# Week 2 Lecture Notes

# 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.

|  |
| --- |
| value of feature *j* in the *ith* training example  the column vector of all the feature inputs of the  *ith*  training example  the number of training examples  ;(the number of features) |

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

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

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

|  |
| --- |
|  |

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 ​ for ( *i*∈1,…,*m*)

[**Note**: So that we can do matrix operations with theta and x, we will set , for all values of i. This makes the two vectors 'theta' and​ 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:

|  |
| --- |
|  |

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

**For the rest of these notes, and other lecture notes, X will represent a matrix of training examples** ​ **stored row-wise.**

## ****Cost function****

For the parameter vector θ (of type or in , the cost function is:

The vectorized version is:

Where denotes the vector of all y values.

## ****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: {  *…*  *}* |

In other words:

|  |
| --- |
| repeat until convergence: {  for j := 0 …. n  *}* |

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

## 

## Matrix Notation

The Gradient Descent rule can be expressed as:

Where is a column vector of the form:

|  |
| --- |
|  |

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

|  |
| --- |
|  |

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

Here, for i = 1,...,m, represents the m elements of the j-th column, ​​ , of the training set X.

The other term is the vector of the deviations between the predictions and the true values. Re-writing , we have:

|  |
| --- |
|  |

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

|  |
| --- |
|  |

# Feature Normalization

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 ≤ *x*(*i*)​ ≤ 1

or

−0.5 ≤ *x*(*i*)​ ≤ 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:

Whereis the **average** of all the values for feature (i) and is the range of values (max - min), or ​ 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: is housing prices with range of 100 to 2000, with a mean value of 1000.

Then, .

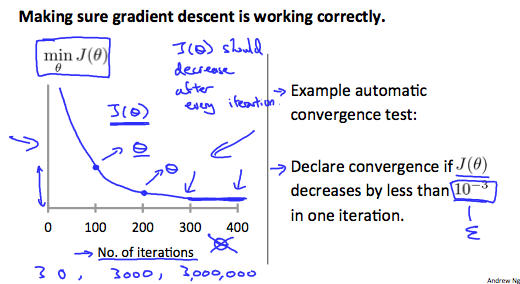
## Quiz question #1 on Feature Normalization (Week 2, Linear Regression with Multiple Variables)

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.

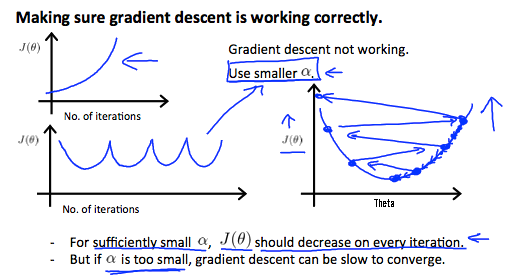
# Gradient Descent Tips: Learning Rate

**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.



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



To summarize:

If is too small: slow convergence.

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

# 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 ​ and ​ into a new feature by taking

### ****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 ​ then we can create additional features based on , to get the quadratic function ​ or the cubic function

In the cubic version, we have created new features​ and ​ where ​ and

To make it a square root function, we could do:

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 *size*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 ​ was negative.

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

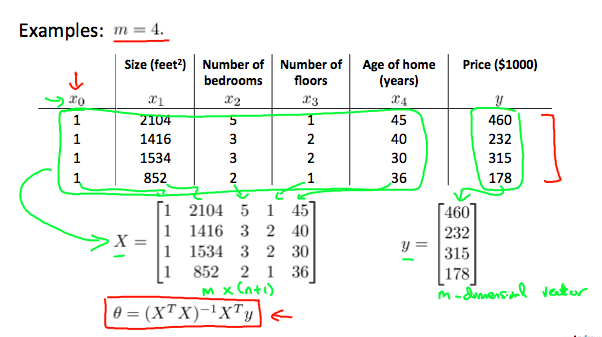
eg. if​ has range 1 - 1000 then range of ​ becomes 1 - 1000000 and that of ​ becomes 1 - 1000000000.

# 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 and ​. 12:56 - The X matrix is m by (n+1) and NOT n by n. ]

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

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:



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:

<https://en.wikipedia.org/wiki/Linear_least_squares_(mathematics)>

<http://eli.thegreenplace.net/2014/derivation-of-the-normal-equation-for-linear-regression>

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 (*kn2*) | O (*n*3), need to calculate inverse of |
| 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.'

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 ≤ 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.