WEEK #2.

(2) MULTIVARIATE LINEAR REGRESSION

2.1. Multiple features:

ex/ 1) size x_1
2) # of bedrooms x_2
3) # of floors x_3
4) Age of home x_4

} Price y

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

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

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$$

(hypothesis)

$$h_\theta(x) = \theta^T x = [\theta_0 \ \theta_1 \ \dots \ \theta_n] \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}$$

2.2. Gradient descend for multiple variables: (n variables/features)

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

Parameters: θ $(\theta_0, \theta_1, \theta_2, \dots, \theta_n)$ "n+1" dimentional vector.

Cost func'n: $J(\theta) = J(\theta_0, \theta_1, \theta_2, \dots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$

Gradient descent:

Repeat {

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

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

for $n=2$:

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

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

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

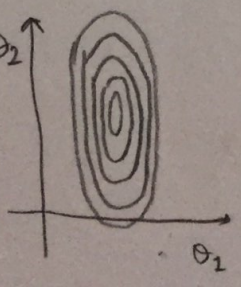
2.3. Gradient descent in Practice 1 - Feature Scaling:

⇒ Make sure that the features are on a similar scale, then gradient descent can converge more quickly!

ex/ x_1 = size $(0 - 2000 \text{ feet}^2)$

x_2 = # of bedrooms $(1-5)$

If you ran grad. descent, on this cost fxn, gradients can take a long time to reach global minimum.

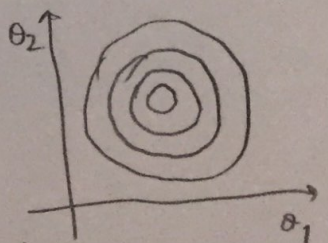


$$x_1 = \frac{\text{size (feet}^2)}{2000}$$

$$x_2 = \frac{\text{\# bedrooms}}{5}$$

$$0 \le x_1, x_2 \le 1$$

Can converge much faster.



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

$0 \le x_1 \le 3$ ✓ $\quad -2 \le x_2 \le 0.5$ ✓ $\quad -100 \le x_3 \le 100$ ✗ $\quad -0.0001 \le x_4 \le 0.0001$ ✗

Mean Normalizatn:

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

ex/ $x_1 = \frac{\text{size} - 1000}{2000}$ $\quad -0.5 \le x_1 \le 0.5$

$x_2 = \frac{\#\text{bedrooms} - 2}{5}$ $\quad -0.5 \le x_2 \le 0.5$

$x_1 \leftarrow \frac{x_1 - \mu_1}{s_1}$ ← avg. value of x_1 in training set

range (or standard deviation) (max-min)

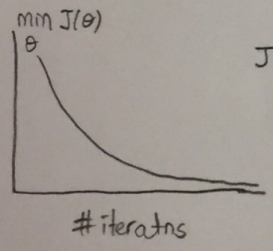
$x_2 \leftarrow \frac{x_2 - \mu_2}{s_2}$

2.4. Gradient Descent in Practice 2 - Learning Rate:

Gradient descent

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

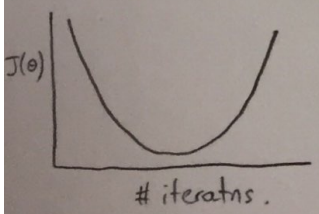
- "Debugging": how to make sure grad.desc. is working correctly
- How to choose learning rate α.



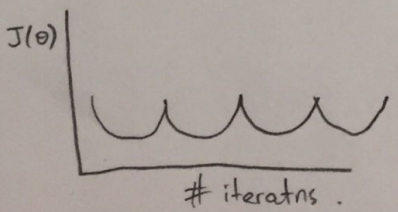
$J(\theta)$ should decrease after every iteratn

Example automatic convergence test:

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



if grad.desc. does not work ⇒ use smaller α.



⇒ For sufficiently small α, $J(\theta)$ should decrease on every iteratn.

⇒ But if α is too small, grad.desc. can be slow to converge

Summary: - if α is too small → slow convergence
- if α is too large → $J(\theta)$ may not decr. on every itrn, may not converge

Try to choose α: 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, ...

2.5. Features and Polynomial Regression:

$h_\theta(x) = \theta_0 + \theta_1 \times \underbrace{\text{frontage}}_{x_1} + \theta_2 \times \underbrace{\text{depth}}_{x_2}$ ⇒ You can create new features by yourself.

$h_\theta(x) = \theta_0 + \theta_1 (x_1)$ ← land area $(x_1 \cdot x_2)$

$h_\theta(x) = \theta_0 + \theta_1 (\text{size}) + \theta_2 (\text{size})^2 + \theta_3 (\text{size})^3$

(3.) COMPUTING PARAMETERS ANALYTICALLY.

3.1. Normal Equatn: method to solve for θ analytically.

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

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

solve for θ

$\theta \in \mathbb{R}$

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

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

solve for $\theta_0, \theta_1, \dots, \theta_n$

$\theta \in \mathbb{R}^{n+1}$

$$X = \begin{bmatrix} -(x^{(1)})^T- \\ -(x^{(2)})^T- \\ -(x^{(m)})^T- \end{bmatrix}_{(m \times (n+1))} \qquad y = \begin{bmatrix} \\ \\ \\ \end{bmatrix}_{(m \times 1)}$$

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

pinv $(X' * X)^{-1} * X' * y$

⇒ if you are using Normal Eqn method ⇒ feature scaling is not actually necessary

m training example, n features:

Gradient descent:	Normal Eqn:
- Need to choose α	- No need to choose α
- Needs many iteratrs.	- Don't need to iterate.
- Works well even when n is large (10K)	- Need to compute $(X^T X)^{-1}$
	- Slow if n is very large ($\geq$ 10K)

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

-what if $X^T X$ is non-invertible? (singular/degenerate)

pinv — pseudo inverse
inv — inverse

→ you may have redundant features (linearly dependent)

→ you may have too many features ($m \leq n$)
- delete some features, or use regularizatn
- use more training examples.

(4.) Programming Assignment (✓)