The Hypothesis Function

Our hypothesis function has the general form:

$$\hat{y} = h_\theta(x) = \theta_0 + \theta_1 x$$

Note that this is like the equation of a straight line. We give to $h_\theta(x)$ values for θ_0 and θ_1 to get our estimated output $\hat{y}$. In other words, we are trying to create a function called h_θ that is trying to map our input data (the x's) to our output data (the y's).

Cost Function

We can measure the accuracy of our hypothesis function by using a **cost function**. This takes an average (actually a fancier version of an average) of all the results of the hypothesis with inputs from x's compared to the actual output y's.

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2 = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)^2$$

To break it apart, it is $\frac{1}{2}\ \bar{x}$ where $\bar{x}$ is the mean of the squares of $h_\theta(x_i) - y_i$, or the difference between the predicted value and the actual value.

This function is otherwise called the "Squared error function", or "Mean squared error". The mean is halved $\left(\frac{1}{2m}\right)$ as a convenience for the computation of the gradient descent, as the derivative term of the square function will cancel out the $\frac{1}{2}$ term.

Now we are able to concretely measure the accuracy of our predictor function against the correct results we have so that we can predict new results we don't have.

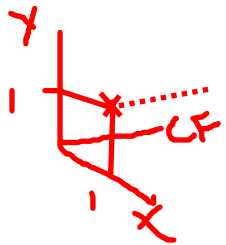
If we try to think of it in visual terms, our training data set is scattered on the x-y plane. We are trying to make straight line (defined by $h_\theta(x)$) which passes through this scattered set of data. Our objective is to get the best possible line. The best possible line will be such so that the average squared vertical distances of the scattered points from the line will be the least. In the best case, the line should pass through all the points of our training data set. In such a case the value of $J(\theta_0, \theta_1)$ will be 0.

ML:Gradient Descent

So we have our hypothesis function and we have a way of measuring how well it fits into the data. Now we need to estimate the parameters in hypothesis function. That's where gradient descent comes in.

Imagine that we graph our hypothesis function based on its fields Theta_0 and theta_1 (actually we are graphing the cost function as a function of the parameter estimates). This can be kind of confusing; we are moving up to a higher level of abstraction. We are not graphing x and y itself, but the parameter range of our hypothesis function and the cost resulting from selecting particular set of parameters.

We put theta_0 on the x axis and theta_1 on the y axis, with the cost function on the vertical z axis. The points on our graph will be the result of the cost function using our hypothesis with those specific theta parameters.



We will know that we have succeeded when our cost function is at the very bottom of the pits in our graph, i.e. when its value is the minimum.

The way we do this is by taking the derivative (the tangential line to a function) of our cost function. The slope of the tangent is the derivative at that point and it will give us a direction to move towards. We make steps down the cost function in the direction with the steepest descent, and the size of each step is determined by the parameter a, which is called the learning rate.

The gradient descent algorithm is:

repeat until convergence: { $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ }

where

j=0,1 represents the feature index number.

Intuitively, this could be thought of as:

repeat until convergence:

$\theta_j := \theta_j - \alpha$[Slope of tangent aka derivative in j dimension][Slope of tangent aka derivative in j dimension]

Gradient Descent for Linear Regression

When specifically applied to the case of linear regression, a new form of the gradient descent equation can be derived. We can substitute our actual cost function and our actual hypothesis function and modify the equation to (the derivation of the formulas are out of the scope of this course, but a really great one can be found here):

Derivative nikal ke, aise likha hai

repeat until convergence: {

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x_i) - y_i)$$

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

}

where m is the size of the training set, θ_0 a constant that will be changing simultaneously with θ_1 and x_i, y_i are values of the given training set (data).

Note that we have separated out the two cases for θ_j into separate equations for θ_0 and θ_1; and that for θ_1 we are multiplying x_i at the end due to the derivative.

The point of all this is that if we start with a guess for our hypothesis and then repeatedly apply these gradient descent equations, our hypothesis will become more and more accurate.

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

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

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

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

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

[**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 x 1) with:

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

For the rest of these notes, and other lecture notes, X will represent a matrix of training examples $x_{(i)}$ stored row-wise.

Cost function

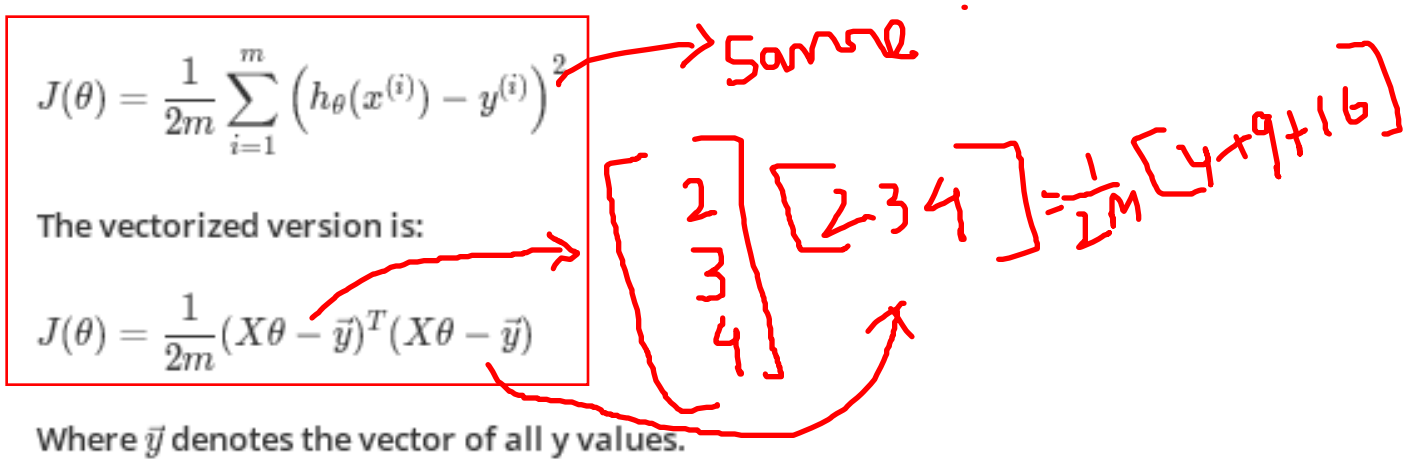
For the parameter vector θ (of type $\mathbb{R}^{n+1}$ or in $\mathbb{R}^{(n+1)\times 1}$, the cost function is:

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

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.



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: {

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

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

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

$\dots$

}

Har ek theta ki value ko:

alpha * 1/m sum((h(x) - y) * x)

yes sab particular theta ki particular feature pe ho raha hai

alpha=1

theta0- 1/m * sum((h(x0) - y0)*x0)

In other words:

repeat until convergence: {

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

}

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,...,m, represents the m elements of the j-th column, $\vec{x}_j$, of the training set X.

The other term $\left(h_\theta(x^{(i)}) - y^{(i)}\right)$ 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:

$$\begin{aligned} \frac{\partial J(\theta)}{\partial \theta_j} &= \frac{1}{m} \vec{x_j}^T (X\theta - \vec{y}) \\ \nabla J(\theta) &= \frac{1}{m} X^T (X\theta - \vec{y}) \end{aligned}$$

Sqaure hata hai dekh, toh derivative nikal diya hai matlab

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

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

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:

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

or

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

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.

Example: x_i is housing prices with range of 100 to 2000, with a mean value of 1000. Then, $x_i := \frac{price - 1000}{1900}$.

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

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 θ_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.

Normal Equation

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.