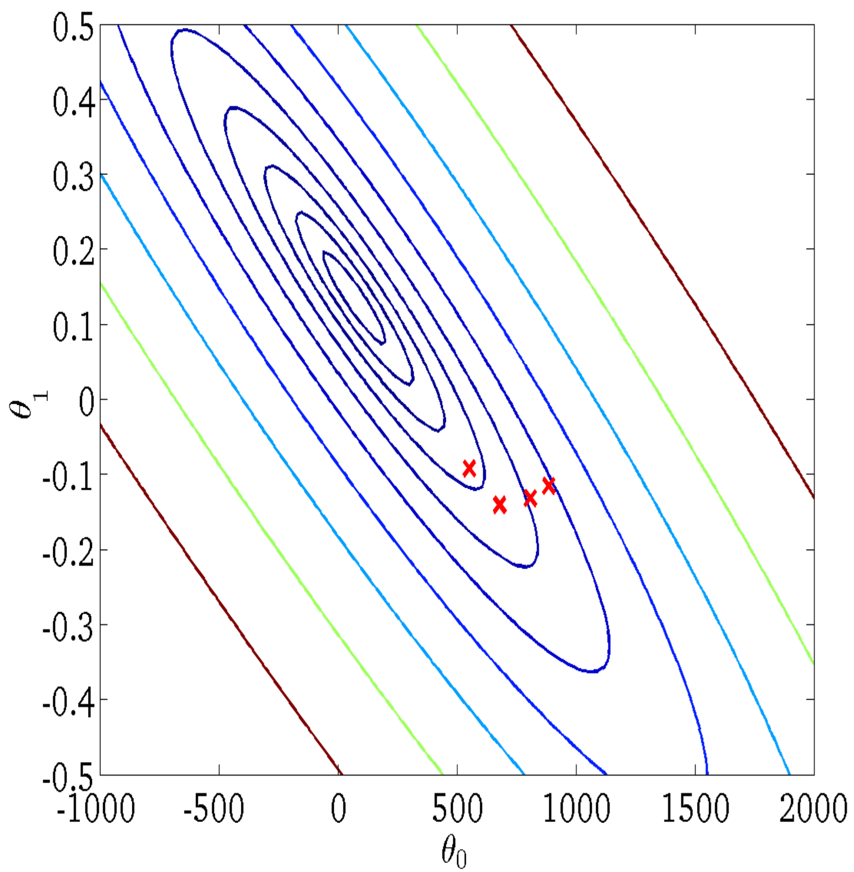




$h_\theta(x)$ $J(\theta_0, \theta_1)$

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



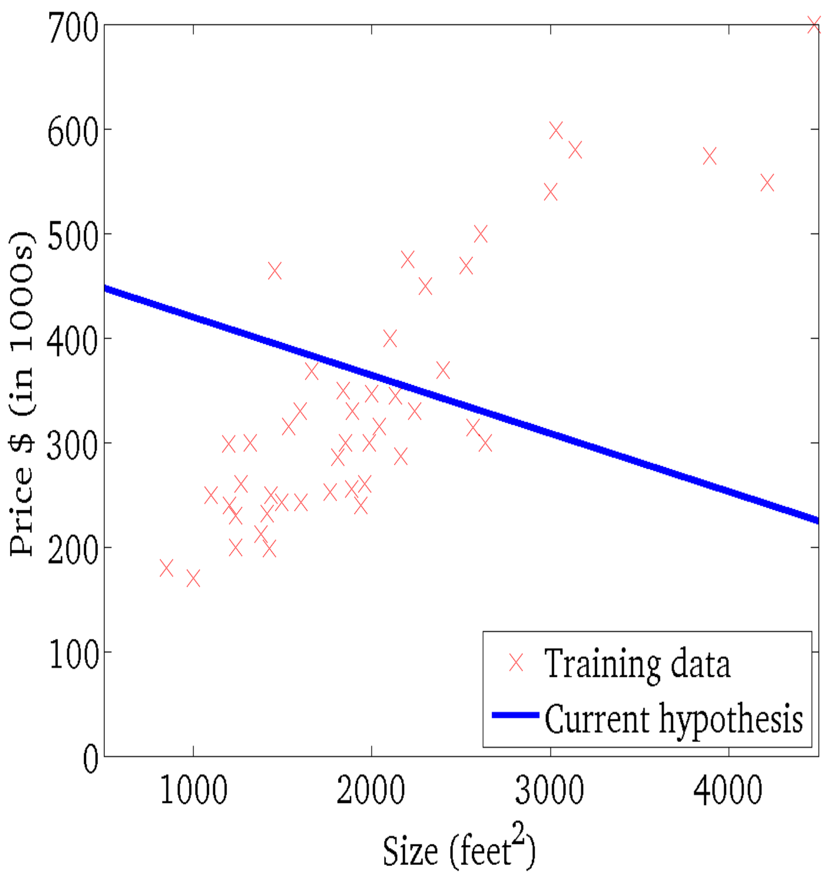


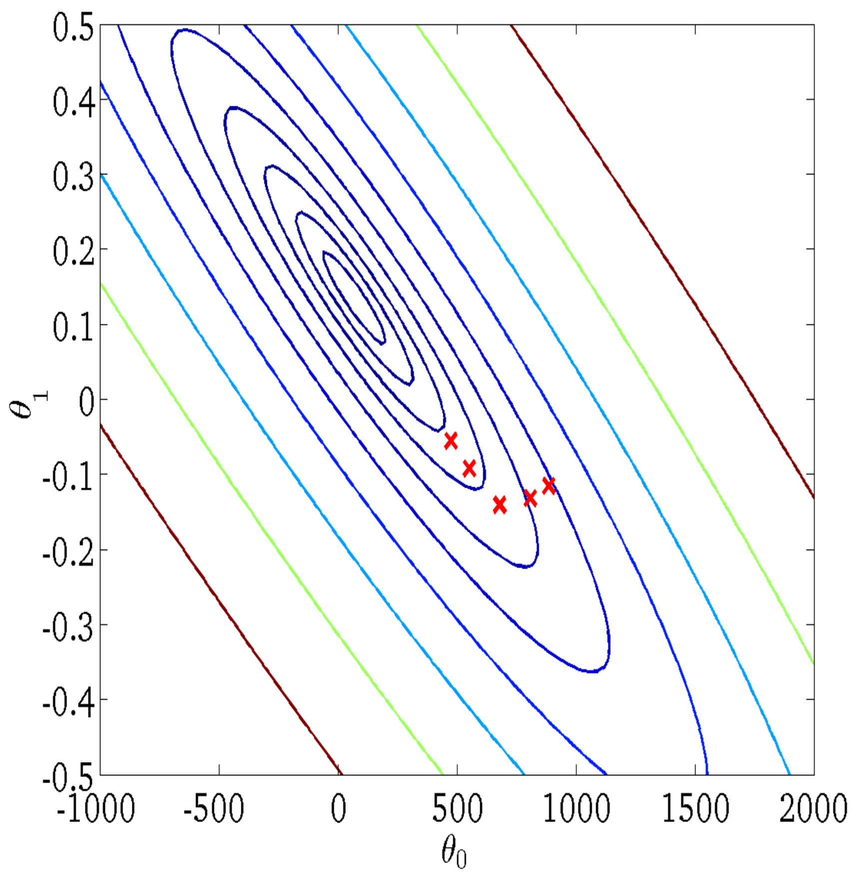
$$h_\theta(x)$$

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

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

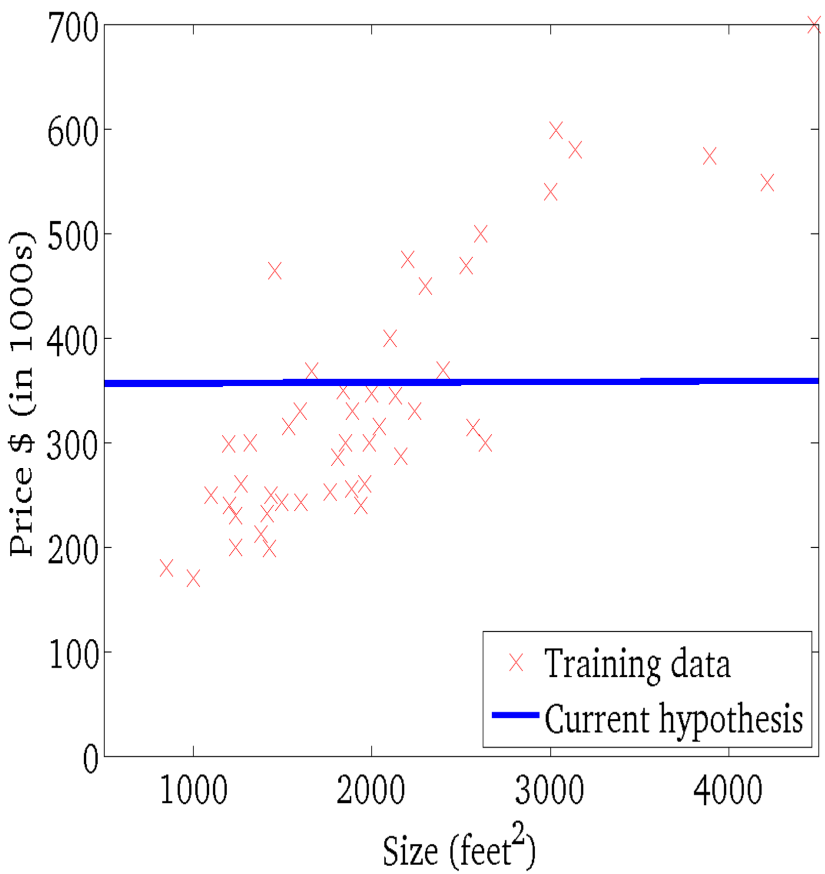
(function of the parameters θ_0, θ_1)





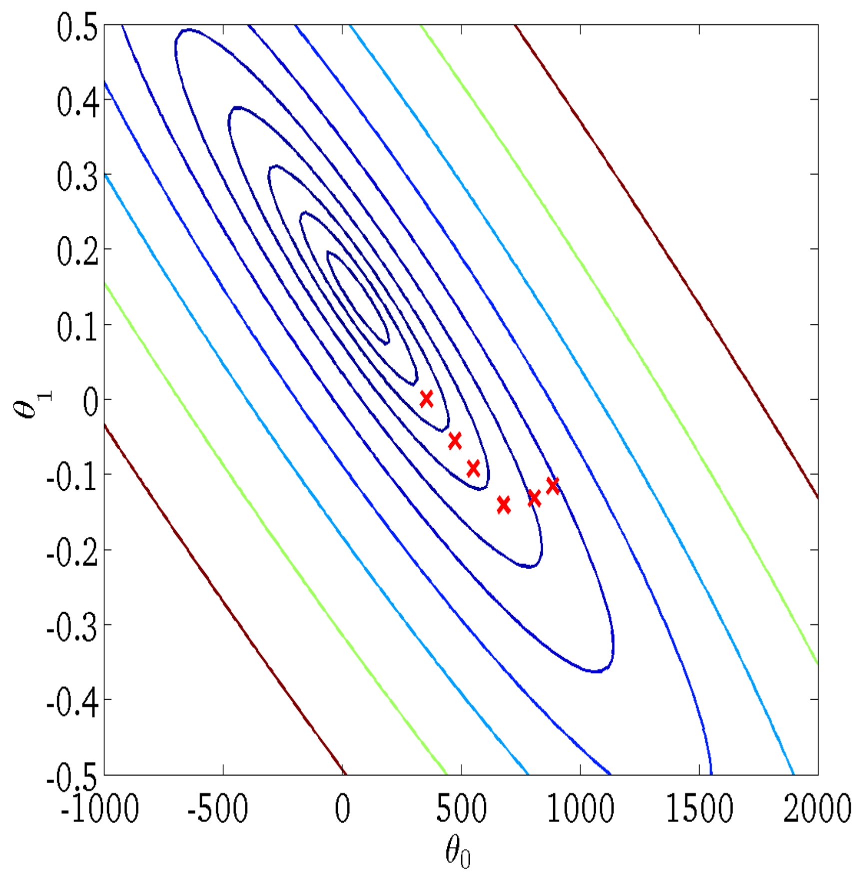
$h_\theta(x)$

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



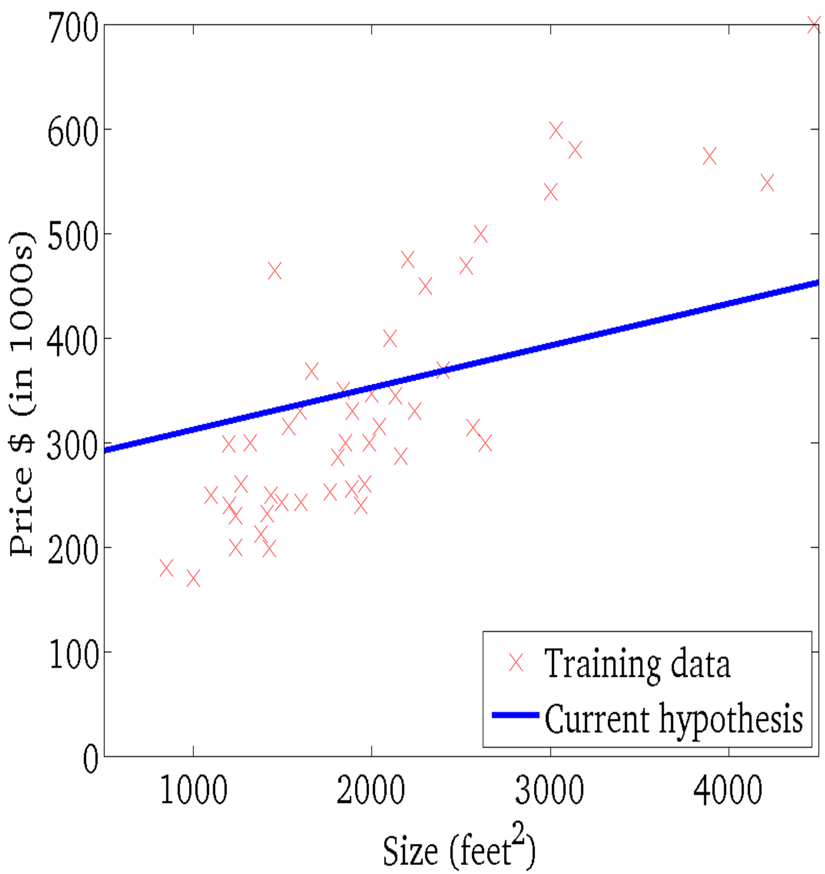
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



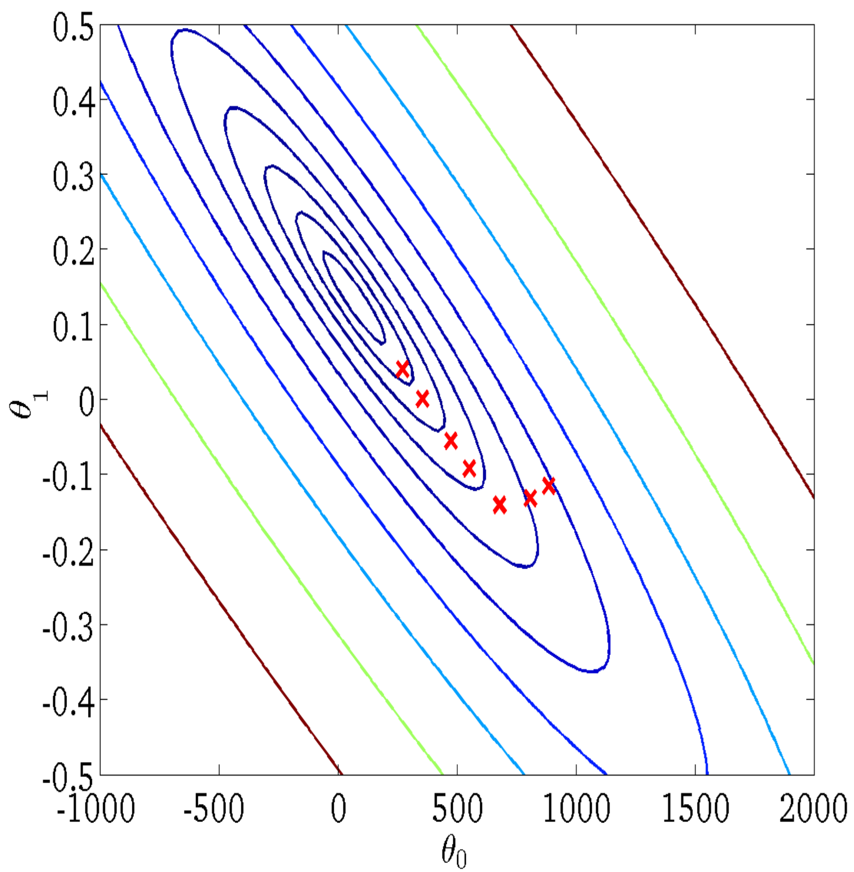
$$h_\theta(x)$$

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



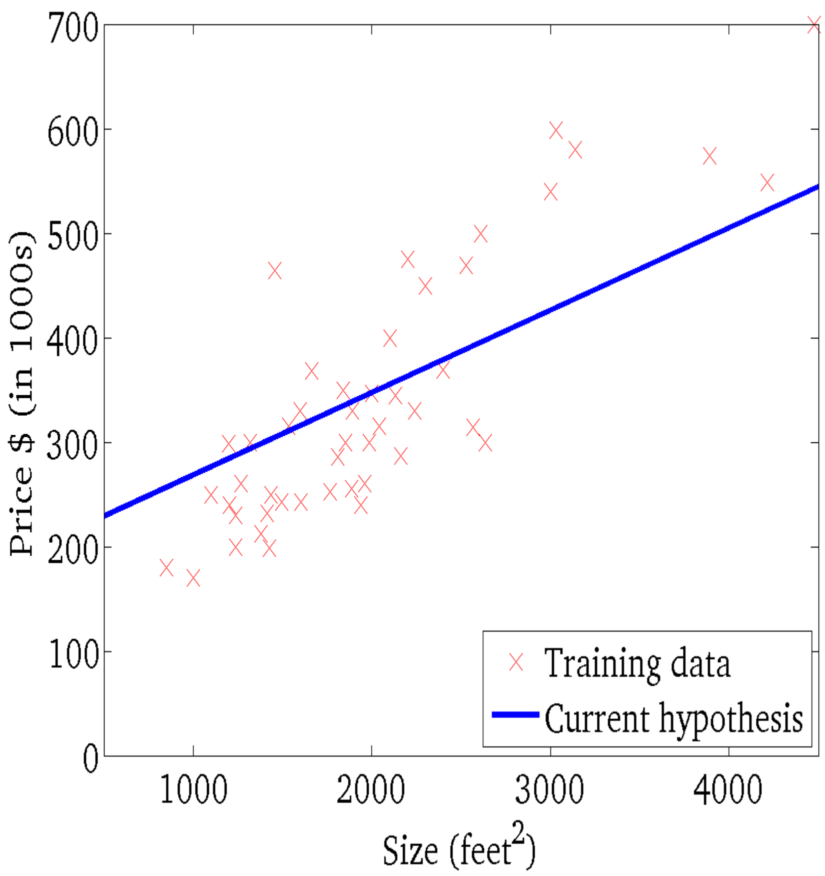
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



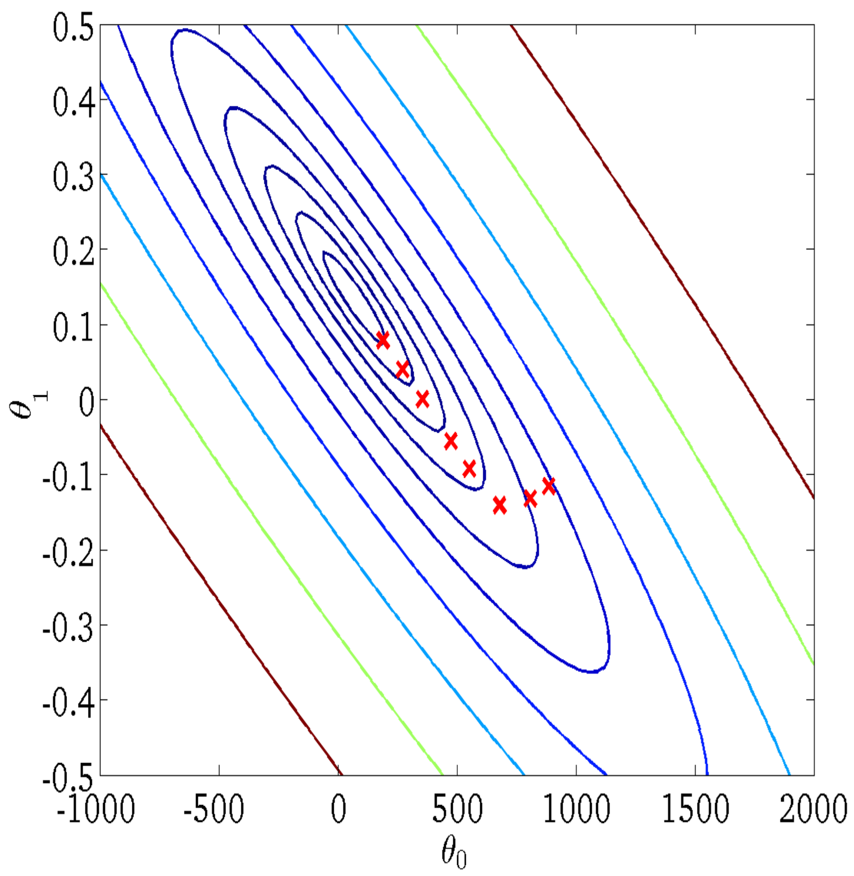
$$h_\theta(x)$$

(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)



Gradient Descent Algorithm

- **“Batch” Gradient Descent**: Each step of gradient descent uses all the training examples.

- **“Stochastic” Gradient Descent** : Each step of gradient descent uses one training example (or a smaller set of training examples). .
  - Good for large datasets

Single Variable Linear Regression

Size (feet^2) x	Price ($1000) y
2104	460
1416	232
1534	315
852	178
…	…

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

Multiple Variable Linear Regression

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
…	…	…	…	…

Linear Regression with 1 variable

$$h(x) = \theta_0 + \theta_1 x$$

Size ($feet^2$) X	Price ($1000) Y
2104	460
1416	232
1534	315
852	178
…	…

Linear Regression with multiple variables

$$h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

Size ($feet^2$) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home x_4	Price ($1000) x_5
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

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:

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

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 , 1, … , n)

Previously (n=1):

Repeat $\{$

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

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

(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 for θ_j

$j = 0, \ldots, n$)

$\}$

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

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

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

$\ldots$

Gradient Descent

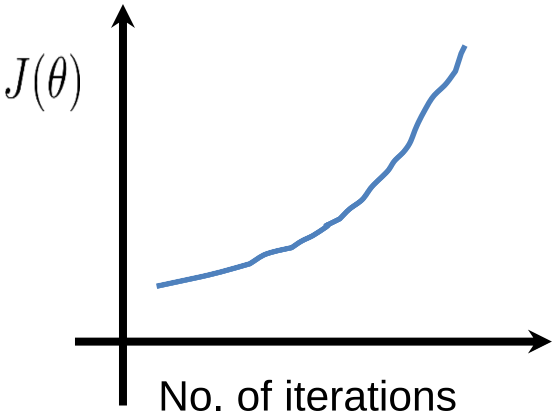
- Its very important to perform feature scaling and mean normalization when performing gradient descent.

Gradient Descent Tinkering

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

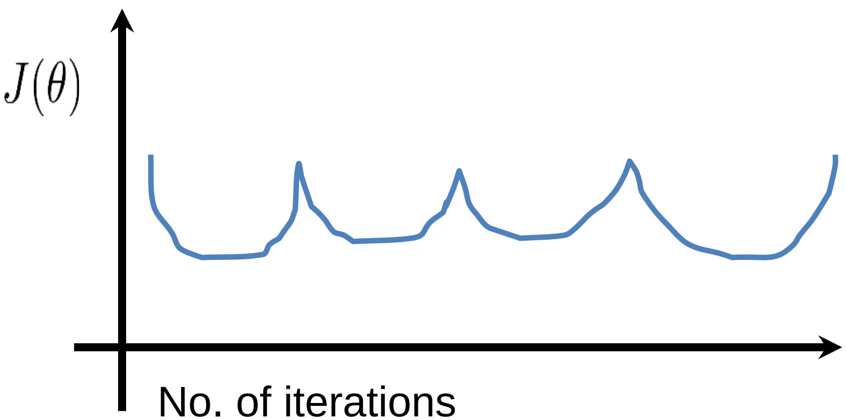
- Debugging – How to make sure gradient descent is working correctly.
- How to choose learning rate **α.**

Making sure gradient descent is working



These examples are of where gradient descent not working.

Solution: Use smaller α



- For sufficiently small α , α should decrease on every iteration.
- But if α is too small, gradient descent can be slow to converge.

Summary

- If **α** is too small: slow convergence
- If **α** is too large, *J(θ)* may not decrease on every iteration; may not converge.

- To choose **α**, try:

0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, …

3x 3x

Outline

- Regression
  - Linear regression
- Classification
  - Logistic Regression

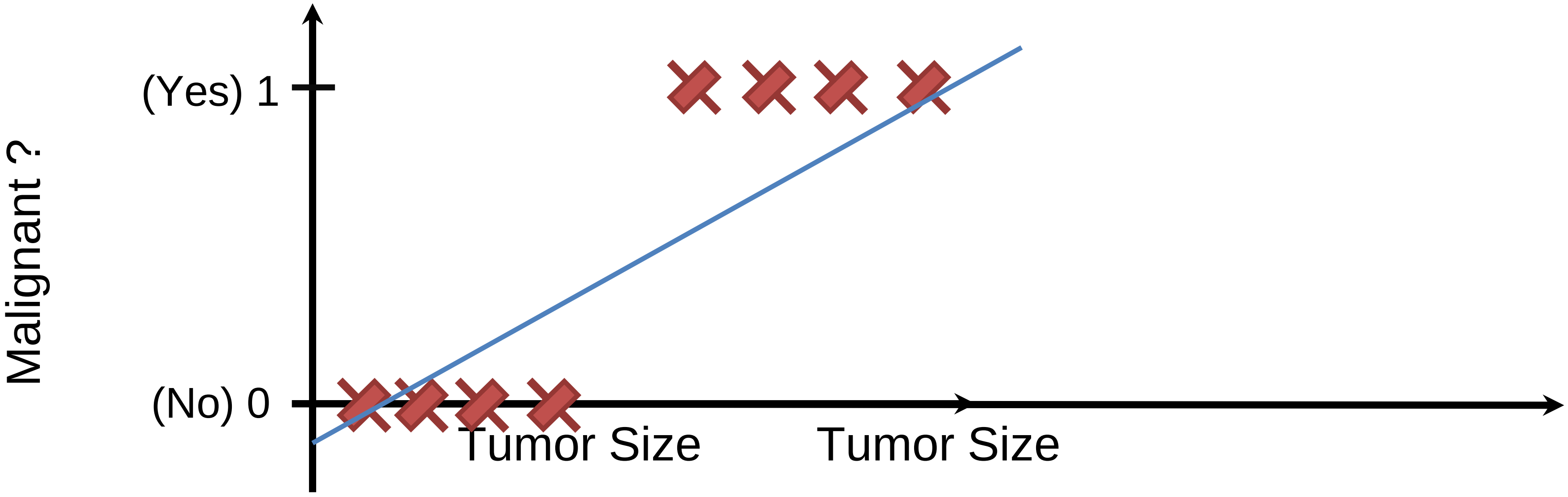
Classification

- **Email:** Spam / Not Spam?
- **Online Transactions:** Fraudulent (Yes / No)?
- **Tumor:** Malignant / Benign ?

$$y \in \{0, 1\}$$

0: “Negative Class” (e.g., benign tumor)

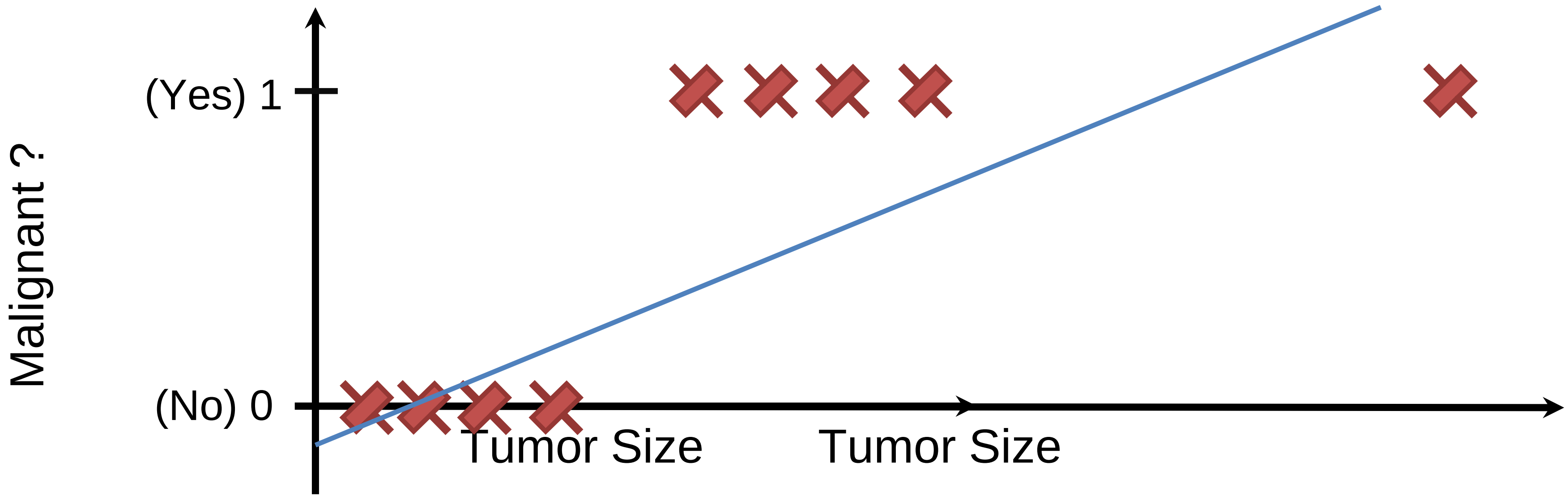
1: “Positive Class” (e.g., malignant tumor)



Threshold classifier output $h_\theta(x)$ at 0.5:

If $h_\theta(x) \geq 0.5$, predict "y = 1"

If $h_\theta(x) < 0.5$, predict "y = 0"



Threshold classifier output $h_\theta(x)$ at 0.5:

If $h_\theta(x) \geq 0.5$, predict "y = 1"

If $h_\theta(x) < 0.5$, predict "y = 0"

Classification: y = 0 or 1

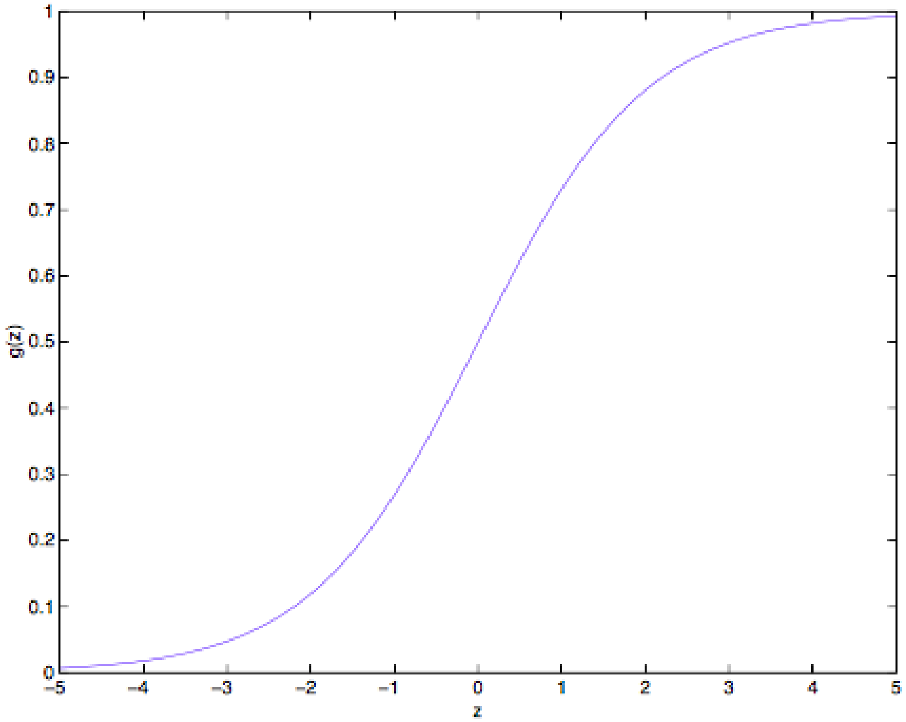
$h_\theta(x)$ can be > 1 or < 0

Hence, Linear regression is not the way to go…need another method for classification problems.

Logistic Regression: $0 \leq h_\theta(x) \leq 1$

Logistic Regression

- We will choose to define our function h as follows:



$$h(x) = g(\theta^T x) = \frac{1}{1+e^{-\theta^T x}}$$

$$g(z) = \frac{1}{1+e^{-z}}$$

Called logistic function or sigmoid function

Notice that g(z) tends towards 1 as $z \to \infty$, and g(z) tends towards 0 as $z \to -\infty$. Hence it is always bounded by 0 and 1.

Logistic Regression Cost Function

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Note: $y = 0$ or 1 always

Logistic Regression Cost Function

$$J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

[redacted]

To fit parameters θ :

$$\min_\theta J(\theta)$$

To make a prediction given new x :

Output $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

Logistic Regression using Gradient Descent

[redacted]

Want $\min_\theta J(\theta)$:

Repeat $\{$

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

(simultaneously update all θ_j)

$\}$

Algorithm looks identical to linear regression!