Week 2

Table of Contents

Multivariate Linear Regression

Multiple Features

Linear regression with multiple variables is also known as "multivariate linear regression".

We now introduce notation for equations where we can have any number of input variables:

Multiple features (variables).

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
...	...	...	...	...

$m = 47$

Notation:

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

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

$x_3^{(2)} = 2$

m = the number of training examples.

The multivariable form of the hypothesis function accommodating these multiple features is as follows:.

$$h\theta(x) = \theta 0 + \theta 1x1 + \theta 2x2 + \theta 3x3 + \cdots + \theta nxn$$

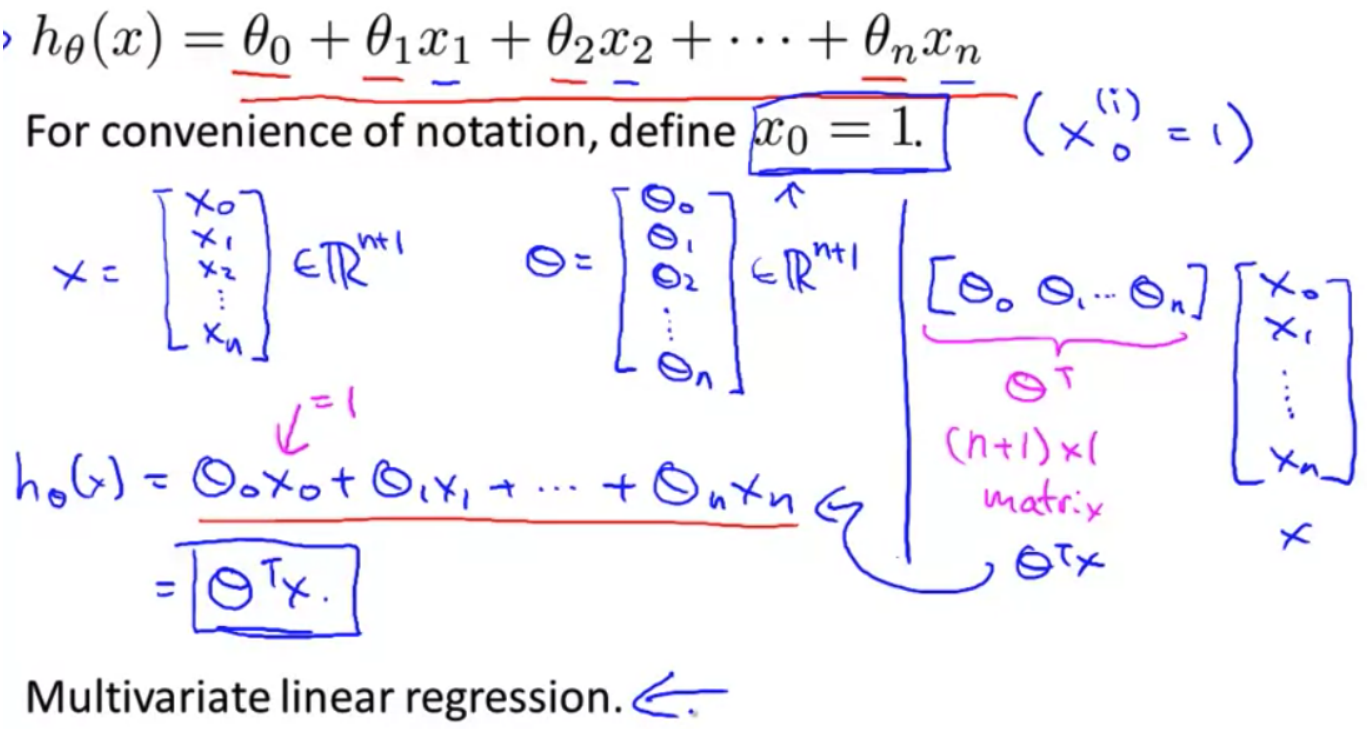
In order to develop intution about this function, we can think about θ0 as the basic price of a house, θ2 as the price per floor, etc. x1 will be the number of square meters in the house, x2 the number of floors, etc.

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

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

This is a vectorization of our hypothesis function for one training example; see the lessons on vectorization to learn more.

Remark: Note that for convenience reasons in this course we assume $x_0^{(i)} = 1$ for $(i \in 1, \dots, m)$. This allows us to do matrix operations with theta and x. Hence making the two vectors 'θ' and $x^{(i)}$ match each other element-wise (that is, have the same number of elements: n+1).]



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:

repeat until convergence: {

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

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

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

...

}

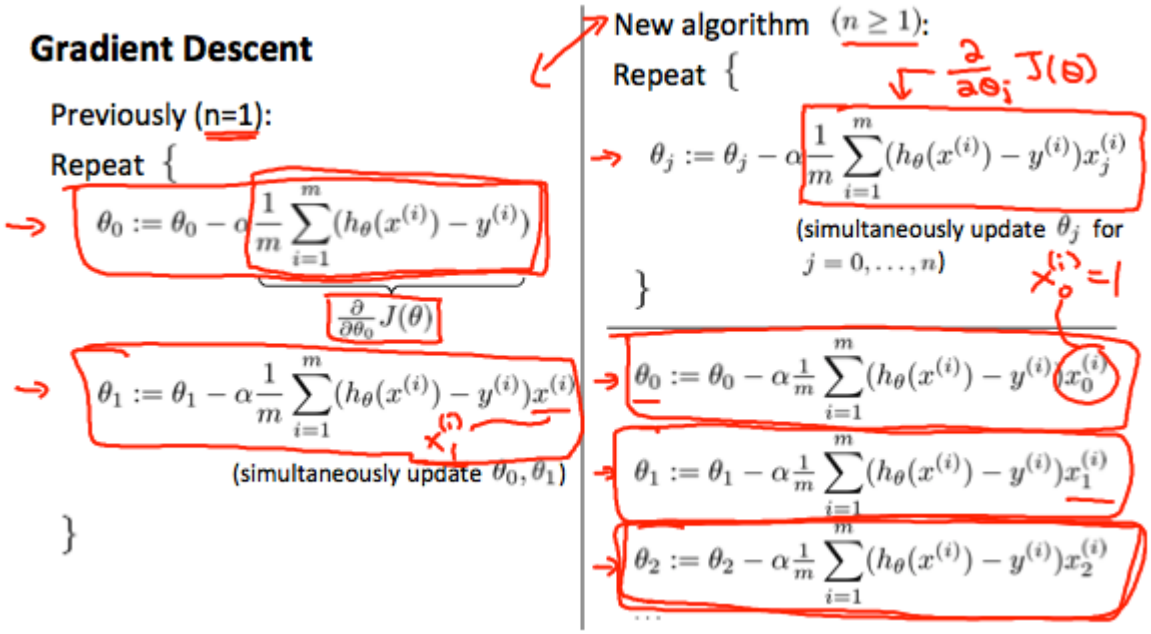
In other words:

repeat until convergence: {

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

}

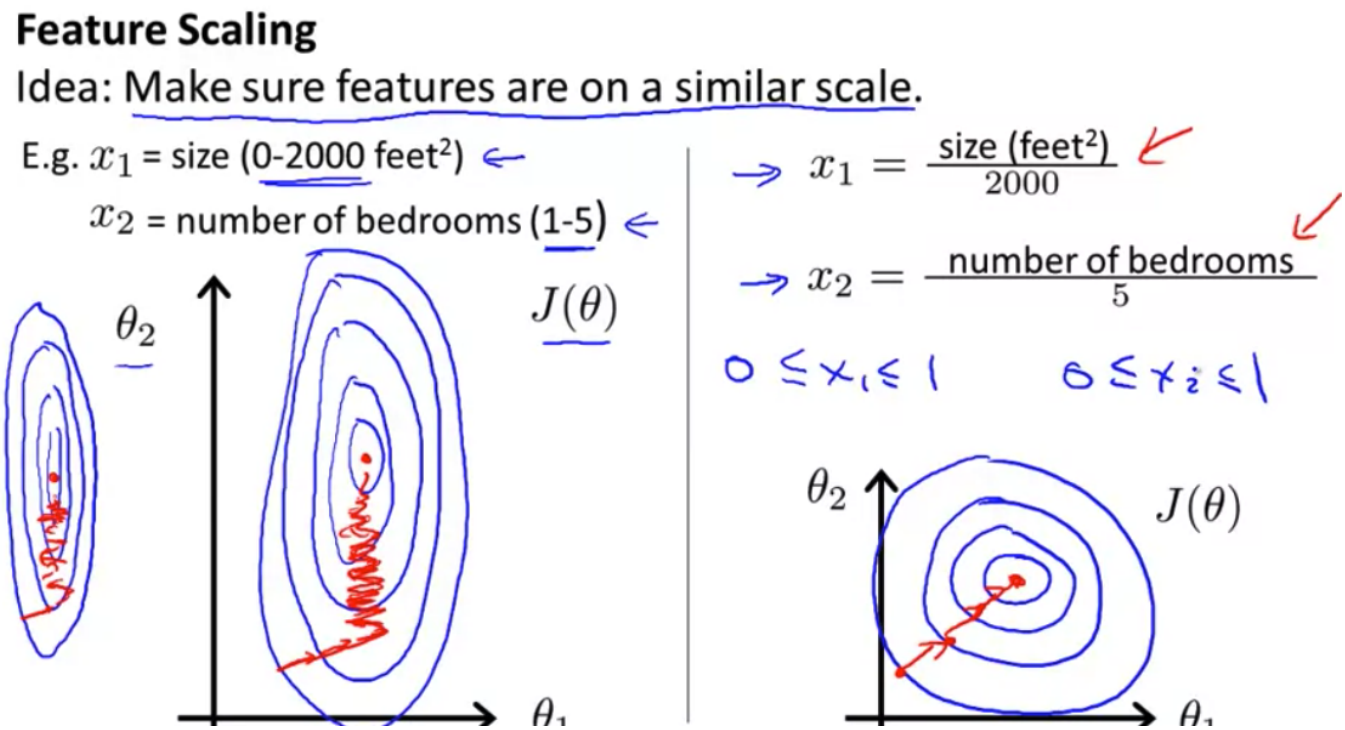
The following image compares gradient descent with one variable to gradient descent with multiple variables:



Gradient Descent in Practice I - Feature Scaling

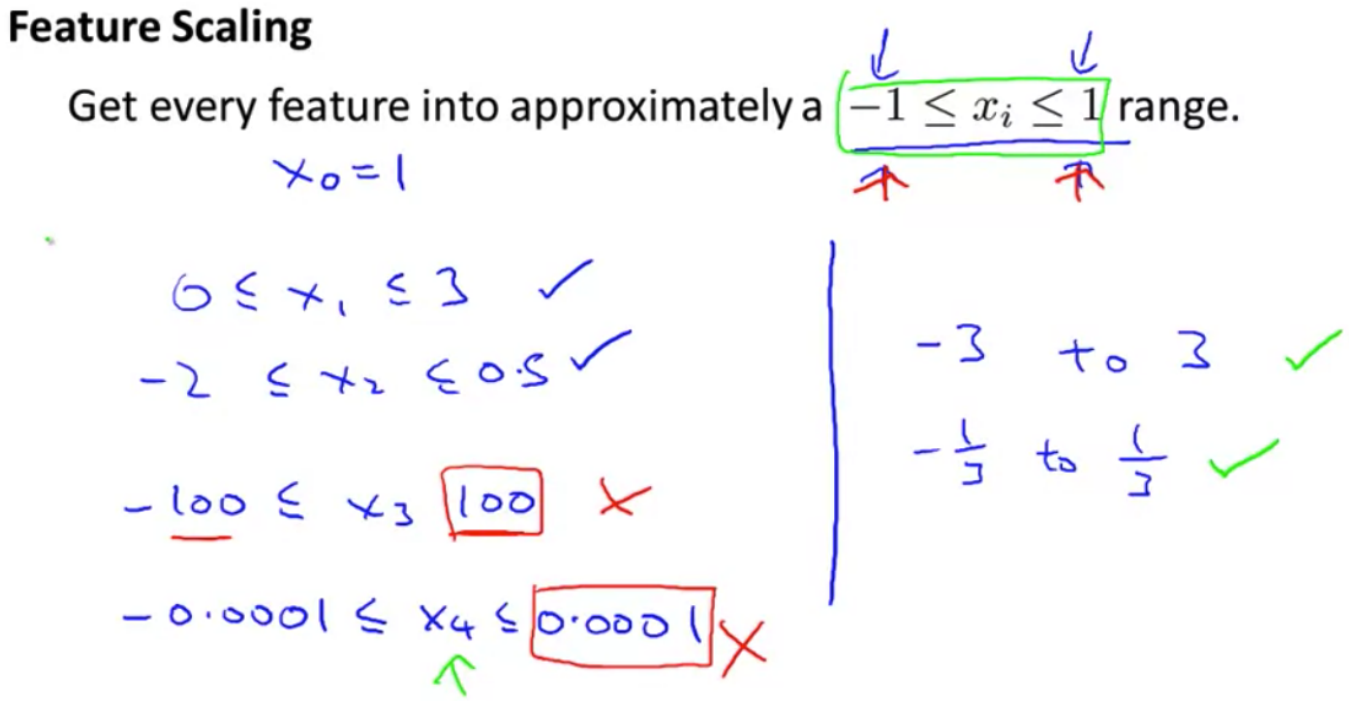
We can speed up gradient descent by having each of our input values in roughly the same range. This is because θ will descend quickly on small ranges and slowly on large ranges, and so will oscillate efficiently down to the optimum when the variables so that they are very uneven.

The Idea of Feature Scaling: Make sure features are on a similar scale.



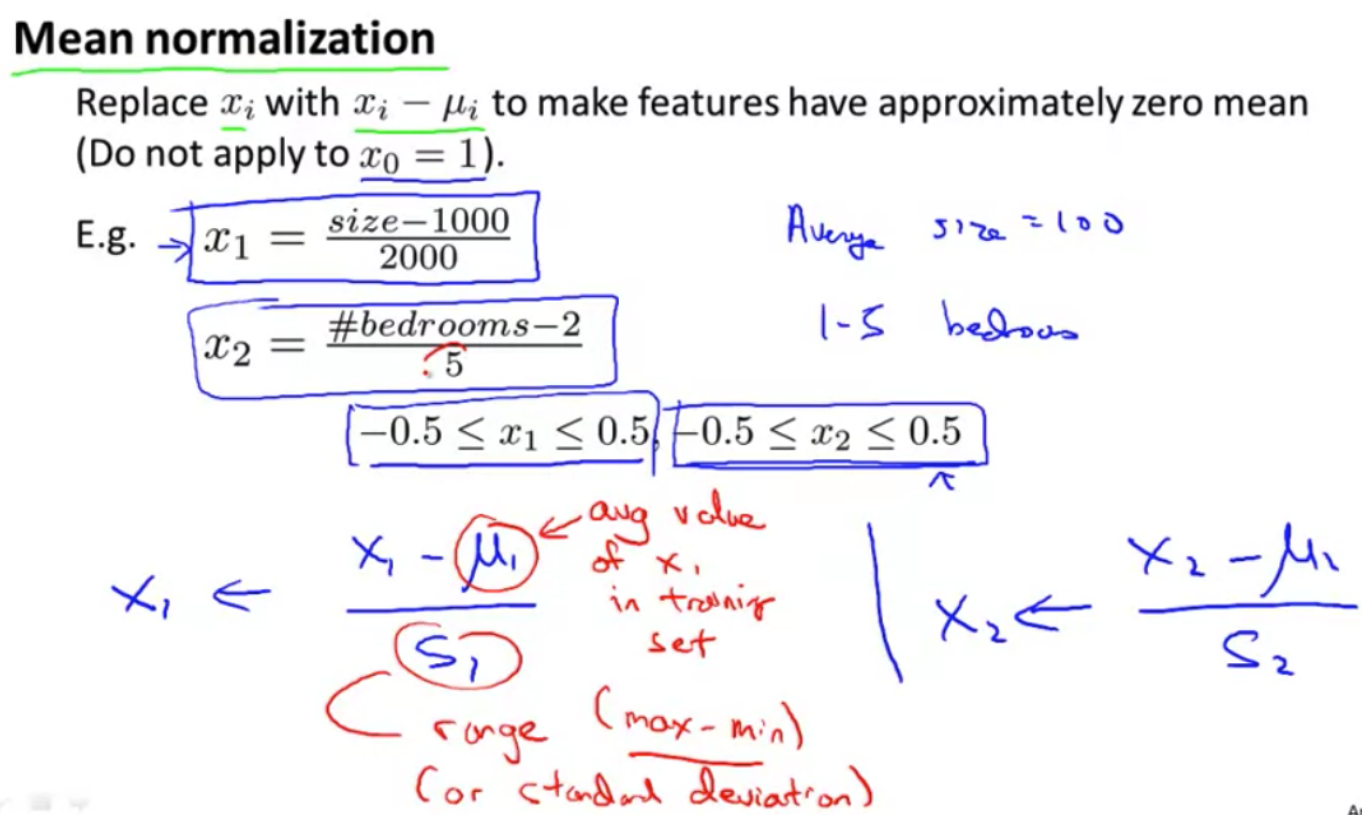
Feature Scaling:

Get every feature into approximately a -1 ≤ xi ≤ 1 range.



small range difference is acceptable

Mean normalization



The goal is to get all input variables into roughly one of these ranges, give or take a few.

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

Feature scaling involves dividing the input values by the range of input variable, resulting in a new range of just 1.

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.

> 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. Ideally:
>
> $-1 \leq x_{(i)} \leq 1$
>
> or
>
> $-0.5 \leq x_{(i)} \leq 0.5$
>
> These aren't exact requirements; we are only trying to speed things up. The goal is to get all input variables into roughly one of these ranges, give or take a few.
>
> 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. 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:
>
> $$x_i := \frac{x_i - \mu_i}{s_i}$$
>
> 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.
>
> Note that dividing by the range, or dividing by the standard deviation, give different results. The quizzes in this course use range - the programming exercises use standard deviation.
>
> For example, if x_i represents housing prices with a range of 100 to 2000 and a mean value of 1000, then, $x_i := \frac{price - 1000}{1900}$.

Purpose:

Make gradient run much faster, and converge in a lot fewer on interactions.

Gradient Descent in Practice II - Learning Rate

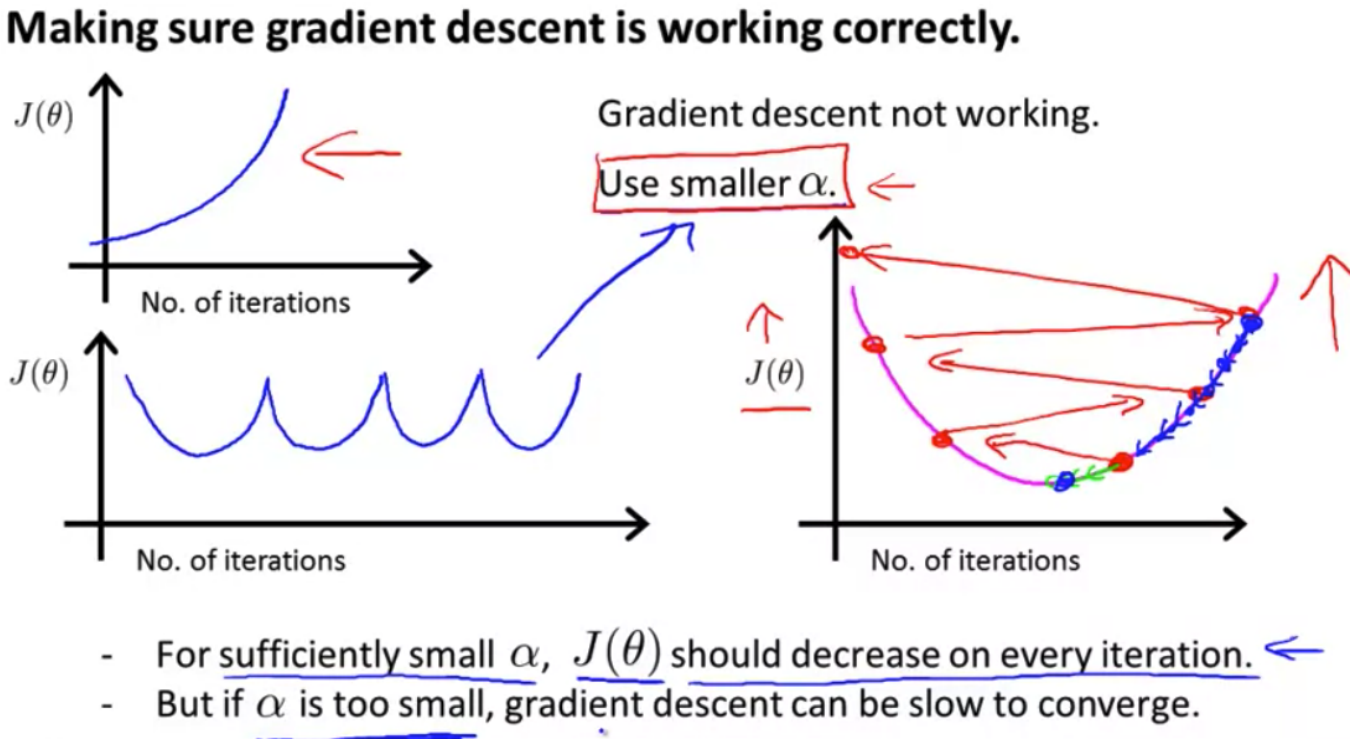
center around the learning rate alpha.

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

  Make a plot with number of iterations on the x-axis. Now plot the cost function, J(θ) over the number of iterations of gradient descent. If J(θ) ever increases, then you probably need to decrease a.

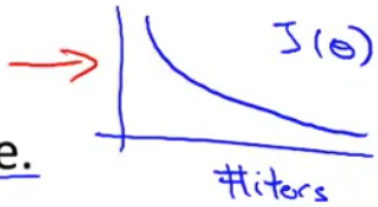
- How to choose learning rate alpha.

  Automatic convergence test. Declare convergence if j(θ) decreases by less than E in one iteration, where E is some small value as 10 ^ -3. However in practice it's difficult to choose this threshold value.



Summary:

Summary:



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

To choose α, try

$$\ldots, 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, \ldots$$

Features and polynomial regression

We can improve our features and the form of our hypothesis function in a couple different ways.

Purpose:

1. The choice of features that you have and how you can get different learning algorithm.
2. Polynomial regression: to fit very complicated, even non-linear functions.

Exp: **Housing prices prediction:**

two features: frontage and depth.

Particular problem depending on what insight you might have. We can combine multiple features into one.

Housing prices prediction

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

Area

x = frontage * depth

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

↑ land area

use frontage and depth or just one feature x = frontage * depth.

Polynomial regression:

Our hypothesis function need not be linear if that does not fit the data well.

We can change the behavior of curve of our hypothesis function by making it a quadratic, cubic or square root function.

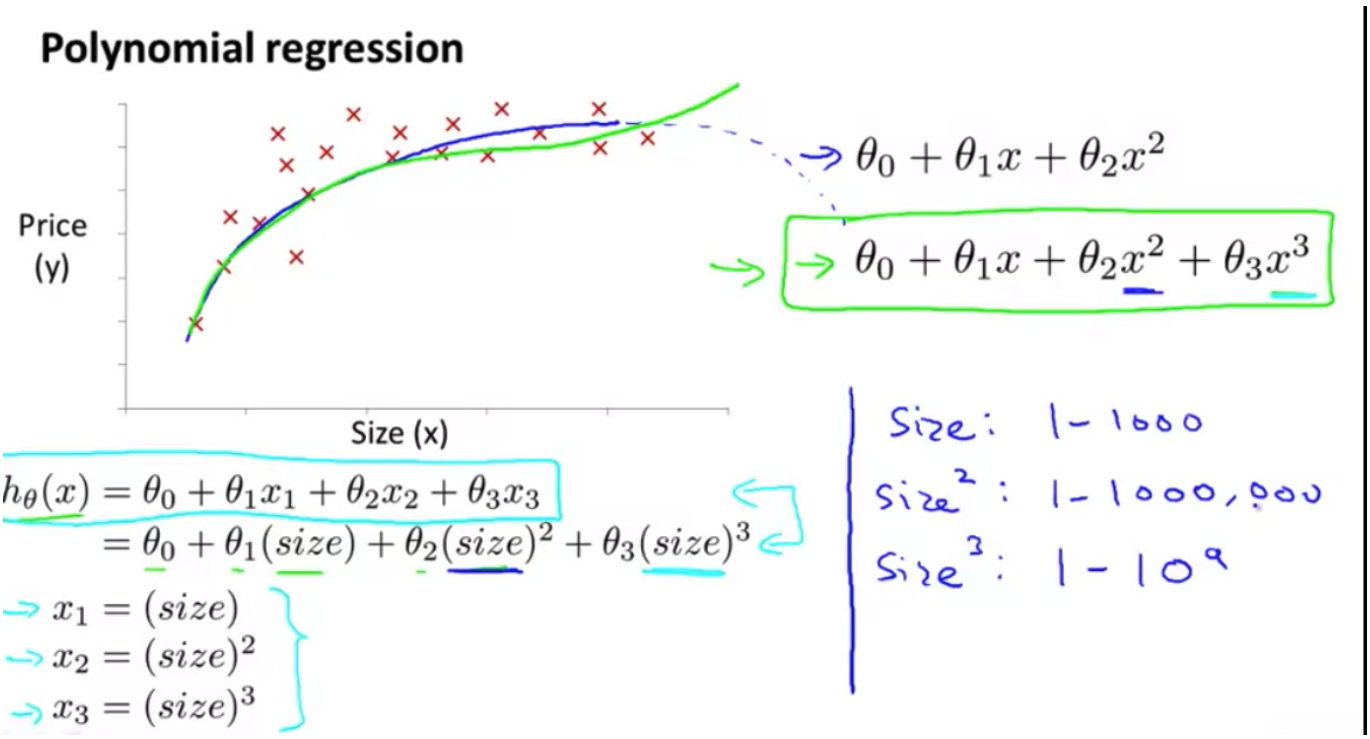
For example, if our hypothesis function is $h_\theta(x) = \theta_0 + \theta_1 x_1$ then we can create additional features based on x_1, to get the quadratic function $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$ or the cubic function $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2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

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: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1}$

One important thing to keep in mind is, if you choose your features this way then feature scaling becomes very important.

How do we actually fit a model to our data?



It's important to apply feature scaling if using gradient descent to get them into compareble ranges of values.

According to this section, you have a choice in what features to use, and by designing different features you can fit more complex functions your data then just fitting a straight line to the data and in particular you can put polynomial functions as well and sometimes by appropriate insight into the features simply get a much better model for your data.

Computing Parameters Analytically

Normal Equation

Normal Equation is another way of minimizing j.

A second way of doing so, this time performing the minimization explicitly and without resorting to an iterative algorithm.

In the "Normal Equation" method, we will minimize J by explicitly taking derivatives with respect to the the θj ’s, and setting them to zero. This allows us to find the optimum theta without iteration. The normal equationo formula is given below:

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

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

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \quad m \times (n+1)$$

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix} \quad m\text{-dimensional vector}$$

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

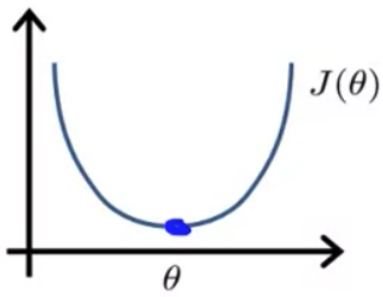
There is no need to do feature scaling with the normal equation.

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

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

$\frac{d}{d\theta} J(\theta) = \dots \overset{\text{set}}{=} 0$

Solve for θ



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

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

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

Andrew

The following is a comparison of gradient descent and 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 $X^T X$
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.

- Octave:

```
pinv(x' * x) * x' * y
```

Normal equation and non-invertibility(optional)

When implementing the normal equaton in octave we want use the 'pinv' function rather than 'inv'. The 'pinv' function will give you a value of θ even if $X^T * X$ is not invertible.

The reason $X^T * X$ is **noninvertible, the common cause might be having :**

- Redundant features, where two features are very closely related (i.e. they are linearly dependent)
- Too many features (e.g. m ≤ n). In this case, delete some features or use "regularization" (to be explained in a later lesson).

Consider about the non-invertible situation.

Normal equation

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

- What if $X^T X$ is non-invertible? (singular/ degenerate)
- Octave: `pinv(X'*X)*X'*y`

What if $X^T X$ is non-invertible?

- Redundant features (linearly dependent).
  E.g. x_1 = size in feet^2
  ~~x_2 = size in m^2~~
  $x_1 = (3.28)^2 x_2$
- Too many features (e.g. $m \leq n$).
  - Delete some features, or use regularization.

- delete some features.
- use regularization.

Review:

2. You run gradient descent for 15 iterations with $\alpha = 0.3$ and compute $J(\theta)$ after each iteration. You find that the value of $J(\theta)$ **decreases** quickly then levels off. Based on this, which of the following conclusions seems most plausible? 1 分

- ○ Rather than use the current value of α, it'd be more promising to try a larger value of α (say $\alpha = 1.0$).
- ○ Rather than use the current value of α, it'd be more promising to try a smaller value of α (say $\alpha = 0.1$).
- ◉ $\alpha = 0.3$ is an effective choice of learning rate.

3. Suppose you have $m = 23$ training examples with $n = 5$ features (excluding the additional all-ones feature for the intercept term, which you should add). The normal equation is $\theta = (X^T X)^{-1} X^T y$. For the given values of m and n, what are the dimensions of θ, X, and y in this equation? 1 分

- ○ X is 23×5, y is 23×1, θ is 5×5
- ○ X is 23×6, y is 23×6, θ is 6×6
- ◉ X is 23×6, y is 23×1, θ is 6×1
- ○ X is 23×5, y is 23×1, θ is 5×1

4. Suppose you have a dataset with $m = 50$ examples and $n = 200000$ features for each example. You want to use multivariate linear regression to fit the parameters θ to our data. Should you prefer gradient descent or the normal equation? 1 分

- ○ The normal equation, since it provides an efficient way to directly find the solution.
- ○ Gradient descent, since it will always converge to the optimal θ.
- ○ The normal equation, since gradient descent might be unable to find the optimal θ.
- ◉ Gradient descent, since $(X^T X)^{-1}$ will be very slow to compute in the normal equation.

5. Which of the following are reasons for using feature scaling? 1 分

- ☐ It prevents the matrix $X^T X$ (used in the normal equation) from being non-invertable (singular/degenerate).
- ☑ It speeds up gradient descent by making it require fewer iterations to get to a good solution.
- ☐ It is necessary to prevent gradient descent from getting stuck in local optima.
- ☐ It speeds up solving for θ using the normal equation.

补充：Normal Equation 公式的推导