

[share.coursera.org](https://share.coursera.org)

## ML:Linear Regression with Multiple Variables

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 column vector of all the feature inputs of the  $i^{th}$  training example

$m$  = the number of training examples

$n = |x^{(i)}|$  ; (the number of features)

Now define the multivariable form of the hypothesis function as follows, accomodating these multiple features:

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

In order to develop intuition about this function, we can think about  $\theta_0$  as the basic price of a house,  $\theta_1$  as the price per square meter,  $\theta_2$  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 Mr. Ng assumes  $x_0^{(i)} = 1$  for  $(i \in 1, \dots, m)$

**[Note:** So that we can do matrix operations with  $\theta$  and  $x$ , we will set  $x_0^{(i)} = 1$ , for all values of  $i$ . This makes the two vectors ' $\theta$ ' and  $x^{(i)}$  match each other element-wise (that is, have the same number of elements:  $n + 1$ ).]

The training examples are stored in  $X$  row-wise, like such:

$$X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} \\ x_0^{(2)} & x_1^{(2)} \\ x_0^{(3)} & x_1^{(3)} \end{bmatrix}, \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

You can calculate the hypothesis as a column vector of size  $(m \times 1)$  with:

$$h_{\theta}(X) = X\theta$$

**For the rest of this page, and other pages of the wiki,  $X$  will represent a matrix of training examples  $x^{(i)}$  stored row-wise.**

For the parameter vector  $\theta$  (of type  $\mathbb{R}^{n+1}$  or in  $\mathbb{R}^{(n+1) \times 1}$ ), the cost function is:

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

The vectorized version is:

$$J(\theta) = \frac{1}{2m} (X\theta - \vec{y})^T (X\theta - \vec{y})$$

Where  $\vec{y}$  denotes the vector of all  $y$  values.

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

## Matrix Notation

The Gradient Descent rule can be expressed as:

$$\theta := \theta - \alpha \nabla J(\theta)$$

Where  $\nabla J(\theta)$  is a column vector of the form:

$$\nabla J(\theta) = \begin{bmatrix} \frac{\partial J(\theta)}{\partial \theta_0} \\ \frac{\partial J(\theta)}{\partial \theta_1} \\ \vdots \\ \frac{\partial J(\theta)}{\partial \theta_n} \end{bmatrix}$$

The j-th component of the gradient is the summation of the product of two terms:

$$\begin{aligned} \frac{\partial J(\theta)}{\partial \theta_j} &= \frac{1}{m} \sum_{i=1}^m \left( h_{\theta}(x^{(i)}) - y^{(i)} \right) \cdot x_j^{(i)} \\ &= \frac{1}{m} \sum_{i=1}^m x_j^{(i)} \cdot \left( h_{\theta}(x^{(i)}) - y^{(i)} \right) \end{aligned}$$

Sometimes, the summation of the product of two terms can be expressed as the product of two vectors.

Here,  $x_j^{(i)}$ , for  $i = 1, \dots, m$ , represents the  $m$  elements of the  $j$ -th column,  $\vec{x}_j$ , of the training set  $X$ .

The other term  $(h_{\theta}(x^{(i)}) - y^{(i)})$  is the vector of the deviations between the predictions  $h_{\theta}(x^{(i)})$  and the true values  $y^{(i)}$ . Re-writing  $\frac{\partial J(\theta)}{\partial \theta_j}$ , we have:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m} \vec{x}_j^T (X\theta - \vec{y})$$

$$\nabla J(\theta) = \frac{1}{m} X^T (X\theta - \vec{y})$$

Finally, the matrix notation (vectorized) of the Gradient Descent rule is:

$$\theta := \theta - \frac{\alpha}{m} X^T (X\theta - \vec{y})$$

We can speed up gradient descent by having each of our input values in roughly the same range. This is because  $\theta$  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  $\mu_i$  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.

Example:  $x_i$  is size of the house in square meters with range of 100 to 2000, with a mean value of 1000. Then,  $x_i$

$$:= \frac{\text{size} - 1000}{1900}.$$

**Take into account** that if we want to do a new prediction based in the hypothesis function with the thetas we have obtained from a normalized set of input values, we will have to normalize as well the input parameters:

Example: After we normalized the size of the housing as in the previous example we obtained a function  $h(x)$ . If we want to predict the prize of a house in which the size is 1500, when implementing  $h(x)$  we will have to apply it not to the **size** but to its **normalized** value:  $h_{\theta}(x) = \theta_0 + \theta_1 x_1 + \dots + \theta_i x_i + \dots + \theta_n x_n = \theta_0 + \theta_1 x_1 + \dots + \theta_i \frac{\text{size} - 1000}{1900} + \dots + \theta_n x_n$

### Quiz question #1 on Feature Normalization (Week 2, Linear Regression with Multiple Variables)

Your answer should be rounded to exactly two decimal

places. Use a '.' for the decimal point, not a ','. The tricky part of this question is figuring out which feature of which training example you are asked to normalize. Note that the mobile app doesn't allow entering a negative number (Jan 2016), so you will need to use a browser to submit this quiz if your solution requires a negative number.

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

**Automatic convergence test.** Declare convergence if  $J(\theta)$  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.

It has been proven that if learning rate  $\alpha$  is sufficiently small, then  $J(\theta)$  will decrease on every iteration. Andrew Ng recommends decreasing  $\alpha$  by multiples of 3.

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

Note that at 2:52 and through 6:22 in the "Features and Polynomial Regression" video, the curve that Prof Ng discusses about "doesn't ever come back down" is in reference to the hypothesis function that uses the `sqrt()` function (shown by the solid purple line), not the one that uses  $x_1^2$  (shown with the dotted blue line). The quadratic form of the hypothesis function would have the shape shown with the blue dotted line if  $\theta_2$  was negative.

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

The "Normal Equation" is a method of finding the optimum theta **without iteration**.

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

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

Mathematical proof of the Normal equation requires knowledge of linear algebra and is fairly involved, so you do not need to worry about the details.

Proofs are available at these links for those who are interested: [Wikipedia](#) [Eli Bendersky](#) [A proof easy to understand](#)

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  $O(n^3)$ . 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

When implementing the normal equation in octave we want to use the 'pinv' function rather than 'inv.'

$X^T X$  may be **noninvertible**. The common causes are:

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

Solutions to the above problems include deleting a feature that is linearly dependent with another or deleting one or more features when there are too many features.

---

Next: [Octave Tutorial](#) Back to Index: [Main](#)