WEEK 2

Multiple Features (Multivariate Linear Regression)

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

Ex: $h_\theta(x) = 80 + 0.1 x_1 + 0.01 x_2 + 3 \times 3 - 2 \times 4$

(x_1: price increases slightly with size; x_2: price increases by 3 for every bedroom; 3×3: price decreases as house ages)

Notation

	Size	No. of bedrooms	No. of Floors	Age	Price
House	2104	5	1	45	460
House 2	1416	3	2	40	232
House 3	1534	3	2	30	315
House 4	852	2	1	36	178
House ...	...	...	...	...	...

m (no. of houses) = 47

n (no. of elements) = 4

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

Not square, $x^{(1)}$ → $x^2 = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}$ ←

$x_j^{(i)}$ - value of j feature in ith training example

$x_3^{(2)} = 2$

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

Let $x_0 = 1$

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 \dots$$

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

Scanned with CamScanner

Gradient Descent for multiple Variables

Let θ = $n+1$ dimension vector containing $\theta_0, \theta_1, \theta_2, \ldots \theta_n$

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

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

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

Gradient Descent:

Repeat {

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

~~$J(\theta_0, \theta_1 \ldots \theta_n)$~~

Simultaneously update for every $j = 0, 1 \ldots n$

}

Gradient Descent for Multivariate Linear Regression

Repeat {

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

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

. . .

}

Therefore,

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 $j = 0, 1 \ldots n$

}

Scanned with CamScanner

Practical Tricks

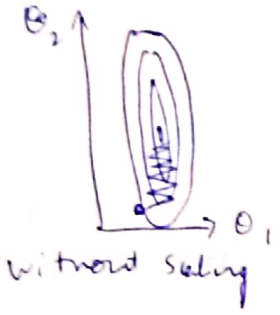
~~Make sure~~

1. Feature Scaling

a) Make sure features are on a similar scale.

Exa: x_1 = size (0 - 2000 feet^2)

x_2 = no. of bedroom (1-5)



Without scaling

Without scaling, convergence takes more time because it will oscillate back and fourth

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

$x_2 = \frac{\text{no. of bedroom}}{5}$

$\therefore x_i = \frac{\text{feature value}}{\text{Maximum size}}$

b) Make sure features are approximately in the range $-1 \leq x_i \leq 1$

2. Mean Normalization

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

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

μ_i ← average value of x in training set

s_i ← Range (Max - Min)

Ex:

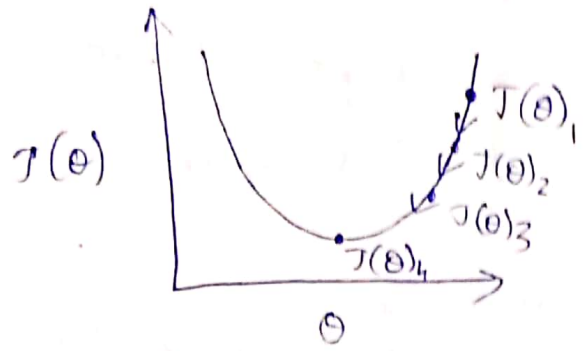
$$x_i = \frac{\text{size (feet}^2) - 1000}{2000}$$

Scanned with CamScanner

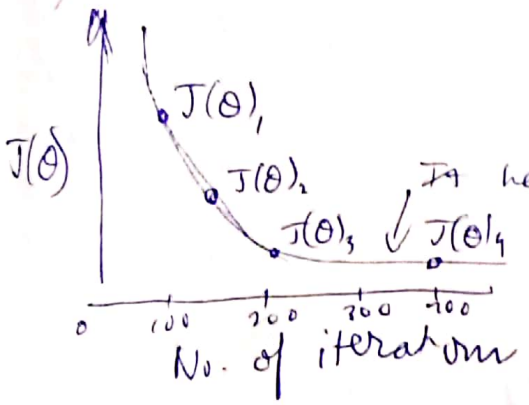
3. Debugging

To make sure gradient descent works properly, plot $J(\theta)$ with no of iterations

w.r.t θ



Therefore gradient descent is supposed to decrease and become uniform.

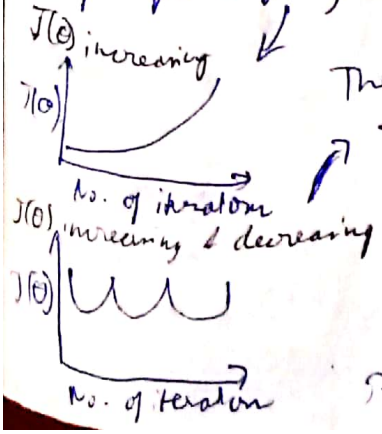


$J(\theta)$ should decrease after every iteration and become uniform

Automatic convergence: if $J(\theta)$ decreases by less than 10^{-3} in one iteration. This is another method but Andrew prefers using plotting.

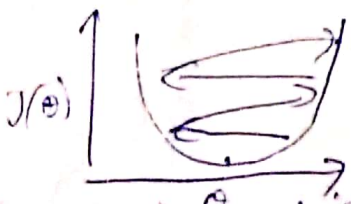
Choosing α

If you get,



Then use smaller α.

This happens because



Step size is too big

- If α is small
  - slow convergence
- If α is big
  - $J(\theta)$ may not decrease on every iteration: no convergence

To choose α try, ..., 0.001, 0.01, 0.1, 1, ...

Scanned with CamScanner

Features

You can choose your features. If length(l) and breadth(b) is a feature

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

We can use area = length × breadth

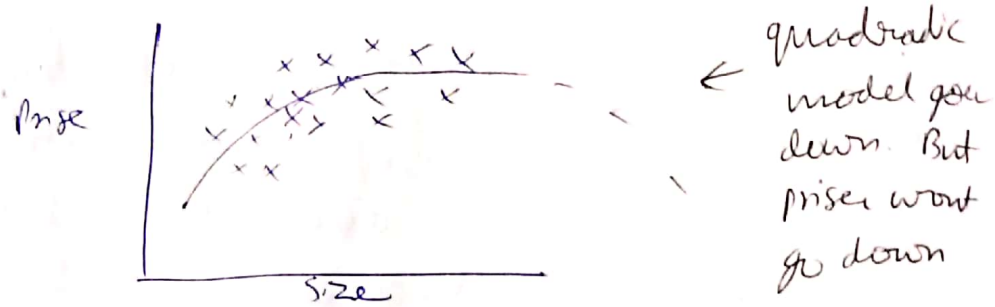
$$h_\theta(x) = \theta_0 + \theta_1 x_1 \leftarrow \text{Area}$$

Polynomial Regression

Used when a curve fits better than a straight line

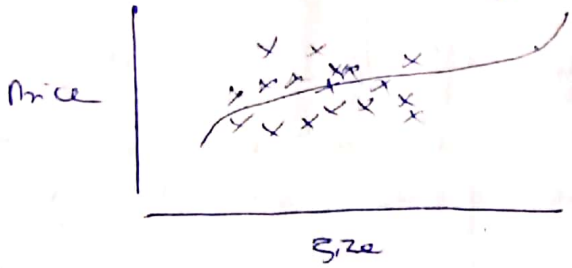
i) Quadratic Model

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



2] Cubic Model

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



Note: We use multivariate linear regression to implement this taking

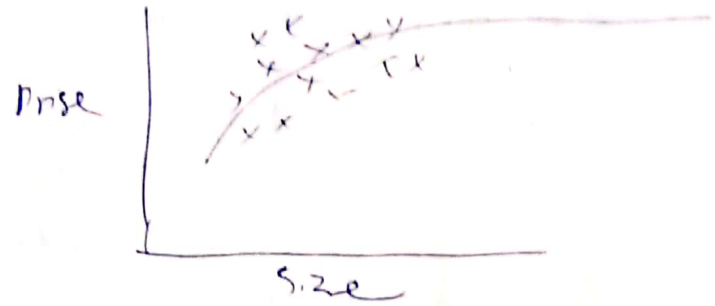
$x_1 = (\text{size})$

$x_2 = (\text{size})^2$

$x_3 = (\text{size})^3$

Scanned with CamScanner

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



Therefore we can use polynomial regression to choose features that produce a curve that best fits our data.

Normal Equation

Another method to find the minima apart from gradient descent. This is an analytical method that helps us directly find the values of θ rather than iterating to find it.

$$X = \begin{bmatrix} 1 & x_1^1 & x_2^1 & x_3^1 \\ 1 & x_1^2 & x_2^2 & x_3^2 \\ 1 & x_1^3 & x_2^3 & x_3^3 \\ 1 & x_1^4 & x_2^4 & x_3^4 \end{bmatrix}$$

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} \qquad \theta = (X^T X)^{-1} X^T y$$

Ex

x_0	Size x_1	No of bedrooms x_2	No. of floors x_3	Age of home x_4	Price 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

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

Scanned with CamScanner

Gradient Descent	Normal Equation
• Need to choose α	• No need to choose α
• Needs iteration	• No iteration (direct answer)
• Works well for large n (no. of features)	• Works well for len n. Since it uses matrix, if we use $n > 10,000$, the computation takes too much time
• Feature scaling required	• Feature scaling not required

Vectorization

We can use vectorization in programming to simplify programs. For $h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j = \theta^T x$

Unvectorized implementation

```
prediction = 0.0;
for j = 1 : n + 1,
    prediction = prediction +
                 theta(j) * x(j)
end;
```

Vectorized implementation

```
prediction = theta' * x;
```

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

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

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

for $n = 2$

Real numbers $\mathbb{R}$

$$\theta := \theta - \alpha \delta$$

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

$\begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \end{bmatrix}$ vector is represented like this in matrix

$\begin{bmatrix} \delta_0 \\ \delta_1 \\ \vdots \end{bmatrix}$

where

$$\delta_0 = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_0^{(i)}$$

$$\delta_1 = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right) x_1^{(i)}$$

Scanned with CamScanner