Stanford ML

WEEK 2:

Linear Regression with Multiple Variables

- Let $n \equiv$ # of features
- $x^{(i)} \equiv$ input features of i^{th} training example
- $x_j^{(i)} \equiv$ value of feature j in i^{th} training example.
- Now for linear regression with n features, our hypothesis becomes:

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

↳ The coefficients can be +VE or -VE and indicates how sensitive our hypothesis is to each feature x_i.

↳ let $x_0 = 1$ ALWAYS $\therefore x_0^{(i)} = 1$ } ALLOWS COMPACT NOTATION

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

↳ Representing the input features for a particular example as an $n+1$ vector and doing same for the parameters we see that

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

MULTIVARIATE LIN. REG.

Multivariate Gradient Descent

- Remember the cost function is defined as follows:

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

↳ where $\theta \in \mathbb{R}^{n+1}$

which is equivalent to:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(\theta^T x^{(i)} - y^{(i)} \right)^2 \quad \text{AND}$$

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left[\left(\sum_{j=0}^{n} \theta_j x_j^{(i)} \right) - y^{(i)} \right]^2$$

Gradient descent:

Repeat {

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

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

}

New Algorithm

$\partial / \partial \theta_j \, J(\theta)$

Repeat {

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

}

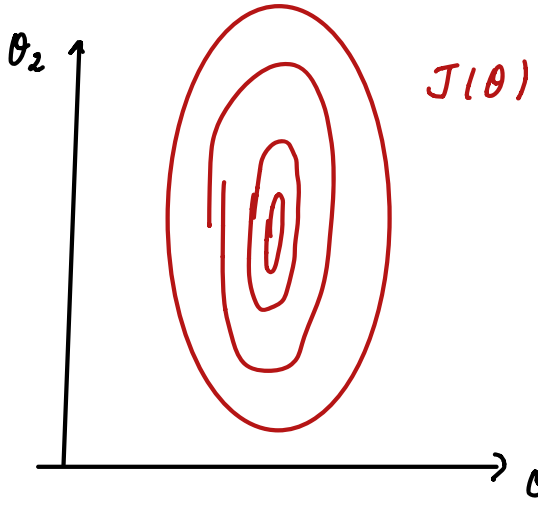
↳ simultaneously update θ_j for $j = 0, \ldots, n$

Feature Scaling

- Trick used to ensure all features are on a similar scale to ensure optimal performance of the gradient descent algorithm

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

x_2 = # of bedrooms (1-5)



- Having one feature taking on much larger ranges LONG, SKINNY contours of $J(\theta)$
- ∴ Re-scale to ensure fast CONVERGENCE

$$\Rightarrow x_1 = \frac{\text{size (feet}^2\text{)}}{2000} \qquad x_2 = \frac{\#\text{ of bedrooms}}{5}$$

- Using the new x_1, x_2 ⟶ more circular contours ⟶ faster convergence
- Try to get every feature:

$$\boxed{-1 \leq x_i \leq 1}$$

↳ note you can poorly scale features both too SMALL and too LARGE

OR

$$x_i' = \frac{x_i - \mu_i}{\sigma_i}$$

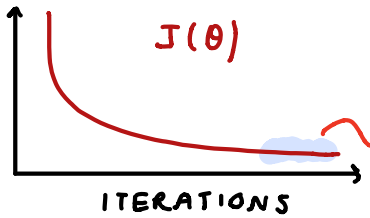
σ_i ← STDEV.

$$\begin{cases} x_1' = \dfrac{\text{size} - 1000}{2000} \\ x_2' = \dfrac{\#\text{bedrooms} - 2}{5} \end{cases}$$

MEAN NORMALIZATION

Setting α and Debugging

- To ensure gradient descent are may wish to plot $J(\theta)$ as a function of time



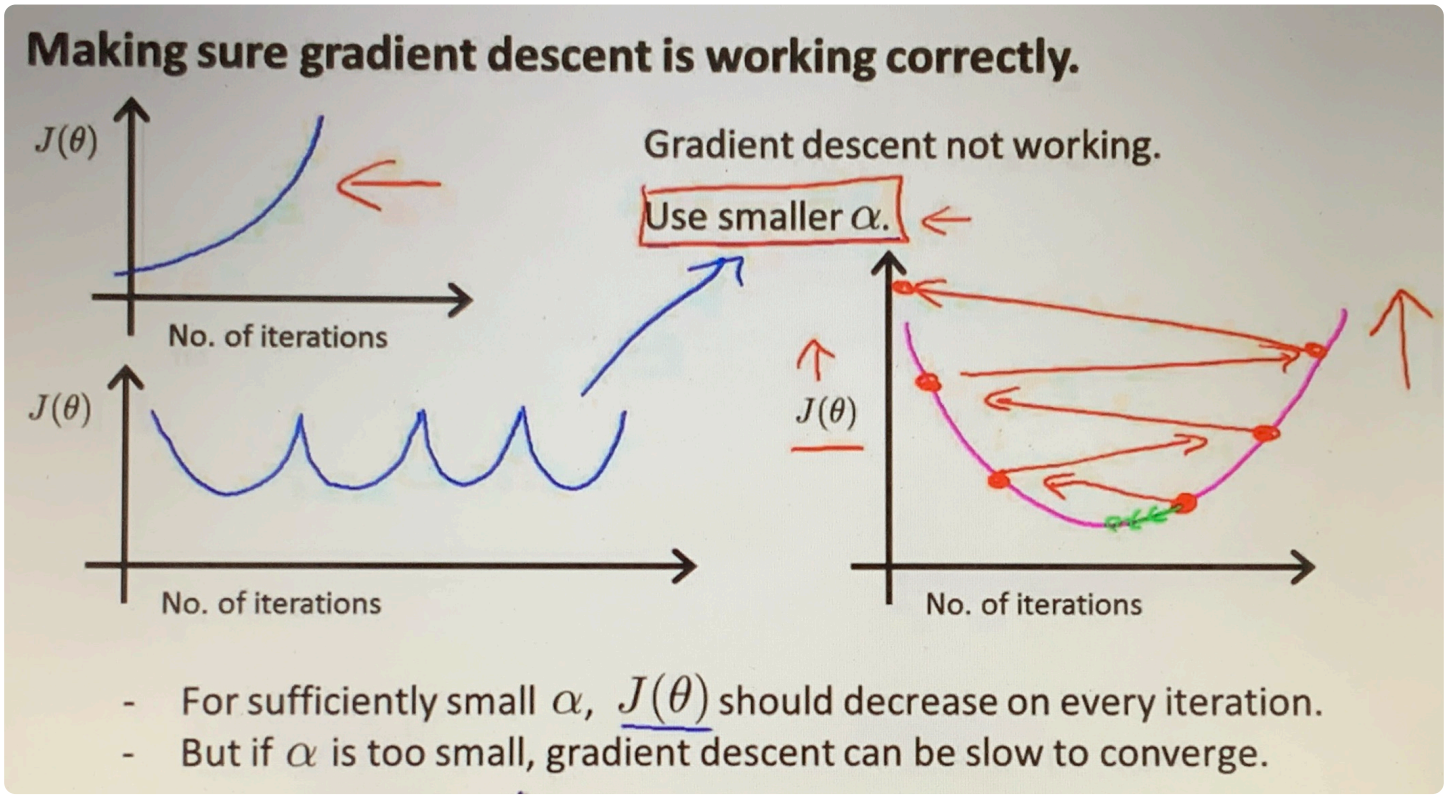
- We expect $J(\theta)$ to DECREASE EVERY ITERATION } MATH PROVEN.

↳ judging convergence

- Or we can simply use some tolerance (10^{-3}) and if $\Delta J(\theta) \leq 10^{-3} \Rightarrow$ CONVERGED!

↳ TOL is HARD TO CHOOSE
∴ PLOT OFTEN BETTER

Common Errors



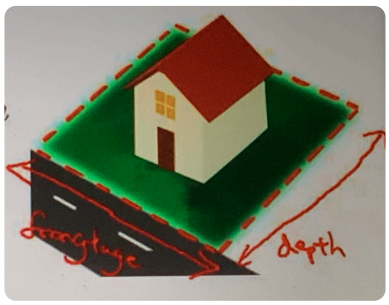
- α takes values

$$\alpha \in [10^{-2}, 1] \text{ ish } \ldots$$

in steps of $\times 3$

Features and Polynomial Regression

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

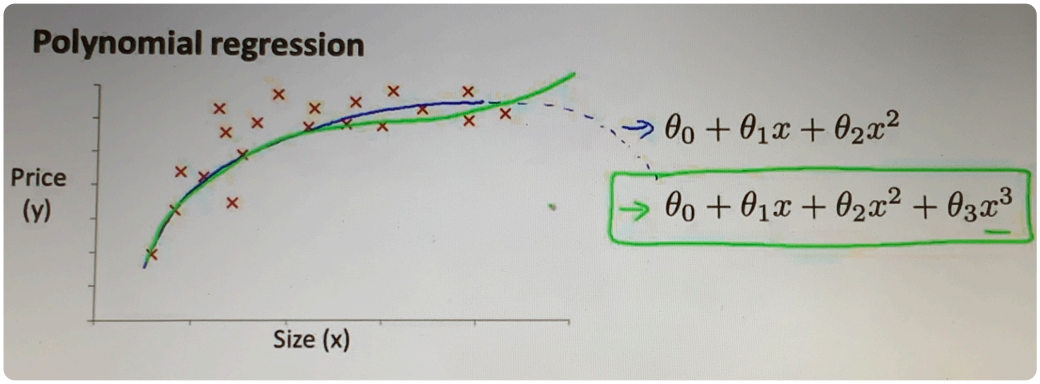


- Could make new feature:

Area: x = frontage x depth

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

↳ AREA



How do we fit higher order polynomials to data?

- We could

$$h_\theta = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$
$$= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3$$

↳ define features as so, then simply use linear regression

TRICK: HIDE THE POWERS IN THE DEFS!

NOTE: Feature scaling is now CRITICAL for fast convergence

- Another reasonable choice:

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

Normal Equation

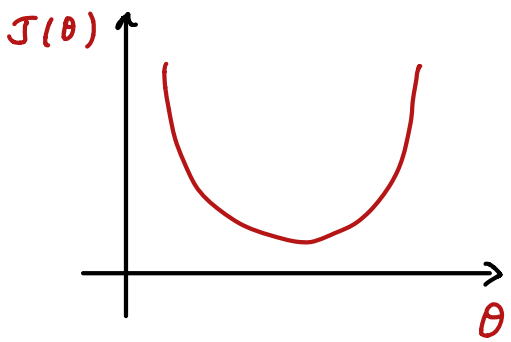
* Method to solve for optimal θ value ANALYTICALLY

INTUITION: Consider 1D case where $\theta \in \mathbb{R}$

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

$$\frac{dJ(\theta)}{d\theta} = 2a\theta + b = 0$$

$$\Rightarrow \theta = -b/2a$$



REALITY:

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

① $\Rightarrow$ $\partial / \partial \theta_j \; J(\theta) \quad \forall \; j = 0, \dots, n$ set $= 0$

② $\Rightarrow$ solve for $\theta_0, \theta_1, \dots, \theta_n$

Ex:

	x_0	Size (feet²) x_1	Number of bedrooms x_2	Number of floors x_3	Age of home (years) x_4	Price ($1000) y
1	1	2104	5	1	45	460
2	1	1416	3	2	40	232
3	1	1534	3	2	30	315
4	1	852	2	1	36	178

$m \times (n+1)$

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}$$

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

m - dimensional vector

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

General Case:

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

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1}$$

$$X = \begin{bmatrix} — (x^{(1)})^T — \\ — (x^{(2)})^T — \\ \vdots \\ — (x^{(m)})^T — \end{bmatrix}$$

(design matrix) $[m \times (n+1)]$

- If using this method FEATURE SCALING NOT NEEDED
- If $n \gtrsim 10^5$ maybe switch to GRADIENT DESCENT

m training examples, n features.

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

Normal Equation Non-Invertibility

- What if:

$X^T X$ is non-invertible?

↳ MATLAB has special methods to handle this so it always works (PSEUDO INVERSE)

CAUSES

① Redundant features < size in feet^2 / size in m^2 } LINEARLY DEPENDENT

② Too many features e.g. ($m \leq n$)

∴ delete some features, or use REGULARIZATION (later)