4. MULTIVARIATE LINEAR REGERESSION:

Multiple features (variables).

Size (feet^2) 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
…	…	…	…	…

$m = 47$

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

Notation:

- n = number of features $\quad n = 4$
- $x^{(i)}$ = input (features) of i^{th} training example.
- $x_j^{(i)}$ = value of feature j in i^{th} training example.

$x_3^{(2)} = 2$

Hypothesis:

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

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

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

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

$$= \boxed{\theta^T x.}$$

$$\underbrace{[\theta_0 \ \theta_1 \cdots \theta_n]}_{\substack{\theta^T \\ (n+1)\times 1 \\ \text{matrix}}} \underbrace{\begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}}_{x}$$

$\theta^T x$

Multivariate linear regression.

X = features vector or design vector

Θ^T = transpose of Θ

Θ = parameter vector

$m =$ the number of training examples
$n =$ the number of features

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

In order to develop intuition about this function, we can think about θ_0 as the basic price of a house, θ_1 as the price per square meter, θ_2 as the price per floor, etc. x_1 will be the number of square meters in the house, x_2 the number of floors, etc.

Cost function:

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

Gradient descent:

Repeat {

$$\rightarrow \theta_j := \theta_j - \alpha \boxed{\frac{\partial}{\partial \theta_j} \cancel{J(\theta_0, \dots, \theta_n)}\ \underline{J(\Theta)}}$$

}

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

- In **linear regression** with **ONE VARIABLES**: n=1

➔thus n+1 = 2 ➔ for Θ_0 and Θ_1

For **multiple variables**: n > 1

New algorithm $(n \geq 1)$:

Repeat {

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

(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

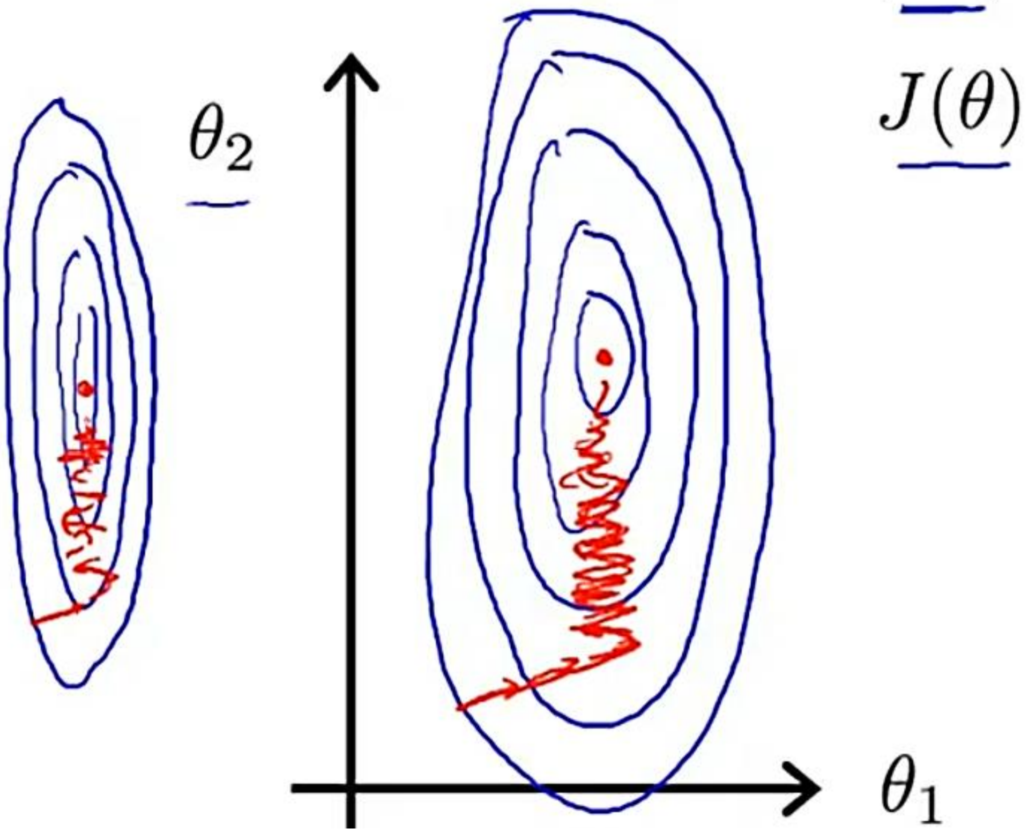
Idea: Make sure features are on a similar scale.

- Used when all input var. have different range of allowed values. This makes optimizing slower. Its tedious to find the local minima.

Example: if diff ranges are used the contours are quite **steep** type

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

x_2 = number of bedrooms (1-5)



- θ will descend quickly on small ranges and slowly on large ranges, and so will oscillate inefficiently down to the optimum when the variables are very uneven.

To solve this: we can change **scaling** of **x_1 and x_2**

This will make the contours more **balanced**.

This is done to bring approximate values of all x_i near a **same ran**ge.

$$-1 \leq x_i \leq 1$$

-1 and 1 are not necessary for all x_i …we can work with nearly equal ranges, like -3 to 3, etc… **comparable ranges**

- We can **speed up gradient descent** by having each of our input values in roughly the same range.

Ranges that would work:

$0 \le x_1 \le 3$ ✓ $\qquad -3$ to 3

$-2 \le x_2 \le 0.5$ ✓ $\qquad -\frac{1}{3}$ to $\frac{1}{3}$ ✓

Ranges that won't work: if ranges are too larger or too smaller than ± 1

$-\underline{100} \le x_3 \ \boxed{100}$ ✗

$-0.0001 \le x_4 \le \boxed{0.0001}$ ✗

NOTE: $x_0 = 1$ **always**. Its scaling is not changed.

- There are two ways to change the ranges of x:
  - Feature scaling
  - Mean normalization

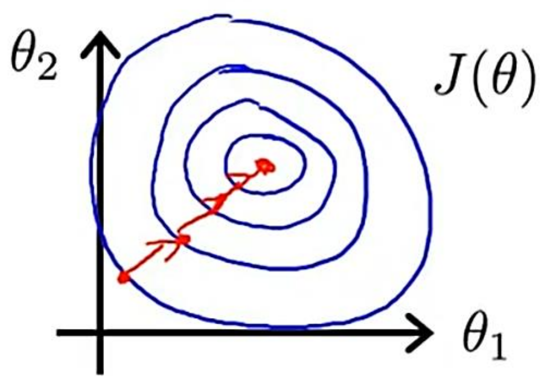
Feature Scaling

Get every feature into approximately a $\boxed{-1 \le x_i \le 1}$ range.

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

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

$0 \le x_1 \le 1 \qquad 0 \le x_2 \le 1$



➔ Feature scaling involves dividing the input values by the range (i.e. the *maximum value minus the minimum value*) of the input variable, resulting in a new range of just 1.

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

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

For x1 : average size = 1000 Range = 2000 = upper limit – lower limit **For x2** : average size = 2 Range = 5

➢ In mean normalization we try to bring x_i in approx. range:

$$\boxed{-0.5 \le x_1 \le 0.5} \quad \boxed{-0.5 \le x_2 \le 0.5}$$

$$x_i := \frac{x_i - \mu_i}{s_i}$$

$$x_1 \leftarrow \frac{x_1 - \mu_1}{s_1}$$

avg value of x_1 in training set

range (max - min)

(or standard deviation)

Where μ_i is the **average** of all the values for feature (i) and s_i is the range of values (max - min), or s_i is the standard deviation.

Note that dividing by the range, or dividing by the **standard deviation**, give different results

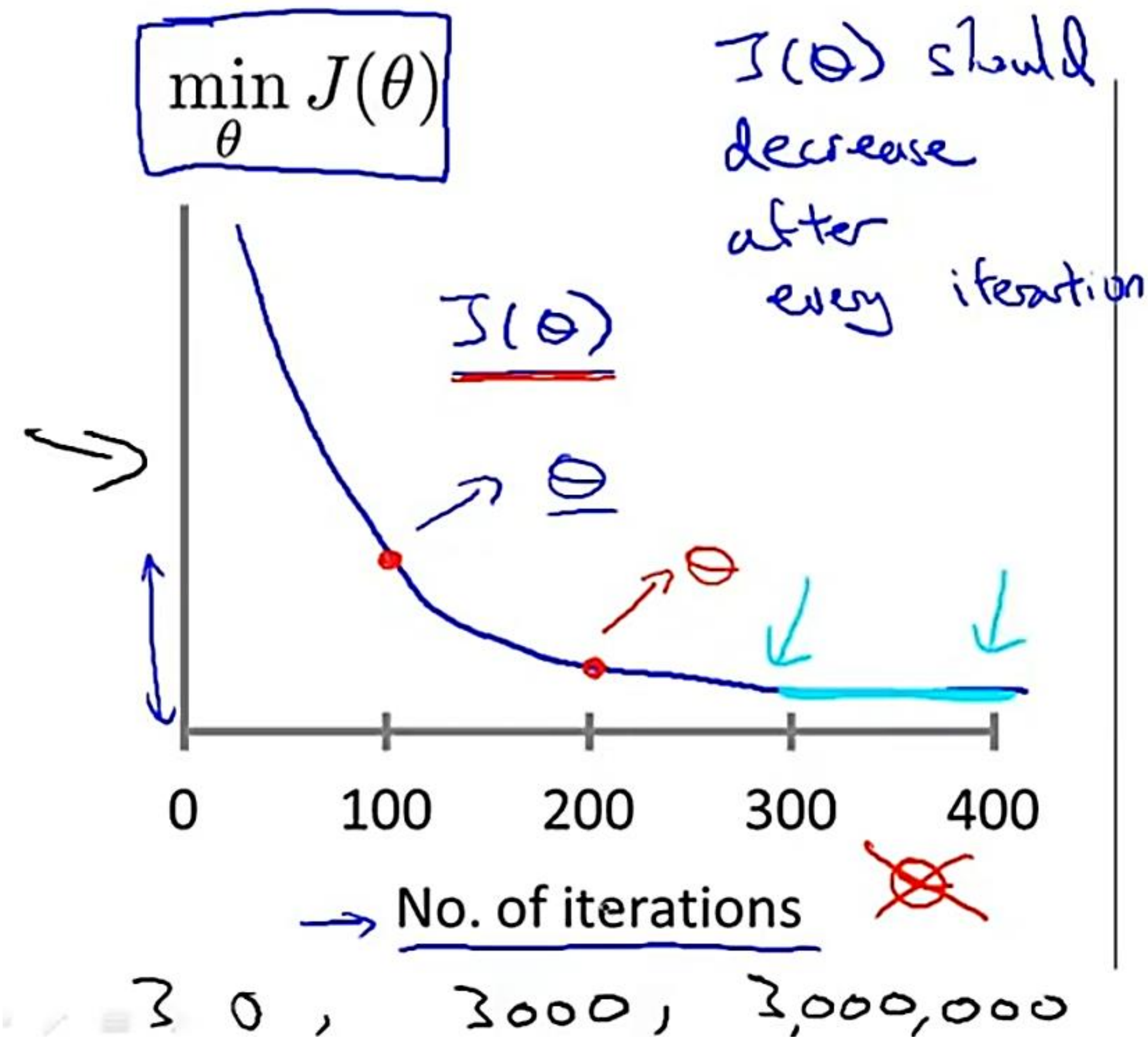
---

PRICTICAL TIPS: for grad desc.

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

- How to choose learning rate $\boxed{\alpha.}$

Making sure gradient descent is working correctly:



The goal is to minimize J

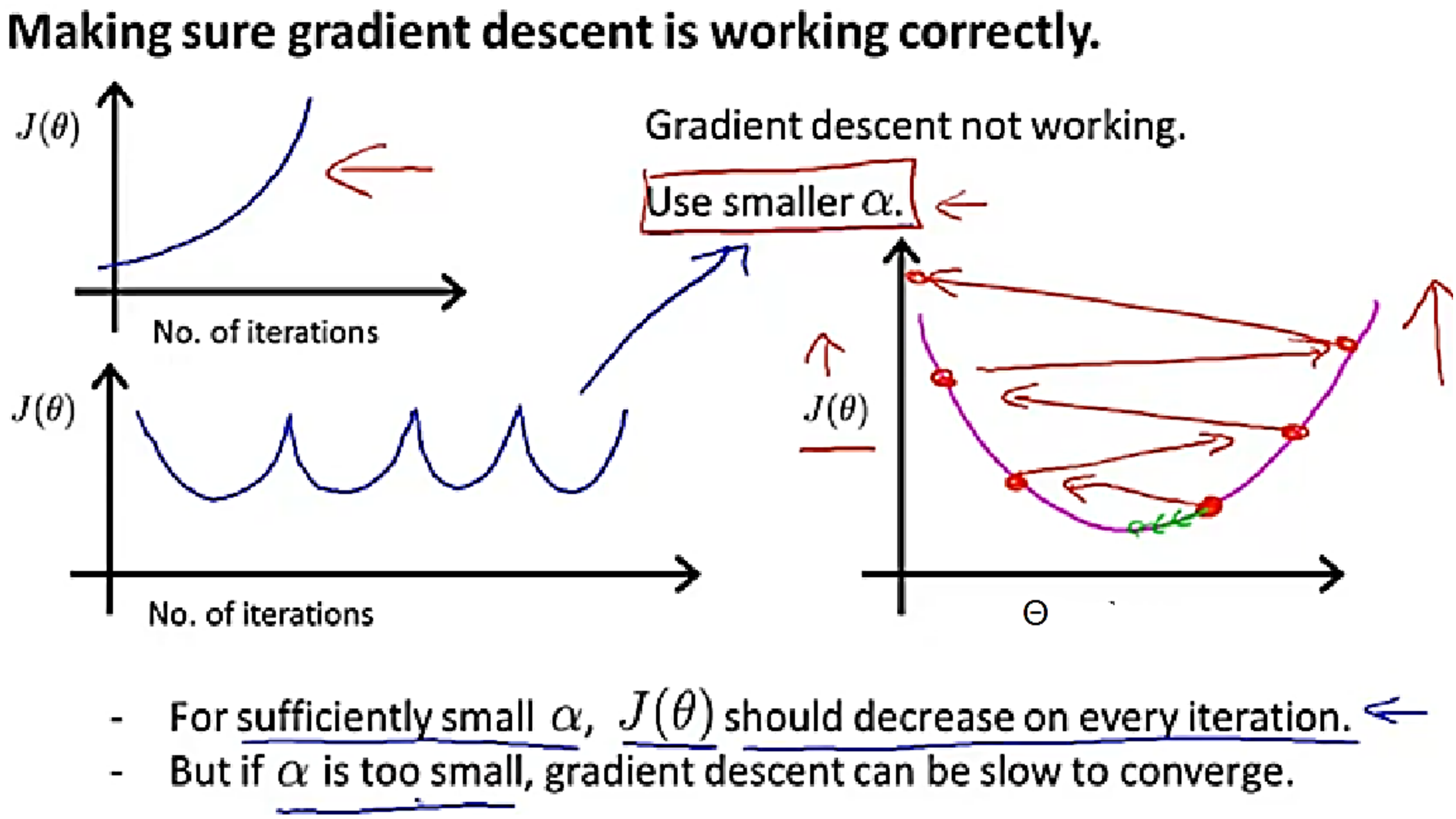
Plot **J vs no of iterations**, (not J vs Θ): J should decrease after every iteration. In this curve, J(Θ) is the vertical height of that point.

After a time, the **curve flattens** – denoting the convergence has occurred.

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

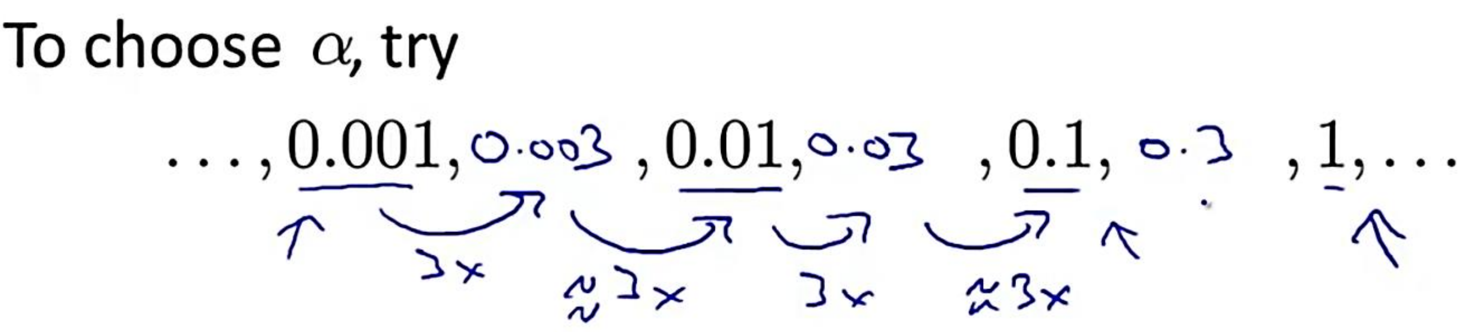
ε

If J(θ) ever increases, then you probably need to decrease **α**.



All these are **wrong curves** for J(Θ) vs iterations. Solution: use **smaller** values of **α** .

But not **too small α** as it **slows** the **convergence**. And not too large either: as it may not converge.



DEFINING NEW FEATURES:

Housing prices prediction

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

<u>Area</u>

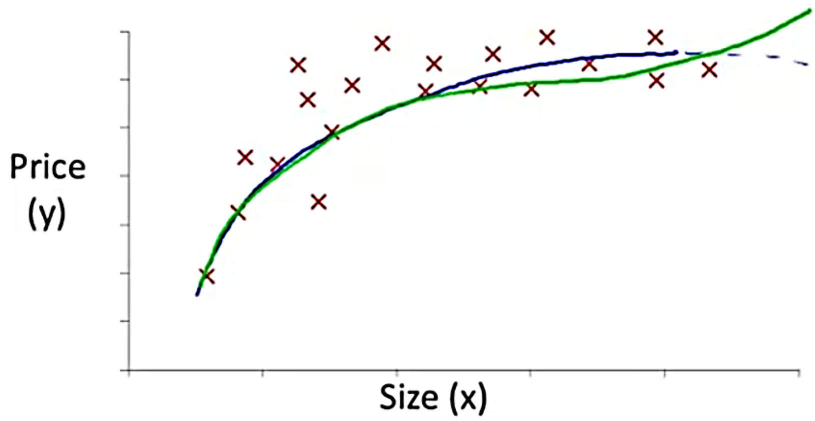
$$x = \underline{frontage * depth}$$

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

land area

POLYNOMIAL REGRESSION: Non-linear hypothesis

Polynomial regression



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

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

➔ We can use different hypothesis equations for a single dataset. Whichever best fits logically.

- ➔ For a multivariate: we can convert all features into functions of each other:

We can **combine** multiple features into one. For example, we can combine x_1 and x_2 into a new feature x_3 by taking $x_1 \cdot x_2$.

Choice of features: We can convert our linear hypothesis into a non-linear one

For example, if our hypothesis function is $h_\theta(x) = \theta_0 + \theta_1 x_1$ then we can create additional features based on x_1, to get the quadratic function $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$ or the cubic function $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2 + \theta_3 x_1^3$

Size (x)

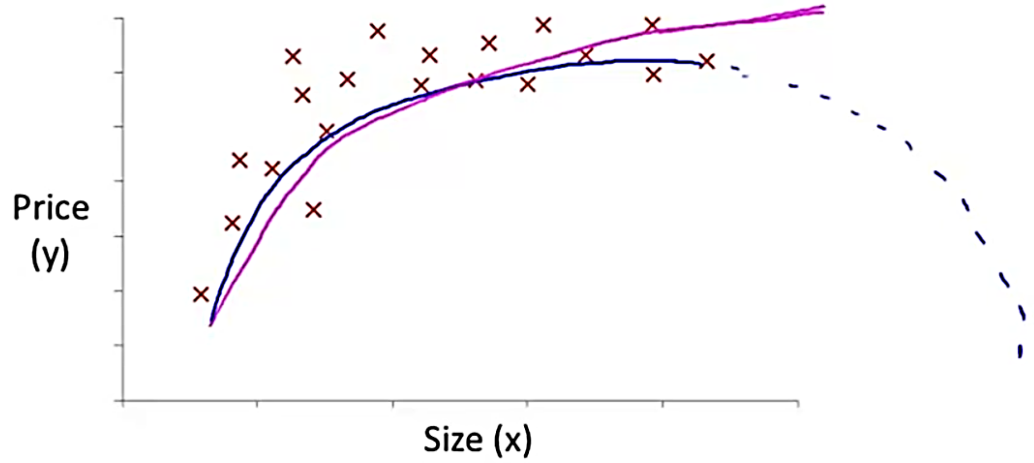
$$\boxed{h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3}$$
$$= \theta_0 + \theta_1(size) + \theta_2(size)^2 + \theta_3(size)^3$$

$\rightarrow x_1 = (size)$
$\rightarrow x_2 = (size)^2$
$\rightarrow x_3 = (size)^3$

size: 1 - 1000
size^2: 1 - 1000,000
size^3: 1 - 10^9

IMPORTANT: if you choose your features this way then feature scaling becomes very important.

eg. if x_1 has range 1 - 1000 then range of x_1^2 becomes 1 - 1000000 and that of x_1^3 becomes 1 - 1000000000



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

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

Thus, we find the **best fitting curve** for h(Θ).

Computing Parameters Analytically:

Up until now, we are using gradient descent Algorithm.. but now we will use new Algo: **NORMAL EQUATIONS**

⇨ **NORMAL EQUATIONS**: method to solve for Θ analytically… unlike grad desc, no need to iterate to minimize the J(Θ).. its minimized directly in one go.

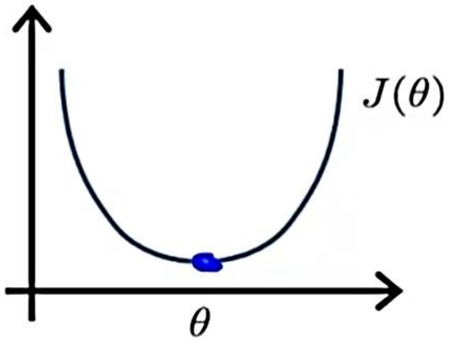
Intuition: for a **single parameter** Θ:

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

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

$$\frac{\partial}{\partial \theta} J(\theta) = \ldots \overset{\text{set}}{=} 0$$

Solve for θ



For **multiple parameters**:

Θ is a set of m Θ's.

⇨ For every Θ_i ➔ we set partial derivative of J wrt to Θ_i == 0

- Then we find Θ corresponding to that eqn
- This is done for each Θ

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

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

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

Examples: $m = 4.$

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	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

$$X = \underbrace{\begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}}_{m \times (n+1)} \qquad y = \underbrace{\begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}}_{m\text{-dimensional vector}}$$

m = number of example datas

n = no of features in input

n + 1 => we give an extra feature **x0=1** to every example.

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

⇨**To construct the X matrix from x_i vectors:**

- **Transpose** them and fill into the X matrix, Such that the x's belonging to a single example.. comes in row

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

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

m x (n+1)

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

m x 2

In above example : for all m training sets there are only 2 features x0 and x1.

Octave: `pinv (X'*X)*X'*y`

$pinv(X^T * X) * X^T * y$

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

$X' \qquad X^T$

~~Feature Scaling~~

$0 \le x_1 \le 1$

$0 \le x_2 \le 1000$

$0 \le x_3 \le 10^{-5}$

Andrew Ng

X' = transpose of X

⇨ **Feature scaling is not required in Normal Equations** method(algo)..Unlike in gradient desc => in which its req

WHEN TO USE GRAD DESC v/s NORMAL EQN:

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+1) x (n+1) $O(n^3)$
$n = 10^6$	• Slow if n is very large.
X is m x (n+1)	$n = 100$ $n = 1000$ $n = 10000$
O (kn^2)	O (n^3), need to calculate inverse of $X^T X$

When **no. of features is small** (upto 10^5) => use **normal eqns.**

As for **large value of n**=> X' * X will be a n x n matrix: and we have to find its inverse:

Inverse is of complexity $O(n^3)$ => thus for **large no. of input features**, **grad desc** is better way to converge to minima.

NON INVERTIBILITY PROBLEM IN NORMAL EQN METHOD:

Sometimes X' * X is not inventible (singular/degenerate)..:

REASONS:

Redundant features – two columns or rows are proportional in the X' * X matrix. (i.e. they are linearly dependent)

- ⇨**SOLUTION**: delete one of the dependent features.

Too many features – the number of features is too large as compares to no of examples… (m << n)

- ⇨**SOLUTION**: delete some features or Use **REGULARIZATION TECHNIQUE.**

$OCTAVE: pinv(X' * X) * X' * y

- ⇨ this would still give the right value of Θ. (Even if X' * X is non invertible).
- ⇨pinv() is pseudo inverse
- ⇨inv() is just inverse