Linear regression with multiple variables

Andrew Ng

Linear regression with multiple variables

Multiple features

Multiple features (variables)

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

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

Multiple features (variables)

Size (feet²) x_1	**Number of bedrooms** x_2	**Number of floors** x_3	**Age of home (years)** x_4	**Price ($1000)** y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

Notation:

n = number of features

$x^{(i)}$ = input (features) of i^{th} training example.

$x_j^{(i)}$ = value of feature j in i^{th} training example.

$$x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}$$

Hypothesis:

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

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$$

For convenience of notation, define $x_0 = 1$.

$$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \in \Re^{n+1} \qquad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix} \in \Re^{n+1}$$

$$h_\theta(x) = \theta^T x$$

Linear regression with multiple variables

Gradient descent for multiple variables

Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Parameters: $\theta_0, \theta_1, \ldots, \theta_n$

Cost function:

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

Gradient descent:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \ldots, \theta_n)$$

$\}$ (simultaneously update for every $j = 0, \ldots, n$)

Gradient descent

Previously (n=1):

Repeat $\{$

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

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

(simultaneously update θ_0, θ_1)

$\}$

New algorithm $(n \geq 1)$:

Repeat $\{$

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

(simultaneously update θ_j for $j = 0, \ldots, n$)

$\}$

$x_0^{(i)} = 1$

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

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

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

$\ldots$

Linear regression with multiple variables

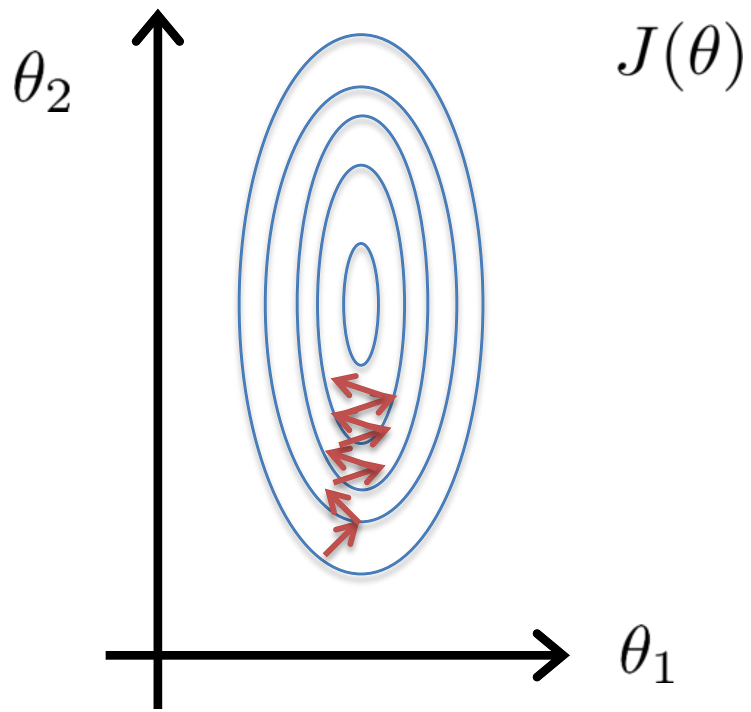
Gradient descent in practice I: Feature scaling

Feature scaling

Idea: Make sure features are on a similar scale.

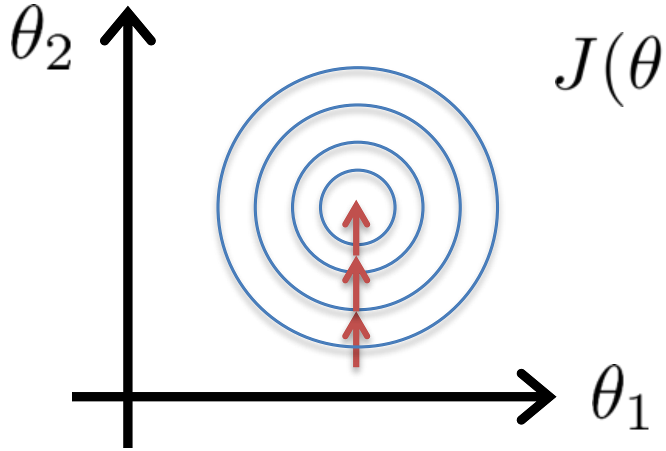
E.g. x_1 = size (0-2000 feet^2)

x_2 = number of bedrooms (1-5)



$$x_1 = \frac{\text{size (feet}^2\text{)}}{2000}$$

$$x_2 = \frac{\text{number of bedrooms}}{5}$$



Feature scaling

Get every feature into approximately a $-1 \leq x_i \leq 1$ range.

Mean normalization

Replace x_i with $x_i - \mu_i$ to make features have approximately zero mean (Do not apply to $x_0 = 1$).

E.g.

$$x_1 = \frac{size - 1000}{2000}$$

$$x_2 = \frac{\#bedrooms - 2}{5}$$

$$x_i \leftarrow \frac{x_i - \mu_i}{S_i}$$

S_i: range or standard deviation

$$-0.5 \leq x_1 \leq 0.5, -0.5 \leq x_2 \leq 0.5$$

Linear regression with multiple variables

Gradient descent in practice II: Learning rate

Gradient descent

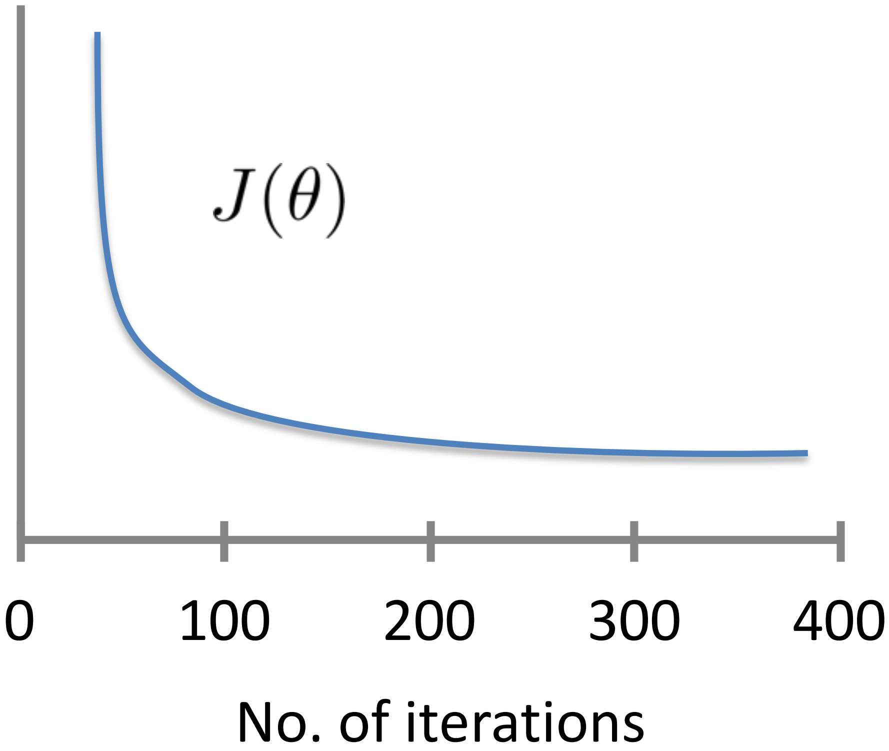
$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

- "Debugging": How to make sure gradient descent is working correctly.

- How to choose learning rate α.

Making sure gradient descent is working correctly

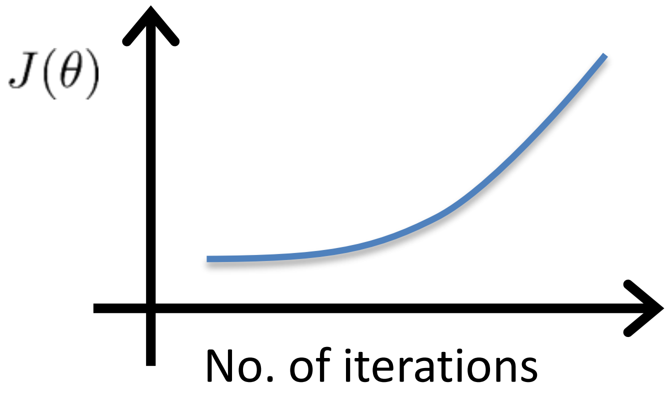
$\min_{\theta} J(\theta)$

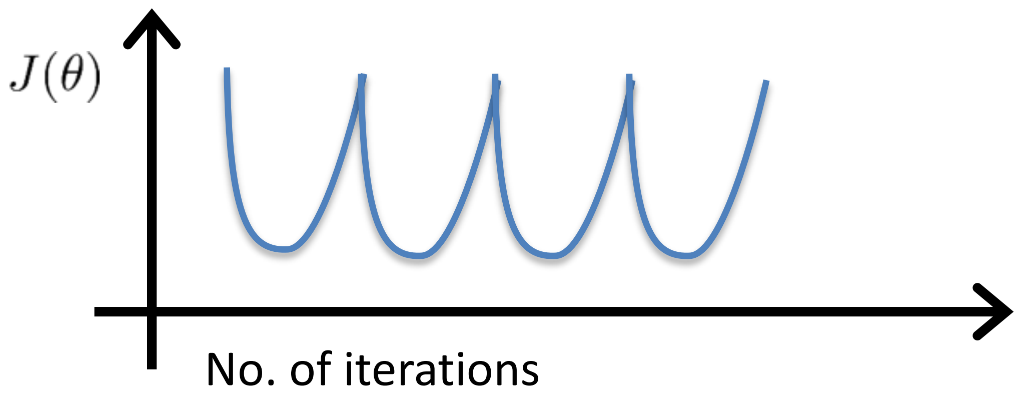


Example automatic convergence test:

Declare convergence if $J(\theta)$ decreases by less than 10^{-3} in one iteration.

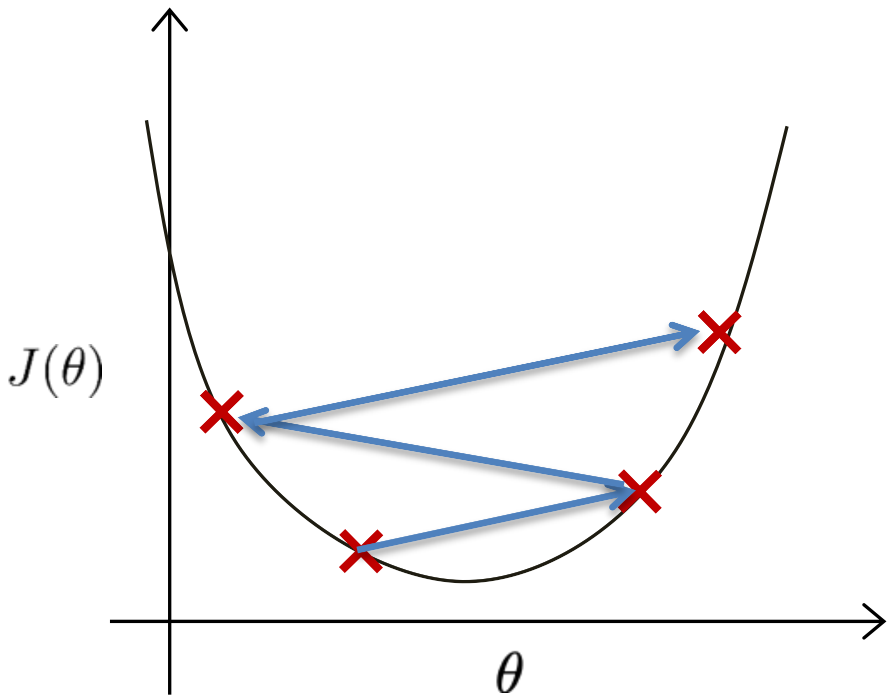
Making sure gradient descent is working correctly





Gradient descent not working.

Use smaller α.



- For sufficiently small α, $J(\theta)$ should decrease on every iteration.
- But if α is too small, gradient descent can be slow to converge.

Summary:

- If α is too small: slow convergence.
- If α is too large: $J(\theta)$ may not decrease on every iteration; may not converge.

To choose α, try

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

Linear regression with multiple variables

Features and polynomial regression

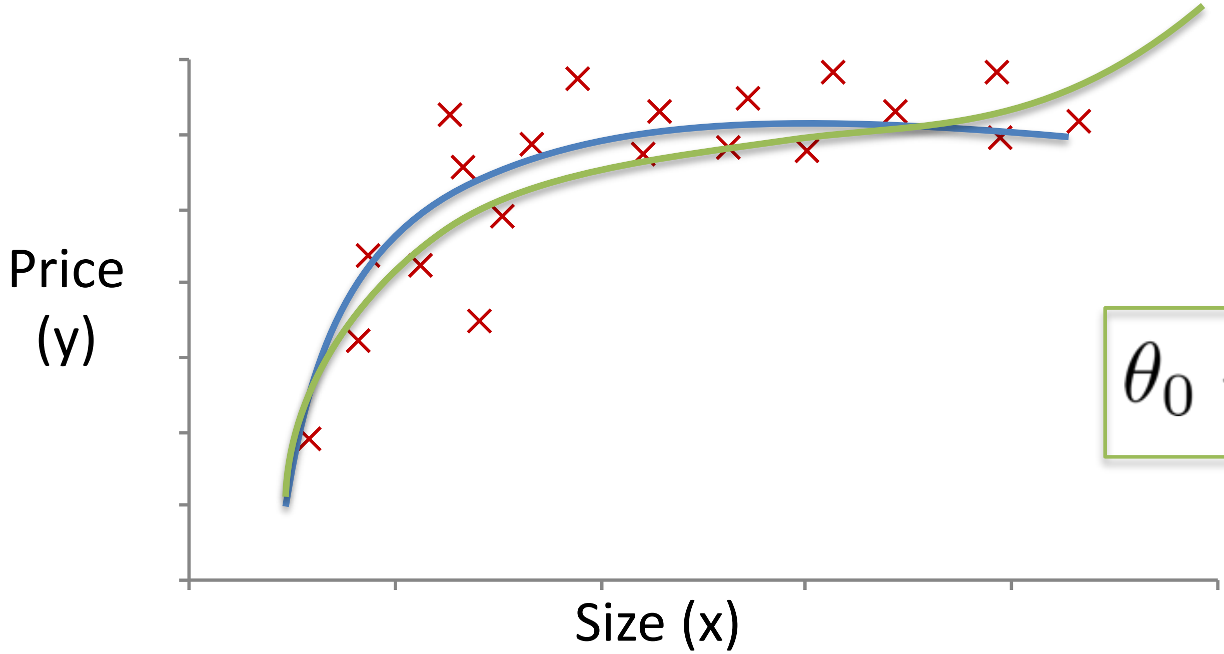
Housing prices prediction

$$h_\theta(x) = \theta_0 + \theta_1 \times \underset{x_1}{frontage} + \theta_2 \times \underset{x_2}{depth}$$

$$x = frontage \times depth$$

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

Polynomial regression



$$\boxed{\theta_0 + \theta_1 x + \theta_2 x^2}$$

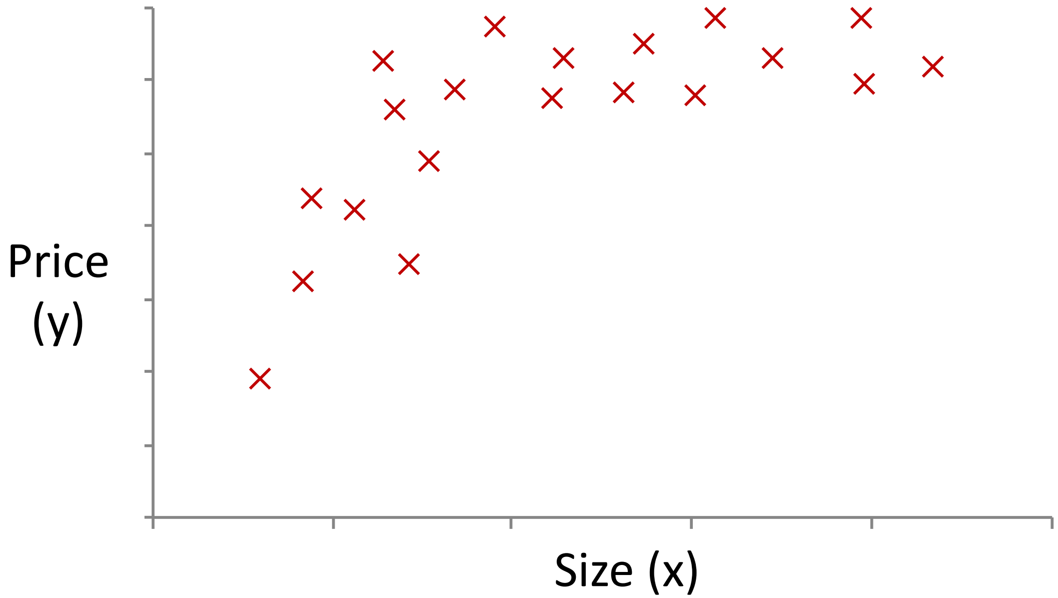
$$\boxed{\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3}$$

$$\begin{aligned} h_\theta(x) &= \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 \\ &= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3 \end{aligned}$$

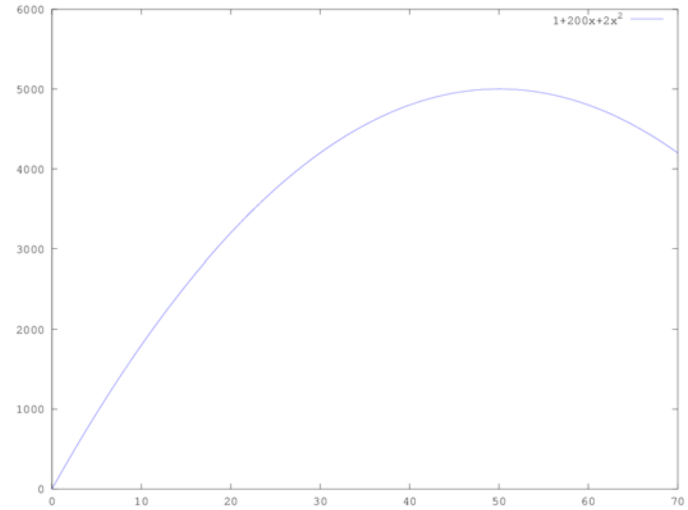
$$\begin{aligned} x_1 &= (size) \\ x_2 &= (size)^2 \\ x_3 &= (size)^3 \end{aligned}$$

important: feature scaling

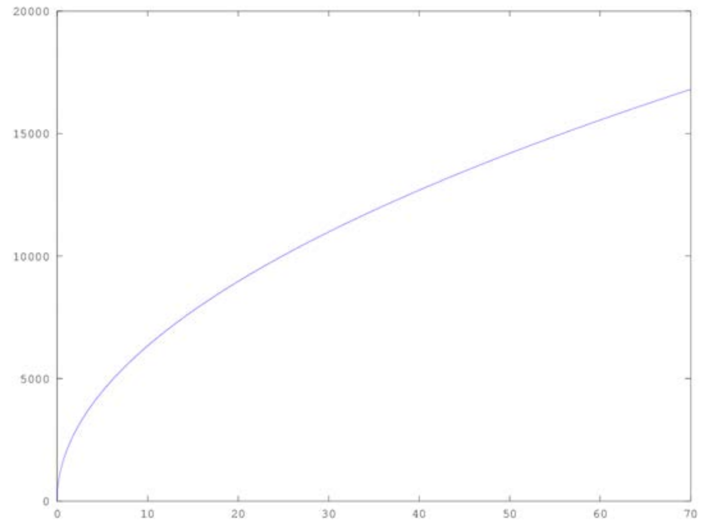
Choice of features



$$1 + 200x - 2x^2$$



$$1 + x + 2000\sqrt{x}$$



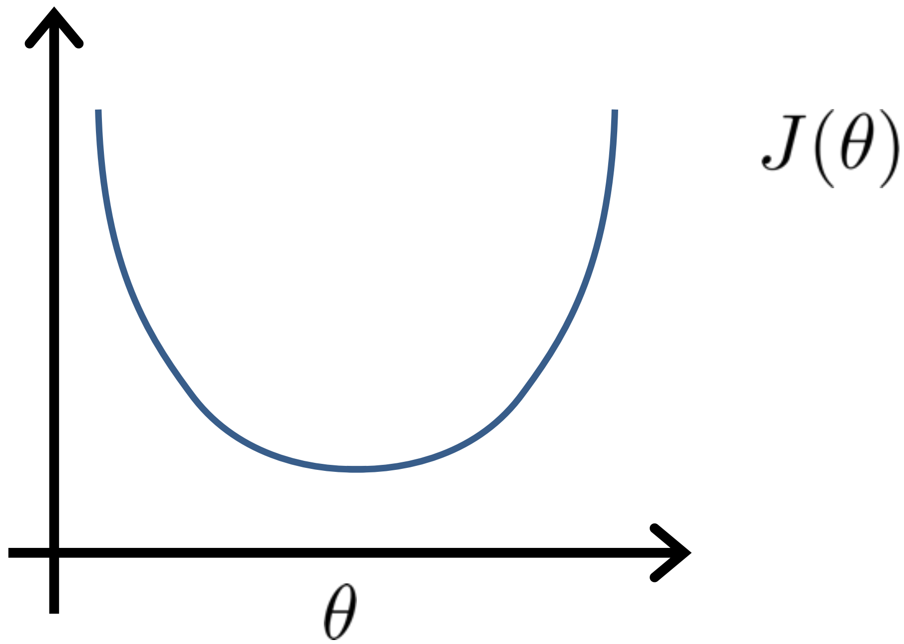
$$h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2(size)^2$$

$$h_\theta(x) = \theta_0 + \theta_1(size) + \theta_2\sqrt{(size)}$$

Linear regression with multiple variables

Normal equation

Gradient Descent



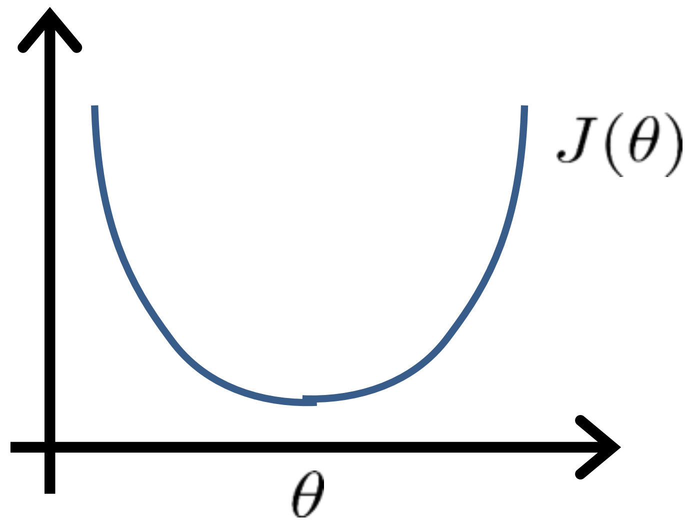
Normal equation: Method to solve for θ analytically

Intuition: If 1D $(\theta \in \mathbb{R})$

$$J(\theta) = a\theta^2 + b\theta + c$$

$$\frac{d}{d\theta} J(\theta) = 0$$

Solve for θ



$\theta \in \mathbb{R}^{n+1}$ $\qquad$ $J(\theta_0, \theta_1, \ldots, \theta_m) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

$\frac{\partial}{\partial \theta_j} J(\theta) = \cdots = 0$ $\qquad$ (for every j)

Solve for $\theta_0, \theta_1, \ldots, \theta_n$

Examples: $m = 4.$

	Size (feet²)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
x_0	x_1	x_2	x_3	x_4	y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \qquad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

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

m **examples** $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})$ **;** n **features.**

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1} \qquad X = \begin{bmatrix} — & (x^{(1)})^T & — \\ — & (x^{(2)})^T & — \\ & \vdots & \\ — & (x^{(m)})^T & — \end{bmatrix}$$

E.g. If $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix}$ $\quad X = \underbrace{\begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_1^{(2)} \\ & \vdots \\ 1 & x_1^{(m)} \end{bmatrix}}_{m \times 2} \quad Y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$

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

$(X^T X)^{-1}$ is inverse of matrix $X^T X$.

Python:

numpy.transpose

numpy.transpose(*a*, *axes=None*)

numpy.matmul

numpy.matmul(*a*, *b*, *out=None*)

Matrix product of two arrays.

numpy.linalg.pinv

numpy.linalg.pinv(*a*, *rcond=1e-15*)

Compute the (Moore-Penrose) pseudo-inverse of a matrix.

No need for feature scaling

m **training examples,** n **features.**

Gradient Descent

- Need to choose α.
- Needs many iterations.
- Works well even when n is large.

Normal Equation

- No need to choose α.
- Don't need to iterate.
- Need to compute $(X^T X)^{-1}$ $O(n^3)$
- Slow if n is very large.
- Up to 10.000

Linear regression with multiple variables

Normal equation and non-invertibility

Normal equation

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

- What if $X^T X$ is non-invertible? (singular/ degenerate)
- Python's pinv (pseudoinverse) will work even if $X^T X$ is non-invertible

numpy.linalg.pinv

numpy.linalg.**pinv**(*a*, *rcond=1e-15*)

Compute the (Moore-Penrose) pseudo-inverse of a matrix.

What if $X^T X$ is non-invertible?

- Redundant features (linearly dependent).
  E.g. $x_1 =$ size in feet^2
  $x_2 =$ size in m^2

- Too many features (e.g. $m \leq n$).
  - Delete some features, or use regularization.