Linear Regression

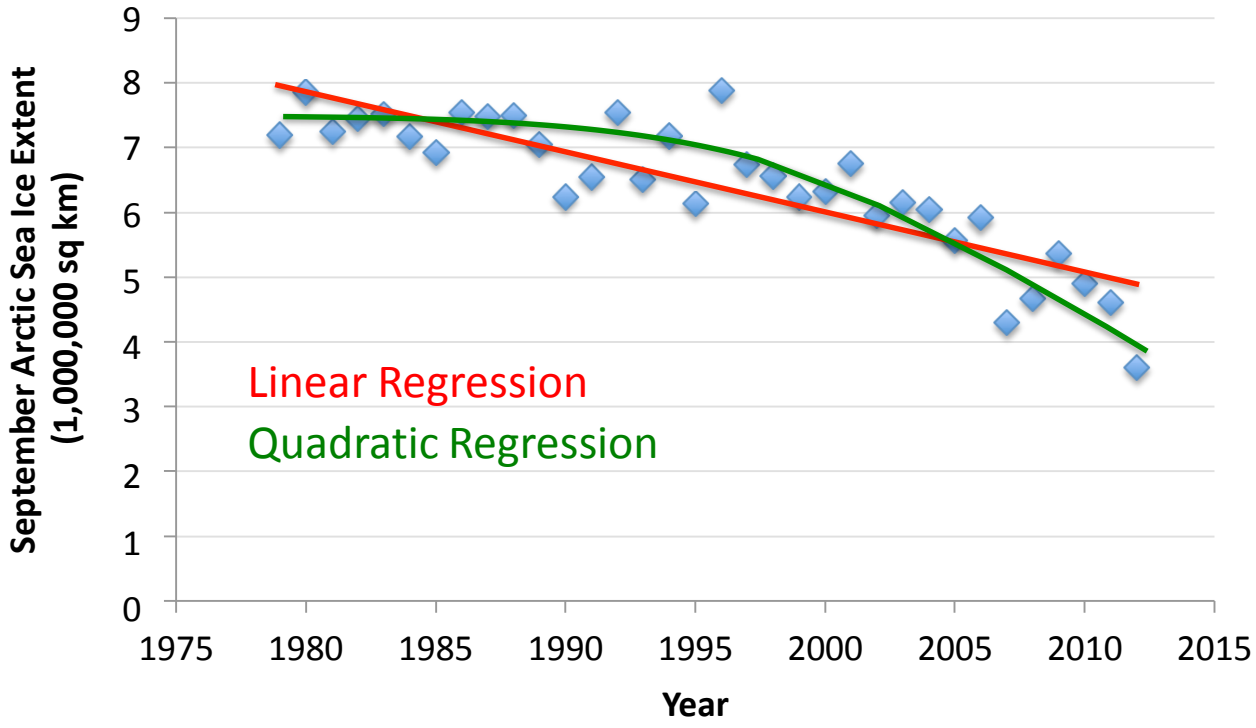
Regression

Initially Given:

- Data $\mathbf{X} = \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(n)}$
- Corresponding labels $\mathbf{y} = y^{(1)}, \ldots, y^{(n)}$

Then →

Given an input $\boldsymbol{X}$ we would like to compute an output $\boldsymbol{Y}$



Regression Techniques

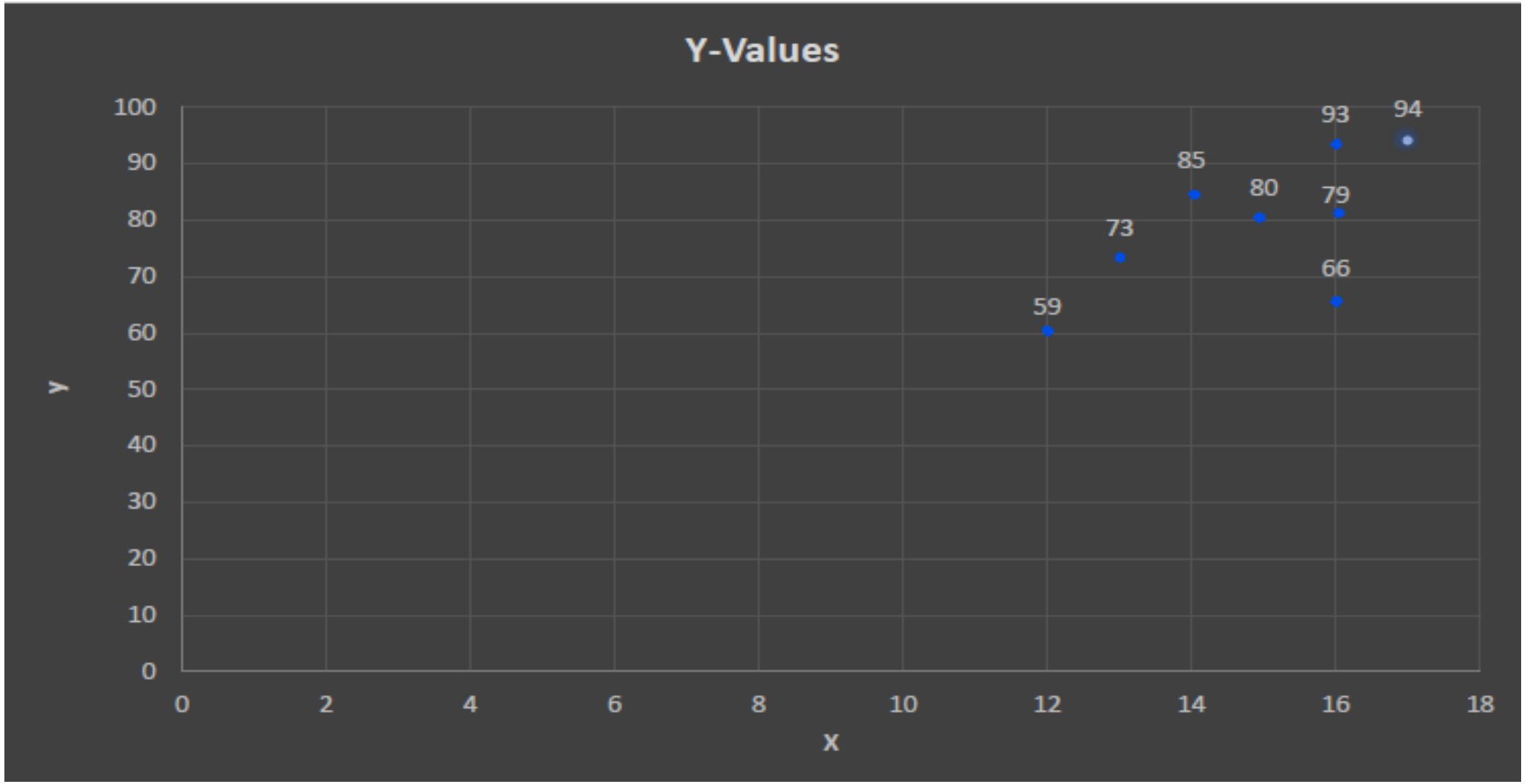
- Pearson correlation coefficient
- Gradient Descent
- Closed Form Solution

Pearson correlation coefficient

Measures the statistical association between two continuous variable

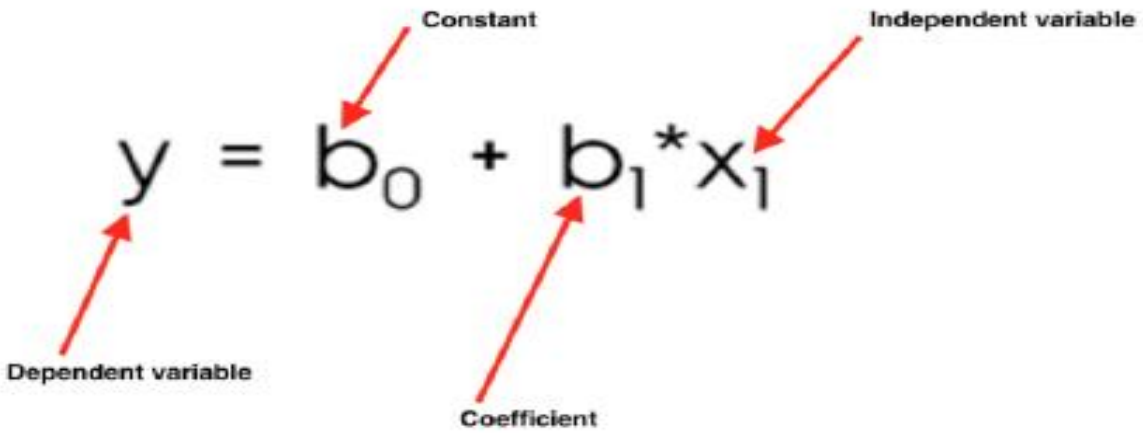
X	Y	(X-X)	(Y - $\bar{Y}$)	(X-$\bar{X}$) (Y- $\bar{Y}$)	(X-$\bar{X}$)²	(Y - $\bar{Y}$)²
17	94	1.4	14.3	20.02	1.96	204.99
13	73	-2.6	- 6.7	17.42	6.76	44.89
12	59	-3.6	- 20.7	74.52	12.96	428.49
15	80	-0.6	0.3	-0.18	0.36	0.09
16	93	0.4	13.3	5.32	0.16	176.89
14	85	-1.6	5.3	-8.48	2.56	28.09
16	66	0.4	-13.7	-5.48	0.16	187.69
16	79	0.4	-0.7	-0.28	0.16	0.49
18	77	2.4	-2.7	-6.48	5.76	7.29
19	91	3.4	11.3	38.42	11.56	127.69
$\bar{x} = 15.6$	$\bar{y} = 79.7$			$\Sigma = 134.8$	$\Sigma = 42.4$	$\Sigma = 1206.1$

Pearson correlation coefficient



Linear regression equation

Pearson correlation coefficient



$$\hat{Y} = m * x + b$$

m ----------> slope

X ----------> feature (input)

b ----------> y_intercept

m(slope) = r * (Sy / Sx)

$$r = \frac{\sum(x-\bar{x})(y-\bar{y})}{\sqrt{\sum(x-\bar{x})^2 \sum(y-\bar{y})^2}}$$

r = Pearson common factor,معامل برسون

$$sy = \sqrt{\frac{(y-\bar{y})^2}{(N-1)}}$$

$$sx = \sqrt{\frac{(x-\bar{x})^2}{(N-1)}}$$

Sx= Standard deviation for x, الانحراف المعيارى

Sy= Standard deviation for y,

$$b = \bar{y} - m\,\bar{x}$$

How to calculate error value

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

Cost function = Error function = Mean square error

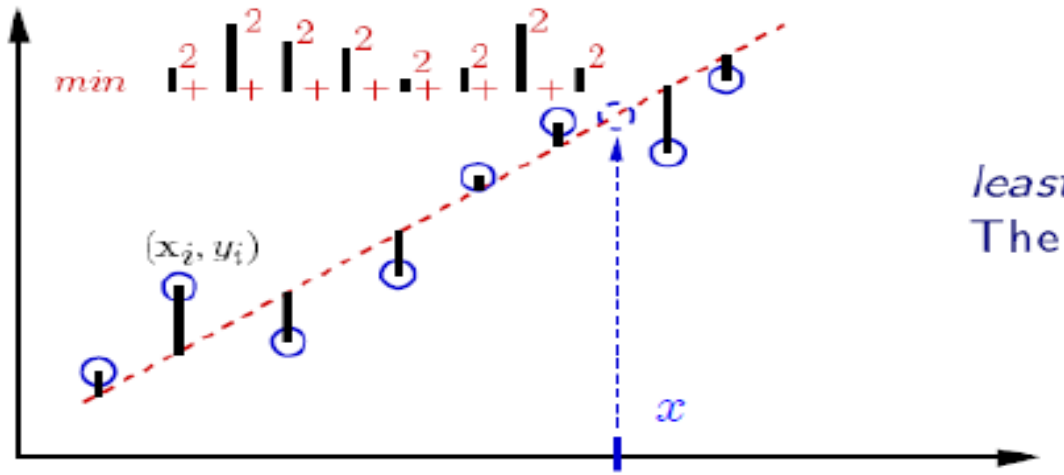
Linear Regression

- Hypothesis:

$$y = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_d x_d = \sum_{j=0}^{d} \theta_j x_j$$

Assume $x_0 = 1$

- Fit model by minimizing sum of squared errors



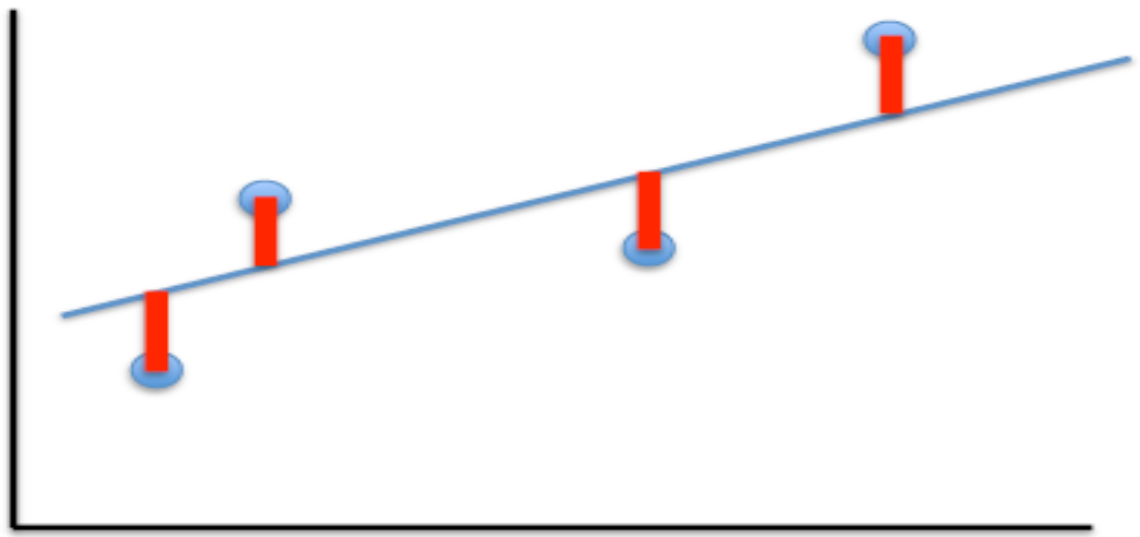
least squares (LSQ)
The fitted line is used as a predictor

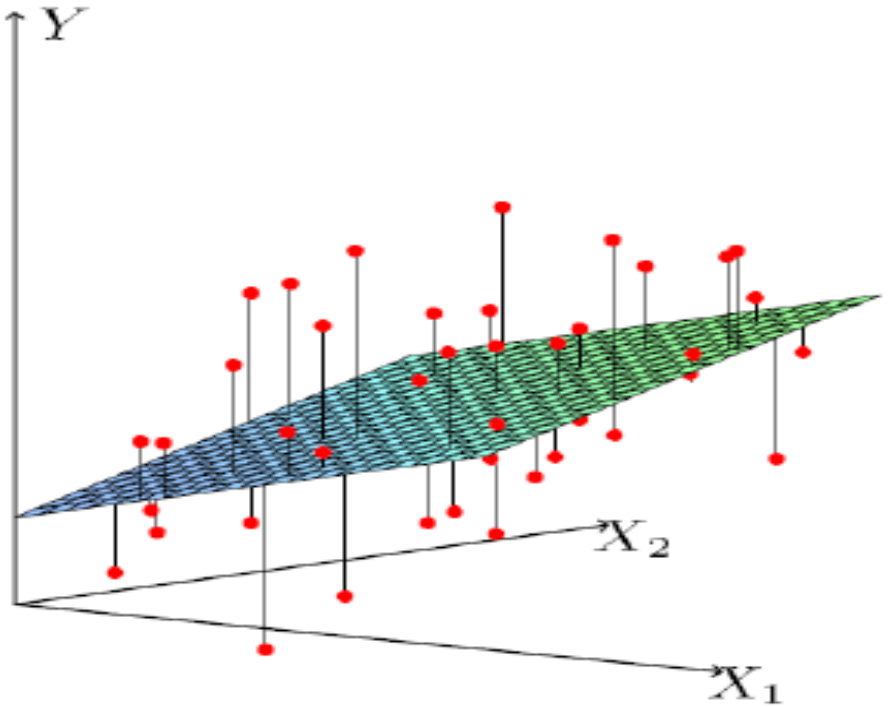
Least Squares Linear Regression

- Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2n} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} \right) - y^{(i)} \right)^2$$

- Fit by solving $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$





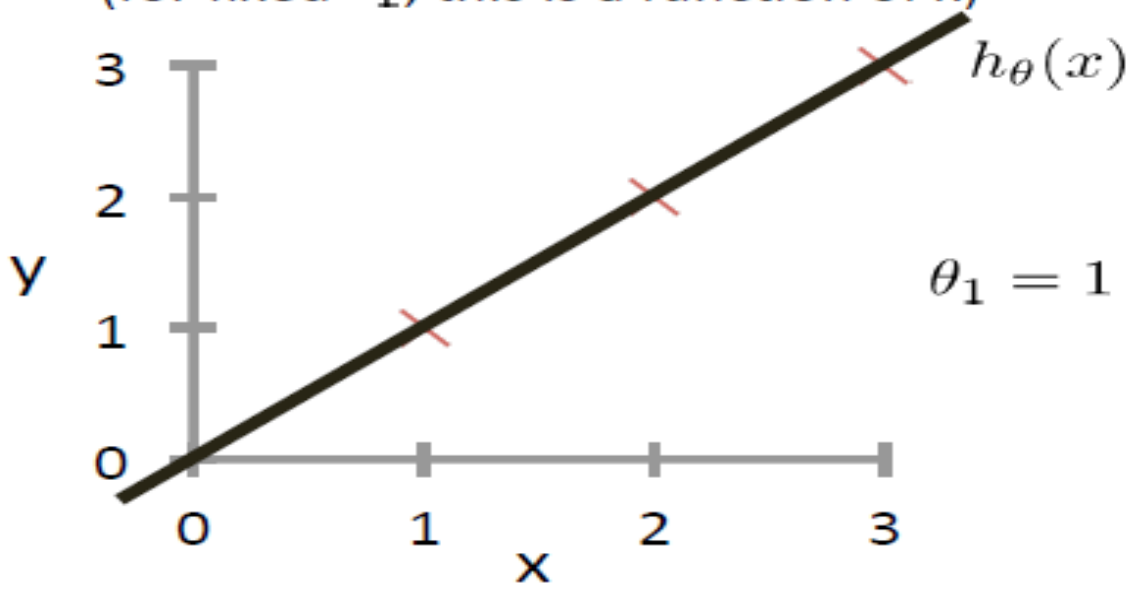
Intuition Behind Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2$$

For insight on J(), let's assume $x \in \mathbb{R}$ so $\boldsymbol{\theta} = [\theta_0, \theta_1]$

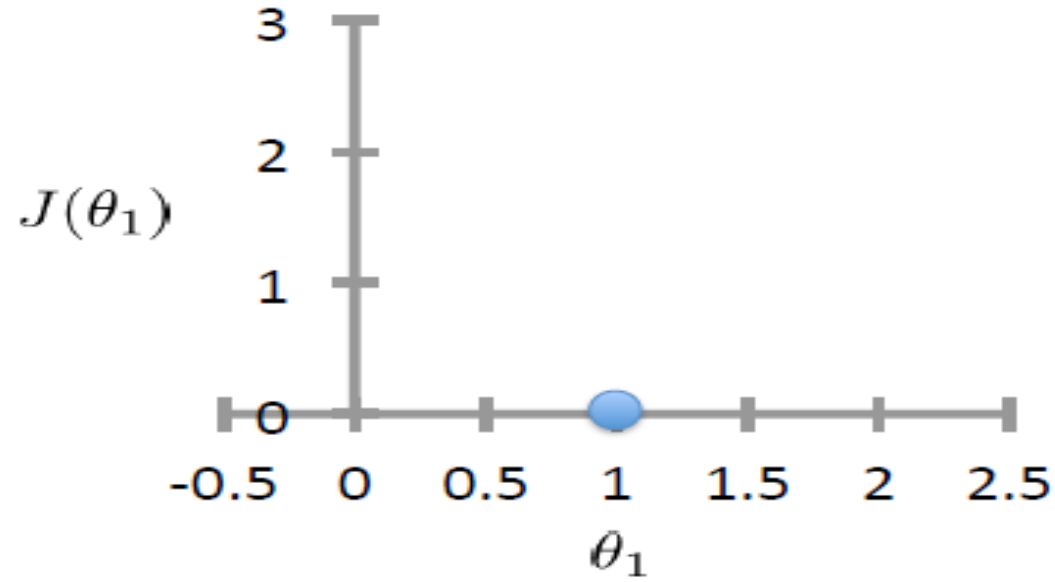
$h_\theta(x)$

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



$J(\theta_1)$

(function of the parameter θ_1)



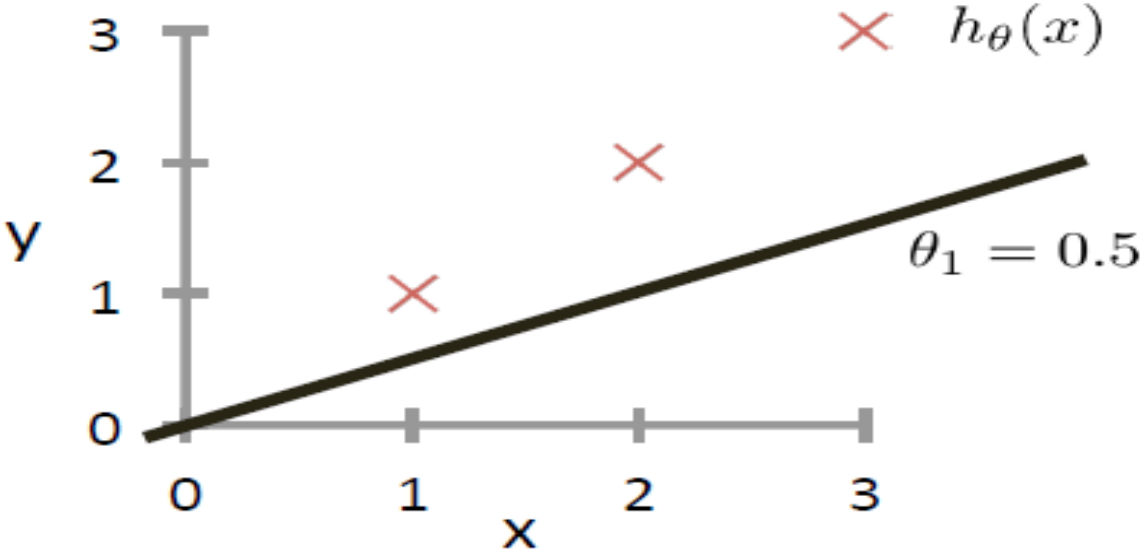
Intuition Behind Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2$$

For insight on J(), let's assume $x \in \mathbb{R}$ so $\boldsymbol{\theta} = [\theta_0, \theta_1]$

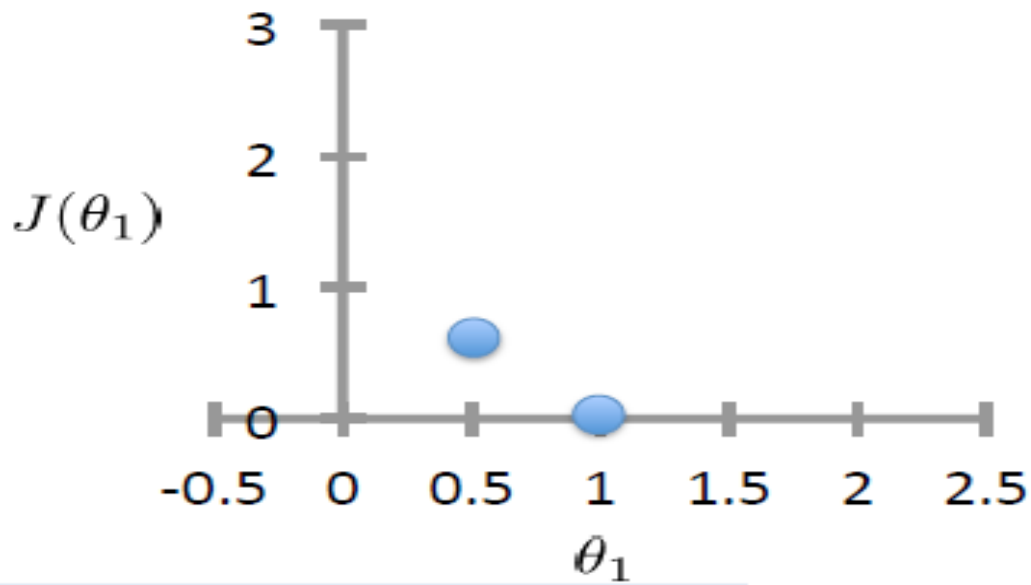
$h_\theta(x)$

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



$J(\theta_1)$

(function of the parameter θ_1)



$$J([0, 0.5]) = \frac{1}{2 \times 3}\left[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2\right] \approx 0.58$$

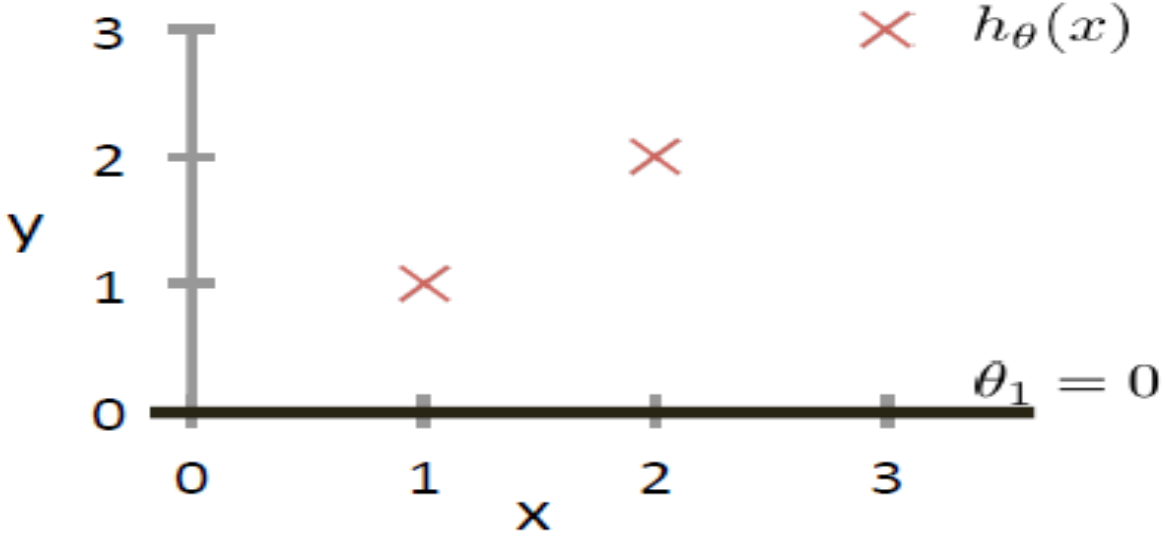
Intuition Behind Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2$$

For insight on J(), let's assume $x \in \mathbb{R}$ so $\boldsymbol{\theta} = [\theta_0, \theta_1]$

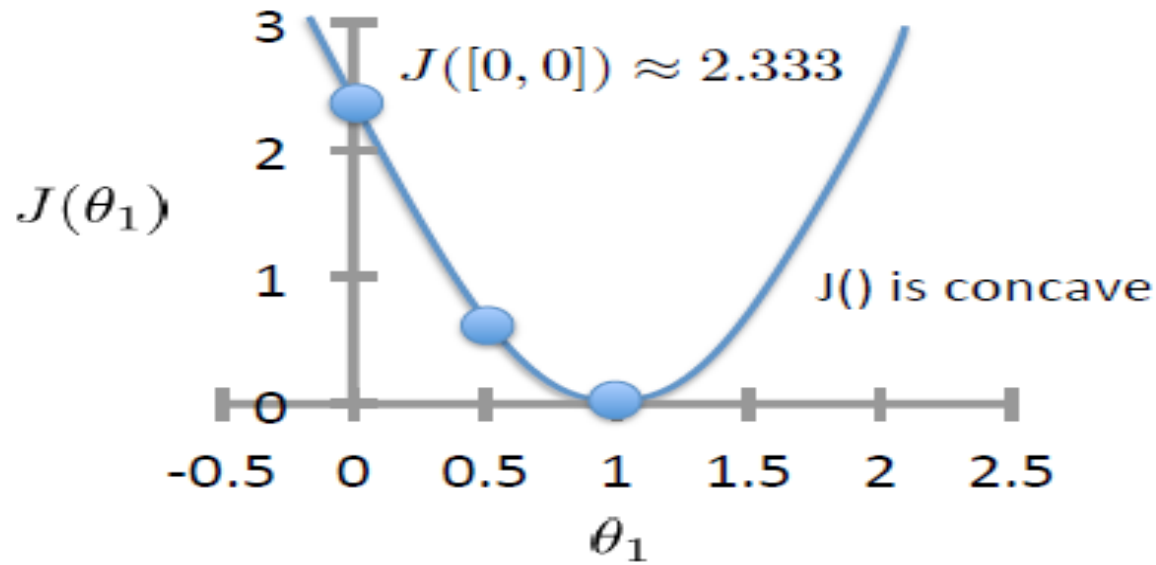
$h_\theta(x)$

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

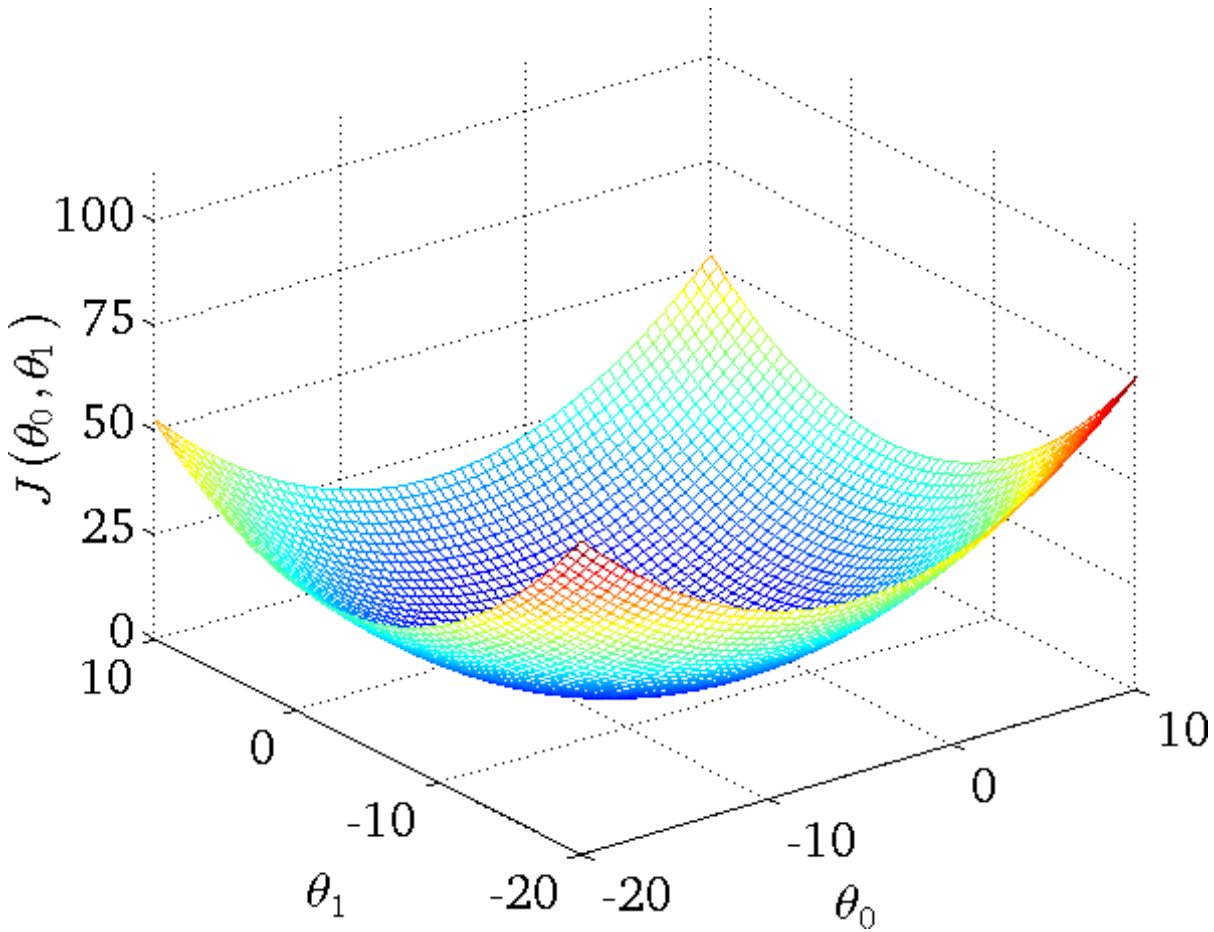


$J(\theta_1)$

(function of the parameter θ_1)



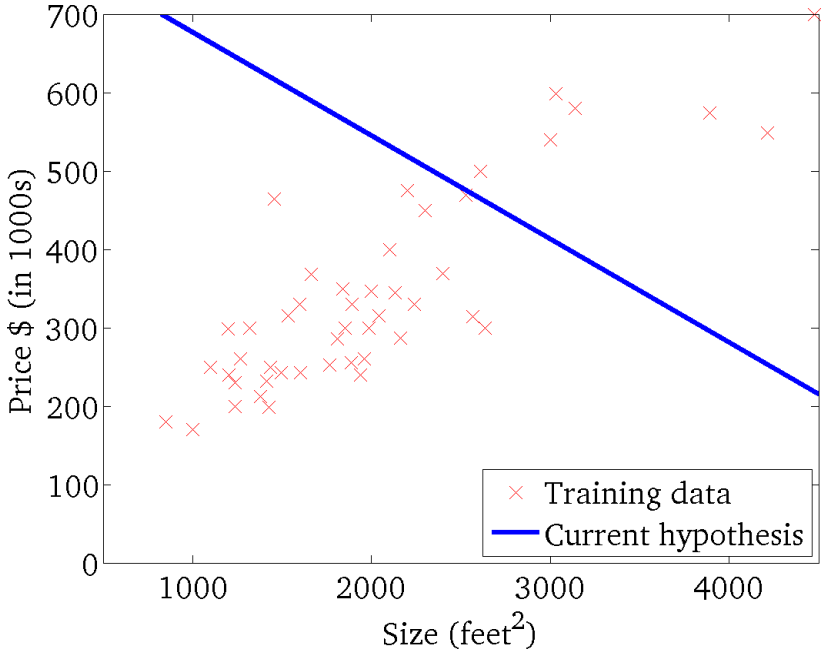
Intuition Behind Cost Function



Intuition Behind Cost Function

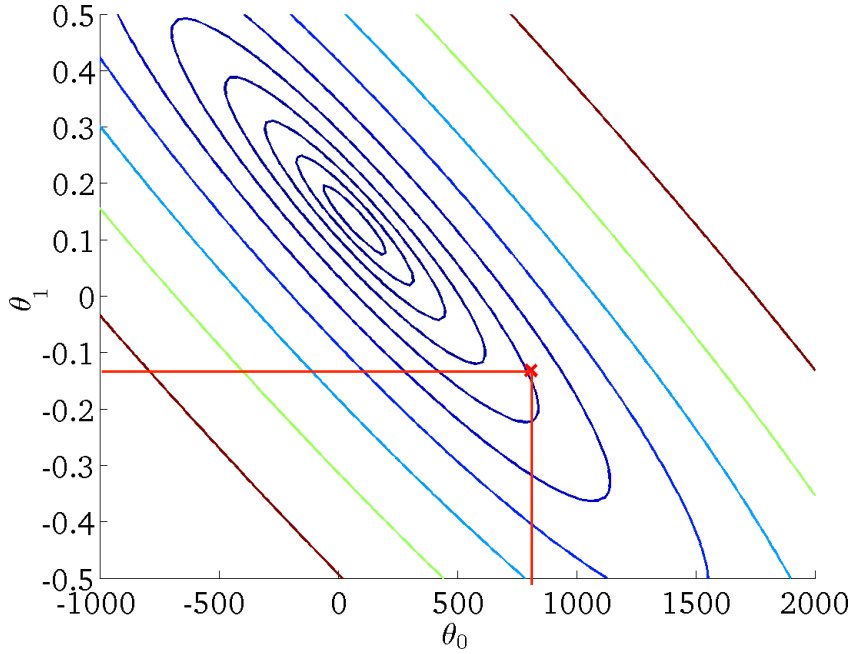
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

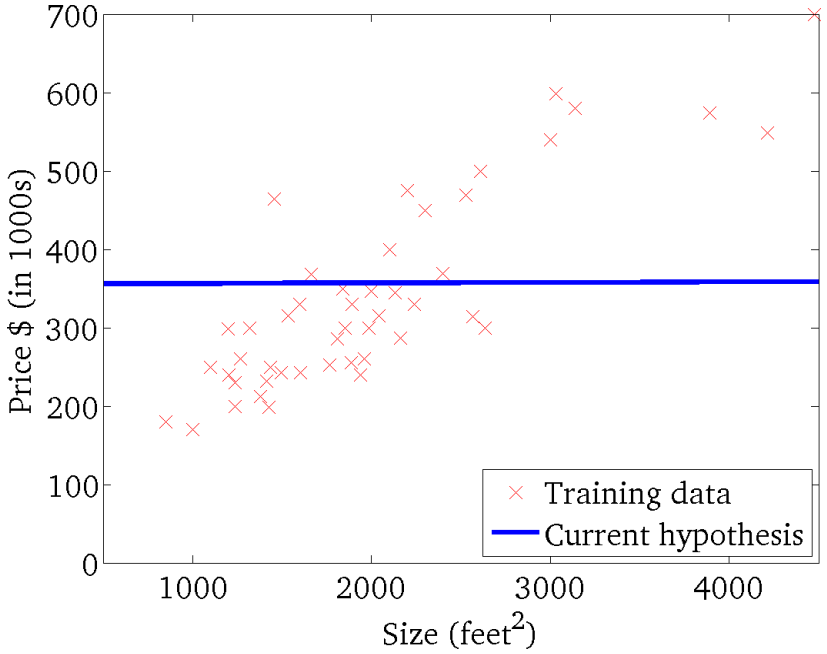
(function of the parameters θ_0, θ_1)



Intuition Behind Cost Function

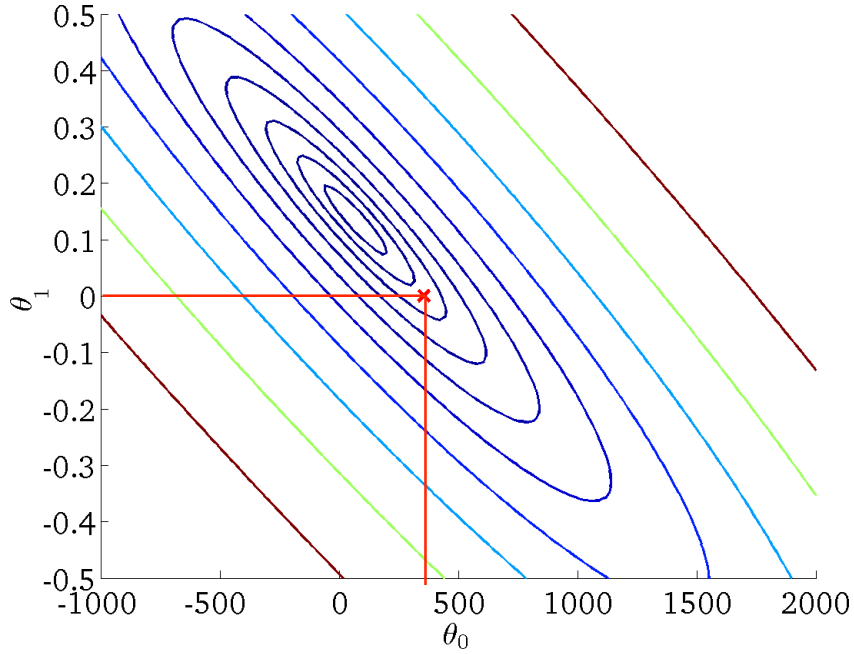
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

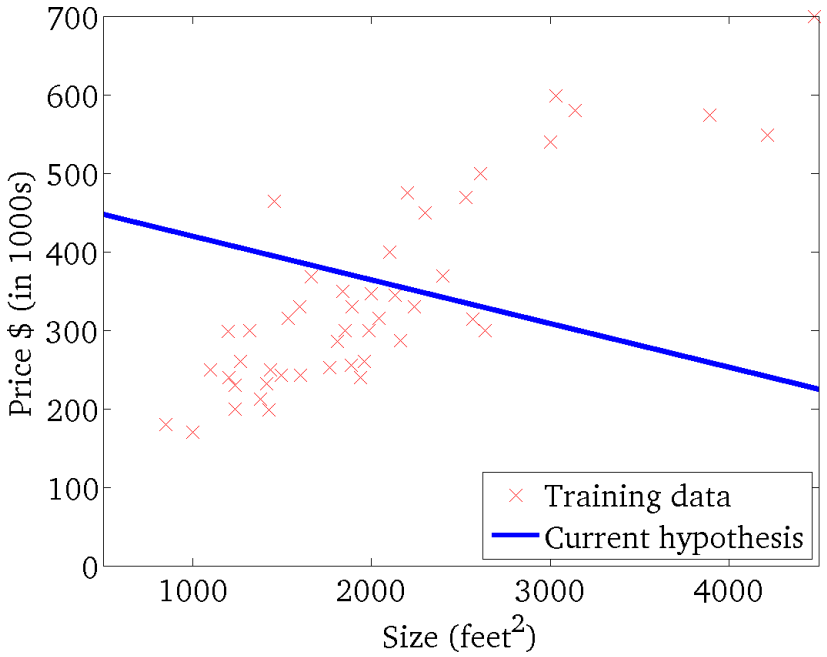
(function of the parameters θ_0, θ_1)



Intuition Behind Cost Function

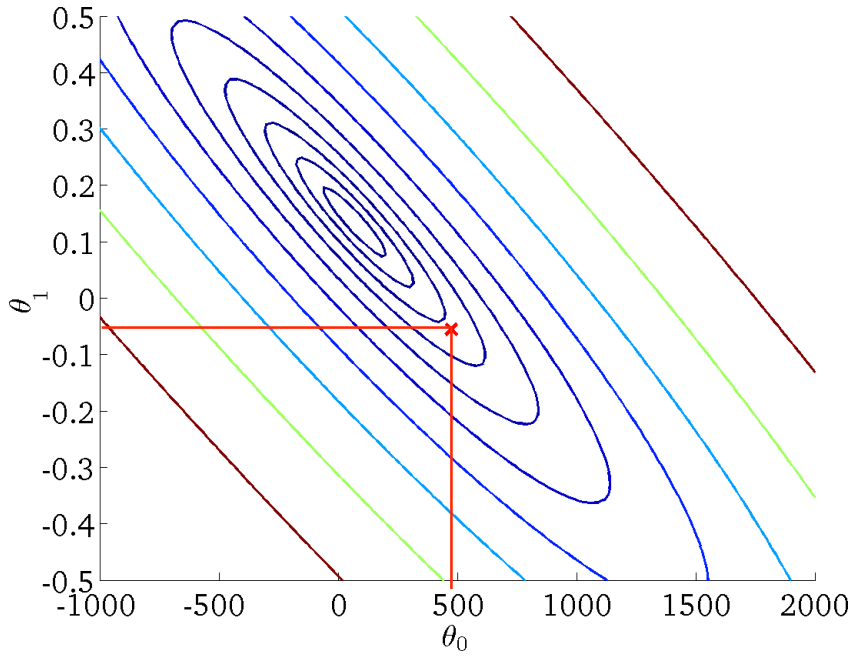
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

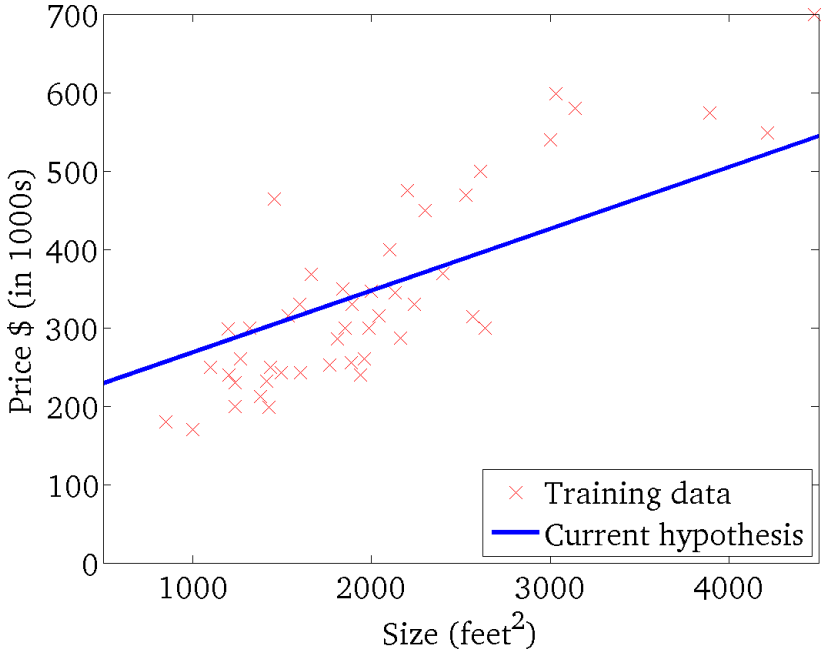
(function of the parameters θ_0, θ_1)



Intuition Behind Cost Function

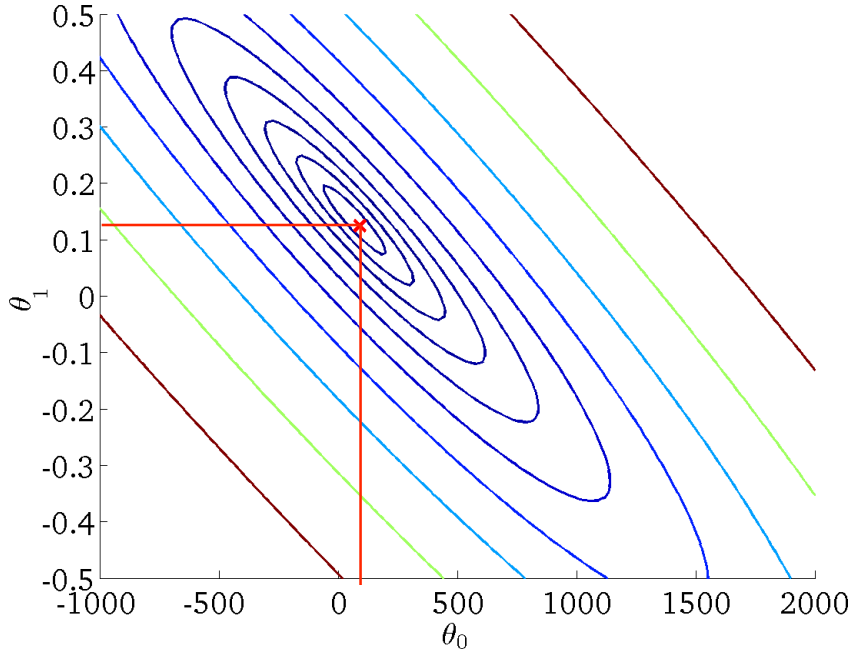
$h_\theta(x)$

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



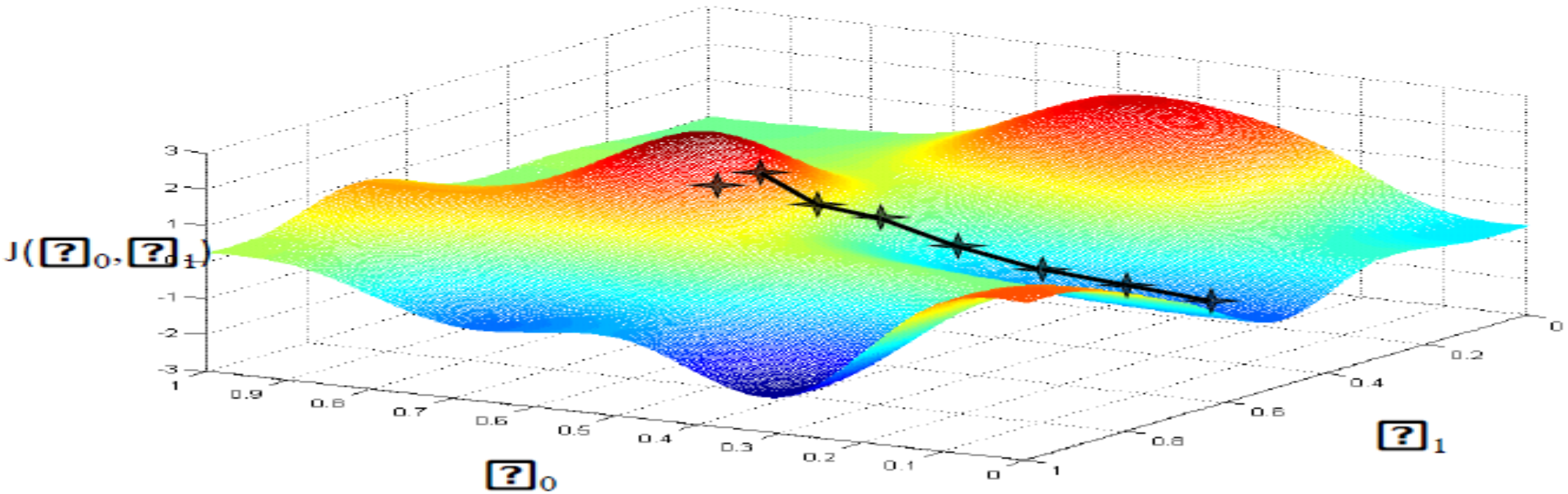
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)



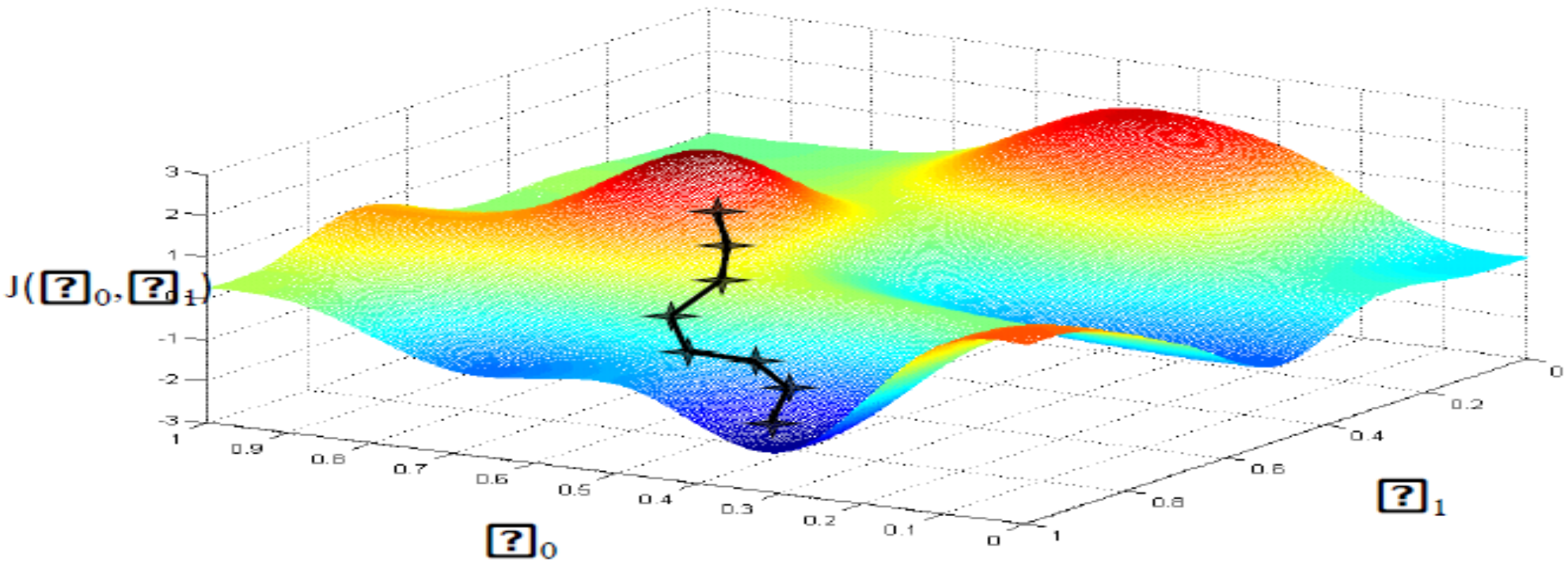
Basic Search Procedure

- Choose initial value for θ
- Until we reach a minimum:
  - Choose a new value for θ to reduce $J(\theta)$



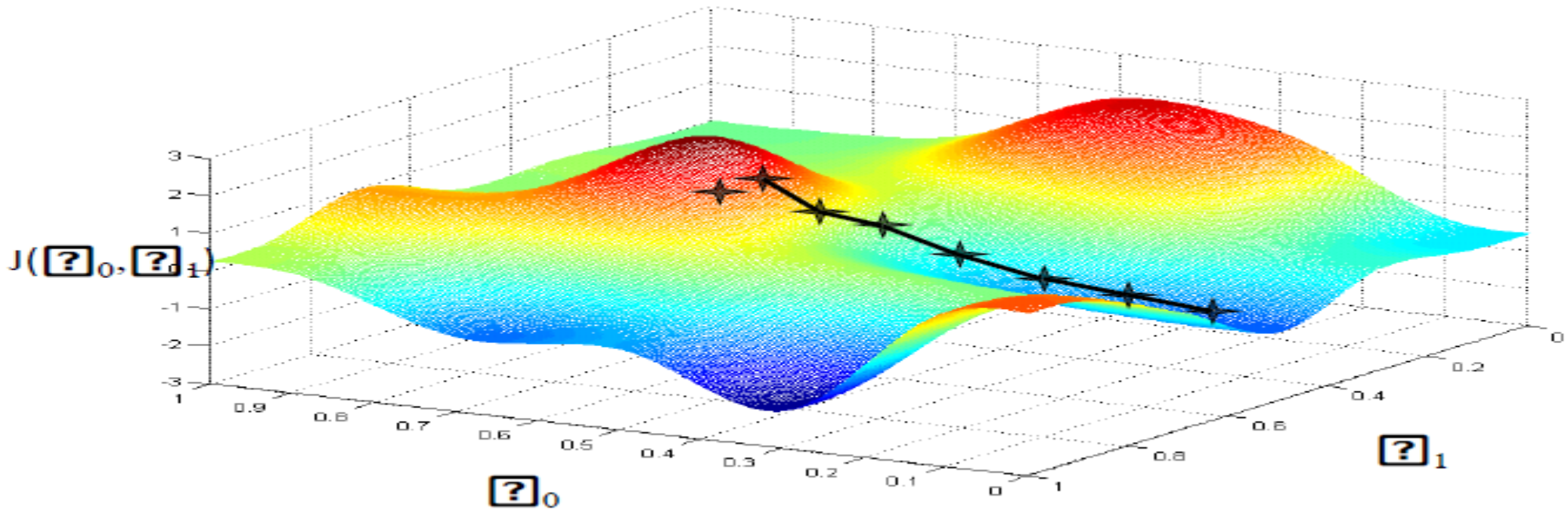
Basic Search Procedure

- Choose initial value for θ
- Until we reach a minimum:
  - Choose a new value for θ to reduce $J(\theta)$



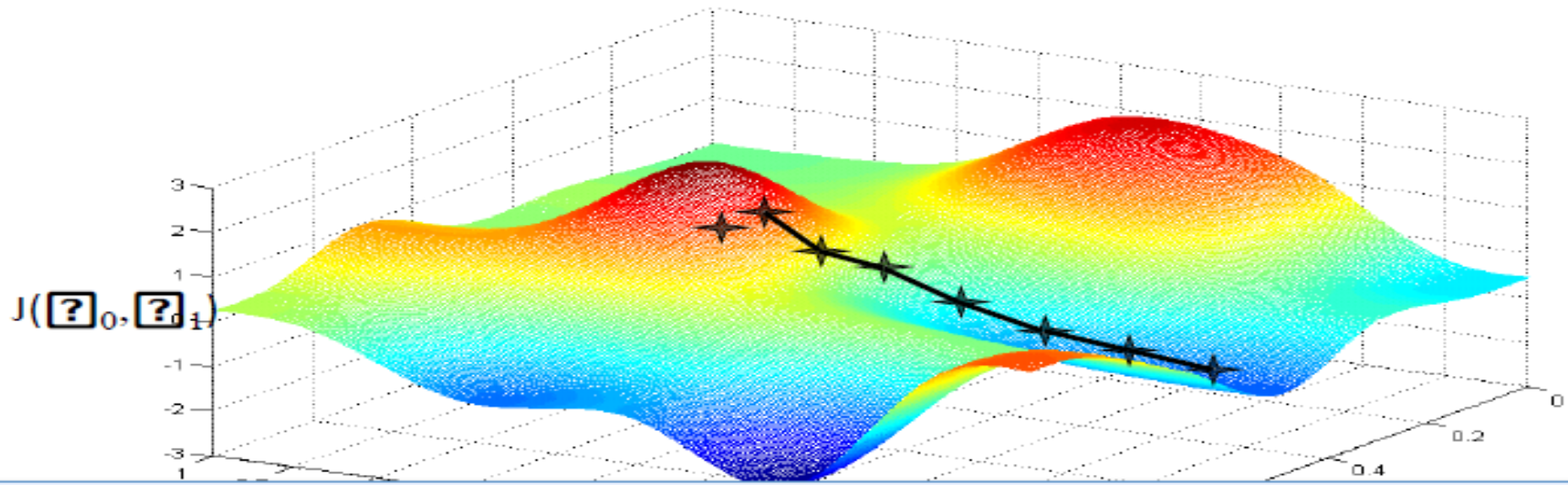
Basic Search Procedure

- Choose initial value for θ
- Until we reach a minimum:
  - Choose a new value for θ to reduce $J(\theta)$



Basic Search Procedure

- Choose initial value for θ
- Until we reach a minimum:
  - Choose a new value for θ to reduce $J(\theta)$



Since the least squares objective function is convex (concave), we don't need to worry about local minima

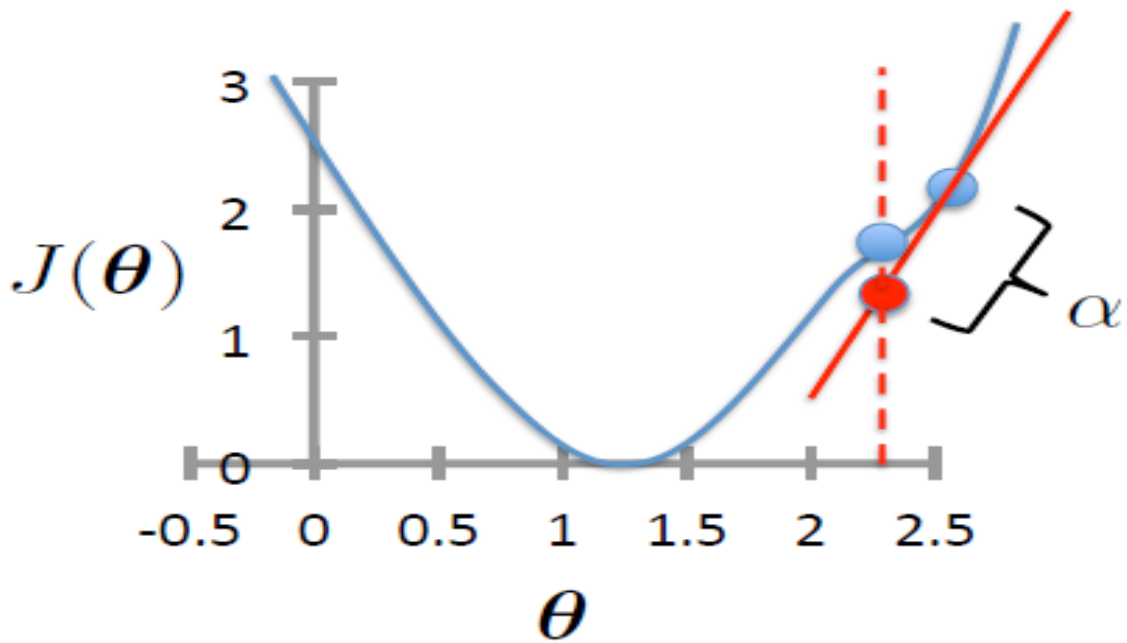
Gradient Descent

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

simultaneous update
for j = 0 ... d

learning rate (small)
e.g., $\alpha = 0.05$



Gradient Descent

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

simultaneous update
for $j = 0 \ldots d$

For Linear Regression:

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta}) &= \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} \right) - y^{(i)} \right)^2 \\
&= \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right)^2 \\
&= \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) \times \frac{\partial}{\partial \theta_j} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) \\
&= \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) x_j^{(i)}
\end{aligned}$$

Gradient Descent for Linear Regression

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{1}{n} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} \right) - y^{(i)} \right) x_j^{(i)}$$

simultaneous update for $j = 0 \ldots d$

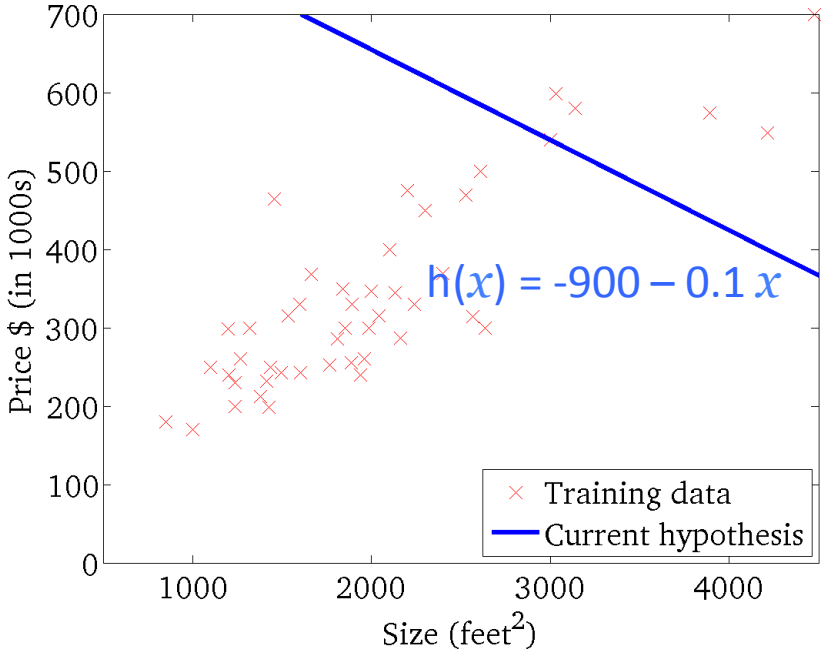
- To achieve simultaneous update
  - At the start of each GD iteration, compute $h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} \right)$
  - Use this stored value in the update step loop
- Assume convergence when $\|\boldsymbol{\theta}_{new} - \boldsymbol{\theta}_{old}\|_2 < \epsilon$

L_2 norm: $$\|\boldsymbol{v}\|_2 = \sqrt{\sum_i v_i^2} = \sqrt{v_1^2 + v_2^2 + \ldots + v_{|v|}^2}$$

Gradient Descent

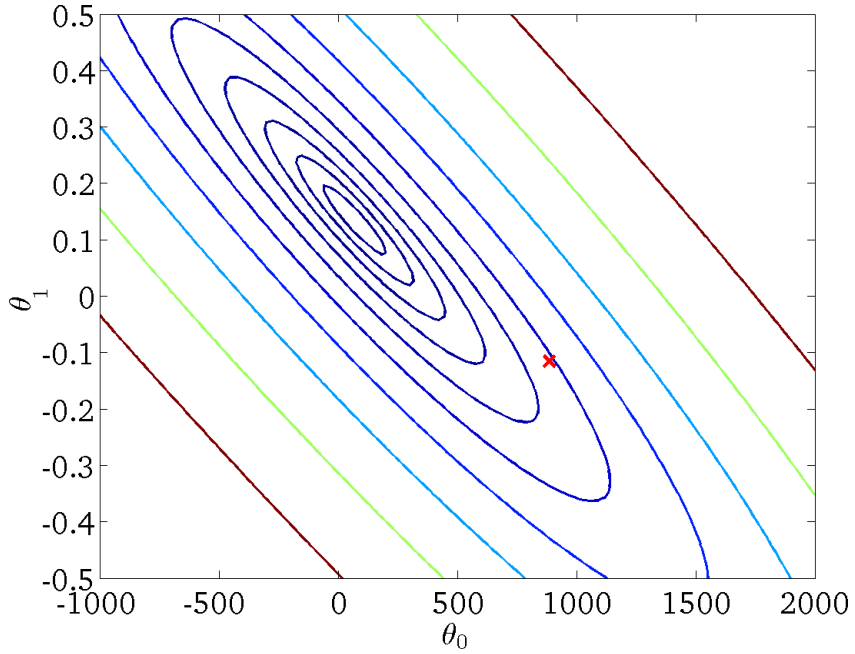
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

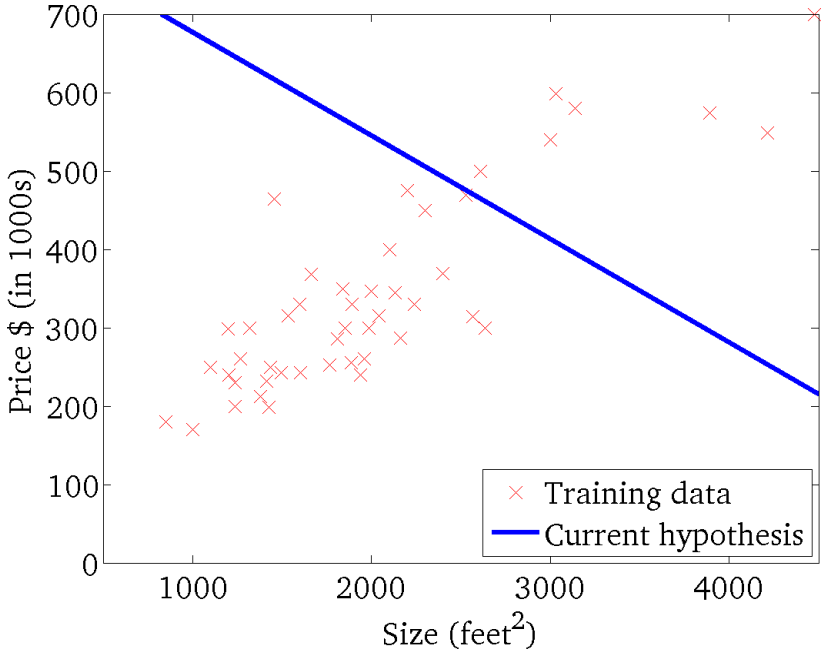
(function of the parameters θ_0, θ_1)



Gradient Descent

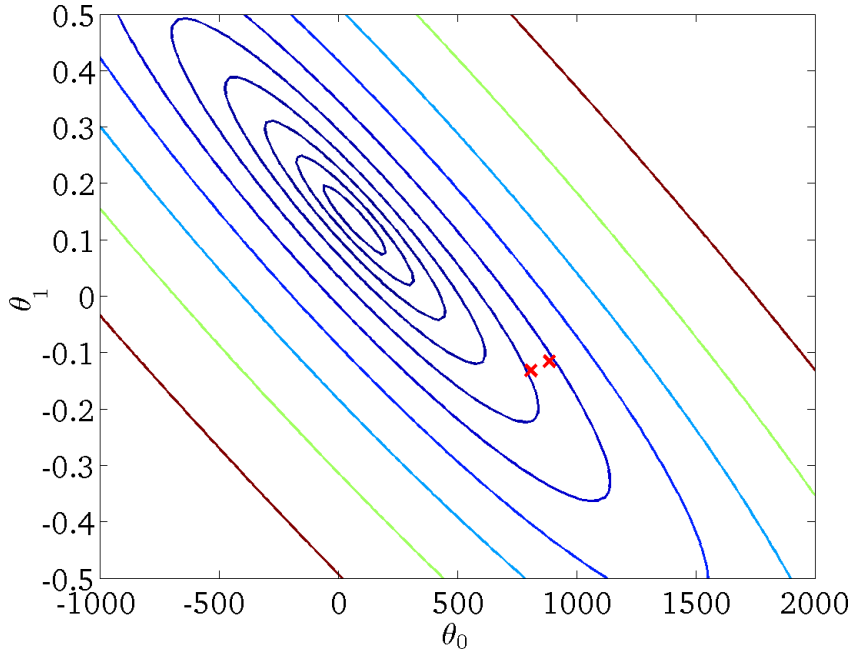
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

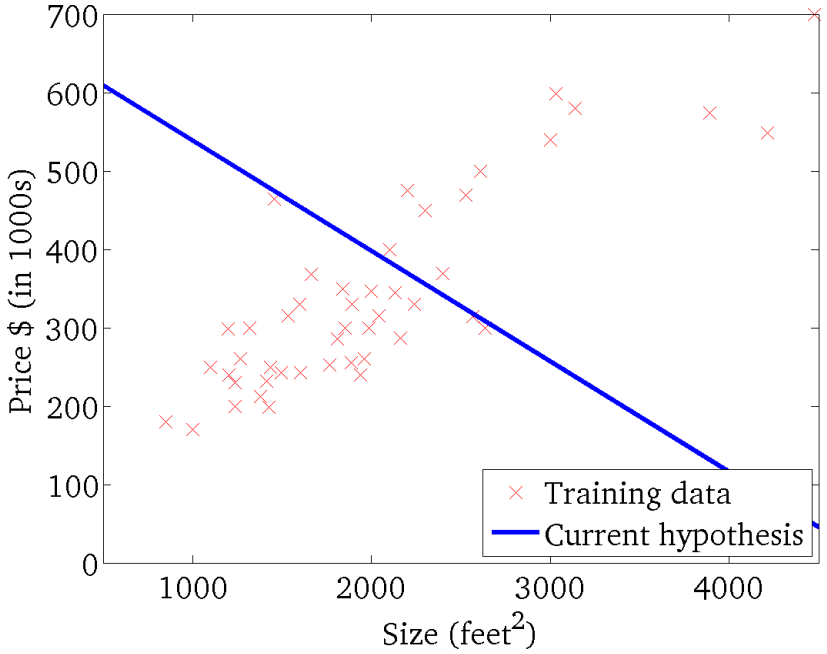
(function of the parameters θ_0, θ_1)



Gradient Descent

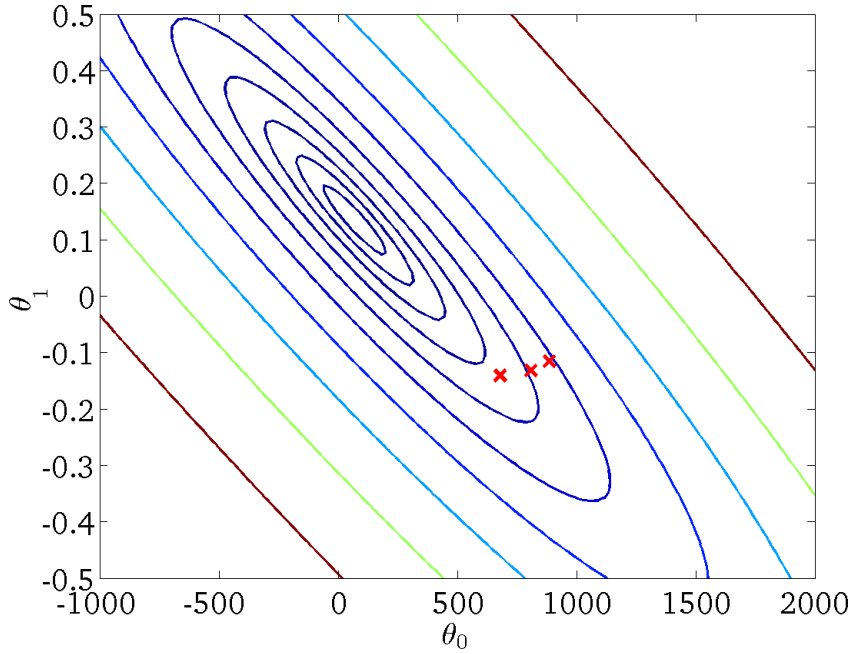
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

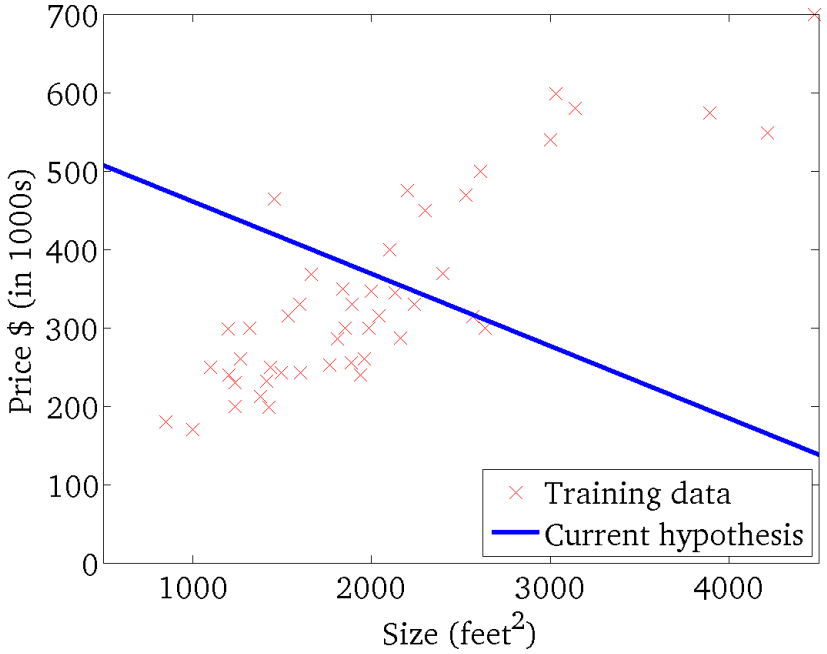
(function of the parameters θ_0, θ_1)



Gradient Descent

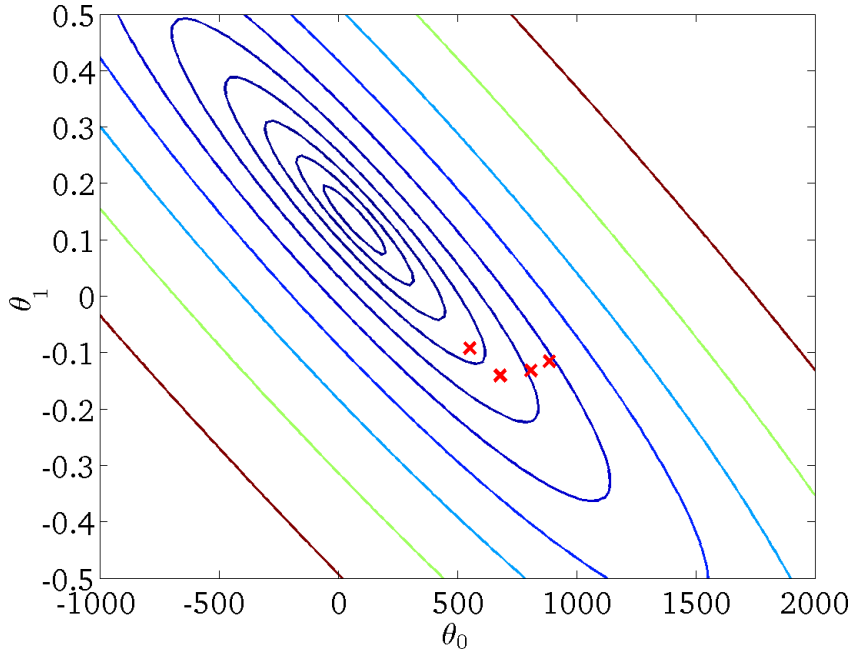
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

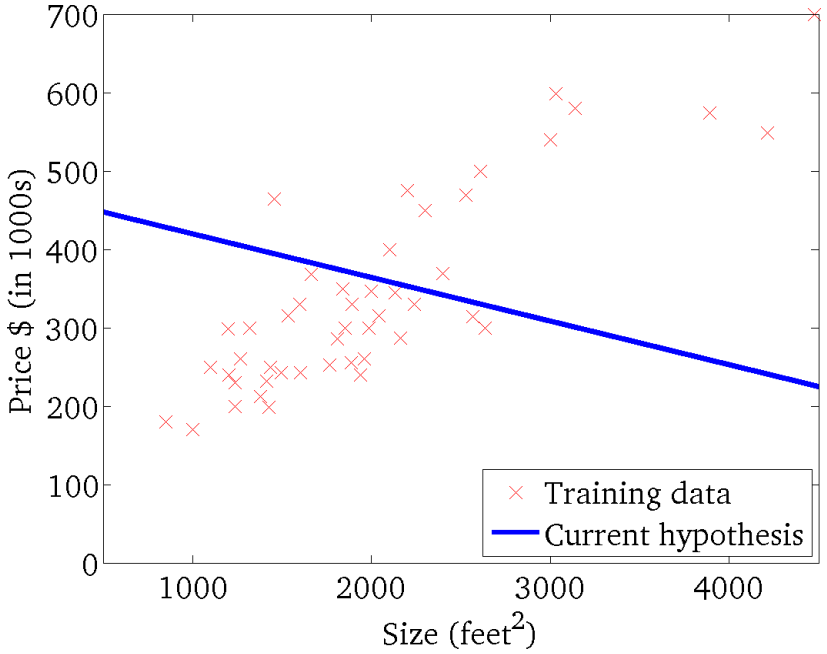
(function of the parameters θ_0, θ_1)



Gradient Descent

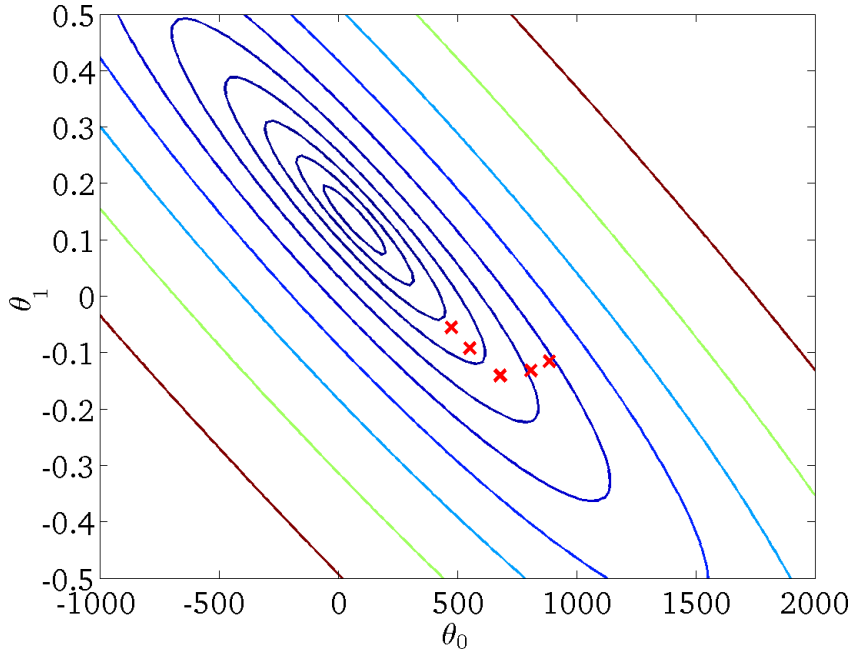
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

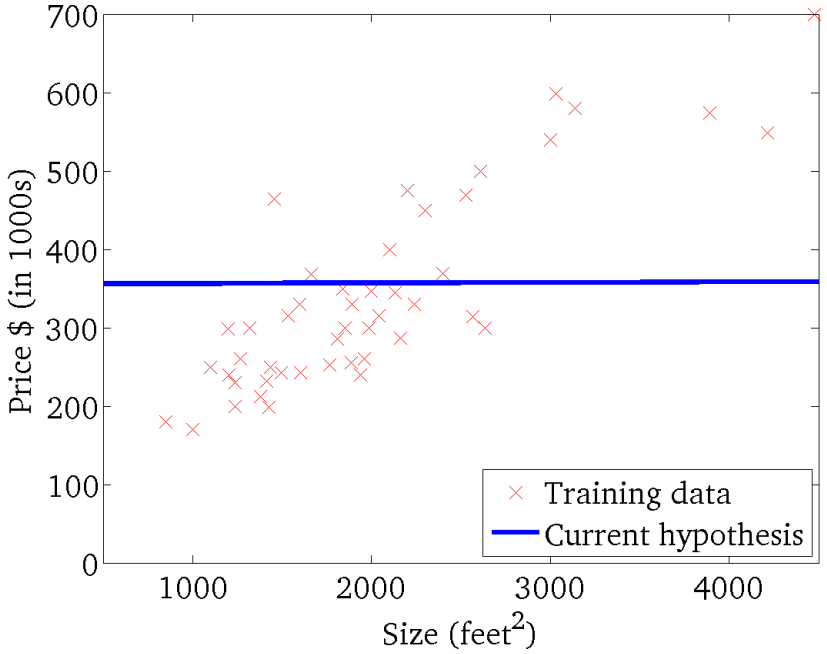
(function of the parameters θ_0, θ_1)



Gradient Descent

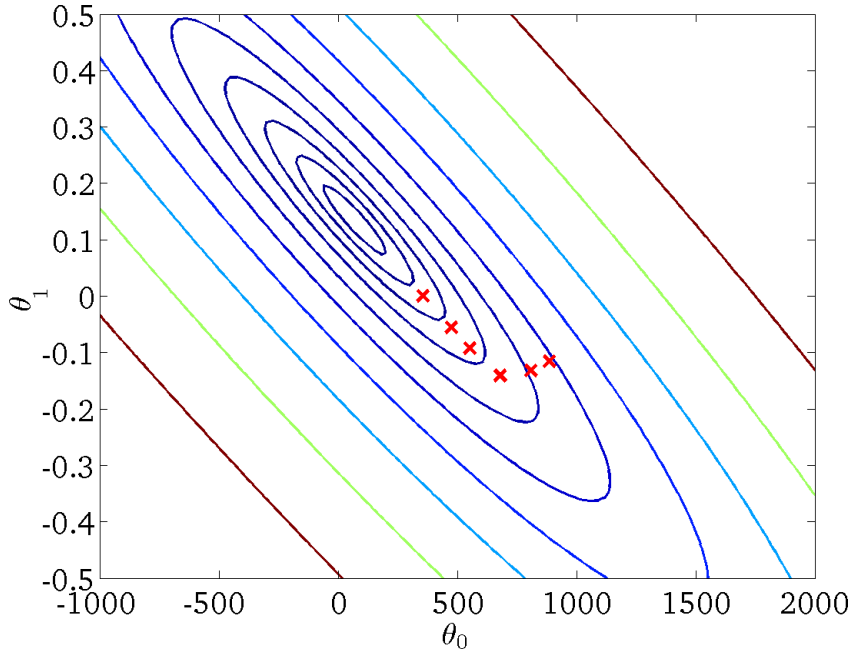
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

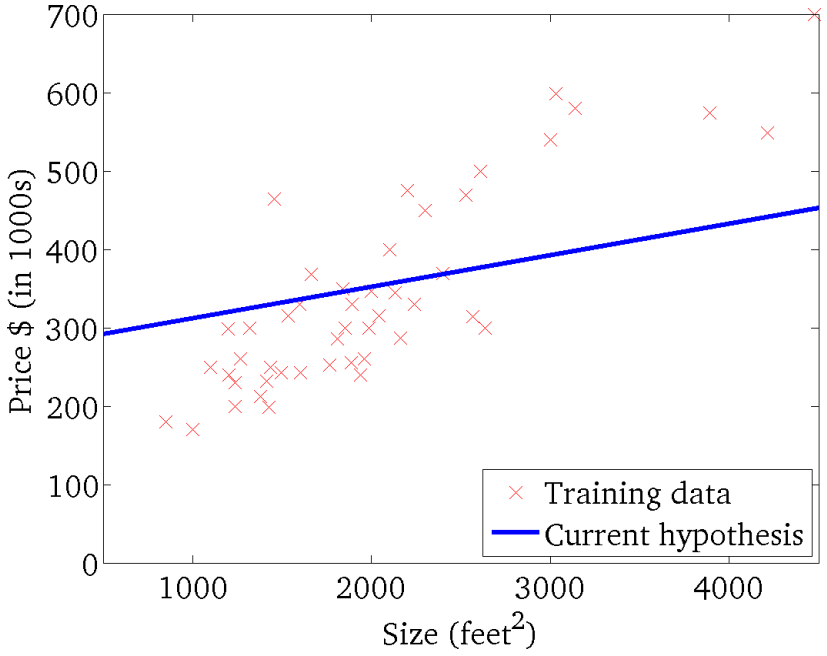
(function of the parameters θ_0, θ_1)



Gradient Descent

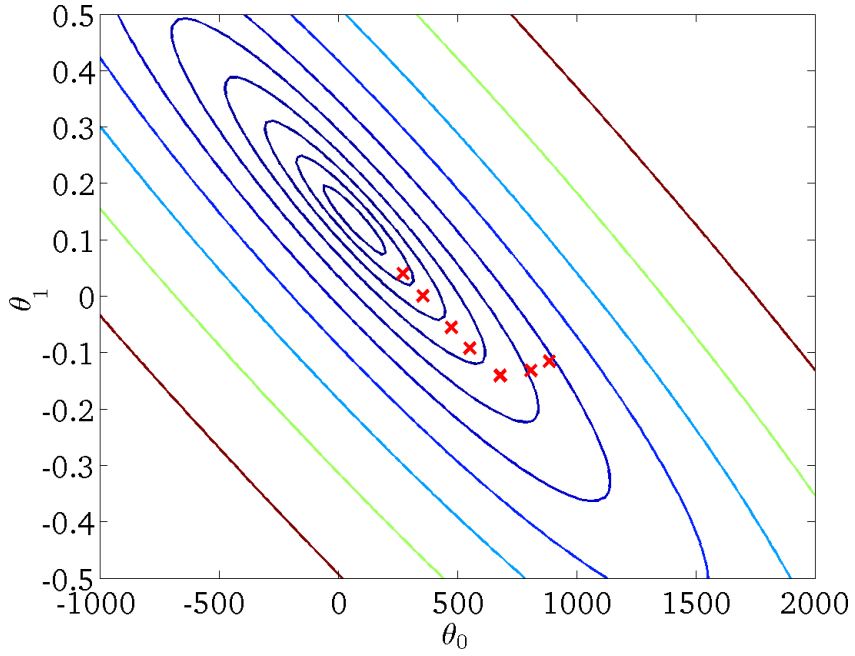
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

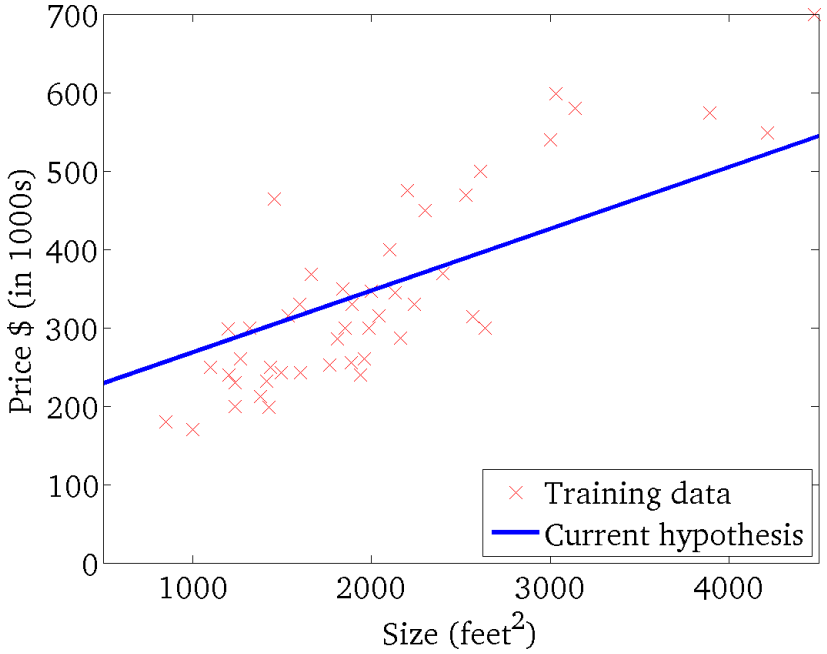
(function of the parameters θ_0, θ_1)



Gradient Descent

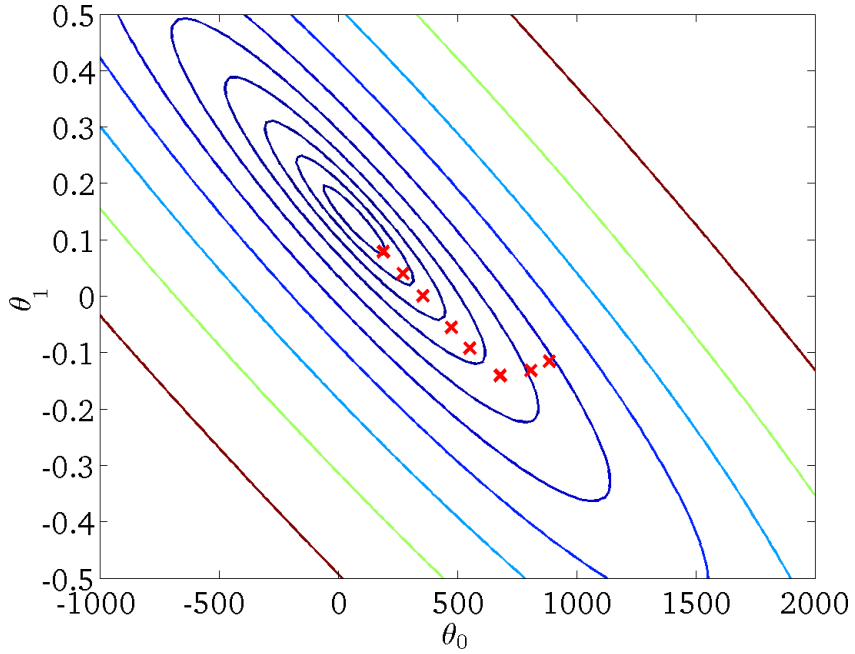
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

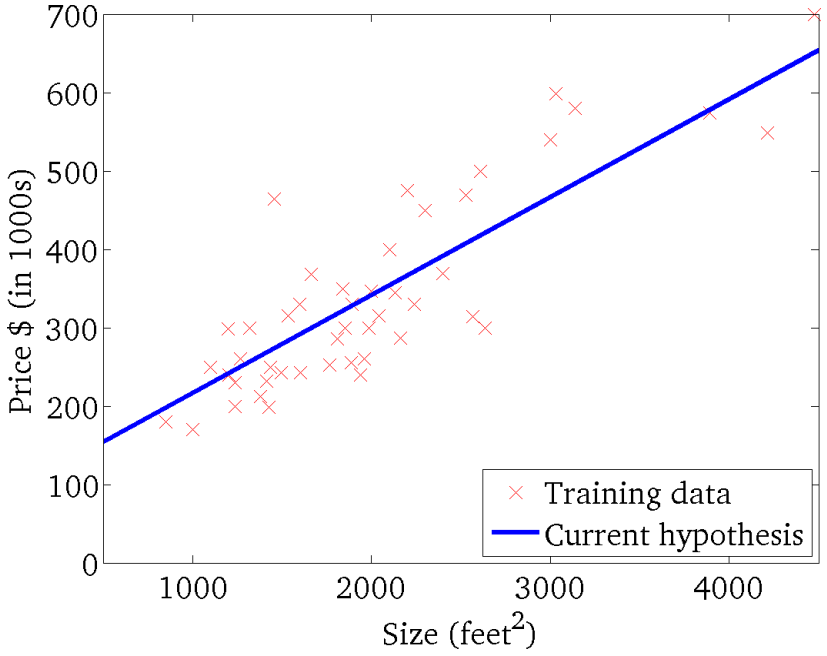
(function of the parameters θ_0, θ_1)



Gradient Descent

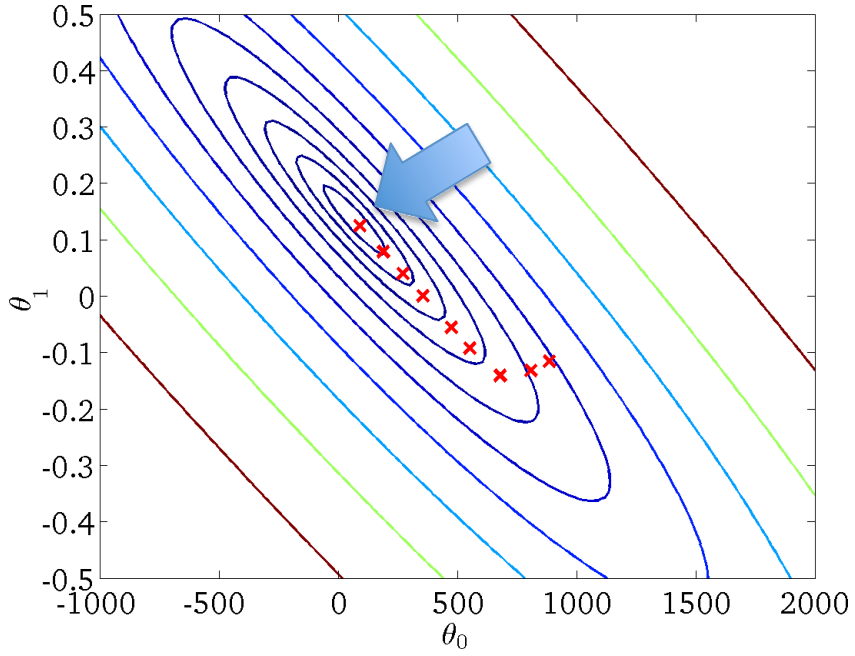
$h_\theta(x)$

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



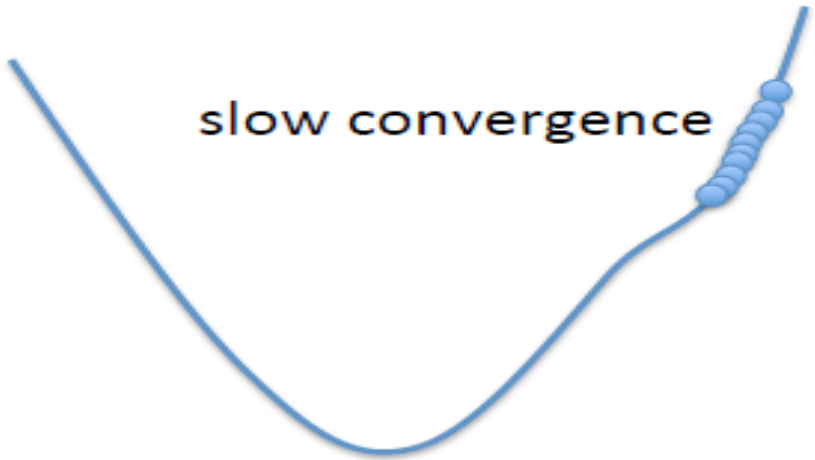
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

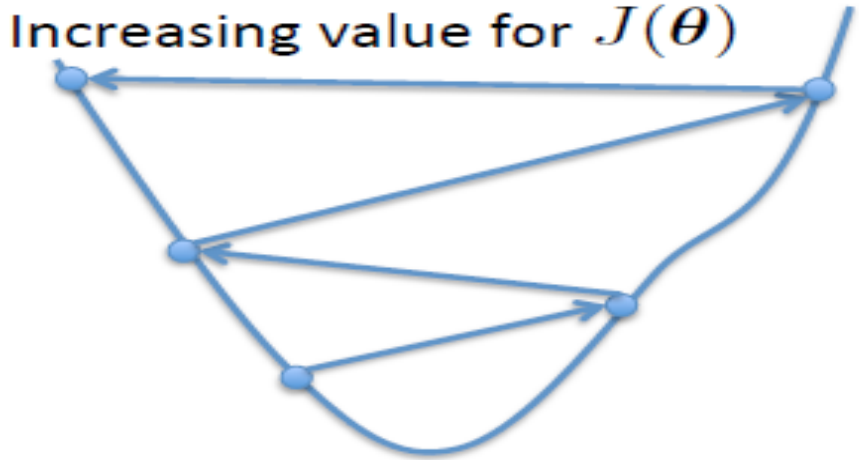


Choosing α

α too small



α too large



- May overshoot the minimum
- May fail to converge
- May even diverge

To see if gradient descent is working, print out $J(\theta)$ each iteration

- The value should decrease at each iteration
- If it doesn't, adjust α

Extending Linear Regression to More Complex Models

- The inputs **X** for linear regression can be:
  - Original quantitative inputs
  - Transformation of quantitative inputs
    - e.g. log, exp, square root, square, etc.
  - Polynomial transformation
    - example: $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$
  - Basis expansions
  - Dummy coding of categorical inputs
  - Interactions between variables
    - example: $x_3 = x_1 \cdot x_2$

This allows use of linear regression techniques to fit non-linear datasets.

Linear Basis Function Models

- Generally,

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=0}^{d} \theta_j \underbrace{\phi_j(\boldsymbol{x})}_{\text{basis function}}$$

- Typically, $\phi_0(\boldsymbol{x}) = 1$ so that θ_0 acts as a bias
- In the simplest case, we use linear basis functions :

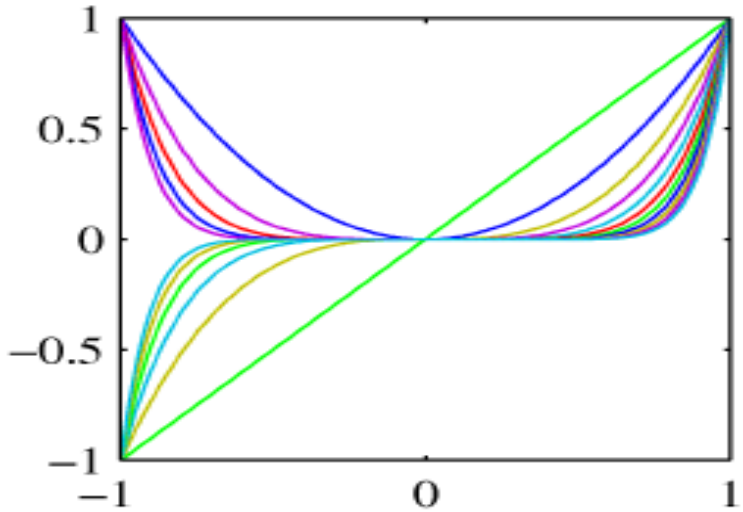
$$\phi_j(\boldsymbol{x}) = x_j$$

Linear Basis Function Models

- Polynomial basis functions:

$$\phi_j(x) = x^j$$

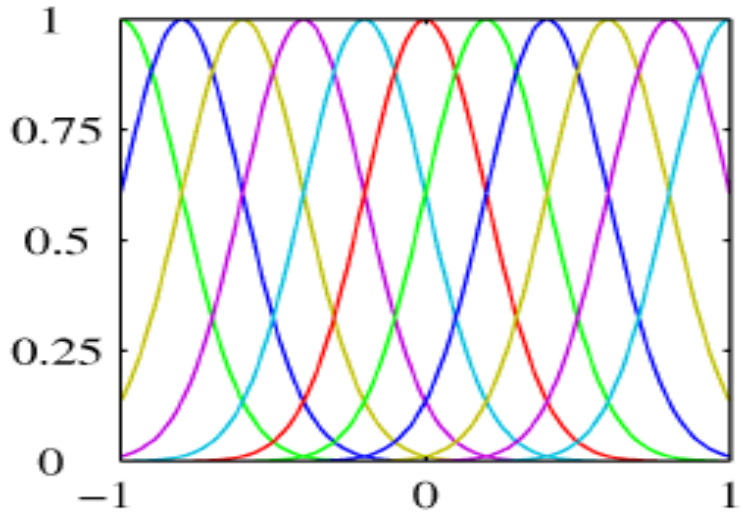
  - These are global; a small change in x affects all basis functions



- Gaussian basis functions:

$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\}$$

  - These are local; a small change in x only affect nearby basis functions. μ_j and s control location and scale (width).



Linear Basis Function Models

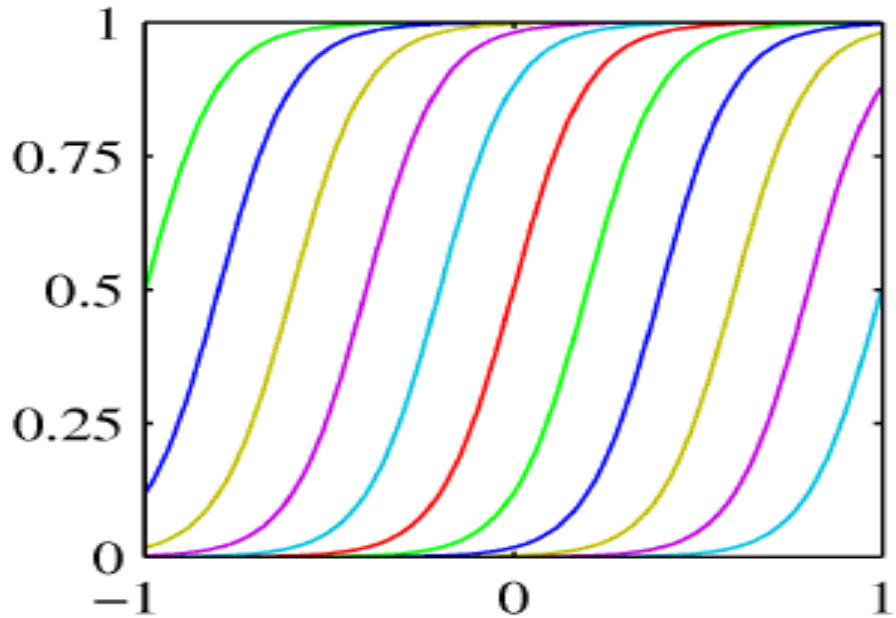
- Sigmoidal basis functions:

$$\phi_j(x) = \sigma\left(\frac{x - \mu_j}{s}\right)$$

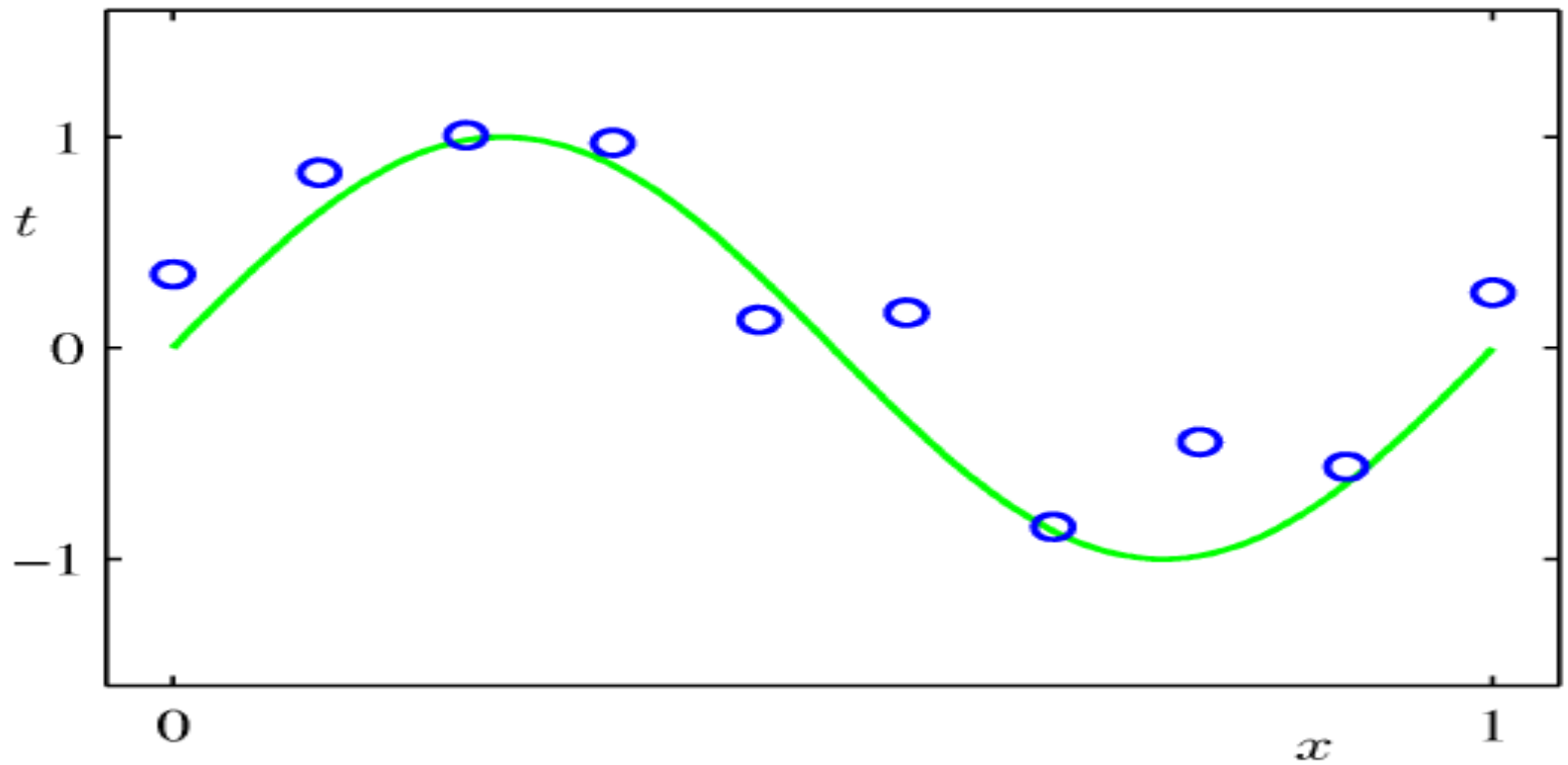
where

$$\sigma(a) = \frac{1}{1 + \exp(-a)}$$

  - These are also local; a small change in x only affects nearby basis functions. μ_j and s control location and scale (slope).



Example of Fitting a Polynomial Curve with a Linear Model



$$y = \theta_0 + \theta_1 x + \theta_2 x^2 + \ldots + \theta_p x^p = \sum_{j=0}^{p} \theta_j x^j$$

Linear Basis Function Models

- Basic Linear Model: $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=0}^{d} \theta_j x_j$
- Generalized Linear Model: $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \sum_{j=0}^{d} \theta_j \phi_j(\boldsymbol{x})$
- Once we have replaced the data by the outputs of the basis functions, fitting the generalized model is exactly the same problem as fitting the basic model
  - Unless we use the kernel trick – more on that when we cover support vector machines
  - Therefore, there is no point in cluttering the math with basis functions

Linear Algebra Concepts

- *Vector* in $\mathbb{R}^d$ is an ordered set of d real numbers
  - e.g., v = [1,6,3,4] is in $\mathbb{R}^4$
  - "[1,6,3,4]" is a column vector:
  - as opposed to a row vector:

$$\begin{pmatrix} 1 \\ 6 \\ 3 \\ 4 \end{pmatrix}$$

$$\begin{pmatrix} 1 & 6 & 3 & 4 \end{pmatrix}$$

- An m-by-n *matrix* is an object with m rows and n columns, where each entry is a real number:

$$\begin{pmatrix} 1 & 2 & 8 \\ 4 & 78 & 6 \\ 9 & 3 & 2 \end{pmatrix}$$

Linear Algebra Concepts

- Transpose: reflect vector/matrix on line:

$$\begin{pmatrix} a \\ b \end{pmatrix}^T = \begin{pmatrix} a & b \end{pmatrix} \qquad \begin{pmatrix} a & b \\ c & d \end{pmatrix}^T = \begin{pmatrix} a & c \\ b & d \end{pmatrix}$$

  - Note: $(Ax)^T = x^T A^T$ (We'll define multiplication soon...)

- Vector norms:
  - L_p norm of $v = (v_1,\ldots,v_k)$ is $\left(\sum_i |v_i|^p\right)^{\frac{1}{p}}$
  - Common norms: L_1, L_2
  - $L_{infinity} = \max_i |v_i|$
- Length of a vector v is $L_2(v)$

Linear Algebra Concepts

- Vector dot product: $u \bullet v = (u_1 \quad u_2) \bullet (v_1 \quad v_2) = u_1 v_1 + u_2 v_2$
  - Note: dot product of u with itself = length$(u)^2 = \|u\|_2^2$
- Matrix product:

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

$$AB = \begin{pmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{pmatrix}$$

Linear Algebra Concepts

- Vector products:
  - Dot product: $u \bullet v = u^T v = \begin{pmatrix} u_1 & u_2 \end{pmatrix} \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = u_1 v_1 + u_2 v_2$
  - Outer product:

$$uv^T = \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \begin{pmatrix} v_1 & v_2 \end{pmatrix} = \begin{pmatrix} u_1 v_1 & u_1 v_2 \\ u_2 v_1 & u_2 v_2 \end{pmatrix}$$

Vectorization

- Benefits of vectorization
  - More compact equations
  - Faster code (using optimized matrix libraries)
- Consider our model:

$$h(\boldsymbol{x}) = \sum_{j=0}^{d} \theta_j x_j$$

- Let

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \qquad \boldsymbol{x}^\intercal = \begin{bmatrix} 1 & x_1 & \dots & x_d \end{bmatrix}$$

- Can write the model in vectorized form as $h(\boldsymbol{x}) = \boldsymbol{\theta}^\intercal \boldsymbol{x}$

Vectorization

- Consider our model for n instances:

$$h\left(\boldsymbol{x}^{(i)}\right) = \sum_{j=0}^{d} \theta_j x_j^{(i)}$$

- Let

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \quad \boldsymbol{X} = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_d^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(i)} & \dots & x_d^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & \dots & x_d^{(n)} \end{bmatrix}$$

$\mathbb{R}^{(d+1)\times 1}$ $\mathbb{R}^{n\times(d+1)}$

- Can write the model in vectorized form as $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{X}\boldsymbol{\theta}$

Vectorization

- For the linear regression cost function:

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2$$

$$= \frac{1}{2n}\sum_{i=1}^{n}\left(\boldsymbol{\theta}^{\intercal}\boldsymbol{x}^{(i)} - y^{(i)}\right)^2$$

$$= \frac{1}{2n}\underbrace{(\boldsymbol{X\theta} - \boldsymbol{y})^{\intercal}}_{\mathbb{R}^{1\times n}}\underbrace{(\boldsymbol{X\theta} - \boldsymbol{y})}_{\mathbb{R}^{n\times 1}}$$

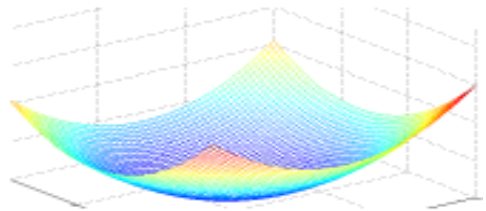
$\boldsymbol{X}$: $\mathbb{R}^{n\times(d+1)}$, $\boldsymbol{\theta}$: $\mathbb{R}^{(d+1)\times 1}$

Let:

$$\boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

Closed Form Solution

- Instead of using GD, solve for optimal $\boldsymbol{\theta}$ analytically
  - Notice that the solution is when $\frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta}) = 0$



- Derivation:

$$\mathcal{J}(\boldsymbol{\theta}) = \frac{1}{2n} (\boldsymbol{X\theta} - \boldsymbol{y})^\intercal (\boldsymbol{X\theta} - \boldsymbol{y})$$

1 x 1

$$\propto \boldsymbol{\theta}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X\theta} - \boxed{\boldsymbol{y}^\intercal \boldsymbol{X\theta}} - \boxed{\boldsymbol{\theta}^\intercal \boldsymbol{X}^\intercal \boldsymbol{y}} + \boldsymbol{y}^\intercal \boldsymbol{y}$$

$$\propto \boldsymbol{\theta}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X\theta} - 2\boldsymbol{\theta}^\intercal \boldsymbol{X}^\intercal \boldsymbol{y} + \boldsymbol{y}^\intercal \boldsymbol{y}$$

Take derivative and set equal to 0, then solve for $\boldsymbol{\theta}$:

$$\frac{\partial}{\partial \boldsymbol{\theta}} (\boldsymbol{\theta}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X\theta} - 2\boldsymbol{\theta}^\intercal \boldsymbol{X}^\intercal \boldsymbol{y} + \cancel{\boldsymbol{y}^\intercal \boldsymbol{y}}) = 0$$

$$(\boldsymbol{X}^\intercal \boldsymbol{X})\boldsymbol{\theta} - \boldsymbol{X}^\intercal \boldsymbol{y} = 0$$

$$(\boldsymbol{X}^\intercal \boldsymbol{X})\boldsymbol{\theta} = \boldsymbol{X}^\intercal \boldsymbol{y}$$

Closed Form Solution: $\boldsymbol{\theta} = (\boldsymbol{X}^\intercal \boldsymbol{X})^{-1} \boldsymbol{X}^\intercal \boldsymbol{y}$

Closed Form Solution

- Can obtain $\boldsymbol{\theta}$ by simply plugging $\boldsymbol{X}$ and $\boldsymbol{y}$ into

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

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_d^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(i)} & \dots & x_d^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & \dots & x_d^{(n)} \end{bmatrix} \qquad \boldsymbol{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

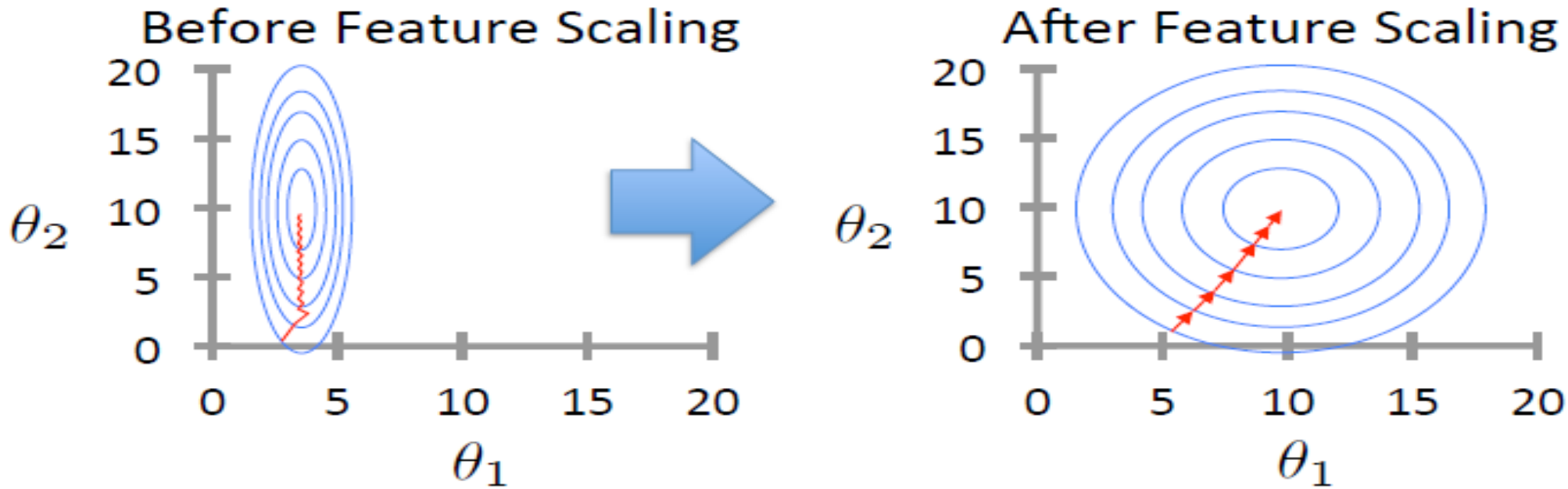
- If $\boldsymbol{X}^\intercal \boldsymbol{X}$ is not invertible (i.e., singular), may need to:
  - Use pseudo-inverse instead of the inverse
    - In python, `numpy.linalg.pinv(a)`
  - Remove redundant (not linearly independent) features
  - Remove extra features to ensure that $d \leq n$

Gradient Descent vs Closed Form

Gradient Descent	Closed Form Solution
• Requires multiple iterations • Need to choose α • Works well when n is large • Can support incremental learning	• Non-iterative • No need for α • Slow if n is large – Computing $(\boldsymbol{X}^{\mathsf{T}}\boldsymbol{X})^{-1}$ is roughly O(n^3)

Improving Learning: Feature Scaling

- **Idea:** Ensure that feature have similar scales



- Makes gradient descent converge *much* faster

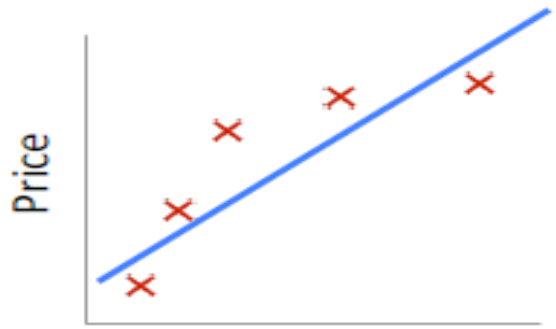
Feature Standardization

- Rescales features to have zero mean and unit variance
  - Let μ_j be the mean of feature j: $\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_j^{(i)}$
  - Replace each value with:

$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j} \qquad \text{for } j = 1 \ldots d \text{ (not } x_0\text{!)}$$

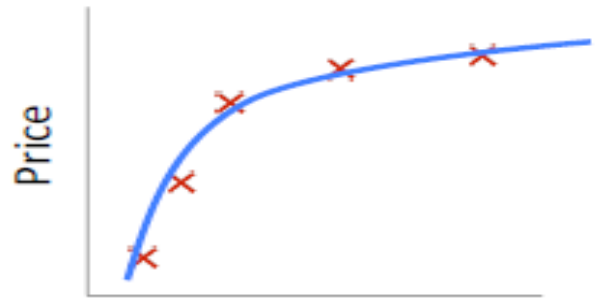
    - s_j is the standard deviation of feature j
    - Could also use the range of feature j ($\max_j - \min_j$) for s_j
- Must apply the same transformation to instances for both training and prediction
- Outliers can cause problems

Quality of Fit



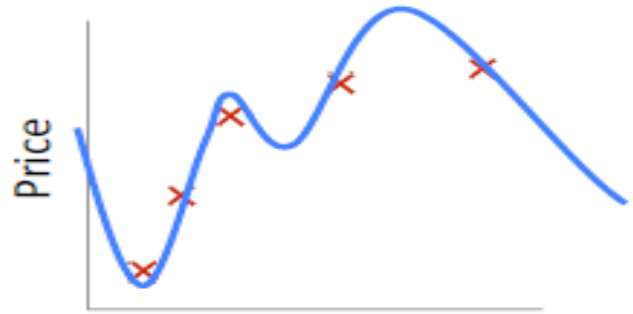
$\theta_0 + \theta_1 x$

Underfitting
(high bias)



$\theta_0 + \theta_1 x + \theta_2 x^2$

Correct fit



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Overfitting
(high variance)

Overfitting:

- The learned hypothesis may fit the training set very well ($J(\boldsymbol{\theta}) \approx 0$)
- ...but fails to generalize to new examples

Regularization

- A method for automatically controlling the complexity of the learned hypothesis
- **Idea**: penalize for large values of θ_j
  - Can incorporate into the cost function
  - Works well when we have a lot of features, each that contributes a bit to predicting the label
- Can also address overfitting by eliminating features (either manually or via model selection)

Regularization

- Linear regression objective function

$$J(\boldsymbol{\theta}) = \underbrace{\frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2}_{\text{model fit to data}} + \underbrace{\frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2}_{\text{regularization}}$$

  - λ is the regularization parameter ($\lambda \geq 0$)
  - No regularization on θ_0!

Understanding Regularization

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- Note that $\sum_{j=1}^{d}\theta_j^2 = \|\boldsymbol{\theta}_{1:d}\|_2^2$
  - This is the magnitude of the feature coefficient vector!

- We can also think of this as:

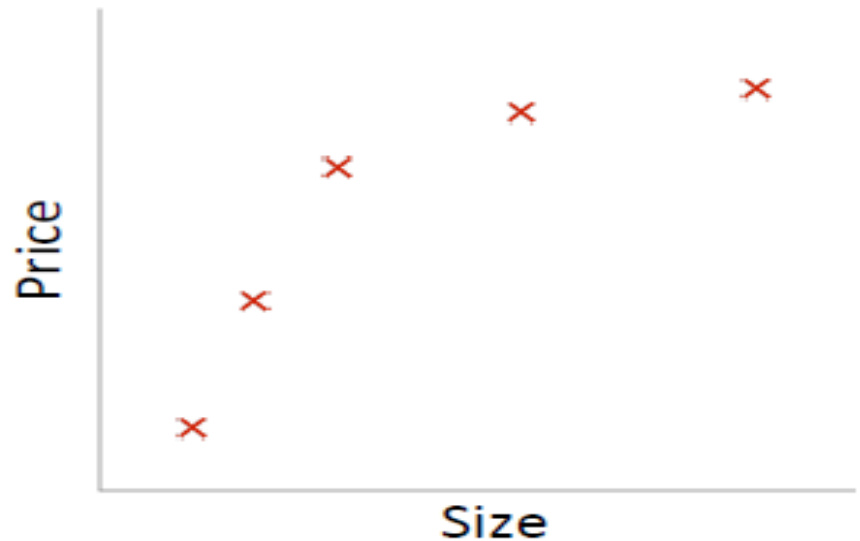
$$\sum_{j=1}^{d}(\theta_j - 0)^2 = \|\boldsymbol{\theta}_{1:d} - \vec{\mathbf{0}}\|_2^2$$

  - L_2 regularization pulls coefficients toward 0

Understanding Regularization

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- What happens if we set λ to be huge (e.g., 10^{10})?

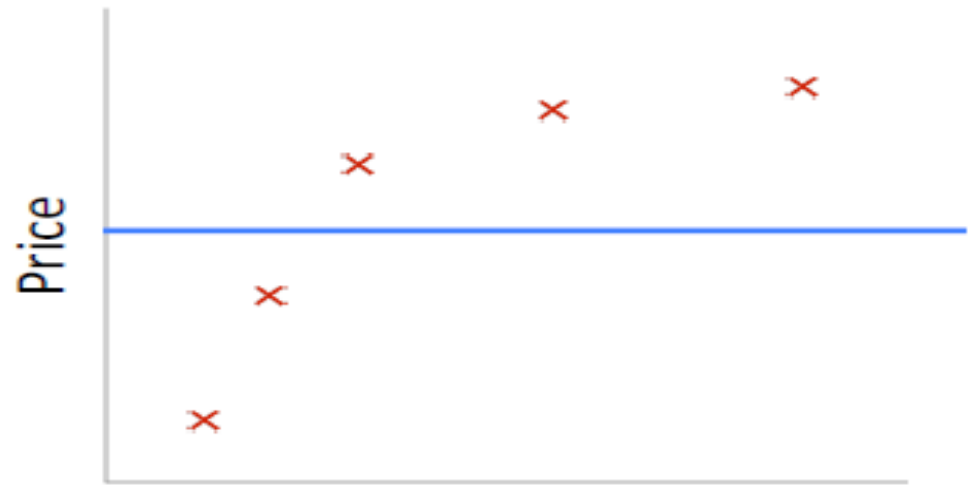


$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Understanding Regularization

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

- What happens if we set λ to be huge (e.g., 10^{10})?



$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

0 0 0 0

Regularized Linear Regression

- Cost Function

$$J(\boldsymbol{\theta}) = \frac{1}{2n} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} \right) - y^{(i)} \right)^2 + \frac{\lambda}{2} \sum_{j=1}^{d} \theta_j^2$$

- Fit by solving $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

- Gradient update:

$\frac{\partial}{\partial \theta_0} J(\theta)$

$$\theta_0 \leftarrow \theta_0 - \alpha \frac{1}{n} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} \right) - y^{(i)} \right)$$

$\frac{\partial}{\partial \theta_j} J(\theta)$

$$\theta_j \leftarrow \theta_j - \alpha \frac{1}{n} \sum_{i=1}^{n} \left(h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} \right) - y^{(i)} \right) x_j^{(i)} \underbrace{- \alpha \lambda \theta_j}_{\text{regularization}}$$

Regularized Linear Regression

$$J(\boldsymbol{\theta}) = \frac{1}{2n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)^2 + \frac{\lambda}{2}\sum_{j=1}^{d}\theta_j^2$$

$$\theta_0 \leftarrow \theta_0 - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)$$

$$\theta_j \leftarrow \theta_j - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)} - \alpha\lambda\theta_j$$

- We can rewrite the gradient step as:

$$\theta_j \leftarrow \theta_j\left(1 - \alpha\lambda\right) - \alpha\frac{1}{n}\sum_{i=1}^{n}\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)x_j^{(i)}$$

Regularized Linear Regression

- To incorporate regularization into the closed form solution:

$$\boldsymbol{\theta} = \left(\boldsymbol{X}^\intercal \boldsymbol{X} + \lambda \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} \right)^{-1} \boldsymbol{X}^\intercal \boldsymbol{y}$$

- Can derive this the same way, by solving $\frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta}) = 0$
- Can prove that for $\lambda > 0$, inverse exists in the equation above