Week 2

Multivariate Linear Regression:

- Multiple Features:

$x_j^{(i)}$ = value of feature j in the i^{th} training example
$x^{(i)}$ = the input (features) of the i^{th} training example
m = the number of training examples
n = the number of features

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

For convenience of notation, define $x_0 = 1$. $(x_0^{(i)} = 1)$

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

$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n$ ($x_0 = 1$)

$= \theta^T x$

$[\theta_0 \ \theta_1 \cdots \theta_n]$ = θ^T, $(n+1) \times 1$ matrix

$\theta^T x$

- Gradient Descent for Multiple Variables:

New algorithm $(n \geq 1)$:

Repeat {

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

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

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

}

$x_0^{(i)} = 1$

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

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

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

...

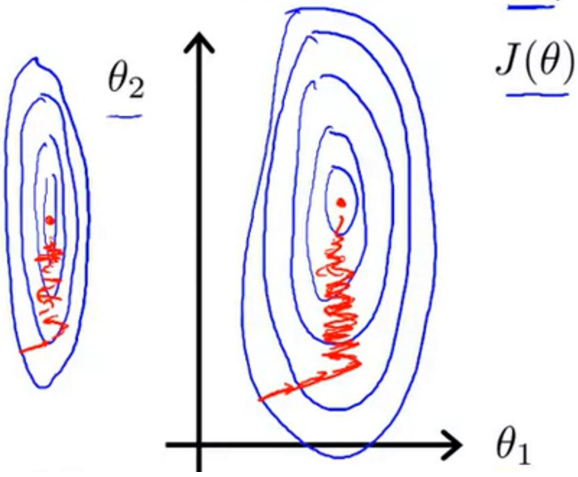
- Feature scaling:**(make gradient descent run much faster and converge)**

Feature Scaling

Idea: Make sure features are on a similar scale.

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

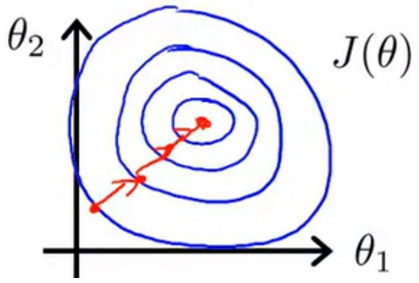
x_2 = number of bedrooms (1-5)



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

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

$0 \le x_1 \le 1$ $\quad 0 \le x_2 \le 1$



The scope of features are different, as a result, It's difficult to converge. So we're going to scale the feature into [-1,1] range. **(range isn't important, we just scale it into a small range.)**:

$0 \le x_1 \le 3$ ✓

$-2 \le x_2 \le 0.5$ ✓

$-100 \le x_3 \le 100$ ✗

$-0.0001 \le x_4 \le 0.0001$ ✗

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

E.g. $x_1 = \frac{size - 1000}{2000}$ Average size = 100

$x_2 = \frac{\#bedrooms - 2}{5}$ 1-5 bedrooms

$-0.5 \le x_1 \le 0.5$ $\quad -0.5 \le x_2 \le 0.5$

$x_1 \leftarrow \frac{x_1 - \mu_1}{s_1}$ $\quad x_2 \leftarrow \frac{x_2 - \mu_1}{s_2}$

μ_1: avg value of x_1 in training set

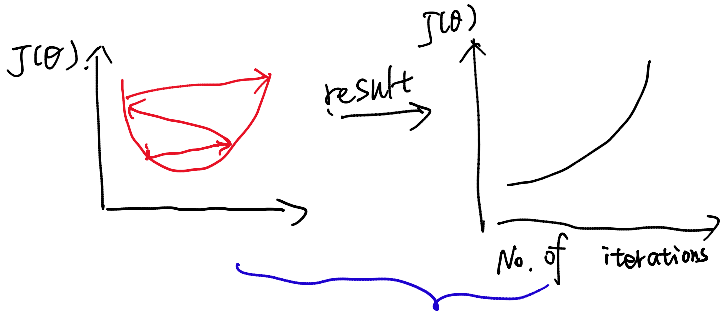
s_1: range (max - min) (or standard deviation)

- Learning rate

  J(θ)should decrease after every iteration

  Example automatic convergence test:

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



α is too big.

plot J(θ) as a function of the number of iterations. to figure ont what's going on.

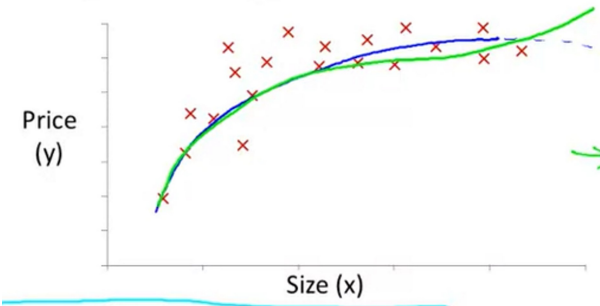
try a range of values for α. and plot the plotting.

- Features and polynomial regression

  Create a new features by yourself

  Transfer the problem into linear regression

Polynomial regression



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

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

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$

$= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3$

$x_1 = (size)$

$x_2 = (size)^2$

$x_3 = (size)^3$

create new features

Size: 1 - 1000

$Size^2$: 1 - 1000,000

$Size^3$: 1 - 10^9

Feature scalaing is very important

Computing Parameters Analytically:

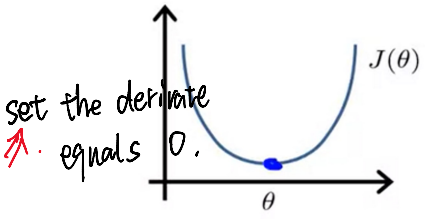
- Normal equation

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

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

$\frac{d}{d\theta} J(\theta) = \cdots \overset{\text{set}}{=} 0$

Solve for θ



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

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

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

to we want: $\theta = (X^T X)^{-1} X^T y$.

inverse of $X^T X$.

$x^{(i)} = \begin{bmatrix} x_0^{(i)} = 1 \\ x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix}$ design matrix $= \begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}$ $m \times (n+1)$.

m training examples, n features.

Gradient Descent	Normal Equation
• Need to choose α. (disadvantage)	• No need to choose α. (advantage)
• Needs many iterations.	• Don't need to iterate.
• Works well even when n is large. ($n = 10^6$)	• Need to compute $(X^T X)^{-1}$ ($n \times n$, $O(n^3)$)
	• Slow if n is very large. ($n = 100$, $n = 1000$, $\leftarrow$ - - - $n = 10000$)

- Normal Equation Non-invertibility

  Like above we use normal equation to compute theta, at that computing we use inverse operate. But sometimes, singular or degenerate matrics is non-invertible.

  At the learning time, the reason for X transpose X to be non-invertible is redunant features and smaller m than n(m <= n, the number of samples are less or equal to the number of features)

Tip : verctorization: