**Classification**:

-- To attempt classification, one method is to use linear regression and map all predictions greater than 0.5 as a 1 and all less than 0.5 as a 0. However, this method doesn't work well because classification is not actually a linear function.

--The classification problem is just like the regression problem, except that the values we now want to predict take on only a small number of discrete values. For now, we will focus on the binary classification problem in which y can take on only two values, 0 and 1.(Most of what we say here will also generalize to the multiple-class case.) For instance, if we are trying to build a spam classifier for email, then x^{(i)} x (i) may be some features of a piece of email, and y may be 1 if it is a piece of spam mail, and 0 otherwise. Hence, y∈{0,1}. 0 is also called the negative class, and 1 the positive class, and they are sometimes also denoted by the symbols “-” and “+.” Given x^{(i)}x (i) , the corresponding y^{(i)}y (i) is also called the label for the training example.

**Logistic regression:**

**What is logistic Regression?**

- It is the classification problem which is one of the most popular and widely used algorithm in machine learning.

- Here the output variable we want to predict is the discrete outcomes eg: y ∈ {0, 1}

- Some Example:

-- Email classification: (Spam/Not Spam)

-- Online Transaction: Fraud (Yes/No)

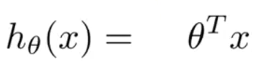
-- Tumour: Malignant/Benign?

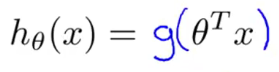
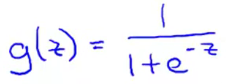
- Linear Regression is not often good idea for the classification problem.

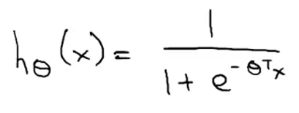
For example let’s say in classification we have the output variable y will be the discrete outcomes y ∈ {0, 1} but in linear regression the h(x) value can be 1> or <0

**Hypothesis function:**

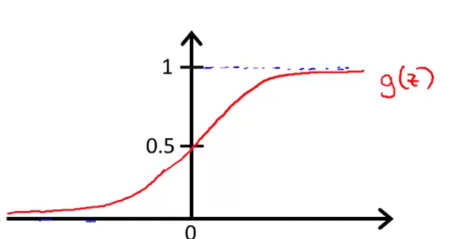
Hypothesis representation implies that what is the function we are going to represent our hypothesis when we have a classification problem. We want our regression model would be 0=<h(x)=<1

In linear regression the hypothesis function represented as 

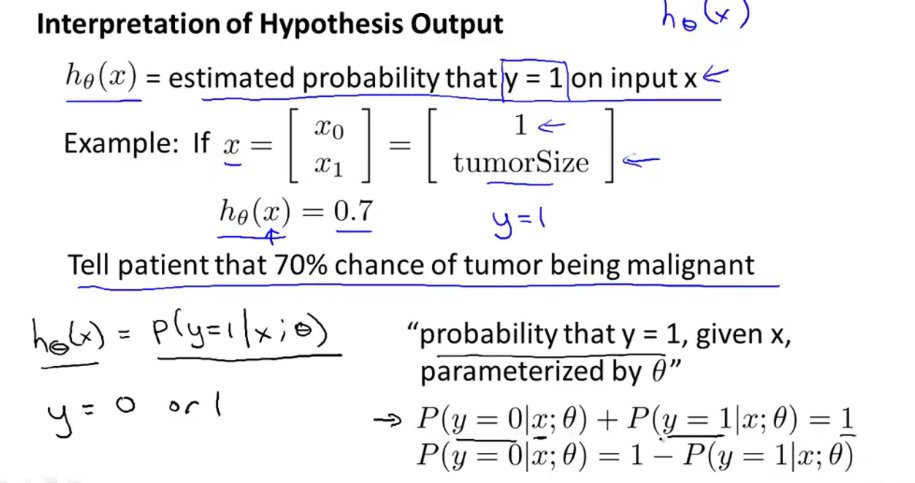
But in logistic regression. This is rep as sigmoid function.



The sigmoid function look like



And you notice that the sigmoid function, while it asymptotes at one and asymptotes at zero, as a z axis, the horizontal axis is z.



Hypothesis Representation

We could approach the classification problem ignoring the fact that y is discrete-valued, and use our old linear regression algorithm to try to predict y given x. However, it is easy to construct examples where this method performs very poorly. Intuitively, it also doesn’t make sense for h\_\theta (x)*hθ*​(*x*) to take values larger than 1 or smaller than 0 when we know that y ∈ {0, 1}. To fix this, let’s change the form for our hypotheses h\_\theta (x)*hθ*​(*x*) to satisfy 0 \leq h\_\theta (x) \leq 10≤*hθ*​(*x*)≤1. This is accomplished by plugging  \theta^Tx*θTx*  into the Logistic Function.

Our new form uses the "Sigmoid Function," also called the "Logistic Function":

|  |
| --- |
| *hθ*(*x*)=*g*(*θTx*)*z*=*θTxg*(*z*)=11+*e*−*z* |

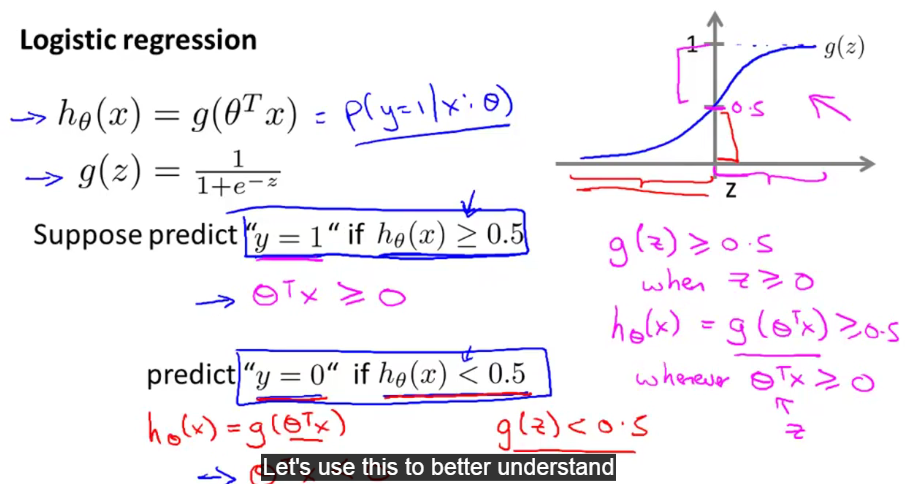
The following image shows us what the sigmoid function looks like:



The function g(z), shown here, maps any real number to the (0, 1) interval, making it useful for transforming an arbitrary-valued function into a function better suited for classification.

h\_\theta(x)*hθ*​(*x*) will give us the **probability** that our output is 1. For example, h\_\theta(x)=0.7*hθ*​(*x*)=0.7 gives us a probability of 70% that our output is 1. Our probability that our prediction is 0 is just the complement of our probability that it is 1 (e.g. if probability that it is 1 is 70%, then the probability that it is 0 is 30%).

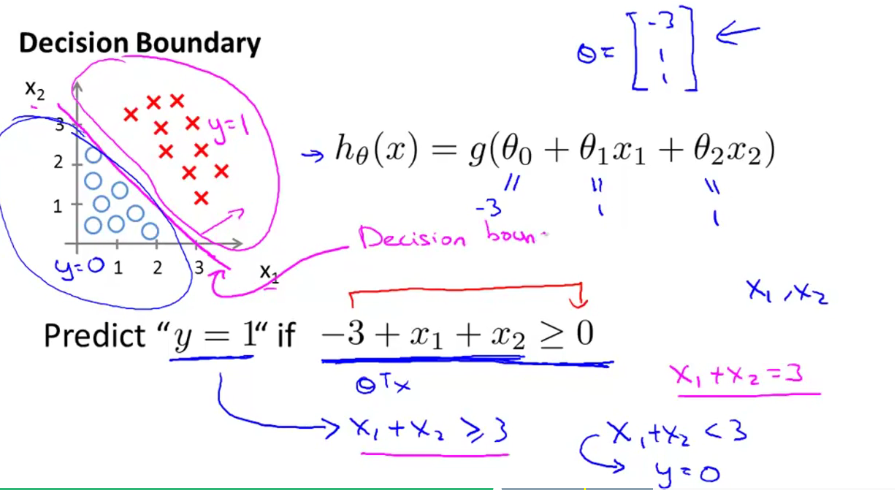
|  |
| --- |
| *hθ*(*x*)=*P*(*y*=1|*x*;*θ*)=1−*P*(*y*=0|*x*;*θ*)*P*(*y*=0|*x*;*θ*)+*P*(*y*=1|*x*;*θ*)=1 |

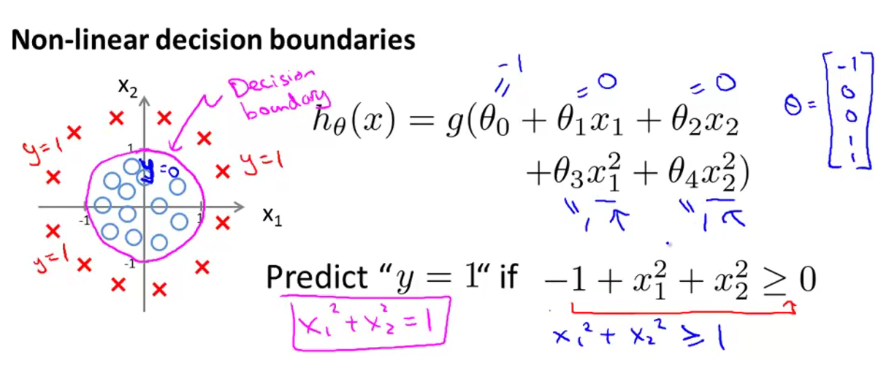


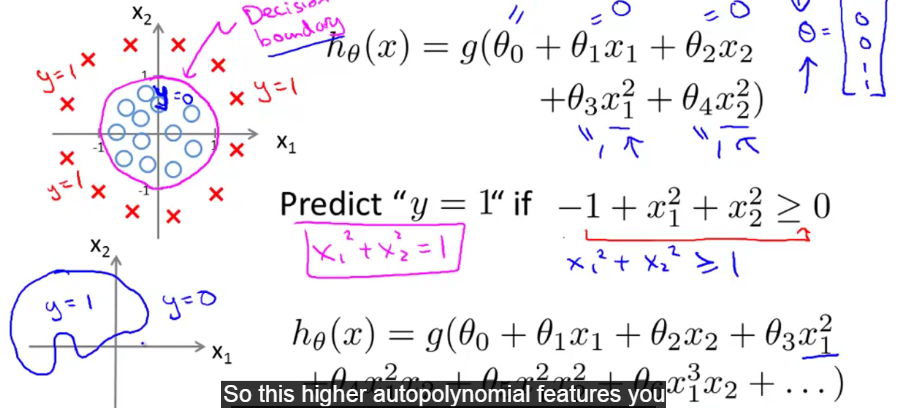
**Decision Boundary**:

This will give us the better intuition or sense about what the logistic regression hypothesis function is computing.

Decision boundary is the property of the hypothesis, not the property of the dataset.

It is the common boundary that separate the positive values and negative values.





**From course**:

Decision Boundary

In order to get our discrete 0 or 1 classification, we can translate the output of the hypothesis function as follows:

|  |
| --- |
| *hθ*(*x*)≥0.5→*y*=1  *hθ*(*x*)<0.5→*y*=0 |

The way our logistic function g behaves is that when its input is greater than or equal to zero, its output is greater than or equal to 0.5:

|  |
| --- |
| *g*(*z*)≥0.5 *when z*≥0 |

Remember.

|  |
| --- |
| *z*=0,*e*0=1⇒*g*(*z*)=1/2  *z*→∞,*e*−∞→0⇒*g*(*z*)=1  *z*→−∞,*e*∞→∞⇒*g*(*z*)=0 |

So if our input to g is *θTX*, then that means:

|  |
| --- |
| *hθ*(*x*)=*g*(*θTx*)≥0.5 *when θTx*≥0 |

From these statements we can now say:

|  |
| --- |
| *θTx*≥0⇒*y*=1  *θTx*<0⇒*y*=0 |

The **decision boundary** is the line that separates the area where y = 0 and where y = 1. It is created by our hypothesis function.

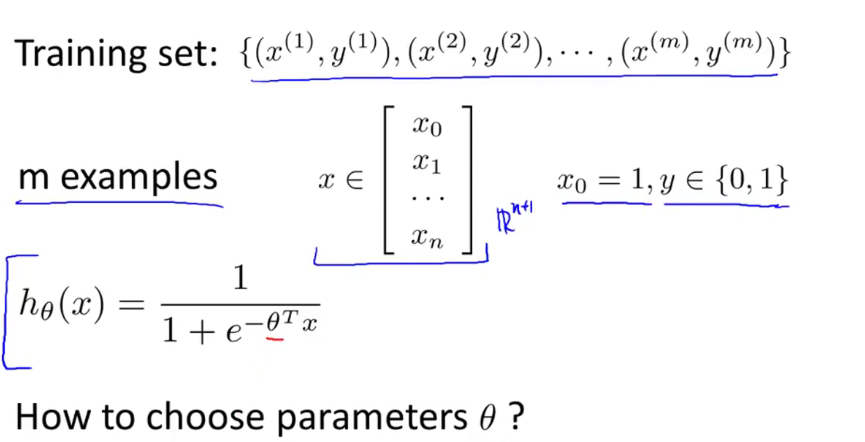
**Example**:

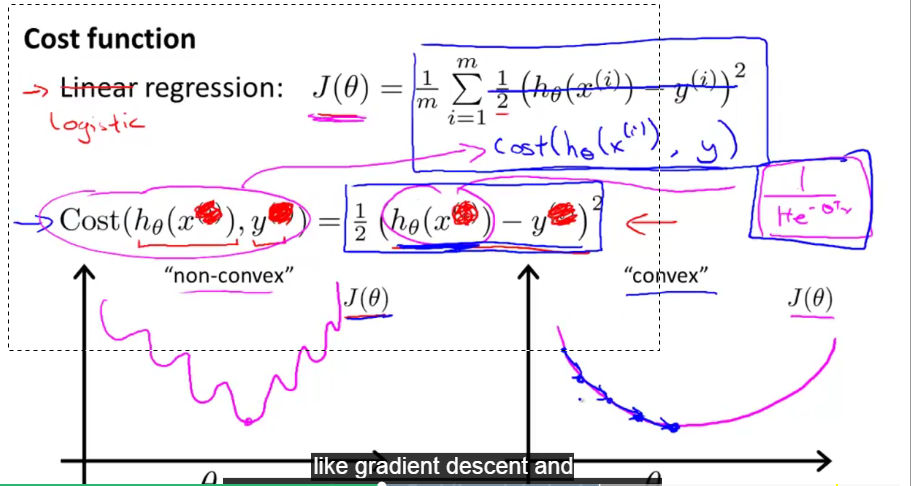
|  |
| --- |
| *θ*=⎡⎣5−10⎤⎦  *y*=1 *if*5+(−1)*x*1+0*x*2≥0  5−*x*1≥0  −*x*1≥−5  *x*1≤5 |

In this case, our decision boundary is a straight vertical line placed on the graph where x\_1 = 5*x*1​=5, and everything to the left of that denotes y = 1, while everything to the right denotes y = 0.

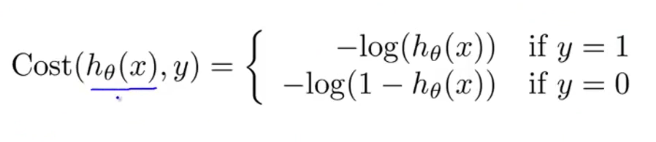
Again, the input to the sigmoid function g(z) (e.g.  *θTX*) doesn't need to be linear, and could be a function that describes a circle (e.g. z = \theta\_0 + \theta\_1 x\_1^2 +\theta\_2 x\_2^2*z*=*θ*0​+*θ*1​*x*12​+*θ*2​*x*22​) or any shape to fit our data.

**COST FUNCTION:**



* Let’s consider the training set in the above image
* It has **m** samples and as usual consider **x0 as 1** and the hypothesis function is given above.
* Our objective is to find out the THETA parameters that suits for our classification problem.
* 
* The cost function for the linear regression is given above but we can’t use the same cost function for logistic regression because **(square cost function)** h(x) hypothesis used in the linear regression is the linear one so the cost function will converge or convex but in logistic regression the hypothesis function is a nonlinear function so the cost function will be a non-convex one. So the same cost function we can’t use it here.
* We cannot use the same cost function that we use for linear regression because the Logistic Function will cause the output to be wavy, causing many local optima. In other words, it will not be a convex function.

So here is the cost function we are going to use for the logistic regression



***J*(*θ*)=1*m*∑*i*=1*m*Cost(*hθ*(*x*(*i*)),*y*(*i*))**

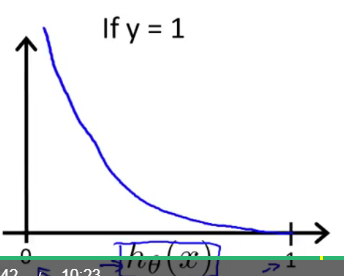
**Cost(*hθ*(*x*),*y*)=−log(*hθ*(*x*)) if y = 1**

**Cost(*hθ*(*x*),*y*)=−log(1−*hθ*(*x*)) if y = 0**

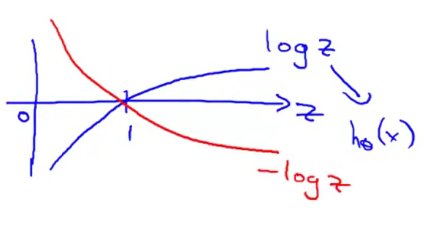
Let’s plot some figures to get the better sense about this cost function.

Let start off with case of y=1

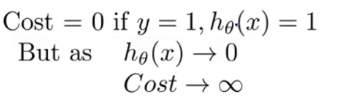
Here the horizontal axis is h(x) and it varies between 0 to 1.If you plot the cost function will look like

 case1

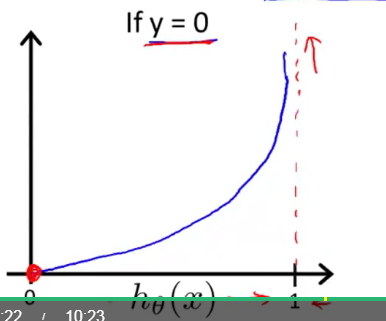
Let’s see why this look like. Let us draw the plot for the log Z .And we need the only the particular portion between 0 and 1.



From the case1, we can say that cost is Zero if h(x) tends to 1 but cost is tends to infinite if h(x) reaches zero. **From these we can capture the intuition that If h(x)=0 but y=1 then the cost will be very high.**



Now let’s do it for when y=0

 case2

**If h(x) reaches 1 then the cost will be very high if h(x) =0 then the cost is ZERO.**

Finally we can say

**Cost(*hθ*(*x*),*y*)=0 if *hθ*(*x*)=*y***

**Cost(*hθ*(*x*),*y*)→∞ if *y*=0 and *hθ*(*x*)→1**

**Cost(*hθ*(*x*),*y*)→∞ if *y*=1 and *hθ*(*x*)→0**

**Note:**

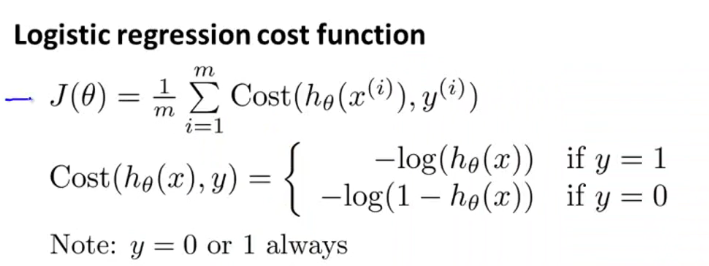
**If our correct answer 'y' is 0, then the cost function will be 0 if our hypothesis function also outputs 0. If our hypothesis approaches 1, then the cost function will approach infinity.**

**If our correct answer 'y' is 1, then the cost function will be 0 if our hypothesis function outputs 1. If our hypothesis approaches 0, then the cost function will approach infinity.**

**Note that writing the cost function in this way guarantees that J(θ) is convex for logistic regression.**

**Simplified cost function and Gradient descent**:

Here we will see the simplest way to write the cost function and also figure out how to apply gradient descent to fit the parameters of logistic regression. Here is our cost function for logistic regression.



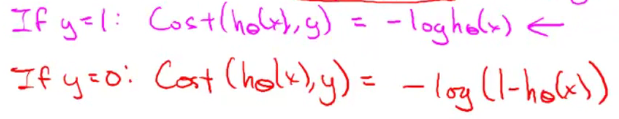
Our cost function is 1 over m times the sum over the trading set of .

Because y is either 0 or 1, we will able to come up with a simpler way to write this cost function. And in particular, rather than writing out of this cost function on two separate lines with two cases,so y equals to 1 and y equals to 0.I am going to show you a way to take these two lines and compress them into one equation. And this would make more convenient to write our cost function and derive gradient descent.

So we can write the cost function as follows:

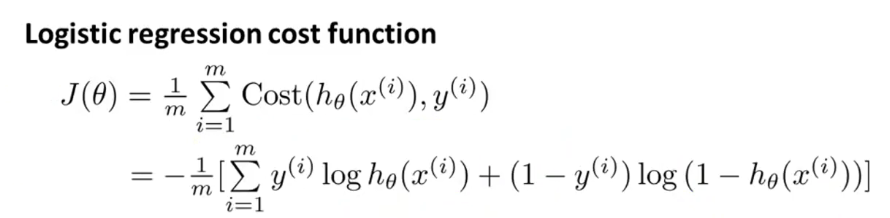


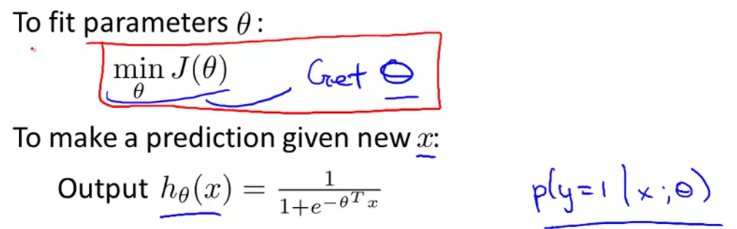
This equation is equivalent to the above cost function or more compact way of writing out this definition of this cost function that we have up here.

We know y must be 0 or 1.

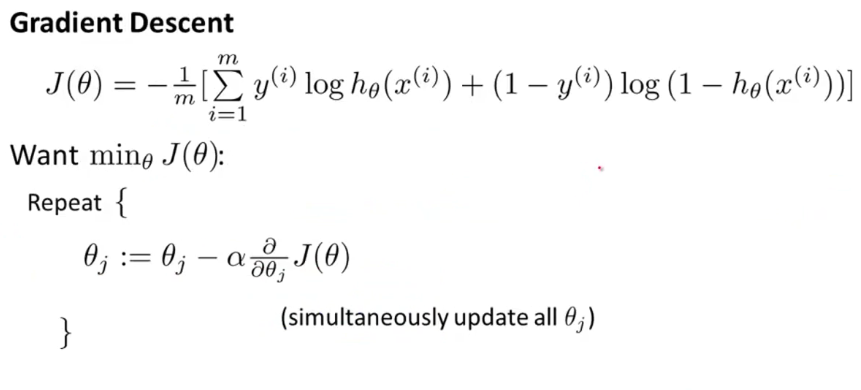
This exactly similar to the above cost function.

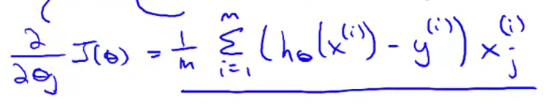
So we can re write as



And to fit the parameters.

The way we are goint to use to minimize the cost function is gradient descent.

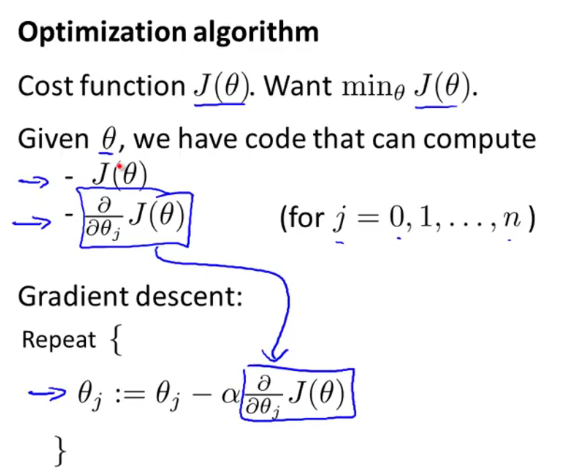




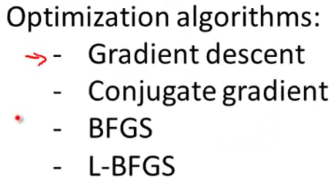
**Advanced optimization:**

So far we have seen the gradient descent algorithm to minimize the logistic regression cost function, now we are going to look into the more advanced optimized concepts to minimize the logistic regression cost function.

Here is what gradient descent is doing.



GD is not only algorithm used to minimize the cost function. We do have other algorithms such as



These algorithms are very useful if we have large number of features and the performance will be very good compared to the GD.

"Conjugate gradient", "BFGS", and "L-BFGS" are more sophisticated, faster ways to optimize θ that can be used instead of gradient descent. We suggest that you should not write these more sophisticated algorithms yourself (unless you are an expert in numerical computing) but use the libraries instead, as they're already tested and highly optimized. Octave provides them.