Data Graduate Program N°2
Cdiscount Academy

Lundi 20/09 – Mercredi 23/10

SOMMAIRE

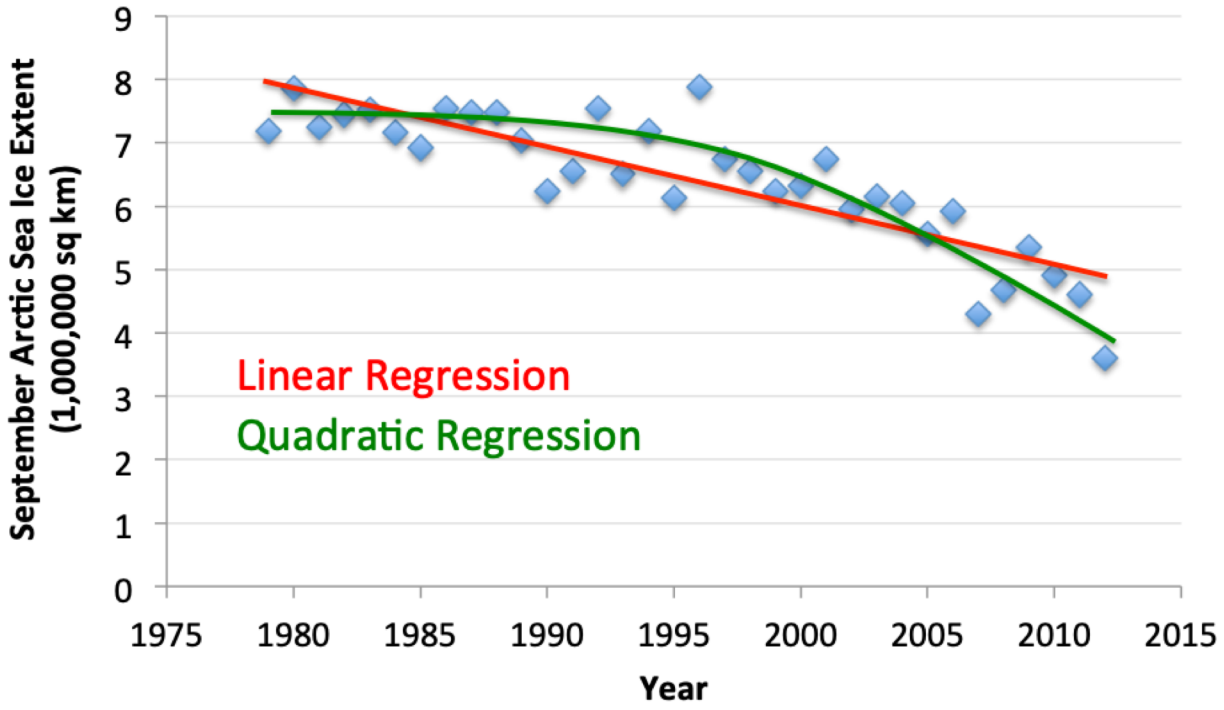
- ML – Predictive Modelling Pipeline

Linear regression

LINEAR REGRESSION

Given:

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



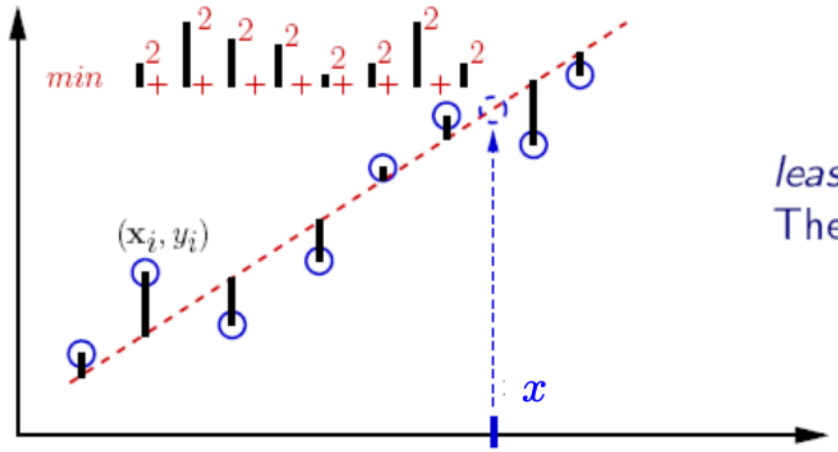
LINEAR REGRESSION

- 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



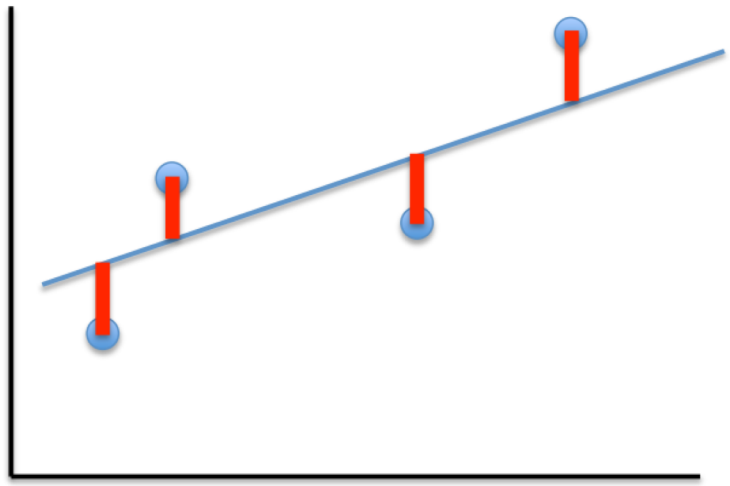
least squares (LSQ)
The fitted line is used as a predictor

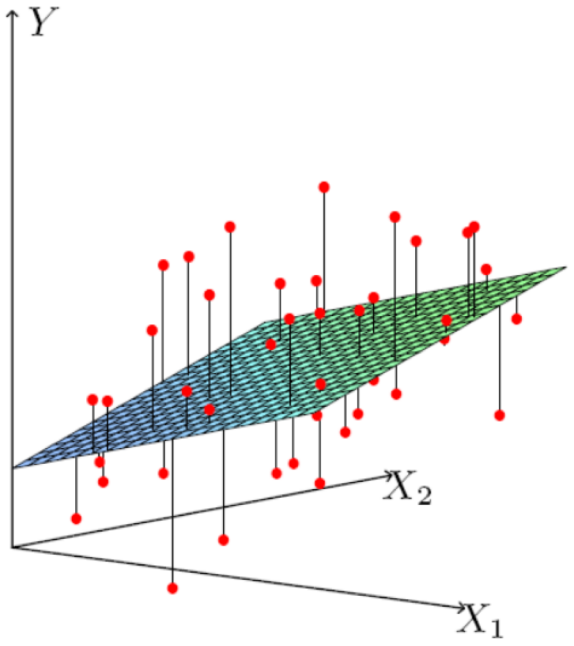
LEAST SQUARES LINEAR REGRESSION

- 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 $\min_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$





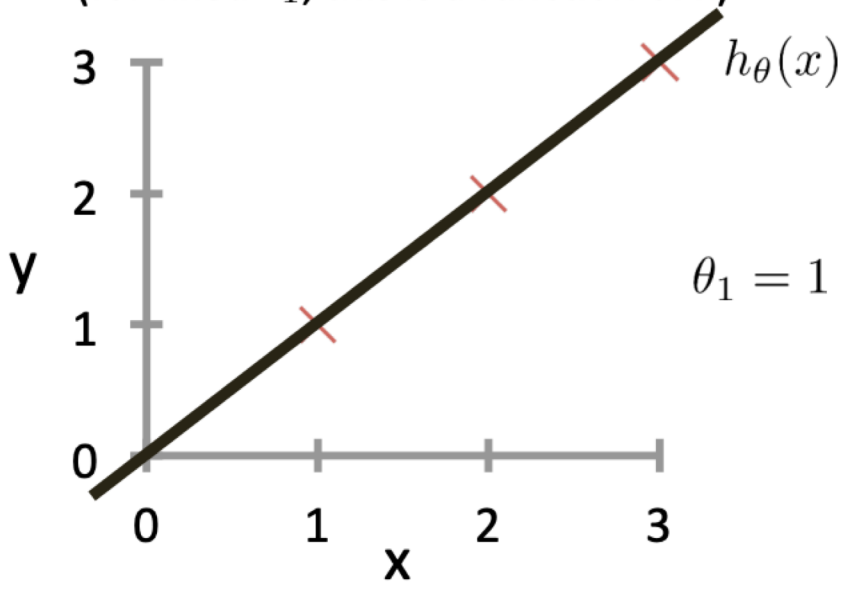
LINEAR REGRESSION – INTUITION BEHIND COST FUNCTION

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

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

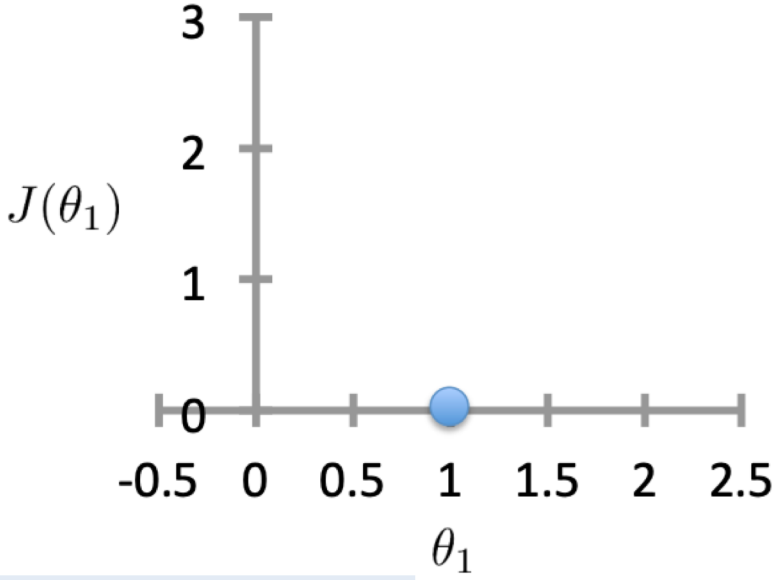
$h_\theta(x)$

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



$J(\theta_1)$

(function of the parameter θ_1)



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

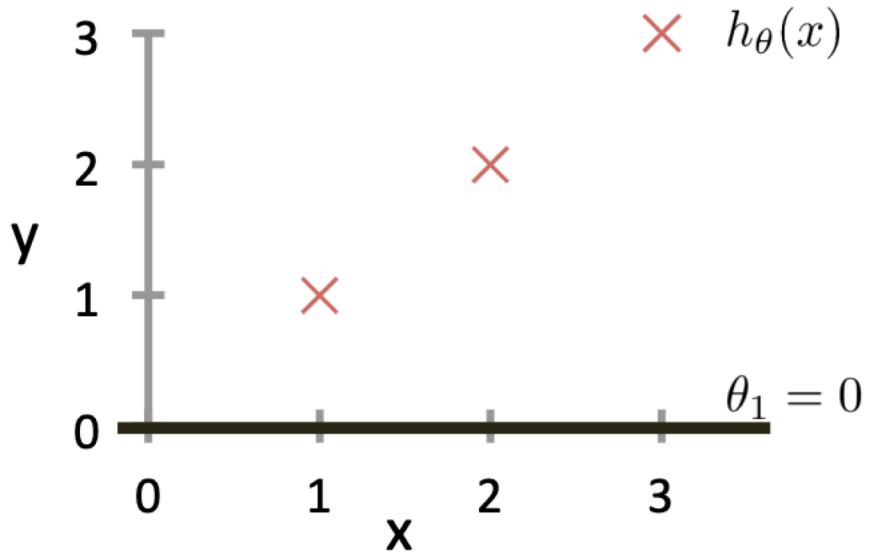
LINEAR REGRESSION – INTUITION BEHIND COST FUNCTION

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

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

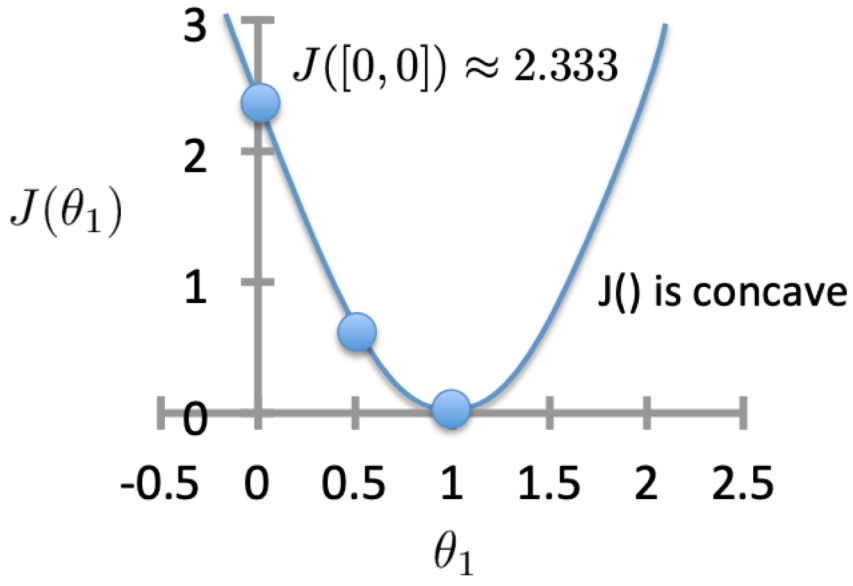
$h_\theta(x)$

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

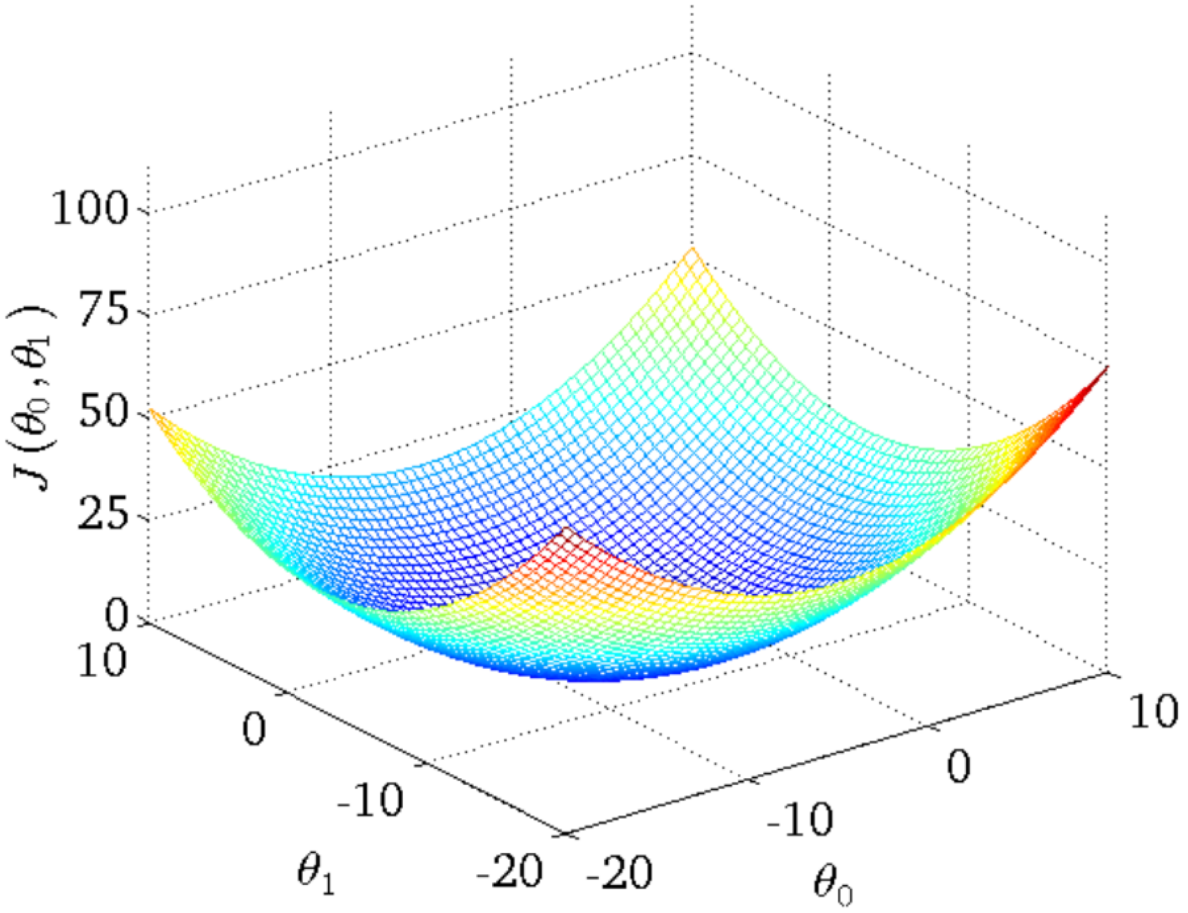


$J(\theta_1)$

(function of the parameter θ_1)



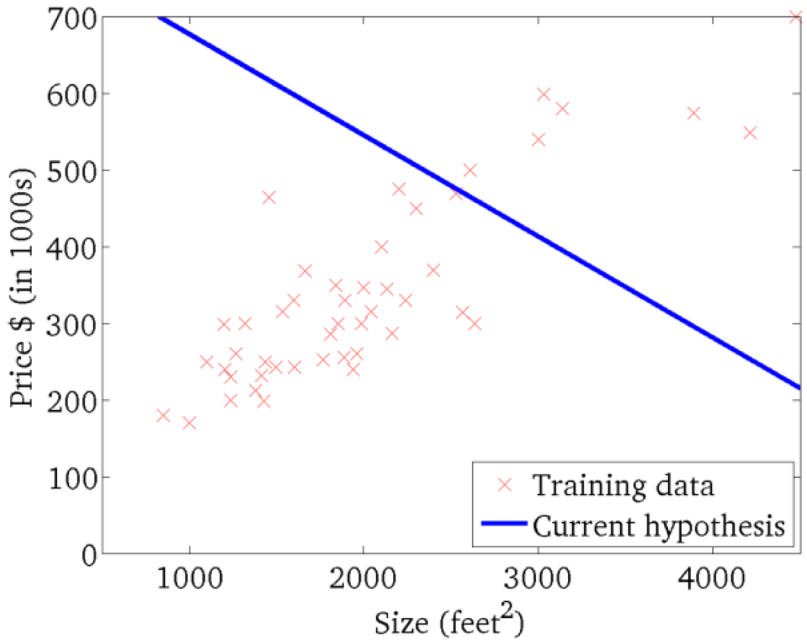
LINEAR REGRESSION – INTUITION BEHIND COST FUNCTION



LINEAR REGRESSION – INTUITION BEHIND COST FUNCTION

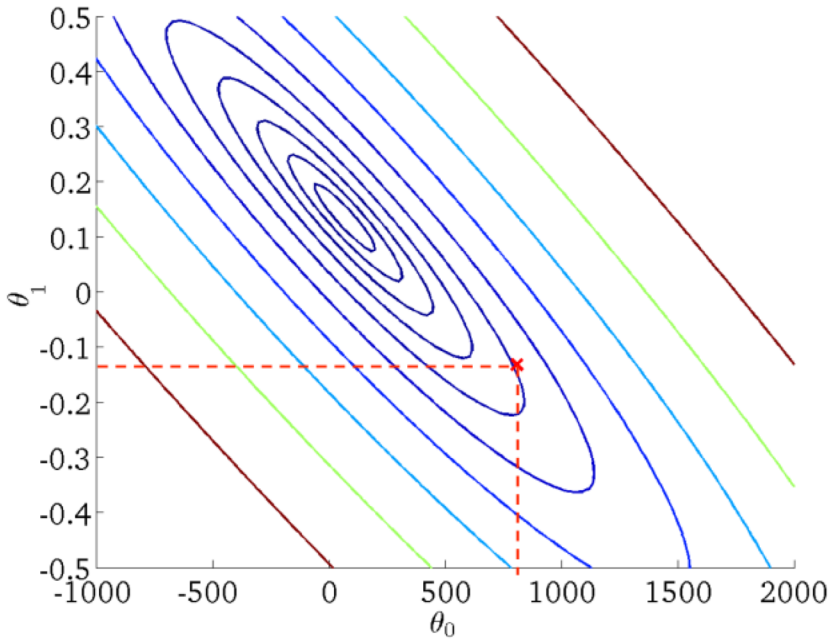
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

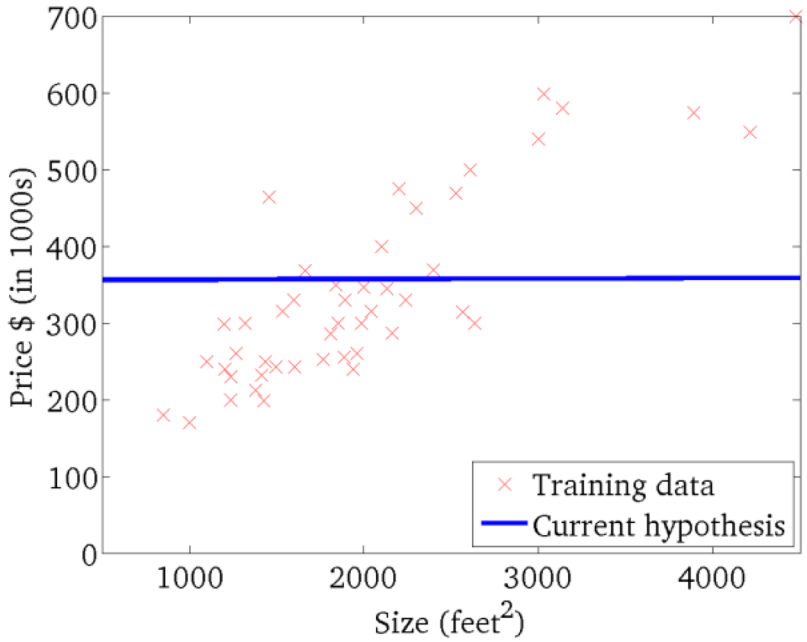
(function of the parameters θ_0, θ_1)



LINEAR REGRESSION – INTUITION BEHIND COST FUNCTION

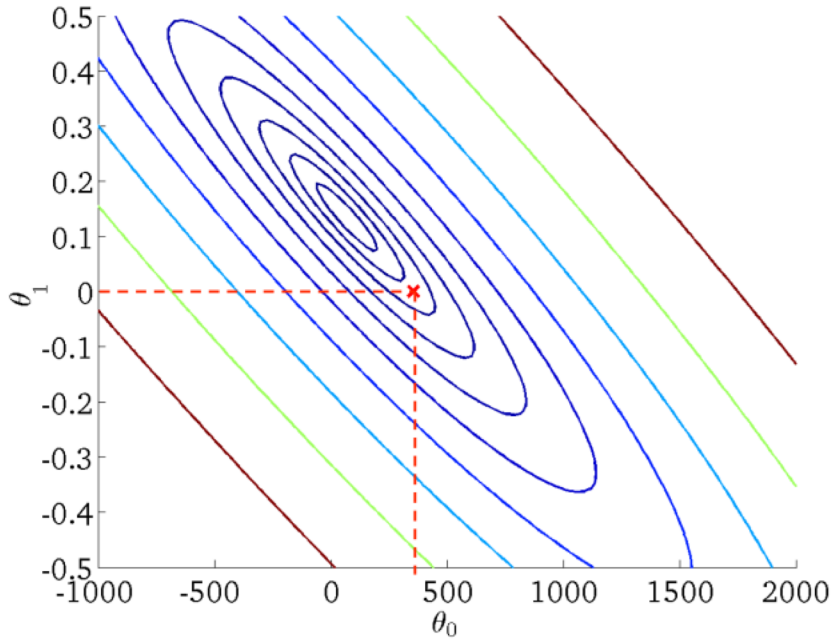
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

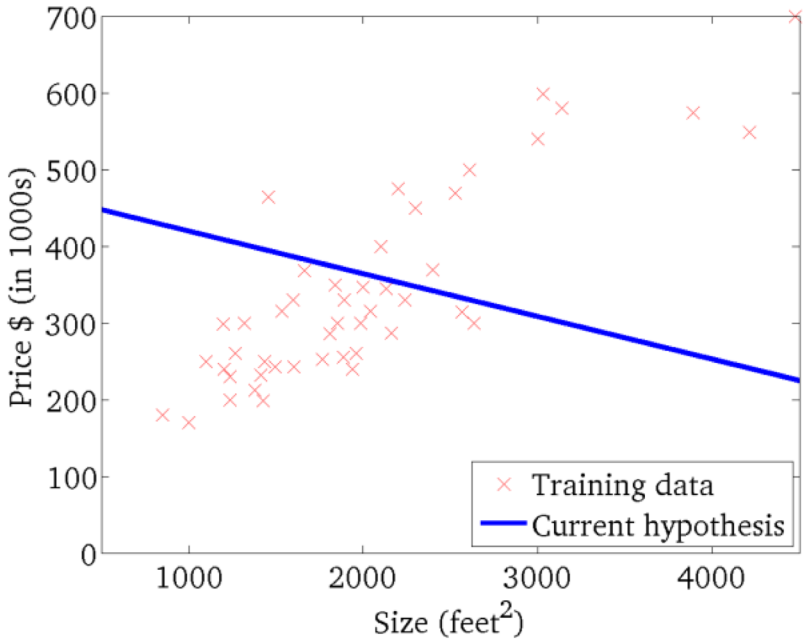
(function of the parameters θ_0, θ_1)



LINEAR REGRESSION – INTUITION BEHIND COST FUNCTION

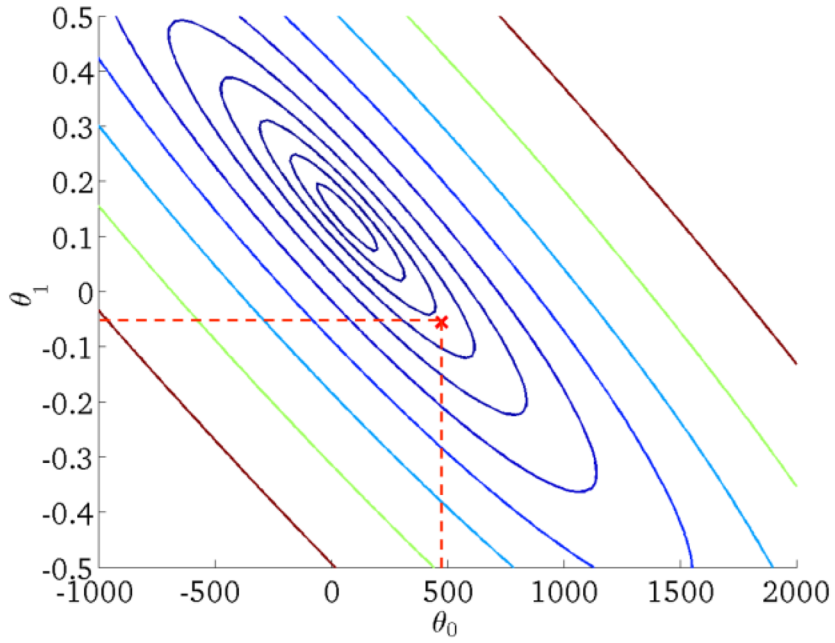
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

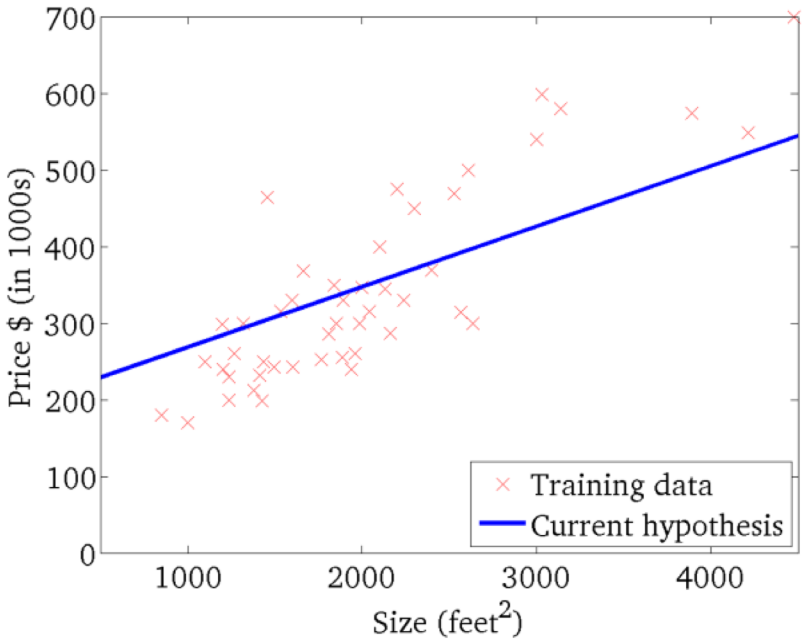
(function of the parameters θ_0, θ_1)



LINEAR REGRESSION – INTUITION BEHIND COST FUNCTION

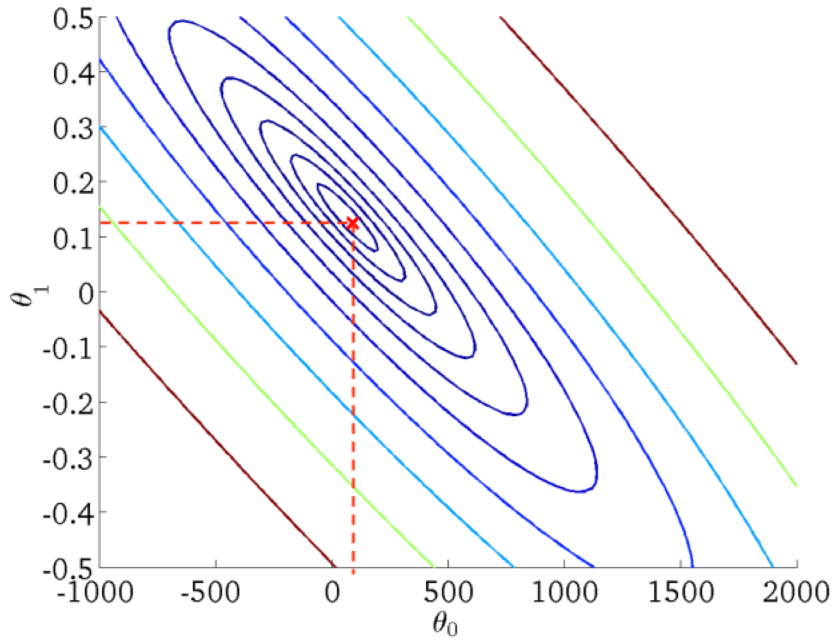
$h_\theta(x)$

(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)



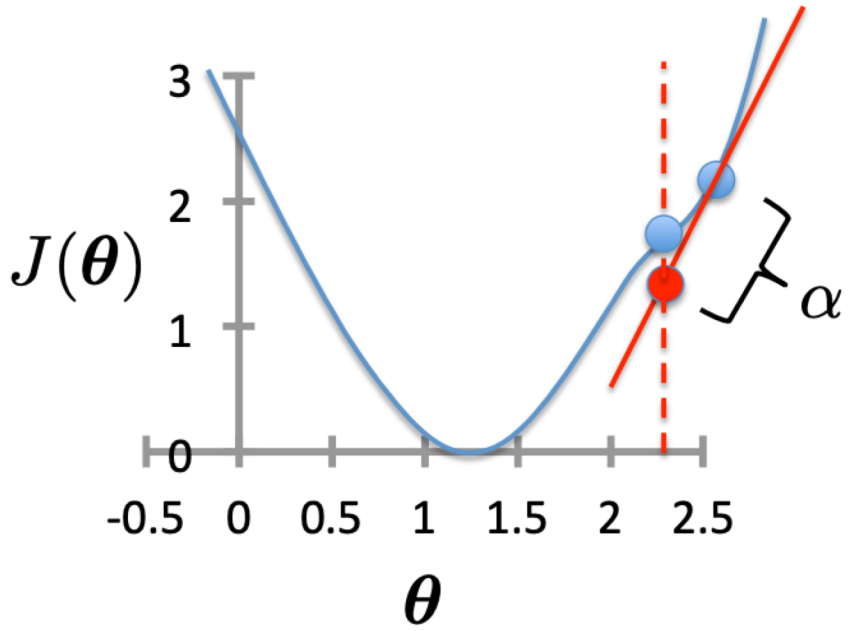
LINEAR REGRESSION – GRADIENT DESCENT

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

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

simultaneous update for $j = 0 \ldots d$

learning rate (small)
e.g., α = 0.05



LINEAR REGRESSION – 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:

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

$$= \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right)^2$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) \times \frac{\partial}{\partial \theta_j} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right)$$

$$= \frac{1}{n} \sum_{i=1}^{n} \left(\sum_{k=0}^{d} \theta_k x_k^{(i)} - y^{(i)} \right) x_j^{(i)}$$

LINEAR REGRESSION – GRADIENT DESCENT

- 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 $j = 0 \ldots d$

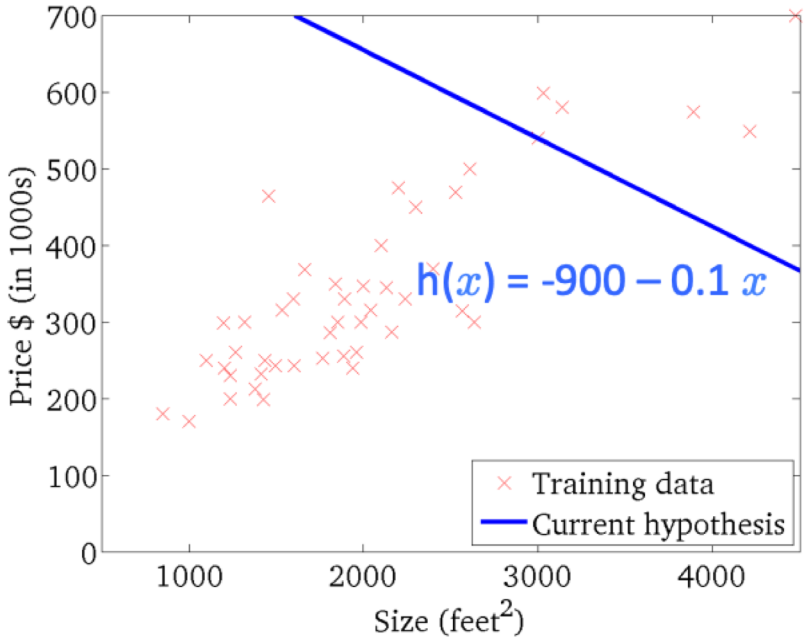
- To achieve simultaneous update
  - At the start of each GD iteration, compute $h_{\boldsymbol{\theta}} \left(\boldsymbol{x}^{(i)} \right)$
  - Use this stored value in the update step loop
- Assume convergence when $\left\| \boldsymbol{\theta}_{new} - \boldsymbol{\theta}_{old} \right\|_2 < \epsilon$

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

LINEAR REGRESSION – GRADIENT DESCENT

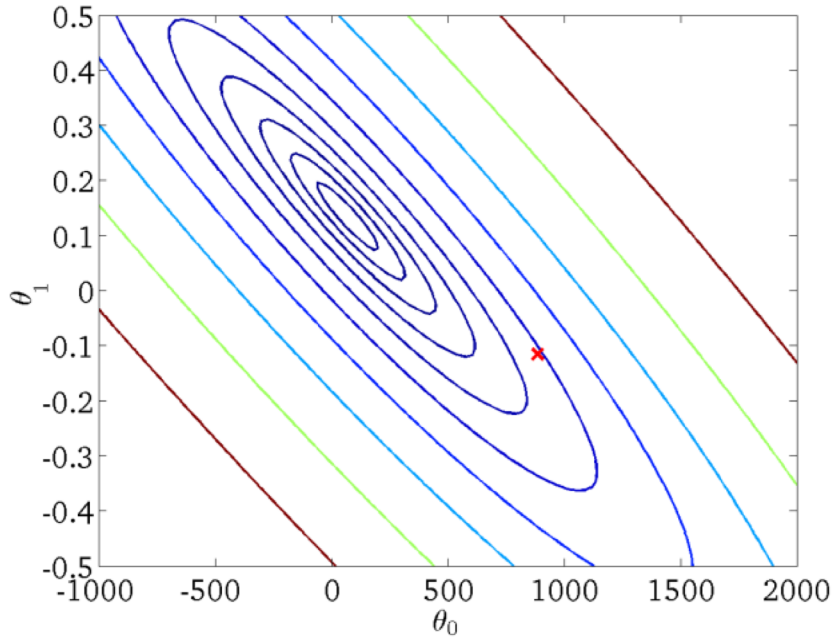
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

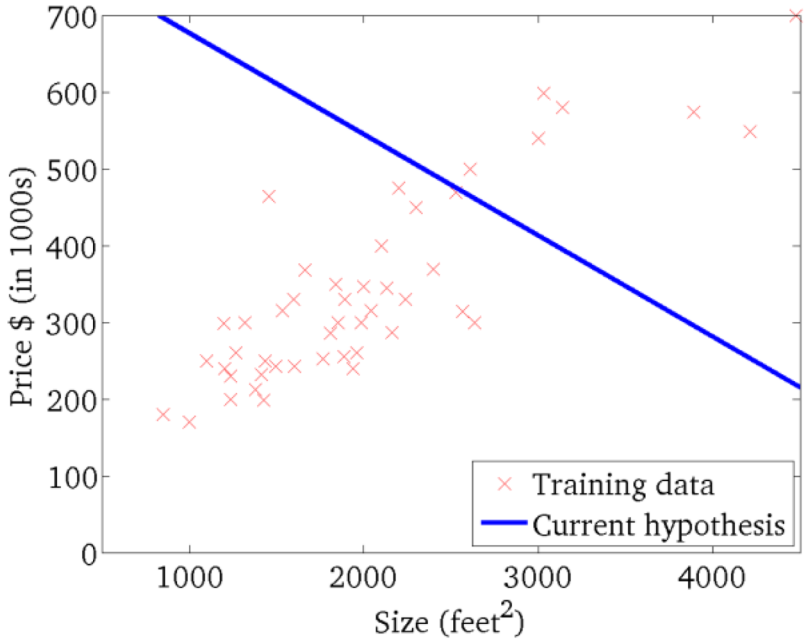
(function of the parameters θ_0, θ_1)



LINEAR REGRESSION – GRADIENT DESCENT

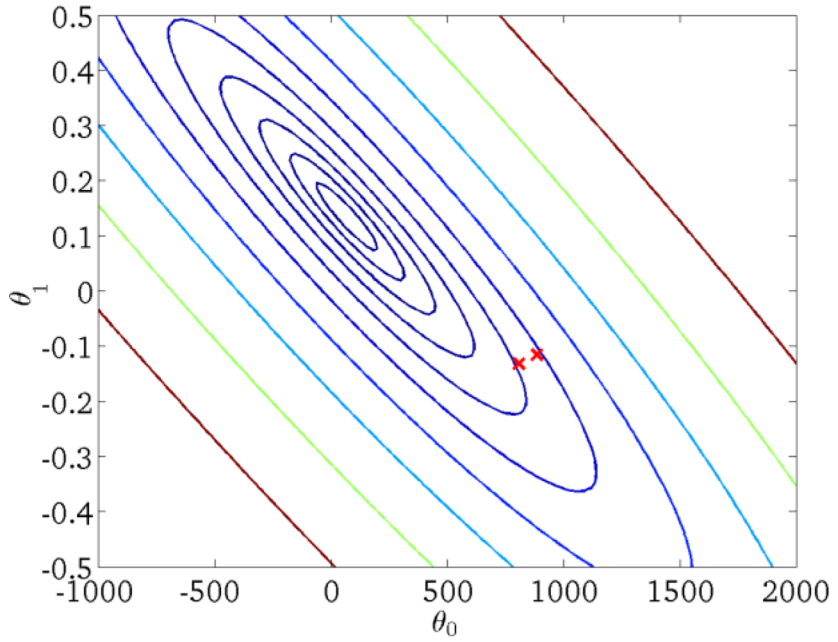
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

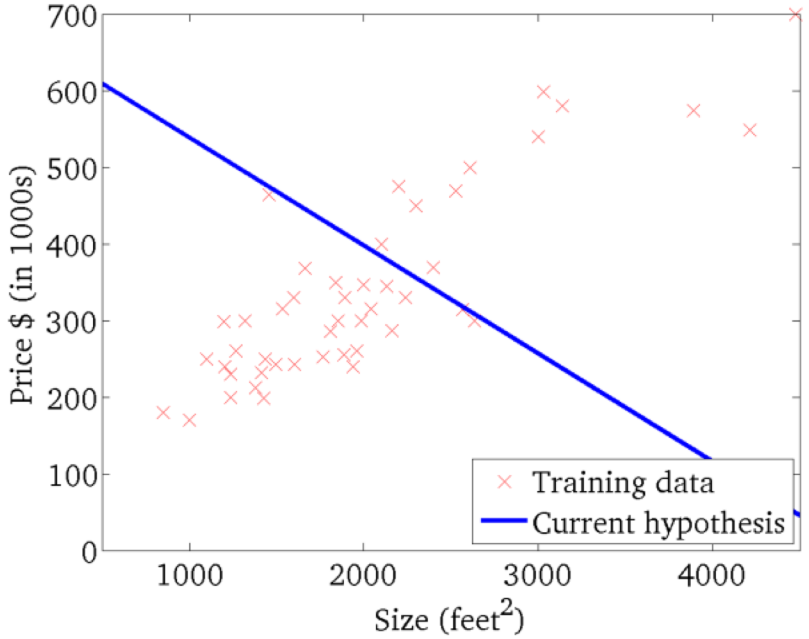
(function of the parameters θ_0, θ_1)



LINEAR REGRESSION – GRADIENT DESCENT

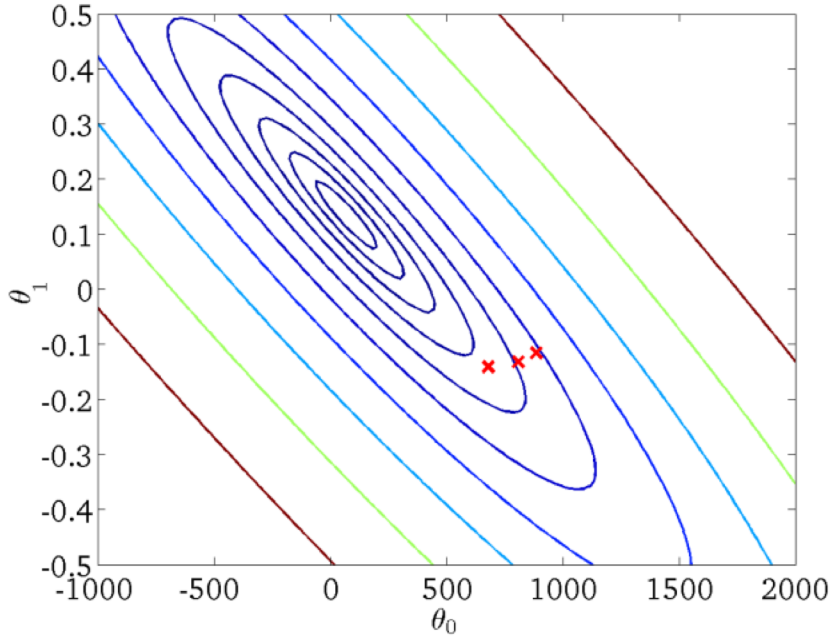
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

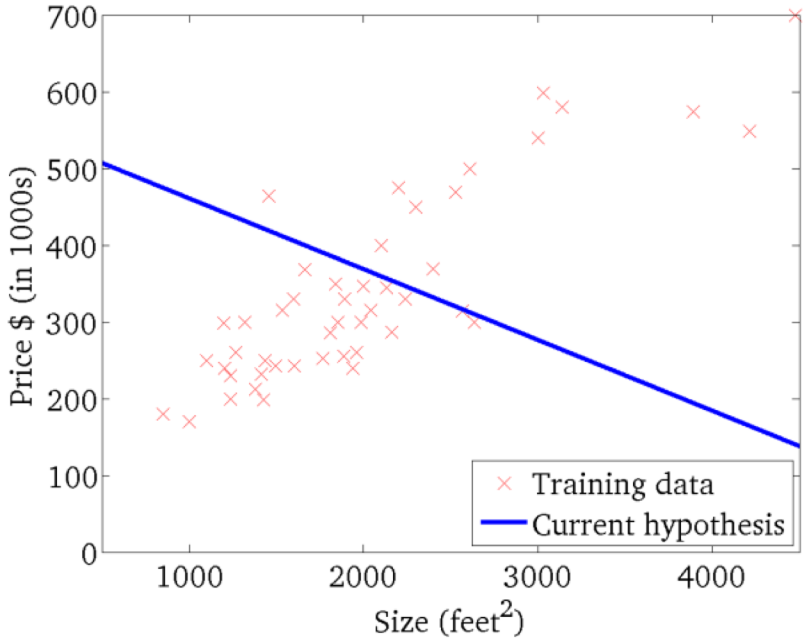
(function of the parameters θ_0, θ_1)



LINEAR REGRESSION – GRADIENT DESCENT

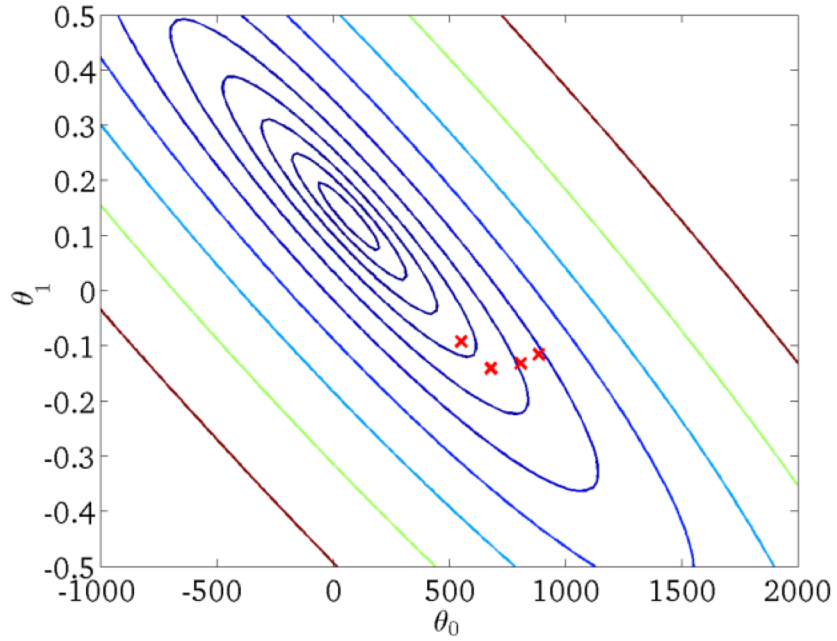
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

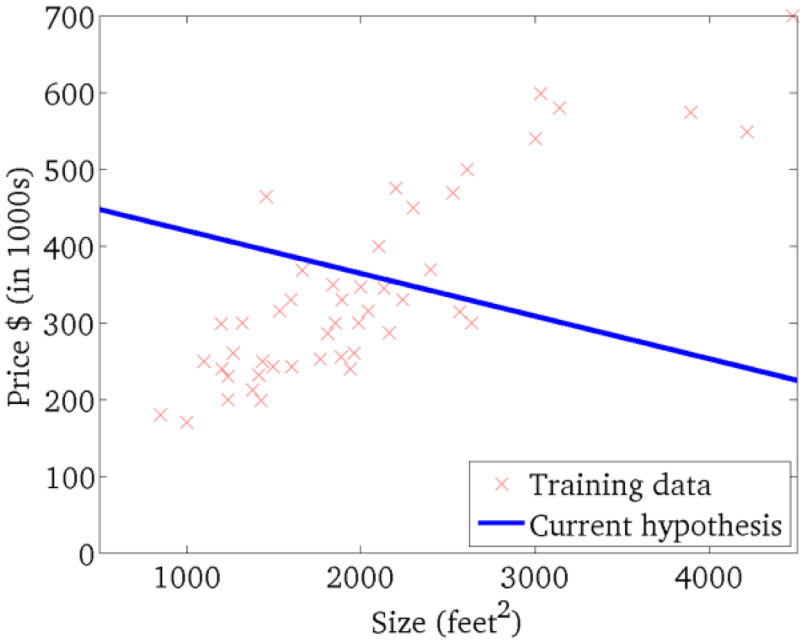
(function of the parameters θ_0, θ_1)



LINEAR REGRESSION – GRADIENT DESCENT

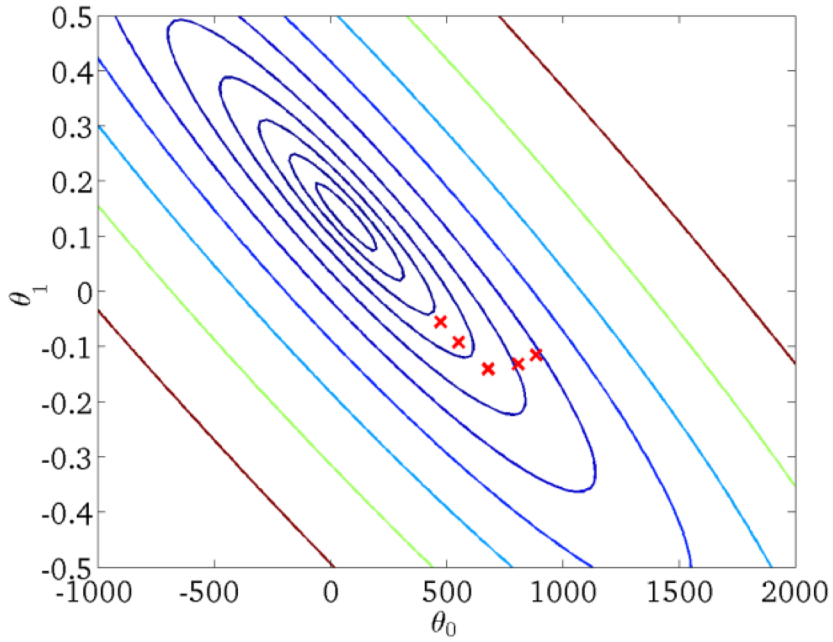
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

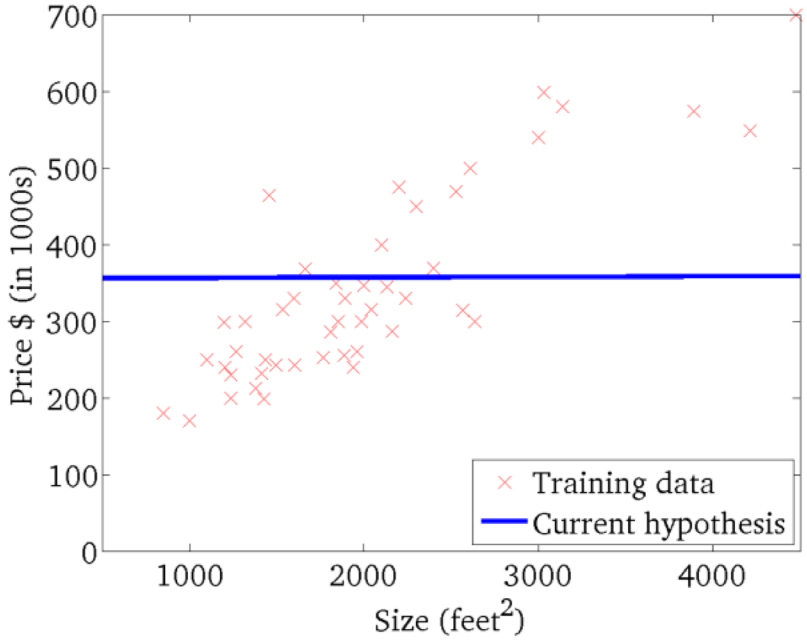
(function of the parameters θ_0, θ_1)



LINEAR REGRESSION – GRADIENT DESCENT

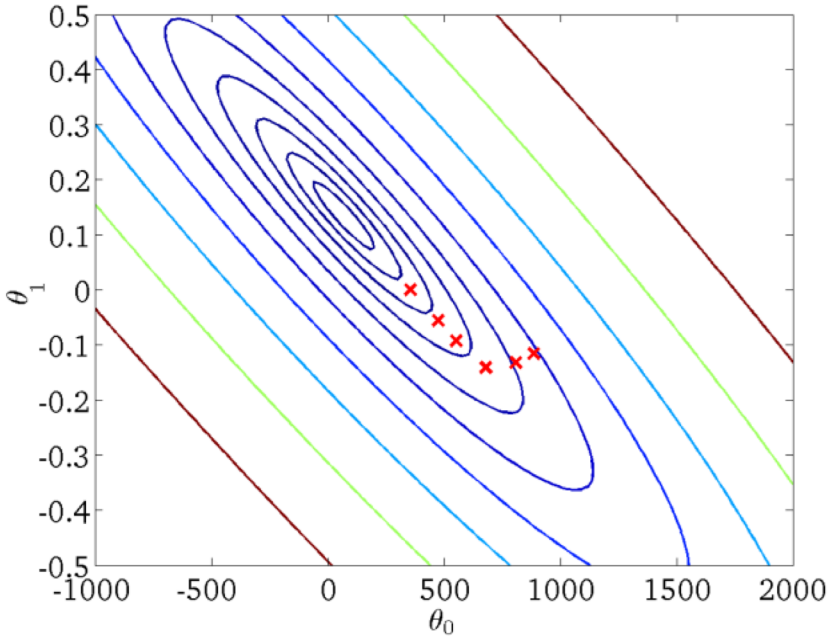
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

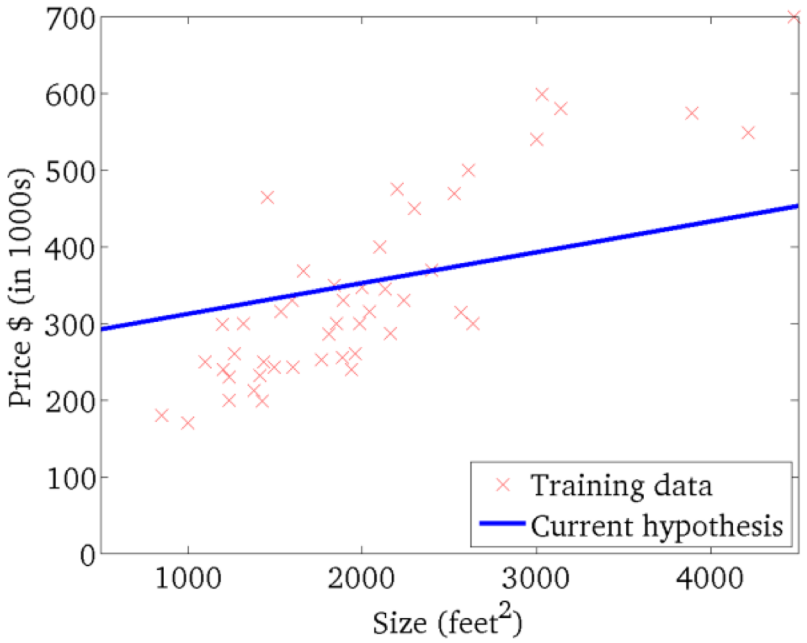
(function of the parameters θ_0, θ_1)



LINEAR REGRESSION – GRADIENT DESCENT

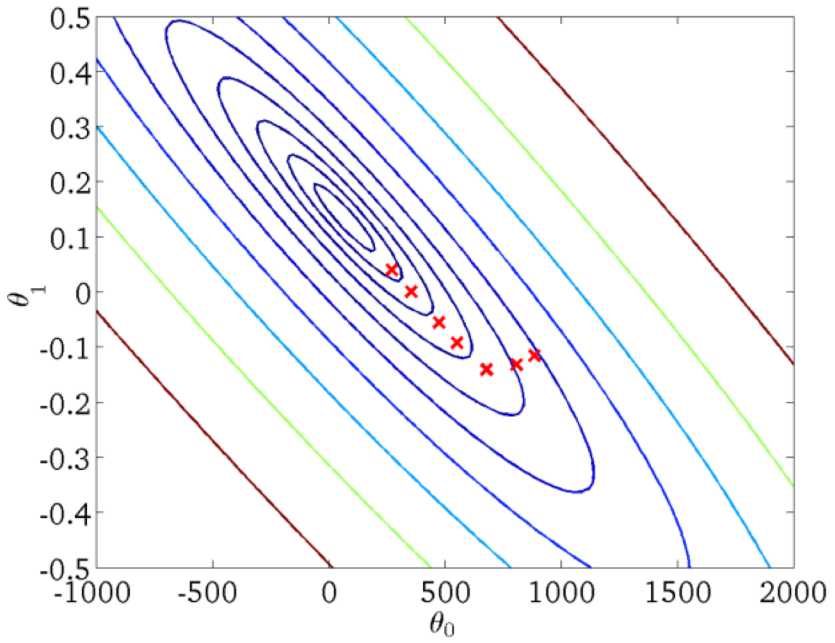
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

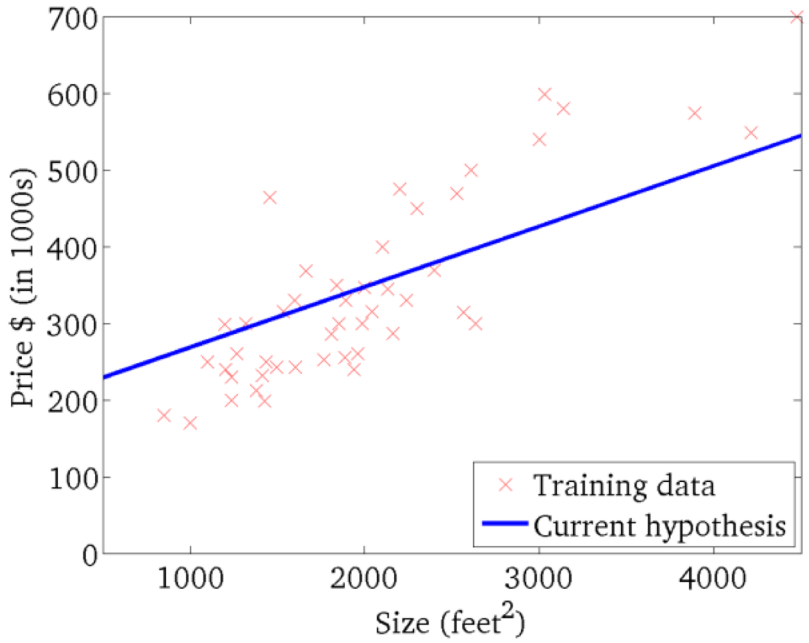
(function of the parameters θ_0, θ_1)



LINEAR REGRESSION – GRADIENT DESCENT

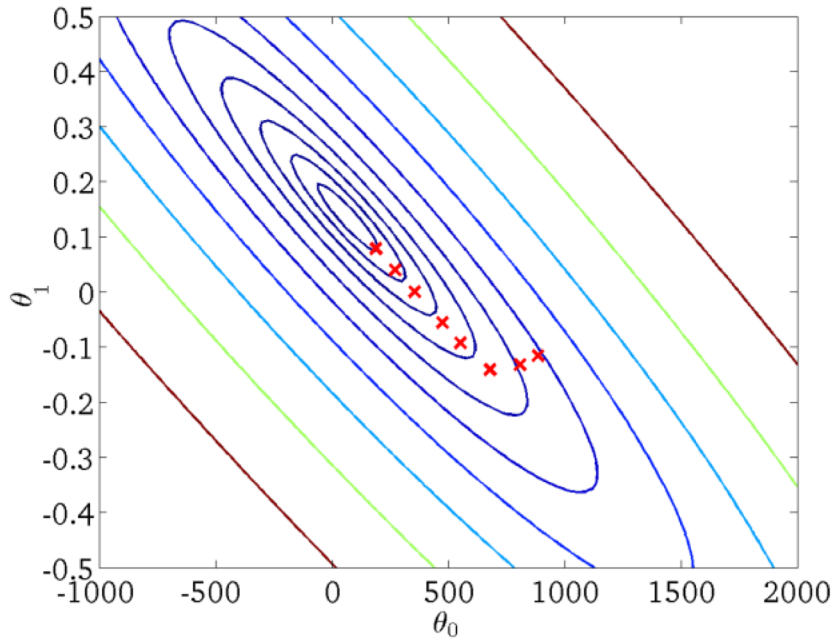
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

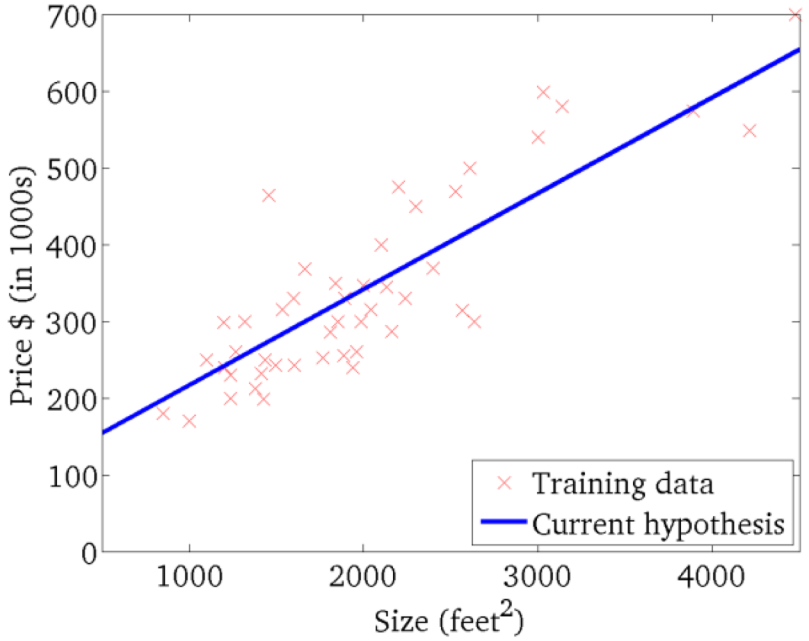
(function of the parameters θ_0, θ_1)



LINEAR REGRESSION – GRADIENT DESCENT

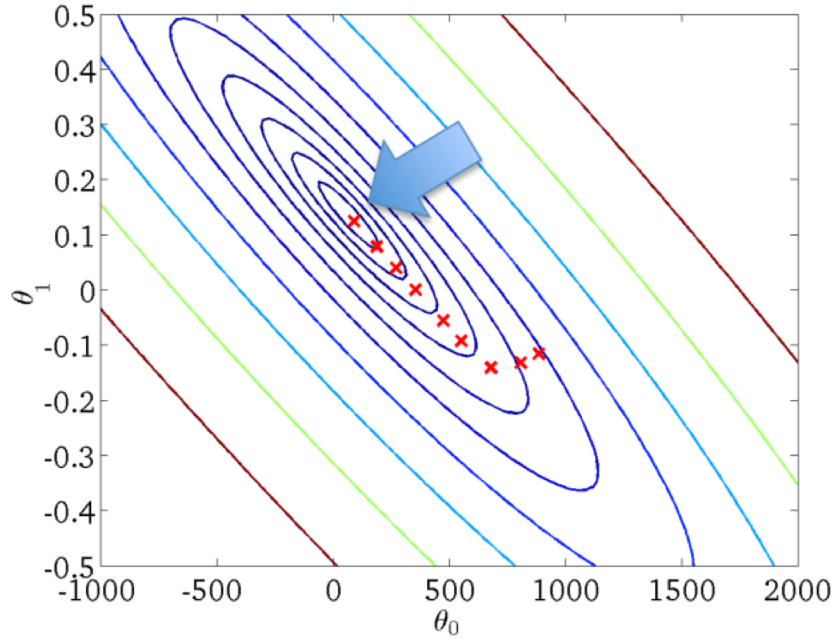
$h_\theta(x)$

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



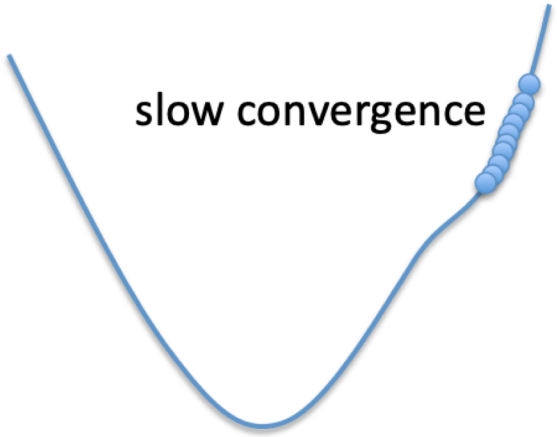
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

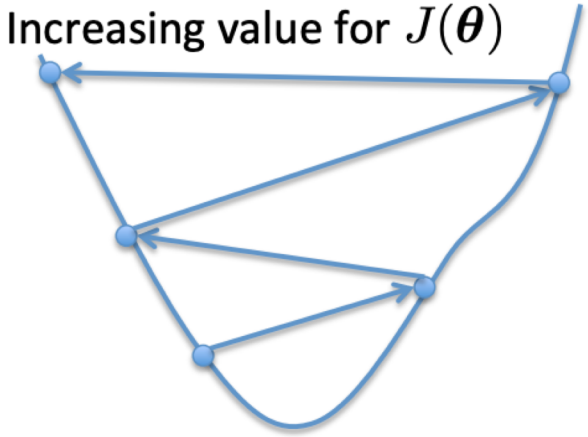


LINEAR REGRESSION – CHOOSING THE LEARNING RATE

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

LINEAR REGRESSION – EXTENDING 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: $x_3 = x_1 \ ? \ x_2$

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

LINEAR REGRESSION – LINEAR BASIS FUNCTION MODELS

- Generally,

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

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 :

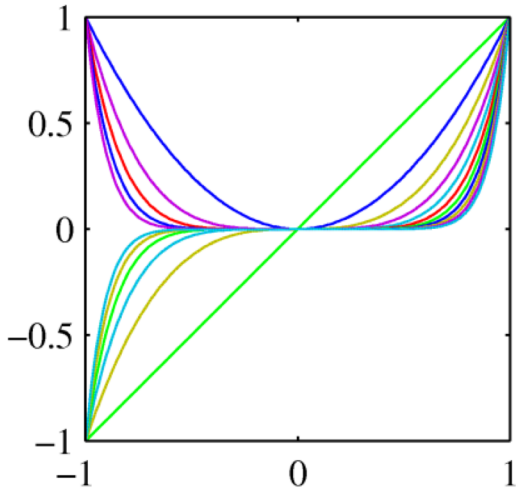
$$\phi_j(\boldsymbol{x}) = x_j$$

LINEAR REGRESSION – LINEAR BASIS FUNCTION MODELS

- Polynomial basis functions:

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

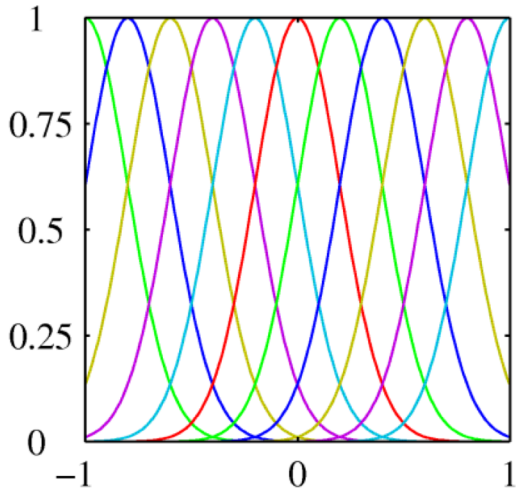
  - These are global; a small change in x affects all basis functions



- 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 REGRESSION – LINEAR BASIS FUNCTION MODELS

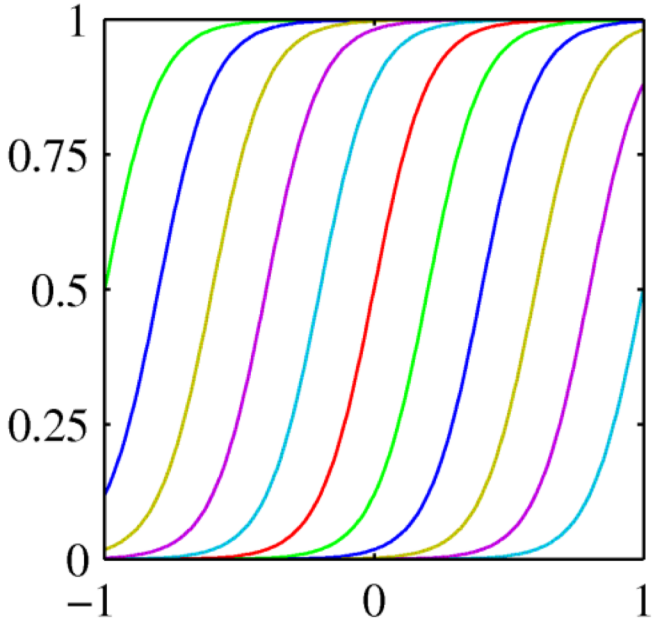
- Sigmoidal basis functions:

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

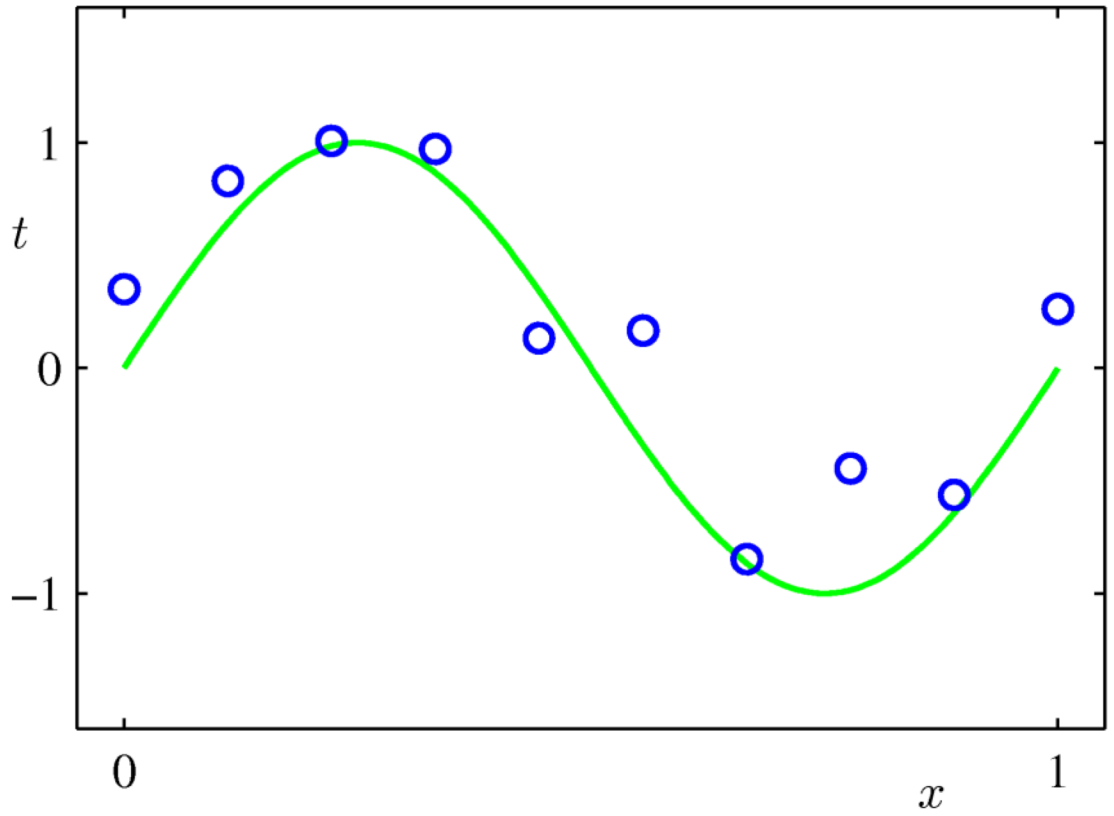
where

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

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



LINEAR REGRESSION – LINEAR BASIS FUNCTION : 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 REGRESSION – 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 REGRESSION – SOME MATH

- *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: $\begin{pmatrix} 1 \\ 6 \\ 3 \\ 4 \end{pmatrix}$
  - as opposed to a row vector: $\begin{pmatrix} 1 & 6 & 3 & 4 \end{pmatrix}$

- 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 REGRESSION – SOME MATH

- 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 REGRESSION – SOME MATH

- Vector dot product: $u \bullet v = \begin{pmatrix} u_1 & u_2 \end{pmatrix} \bullet \begin{pmatrix} v_1 & v_2 \end{pmatrix} = u_1 v_1 + u_2 v_2$
  - 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 REGRESSION – SOME MATH

- 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:

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

LINEAR REGRESSION – VACTORIZATION

- 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 $h(\boldsymbol{x}) = \boldsymbol{\theta}^\intercal \boldsymbol{x}$

LINEAR REGRESSION – VACTORIZATION

- 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 $h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \boldsymbol{X}\boldsymbol{\theta}$

LINEAR REGRESSION – VACTORIZATION

- For the linear regression cost function:

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

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

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

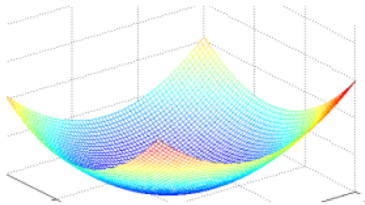
$\boldsymbol{X}: \mathbb{R}^{n\times(d+1)}$, $\boldsymbol{\theta}: \mathbb{R}^{(d+1)\times 1}$

Let:

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

LINEAR REGRESSION – 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

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

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

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

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

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

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

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

LINEAR REGRESSION – CLOSED FORM SOLUTION

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

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

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

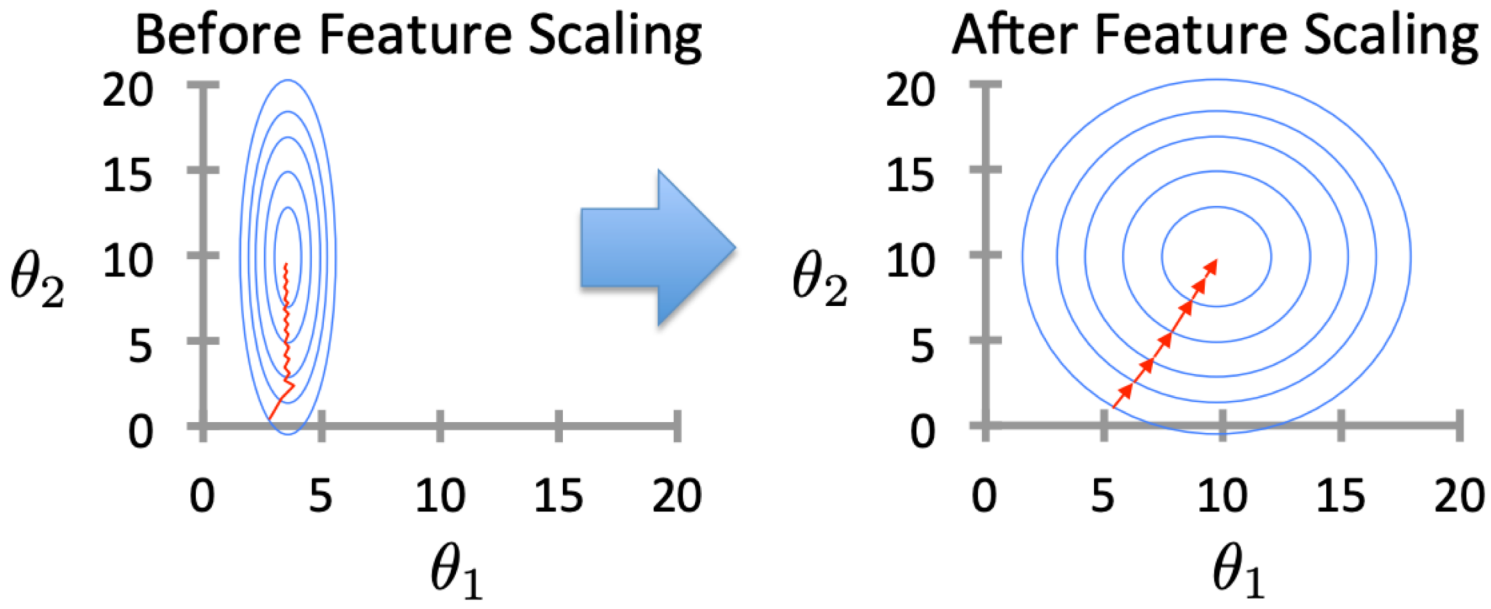
- If $\boldsymbol{X}^\intercal \boldsymbol{X}$ is not invertible (i.e., singular), may need to:
  - Use pseudo-inverse instead of the inverse
    - In python, `numpy.linalg.pinv(a)`
  - Remove redundant (not linearly independent) features
  - Remove extra features to ensure that $d \leq n$

LINEAR REGRESSION – GRADIENT DESCENT V/S CLOSED FORM SOLUTION

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 $O(n^3)$

LINEAR REGRESSION – IMPROVING LEARNING : FEATURE SCALING

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



- Makes gradient descent converge *much* faster

LINEAR REGRESSION – FEATURE STANDARDIZATION

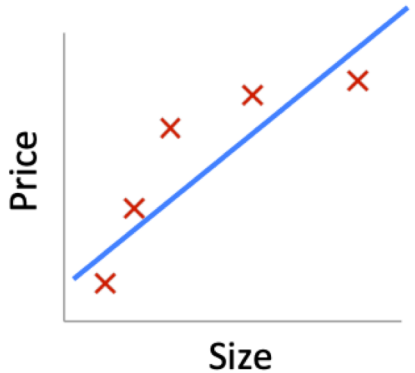
- Rescales features to have zero mean and unit variance
  - Let μ_j be the mean of feature j: $\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_j^{(i)}$
  - Replace each value with:

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

    - s_j is the standard deviation of feature j
    - Could also use the range of feature j ($\max_j - \min_j$) for s_j

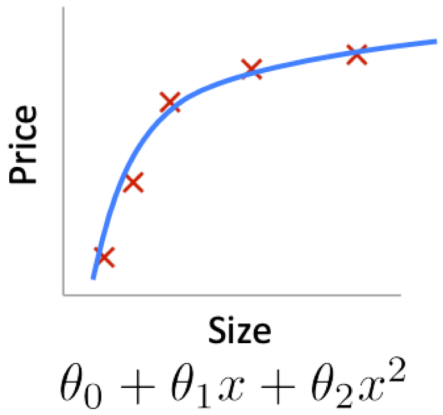
- Must apply the same transformation to instances for both training and prediction
- Outliers can cause problems

LINEAR REGRESSION – QUALITY OF FIT



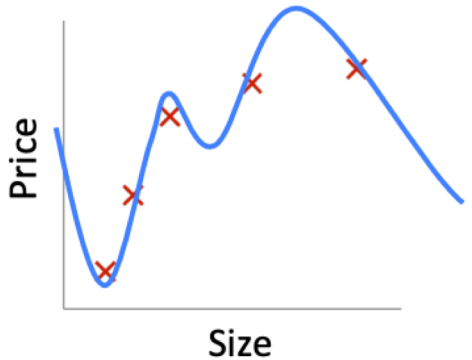
$\theta_0 + \theta_1 x$

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

LINEAR REGRESSION – 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)

LINEAR REGRESSION – REGULARIZATION

- Linear regression objective function

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

model fit to data — regularization

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

LINEAR REGRESSION – 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:

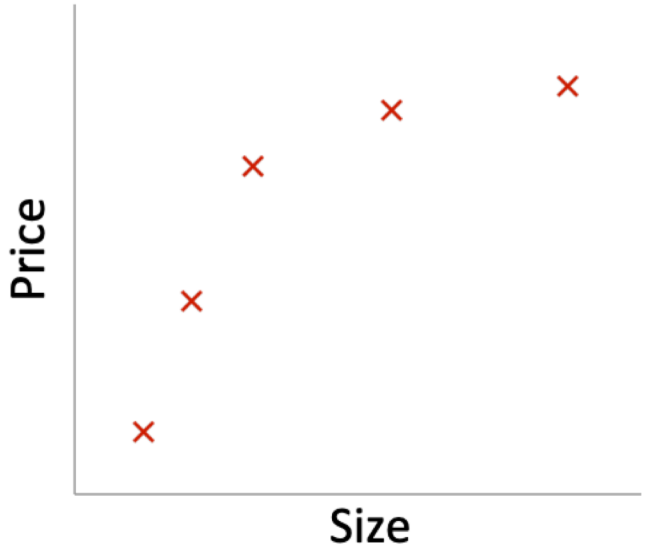
$$\sum_{j=1}^{d}(\theta_j - 0)^2 = \|\boldsymbol{\theta}_{1:d} - \vec{\mathbf{0}}\|_2^2$$

  - L_2 regularization pulls coefficients toward 0

LINEAR REGRESSION – 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})?

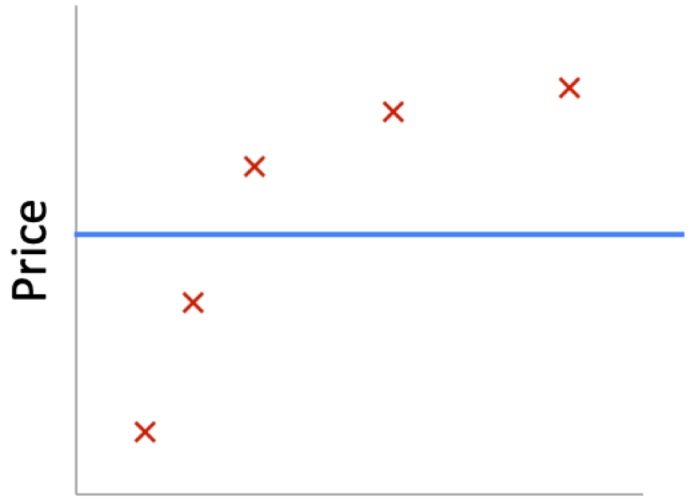


$$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

LINEAR REGRESSION – 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

LINEAR REGRESSION – 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:

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

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

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

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

regularization

LINEAR REGRESSION – 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:

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

LINEAR REGRESSION – 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

Linear regression – without scikit learn!

LINEAR REGRESSION

Cdiscount

Our goal is to learn a linear model $\hat{y}$ that models y given X using weights W and bias b:

$$\hat{y} = XW + b$$

Variable	Description
N	total numbers of samples
$\hat{y}$	predictions $\in \mathbb{R}^{NX1}$
X	inputs $\in \mathbb{R}^{NXD}$
W	weights $\in \mathbb{R}^{DX1}$
b	bias $\in \mathbb{R}^{1}$

KEYRUS data

LINEAR REGRESSION

- **Objective**:
  - Use inputs X to predict the output $\hat{y}$ using a linear model. The model will be a line of best fit that minimizes the distance between the predicted (model's output) and target (ground truth) values. Training data (X, y) is used to train the model and learn the weights W using gradient descent.
- **Advantages**:
  - Computationally simple.
  - Highly interpretable.
  - Can account for continuous and categorical features.

- **Disadvantages**:
  - The model will perform well only when the data is linearly separable (for classification).

LINEAR REGRESSION : GENERATE DATA

We're going to generate some simple dummy data to apply linear regression on. It's going to create roughly linear data (`y = 3.5X + noise`); the random noise is added to create realistic data that doesn't perfectly align in a line. Our goal is to have the model converge to a similar linear equation (there will be slight variance since we added some noise).

```
1  import numpy as np
2  import pandas as pd
3  import matplotlib.pyplot as plt
```

```
1  SEED = 1234
2  NUM_SAMPLES = 50
```

```
1  # Set seed for reproducibility
2  np.random.seed(SEED)
```

LINEAR REGRESSION : GENERATE DATA

```python
# Generate synthetic data
def generate_data(num_samples):
    """Generate dummy data for linear regression."""
    X = np.array(range(num_samples))
    random_noise = np.random.uniform(-10, 20, size=num_samples)
    y = 3.5*X + random_noise # add some noise
    return X, y
```

```python
# Generate random (linear) data
X, y = generate_data(num_samples=NUM_SAMPLES)
data = np.vstack([X, y]).T
print (data[:5])
```

```
[[ 0.         -4.25441649]
 [ 1.         12.16326313]
 [ 2.         10.13183217]
 [ 3.         24.06075751]
 [ 4.         27.39927424]]
```

LINEAR REGRESSION : GENERATE DATA

```python
# Generate synthetic data
def generate_data(num_samples):
    """Generate dummy data for linear regression."""
    X = np.array(range(num_samples))
    random_noise = np.random.uniform(-10, 20, size=num_samples)
    y = 3.5*X + random_noise # add some noise
    return X, y
```

```python
# Generate random (linear) data
X, y = generate_data(num_samples=NUM_SAMPLES)
data = np.vstack([X, y]).T
print (data[:5])
```

```
[[ 0.         -4.25441649]
 [ 1.         12.16326313]
 [ 2.         10.13183217]
 [ 3.         24.06075751]
 [ 4.         27.39927424]]
```

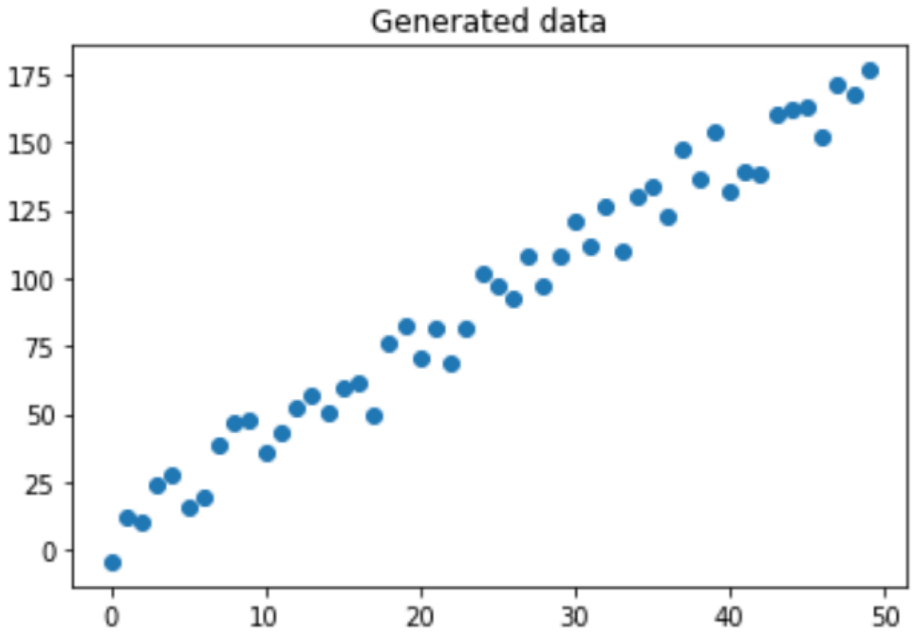
LINEAR REGRESSION : LOAD DATA

```
# Load into a Pandas DataFrame
df = pd.DataFrame(data, columns=["X", "y"])
X = df[["X"]].values
y = df[["y"]].values
df.head()
```

	X	y
0	0.0	-4.254416
1	1.0	12.163263
2	2.0	10.131832
3	3.0	24.060758
4	4.0	27.399274

LINEAR REGRESSION : PLOT DATA

```
1  # Scatter plot
2  plt.title("Generated data")
3  plt.scatter(x=df["X"], y=df["y"])
4  plt.show()
```



LINEAR REGRESSION : WITHOUT SCIKITLEARN

Since our task is a regression task, we will randomly split our dataset into three sets: train, validation and test data splits.

- `train` : used to train our model.
- `val` : used to validate our model's performance during training.
- `test` : used to do an evaluation of our fully trained model.

```
1  TRAIN_SIZE = 0.7
2  VAL_SIZE = 0.15
3  TEST_SIZE = 0.15
```

```
1  # Shuffle data
2  indices = list(range(NUM_SAMPLES))
3  np.random.shuffle(indices)
4  X = X[indices]
5  y = y[indices]
```

LINEAR REGRESSION : WITHOUT SCIKITLEARN

```python
# Split indices
train_start = 0
train_end = int(0.7*NUM_SAMPLES)
val_start = train_end
val_end = int((TRAIN_SIZE+VAL_SIZE)*NUM_SAMPLES)
test_start = val_end
```

```python
# Split data
X_train = X[train_start:train_end]
y_train = y[train_start:train_end]
X_val = X[val_start:val_end]
y_val = y[val_start:val_end]
X_test = X[test_start:]
y_test = y[test_start:]
print (f"X_train: {X_train.shape}, y_train: {y_train.shape}")
print (f"X_val: {X_val.shape}, y_test: {y_val.shape}")
print (f"X_test: {X_test.shape}, y_test: {y_test.shape}")
```

```
X_train: (35, 1), y_train: (35, 1)
X_val: (7, 1), y_test: (7, 1)
X_test: (8, 1), y_test: (8, 1)
```

LINEAR REGRESSION : STANDARDIZE DATA

We need to standardize our data (zero mean and unit variance) so a specific feature's magnitude doesn't affect how the model learns its weights.

$$z = \frac{x_i - \mu}{\sigma}$$

Variable	Description
z	standardized value
x_i	inputs
μ	mean
σ	standard deviation

LINEAR REGRESSION : STANDARDIZE DATA

```python
def standardize_data(data, mean, std):
    return (data - mean)/std
```

```python
# Determine means and stds
X_mean = np.mean(X_train)
X_std = np.std(X_train)
y_mean = np.mean(y_train)
y_std = np.std(y_train)
```

We need to treat the validation and test sets as if they were hidden datasets. So we only use the train set to determine the mean and std to avoid biasing our training process.

LINEAR REGRESSION : STANDARDIZE DATA

```python
# Standardize
X_train = standardize_data(X_train, X_mean, X_std)
y_train = standardize_data(y_train, y_mean, y_std)
X_val = standardize_data(X_val, X_mean, X_std)
y_val = standardize_data(y_val, y_mean, y_std)
X_test = standardize_data(X_test, X_mean, X_std)
y_test = standardize_data(y_test, y_mean, y_std)
```

```python
# Check (means should be ~0 and std should be ~1)
# Check (means should be ~0 and std should be ~1)
print (f"mean: {np.mean(X_test, axis=0)[0]:.1f}, std: {np.std(X_test, axis=0)
print (f"mean: {np.mean(y_test, axis=0)[0]:.1f}, std: {np.std(y_test, axis=0)
```

```
mean: -0.4, std: 0.9
mean: -0.3, std: 1.0
```

LINEAR REGRESSION : WEIGHTS

Our goal is to learn a linear model $\hat{y}$ that models y given X using weights W and bias b
$\rightarrow \hat{y} = XW + b$

`Step 1` : Randomly initialize the model's weights W.

```
1  INPUT_DIM = X_train.shape[1] # X is 1-dimensional
2  OUTPUT_DIM = y_train.shape[1] # y is 1-dimensional
```

```
1  # Initialize random weights
2  W = 0.01 * np.random.randn(INPUT_DIM, OUTPUT_DIM)
3  b = np.zeros((1, 1))
4  print (f"W: {W.shape}")
5  print (f"b: {b.shape}")
```

```
W: (1, 1)
b: (1, 1)
```

LINEAR REGRESSION : MODEL

`Step 2` : Feed inputs X into the model to receive the predictions $\hat{y}$

```
1  # Forward pass [NX1] · [1X1] = [NX1]
2  y_pred = np.dot(X_train, W) + b
3  print (f"y_pred: {y_pred.shape}")
```

```
y_pred: (35, 1)
```

LINEAR REGRESSION : LOSS

`Step 3` : Compare the predictions $\hat{y}$ with the actual target values y using the objective (cost) function to determine the loss J. A common objective function for linear regression is mean squared error (MSE). This function calculates the difference between the predicted and target values and squares it.

$$J(\theta) = MSE = \frac{1}{N}\sum_{i-1}^{N}(y_i - \hat{y}_i)^2$$

```python
# Loss
N = len(y_train)
loss = (1/N) * np.sum((y_train - y_pred)**2)
print (f"loss: {loss:.2f}")
```

```
loss: 0.99
```

LINEAR REGRESSION : GRADIENT

`Step 4` : Calculate the gradient of loss $J(\theta)$ w.r.t to the model weights.

$$J(\theta) = \frac{1}{N}\sum_i (y_i - \hat{y}_i)^2 = \frac{1}{N}\sum_i (y_i - X_i W)^2$$

$$\rightarrow \frac{\partial J}{\partial W} = -\frac{2}{N}\sum_i (y_i - X_i W) X_i = -\frac{2}{N}\sum_i (y_i - \hat{y}_i) X_i$$

$$\rightarrow \frac{\partial J}{\partial b} = -\frac{2}{N}\sum_i (y_i - X_i W) 1 = -\frac{2}{N}\sum_i (y_i - \hat{y}_i) 1$$

```
# Backpropagation
dW = -(2/N) * np.sum((y_train - y_pred) * X_train)
db = -(2/N) * np.sum((y_train - y_pred) * 1)
```

LINEAR REGRESSION : GRADIENT

> The gradient is the derivative, or the rate of change of a function. It's a vector that points in the direction of greatest increase of a function. For example the gradient of our loss function (J) with respect to our weights (W) will tell us how to change W so we can maximize J. However, we want to minimize our loss so we subtract the gradient from W.

LINEAR REGRESSION : UPDATE WEIGHTS

`Step 5` : Update the weights W using a small learning rate α.

$$W = W - \alpha \frac{\partial J}{\partial W}$$

$$b = b - \alpha \frac{\partial J}{\partial b}$$

```
LEARNING_RATE = 1e-1
```

```
# Update weights
W += -LEARNING_RATE * dW
b += -LEARNING_RATE * db
```

LINEAR REGRESSION : UPDATE WEIGHTS

> The learning rate α is a way to control how much we update the weights by. If we choose a small learning rate, it may take a long time for our model to train. However, if we choose a large learning rate, we may overshoot and our training will never converge. The specific learning rate depends on our data and the type of models we use but it's typically good to explore in the range of $[1e^{-8}, 1e^{-1}]$. We'll explore learning rate update strategies in later lessons.

LINEAR REGRESSION : TRAINING

Cdiscount

`Step 6` : Repeat steps 2 - 5 to minimize the loss and train the model.

```
NUM_EPOCHS = 100
```

```
# Initialize random weights
W = 0.01 * np.random.randn(INPUT_DIM, OUTPUT_DIM)
b = np.zeros((1, ))

# Training loop
for epoch_num in range(NUM_EPOCHS):

    # Forward pass [NX1] · [1X1] = [NX1]
    y_pred = np.dot(X_train, W) + b

    # Loss
    loss = (1/len(y_train)) * np.sum((y_train - y_pred)**2)

    # Show progress
    if epoch_num%10 == 0:
        print (f"Epoch: {epoch_num}, loss: {loss:.3f}")

    # Backpropagation
    dW = -(2/N) * np.sum((y_train - y_pred) * X_train)
    db = -(2/N) * np.sum((y_train - y_pred) * 1)

    # Update weights
    W += -LEARNING_RATE * dW
    b += -LEARNING_RATE * db
```

KEYRUS data

LINEAR REGRESSION : TRAINING

```
Epoch: 0, loss: 0.990
Epoch: 10, loss: 0.039
Epoch: 20, loss: 0.028
Epoch: 30, loss: 0.028
Epoch: 40, loss: 0.028
Epoch: 50, loss: 0.028
Epoch: 60, loss: 0.028
Epoch: 70, loss: 0.028
Epoch: 80, loss: 0.028
Epoch: 90, loss: 0.028
```

LINEAR REGRESSION : EVALUATION

Now we're ready to see how well our trained model will perform on our test (hold-out) data split. This will be our best measure on how well the model would perform on the real world, given that our dataset's distribution is close to unseen data.

```
1  # Predictions
2  pred_train = W*X_train + b
3  pred_test = W*X_test + b
```

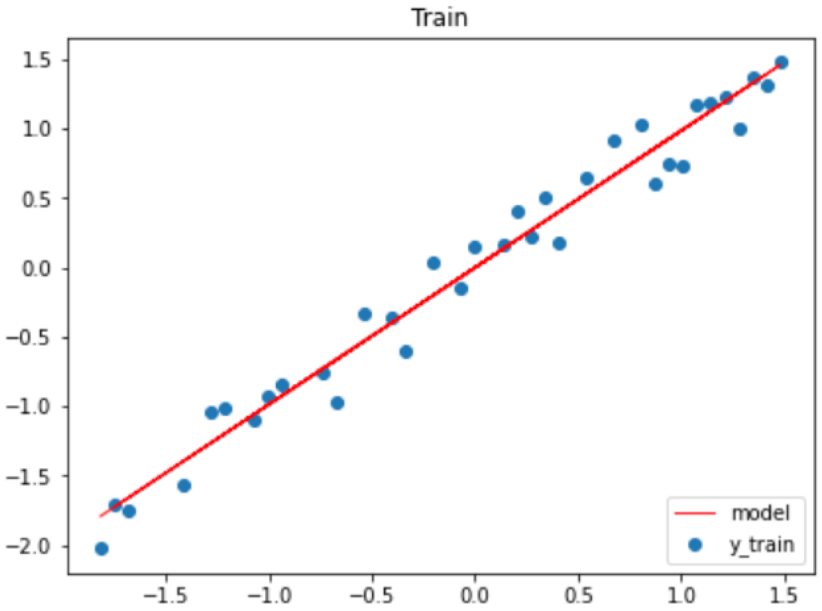
```
1  # Train and test MSE
2  train_mse = np.mean((y_train - pred_train) ** 2)
3  test_mse = np.mean((y_test - pred_test) ** 2)
4  print (f"train_MSE: {train_mse:.2f}, test_MSE: {test_mse:.2f}")
```

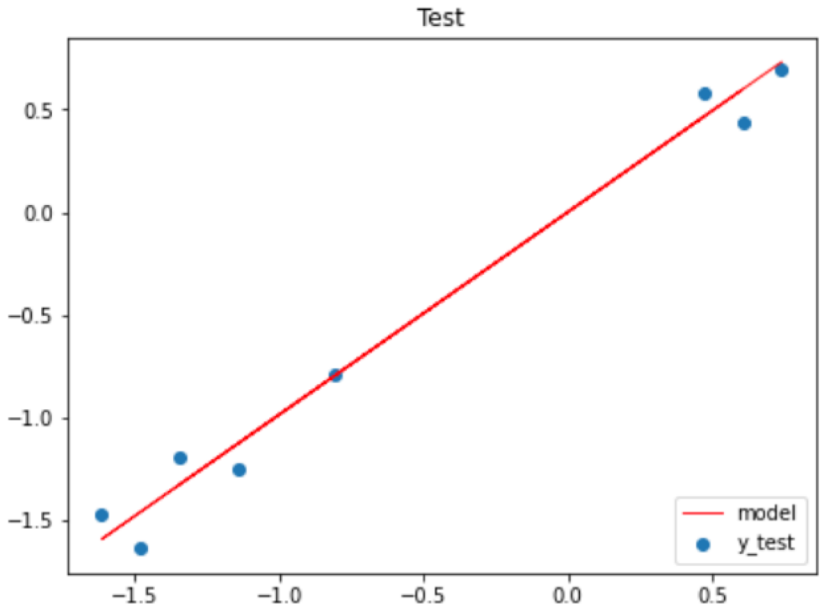
```
train_MSE: 0.03, test_MSE: 0.01
```

LINEAR REGRESSION : EVALUATION

```
# Figure size
plt.figure(figsize=(15,5))

# Plot train data
plt.subplot(1, 2, 1)
plt.title("Train")
plt.scatter(X_train, y_train, label="y_train")
plt.plot(X_train, pred_train, color="red", linewidth=1, linestyle="-", label=
plt.legend(loc="lower right")

# Plot test data
plt.subplot(1, 2, 2)
plt.title("Test")
plt.scatter(X_test, y_test, label='y_test')
plt.plot(X_test, pred_test, color="red", linewidth=1, linestyle="-", label="r
plt.legend(loc="lower right")

# Show plots
plt.show()
```

LINEAR REGRESSION : EVALUATION





LINEAR REGRESSION : INTERPRETABILITY

Since we standardized our inputs and outputs, our weights were fit to those standardized values. So we need to unstandardize our weights so we can compare it to our true weight (3.5).

> Note that both X and y were standardized.

$$\hat{y}_{scaled} = b_{scaled} + \sum_{j=1}^{k} W_{scaled\,j} x_{scaled\,j}$$

Variable	Description
y_{scaled}	$\frac{\hat{y}-\bar{y}}{\sigma_y}$
x_{scaled}	$\frac{x_j-\bar{x}_j}{\sigma_j}$

LINEAR REGRESSION : INTERPRETABILITY

$$\frac{\hat{y} - \bar{y}}{\sigma_y} = b_{scaled} + \sum_{j=1}^{k} W_{scaled\,j} \frac{x_j - \bar{x}_j}{\sigma_j}$$

$$\hat{y}_{scaled} = \frac{\hat{y}_{unscaled} - \bar{y}}{\sigma_y} = b_{scaled} + \sum_{j=1}^{k} W_{scaled\,j} \left(\frac{x_j - \bar{x}_j}{\sigma_j}\right)$$

$$\hat{y}_{unscaled} = b_{scaled}\sigma_y + \bar{y} - \sum_{j=1}^{k} W_{scaled\,j} \left(\frac{\sigma_y}{\sigma_j}\right)\bar{x}_j + \sum_{j=1}^{k} W_{scaled\,j} \left(\frac{\sigma_y}{\sigma_j}\right)x_j$$

LINEAR REGRESSION : INTERPRETABILITY

In the expression above, we can see the expression:

$$\hat{y}_{unscaled} = W_{unscaled}x + b_{unscaled}$$

Variable	Description
$W_{unscaled}$	$\sum_{j=1}^{k} W_j(\frac{\sigma_y}{\sigma_j})$
$b_{unscaled}$	$b_{scaled}\sigma_y + \bar{y} - \sum_{j=1}^{k} W_j(\frac{\sigma_y}{\sigma_j})\bar{x}_j$

```python
# Unscaled weights
W_unscaled = W * (y_std/X_std)
b_unscaled = b * y_std + y_mean - np.sum(W_unscaled*X_mean)
print ("[actual] y = 3.5X + noise")
print (f"[model] y_hat = {W_unscaled[0][0]:.1f}X + {b_unscaled[0]:.1f}")
```

```
[actual] y = 3.5X + noise
[model] y_hat = 3.4X + 7.8
```

Shriman TIWARI

Tech Lead/Manager Data Science

Mobile: +33 (0)6 49 71 80 68
shriman.tiwari@keyrus.com

www.keyrus.com

KEYRUS