Multiple Feature

- Single var:

Size (feet2) x	Price ($1000) y
2104	460
1416	232
1534	315
852	178

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

-

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

$x_3^{(2)} = 2$.

$x^{(1)}: \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 40 \end{bmatrix}$

n: num of features

$x^{(i)}$: input training examples (ith row) → n dimension.

$y^{(i)}$: value of feature j in ith training example.

Question

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

In the training set above, what is $x_1^{(4)}$?

- ○ The size (in feet^2) of the 1st home in the training set
- ○ The age (in years) of the 1st home in the training set
- ◉ The size (in feet^2) of the 4th home in the training set
- ○ The age (in years) of the 4th home in the training set

✓ Correct

Hypothesis:

One var: $h_\theta(x) = \theta_0 + \theta_1 x$.

mull:

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

eg: $h_\theta(x) = 80 + 0.1 x_1 + 0.01 x_2$.

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

For convience, define $x_0 = 1 \quad (x_0^{(i)} = 1)$

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

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n.$$

$$= \theta^T x.$$

$$\underbrace{[\theta_0, \theta_1 \cdots \theta_n]}_{\theta^T}$$

$(n+1) \times 1$ matrix.

Multivariate linear regression

Multiple Features

Note: [7:25 - θ^T is a 1 by (n+1) matrix and not an (n+1) by 1 matrix]

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.

$x_j^{(i)}$ = value of feature j in the i^{th} training example
$x^{(i)}$ = the input (features) of the i^{th} training example
m = the number of training examples
n = the number of features

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

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

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.

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

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

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

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

parameter: $\theta_0 \cdots \theta_n$ ($n+1$ dimension)

Cost function:

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

$J(\theta)$

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 $j = 0, \ldots, n$)

Question

When there are n features, we define the cost function as

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

For linear regression, which of the following are also equivalent and correct definitions of $J(\theta)$?

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

  ✓ Correct

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

  ✓ Correct

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

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

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 {

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

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

}

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

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

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

...

Gradient Descent For Multiple Variables

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: {

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

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

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

...

}

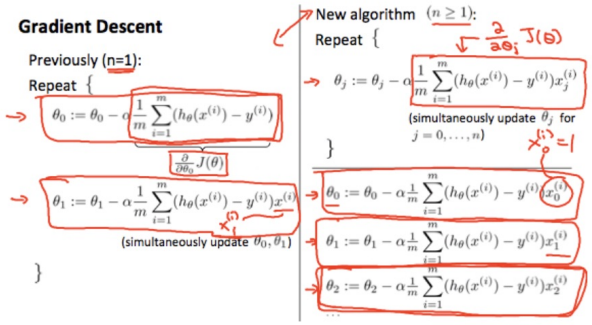
In other words:

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:



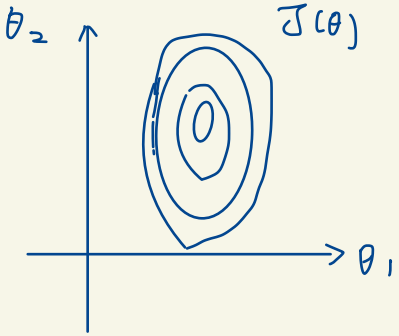
Feature Scaling

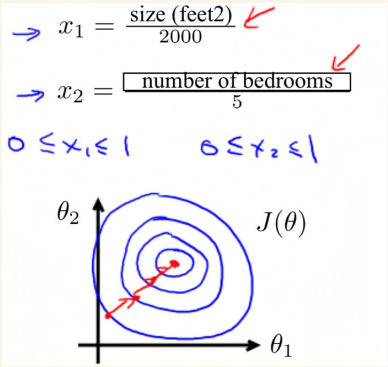
- Feature Scaling

Idea: Mk sure features are on a similar scale.

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

x_2 = num of bedrooms (1~5)





Feature Scaling

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

$x_0 = 1$

$0 \le x_1 \le 3$ ✓

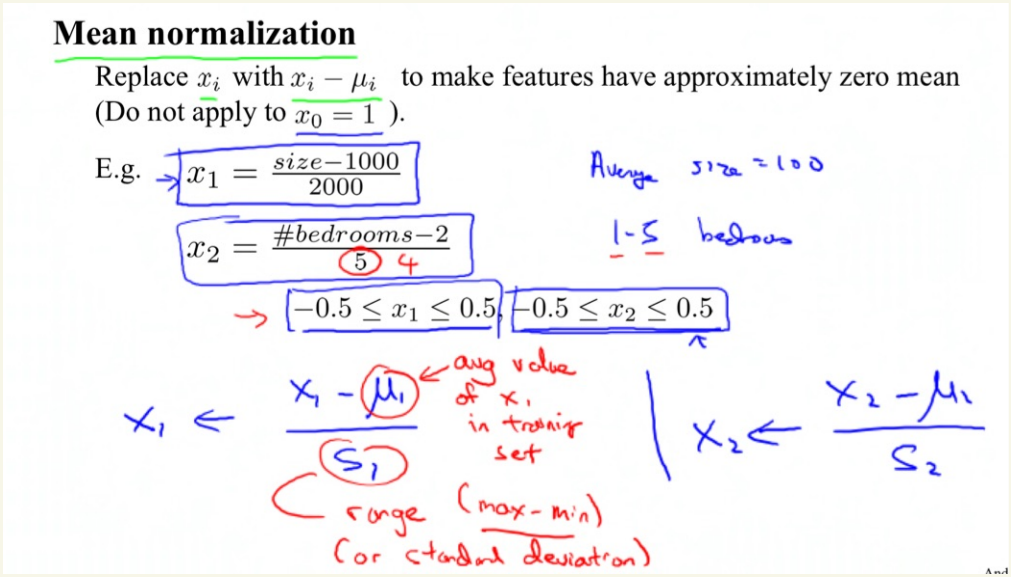
$-2 \le x_2 \le 0.5$ ✓

range too big: $-100 \le x_3 \le 100$ ✗

range too small: $-0.0001 \le x_4 \le 0.0001$ ✗

-3 to 3 ✓

$-\frac{1}{3}$ to $\frac{1}{3}$ ✓



Question

Suppose you are using a learning algorithm to estimate the price of houses in a city. You want one of your features x_i to capture the age of the house. In your training set, all of your houses have an age between 30 and 50 years, with an average age of 38 years. Which of the following would you use as features, assuming you use feature scaling and mean normalization?

- $x_i = \text{age of house}$
- $x_i = \frac{\text{age of house}}{50}$
- $x_i = \frac{\text{age of house} - 38}{50}$
- $x_i = \frac{\text{age of house} - 38}{20}$

Correct

Gradient Descent in Practice I - Feature Scaling

Note: [6:20 - The average size of a house is 1000 but 100 is accidentally written instead]

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, $x_i := \frac{price - 1000}{1900}$.

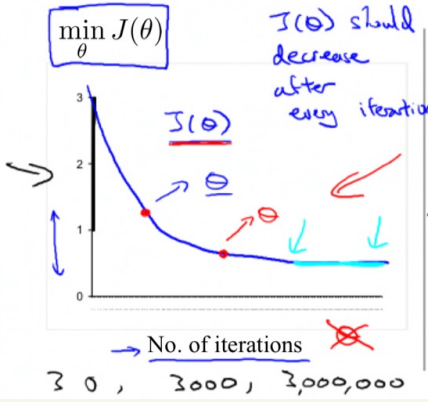
Learning Rate

Gradient descent

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

- "Debugging": How to make sure gradient descent is working correctly.
- How to choose learning rate α

Making sure gradient descent is working correctly.



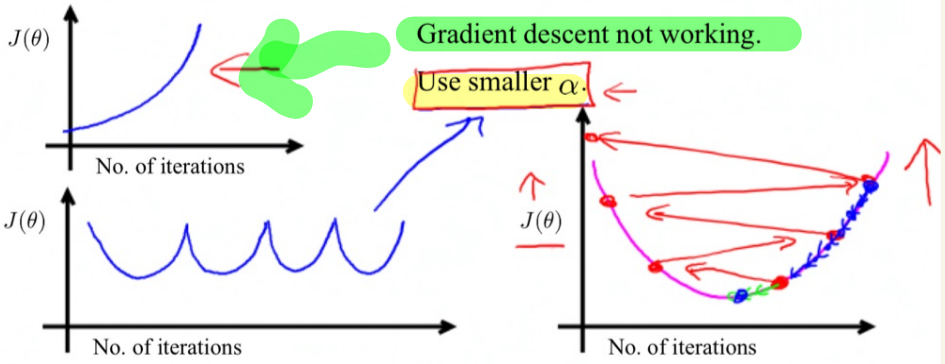
Example automatic convergence test:

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

10^{-3}
ε

And

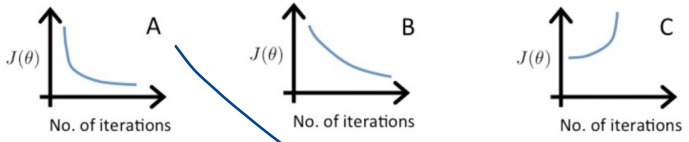
Making sure gradient descent is working correctly.



- For sufficiently small α, $J(\theta)$ should decrease on every iteration.
- But if α is too small, gradient descent can be slow to converge.

Question

Suppose a friend ran gradient descent three times, with $\alpha = 0.01$, $\alpha = 0.1$, and $\alpha = 1$, and got the following three plots (labeled A, B, and C):



Which plots corresponds to which values of α?

- A is $\alpha = 0.01$, B is $\alpha = 0.1$, C is $\alpha = 1$.
- (selected) A is $\alpha = 0.1$, B is $\alpha = 0.01$, C is $\alpha = 1$.
- A is $\alpha = 1$, B is $\alpha = 0.01$, C is $\alpha = 0.1$.
- A is $\alpha = 1$, B is $\alpha = 0.1$, C is $\alpha = 0.01$.

✓ **Correct**

In graph C, the cost function is increasing, so the learning rate is set too high. Both graphs A and B converge to an optimum of the cost function, but graph B does so very slowly, so its learning rate is set too low. Graph A lies between the two.

too small α $\Rightarrow$ too slow

too big α $\Rightarrow$ plot won't converge

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

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

(×3, ≈×3, ×3, ≈×3)

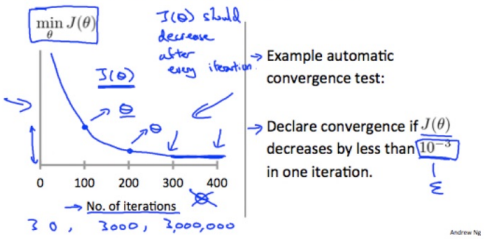
Gradient Descent in Practice II - Learning Rate

Note: [5:20 - the x -axis label in the right graph should be θ rather than No. of iterations]

Debugging gradient descent. 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 α.

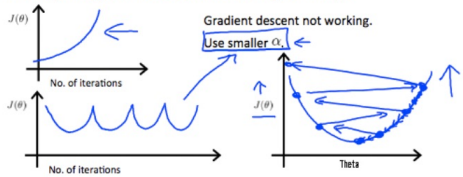
Automatic convergence test. Declare convergence if J(θ) decreases by less than E in one iteration, where E is some small value such as 10^{-3}. However in practice it's difficult to choose this threshold value.

Making sure gradient descent is working correctly.



It has been proven that if learning rate α is sufficiently small, then J(θ) will decrease on every iteration.

Making sure gradient descent is working correctly.



To summarize:

If α is too small: slow convergence.

If α is too large: may not decrease on every iteration and thus may not converge.

Features & Polynomial Regression

Housing prices prediction

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

x_1 x_2

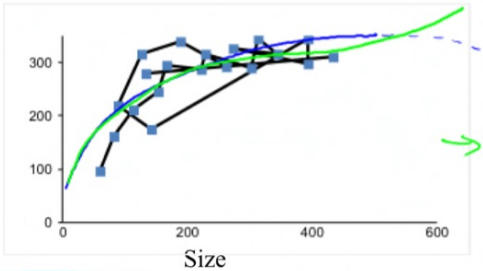
Area

x = frontage * depth

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

↑ land area

Polynomial regression



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

$\rightarrow \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$

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

$= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3$

$\rightarrow x_1 = (size)$

$\rightarrow x_2 = (size)^2$

$\rightarrow x_3 = (size)^3$

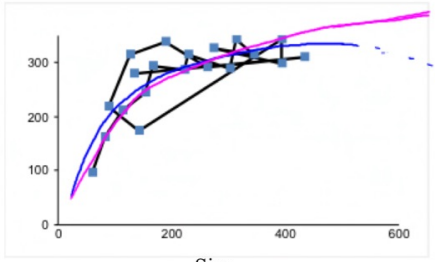
Size: 1 - 1000

Size^2: 1 - 1000,000

Size^3: 1 - 10^9

And

Choice of features



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

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

Question

Suppose you want to predict a house's price as a function of its size. Your model is

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

Suppose size ranges from 1 to 1000 (feet^2). You will implement this by fitting a model

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

Finally, suppose you want to use feature scaling (without mean normalization).

Which of the following choices for x_1 and x_2 should you use? (Note: $\sqrt{1000} \approx 32$.)

- $x_1 = \text{size},\ x_2 = 32\sqrt{(\text{size})}$
- $x_1 = 32(\text{size}),\ x_2 = \sqrt{(\text{size})}$
- (selected) $x_1 = \frac{\text{size}}{1000},\ x_2 = \frac{\sqrt{(\text{size})}}{32}$
- $x_1 = \frac{\text{size}}{32},\ x_2 = \sqrt{(\text{size})}$.

To feature scaling.

$x_1 = \frac{\text{Size}}{1000}$

$x_2 = \frac{\sqrt{\text{Size}}}{1000} = \frac{\sqrt{\text{Size}}}{32}$

Correct

Features and Polynomial Regression

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

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).

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 $x_3 = x_1^3$.

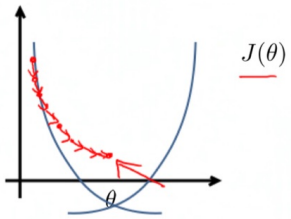
To make it a square root function, we could do: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 \sqrt{x_1}$

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

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

Normal Equation

Gradient Descent

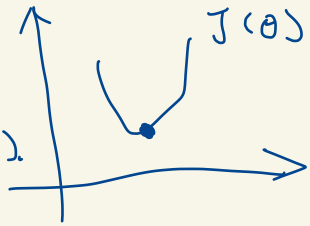


Normal equation: Method to solve for θ analytically.

Intuition: If $\theta \in \mathbb{R}$

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

Solve for θ. $\frac{\partial}{\partial \theta} J(\theta)$.



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

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

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

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

$M \times (n+1)$

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

M dim vector

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

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

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

(design matrix)

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

training example x_1

$M \times (n+1)$

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

$m \times 2$.

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

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

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

Octave: pinv (X' * X) * X' * y (X' = X^T)

pinv $X^T * X$

$(X^T * X)^{-1}$

- Gradient descent : Feature scaling needed

  Normal equation : No need.

m training examples, n features.

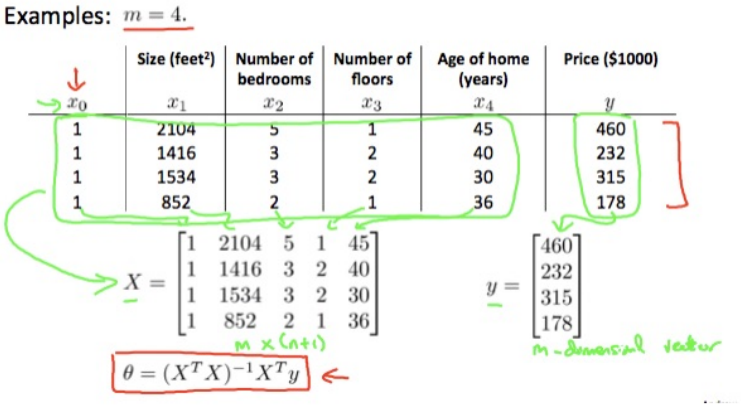
Gradient Descent	Normal Equation
• Need to choose α.	• No need to choose α.
• Needs many iterations.	• Don't need to iterate.
• Works well even when n is large.	• Need to compute $(X^T X)^{-1}$ $O(n^3)$
	• Slow if n is very large.

Normal Equation

Note: [8:00 to 8:44 - The design matrix X (in the bottom right side of the slide) given in the example should have elements x with subscript 1 and superscripts varying from 1 to m because for all m training sets there are only 2 features x_0 and x_1. 12:56 - The X matrix is m by (n+1) and NOT n by n.]

Gradient descent gives one way of minimizing J. Let's discuss 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 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:

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



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 $X^T X$
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.

Normal Equation Noninvertibility

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 feet2
  $x_2 =$ size in m2
- Too many features (e.g. $m \leq n$).
  - Delete some features, or use regularization.