Machine Learning

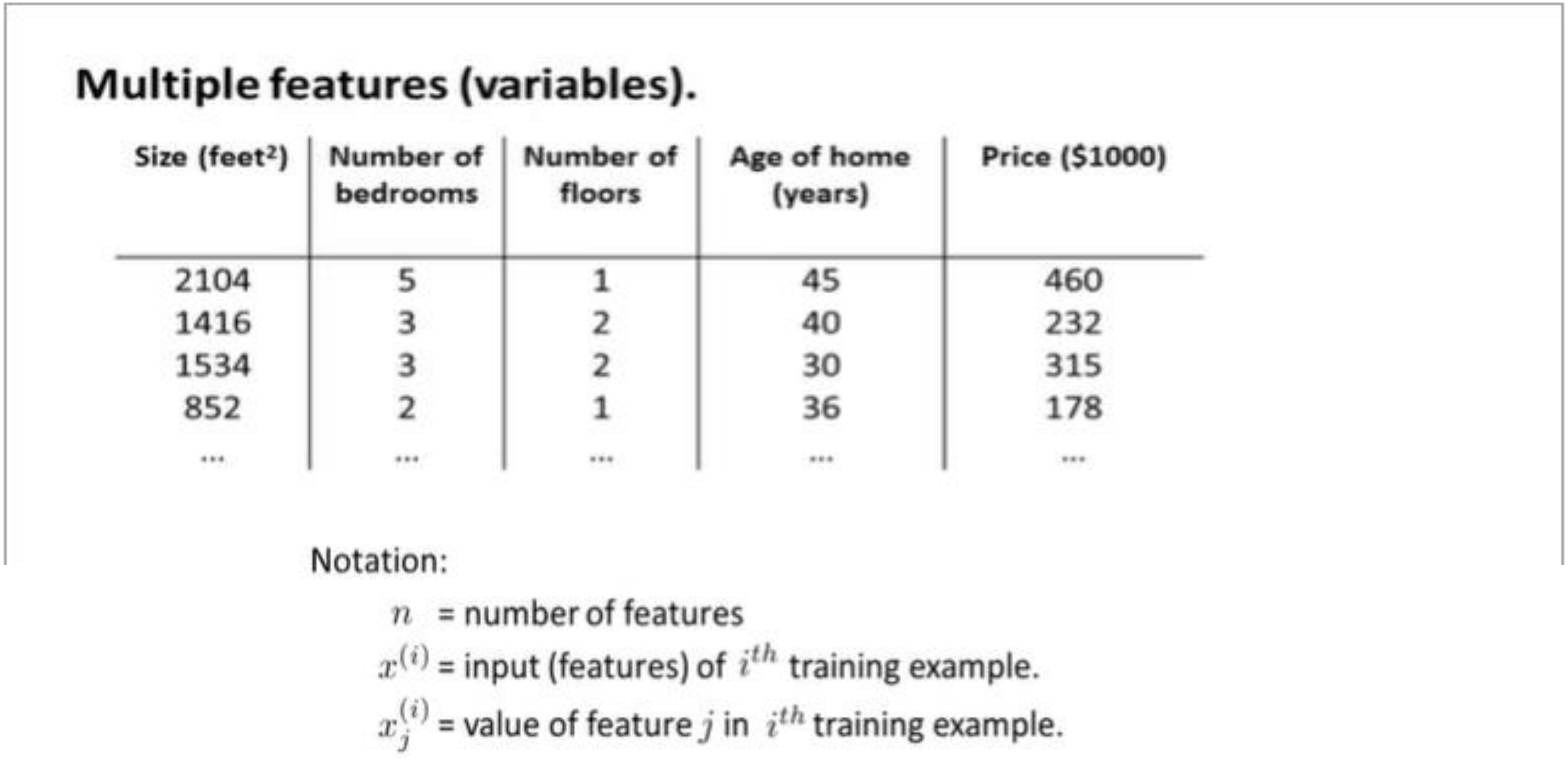
Linear Regression with Two Variables

Linear Regression with two variables

- Linear regression with multiple variables is also known as "multivariate linear regression".
- The multivariable form of the hypothesis function accommodating these multiple features is as follows:

Linear Regression with two variables

Multiple Features



Multiple features (variables).

Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

Notation:

n = number of features

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

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

Multivariate Linear Regression

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

For convenience of notation, define $x_0 = 1$.

Using the definition of matrix multiplication, our multivariable hypothesis function can be concisely represented as:

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

For convenience reasons, we assume as given as this allows us to do matrix multiplication with Theta and x:

$$x_0^{(i)} = 1 \text{ for } (i \in 1, \dots, m).$$

Gradient Descent for Multiple Variables

The gradient descent equation itself is generally the same form; we just have to repeat it for our 'n' features:

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

Parameters: $\theta_0, \theta_1, \ldots, \theta_n$

Cost function:

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

In other words:

```
repeat until convergence: {
    $\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$    for j := 0...n
}
```

All parameters (Qj) change to reduce j (cost function)

Gradient Descent for Multiple Variables

Gradient Descent

Previously (n=1):

Repeat $\{$

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

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

(simultaneously update θ_0, θ_1)

$\}$

New algorithm $(n \geq 1)$:

Repeat $\{$

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

(simultaneously update θ_j

$j = 0, \ldots, n$)

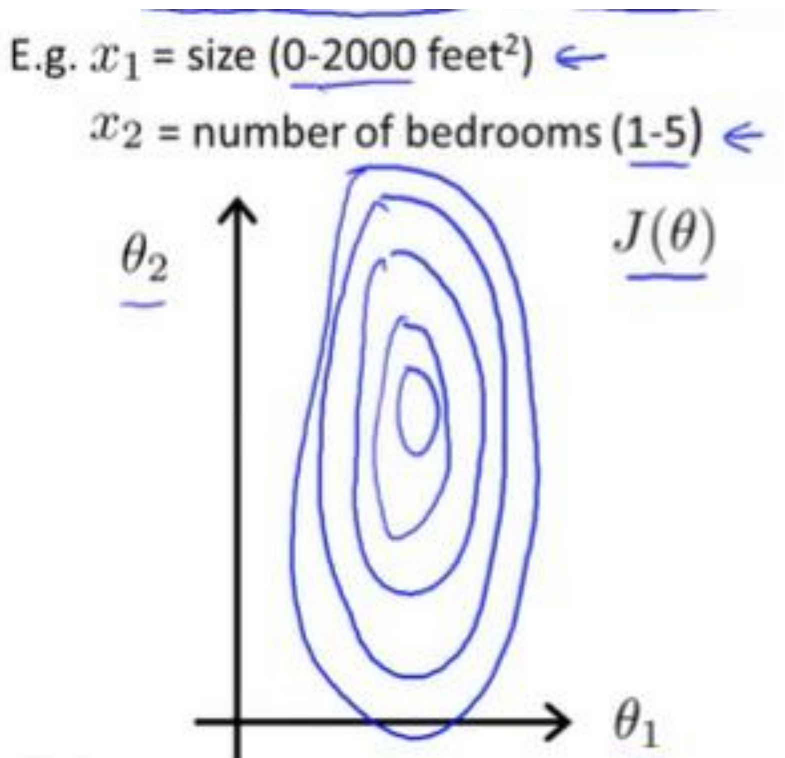
$\}$

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

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

Gradient Descent : Performance

- Make sure to have features on a similar scale, ie, features take similar ranges of values so that GD converges more quickly.
- If one plots the contours of cost J of Theta, they will take very skewed elliptical shape. On this GD may take long time and oscillate back and forth to reach Global minimum

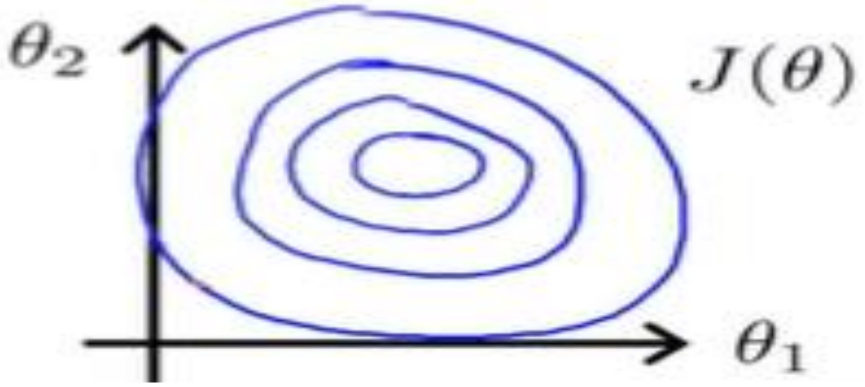

E.g. x_1 = size (0-2000 feet^2)
x_2 = number of bedrooms (1-5)
θ_2
$J(\theta)$
θ_1

Gradient Descent : Performance

- If one plots the contours of cos J of Theta now, they will be much less skewed and look like circles. On this GD will find direct path or less complicated trajectory and take less time and converge quickly to reach Global minimum

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

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



Gradient Descent - Performance

- We can speed up gradient descent by having each of our input values in roughly the same range. This is because 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.
- The way to prevent this is to modify the ranges of our input variables so that they are all roughly the same. Ranges like 0 to 3 or -2 to 1.5 is ok. But ideally, the following rule applies:

$$-1 \leq x_{(i)} \leq 1$$

or

$$-0.5 \leq x_{(i)} \leq 0.5$$

Gradient Descent - Performance

Two techniques to help with this are **feature scaling** and **mean normalization**.

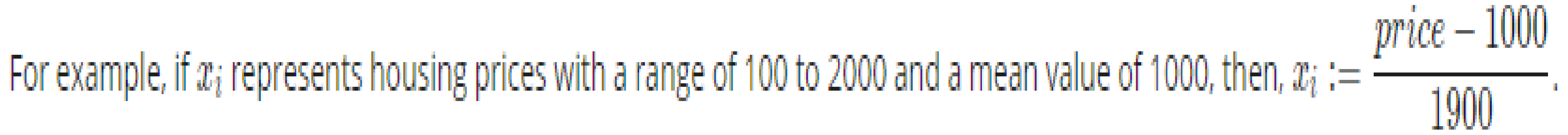
Feature scaling involves dividing the input values by the range (i.e. the maximum value minus the minimum value) of the input variable, resulting in a new range of just 1.

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

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.

Gradient Descent - Performance

Mean normalization involves subtracting the average value for an input variable from the values for that input variable resulting in a new average value for the input variable of just zero. To implement both of these techniques, adjust your input values as shown in this formula:



For example, if x_i represents housing prices with a range of 100 to 2000 and a mean value of 1000, then, $x_i := \frac{price - 1000}{1900}$.

Mean Normalisation

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

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

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

$$-0.5 \le x_1 \le 0.5, -0.5 \le x_2 \le 0.5$$

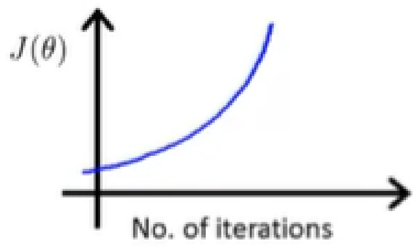
Gradient Descent Performance:Value of Alpha

- How to debug to know GD is working properly?
- How to set the value of learning rate alpha?

Gradient Descent Performance:Value of Alpha

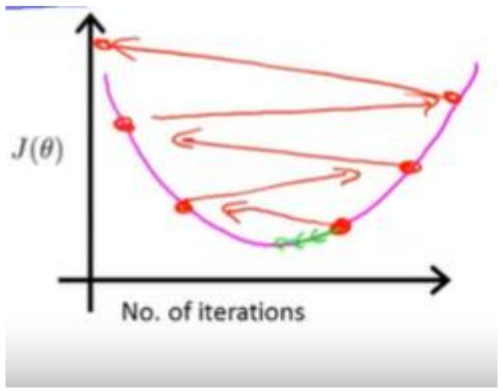
- As GD runs, one can plot cost functions.
- X axis is no of iterations
- Evaluate cost function J after some iterations
- It must be less than the previous value, if GD is working properly
- It also tells us if the curve flattens, which means cost function is not changing
- For different applications, there may be different iterations to converge, may be 200 iterations for one application, may be 2000 or 20000 for other applications.

Gradient Descent Performance:Value of Alpha

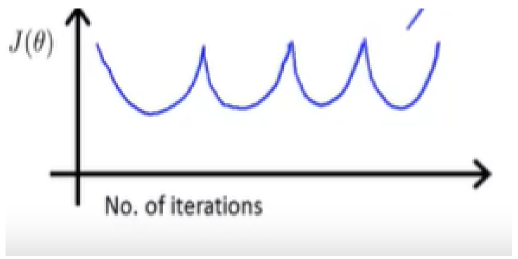


Gradient descent not working.

Use smaller α.

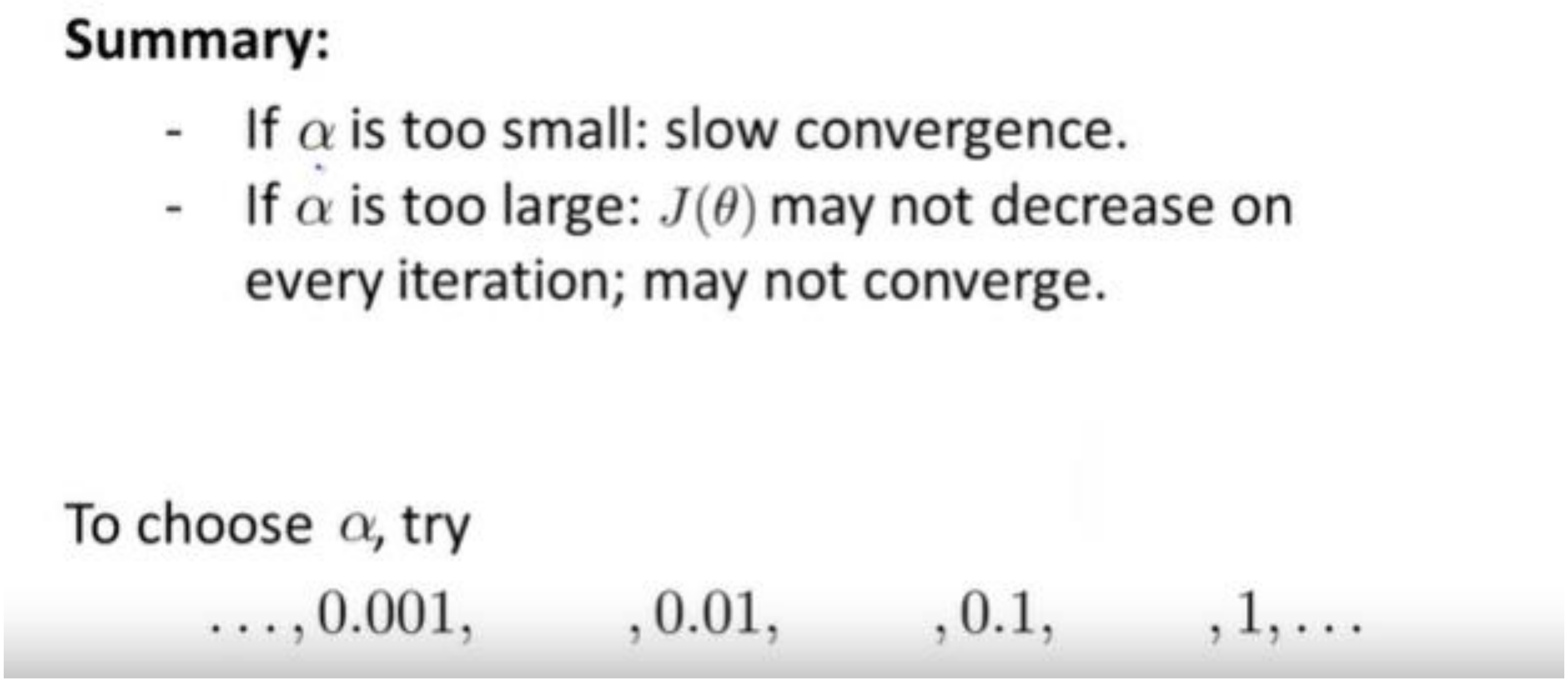


- Learning rate big
- GD overshoots the minimum, sends to back and forth the curve



- Learning rate big
- SO use smaller value of Learning Rate

Conclusion



Summary:

- If α is too small: slow convergence.
- If α is too large: $J(\theta)$ may not decrease on every iteration; may not converge.

To choose α, try

$$\ldots, 0.001, \qquad , 0.01, \qquad , 0.1, \qquad , 1, \ldots$$

Polynomial Regression

- We can improve our features and the form of our hypothesis function in a couple of different ways.
- 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 $x_1 \cdot x_2$.

Housing prices prediction

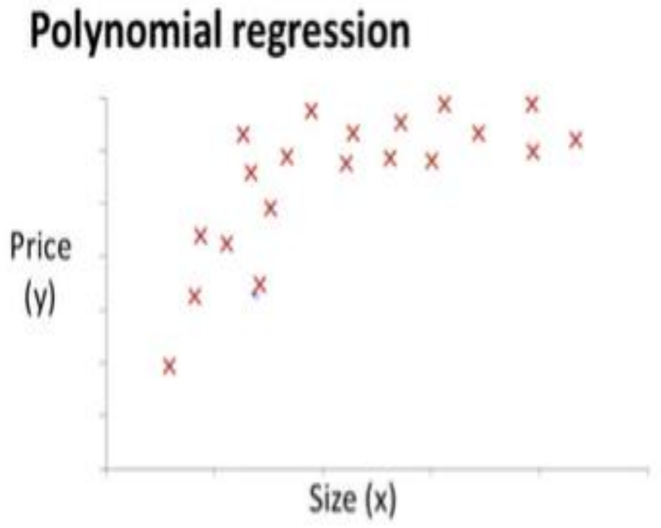
$$h_\theta(x) = \theta_0 + \theta_1 \times frontage + \theta_2 \times depth$$

- Lets frontage be x1 and depth x2, then what really determines the house is the area, so add a new feature area, now my hypothesis uses just one feature x1 (which is area). This might get a better model.

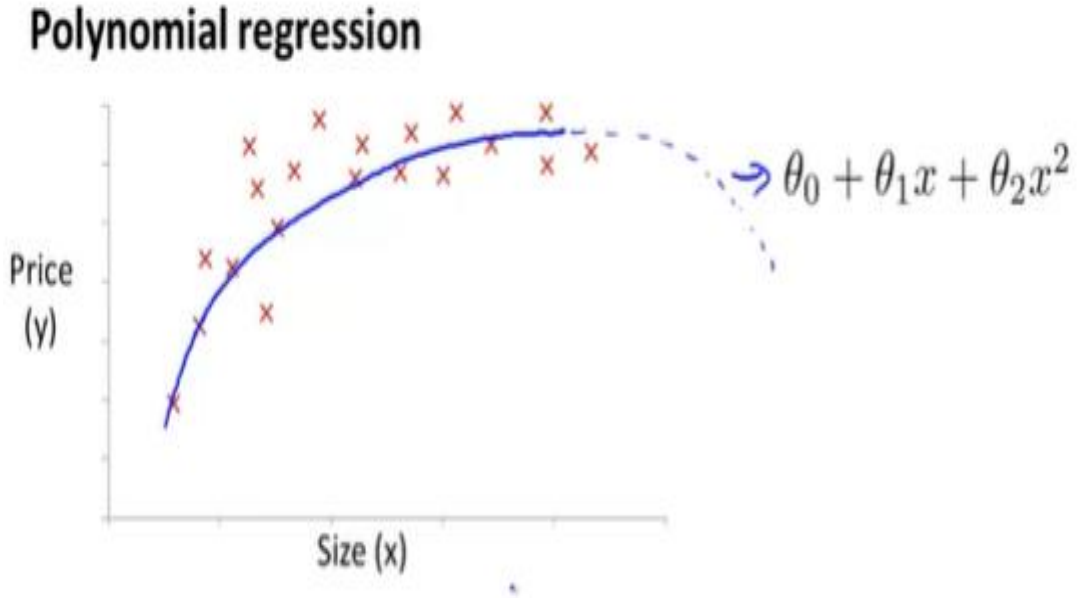
Polynomial Regression

- Our hypothesis function need not be linear (a straight line) if that does not fit the data well.
- We can **change the behavior or curve** of our hypothesis function by making it a quadratic, cubic or square root function (or any other form).

Polynomial Regression

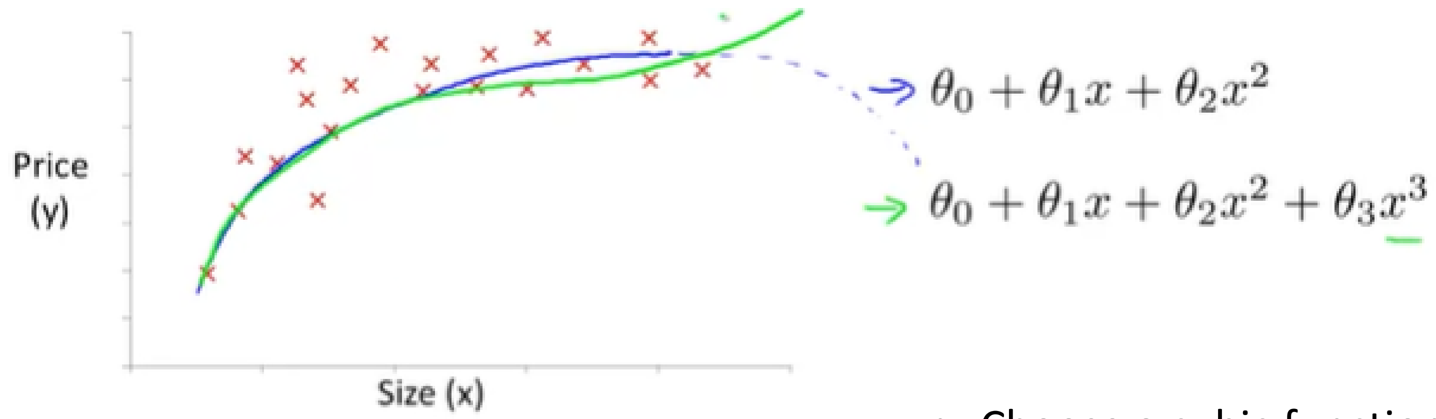


$\theta_0 + \theta_1 x + \theta_2 x^2$



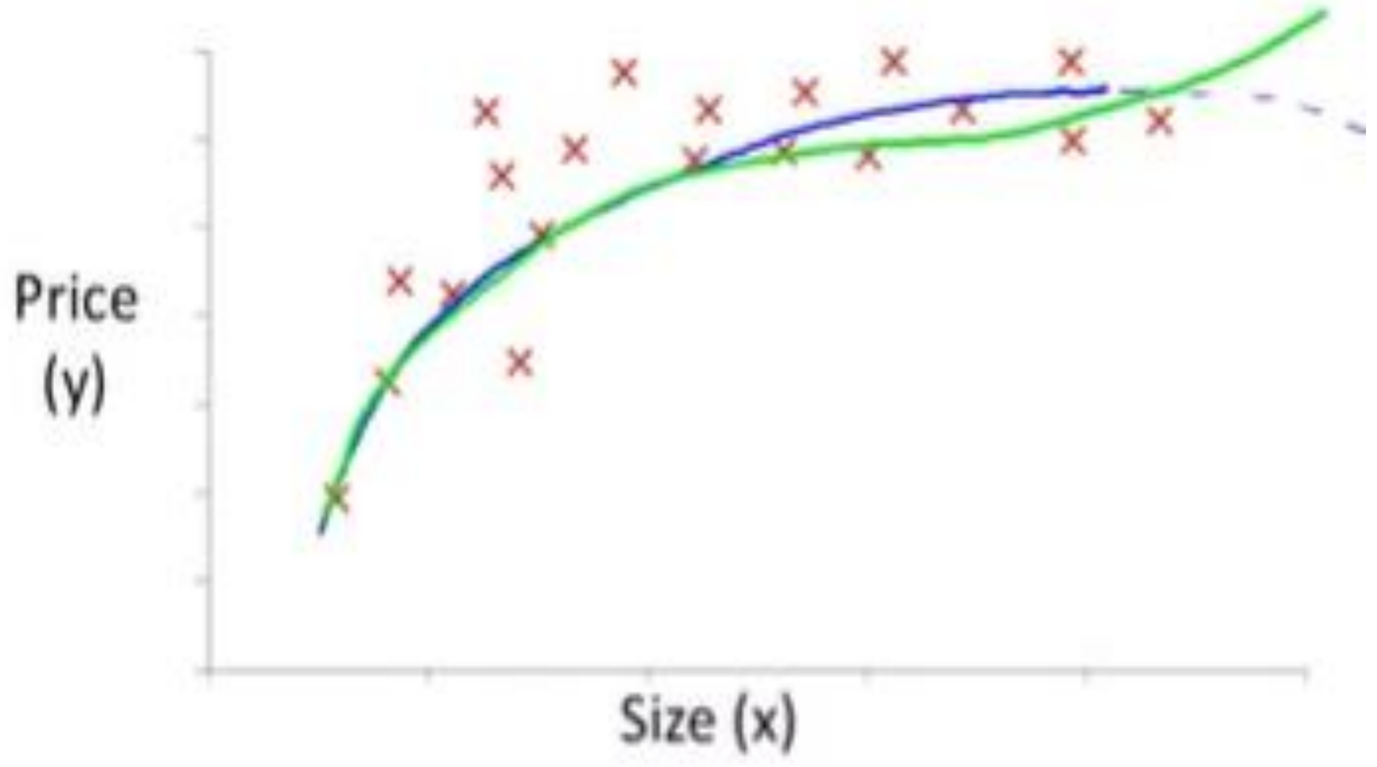
- Prizes go down as size increases
- As quadric function finally goes down

Polynomial Regression



- Choose a cubic function where the curve is always in the increasing form
- with a third term
- The green line fits better
- Theta values determine the best curve, tries to fit near all the points

Polynomial Regression



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

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

Polynomial Regression

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

In the cubic version, we have created new features x_2 and x_3 where $x_2 = x_1^2$ and $x_3 = x_1^3$.

To make it a square root function, we could do: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1}$

Polynomial Regression

Here Feature scaling becomes very important

- Size   1-100
- Size^2 1- 10000
- Size^3 1- 100000000

Computing Parameters Analytically : Normal Equation

- In contrast to Gradient Descent, Normal Equation is a method to solve for Theta analytically.
- No iteration required.
- In one step you reach global minimum
- In the "Normal Equation" method, we will minimize J by explicitly taking its derivatives with respect to the θj 's, and setting them to zero.
- This allows us to find the optimum theta without iteration. The normal equation formula is given below:

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

Normal Equation

Examples: $m = 4$.

Size (feet²) 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

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

m by (n+1) - dimensional matrix, and

a m-dimensional vector

What is normal equation?

- **Normal Equation**. Given a matrix **equation**, the **normal equation** is that which minimizes the sum of the square differences between the left and right sides.
- **Normal Equation** is an analytical approach to **Linear Regression** with a Least Square Cost Function. We can directly find out the value of θ without using Gradient Descent.
- Following this approach is an effective and a time-saving option when are working with a dataset with small features.

Normal Equation

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

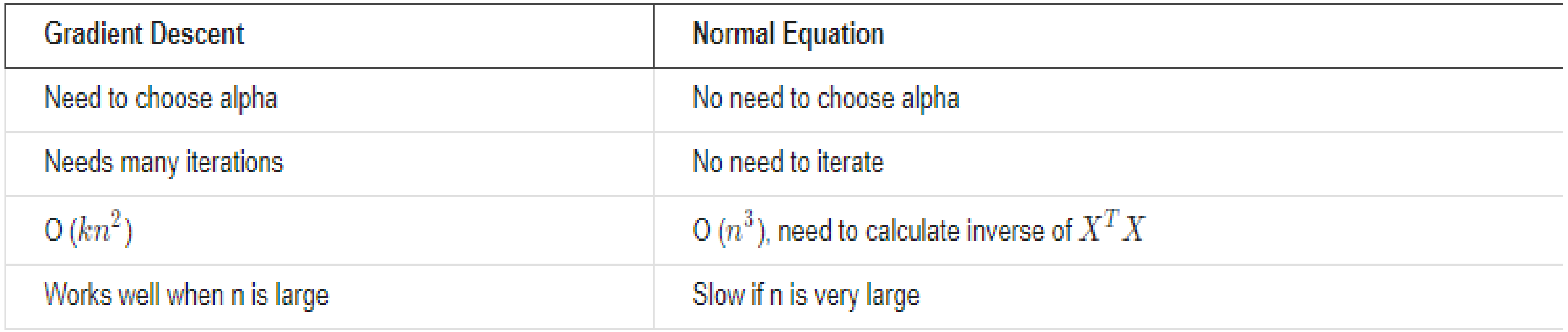
$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1}$$

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

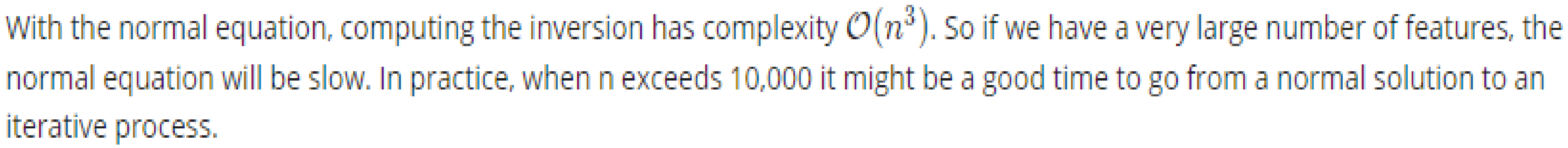
$(X^T X)^{-1}$ is inverse of matrix $X^T X$.

Normal Equation

- There is **no need** to do feature scaling with the normal equation.
- The following is a comparison of gradient descent and the normal equation:



Gradient Descent	Normal Equation
Need to choose alpha	No need to choose alpha
Needs many iterations	No need to iterate
O (kn^2)	O (n^3), need to calculate inverse of X^TX
Works well when n is large	Slow if n is very large



With the normal equation, computing the inversion has complexity $\mathcal{O}(n^3)$. So if we have a very large number of features, the normal equation will be slow. In practice, when n exceeds 10,000 it might be a good time to go from a normal solution to an iterative process.