Machine Learning

CS 539
Worcester Polytechnic Institute
Department of Computer Science
Instructor: Prof. Kyumin Lee

Project Teams

- 4 Yu-Chi Liang, William Ryan, Riley Blair, and Stephen Fanning
- 5 Chris Lee, Andrew Kerekon, Amulya Mohan, Alex Siracusa, and Sulaiman Moukheiber
- 5 Vagmi Bhagavathula, Deepti Gosukonda, Adina Palayoor, Bishoy Soliman Hanna, and Jared Chan
- 4 Sreeram Marimuthu, Oruganty Nitya Phani Santosh, Sarah Olson, and Thomas Pianka
- 4 Shubham Dashrath Wagh, Atharva Pradip Kulkarni, Amit Virchandbhai Prajapati, Niveditha Narasimha Murthy
- 5 Aria Yan, Alisha Peeriz, Nupur Kalkumbe, Pavan Antala, Rutuja Madhumilind Dongre
- 5 Noushin Khosravi Largani, Jinqin Xiong, Kexin Li, Ronit Kapoor, and Yiqun Duan
- 4 Phil Brush, Liam Hall, Jared Morgan, Alex
- 4 Khang Luu, Austin Aguirre, Brock Dubey, Ivan Klevanski,
- 5 Adhiraj, Karl, Shariq Madha, Yue Bao, Vasilli Gorbunov
- 5 Edward Smith, Michael Alicea, Cutter Beck, Blake Bruell, Anushka Bangal
- 4 Rahul Chhatbar, Sonu Tejwani, Deep Suchak, Shoan Bhatambare
- 5 Daniel Fox, Bijesh Shrestha, Chad Hucey, Aayush Sangani, Ivan Lim
- 5 Devesh Bhangale, Shipra Poojary, Jagruti Chitte, Parth Shroff, Saurabh Pande
- 5 Alessandra Serpes, Khushita Joshi, Sanjeeeth Nagappa Chakrasali, Shaun Noronha, Sankalp Vyas
- 2 Rohan Rana, Theo Coppola

So far, 71 students formed teams.

Missing 1 student

Email me names of your team members

Form a team by Jan 25

HW1

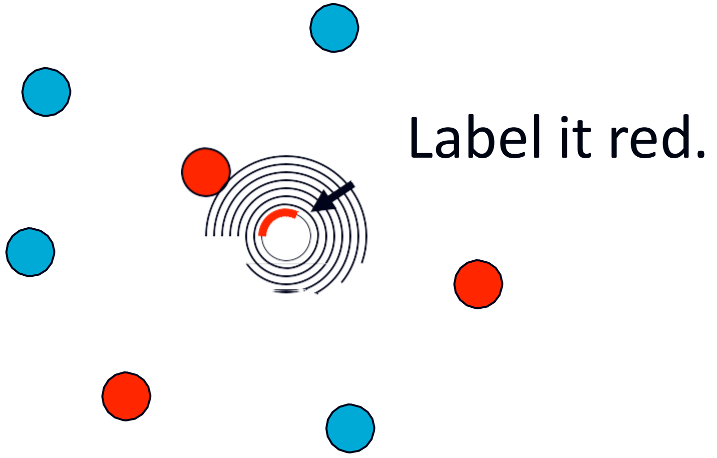
- https://canvas.wpi.edu/courses/57384/assignments/338207

- Due date is January 26th 11:59pm.

k-Nearest Neighbor
& Instance-based Learning

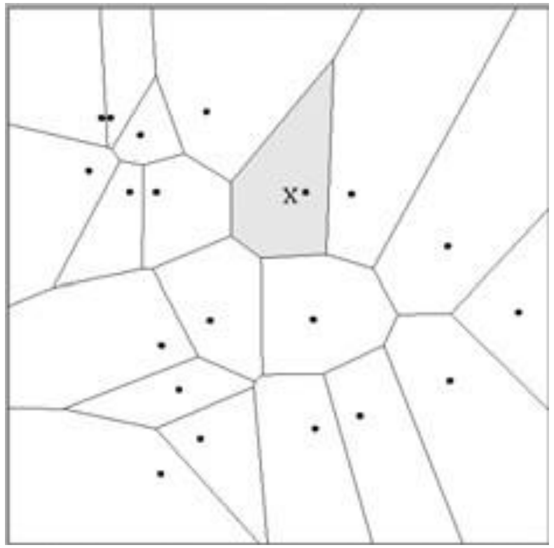
1-Nearest Neighbor

- One of the simplest of all machine learning classifiers
- Simple idea: label a new point the same as the closest known point



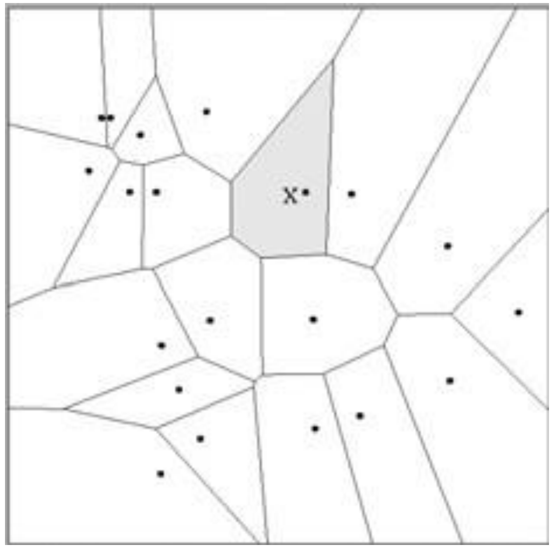
1-Nearest Neighbor

- A type of instance-based learning
  - Also known as “memory-based” learning
- Forms a Voronoi tessellation of the instance space

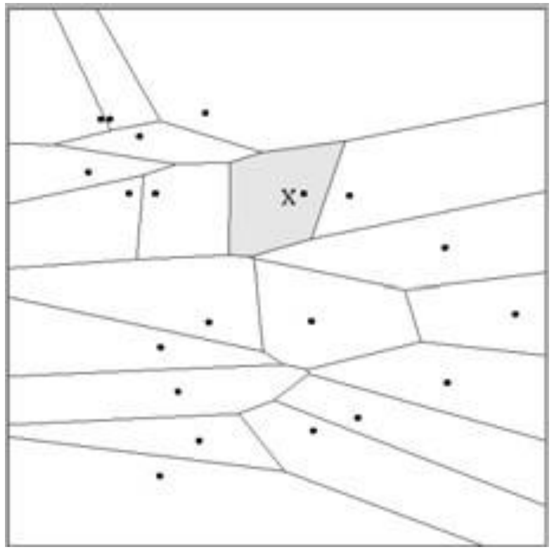


Distance Metrics

- Different metrics can change the decision surface



$\text{Dist}(\mathbf{a},\mathbf{b}) = (a_1 - b_1)^2 + (a_2 - b_2)^2$



$\text{Dist}(\mathbf{a},\mathbf{b}) = (a_1 - b_1)^2 + (3a_2 - 3b_2)^2$

- Standard Euclidean distance metric:
  - Two-dimensional: $\text{Dist}(a,b) = \text{sqrt}((a_1 - b_1)^2 + (a_2 - b_2)^2)$
  - Multivariate: $\text{Dist}(a,b) = \text{sqrt}(\sum (a_i - b_i)^2)$

Four Aspects of an Instance-Based Learner:

1. A distance metric

2. How many nearby neighbors to look at?

3. A weighting function (optional)

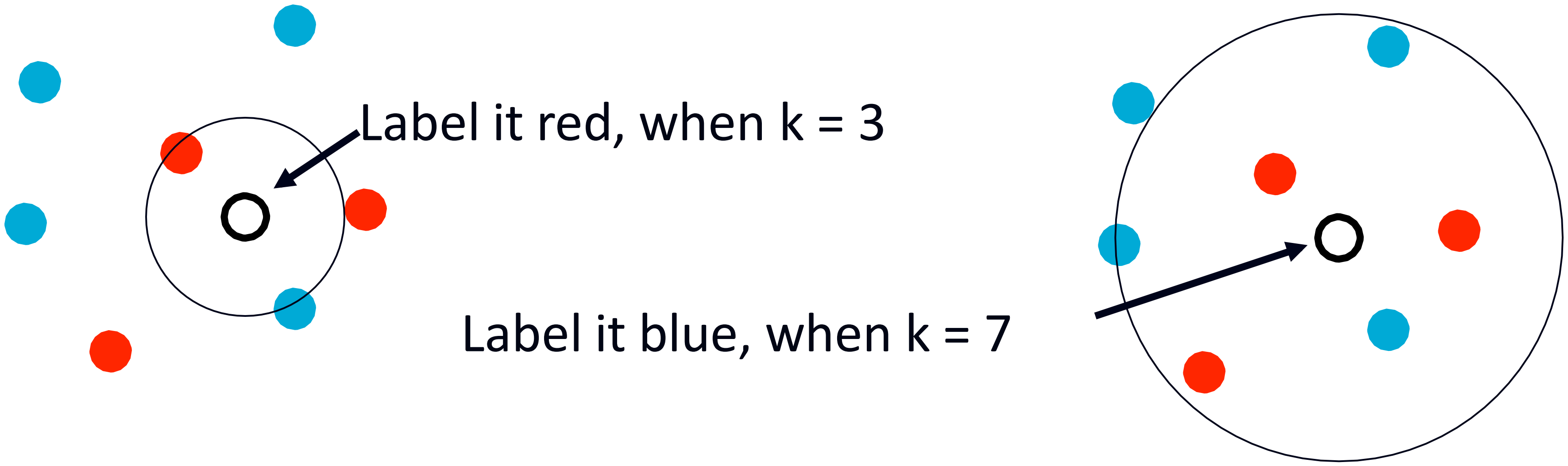
4. How to fit with the local points?

1-NN's Four Aspects as an Instance-Based Learner:

1. A distance metric
   - *Euclidean*
2. How many nearby neighbors to look at?
   - *One*
3. A weighting function (optional)
   - *Unused*
4. How to fit with the local points?
   - *Just predict the same output as the nearest neighbor.*

k – Nearest Neighbor

- Generalizes 1-NN to smooth away noise in the labels
- A new point is now assigned the most frequent label of its *k* nearest neighbors



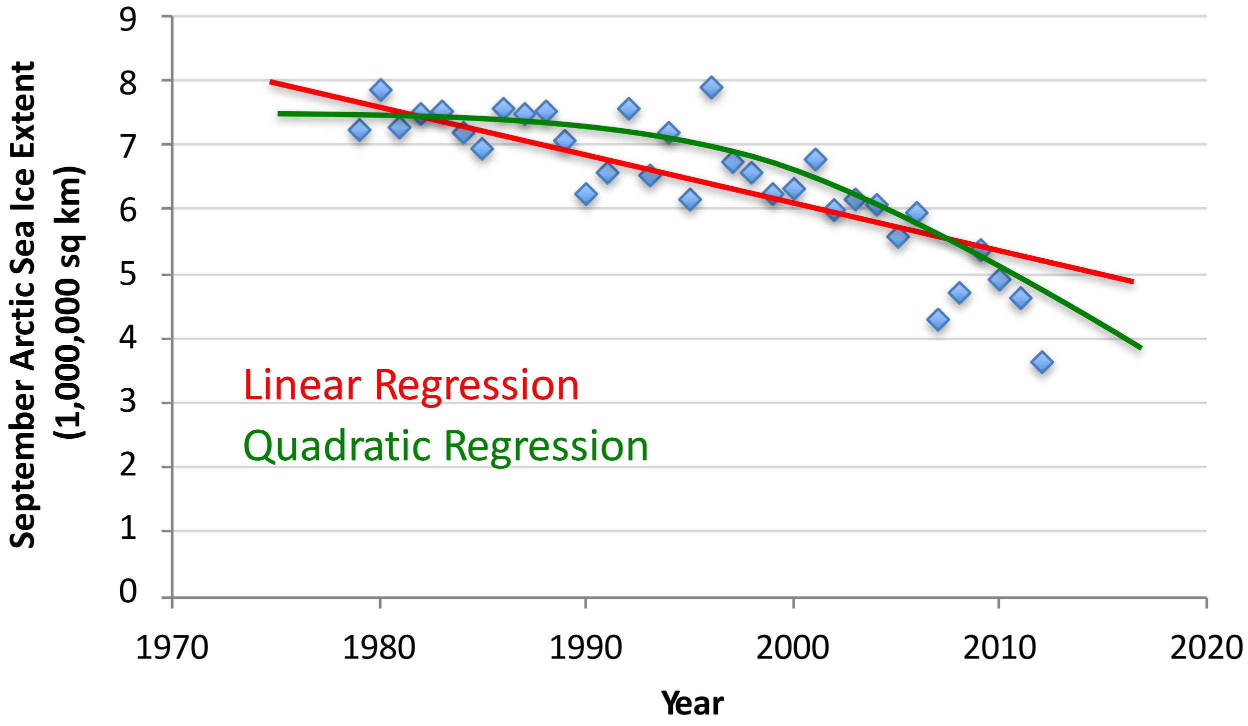
k-NN

- Instance-based learning & lazy learning
- Memorize training data, and measure all each pair of instance in the training set and new instance in the test set
- However, computationally expensive
  - Require N comparison for the prediction
  - Refer to https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/
- To reduce the search time (i.e., reduce O(n))
  - We may use some data structure
  - e.g., k-d tree (k- dimensional tree)
    - https://en.wikipedia.org/wiki/K-d_tree
- k-NN regression
  - k nearest neighbors' average target class value
  - http://www.saedsayad.com/k_nearest_neighbors_reg.htm
  - http://scikit-learn.org/stable/modules/neighbors.html

Linear Regression

Regression

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



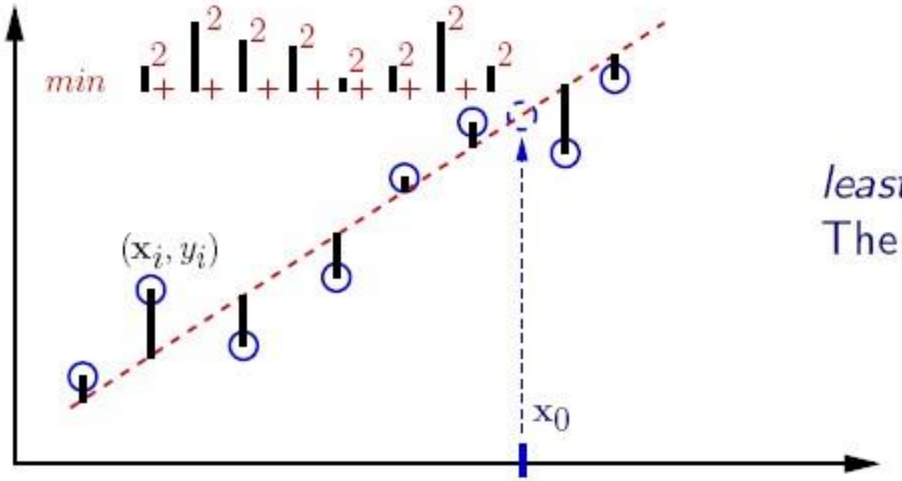
Linear Regression

- Hypothesis:

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

Assume $x_0 = 1$

- Fit model by minimizing sum of squared errors



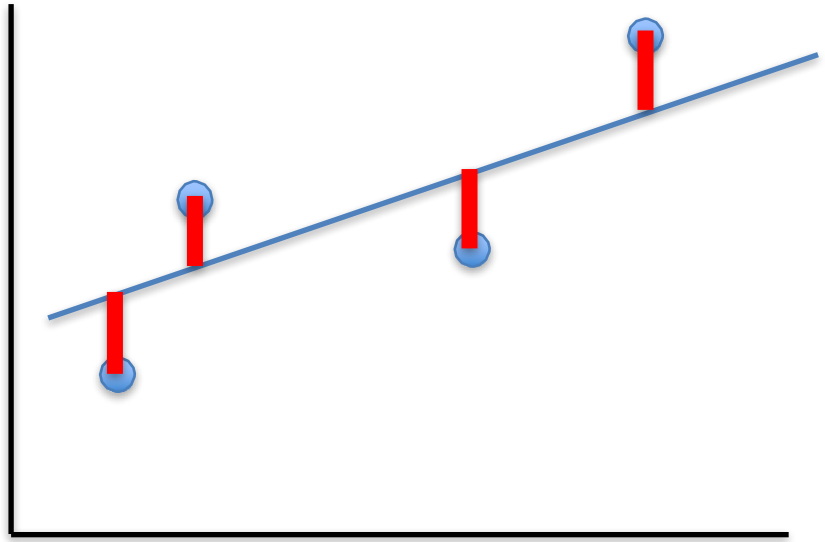
least squares (LSQ)
The fitted line is used as a predictor

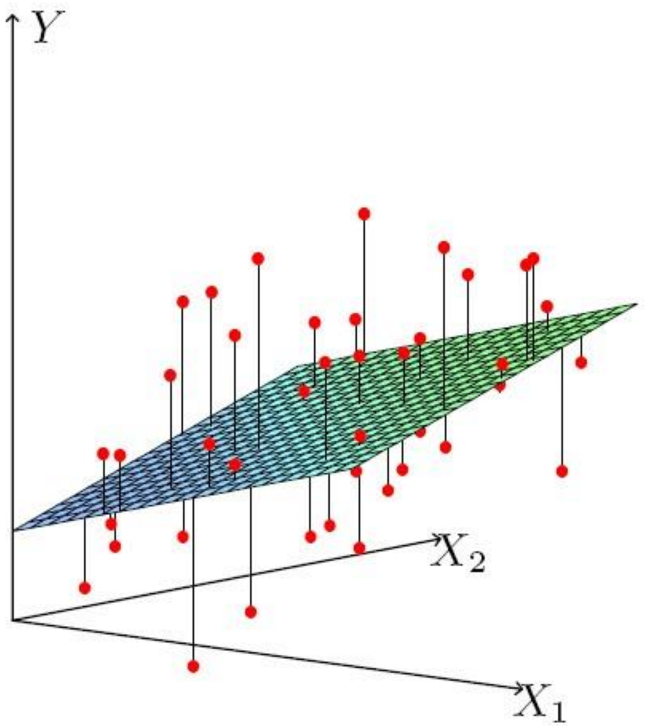
Least Squares Linear Regression

- Cost Function

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

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





Intuition Behind Cost Function

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

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

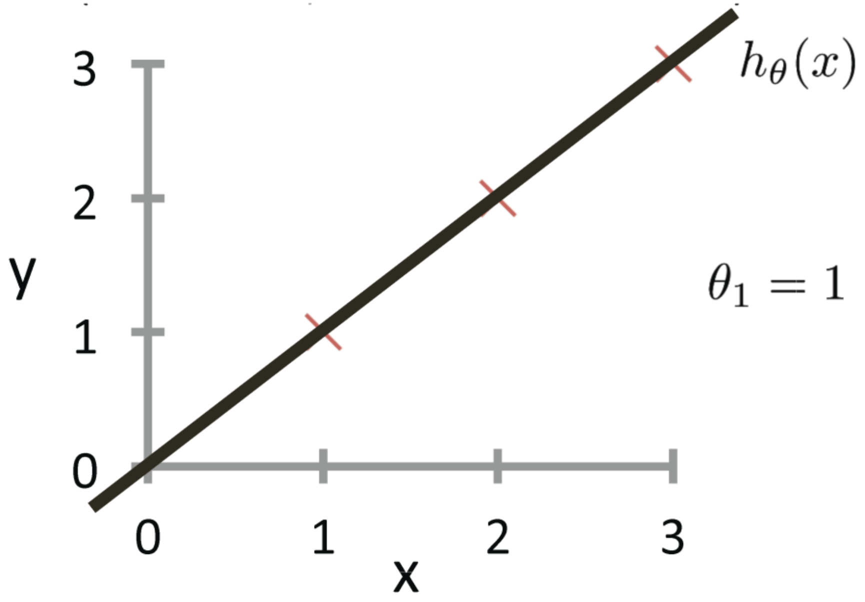
Intuition Behind Cost Function

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

For insight on J(), let's assume $x \in \mathbb{R}$ so $\theta = [\theta_0, \theta_1]$ ➔ $\theta_0 = 0$

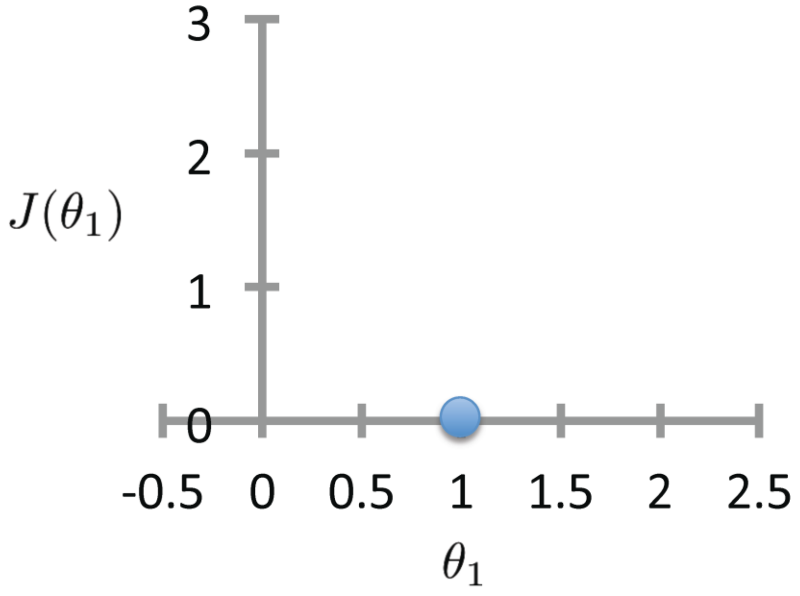
$h_\theta(x)$

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



$J(\theta)$

(function of the parameter θ_1)



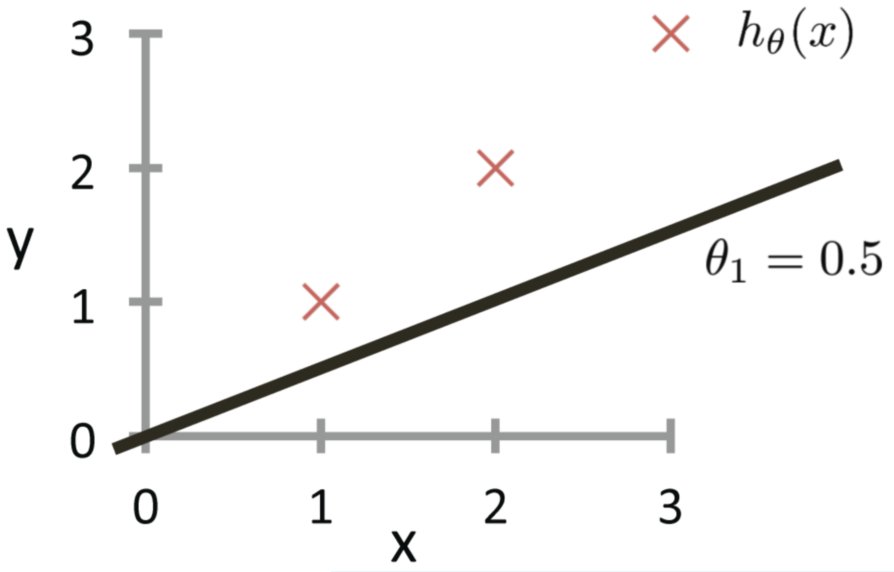
Intuition Behind Cost Function

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

For insight on J(), let's assume $x \in \mathbb{R}$ so $\theta = [\theta_0, \theta_1]$ ➔ $\theta_0 = 0$

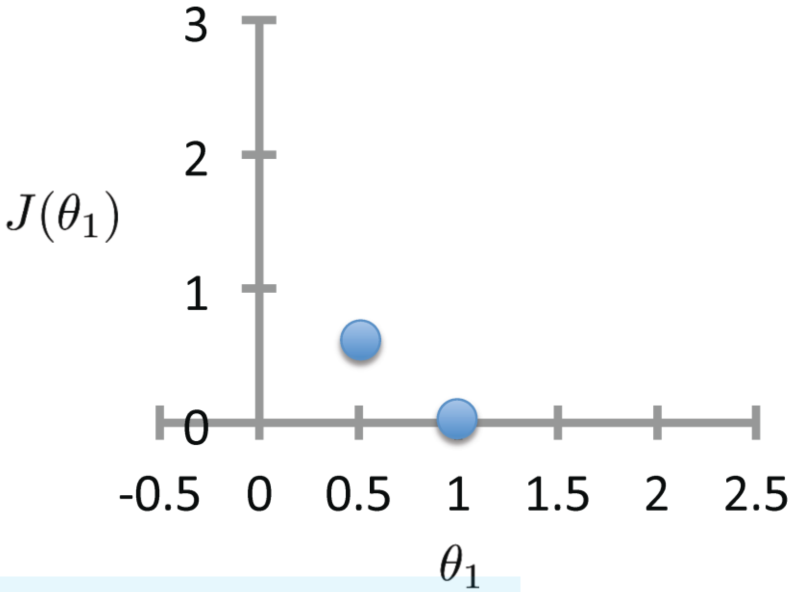
$h_\theta(x)$

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



$J(\theta)$

(function of the parameter θ_1)



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

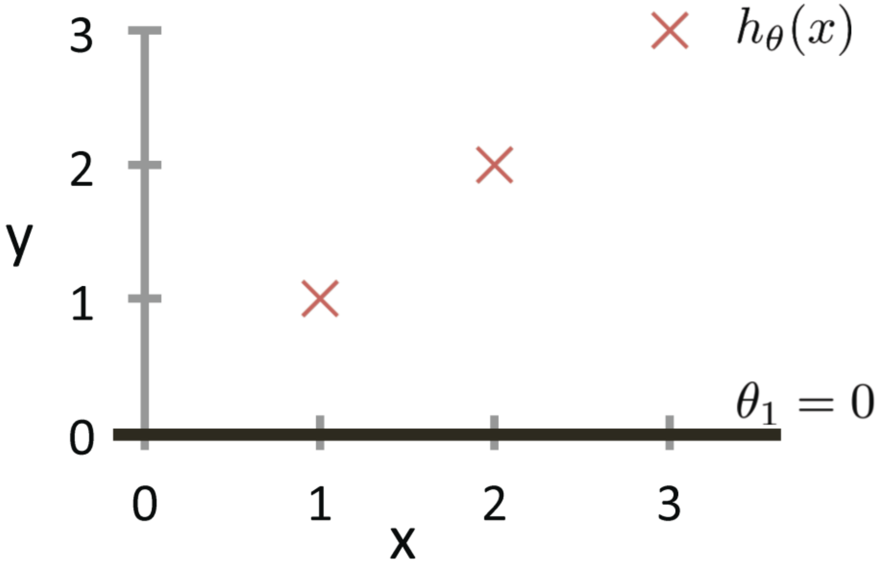
Intuition Behind Cost Function

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

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

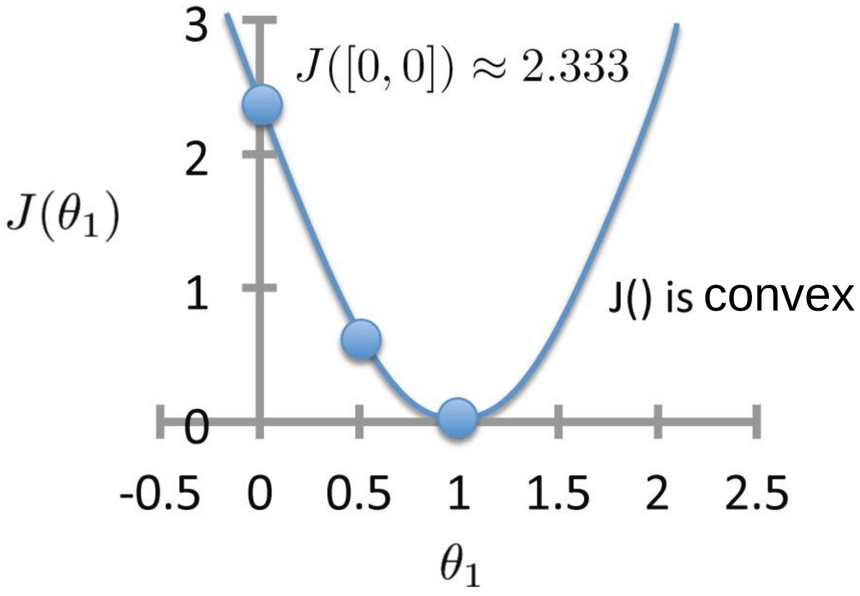
$h_\theta(x)$

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



$J(\theta)$ ➔ $\theta_0 = 0$

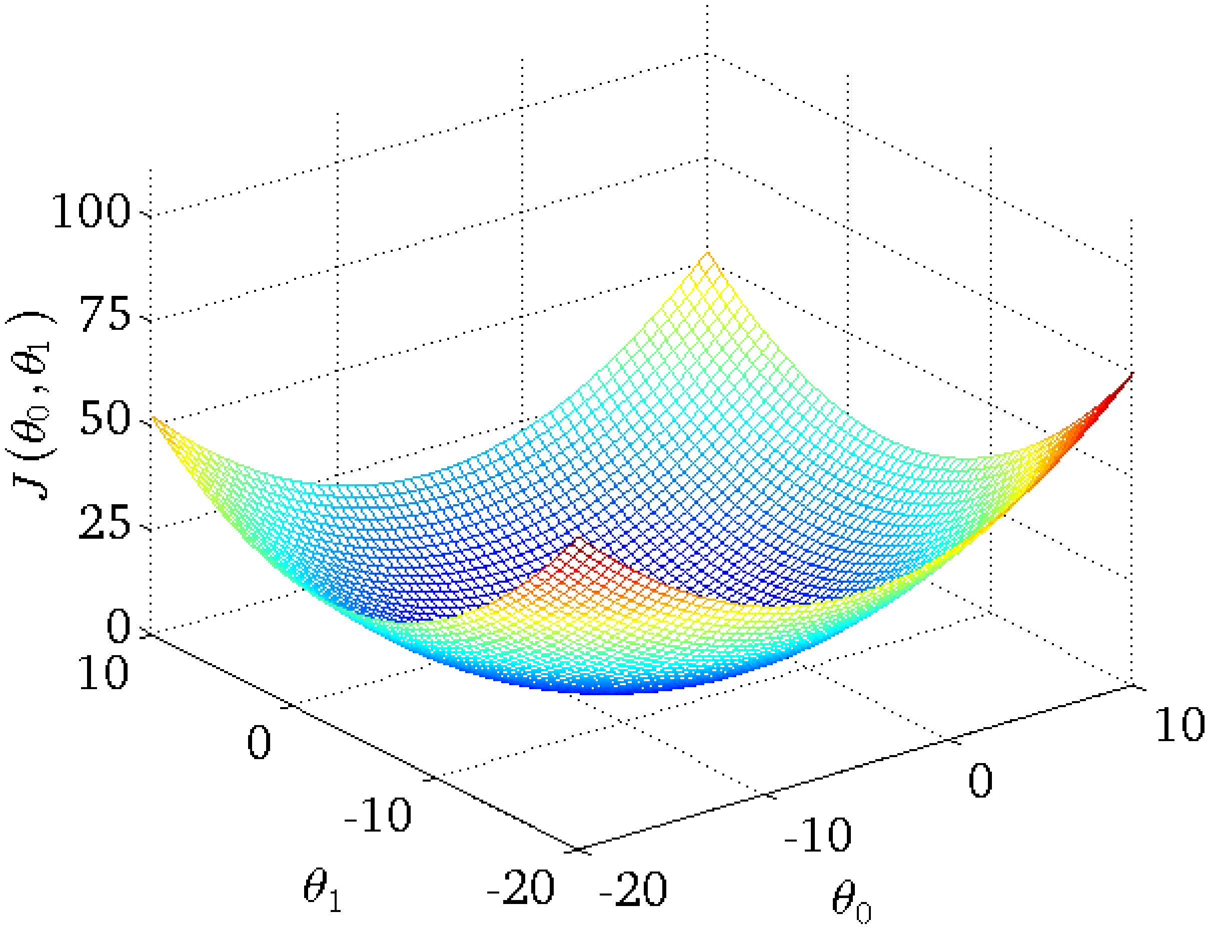
(function of the parameter θ_1)



http://mathworld.wolfram.com/ConvexFunction.html

https://www.desmos.com/calculator/kreo2ssqj8

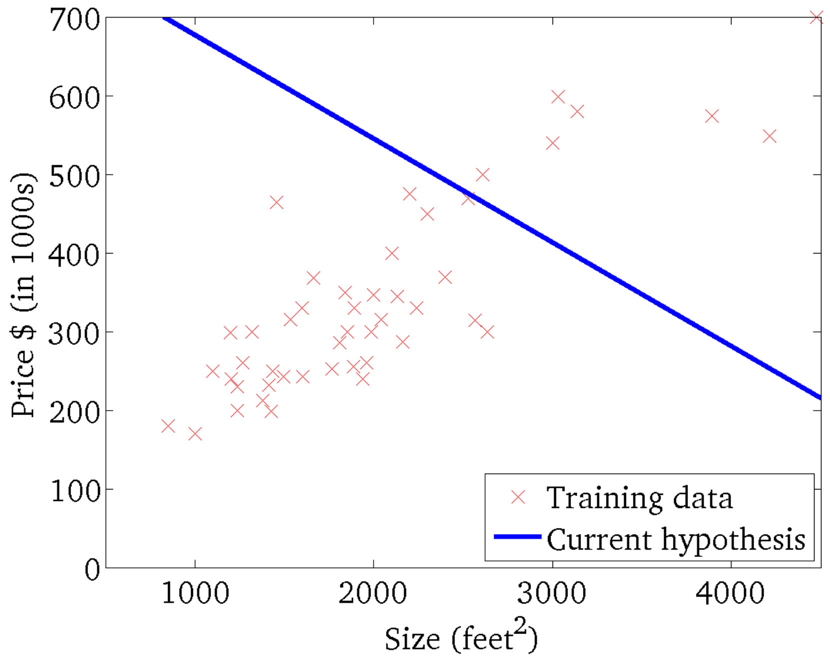
Intuition Behind Cost Function (3-D surface plot)



Intuition Behind Cost Function

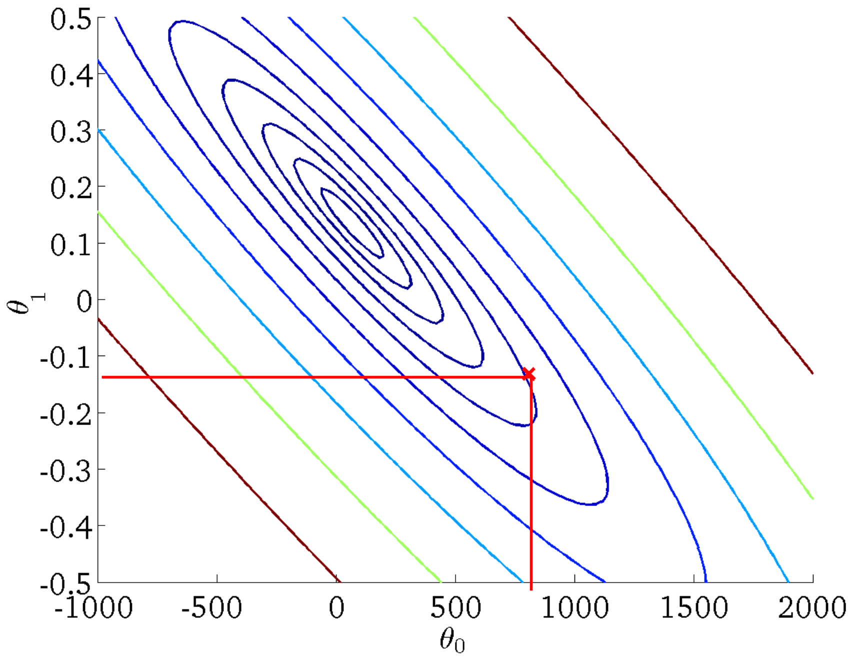
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

(function of the parameter θ_0, θ_1)

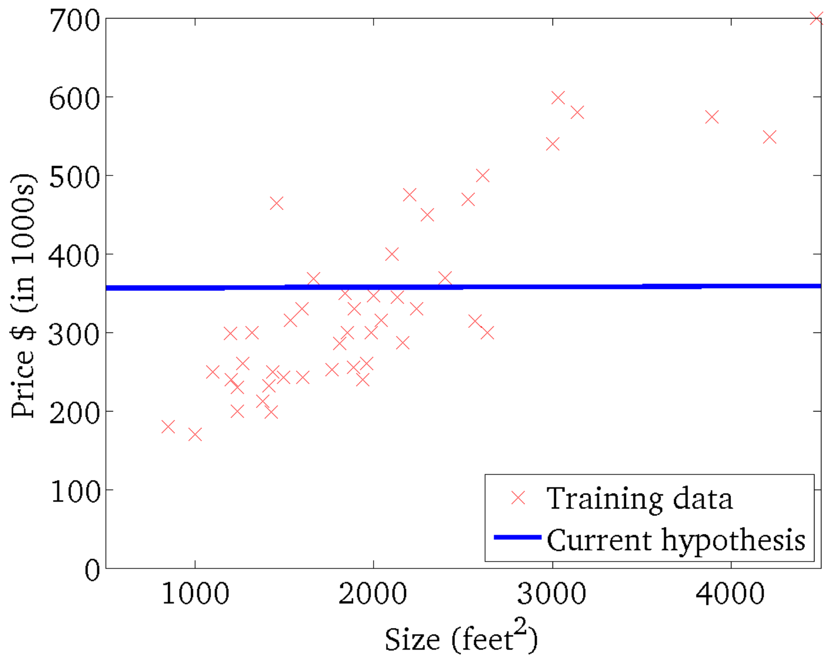


Contour Figure

Intuition Behind Cost Function

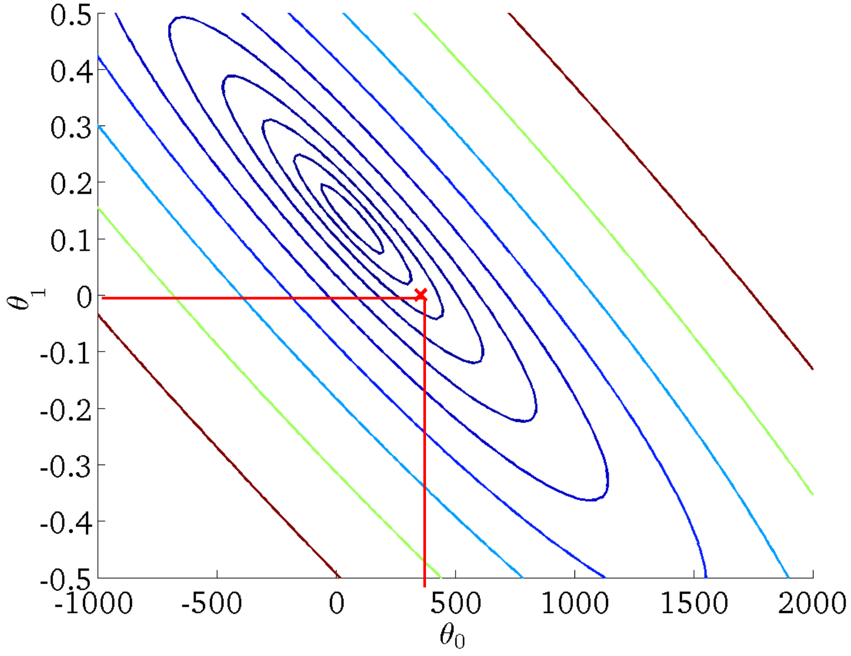
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

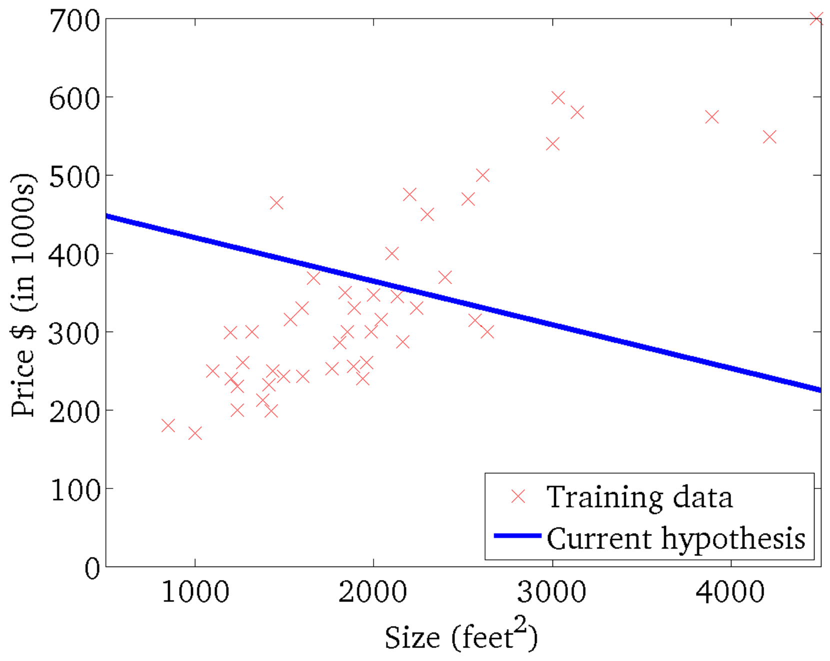
(function of the parameter θ_0, θ_1)



Intuition Behind Cost Function

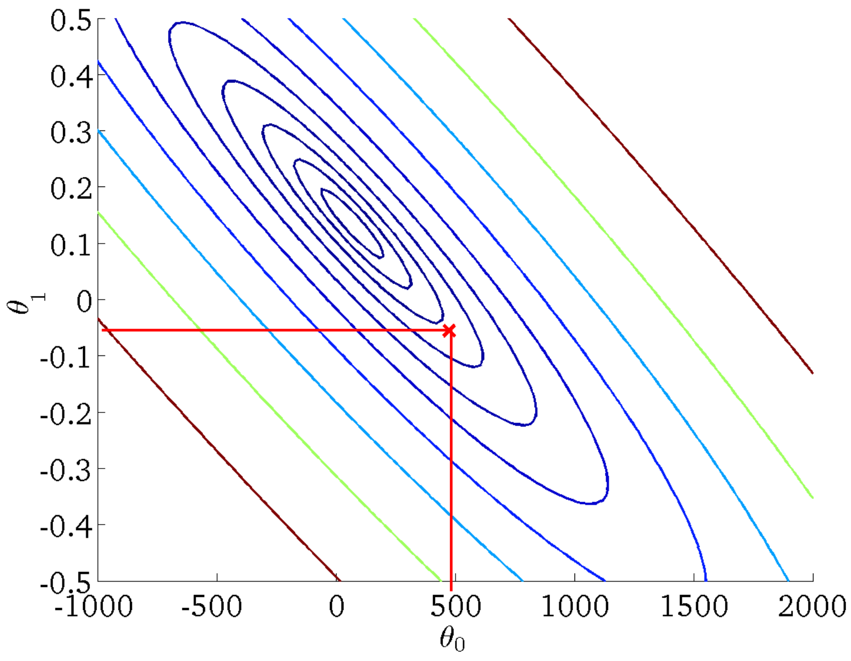
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

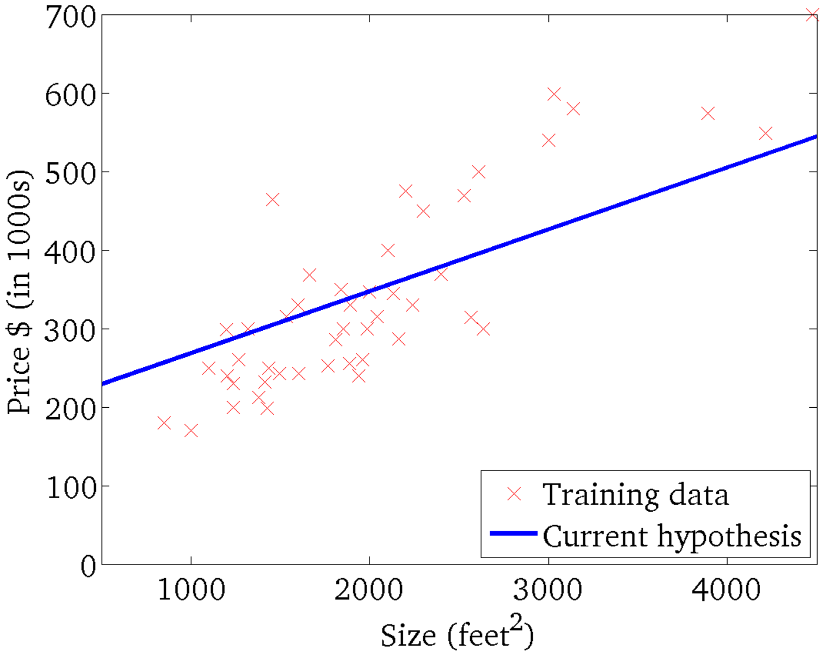
(function of the parameter θ_0, θ_1)



Intuition Behind Cost Function

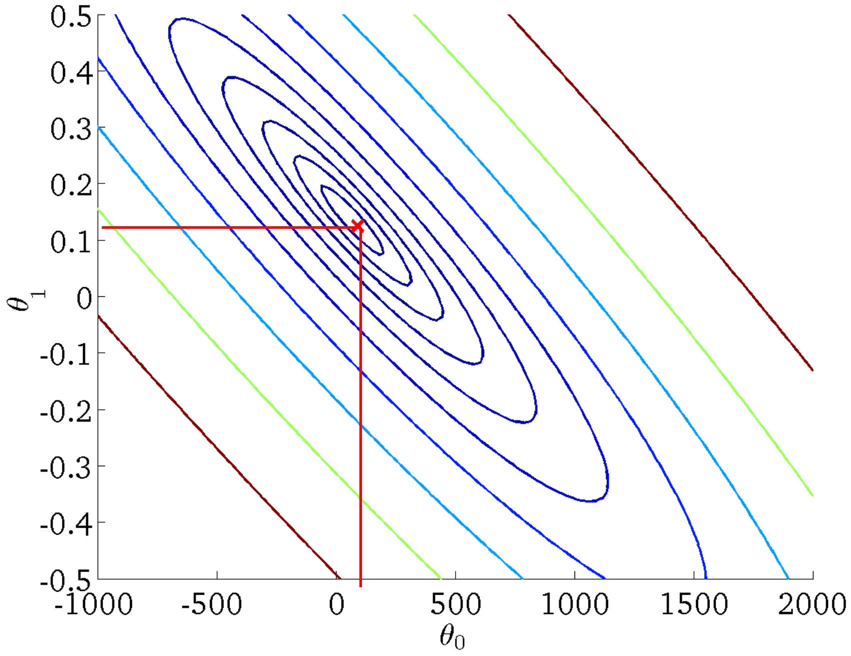
$h_\theta(x)$

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



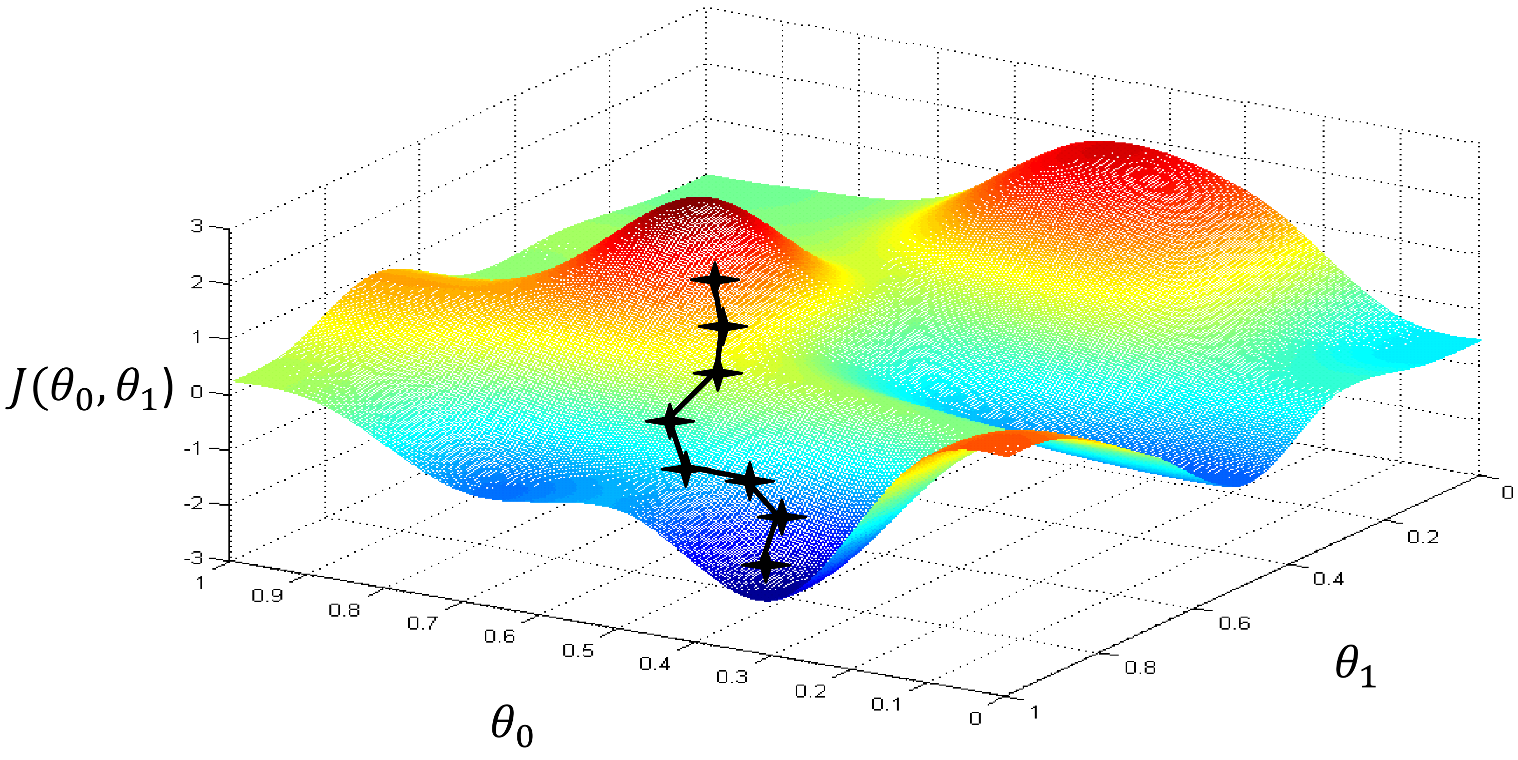
$J(\theta_0, \theta_1)$

(function of the parameter θ_0, θ_1)



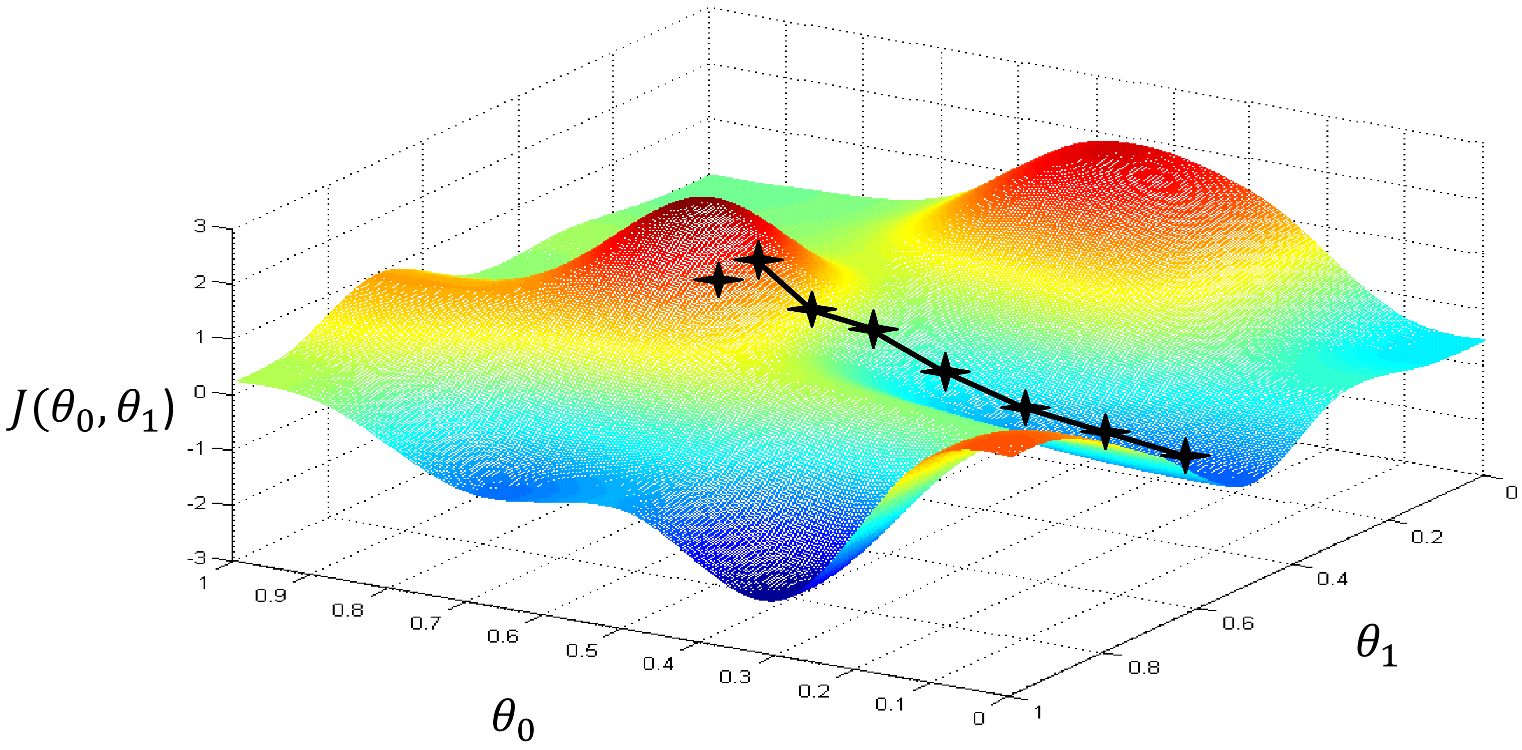
Basic Search Procedure

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



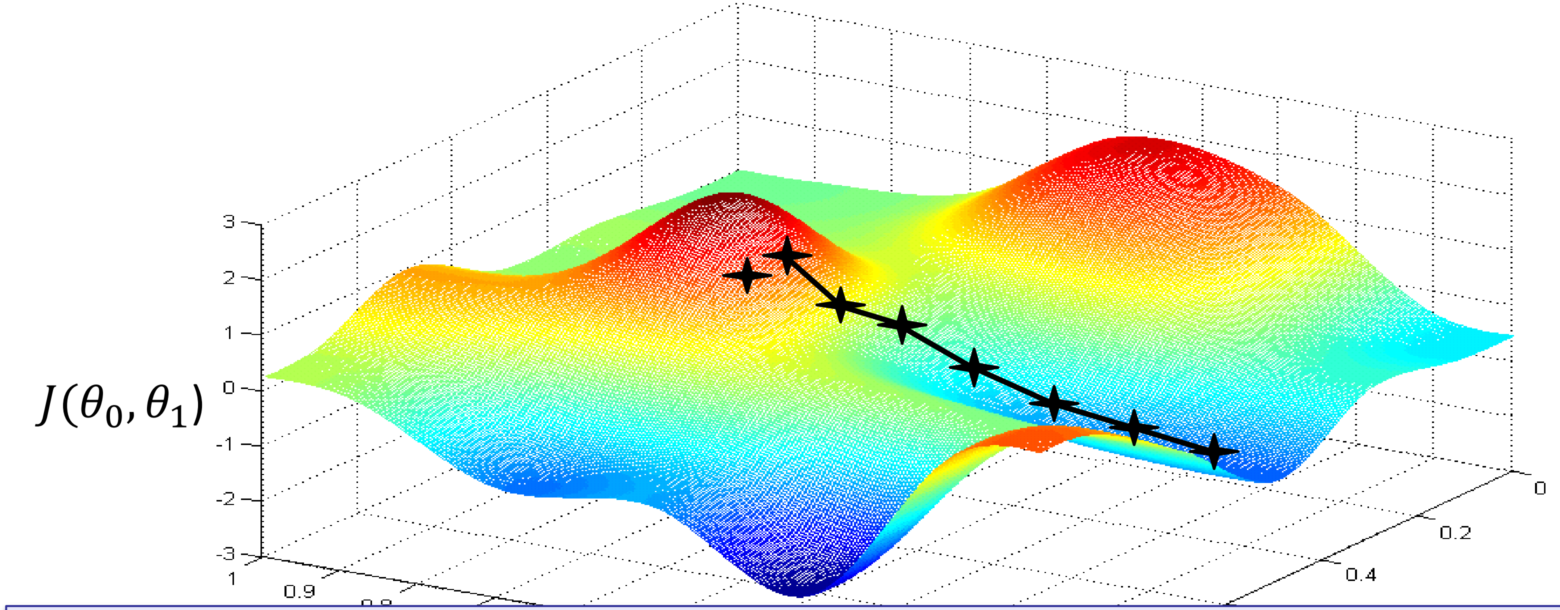
Basic Search Procedure

- 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, we don't need to worry about local minima

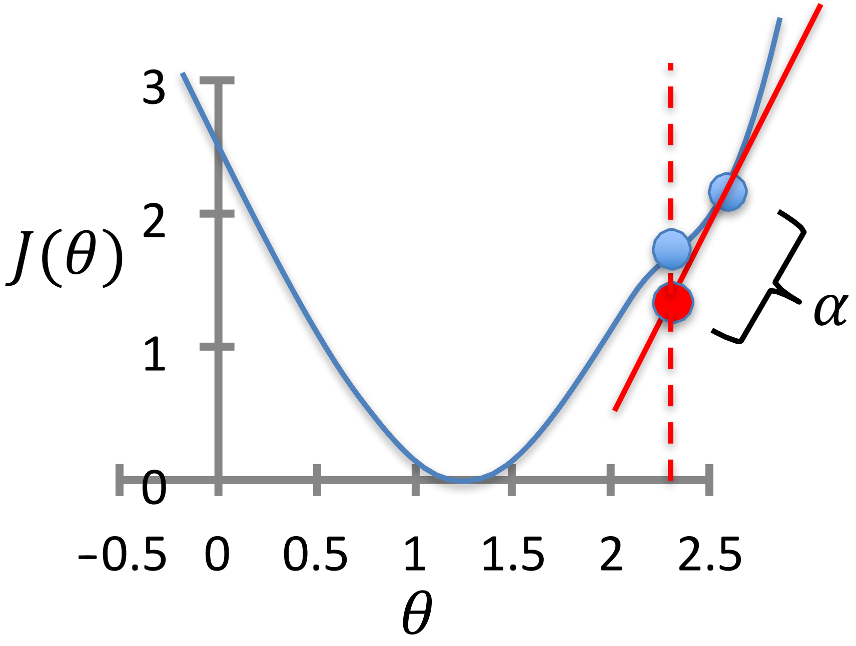
Gradient Descent

- Initialize θ
- Repeat until convergence

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

simultaneous update
for j = 0 … d

learning rate (small)
e.g., α = 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 … d

For Linear Regression: $\frac{\partial}{\partial \theta_j} J(\theta) = \frac{\partial}{\partial \theta_j} \frac{1}{2n} \sum_{i=1}^{n} (h_\theta(x^{(i)}) - y^{(i)})^2$

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

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

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

http://mccormickml.com/2014/03/04/gradient-descent-derivation/

Gradient Descent for Linear Regression

- Initialize θ
- Repeat until convergence

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

simultaneous update
for j = 0 … d

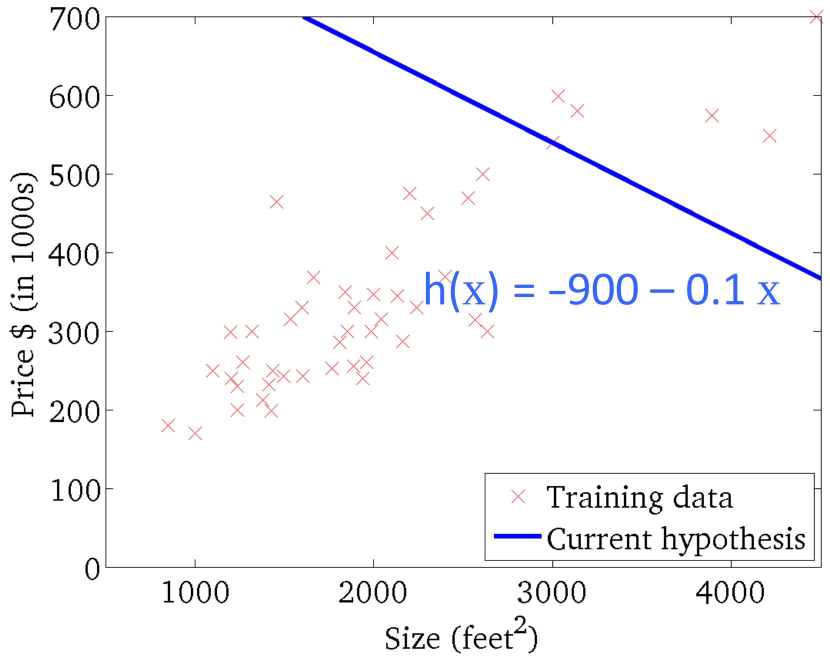
- To achieve simultaneous update
  - At the start of each GD iteration, compute $h_\theta\left(x^{(i)}\right)$
  - Use this stored value in the update step loop
- Assume convergence when $\left\|\theta_{new} - \theta_{old}\right\|_2 < \epsilon$

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

Gradient Descent

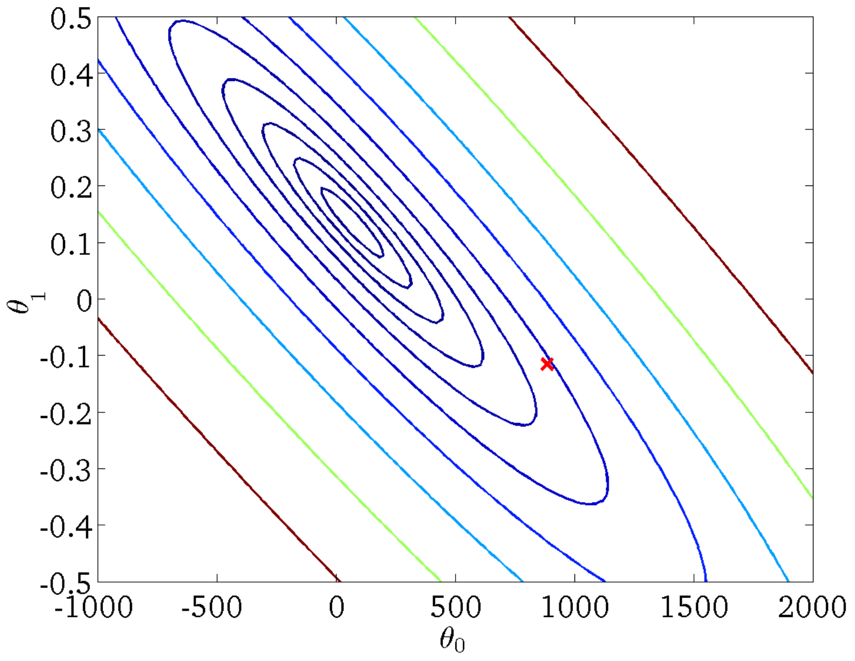
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

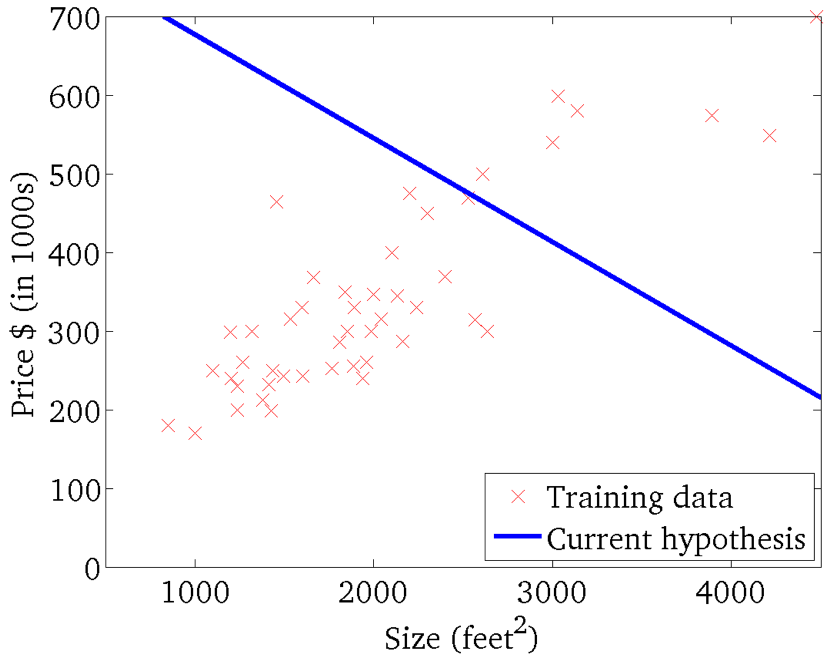
(function of the parameters θ_0, θ_1)



Gradient Descent

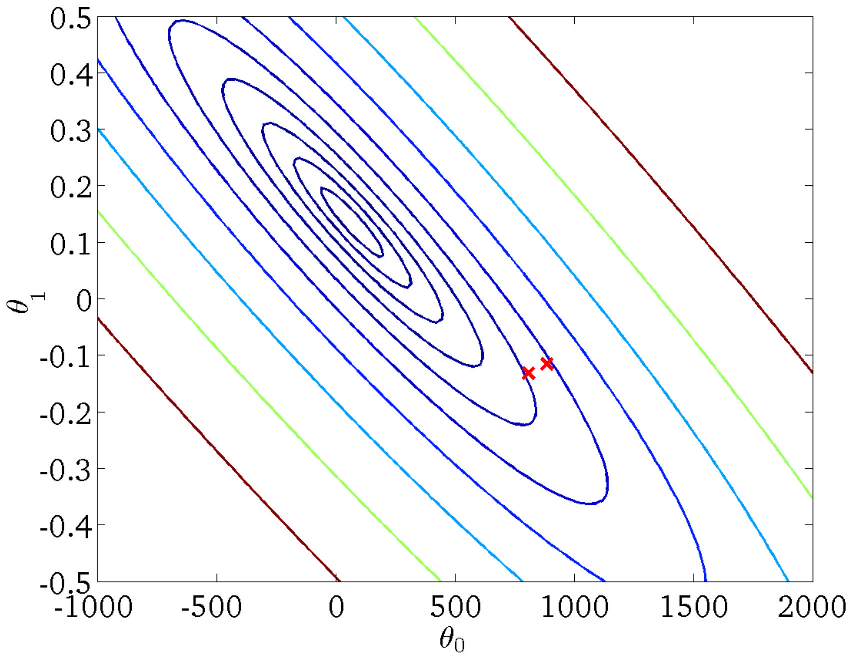
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

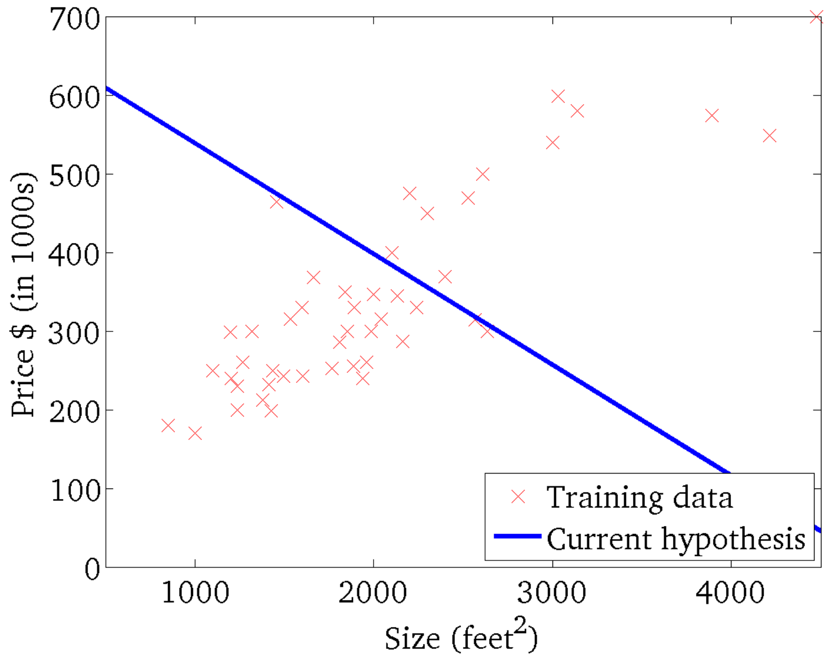
(function of the parameters θ_0, θ_1)



Gradient Descent

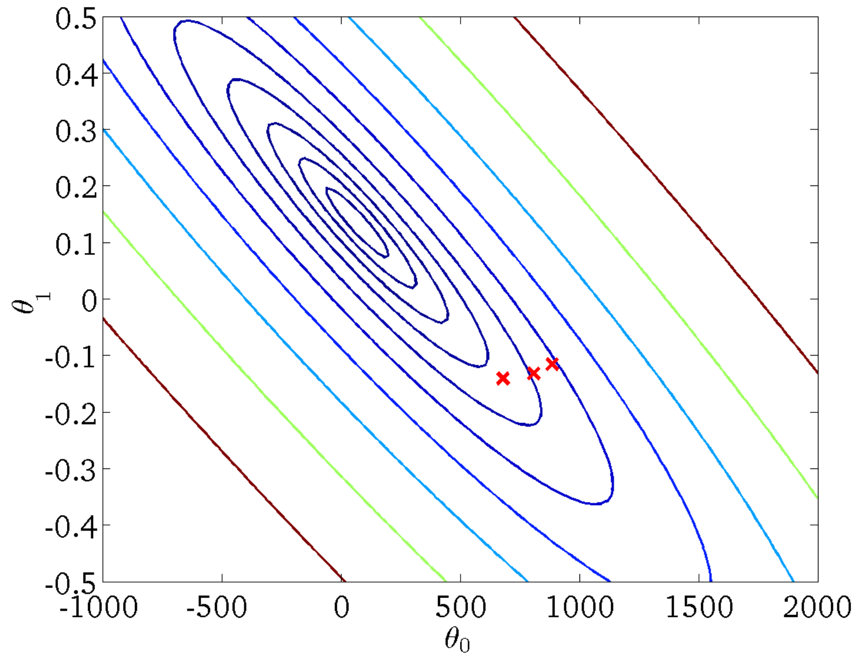
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

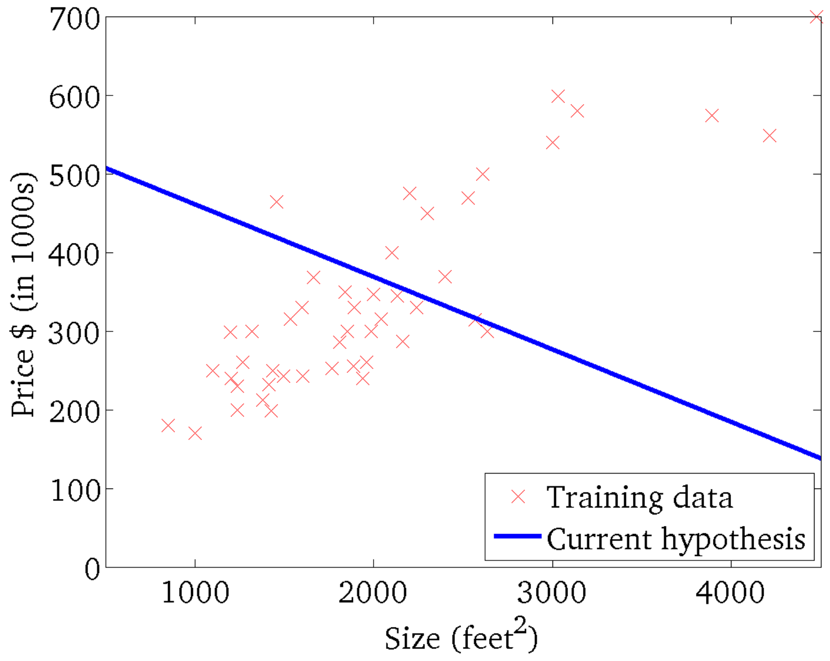
(function of the parameters θ_0, θ_1)



Gradient Descent

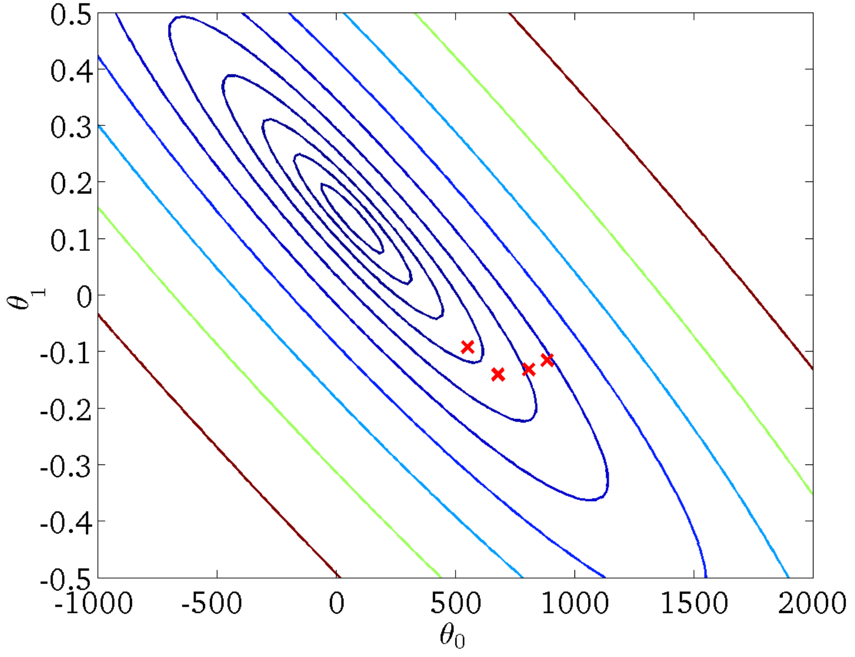
$$h_\theta(x)$$

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



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

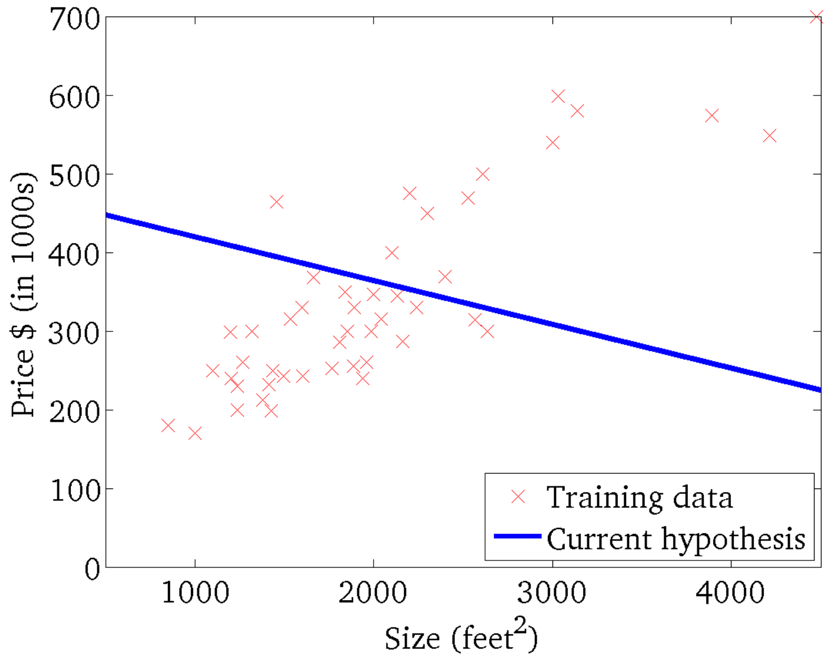
(function of the parameters θ_0, θ_1)



Gradient Descent

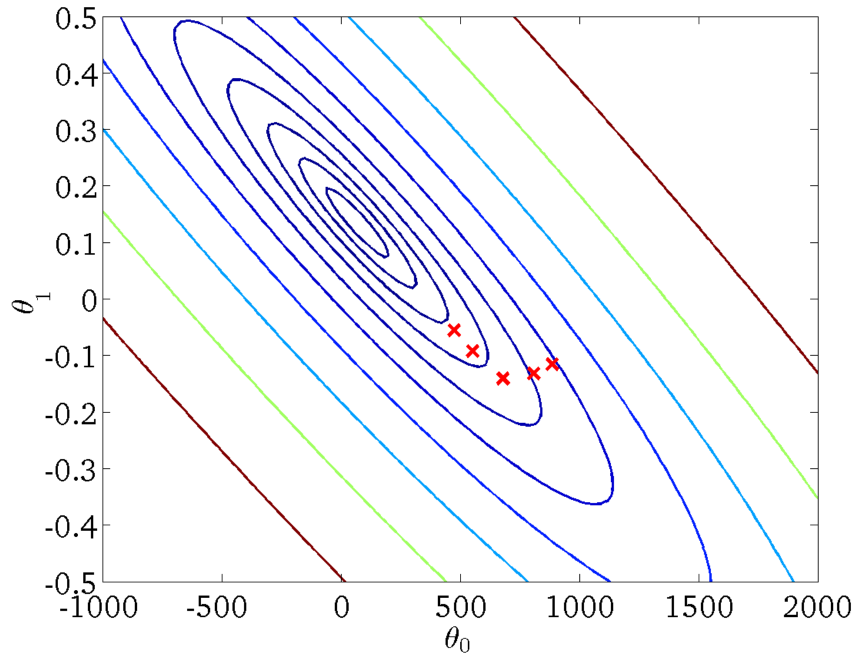
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

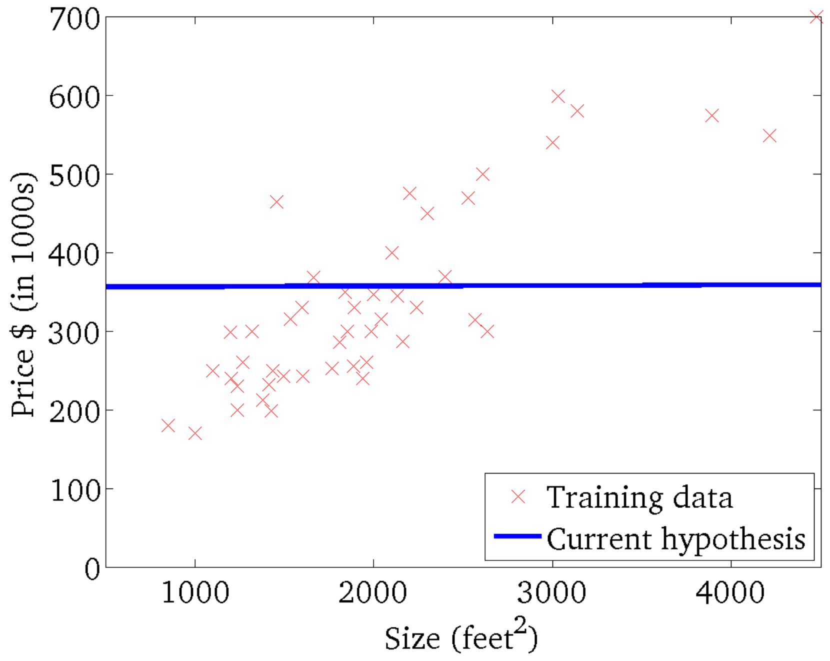
(function of the parameters θ_0, θ_1)



Gradient Descent

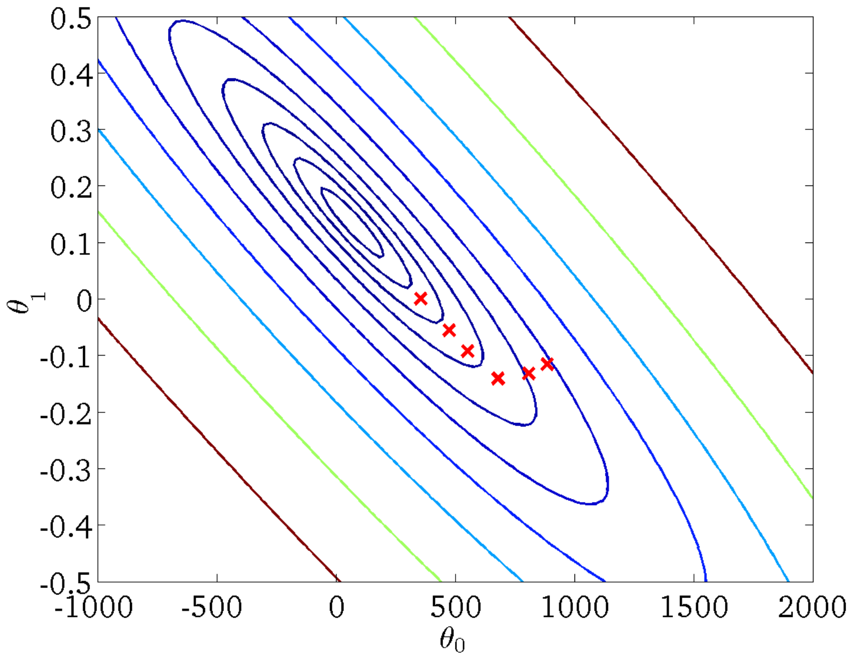
$$h_\theta(x)$$

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



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

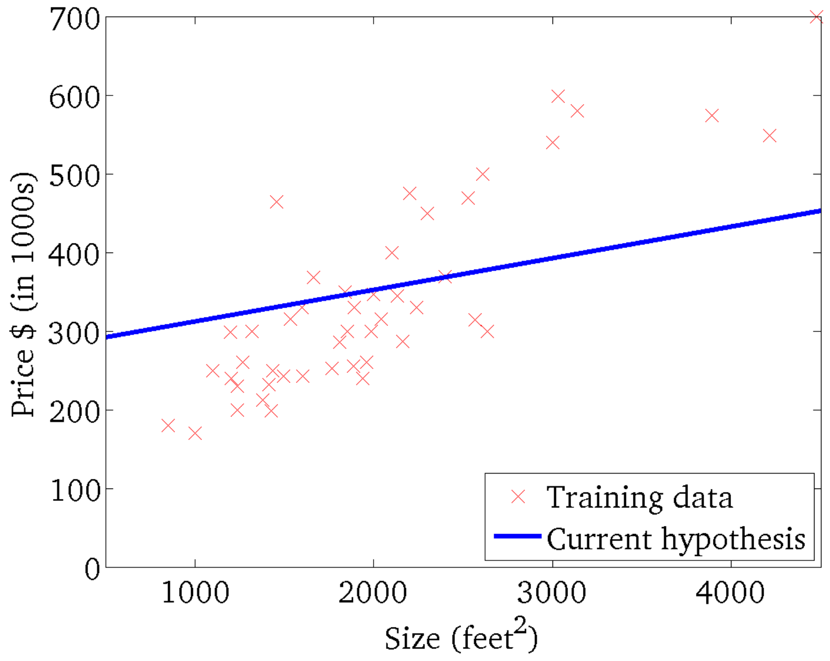
(function of the parameters θ_0, θ_1)



Gradient Descent

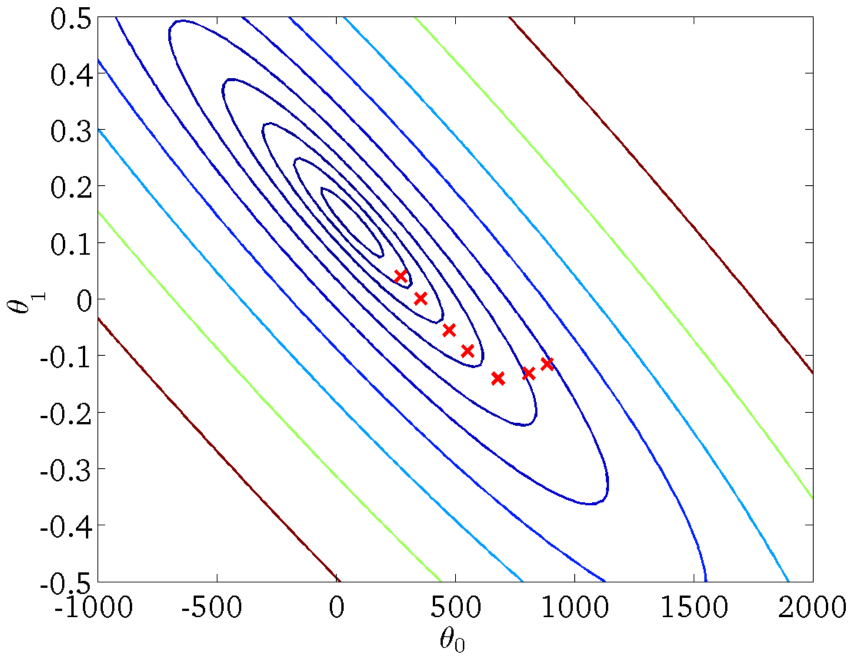
$h_\theta(x)$

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



$J(\theta_0, \theta_1)$

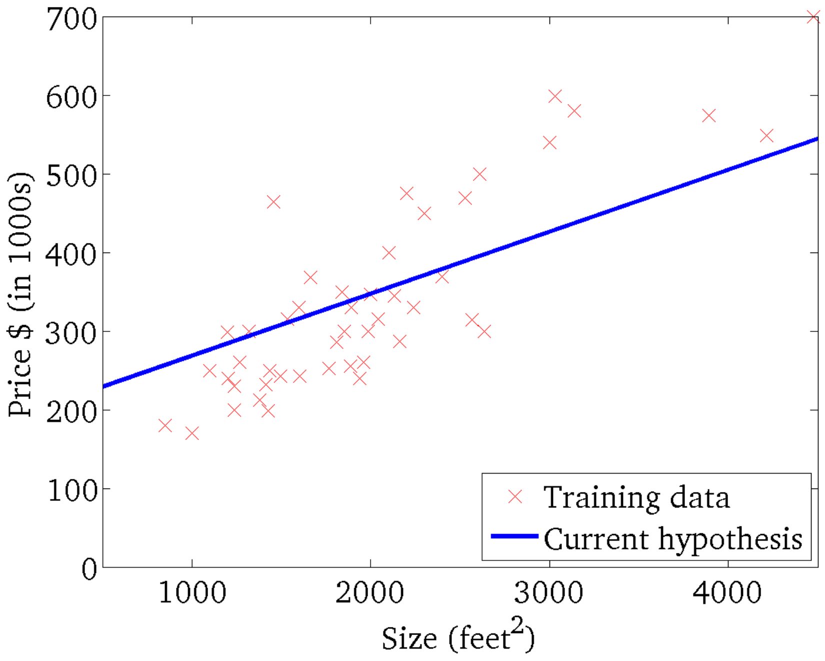
(function of the parameters θ_0, θ_1)



Gradient Descent

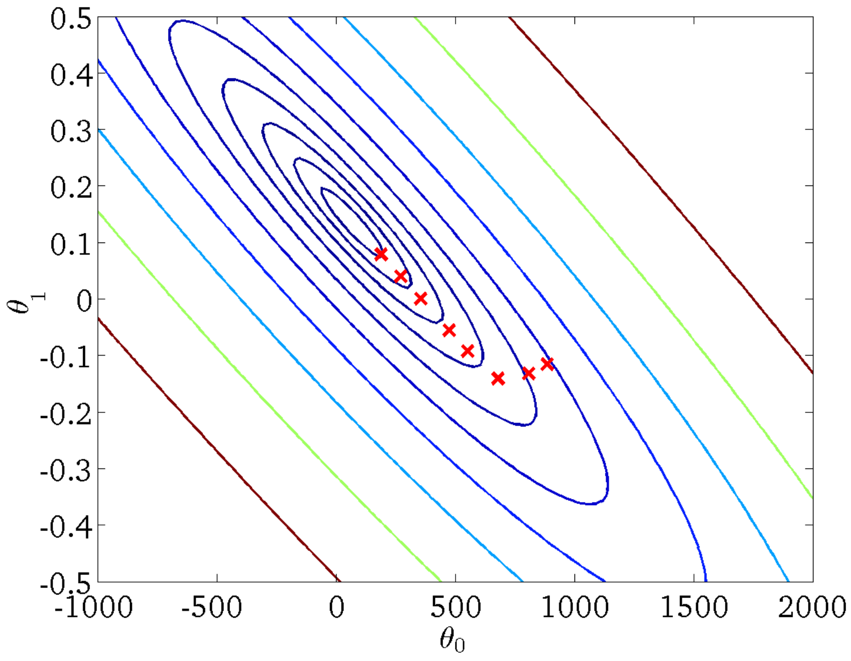
$$h_\theta(x)$$

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



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

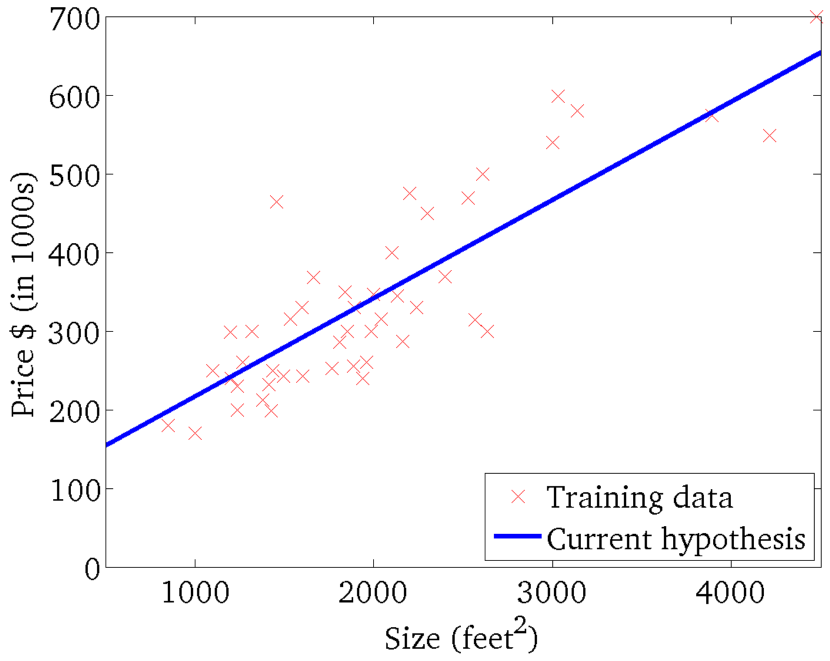
(function of the parameters θ_0, θ_1)



Gradient Descent

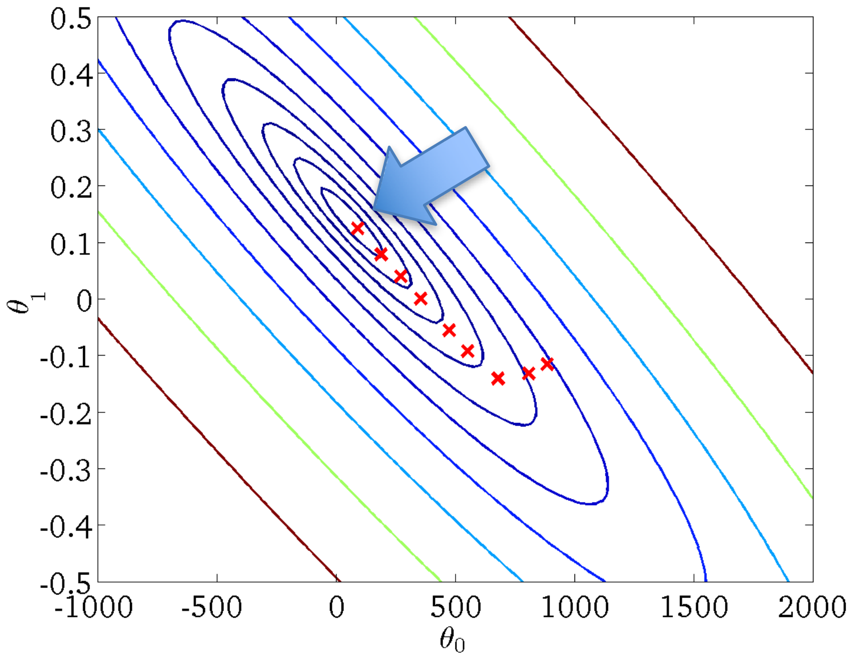
$h_\theta(x)$

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



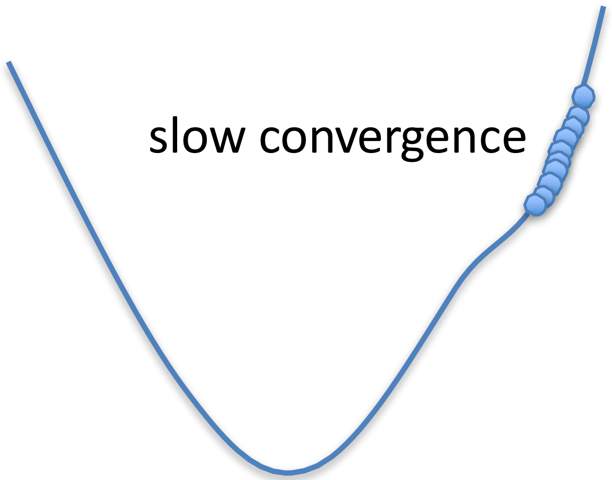
$J(\theta_0, \theta_1)$

(function of the parameters θ_0, θ_1)

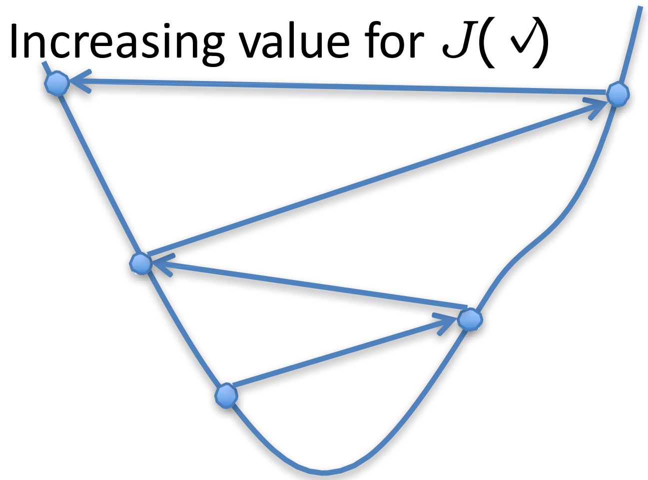


Choosing α

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:
  - Transformation of quantitative inputs
    - e.g. log, exp, square root, square, etc.
  - Interactions between variables
    - example: $x_3 = x_1 \times x_2$
  - Polynomial transformation
    - example: $y = \beta_0 + \beta_1 x + \beta_2 x^2 + \beta_3 x^3$
  - Basis expansions
    - Take the input x, and add “expand” it to make whatever method you are using more powerful.

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

Linear Basis Function Models

- Generally,

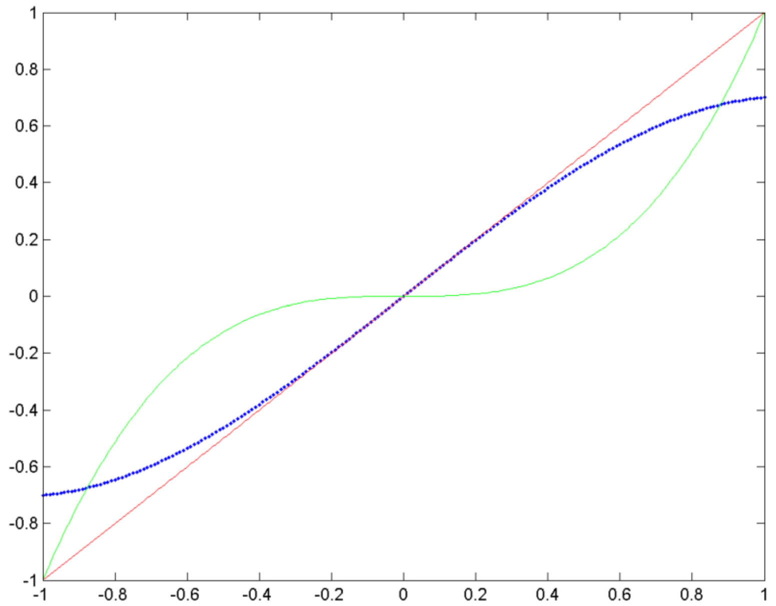
$$h_\theta(x) = \sum_{j=0}^{d} \theta_j \underbrace{\phi_j(x)}_{\text{basis function}}$$

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

$$\phi_j(x) = x_j$$

Example of Basis Function Models

$$f(x) = x - 0.3x^3$$



Basis functions $\phi_1(x) = 1, \phi_2(x) = x, \phi_3(x) = x^2, \phi_4(x) = x^3$ und $\boldsymbol{\theta} = (0, 1, 0, -0.3)$

https://www.desmos.com/calculator/kreo2ssqj8

Basic Idea of Basis Function Models

- The simple idea: in addition to the original inputs, we add inputs that are calculated as deterministic functions of the existing inputs and treat them as additional inputs
- Example: Polynomial Basis Functions

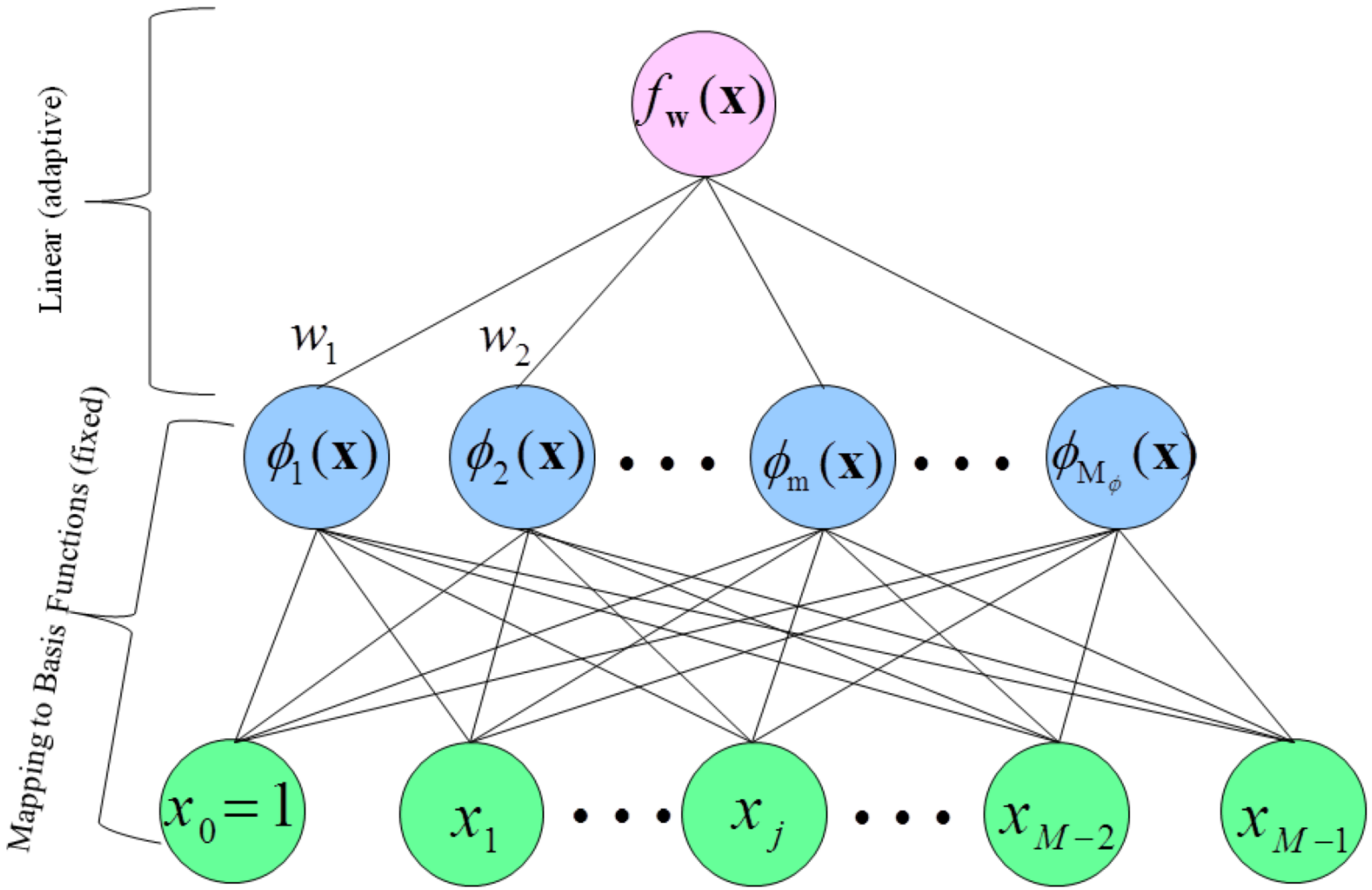
$$\{1, x_1, x_2, x_3, x_1x_2, x_1x_3, x_2x_3, x_1^2, x_2^2, x_3^2\}$$

- Basis functions $\{\phi_m(\mathbf{x})\}_{m=1}^{M_\phi}$
- In the example:

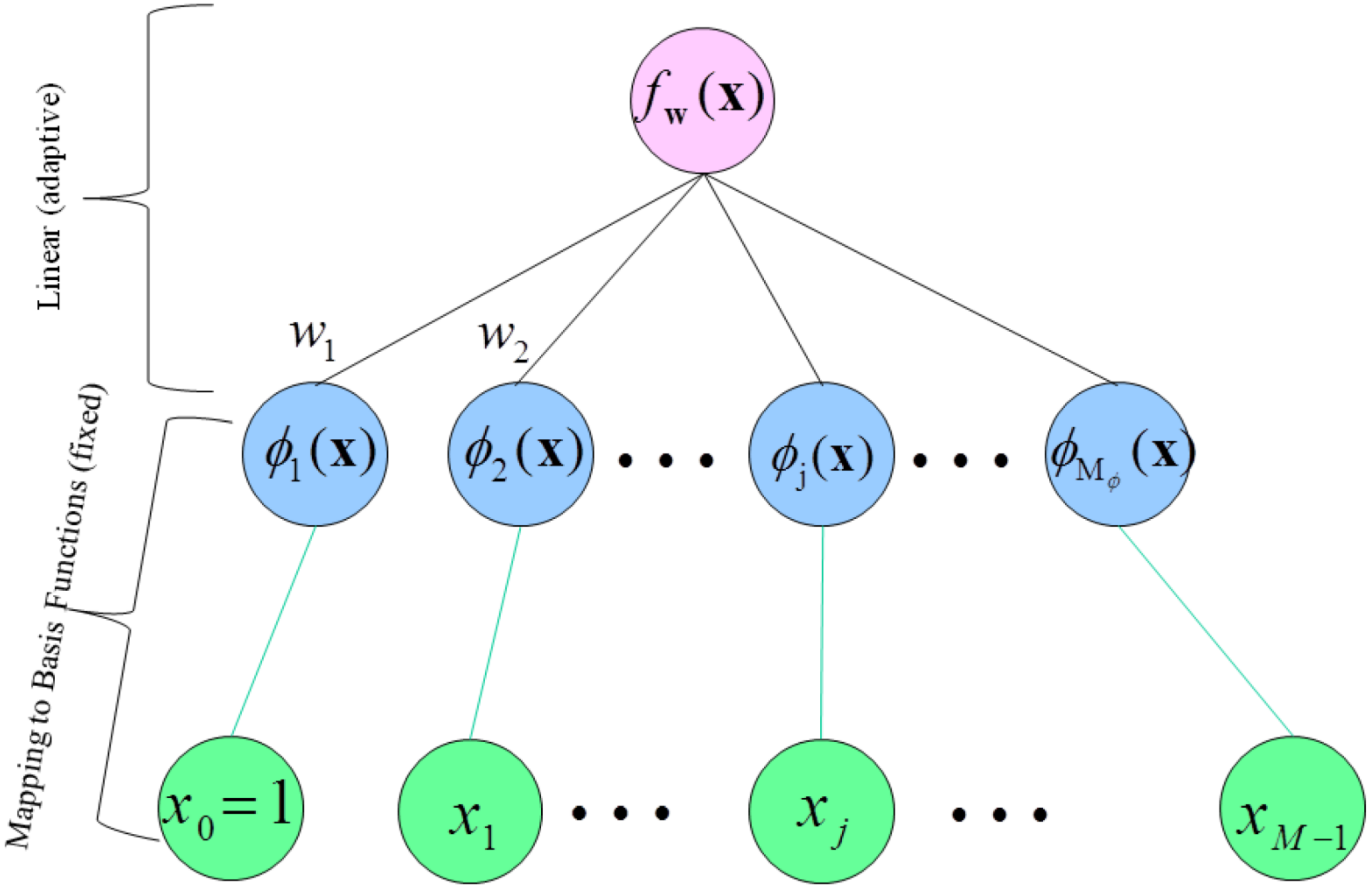
$$\phi_1(\mathbf{x}) = 1 \quad \phi_2(\mathbf{x}) = x_1 \quad \phi_6(\mathbf{x}) = x_1x_3 \quad \ldots$$

- Independent of the choice of basis functions, the regression parameters are calculated using the well-known equations for linear regression

Network of Basis Functions



Network of Linear Basis Functions

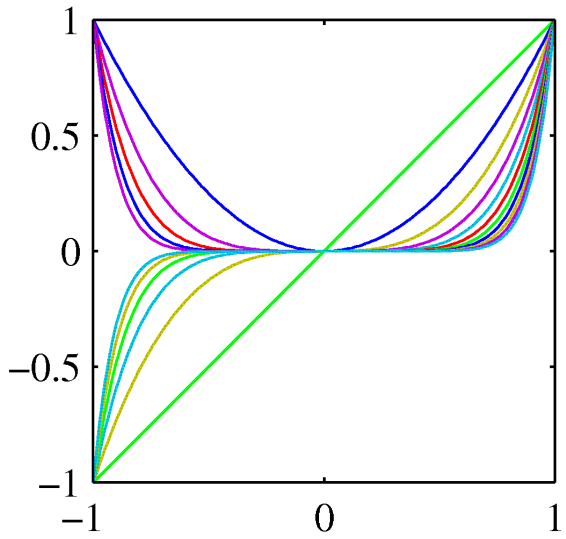


Linear Basis Function Models

- Polynomial basis functions:

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

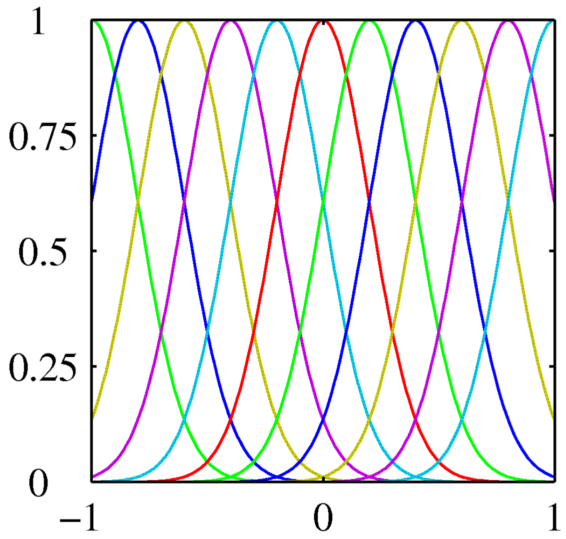
  - These are global; a small change in x affects all basis functions



- 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).



https://jakevdp.github.io/PythonDataScienceHandbook/05.06-linear-regression.html

Linear Basis Function Models

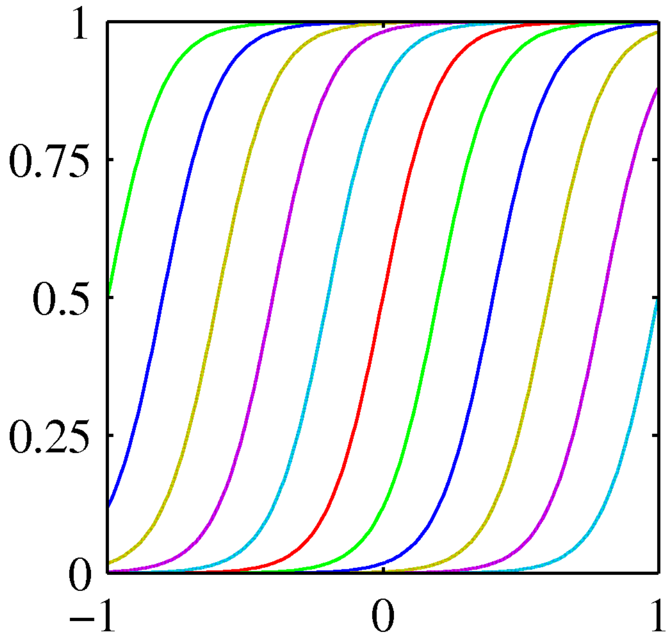
- Sigmoidal basis functions

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

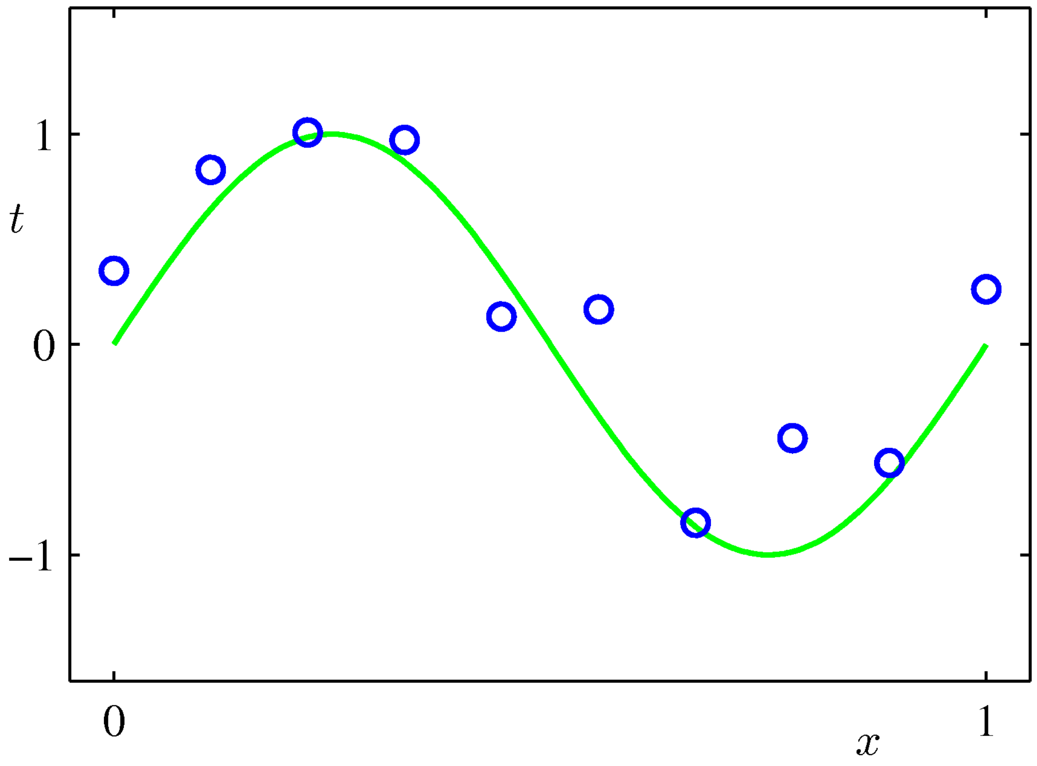
where

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

  – 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



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

Linear Basis Function Models

- Basic Linear Model:

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

- Generalized Linear Model:

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

- 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

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: $\begin{pmatrix} 1 \\ 6 \\ 3 \\ 4 \end{pmatrix}$
  - 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 Algebra Concepts

- Transpose: flips a matrix over its diagonal

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

  – Note: $(Ax)^T = x^T A^T$

(We'll define multiplication soon…)

- Vector norms:
  – L_p norm of $\mathbf{v} = (\mathbf{v}_1,\ldots,\mathrm{v}_k)$ is $\left(\sum_i |v_i|^p\right)^{\frac{1}{p}}$
  – Common norms: L_1, L_2
  – $L_{infinity} = \max_i |\mathrm{v}_i|$
- Length of a vector v is $L_2(\mathrm{v})$

Linear Algebra Concepts

- 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:

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

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

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

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

- 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

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

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

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

Vectorization

- For the linear regression cost function:

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

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

$$= \frac{1}{2n}\underbrace{(\boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{y})^{\intercal}}_{\mathbb{R}^{1\times n}}\left(\underbrace{\overset{\mathbb{R}^{n\times(d+1)}}{\boldsymbol{X}}\overset{\mathbb{R}^{(d+1)\times 1}}{\boldsymbol{\theta}}}_{\mathbb{R}^{n\times 1}} - \underset{\mathbb{R}^{n\times 1}}{\boldsymbol{y}}\right)$$

Let:

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