# Week3

### Table of Contents



# Classification and Representation

## Classification

Explain why the linear regression is unsuitable for classification problems.

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

Apply LinearRegression to a classification problem is not a great idea.

## Question

Which of the following statements is true?

- ○ If linear regression doesn't work on a classification task as in the previous example shown in the video, applying feature scaling may help.
- ○ If the training set satisfies $0 \leq y^{(i)} \leq 1$ for every training example $(x^{(i)}, y^{(i)})$, then linear regression's prediction will also satisfy $0 \leq h_\theta(x) \leq 1$ for all values of $x$.
- ○ If there is a feature $x$ that perfectly predicts $y$, i.e. if $y = 1$ when $x \geq c$ and $y = 0$ whenever $x < c$ (for some constant $c$), then linear regression will obtain zero classification error.
- ◉ None of the above statements are true.

✓ **Correct**

The classification problem is just like the regression problem, except that the values we now want to predict take on 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.

> 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 $x^{(i)}$ 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 $x^{(i)}$, the corresponding $y^{(i)}$ is also called the label for the training example.

So we develop an algorithm called **Logistic Regression.** Which has the property the output of the predictions of regression are always between zero and one.

# Hypothesis Representation

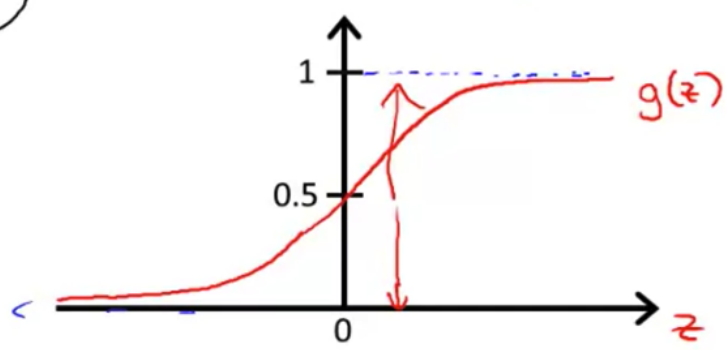
## Logistic Regression Model

Want $0 \le h_\theta(x) \le 1$

$h_\theta(x) = g(\theta^T x)$

$g(z) = \frac{1}{1+e^{-z}}$, $z = \theta^T x$

$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$




Sigmoid function

Logistic function

Parameters $\theta$.

## Interpretation of Hypothesis Output

$h_\theta(x)$

$h_\theta(x)$ = estimated probability that y = 1 on input x

Example: If $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorSize} \end{bmatrix}$

$h_\theta(x) = 0.7$ $\quad y=1$

Tell patient that 70% chance of tumor being malignant

$h_\theta(x) = P(y=1|x;\theta)$

"probability that y = 1, given x, parameterized by $\theta$"

$y = 0$ or $1$

$P(y=0|x;\theta) + P(y=1|x;\theta) = 1$

$P(y=0|x;\theta) = 1 - P(y=1|x;\theta)$

**Summary**

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 $h_\theta(x)$ 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 $h_\theta(x)$ to satisfy $0 \leq h_\theta(x) \leq 1$. This is accomplished by plugging $\theta^T x$ into the Logistic Function.

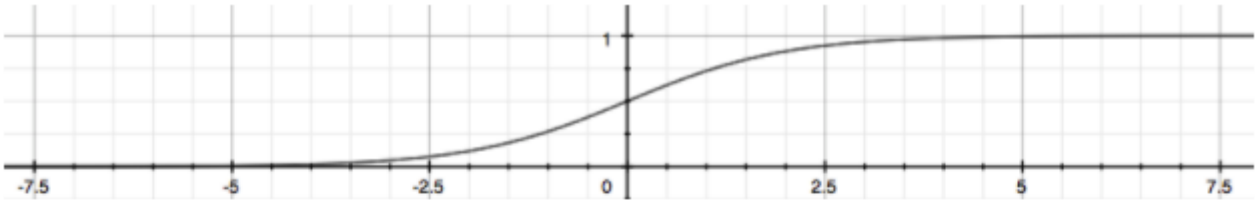
Our new form uses the "Sigmoid Function," also called the "Logistic Function":

$$h_\theta(x) = g(\theta^T x)$$

$$z = \theta^T x$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

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.

$h_\theta(x)$ will give us the **probability** that our output is 1. For example, $h_\theta(x) = 0.7$ 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%).

$$h_\theta(x) = P(y = 1|x; \theta) = 1 - P(y = 0|x; \theta)$$
$$P(y = 0|x; \theta) + P(y = 1|x; \theta) = 1$$

Chinese：

Logistic 函数可以看成是一个“挤压”函数，把一个实数域的输入“挤压”到 (0, 1).当输入值在 0 附近时，Sigmoid 型函数近似为线性函数;当输入值靠近两端 时，对输入进行抑制.输入越小，越接近于 0;输入越大，越接近于 1.这样的特点 也和生物神经元类似，对一些输入会产生兴奋(输出为 1)，对另一些输入产生抑 制(输出为 0).和感知器使用的阶跃激活函数相比，Logistic 函数是连续可导的， 其数学性质更好.

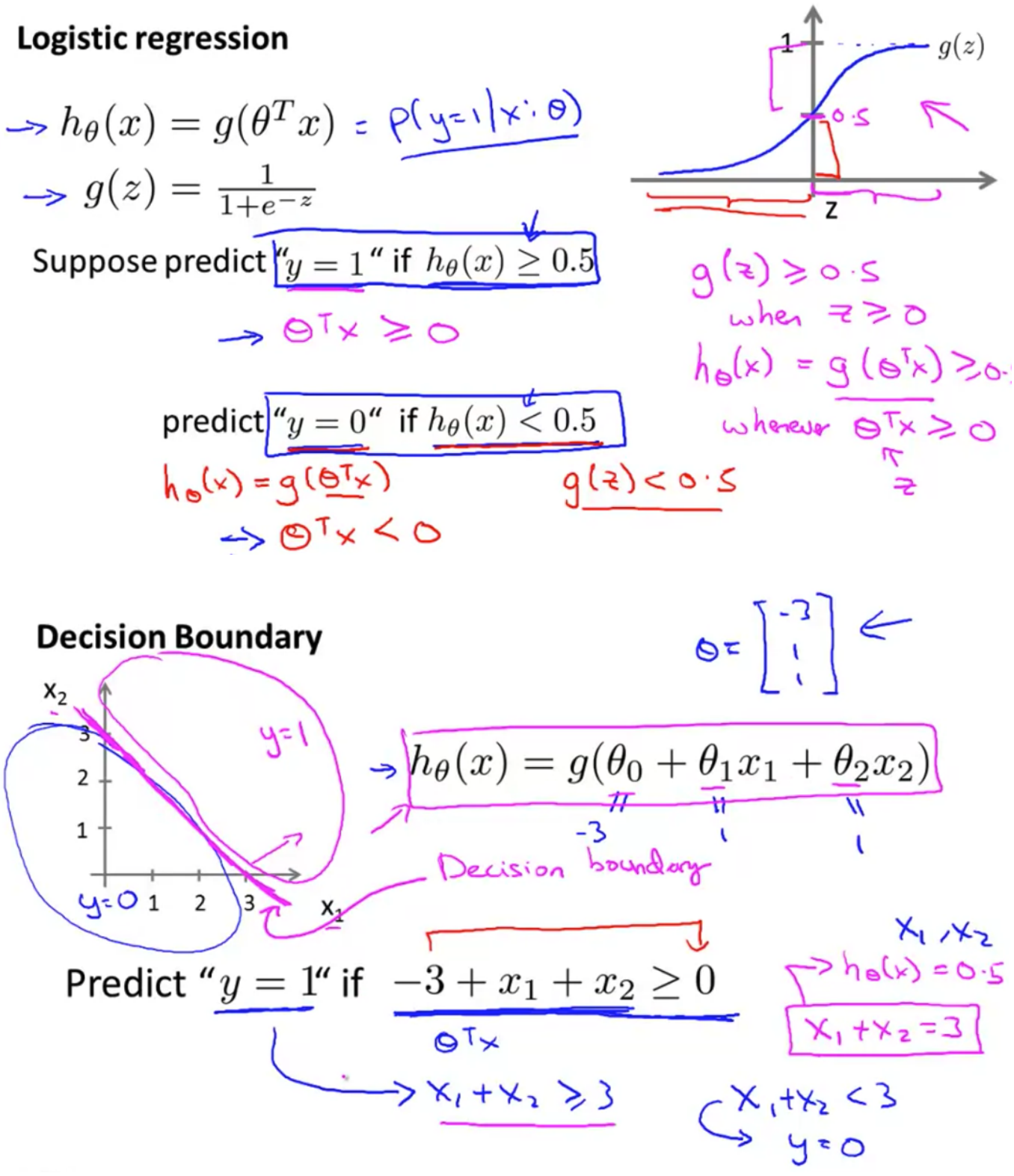
因为 Logistic 函数的性质，使得装备了 Logistic 激活函数的神经元具有以下 两点性质:

1)其输出直接可以看作概率分布，使得神经网络可以更好地和统计学习模型进行结合.

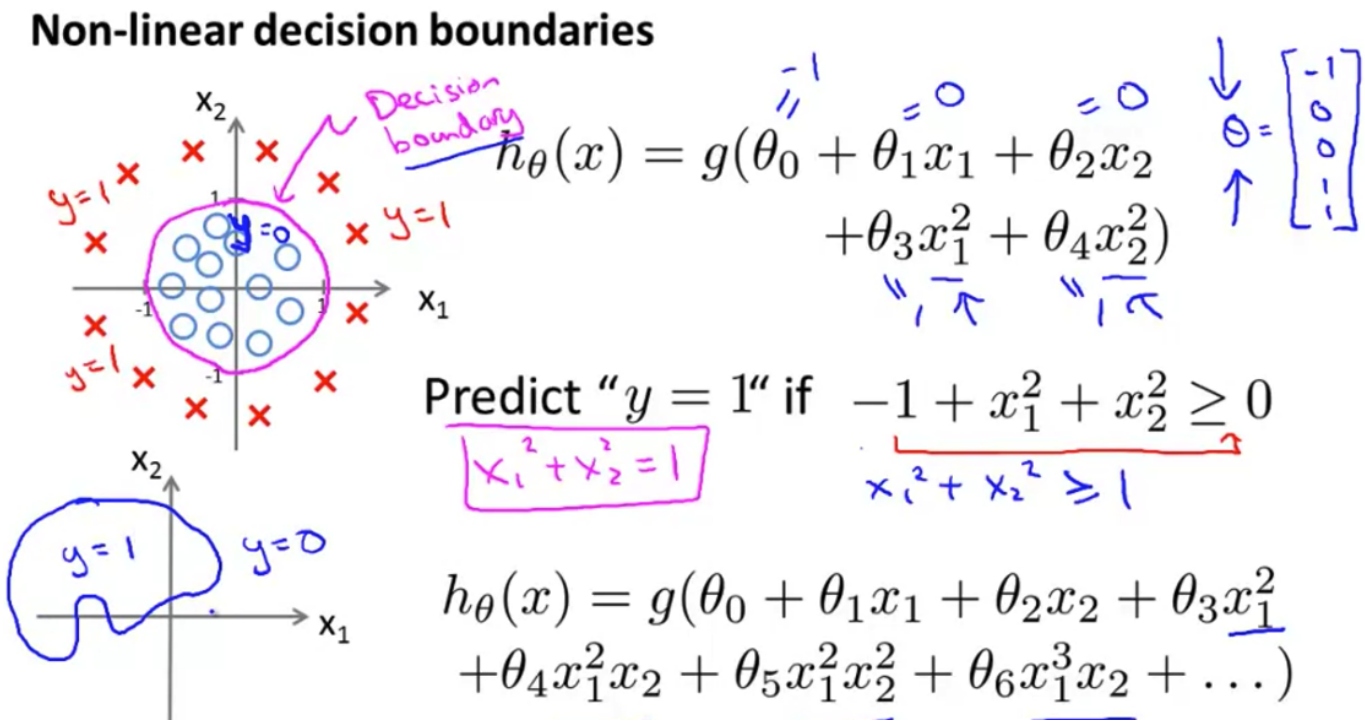
2)其可以看作一个软性门(Soft Gate)，用来控制其他神经元输出信息的数量.

## Decision Boundary

Give us a better sense of what the logistic regressions hypothesis function is computing.



Non-linear decision boundaries




Summary

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

$$h_\theta(x) \geq 0.5 \rightarrow y = 1$$
$$h_\theta(x) < 0.5 \rightarrow y = 0$$

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:

$$g(z) \geq 0.5$$
$$when\ z \geq 0$$

Remember.

$$z = 0, e^0 = 1 \Rightarrow g(z) = 1/2$$
$$z \rightarrow \infty, e^{-\infty} \rightarrow 0 \Rightarrow g(z) = 1$$
$$z \rightarrow -\infty, e^{\infty} \rightarrow \infty \Rightarrow g(z) = 0$$

So if our input to g is $\theta^T X$, then that means:

$$h_\theta(x) = g(\theta^T x) \geq 0.5$$
$$when\ \theta^T x \geq 0$$

From these statements we can now say:

$$\theta^T x \geq 0 \Rightarrow y = 1$$
$$\theta^T x < 0 \Rightarrow y = 0$$

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

# Logistic Regression Model

## Cost Function

How do we fit the parameter's theta.

We can't use the same function that we use for linear regression because the Logistic Function will cause the output to be wavy[non-convex],causing many local optima. In other words, it will not be a convex function.

Training set: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \cdots, (x^{(m)}, y^{(m)})\}$

m examples $\quad x \in \begin{bmatrix} x_0 \\ x_1 \\ \cdots \\ x_n \end{bmatrix} \in \mathbb{R}^{n+1} \quad x_0 = 1, y \in \{0, 1\}$

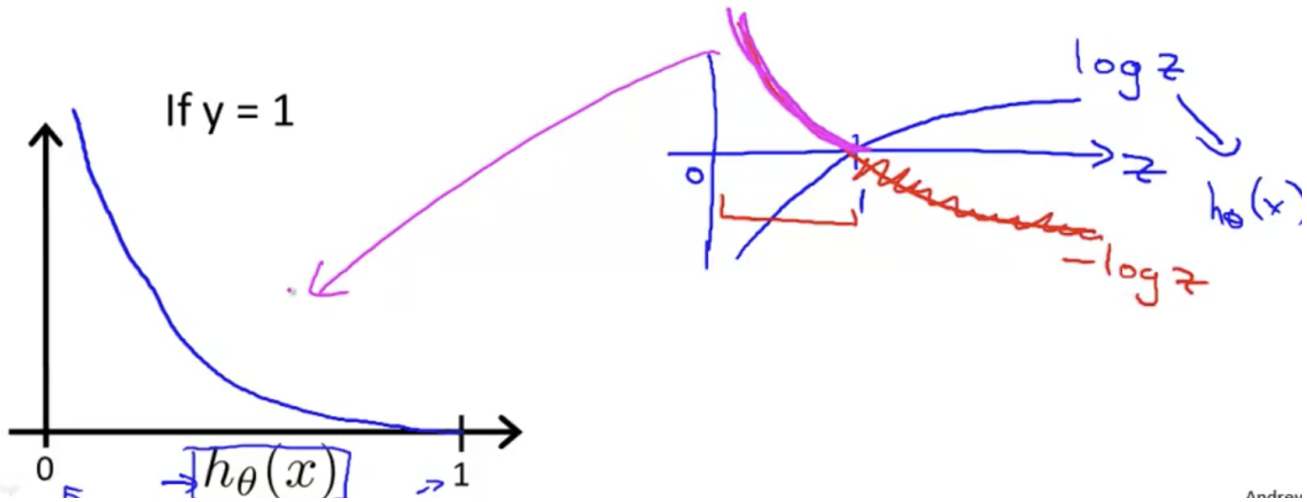
$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

How to choose parameters $\theta$ ?

Logistic regression cost function

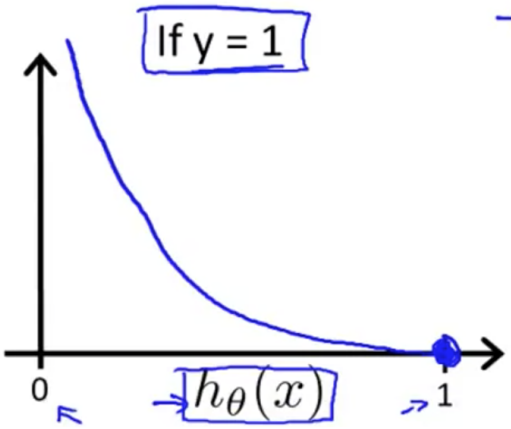
**Logistic regression cost function**

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



## Logistic regression cost function

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

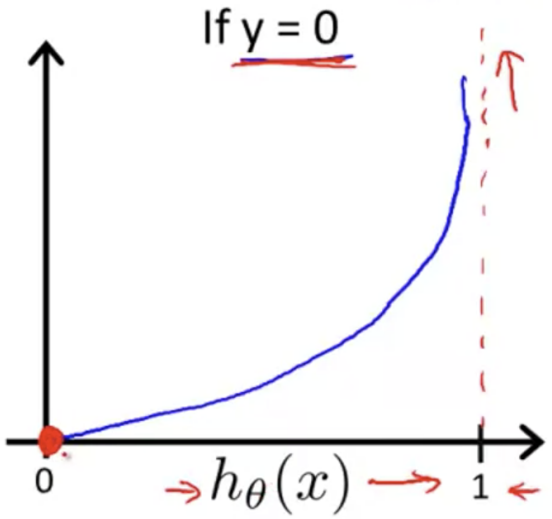


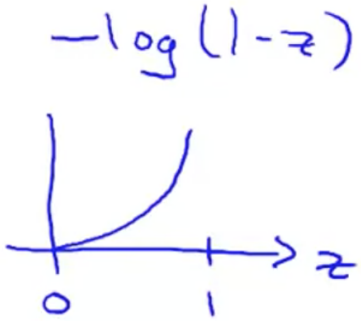

$\text{Cost} = 0$ if $y = 1, h_\theta(x) = 1$
But as $h_\theta(x) \to 0$
$Cost \to \infty$

Captures intuition that if $h_\theta(x) = 0$, (predict $P(y = 1|x; \theta) = 0$), but $y = 1$, we'll penalize learning algorithm by a very large cost.

## Logistic regression cost function

$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$




quiz:

In logistic regression, the cost function for our hypothesis outputting (predicting) $h_\theta(x)$ on a training example that has label $y \in \{0, 1\}$ is:

$$\text{cost}(h_\theta(x), y) = \begin{cases} -\log h_\theta(x) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$

Which of the following are true? Check all that apply.

- [ ] If $h_\theta(x) = y$, then $\text{cost}(h_\theta(x), y) = 0$ (for $y = 0$ and $y = 1$).
- [x] If $y = 0$, then $\text{cost}(h_\theta(x), y) \to \infty$ as $h_\theta(x) \to 1$.
- [ ] If $y = 0$, then $\text{cost}(h_\theta(x), y) \to \infty$ as $h_\theta(x) \to 0$.
- [ ] Regardless of whether $y = 0$ or $y = 1$, if $h_\theta(x) = 0.5$, then $\text{cost}(h_\theta(x), y) > 0$.

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.

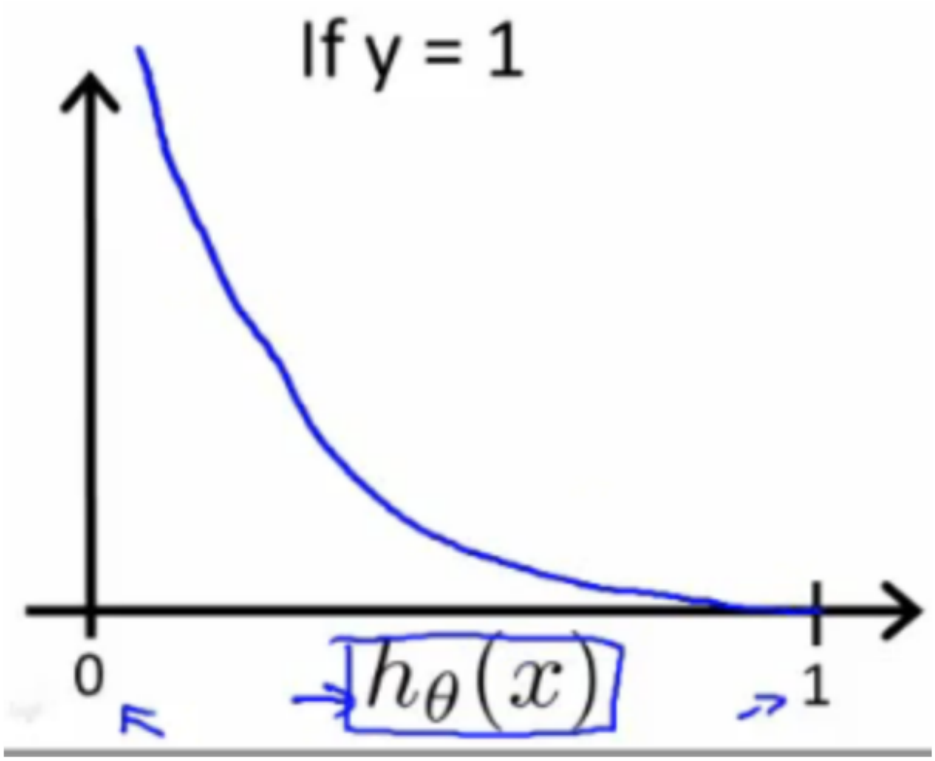
Instead, our cost function for logistic regression looks like:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

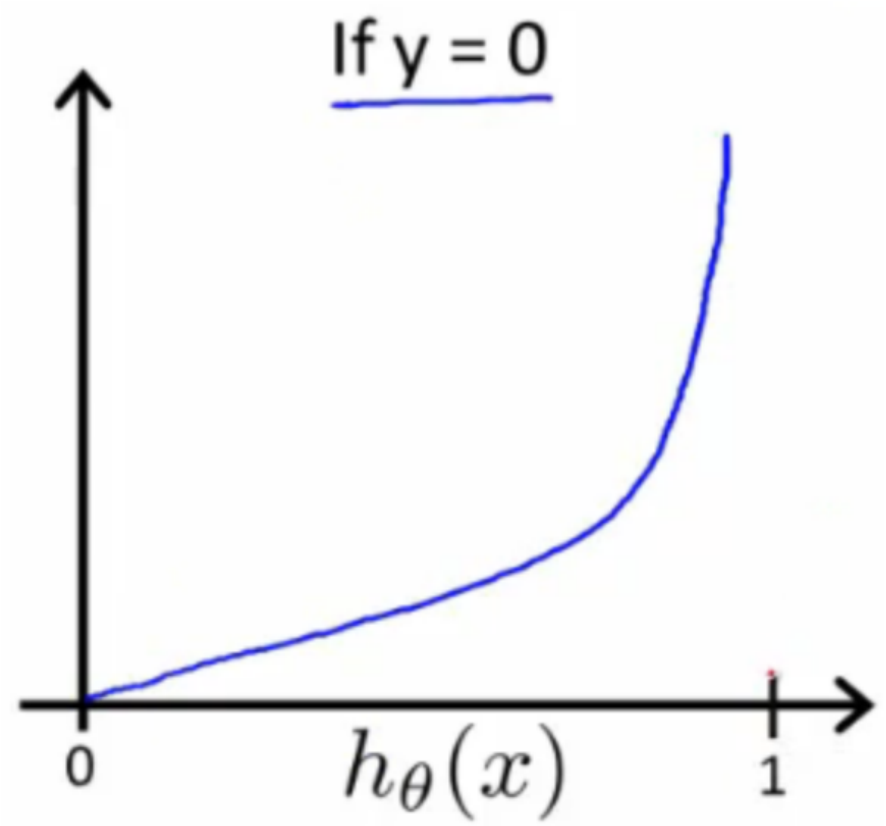
$$\text{Cost}(h_\theta(x), y) = -\log(h_\theta(x)) \qquad \text{if y} = 1$$

$$\text{Cost}(h_\theta(x), y) = -\log(1 - h_\theta(x)) \qquad \text{if y} = 0$$

When y = 1, we get the following plot for $J(\theta)$ vs $h_\theta(x)$:



Similarly, when y = 0, we get the following plot for $J(\theta)$ vs $h_\theta(x)$:




$$\text{Cost}(h_\theta(x), y) = 0 \text{ if } h_\theta(x) = y$$
$$\text{Cost}(h_\theta(x), y) \to \infty \text{ if } y = 0 \text{ and } h_\theta(x) \to 1$$
$$\text{Cost}(h_\theta(x), y) \to \infty \text{ if } y = 1 \text{ and } h_\theta(x) \to 0$$

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

## Logistic regression cost function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \text{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$= -\frac{1}{m}[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log (1 - h_\theta(x^{(i)}))]$$

To fit parameters $\theta$:

$$\min_\theta J(\theta)$$

Get $\theta$

To make a prediction given new $x$:

Output $h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$ $\quad p(y=1 \mid x;\theta)$

## Gradient Descent

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

Want $\min_\theta J(\theta)$:

Repeat $\{$

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

(simultaneously update all $\theta_j$)

$\}$

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

We can fully write out our entire cost function as follows:

## Gradient Descent

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

Want $\min_\theta J(\theta)$:

Repeat $\{$

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

(simultaneously update all $\theta_j$)

$\}$

Notice that this algorithm is identical to the one we used in linear regression. We still have to simultaneously update all values in theta.

A vectorized implementation is:

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

One iteration of gradient descent simultaneously performs these updates:

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

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

$\vdots$

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

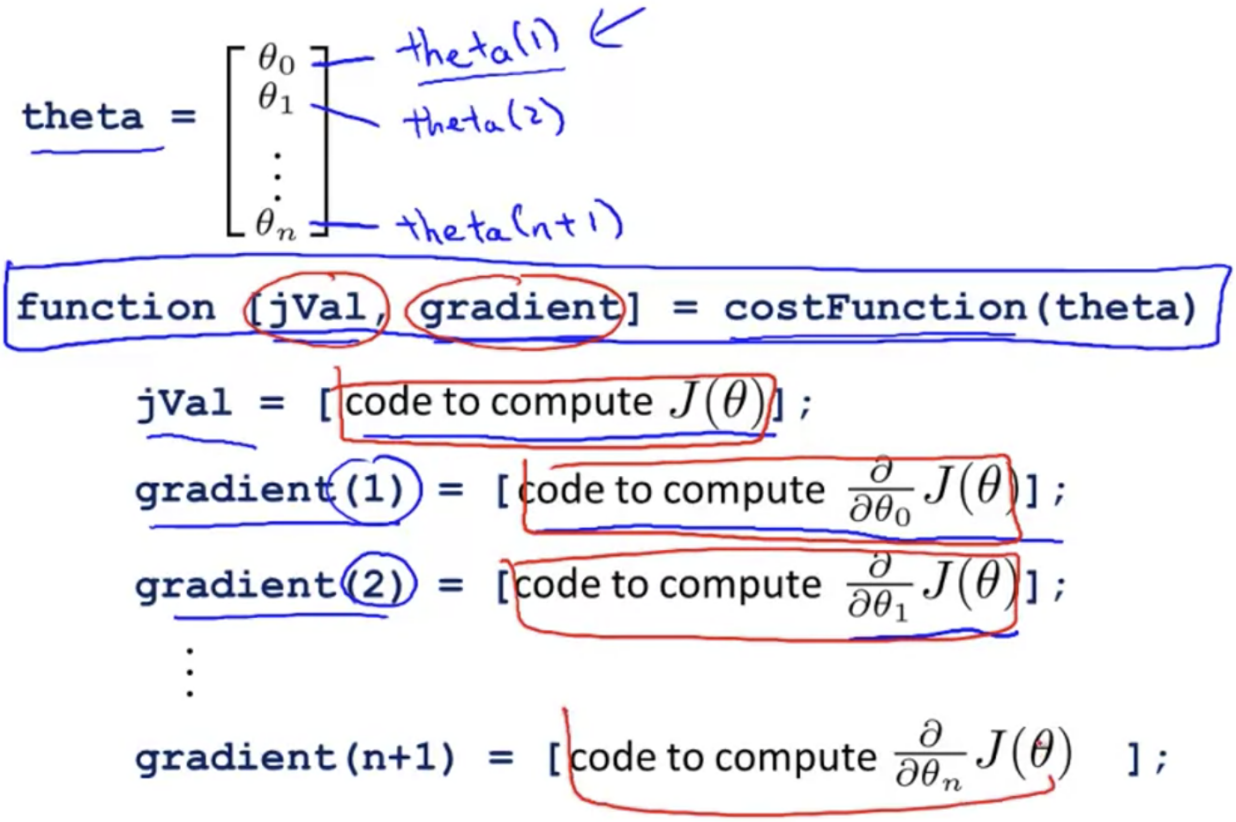
We would like a vectorized implementation of the form $\theta := \theta - \alpha\delta$ (for some vector $\delta \in \mathbb{R}^{n+1}$).

What should the vectorized implementation be?

- (selected) $\theta := \theta - \alpha \frac{1}{m} \sum_{i=1}^{m} [(h_\theta(x^{(i)}) - y^{(i)}) \cdot x^{(i)}]$
- $\theta := \theta - \alpha \frac{1}{m} [\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})] \cdot x^{(i)}$
- $\theta := \theta - \alpha \frac{1}{m} x^{(i)} [\sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})]$
- All of the above are correct implementations.

✓ **Correct**

## Advanced Optimization


theta =
$\theta_0$
$\theta_1$
$\theta_n$
theta(1)
theta(2)
theta(n+1)
function [jVal, gradient] = costFunction(theta)
jVal = [code to compute $J(\theta)$];
gradient(1) = [code to compute $\frac{\partial}{\partial\theta_0}J(\theta)$];
gradient(2) = [code to compute $\frac{\partial}{\partial\theta_1}J(\theta)$];
gradient(n+1) = [code to compute $\frac{\partial}{\partial\theta_n}J(\theta)$ ];

Suppose you want to use an advanced optimization algorithm to minimize the cost function for logistic regression with parameters $\theta_0$ and $\theta_1$. You write the following code:

```
function [jVal, gradient] = costFunction(theta)
  jVal = % code to compute J(theta)
  gradient(1) = CODE#1 % derivative for theta_0
  gradient(2) = CODE#2 % derivative for theta_1
```

What should CODE#1 and CODE#2 above compute?

- ○ CODE#1 and CODE#2 should compute $J(\theta)$.
- ○ CODE#1 should be theta(1) and CODE#2 should be theta(2).
- ◉ CODE#1 should compute $\frac{1}{m}\sum_{i=1}^{m}[(h_\theta(x^{(i)})-y^{(i)})\cdot x_0^{(i)}](=\frac{\partial}{\partial\theta_0}J(\theta))$, and

  CODE#2 should compute $\frac{1}{m}\sum_{i=1}^{m}[(h_\theta(x^{(i)})-y^{(i)})\cdot x_1^{(i)}](=\frac{\partial}{\partial\theta_1}J(\theta))$.
- ○ None of the above.

✓ **Correct**

Summary:

"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.

We first need to provide a function that evaluates the following two functions for a given input value θ:

$$J(\theta)$$
$$\frac{\partial}{\partial\theta_j} J(\theta)$$

We can write a single function that returns both of these:

```
function [jVal, gradient] = costFunction(theta)
  jVal = [...code to compute J(theta)...];
  gradient = [...code to compute derivative of J(theta)...];
end
```

Then we can use octave's "fminunc()" optimization algorithm along with the "optimset()" function that creates an object containing the options we want to send to "fminunc()". (Note: the value for MaxIter should be an integer, not a character string - errata in the video at 7:30)

```
options = optimset('GradObj', 'on', 'MaxIter', 100);
initialTheta = zeros(2,1);
   [optTheta, functionVal, exitFlag] = fminunc(@costFunction, initialTheta, options);

```

We give to the function "fminunc()" our cost function, our initial vector of theta values, and the "options" object that we created beforehand.

## Multiclass Classification: One-vs-all

How to get Logistic Regression to work for Multi-class classification: One-vs-all

Now we will approach the classification of data when we have more than two categories. Instead of y = {0, 1}, we will expend our defination so that y = {0, 1, 2...n}.

Since y = {0, 1... n} we divide our problem into n+1 (+1 because the index starts at 0) binary classification problems; in each one, we predict the probability that 'y' is a member of one of our classes.

# The Problem of Overfitting

Overfitting: **if we have too many features, the learned hypothesis may fit the training set very well, but fail to generalize to new examples(predict prices on new examples.**

Addressing overfitting: 1.Reduce number of features.

- Manually select which features to keep.
- Model selection algorithm

2.Regularization

- Keep all the features, but reduce magnitude \ values of parameters theta J.
- Works well when we have a lot of features, each of which contributes a bit to predicting y.

## Regularization Cost function

Summary

If we have overfitting from our hypothesis function, we can reduce the weight that some of the terms in our function carry by increasing their cost.

Say we wnated to make the following function more quadratic[二次的]:

Say we wanted to make the following function more quadratic:

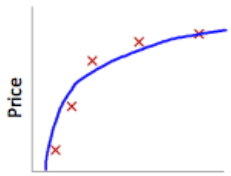
$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

We'll want to eliminate the influence of $\theta_3 x^3$ and $\theta_4 x^4$ . Without actually getting rid of these features or changing the form of our hypothesis, we can instead modify our **cost function**:

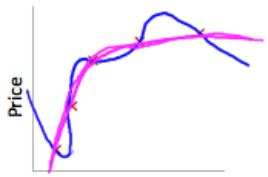
$min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + 1000 \cdot \theta_3^2 + 1000 \cdot \theta_4^2$

We've added two extra terms at the end to inflate the cost of $\theta_3$ and $\theta_4$. Now, in order for the cost function to get close to zero, we will have to reduce the values of $\theta_3$ and $\theta_4$ to near zero. This will in turn greatly reduce the values of $\theta_3 x^3$ and $\theta_4 x^4$ in our hypothesis function. As a result, we see that the new hypothesis (depicted by the pink curve) looks like a quadratic function but fits the data better due to the extra small terms $\theta_3 x^3$ and $\theta_4 x^4$.

**Intuition**




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




$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

**Suppose we penalize and make $\theta_3, \theta_4$ really small.**

$$\rightarrow \min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + 1000\,\theta_3^2 + 1000\,\theta_4^2$$

$\theta_3 \approx 0 \quad \theta_4 \approx 0$

We could also regularize all of our theta parameters in a single summation as:

$$min_\theta \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2$$

The λ, or lambda, is the **regularization parameter**. It determines how much the costs of our theta parameters are inflated.

Using the above cost function with the extra summation, we can smooth the output of our hypothesis function to reduce overfitting. If lambda is chosen to be too large, it may smooth out the function too much and cause underfitting. Hence, what would happen if $\lambda = 0$ or is too small ?

# Regularized linear regression

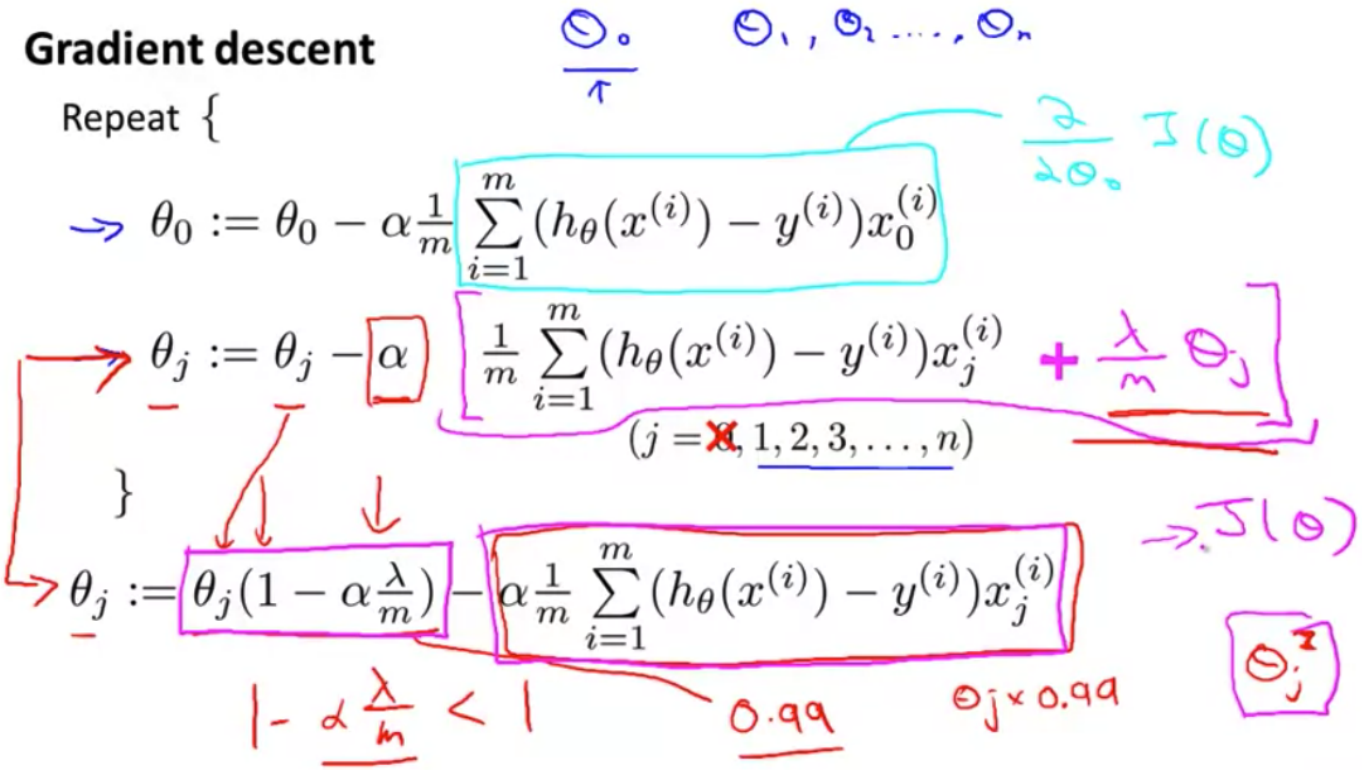
Suppose you are doing gradient descent on a training set of $m > 0$ examples, using a fairly small learning rate $\alpha > 0$ and some regularization parameter $\lambda > 0$. Consider the update rule:

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

Which of the following statements about the term $(1 - \alpha\frac{\lambda}{m})$ must be true?

- $1 - \alpha\frac{\lambda}{m} > 1$
- $1 - \alpha\frac{\lambda}{m} = 1$
- $1 - \alpha\frac{\lambda}{m} < 1$ (selected)
- None of the above.

✓ Correct




We can apply regularization to both linear regression and logistic regression. We will approach linear regression first.

**Gradient Descent:**

### Gradient Descent

We will modify our gradient descent function to separate out $\theta_0$ from the rest of the parameters because we do not want to penalize $\theta_0$.

Repeat {

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

$$\theta_j := \theta_j - \alpha \left[ \left( \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \right) + \frac{\lambda}{m} \theta_j \right] \qquad j \in \{1, 2 \ldots n\}$$

}

The term $\frac{\lambda}{m}\theta_j$ performs our regularization. With some manipulation our update rule can also be represented as:

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

The first term in the above equation, $1 - \alpha\frac{\lambda}{m}$ will always be less than 1. Intuitively you can see it as reducing the value of $\theta_j$ by some amount on every update. Notice that the second term is now exactly the same as it was before.

## Normal Equation

### Normal Equation

Now let's approach regularization using the alternate method of the non-iterative normal equation.

To add in regularization, the equation is the same as our original, except that we add another term inside the parentheses:

$$\theta = \left(X^T X + \lambda \cdot L\right)^{-1} X^T y$$

$$\text{where } L = \begin{bmatrix} 0 & & & & \\ & 1 & & & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \end{bmatrix}$$

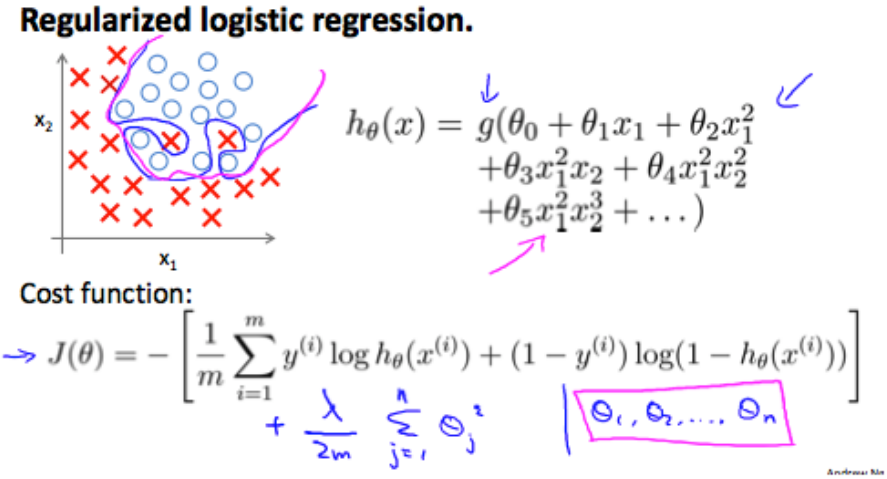
L is a matrix with 0 at the top left and 1's down the diagonal, with 0's everywhere else. It should have dimension (n+1)×(n+1). Intuitively, this is the identity matrix (though we are not including $x_0$), multiplied with a single real number λ.

Recall that if m < n, then $X^T X$ is non-invertible. However, when we add the term λ·L, then $X^T X$ + λ·L becomes invertible.

# 🌟Regularized Logistic Regression

The regularized can help solve the overfitting problems.

We can regularize logistic regression in a similar way that we regularize linear regression. As a result, we can avoid overfitting. The following image shows how the regularized function represented by the blue line:




## Cost Function

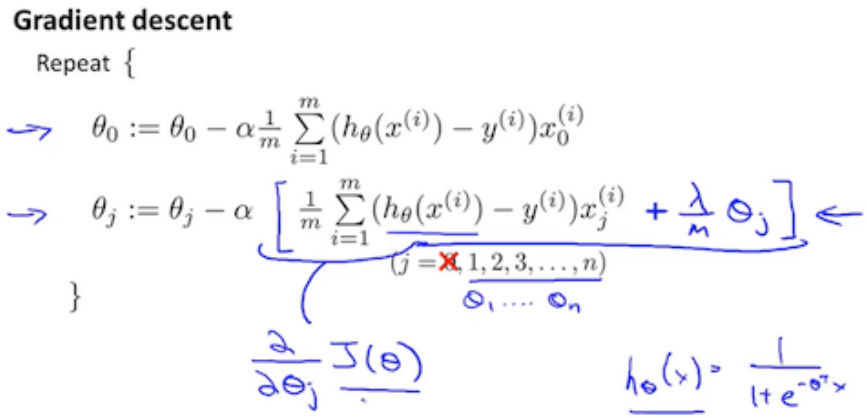
Recall that our cost function for logistic regression was:

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

We can regularize this equation by adding a term to the end:

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\ \log(h_\theta(x^{(i)})) + (1 - y^{(i)})\ \log(1 - h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

The second sum, $\sum_{j=1}^{n}\theta_j^2$ **means to explicitly exclude** the bias term, $\theta_0$. I.e. the θ vector is indexed from 0 to n (holding n+1 values, $\theta_0$ through $\theta_n$), and this sum explicitly skips $\theta_0$, by running from 1 to n, skipping 0. Thus, when computing the equation, we should continuously update the two following equations:



Quiz:

When using regularized logistic regression, which of these is the best way to monitor whether gradient descent is working correctly?

- ○ Plot $-[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))]$ as a function of the number of iterations and make sure it's decreasing.
- ○ Plot $-[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))] - \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$ as a function of the number of iterations and make sure it's decreasing.
- ◉ Plot $-[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$ as a function of the number of iterations and make sure it's decreasing.
- ○ Plot $\sum_{j=1}^{n} \theta_j^2$ as a function of the number of iterations and make sure it's decreasing.

✓ **Correct**