

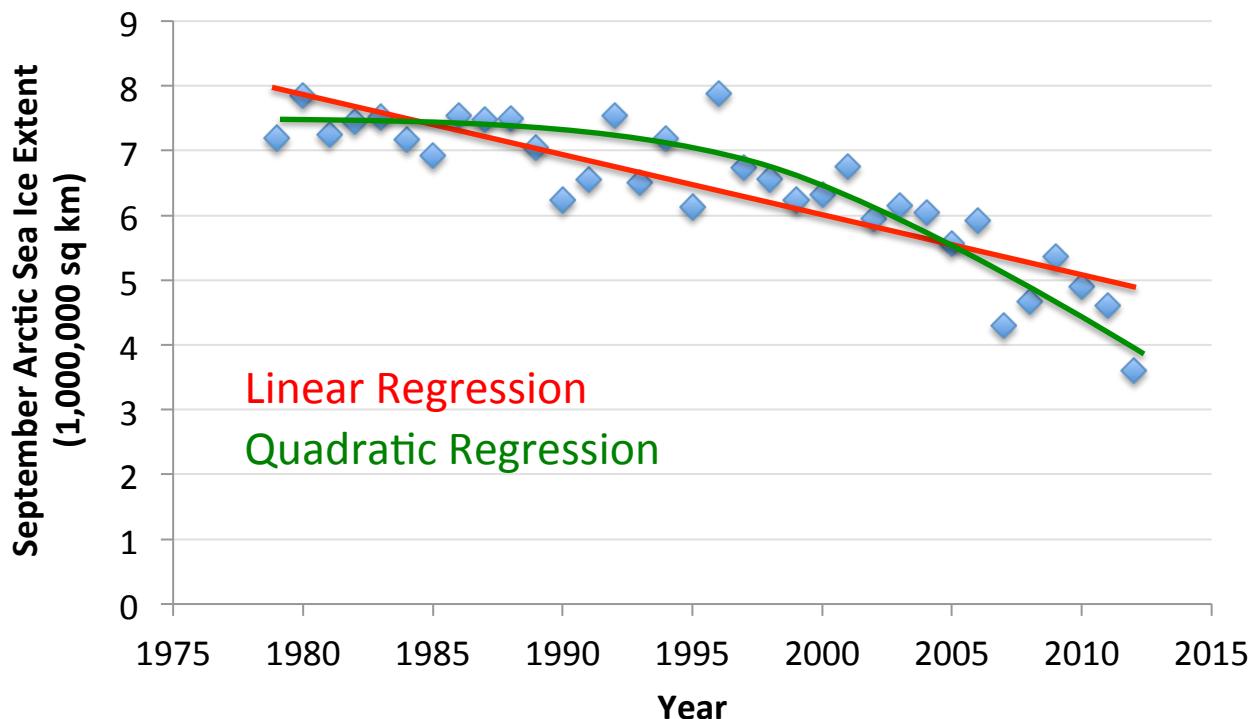
Linear Regression



Regression

Given:

- Data $\mathbf{X} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(n)}\}$ where $\mathbf{x}^{(i)} \in \mathbb{R}^d$
- Corresponding labels $\mathbf{y} = \{y^{(1)}, \dots, y^{(n)}\}$ where $y^{(i)} \in \mathbb{R}$



Prostate Cancer Dataset

- 97 samples, partitioned into 67 train / 30 test
- Eight predictors (features):
 - 6 continuous (4 log transforms), 1 binary, 1 ordinal
- Continuous outcome variable:
 - Ipsi: log(prostate specific antigen level)

TABLE 3.2. *Linear model fit to the prostate cancer data. The Z score is the coefficient divided by its standard error (3.12). Roughly a Z score larger than two in absolute value is significantly nonzero at the $p = 0.05$ level.*

| Term | Coefficient | Std. Error | Z Score |
|-----------|-------------|------------|---------|
| Intercept | 2.46 | 0.09 | 27.60 |
| lcavol | 0.68 | 0.13 | 5.37 |
| lweight | 0.26 | 0.10 | 2.75 |
| age | -0.14 | 0.10 | -1.40 |
| lbph | 0.21 | 0.10 | 2.06 |
| svi | 0.31 | 0.12 | 2.47 |
| lcp | -0.29 | 0.15 | -1.87 |
| gleason | -0.02 | 0.15 | -0.15 |
| pgg45 | 0.27 | 0.15 | 1.74 |

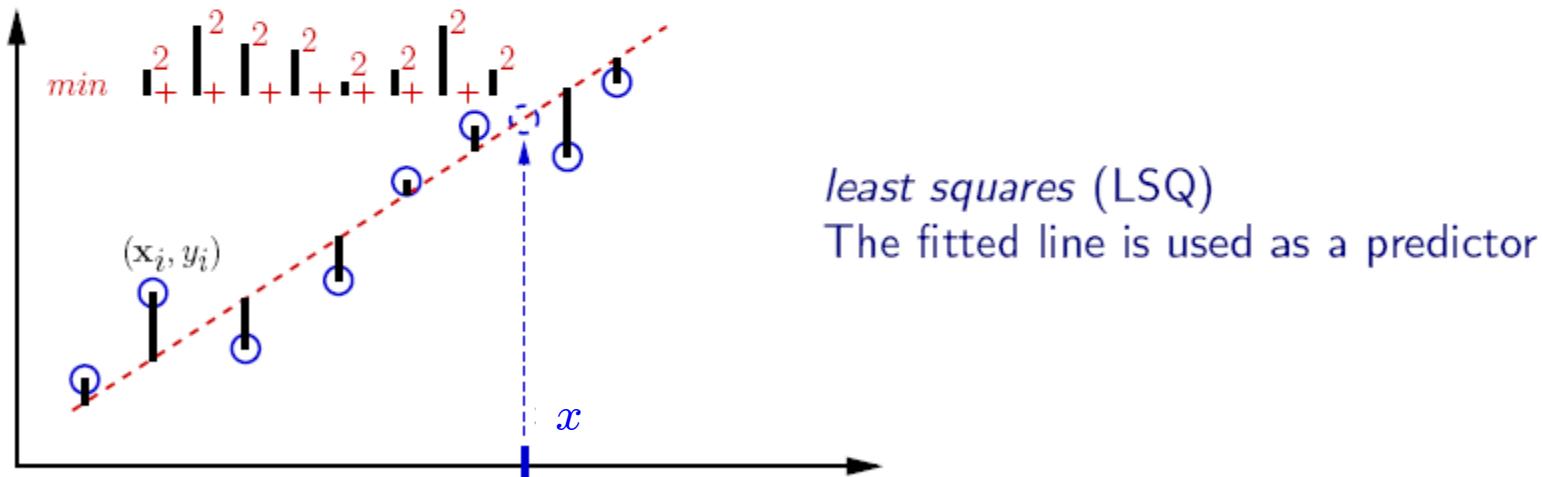
Linear Regression

- Hypothesis:

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

Assume $x_0 = 1$

- Fit model by minimizing sum of squared errors

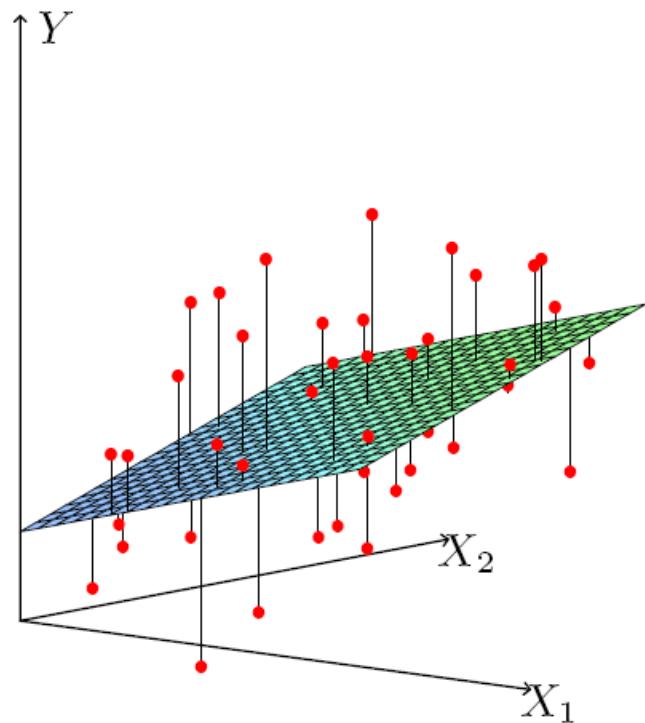
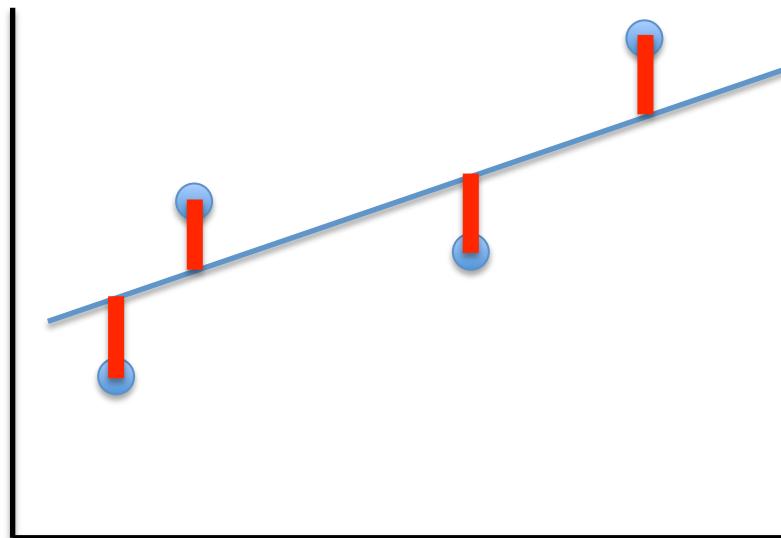


Least Squares Linear Regression

- Cost Function

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

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



Intuition Behind Cost Function

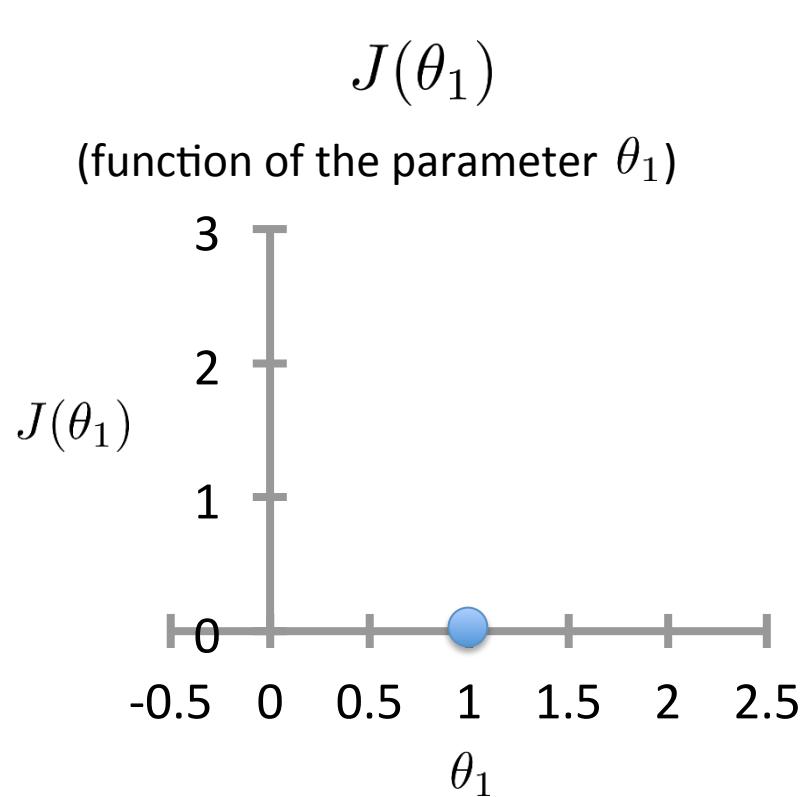
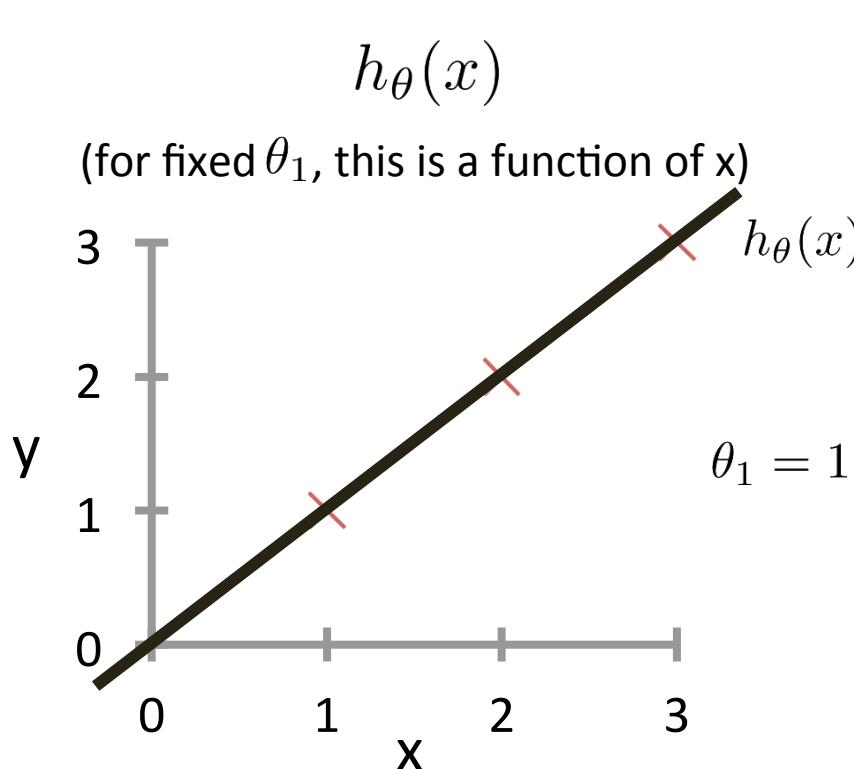
$$J(\theta) = \frac{1}{2n} \sum_{i=1}^n \left(h_{\theta} \left(x^{(i)} \right) - y^{(i)} \right)^2$$

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

Intuition Behind Cost Function

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

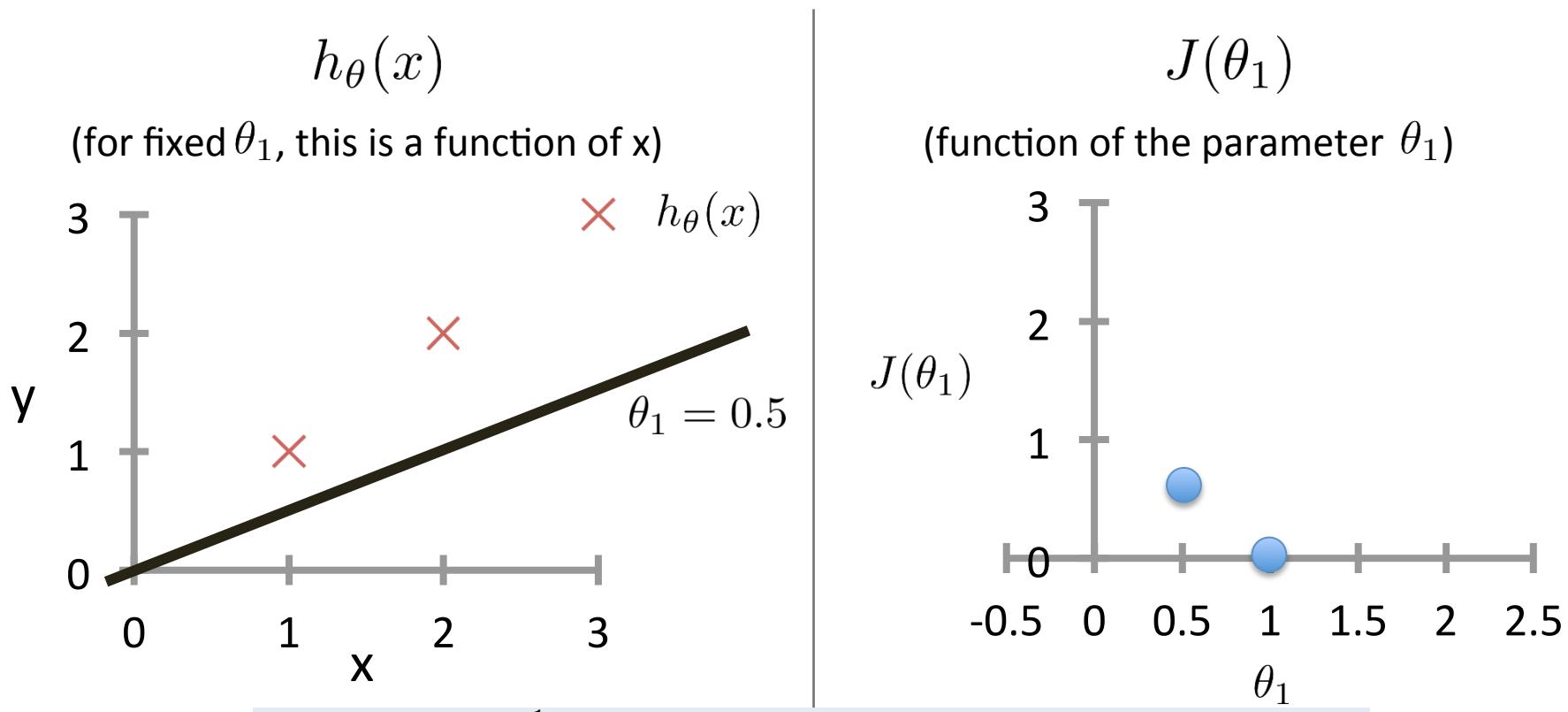
For insight on $J()$, let's assume $x \in \mathbb{R}$ so $\theta = [\theta_0, \theta_1]$



Intuition Behind Cost Function

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

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

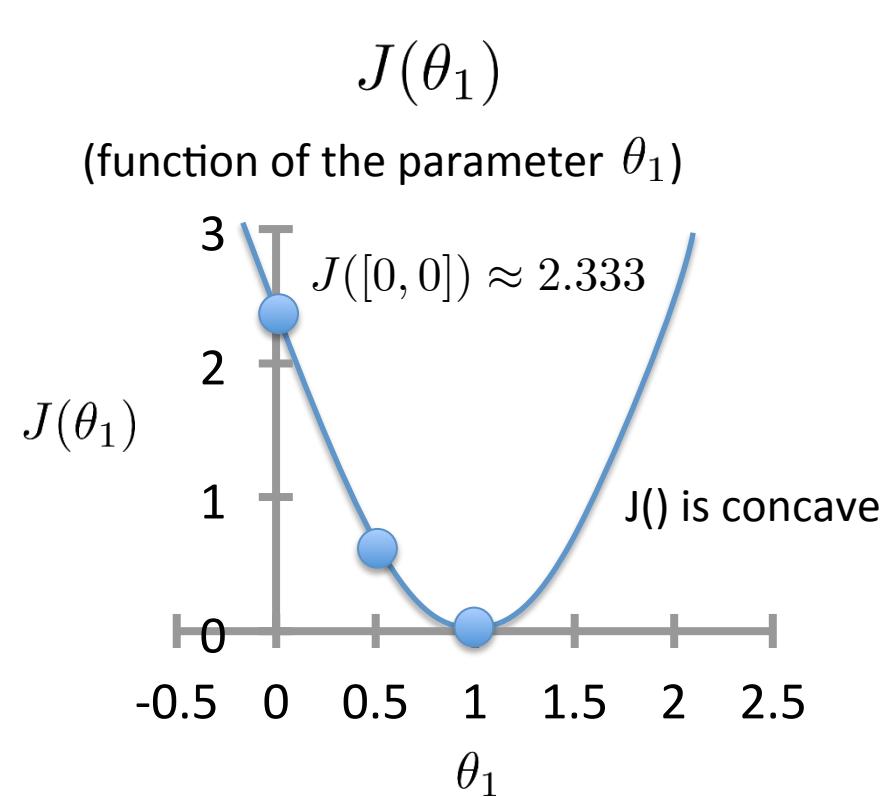
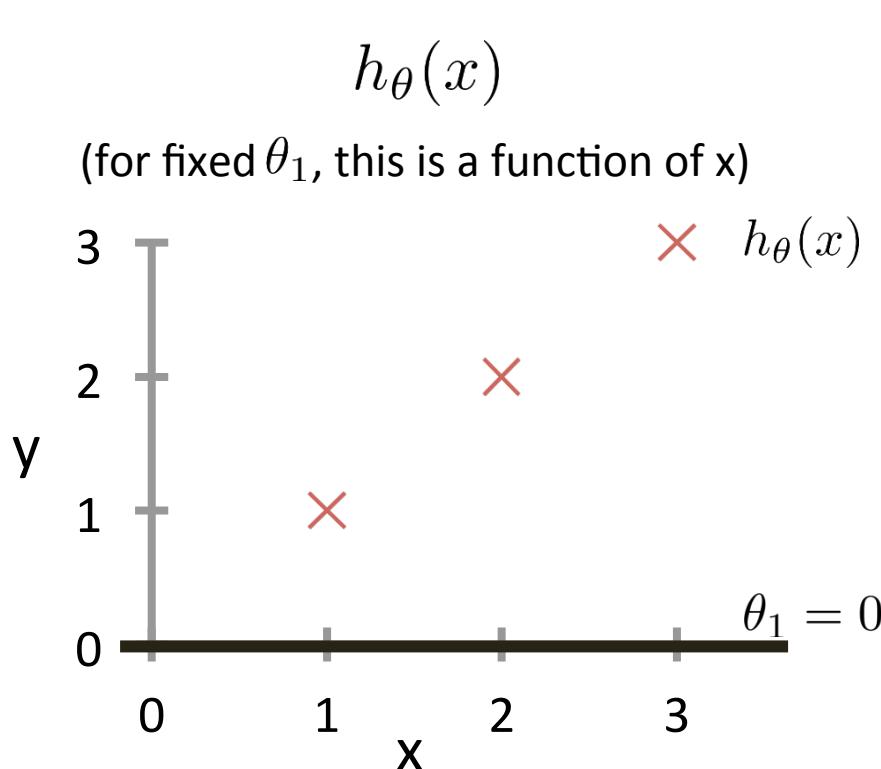


$$J([0, 0.5]) = \frac{1}{2 \times 3} [(0.5 - 1)^2 + (1 - 2)^2 + (1.5 - 3)^2] \approx 0.58$$

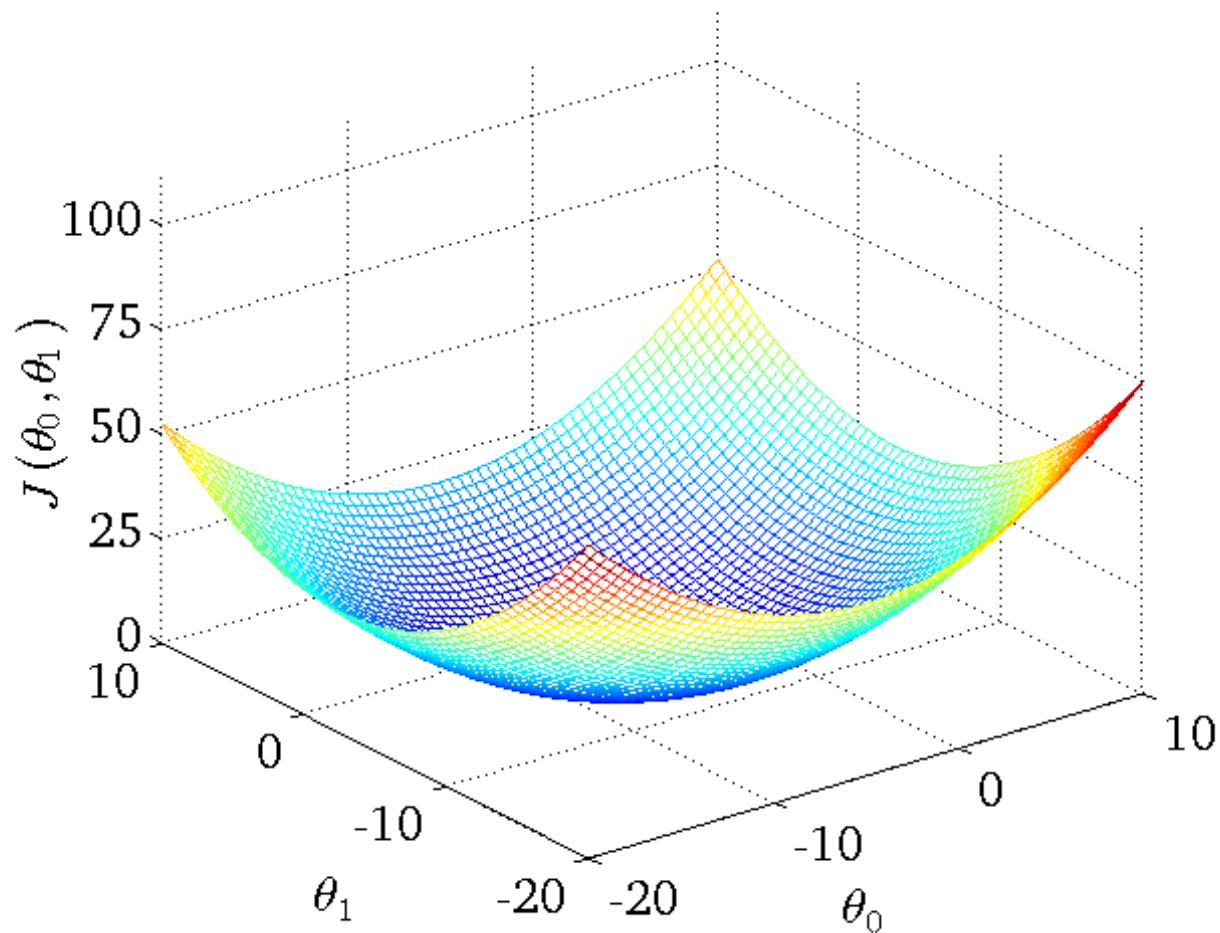
Intuition Behind Cost Function

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

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



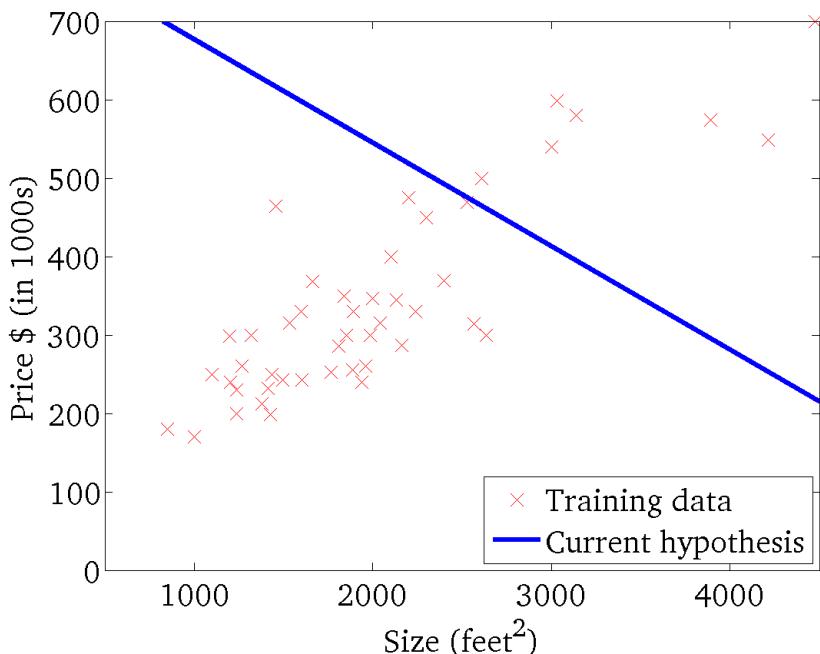
Intuition Behind Cost Function



Intuition Behind Cost Function

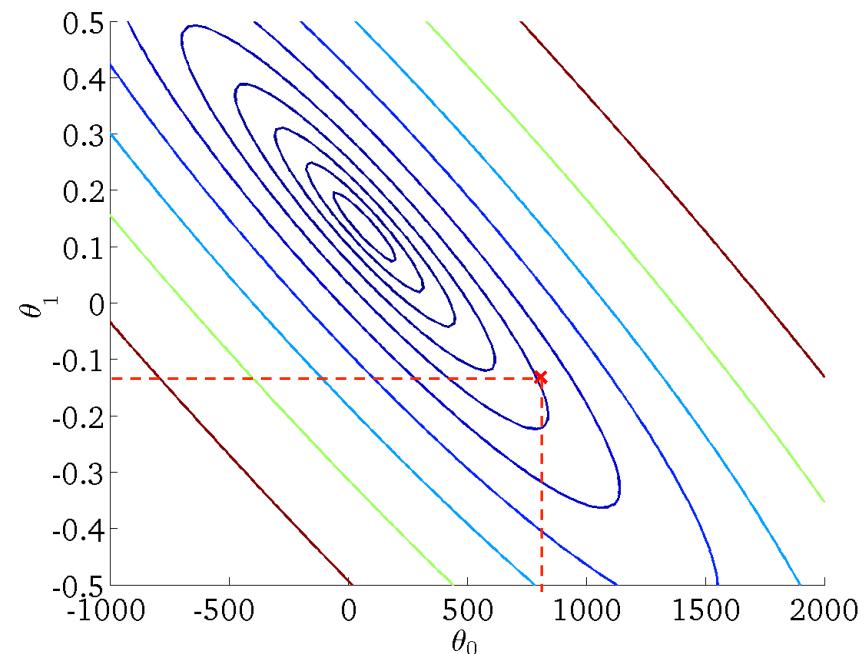
$$h_{\theta}(x)$$

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



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

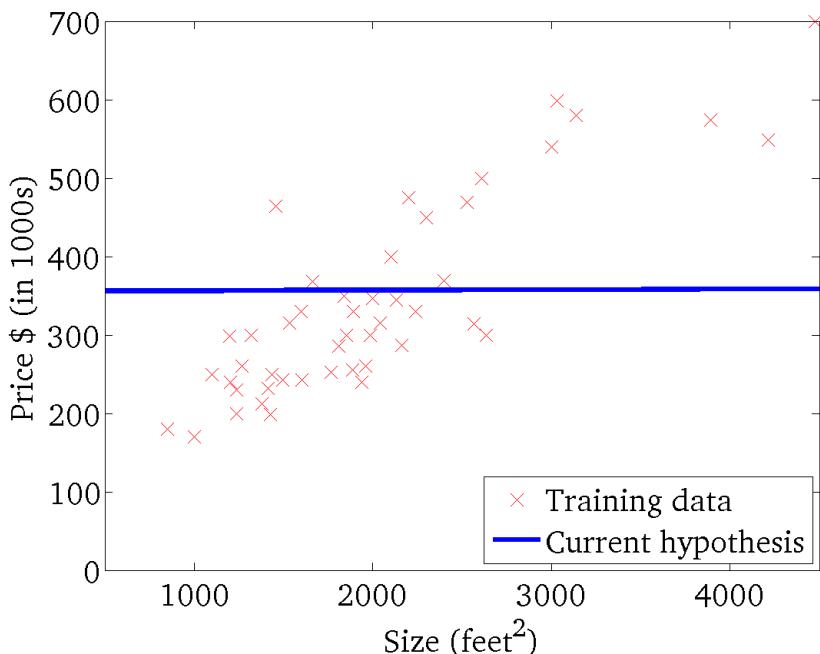
(function of the parameters θ_0, θ_1)



Intuition Behind Cost Function

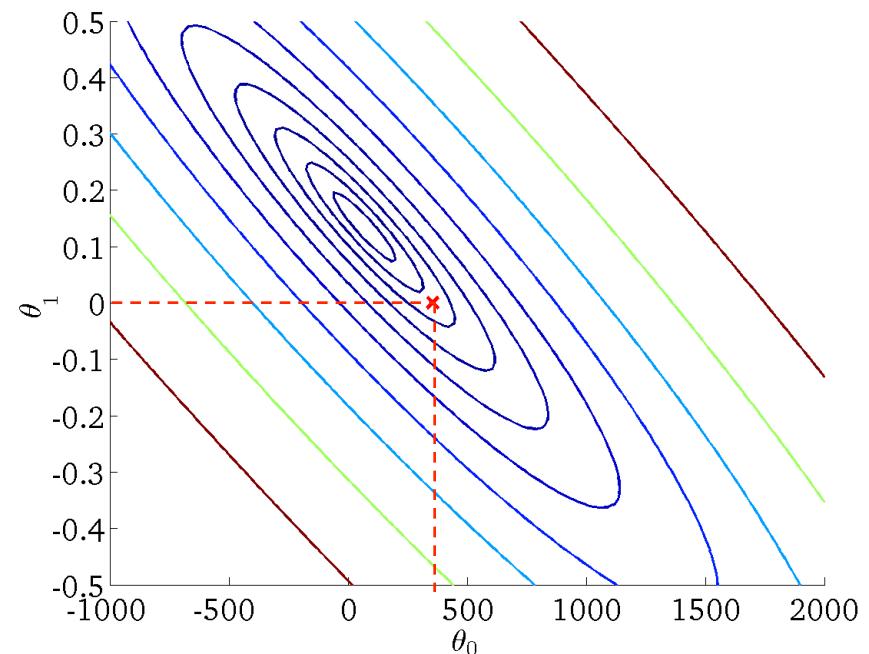
$$h_{\theta}(x)$$

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



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

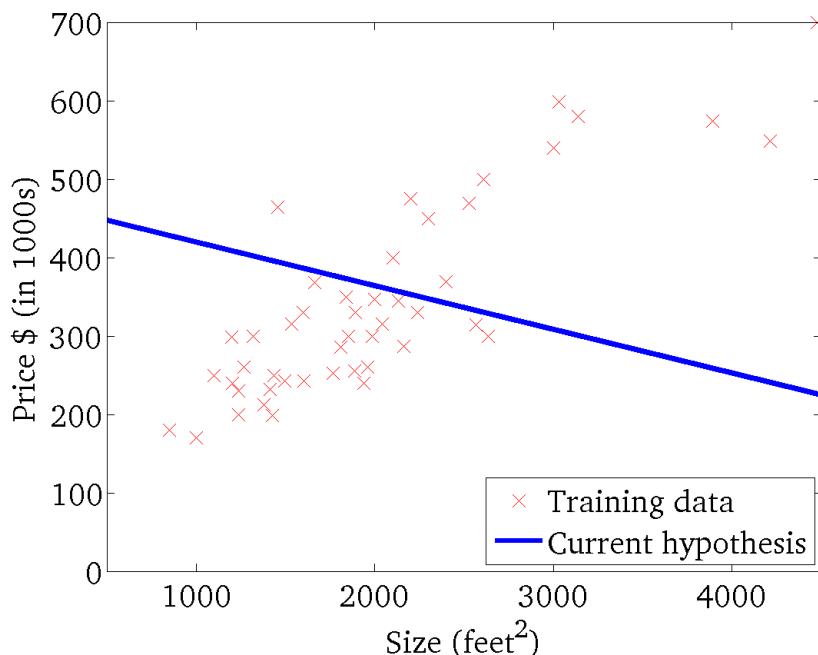
(function of the parameters θ_0, θ_1)



Intuition Behind Cost Function

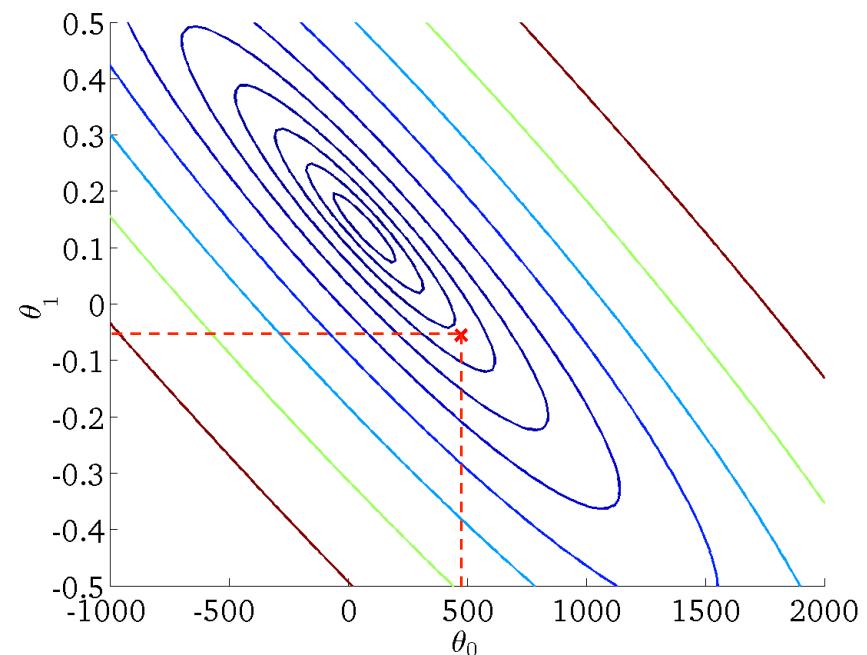
$$h_{\theta}(x)$$

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



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

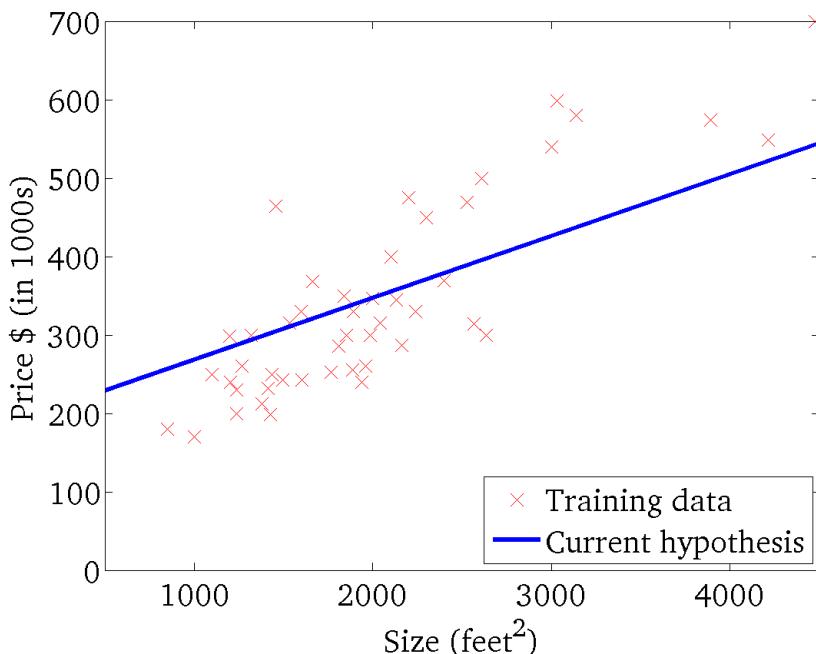
(function of the parameters θ_0, θ_1)



Intuition Behind Cost Function

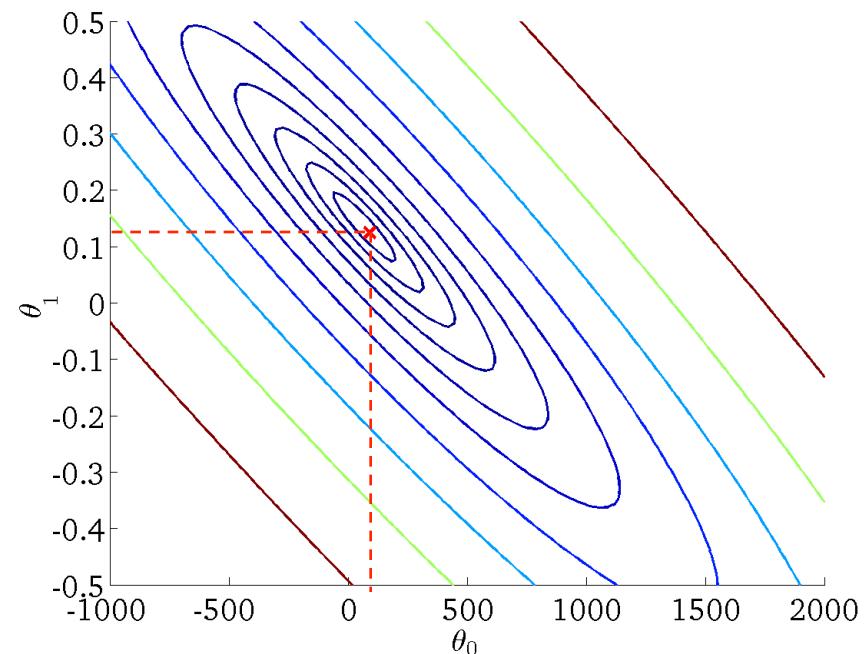
$$h_{\theta}(x)$$

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



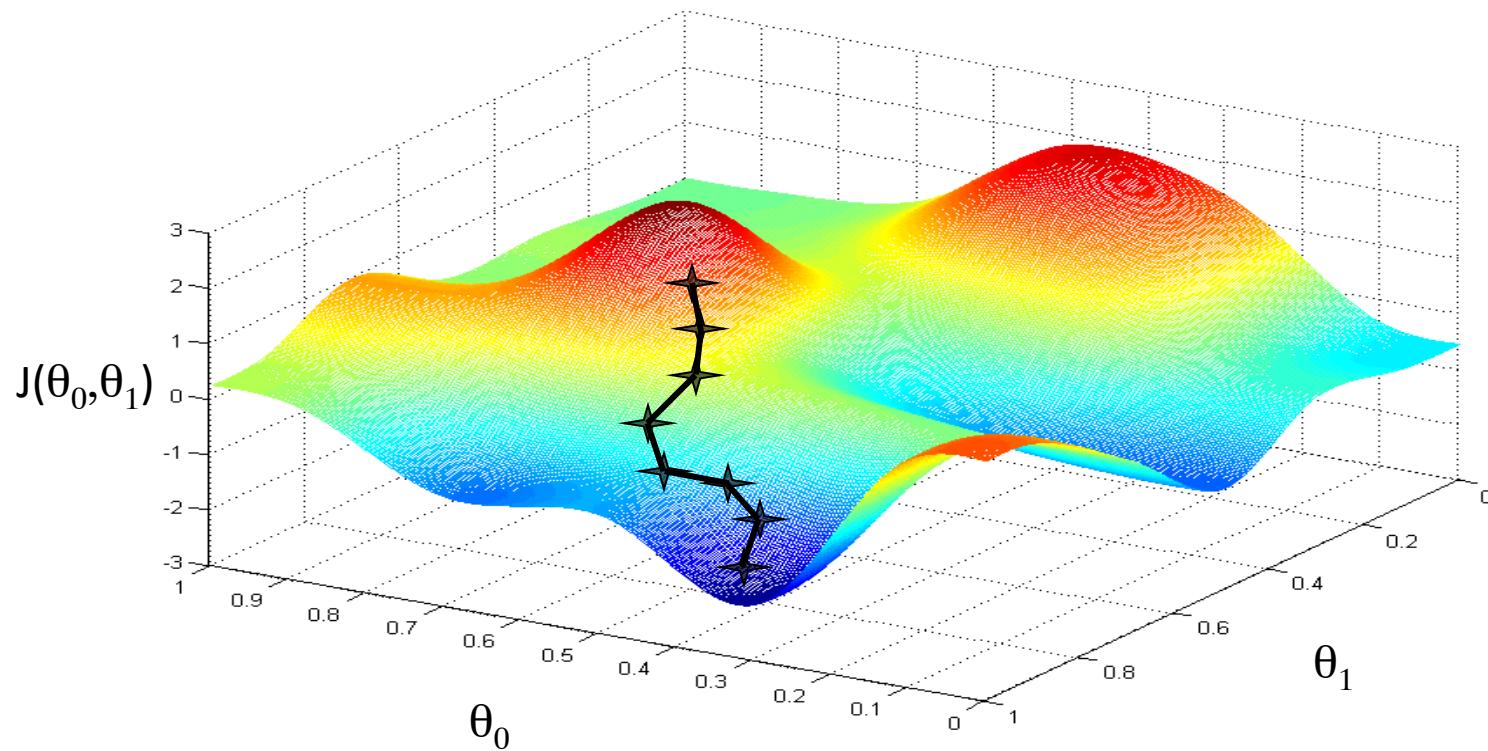
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



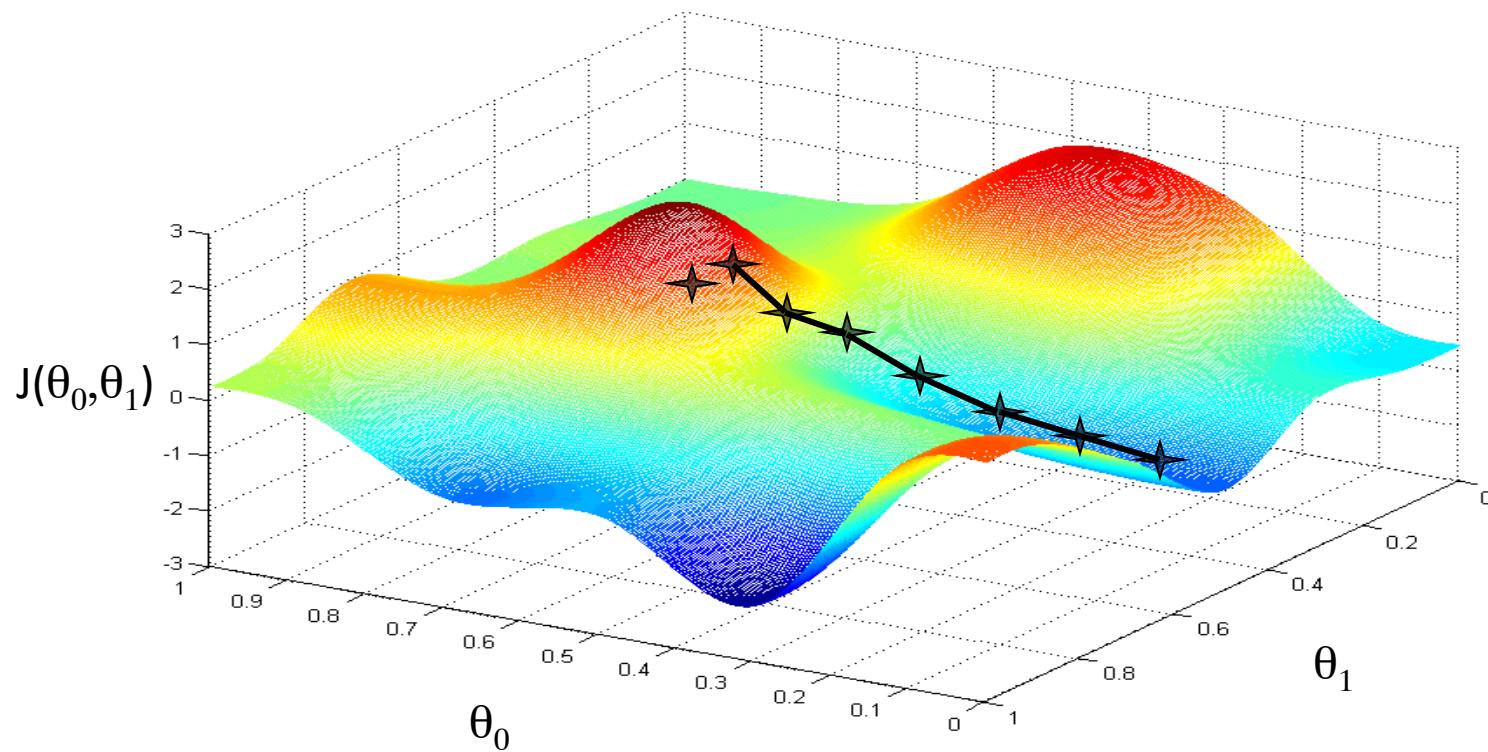
Basic Search Procedure

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



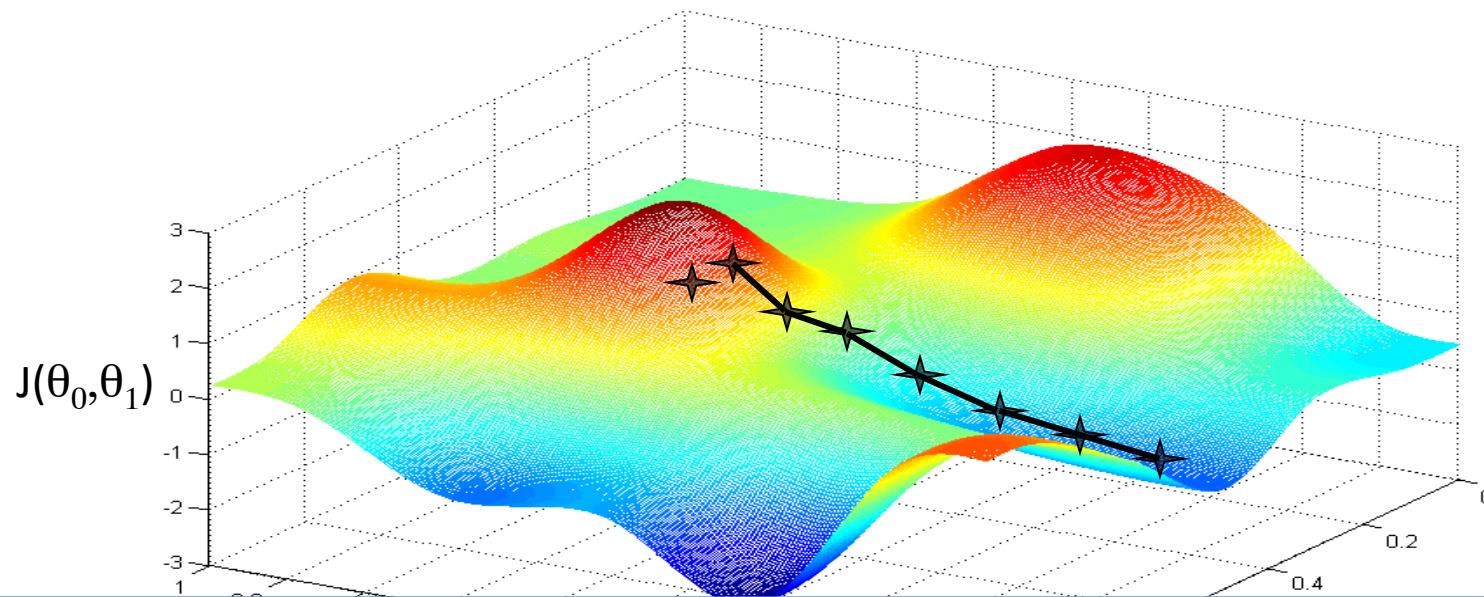
Basic Search Procedure

- 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 $J(\theta)$



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

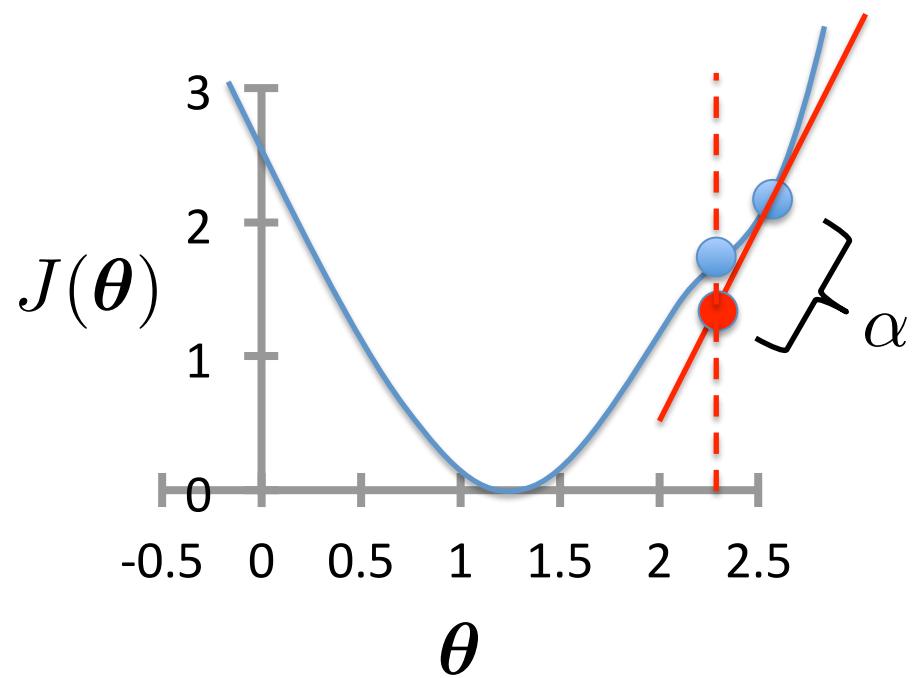
Gradient Descent

- Initialize θ
- Repeat until convergence

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

simultaneous update
for $j = 0 \dots d$

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



Gradient Descent

- Initialize θ
- Repeat until convergence

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

simultaneous update
for $j = 0 \dots d$

For Linear Regression:

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

Gradient Descent

- Initialize θ
- Repeat until convergence

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

simultaneous update
for $j = 0 \dots d$

For Linear Regression:

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

Gradient Descent

- Initialize θ
- Repeat until convergence

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

simultaneous update
for $j = 0 \dots d$

For Linear Regression:

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

Gradient Descent

- Initialize θ
- Repeat until convergence

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

simultaneous update
for $j = 0 \dots d$

For Linear Regression:

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

Gradient Descent for Linear Regression

- Initialize θ
- Repeat until convergence

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

simultaneous
update
for $j = 0 \dots d$

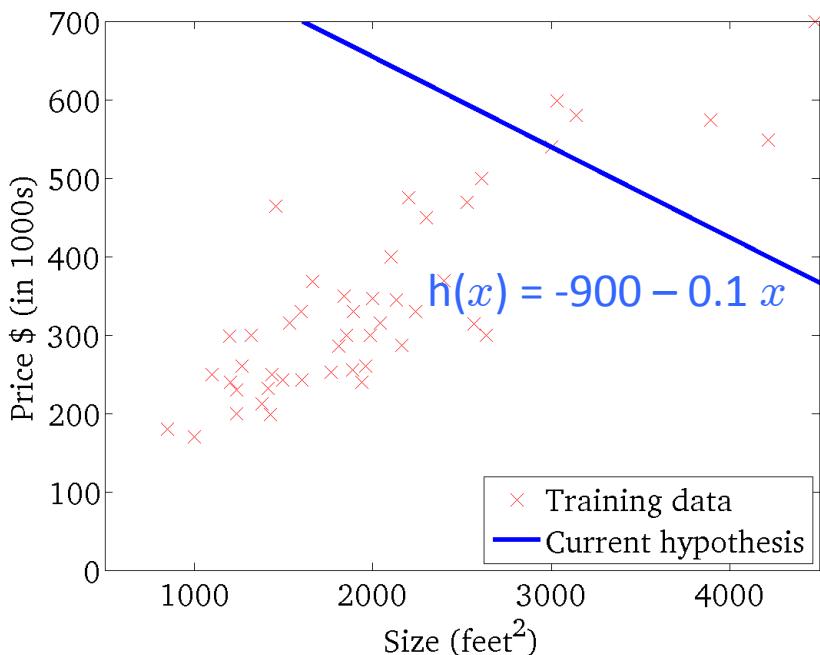
- To achieve simultaneous update
 - At the start of each GD iteration, compute $h_{\theta} \left(\mathbf{x}^{(i)} \right)$
 - Use this stored value in the update step loop
- Assume convergence when $\|\theta_{new} - \theta_{old}\|_2 < \epsilon$

L₂ norm: $\|\mathbf{v}\|_2 = \sqrt{\sum_i v_i^2} = \sqrt{v_1^2 + v_2^2 + \dots + v_{|v|}^2}$

Gradient Descent

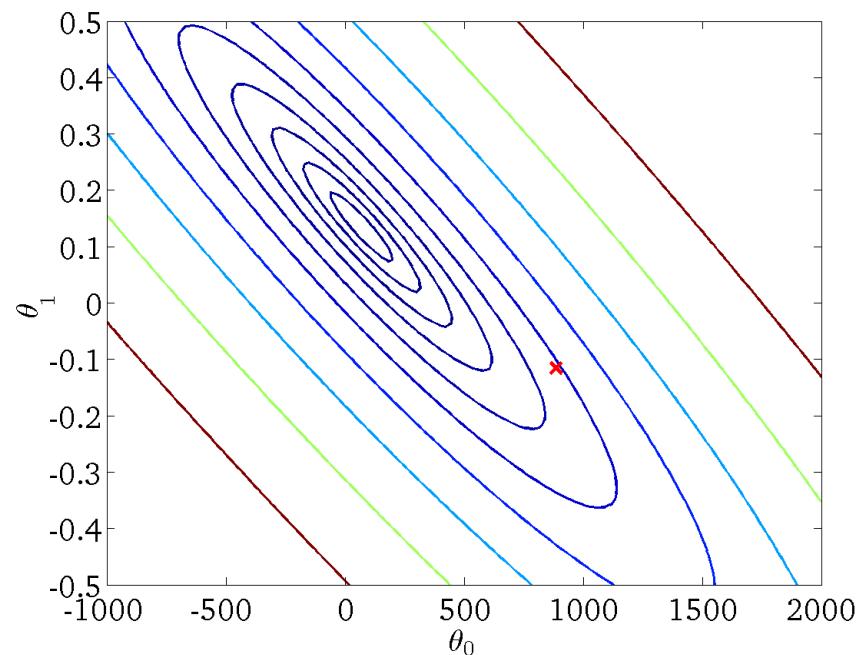
$$h_{\theta}(x)$$

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



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

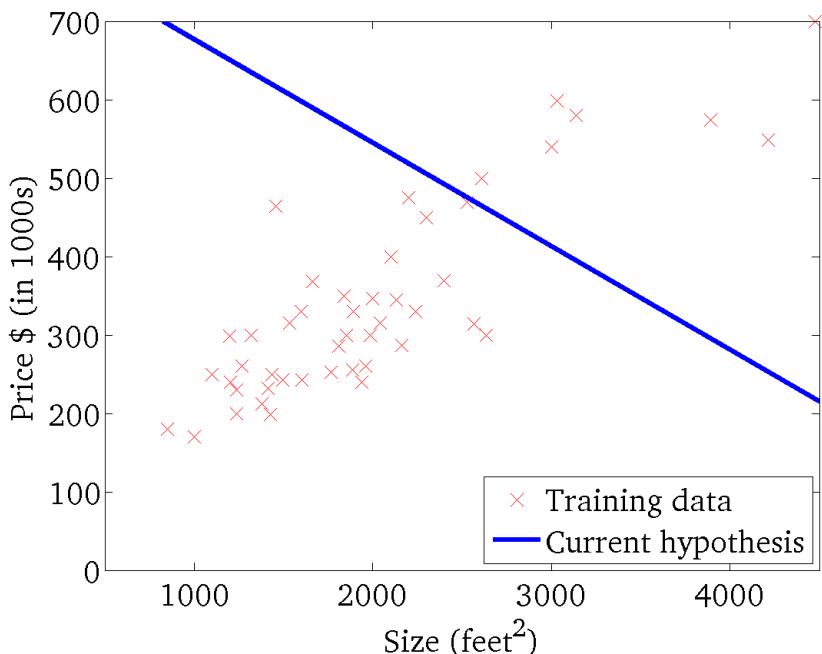
(function of the parameters θ_0, θ_1)



Gradient Descent

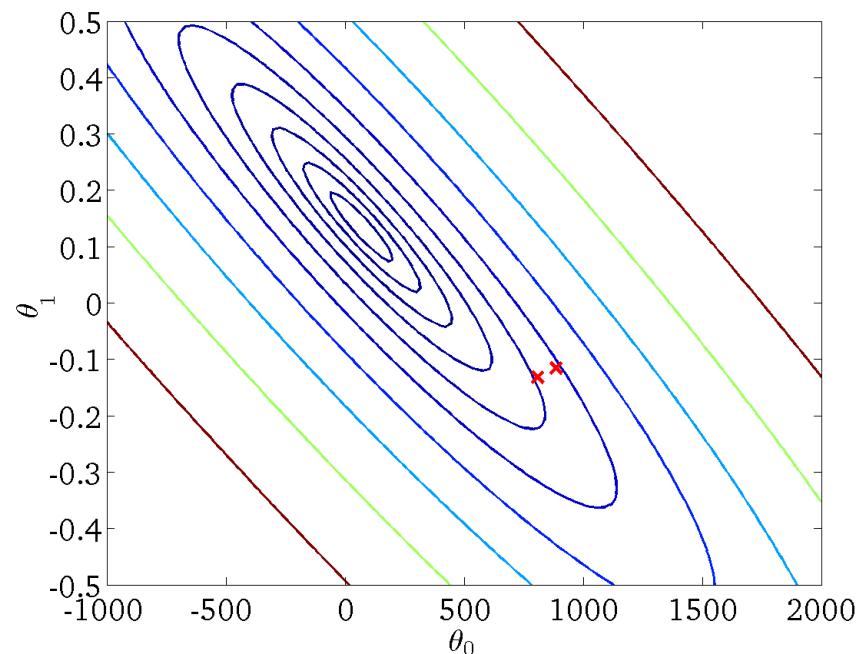
$$h_{\theta}(x)$$

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



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

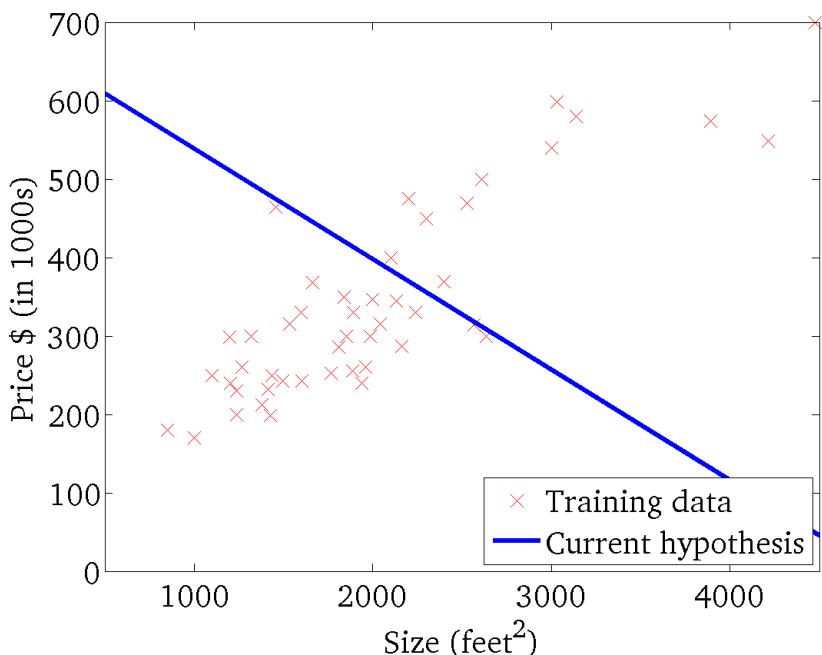
(function of the parameters θ_0, θ_1)



Gradient Descent

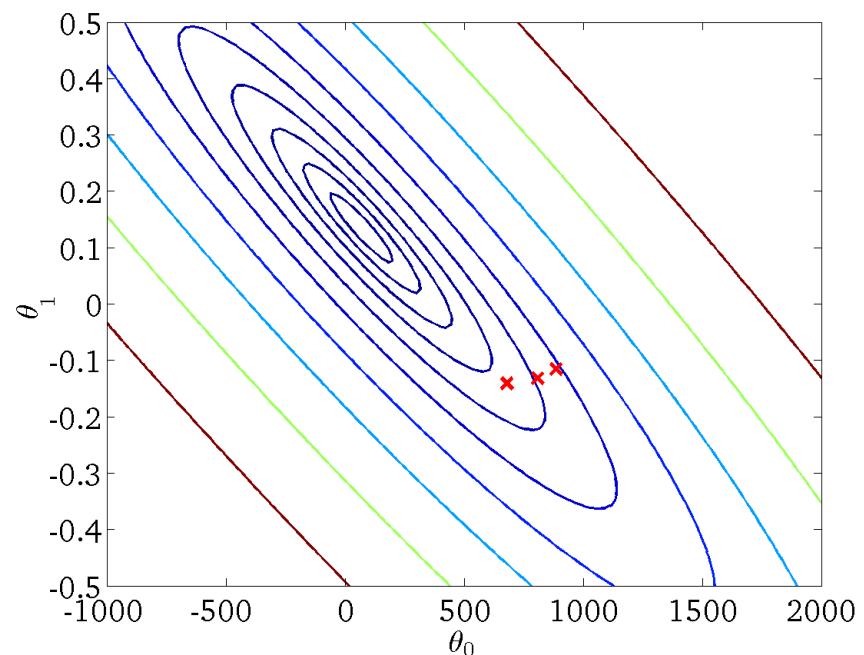
$$h_{\theta}(x)$$

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



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

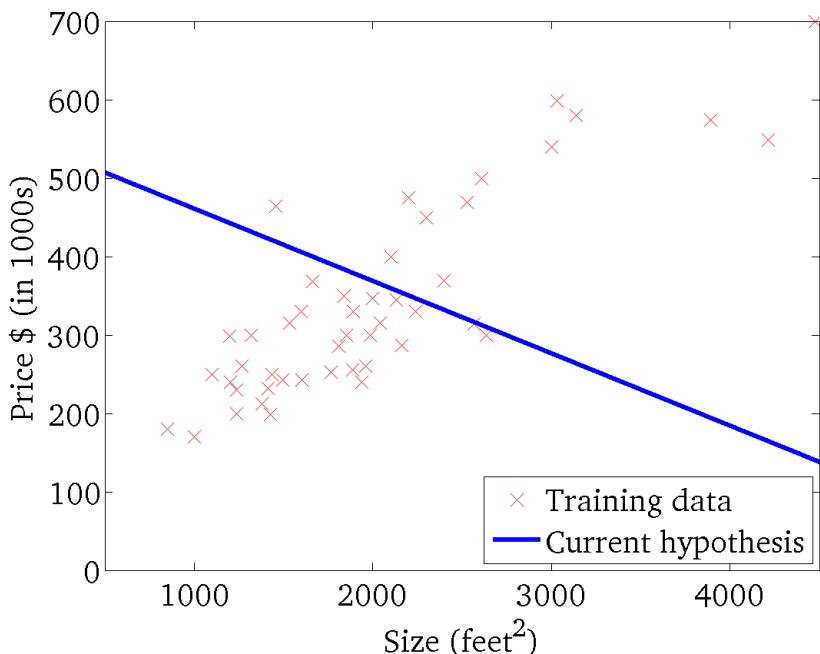
(function of the parameters θ_0, θ_1)



Gradient Descent

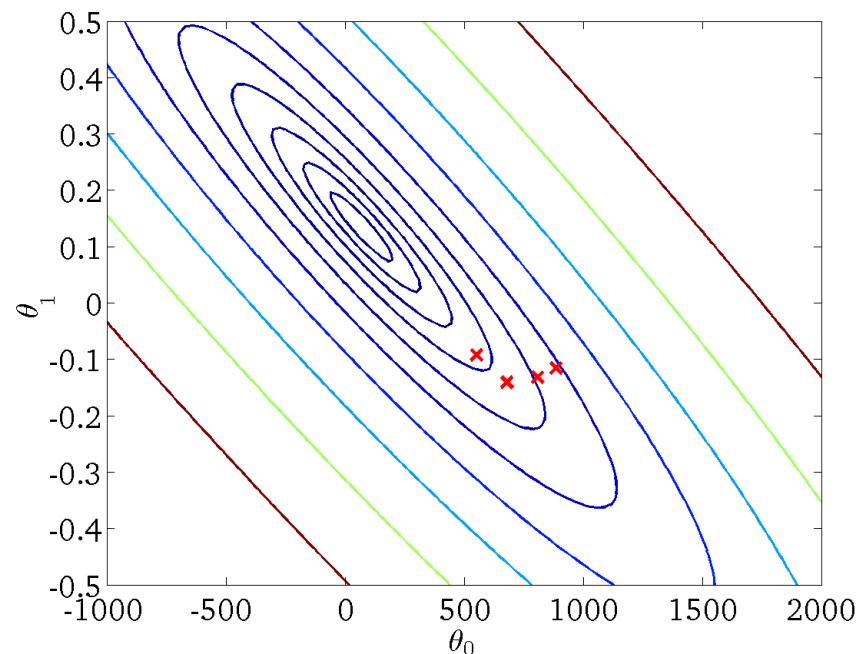
$$h_{\theta}(x)$$

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



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

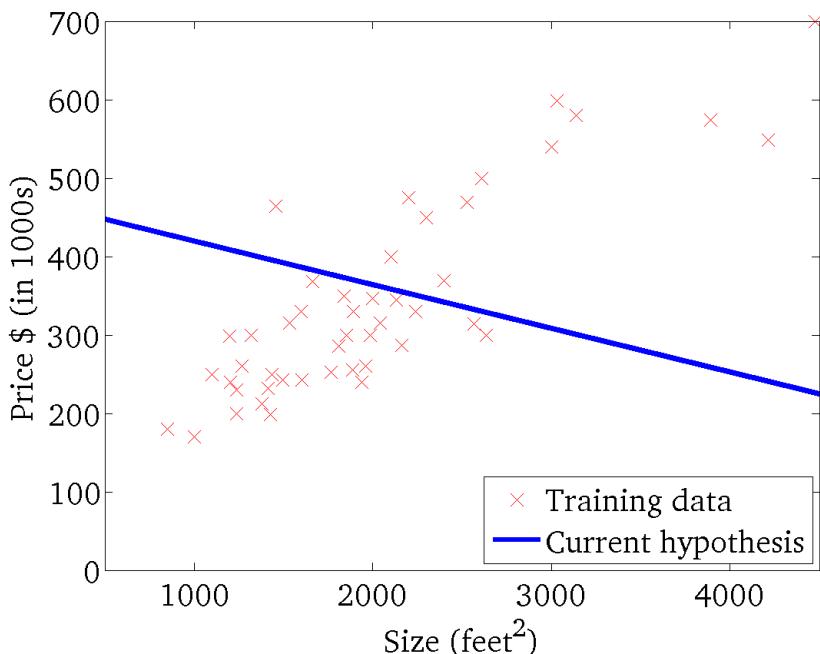
(function of the parameters θ_0, θ_1)



Gradient Descent

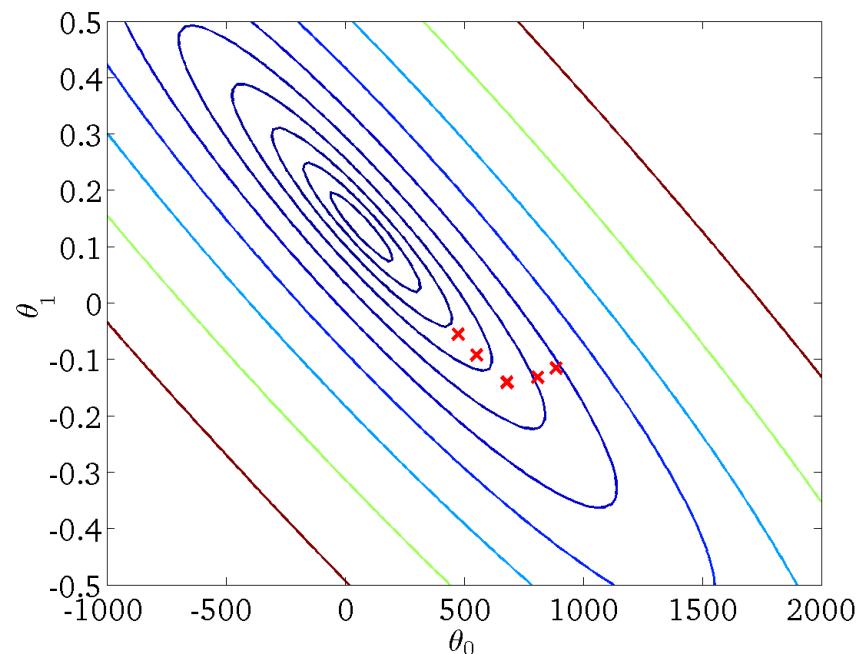
$$h_{\theta}(x)$$

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



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

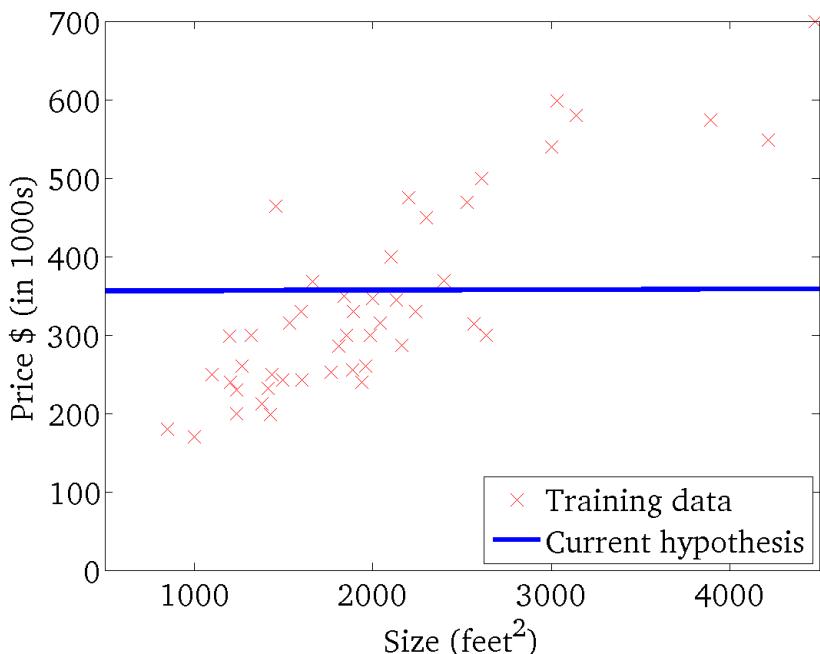
(function of the parameters θ_0, θ_1)



Gradient Descent

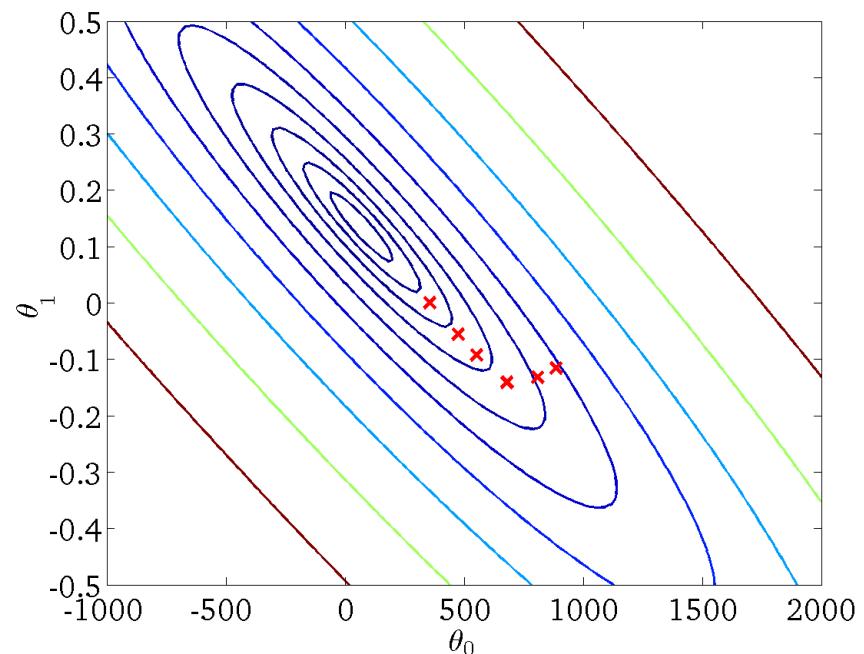
$$h_{\theta}(x)$$

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



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

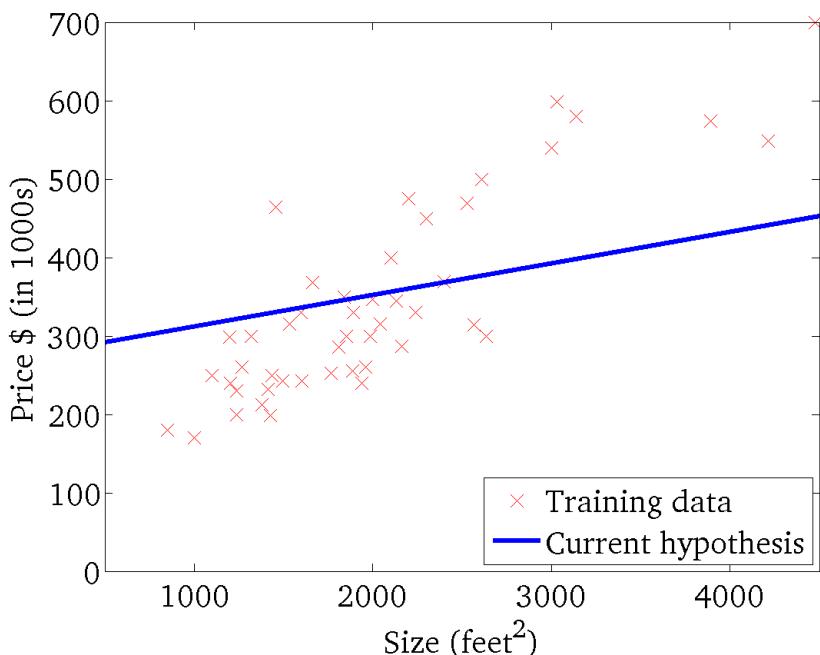
(function of the parameters θ_0, θ_1)



Gradient Descent

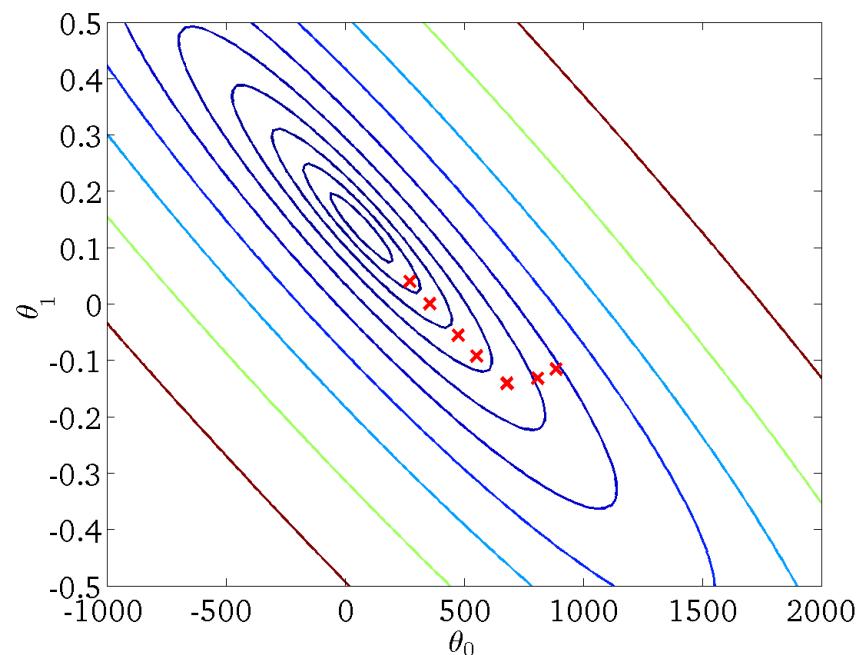
$$h_{\theta}(x)$$

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



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

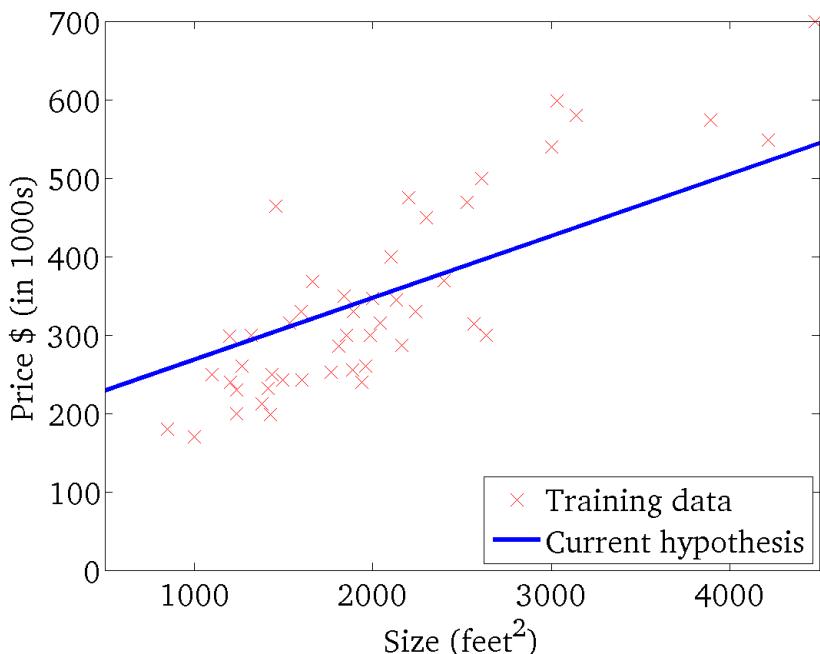
(function of the parameters θ_0, θ_1)



Gradient Descent

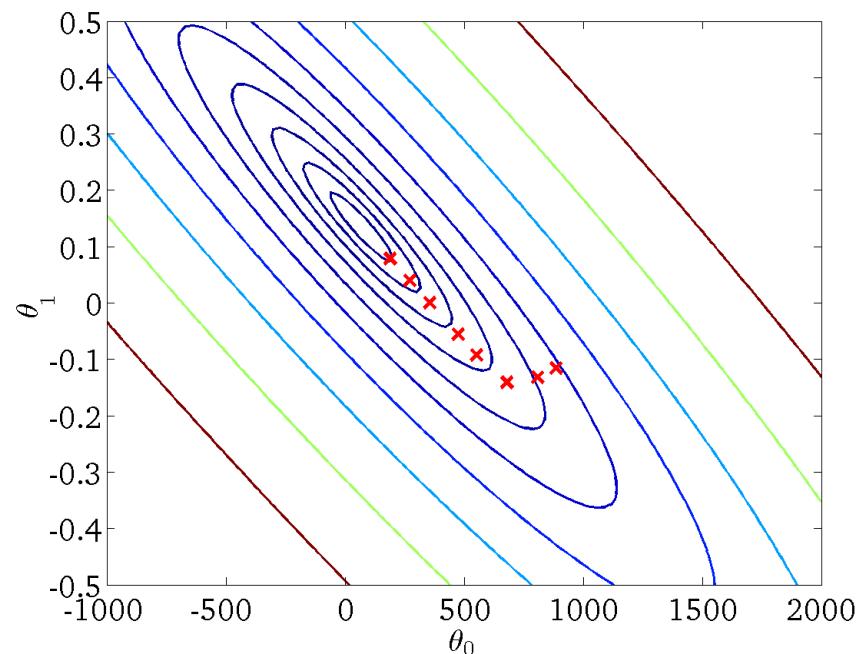
$$h_{\theta}(x)$$

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



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

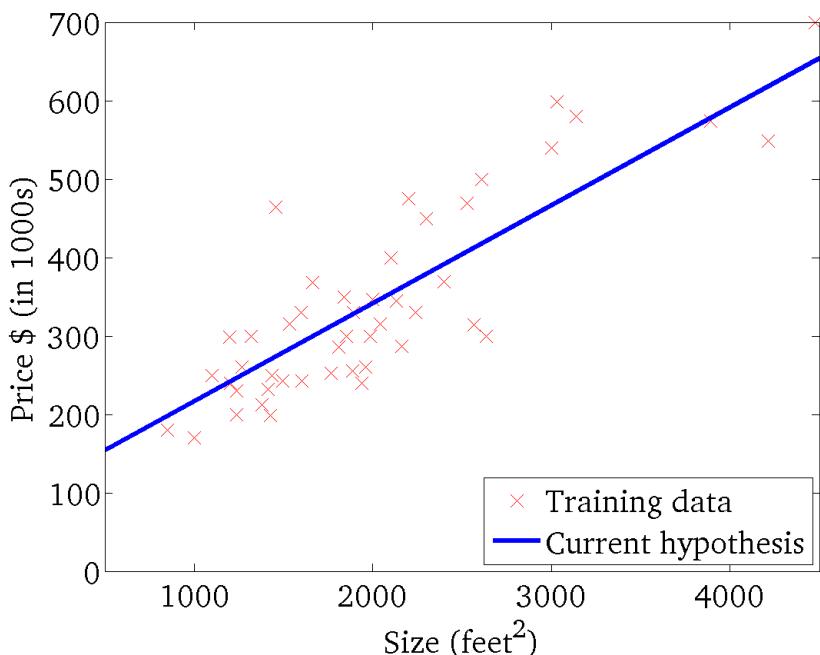
(function of the parameters θ_0, θ_1)



Gradient Descent

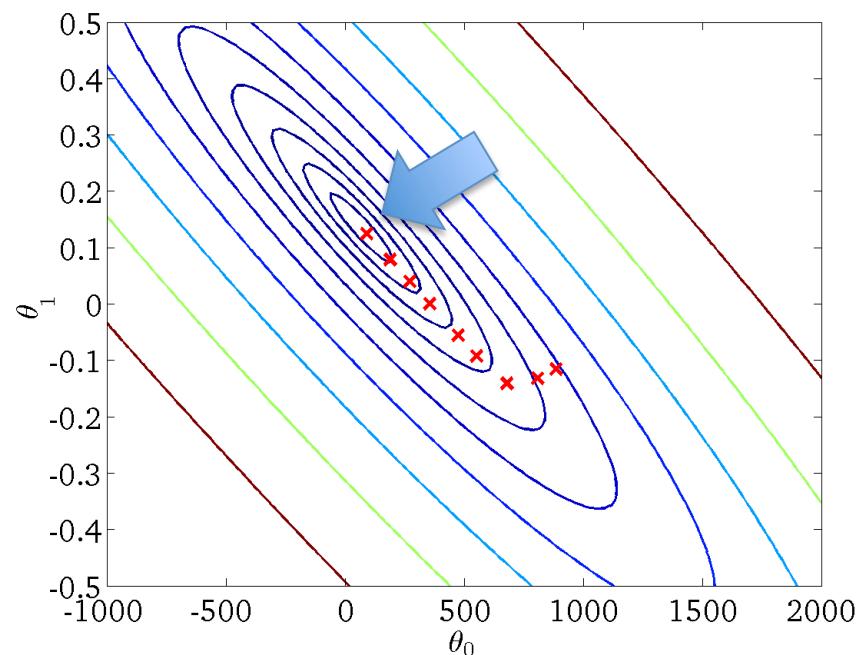
$$h_{\theta}(x)$$

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



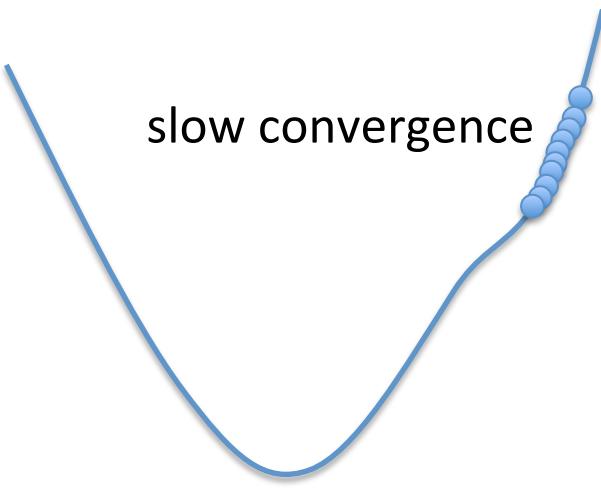
$$J(\theta_0, \theta_1)$$

(function of the parameters θ_0, θ_1)



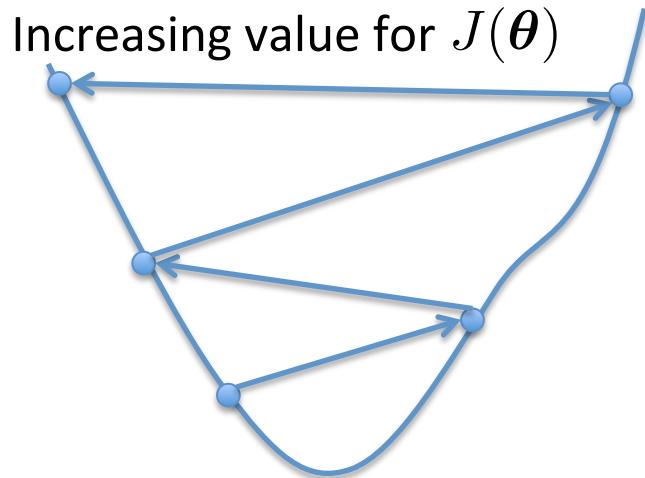
Choosing α

α too small



α too large

Increasing value for $J(\theta)$



- 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 \mathbf{X} for linear regression can be:
 - Original quantitative inputs
 - Transformation of quantitative inputs
 - e.g. log, exp, square root, square, etc.
 - Polynomial transformation
 - example: $y = \beta_0 + \beta_1 \cdot x + \beta_2 \cdot x^2 + \beta_3 \cdot x^3$
 - 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,

$$h_{\theta}(x) = \sum_{j=0}^d \theta_j \phi_j(x)$$


basis function

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

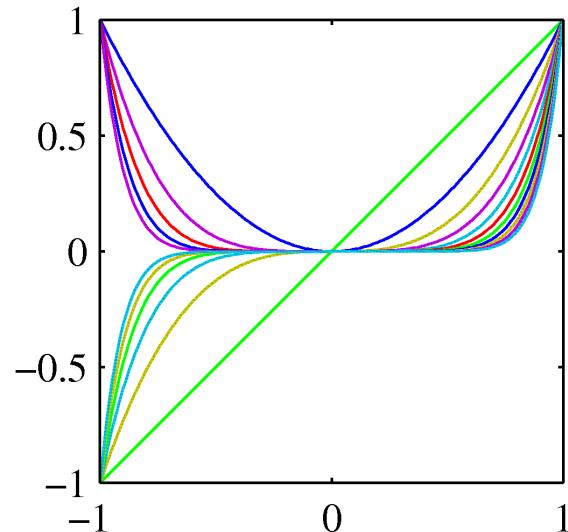
$$\phi_j(x) = x_j$$

Linear Basis Function Models

- Polynomial basis functions:

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

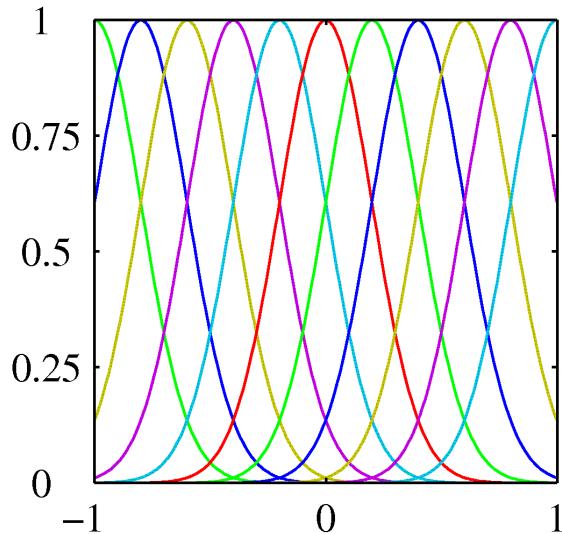
- These are global; a small change in x affects all basis functions



- 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

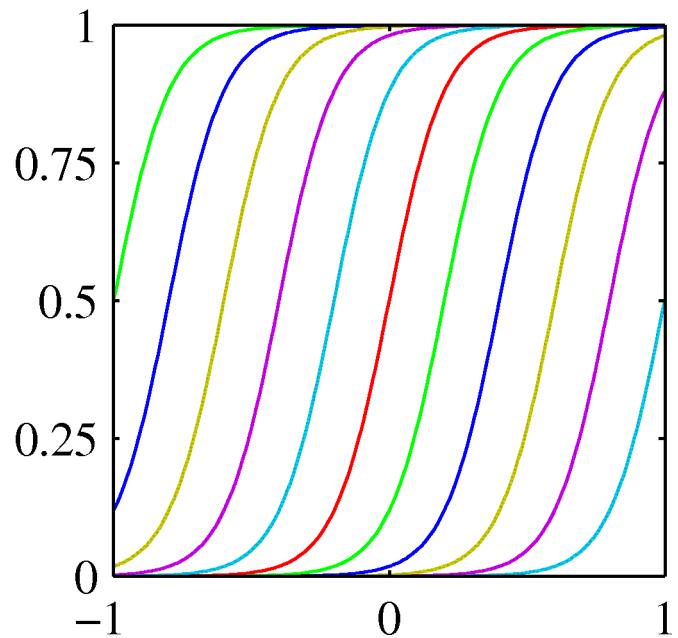
- Sigmoidal basis functions:

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

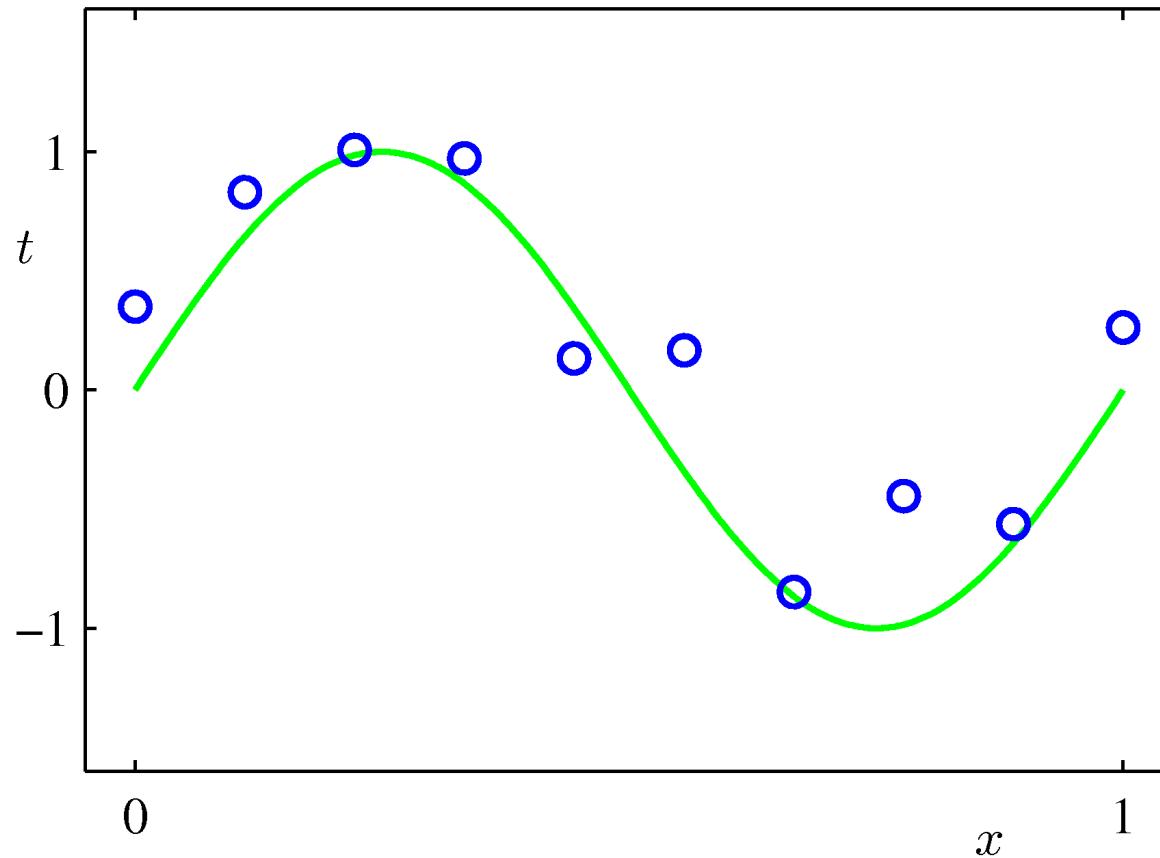
where

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

- 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



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

Linear Basis Function Models

- Basic Linear Model:

$$h_{\theta}(\mathbf{x}) = \sum_{j=0}^d \theta_j x_j$$

- Generalized Linear Model:

$$h_{\theta}(\mathbf{x}) = \sum_{j=0}^d \theta_j \phi_j(\mathbf{x})$$

- 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:

The diagram illustrates the two representations of a vector. On the left, a horizontal arrow points from the text “[1,6,3,4] is a column vector:” to a column vector $\begin{pmatrix} 1 \\ 6 \\ 3 \\ 4 \end{pmatrix}$. On the right, a diagonal arrow points from the text “as opposed to a row vector:” to a row vector $(1 \ 6 \ 3 \ 4)$.

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

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

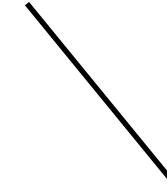
Linear Algebra Concepts

- Transpose: reflect vector/matrix on line:

$$\begin{pmatrix} a \\ b \end{pmatrix}^T = (a \quad b)$$

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

- Note: $(Ax)^T = x^T A^T$ (We'll define multiplication soon...)
- Vector norms:
 - L_p norm of $v = (v_1, \dots, v_k)$ is $\left(\sum_i |v_i|^p \right)^{\frac{1}{p}}$
 - Common norms: L_1, L_2
 - $L_{\infty} = \max_i |v_i|$
- Length of a vector v is $L_2(v)$



Linear Algebra Concepts

- Vector dot product: $u \bullet v = (u_1 \ u_2) \bullet (v_1 \ v_2) = u_1v_1 + u_2v_2$
 - 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: $u \bullet v = u^T v = (u_1 \quad u_2) \begin{pmatrix} v_1 \\ v_2 \end{pmatrix} = u_1 v_1 + u_2 v_2$

- 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:

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

- Let

$$\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix} \quad \mathbf{x}^\top = [1 \ x_1 \ \dots \ x_d]$$

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

Vectorization

- Consider our model for n instances:

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

- Let

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix}$$

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

$$X = \begin{bmatrix} 1 & x_1^{(1)} & \dots & x_d^{(1)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(i)} & \dots & x_d^{(i)} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_1^{(n)} & \dots & x_d^{(n)} \end{bmatrix}$$
$$\mathbb{R}^{n \times (d+1)}$$

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

Vectorization

- For the linear regression cost function:

$$\begin{aligned} J(\theta) &= \frac{1}{2n} \sum_{i=1}^n \left(h_{\theta} \left(\mathbf{x}^{(i)} \right) - y^{(i)} \right)^2 \\ &= \frac{1}{2n} \sum_{i=1}^n \left(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} - y^{(i)} \right)^2 \\ &= \frac{1}{2n} (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^\top (\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) \end{aligned}$$

The diagram illustrates the dimensions of the vectors in the vectorized cost function. It shows three vectors: \mathbf{y} (blue arrow) is a column vector in $\mathbb{R}^{n \times 1}$; $\mathbf{X}\boldsymbol{\theta}$ (red arrow) is a column vector in $\mathbb{R}^{n \times 1}$; and \mathbf{y} (black arrow) is a column vector in $\mathbb{R}^{n \times 1}$. A horizontal blue bracket under the term $(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^\top$ indicates that it is a row vector in $\mathbb{R}^{1 \times n}$.

Let:

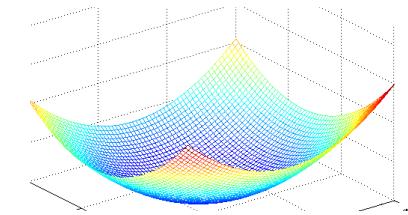
$$\mathbf{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(n)} \end{bmatrix}$$

Closed Form Solution

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

$$\begin{aligned}\mathcal{J}(\theta) &= \frac{1}{2n} (X\theta - y)^T (X\theta - y) \\ &\propto \theta^T X^T X \theta - \boxed{y^T X \theta} - \boxed{\theta^T X^T y} + y^T y \\ &\propto \theta^T X^T X \theta - 2\theta^T X^T y + y^T y\end{aligned}$$

1 x 1



Take derivative and set equal to 0, then solve for θ :

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

$$(X^T X)\theta - X^T y = 0$$

$$(X^T X)\theta = X^T y$$

Closed Form Solution:

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

Closed Form Solution

- Can obtain θ by simply plugging X and y into

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

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

- If $X^\top X$ 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

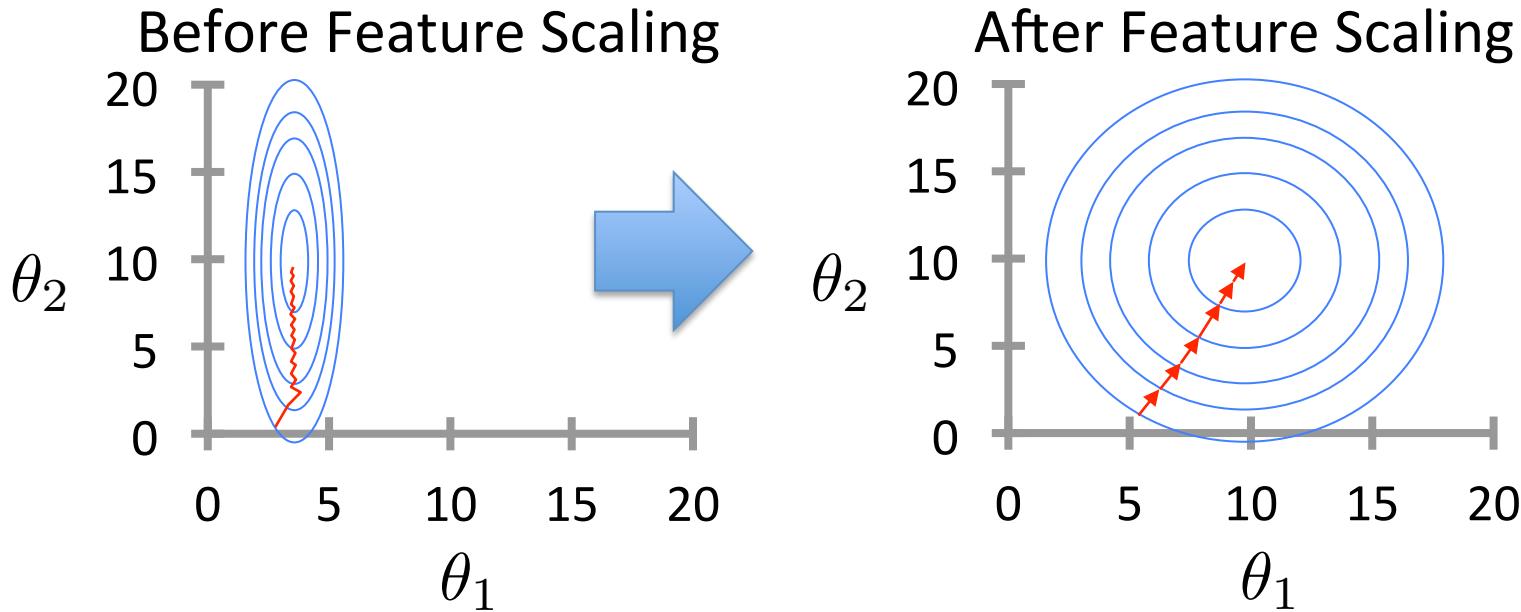
- Requires multiple iterations
- Need to choose α
- Works well when n is large
- Can support incremental learning

Closed Form Solution

- Non-iterative
- No need for α
- Slow if n is large
 - Computing $(X^T X)^{-1}$ is roughly $O(n^3)$

Improving Learning: Feature Scaling

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

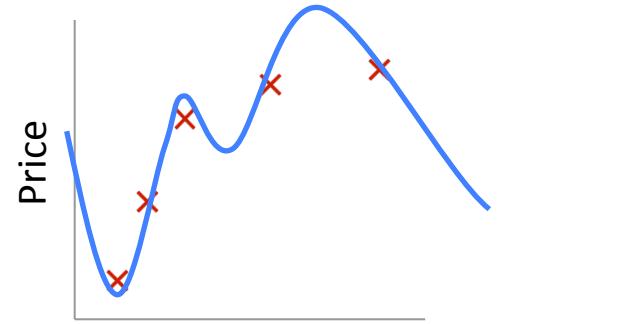
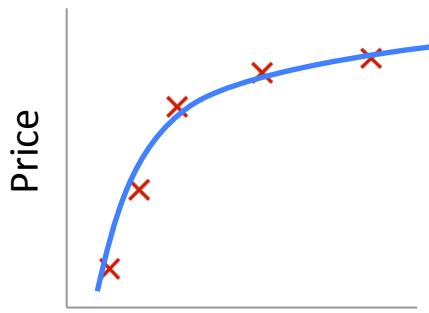
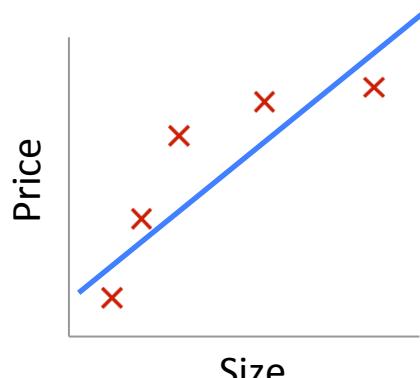


- Makes gradient descent converge *much* faster

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:
$$x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{s_j} \quad \begin{matrix} \text{for } j = 1 \dots d \\ (\text{not } x_0!) \end{matrix}$$
 - 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



Size
 $\theta_0 + \theta_1 x$
Underfitting
(high bias)

Size
 $\theta_0 + \theta_1 x + \theta_2 x^2$
Correct fit

Size
 $\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$
Overfitting
(high variance)

Overfitting:

- The learned hypothesis may fit the training set very well ($J(\theta) \approx 0$)
- ...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

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


model fit to data regularization

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

Understanding Regularization

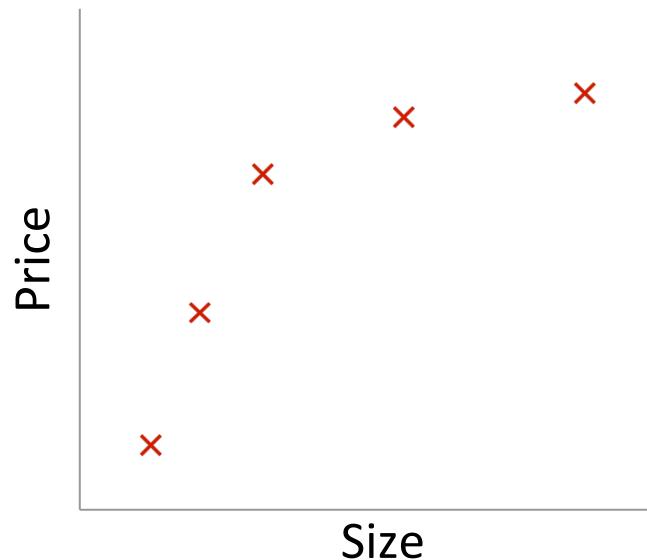
$$J(\boldsymbol{\theta}) = \frac{1}{2n} \sum_{i=1}^n \left(h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2 + \lambda \sum_{j=1}^d \theta_j^2$$

- 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:
$$\sum_{j=1}^d (\theta_j - 0)^2 = \|\boldsymbol{\theta}_{1:d} - \vec{\mathbf{0}}\|_2^2$$
 - L₂ regularization pulls coefficients toward 0

Understanding Regularization

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

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

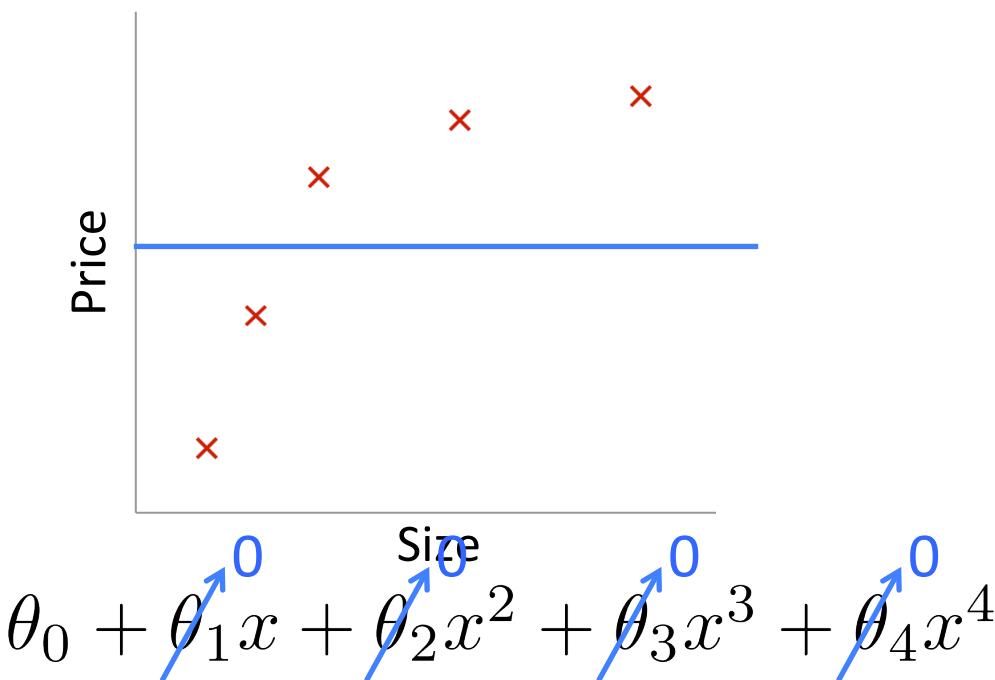


$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

Understanding Regularization

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

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



Regularized Linear Regression

- Cost Function

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

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

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

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

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

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

regularization

Regularized Linear Regression

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

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

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

- We can rewrite the gradient step as:

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

Regularized Linear Regression

- To incorporate regularization into the closed form solution:

$$\theta = \left(\begin{matrix} X^\top X \\ -1 \end{matrix} \right)^{-1} X^\top y$$

Regularized Linear Regression

- To incorporate regularization into the closed form solution:

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

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