



1. Introduction

1. 机器学习：给电脑学习未经人类编码的能力（1959）

tom mitchell 1998: 机器学习就是从经验e中，学习完成任务t，并有着表现p。如果他在任务t上的表现为p，可以用经验e来优化。

2. 监督学习：教会机器学习

- 回归模型：给予数值让机器模拟出曲线
- 分类模型：预测得恶性肿瘤的概率，预测离散值得概率

3. 非监督学习：让机器自己学习

- 聚类算法，给机器没有标示的数据，让机器判断出不同的cluster，例如google给出某主题的搜索结果
- 鸡尾酒算法 Octave软件

2. 单变量线性回归（梯度算法）

1. 模型分析

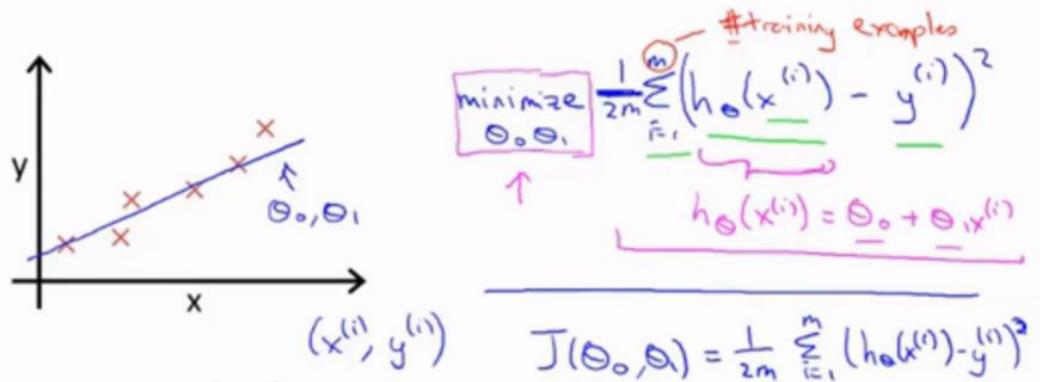
- 回归模型
 - x 输入
 - y 输出
 - h 假设函数，连接x和y

Hypothesis: $h_{\theta}(x) = \theta_0 + \theta_1 x$

θ_i 's: Parameters

- How to choose θ_i 's?

- m 样本数
- j 代价函数 (cost function)



Idea: Choose θ_0, θ_1 so that
 $h_{\theta}(x)$ is close to y for our
training examples (x, y)

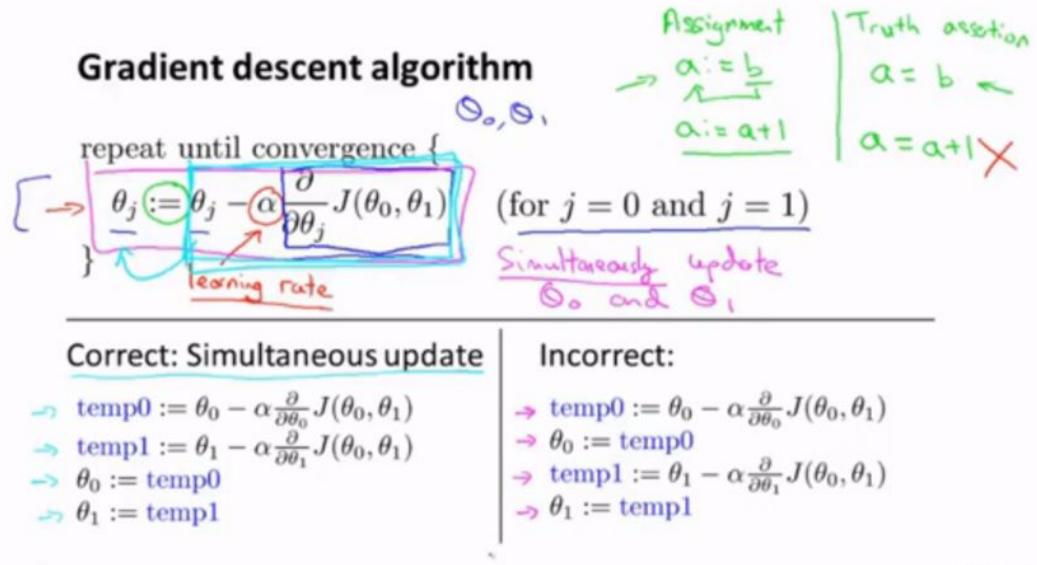
minimize $J(\theta_0, \theta_1)$
Cost Function

Andrew Ng

x, y
cost function is also called the squared
error function or sometimes called the
代价函数也被称作平方误差函数 有时也被称为

-

- 梯度算法



再计算 temp1 再更新 temp1

- 同步梯度算法（通常说的梯度算法）和非同布梯度算法
- 如何理解？
 - 如果J对θ求导>0，说明如果θ增加，J增加。
 - 但是目标是降低J(代价函数)，所以此时θ值应当减小。
 - 减小的程度由α决定。
- 与线性回归算法结合：将j的公式可以带入梯度函数

Gradient descent algorithm

```

repeat until convergence {
     $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$ 
}
102 (for  $j = 1$  and  $j = 0$ )

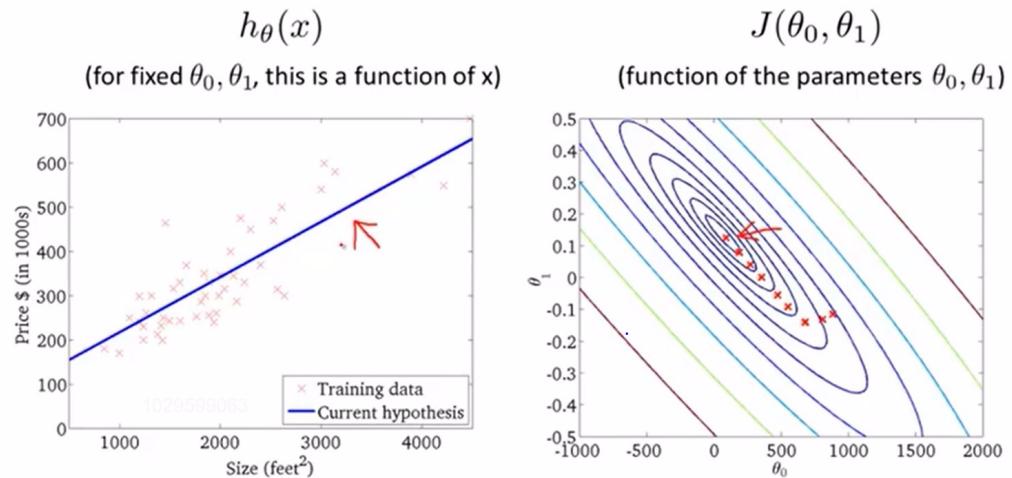
```

Linear Regression Model

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

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

- 解决函数问题如下（convex function：只有一个最低点，cost function 取值最低）



CONVEX Function

3. 矩阵回顾

1. 矩阵向量乘法的应用（任何方程都可等效为矩阵向量乘法）

House sizes:

- $\rightarrow 2104$
- $\rightarrow 1416$
- $\rightarrow 1534$
- $\rightarrow 852$

Matrix x

$$\begin{bmatrix} 1 & 2104 \\ 1 & 1416 \\ 1 & 1534 \\ 1 & 852 \end{bmatrix}$$

4×2

$h_{\theta}(x) = -40 + 0.25x$

$h_{\theta}(x)$

2×1

Vector

$$\begin{bmatrix} -40 \\ 0.25 \end{bmatrix}$$

4×1 matrix

$$\begin{bmatrix} -40 \times 1 + 0.25 \times 2104 \\ -40 \times 1 + 0.25 \times 1416 \\ -40 \times 1 + 0.25 \times 1534 \\ -40 \times 1 + 0.25 \times 852 \end{bmatrix}$$

$h_{\theta}(2104)$

$h_{\theta}(1416)$

$h_{\theta}(1534)$

$h_{\theta}(852)$

$\boxed{\text{prediction}} = \text{Data Matrix} \times \boxed{\text{parameters}}$

for $i = 1:1000$, prediction(i) = ...

Andrew Ng

2. 矩阵矩阵乘法应用

3. 矩阵特征

- 矩阵满足结合率

■ $(A \times B) \times C = A \times (B \times C)$

- 单位矩阵 I

- 通常矩阵不满足 $A \times B \neq B \times A$
 - 但是对于单位矩阵满足上述可逆性

4. 逆矩阵和转置矩阵

- 逆矩阵
 - 满足 $A \times A(-1) = A(-1) \times A = I$ (单位矩阵)
 - 没有逆矩阵的矩阵，可以想象成没有逆数的实数 0 (奇异矩阵)
- 转置矩阵 $A(ij) = A^{(\text{转置})}(ji)$

5. 多元变量线性回归

1. 多元线性回归的矩阵表示

Hypothesis: $\underline{h_{\theta}(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n}$ $\rightarrow x_0 = 1$

Parameters: $\underline{\theta_0, \theta_1, \dots, \theta_n}$ Θ n+1-dimensional vector

Cost function: $J(\theta_0, \theta_1, \dots, \theta_n) = \underline{J(\Theta)}$ $\frac{1}{2m} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$

Gradient descent:

Repeat {
 $\rightarrow \theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \dots, \theta_n)$ $J(\Theta)$
} ↑ (simultaneously update for every $j = 0, \dots, n$)

Andrew Ng

2. 图解 (结合下面三者)

- 线性回归方程

■ $\underline{h_{\theta}(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n}$ $\rightarrow x_0 = 1$

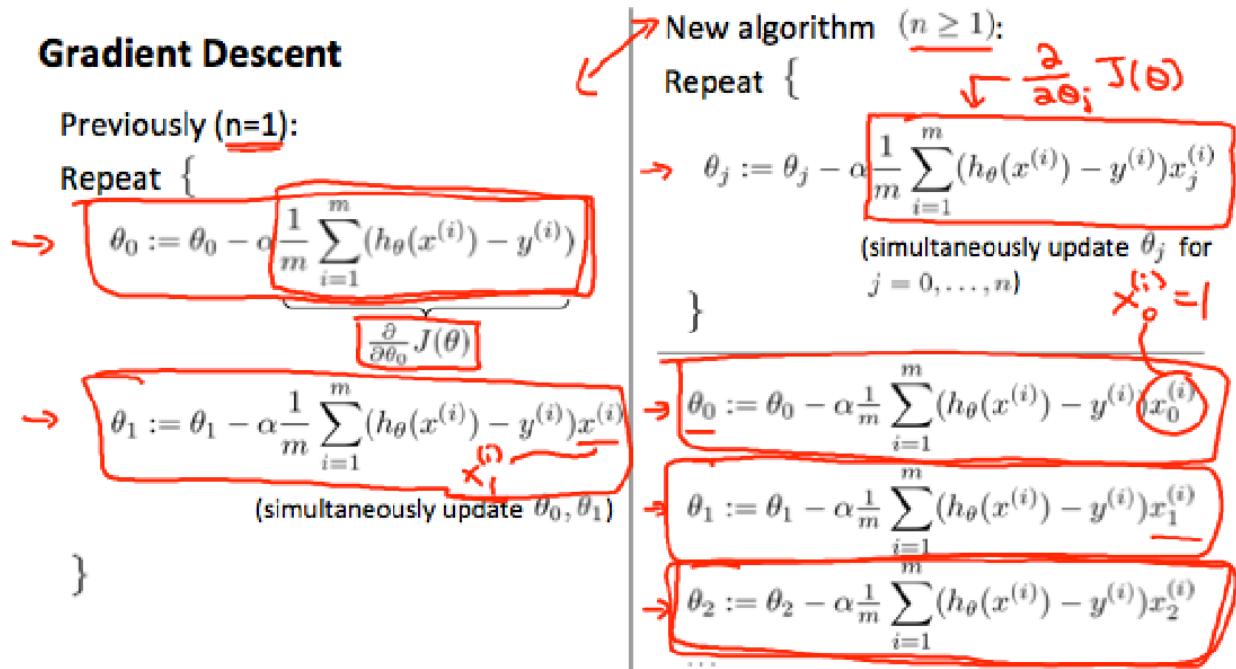
- Cost Function

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

- 梯度下降算法

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

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



此处对J(θ)进行求导带入的,因为是对θn求导, 所以乘以2后再乘以x^(n)。

```

1 def computeCost(X, y, theta):
2     inner = np.power(((X * theta.T) - y), 2)
3     return np.sum(inner) / (2 * len(X))

```

3. 多元梯度下降 I 特征缩放

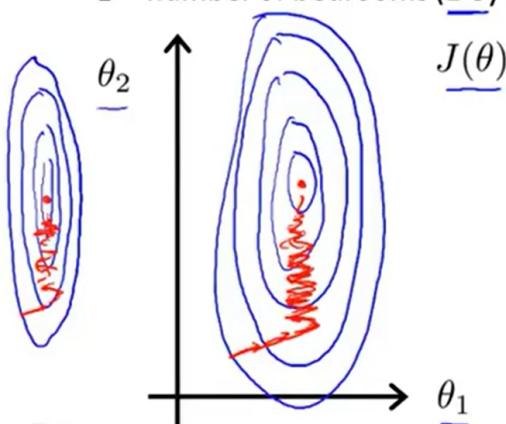
- 特征缩放
 - 除以size , 使特征值 x_j 范围尽量接近 $-1 \leq x_j \leq 1$

Feature Scaling

Idea: Make sure features are on a similar scale.

E.g. $x_1 = \text{size (0-2000 feet}^2)$

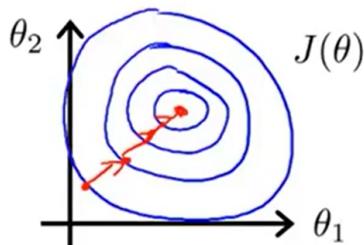
$x_2 = \text{number of bedrooms (1-5)}$



$$\rightarrow x_1 = \frac{\text{size (feet}^2)}{2000}$$

$$\rightarrow x_2 = \frac{\text{number of bedrooms}}{5}$$

$$0 \leq x_1 \leq 1 \quad 0 \leq x_2 \leq 1$$



Andrew Ng

- 平均归一化

- $(\text{size} - \text{均值}) \div \text{总范围}$
- 限制 x_j 在 $-0.5 \leq x_j \leq 0.5$

Mean normalization

Replace x_i with $x_i - \mu_i$ to make features have approximately zero mean
(Do not apply to $x_0 = 1$).

E.g. $\rightarrow x_1 = \frac{\text{size}-1000}{2000}$

Average size = 100

$$x_2 = \frac{\#\text{bedrooms}-2}{5}$$

1-5 bedrooms

$$[-0.5 \leq x_1 \leq 0.5] \quad [-0.5 \leq x_2 \leq 0.5]$$

$$x_1 \leftarrow \frac{x_1 - \mu_1}{\sigma_1}$$

avg value of x_1 in training set
 range ($\max - \min$)
 (or standard deviation)

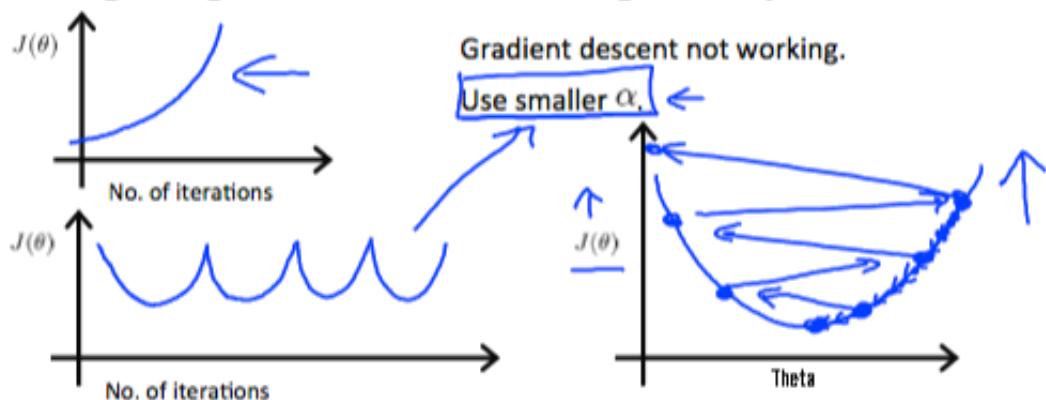
$$x_2 \leftarrow \frac{x_2 - \mu_2}{\sigma_2}$$

Andrew Ng

4. 多元梯度下降 II 学习率 α

It has been proven that if learning rate α is sufficiently small, then $J(\theta)$ will decrease on every iteration.

Making sure gradient descent is working correctly.



- For sufficiently small α , $J(\theta)$ should decrease on every iteration.
- But if α is too small, gradient descent can be slow to converge.

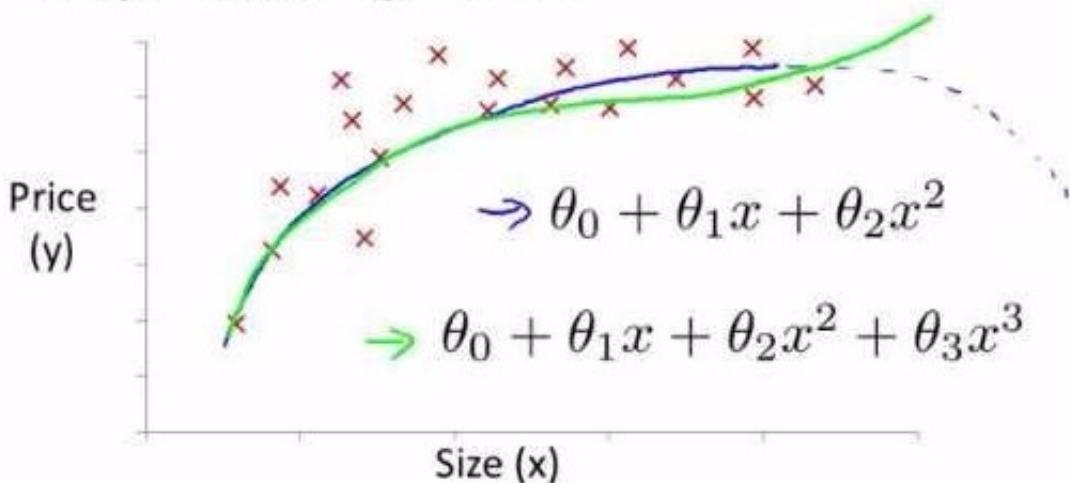
- To summarize:

- If α is too small: slow convergence.
- If α is too large: may not decrease on every iteration and thus may not converge.

5. 特征和多项式回归

- 可以根据函数的图像信息，来选择特征值的类型
- 意思是，有可能满足房价的拟合为直线，或二次方程，或三次方程，或根号方程。根据图像信息找到最合适的拟合方程。
- 如果采用多项式拟合，为了简便。在运行梯度下降算法之前需要进行特征缩放。

Polynomial regression



6. 正规化 (normalization)

- 倒数为零，快速找到拐点
- 用矩阵表示
 - x 表示不同的 figure (特征值)
 - y 是价格

Examples: $m = 4$.

	Size (feet ²)	Number of bedrooms	Number of floors	Age of home (years)	Price (\$1000)
x_0	x_1	x_2	x_3	x_4	y
1	2104	5	1	45	460
1	1416	3	2	40	232
1	1534	3	2	30	315
1	852	2	1	36	178

→ $X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}$ $y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$

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

Andrew Ng

- 运用正规方程求解参数 θ

$$\left(\begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \times \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}^{-1} \times \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \times \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix} \right)$$

Octave 中，正规方程写作：

```

1 pinv(X'*X)*X'*y
2

```

- 为什么 θ 取值为上述时，可以得到最小值？

- 答： θ 为 h 方程的系数！

$$h_{\theta}(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots$$

∴ 矩阵形式

$$X = \begin{bmatrix} x_0^{(1)} & x_1^{(1)} & x_2^{(1)} & \dots \\ x_0^{(2)} & x_1^{(2)} & \dots & \dots \\ \vdots & & & \\ x_0^{(m)} & \dots & x_n^{(m)} & \end{bmatrix} \quad \theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_m \end{bmatrix}$$

$$J(\theta_0, \dots, \theta_m) = \frac{1}{m} \sum \dots$$

$$\frac{\partial J}{\partial \theta_j} = \dots = 0 \quad \text{用梯度下降} \rightarrow \text{solve } \theta_0, \dots, \theta_n$$

$\therefore (X^T \cdot X)^{-1} \cdot X^T \cdot y$ 为闭式
矩阵 $n \times n$ $n \times m$ $m \times 1$
 $n \times 1$

∴ 对于正常方程 (将矩阵对地正规方程会更容易理解)

$$X \cdot \theta = y \Rightarrow \theta = \frac{1}{X} y$$

对于矩阵来说如何求 $\frac{1}{X}$, 并不是所有 X 都可以求逆

- X 的表示方法如下

m examples $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})$; n features.

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ x_2^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \in \mathbb{R}^{n+1} \quad | \quad X = \begin{bmatrix} (x^{(1)})^T \\ (x^{(2)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}$$

(design matrix)

E.g. If $x^{(i)} = \begin{bmatrix} 1 \\ x_1^{(i)} \end{bmatrix}$

$$X = \begin{bmatrix} 1 & x_1^{(1)} \\ 1 & x_1^{(2)} \\ \vdots & \vdots \\ 1 & x_1^{(m)} \end{bmatrix} \quad | \quad \underline{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

$m \times (n+1)$

X 转置乘以 X 取逆 乘以 X 转置 乘以 y

Andrew Ng

- 梯度下降和正规算法的优劣

- n 大于 1000 的建议用梯度下降，因为计算量的原因
- 对于那些不可逆的矩阵（通常是因为特征之间不独立，如同时包含英尺为单位的尺寸和米为单位的尺寸两个特征，也有可能是特征数量大于训练集的数量），正规方程方法是不能用的。

m training examples, n features.

<u>Gradient Descent</u>	<u>Normal Equation</u>
→ • Need to choose α .	→ • No need to choose α .
→ • Needs many iterations.	→ • Don't need to iterate.
• Works well even when n is large.	• Need to compute $(X^T X)^{-1}$ $\frac{n \times n}{n \times n} \mathcal{O}(n^3)$
$n = 10^6$	• Slow if n is very large.
	$n = 100$ $n = 1000$ --- $n = 10000$

Andrew Ng

7. 如果 $X^T X$ 为不可逆的情况

- 减少线性相关的 Features(特征值)
- 减少特征值 ($m \leq n$)

What if $X^T X$ is non-invertible?

- Redundant features (linearly dependent).

E.g. $\begin{cases} x_1 = \text{size in feet}^2 \\ x_2 = \text{size in m}^2 \end{cases}$

$$1m = 3.28 \text{ feet}$$

$$\begin{cases} x_1 = (3.28)^2 x_2 \end{cases}$$

$$\begin{array}{l} \rightarrow n = 10 \leftarrow \\ \rightarrow n = 100 \leftarrow \end{array}$$

$$\Theta \in \mathbb{R}^{10 \times 100}$$

- Too many features (e.g. $m \leq n$).

- Delete some features, or use regularization.

↓ later

Andrew Ng

- 利用 octave 中 pinv 函数，不用 inv 函数
- 注意：总结一下，只要特征变量的数目并不大，正规方程(倒数为 0)是一个很好的计算参数的替代方

法。具体地说，只要特征变量数量小于一万，我通常使用标准方程法，而不使用梯度下降法。

随着我们要讲的学习算法越来越复杂，例如，当我们讲到分类算法，像逻辑回归算法，我们会看到，实际上对于那些算法，并不能使用方程法。对于那些更复杂的学习算法，我们将不得不仍然使用梯度下降法。因此，梯度下降法是一个非常有用的算法，可以用在有大量特征变量的线性回归问题。或者我们以后在课程中，会讲到的一些其他的算法，因为标准方程法不适合或者不能用在它们上。但对于这个特定的线性回归模型，标准方程法是一个比梯度下降法更快的替代算法。所以，根据具体的问题，以及你的特征变量的数量，这两种算法都是值得学习的。(标准方程就是正规方程)

- 来源：<http://www.ai-start.com/ml2014/html/week2.html>

```
1 正规方程的python实现
2
3 import numpy as np
4
5 def normalEqn(X, y):
6     theta = np.linalg.inv(X.T@X)@X.T@y      #X.T@X等价于X.T.dot(X)
7
8     return theta
```

- 补充：

备注：本节最后我把推导过程写下。

有些同学曾经问我，当计算 $=\text{inv}(X'X)X'y$ ，那对于矩阵的结果是不可逆的情况咋办呢？如果你懂一点线性代数的知识，你或许会知道，有些矩阵可逆，而有些矩阵不可逆。我们称那些不可逆矩阵为奇异或退化矩阵。问题的重点在于的不可逆的问题很少发生，在Octave里，如果你用它来实现的计算，你将会得到一个正常的解。在Octave里，有两个函数可以求解矩阵的逆，一个被称为`pinv()`，另一个是`inv()`，这两者之间的差异是些许计算过程上的，一个是所谓的伪逆，另一个被称为逆。使用`pinv()`函数可以展现数学上的过程，这将计算出的值，即便矩阵是不可逆的。

在`pinv()`和`inv()`之间，又有哪些具体区别呢？

其中`inv()`引入了先进的数值计算的概念。例如，在预测住房价格时，如果是以英尺为尺寸规格计算的房子，是以平方米为尺寸规格计算的房子，同时，你也知道1米等于3.28英尺(四舍五入到两位小数)，这样，你的这两个特征值将始终满足约束：。实际上，你可以用这样的一个线性方程，来展示那两个相关联的特征值，矩阵将是不可逆的。

第二个原因是，在你想用大量的特征值，尝试实践你的学习算法的时候，可能会导致矩阵的结果是不可逆的。具体地说，在小于或等于n的时候，例如，有等于10个的训练样本也有等于100的特征数量。要找到适合的维参数矢量，这将会变成一个101维的矢量，尝试从10个训练样本中找到满

是101个参数的值，这工作可能会让你花上一阵子时间，但这并不总是一个好主意。因为，正如我们所看到你只有10个样本，以适应这100或101个参数，数据还是有些少。

稍后我们将看到，如何使用小数据样本以得到这100或101个参数，通常，我们会使用一种叫做正则化的线性代数方法，通过删除某些特征或者是使用某些技术，来解决当比小的时候的问题。即使你有一个相对较小的训练集，也可使用很多的特征来找到很多合适的参数。总之当你发现的矩阵的结果是奇异矩阵，或者找到的其它矩阵是不可逆的，我会建议你这么做。

首先，看特征值里是否有一些多余的特征，像这些和是线性相关的，互为线性函数。同时，当有一些多余的特征时，可以删除这两个重复特征里的其中一个，无须两个特征同时保留，将解决不可逆性的问题。因此，首先应该通过观察所有特征检查是否有多余的特征，如果有多余的就删除掉，直到他们不再是多余的为止，如果特征数量实在太多，我会删除些用较少的特征来反映尽可能多内容，否则我会考虑使用正规化方法。如果矩阵是不可逆的，（通常来说，不会出现这种情况），如果在Octave里，可以用伪逆函数`pinv()`来实现。这种使用不同的线性代数库的方法被称为伪逆。即使的结果是不可逆的，但算法执行的流程是正确的。总之，出现不可逆矩阵的情况极少发生，所以在大多数实现线性回归中，出现不可逆的问题不应该过多的关注是不可逆的。

- 增加内容：

的推导过程：

其中：

将向量表达形式转为矩阵表达形式，则有，其中为行列的矩阵（为样本个数，为特征个数），为行1列的矩阵，为行1列的矩阵，对进行如下变换

接下来对偏导，需要用到以下几个矩阵的求导法则：

所以有：

令，

则有

6. Octave Tutorial

1. Simple variable

Variable		
<code>~=</code>	不等于	
<code>==</code>	等于	
<code>&&</code>	与	
<code> </code>	或	
<code>PS1('>>');</code>	改变提示符	
<code>pi</code>	3.14...。	
<code>a = 'hi';</code>	赋值，不输出结果	
<code>disp(sprintf('2 decimals: %0.2f', a))</code>	输出：2 decimals: 3.14	

2. Matrix generation

Matrix	
<code>A = [1 2; 3 4; 5 6]</code>	矩阵 3x2
<code>v = 1:0.1:2</code>	[1 1.1 1.22]
<code>2 * ones(2,3)</code>	2乘以单位矩阵2x3
<code>rand(2,3)</code>	随机生成矩阵2x3
<code>w = randn(2,3)</code>	高斯分布，均值0，标准差方差为1
<code>hist (w,50)</code>	直方图，横坐标50，纵坐标w
<code>eye(4), flipud (eye(4))</code>	4x4单位矩阵，垂直翻转单位矩阵
<code>help eye</code>	帮助

3. Complex functions

<code>size (A)</code>	矩阵维数
<code>size (A,2)</code>	A矩阵的列数，(A, 1) 为A矩阵的行数
<code>pwd</code>	当前路径

cd 'C://mncui'	
ls	
load filename / load('filename')	加载, 加载之后可以直接输入文件名 查看文件内容
who	输出filename文件的变量
whos	更精确的输出
clear filename (clear后不加变量)	删除加载filename (删除所有已加载变量)
v = filename(1:10)	赋值前10个数据给v
save newfilename.mat v;	将v保存为newfilename.mat文件
save newfilename.txt v -ascii	保存成ascii编码文档, 人类可读
A(:,), A(:,1), A(:,2), A(:,2)=[1;2;3]	读取, 赋值
C = [A, B] , C = [A; B]	B加在A的右边, B加在A的下面

4. Load and store data

A * B, A .*B	矩阵乘法, 矩阵点乘
A.^2	每个元素平方
1 ./ A	每个元素倒数
A'	转置
A < 3, find(A<3)	判断每个元素是否小于3 (1/0) , 输出满足条件的为第几条
sum(A), prod(A), floor(A), ceil(A) sum(A,1), sum(A,2)	对每一个元素求和, 乘积, 向下取整, 向上取整 对A每一列求和, 每一行求和
max(A,[],1), max(A,[],2)	A每列最大值, 每行最大值
sum(sum(A .* eye(9)))	求对角线和

pinv(A)	逆矩阵

5. Visualize matrix using Octave

plot (x ,y, 'r')	绘制图形, 红色
hold on;	在原来图像中继续绘图
xlabel('time') ylabel('time')	标签
legend (' sin', 'cos')	图例
title('my plot')	
print -dpng 'myPlot.png'	
subplot(1,2,1); plot(x,y1); subplot(1,2,2); plot(x,y2);	1x2 grid显示图片
axis([0.5 1 -1 1])	调整x, y轴显示区间
imagesc(magic(15)), colorbar, colormap gray;	绘制灰度图像

6. 控制语句 if which while

for i=1:10, sentences; end;	sentences前面的空格没有意义
i = 1; while i <= 5, sentence; end;	
while true, sentences; if i == 6, break; end;	

end;	
<pre>if a==1, disp('sentences'); elseif a==2, disp('sentences2'); else disp('sentences3'); end;</pre>	
addpath('C:\Users\mnncui....')	Octave会自动搜索该目录下的函数
functionname(5)	函数名（x值）

6. defining function Using Octave

```
1 function [y1,y2] = functionname(x) %没有符号，自动识别自变量x，因变量y
2 y1 = x^2; %分号
3 y2 = x^3;
```

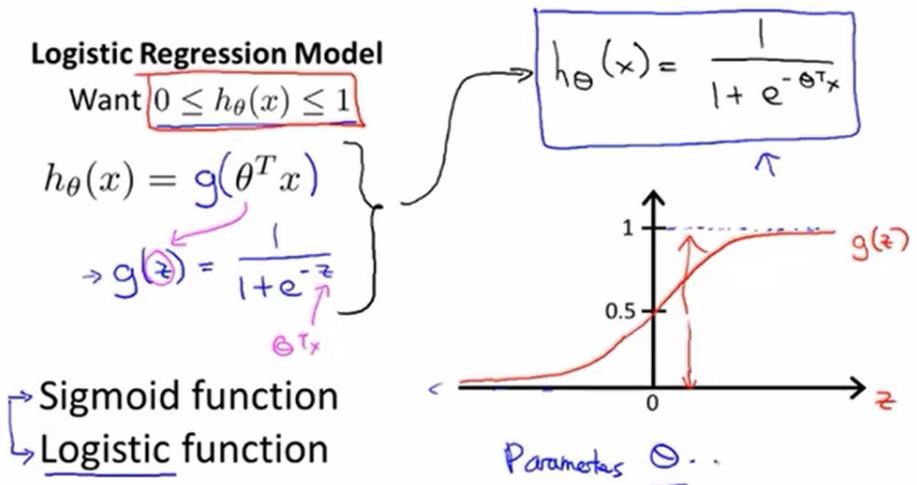
7. Logistic 回归

1. 分类

- 结果只有 0 or 1
- 用线性回归的话，会出现数据越多越不准确的效果

2. 假设陈述

- logistic 回归模型
 - σ函数 使函数范围 在0~1之间



Andrew Ng

◦ 假设输出

- 条件概率：在x参数取θ时，y=1的概率
- $P(y=1 | x; \theta) + P(y=0 | x; \theta) = 1$

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 θ ”

$$y = 0 \text{ or } 1$$

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

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

Andrew Ng

3. 寻找边界条件

- 根据σ函数确定公式，
- 知道θ就可以确定边界

Logistic regression

$$\rightarrow h_{\theta}(x) = g(\theta^T x) = P(y=1|x; \theta)$$

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

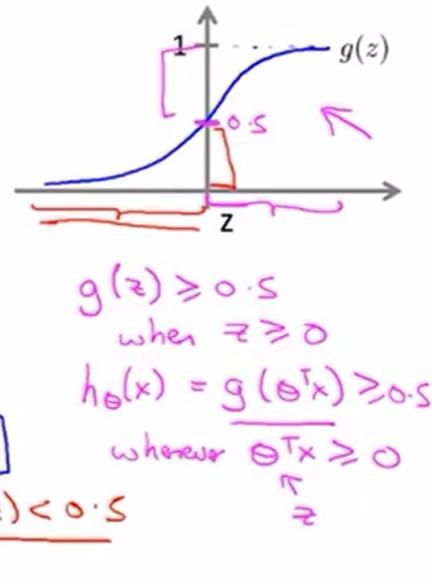
Suppose predict " $y = 1$ " if $h_{\theta}(x) \geq 0.5$

$$\rightarrow \theta^T x \geq 0$$

predict " $y = 0$ " if $h_{\theta}(x) < 0.5$

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

$$\theta^T x < 0$$



Andrew Ng

4. Cost (代价) 函数的选取

- 为什么重新定义Cost函数?
- 下图是目前cost函数面临的问题:
 - 当把Logistic函数带入平方和公式里会发现，画出来的图像是非凸函数。
 - 这样的问题是梯度算法找到的最低点不一定是global minimum
 - 所以需要对cost函数做优化！

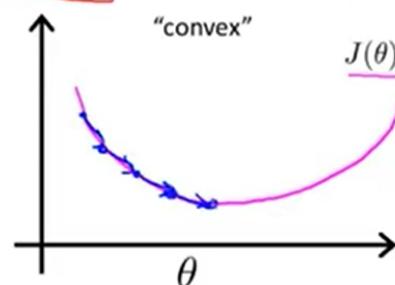
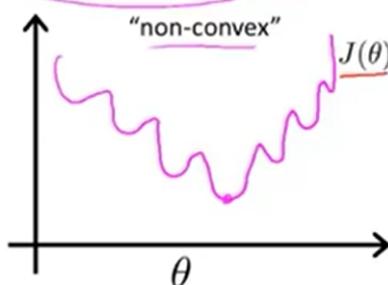
Cost function

$$\rightarrow \text{Linear regression: } J(\theta) = \frac{1}{m} \sum_{i=1}^m \frac{1}{2} (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

$\rightarrow \text{cost}(h_{\theta}(x^{(i)}), y^{(i)})$

$$\rightarrow \text{Cost}(h_{\theta}(x), y) = \frac{1}{2} (h_{\theta}(x) - y)^2$$

$\frac{1}{2} (h_{\theta}(x) - y)^2$



Andrew Ng

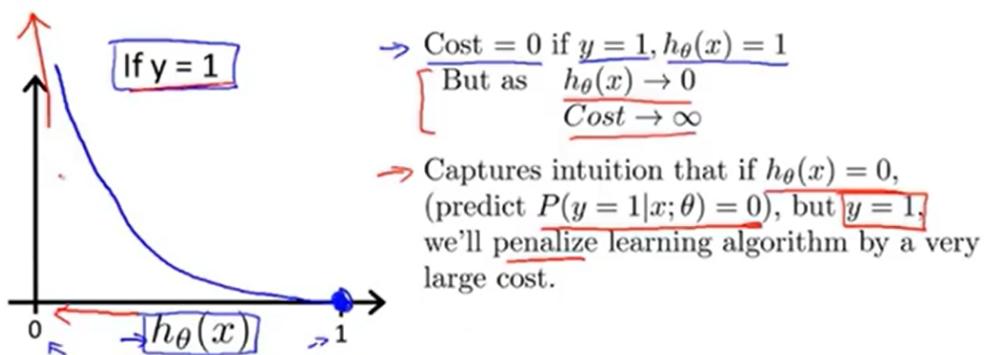
- 重新定义Cost函数
 - 利用对数函数，当y=1时
 - $h=1, \text{ cost} \rightarrow 0$

- $h=0$, $\text{cost} \rightarrow \infty$, 预测错误的代价是无穷大
- 利用对数函数, 当 $y=0$ 时

- $h=0$, $\text{cost} \rightarrow 0$
- $h=1$, $\text{cost} \rightarrow \infty$, 预测错误的代价是无穷大
- 所以定义Cost函数如下:
- $h=1$ 时, 满足与y的关系

Logistic regression cost function

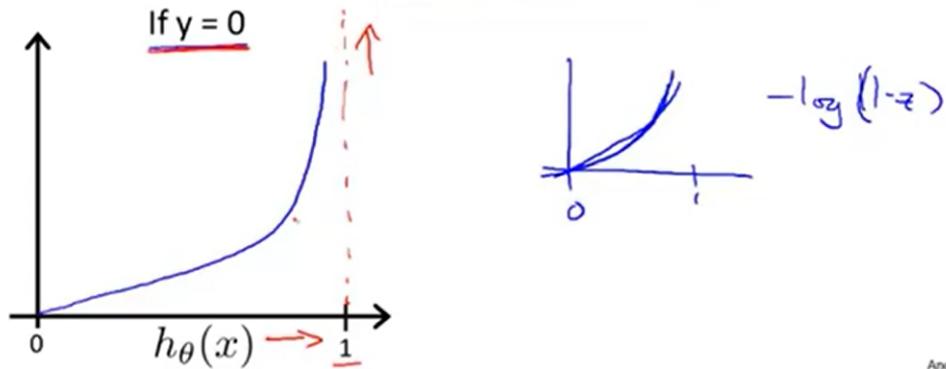
$$\text{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



- $h=0$ 时, 同样满足与y的关系

Logistic regression cost function

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



Andrew Ng

```

1 import numpy as np
2
3 def cost(theta, X, y):
4
5     theta = np.matrix(theta)
6
7     X = np.matrix(X)
8
9     y = np.matrix(y)
10
11    first = np.multiply(-y, np.log(sigmoid(X* theta.T)))
12
13    second = np.multiply((1 - y), np.log(1 - sigmoid(X* theta.T)))
14
15    return np.sum(first - second) / (len(X))

```

5. Cost函数的简化与梯度下降算法

- cost简化为一个公式（包括上面的y=1或0）
- 梯度下降算法仍需要对J求导，得到如下
 - 注意：此时的梯度下降公式和线性回归表面相同
 - 但是h函数构成不同，logistic回归中运用了 σ 函数来表示。

Gradient Descent

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

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 θ_j)

$\Theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$

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

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

Algorithm looks identical to linear regression!

Andrew Ng

- 推导过程：

$$\begin{aligned}
J(\theta) &= -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(h_\theta(x^{(i)})) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))] \text{ 考虑: } h_\theta(x^{(i)}) = \frac{1}{1+e^{-\theta^T x^{(i)}}} \text{ 则:} \\
y^{(i)} \log(h_\theta(x^{(i)})) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)})) &= y^{(i)} \log\left(\frac{1}{1+e^{-\theta^T x^{(i)}}}\right) + (1-y^{(i)}) \log\left(1-\frac{1}{1+e^{-\theta^T x^{(i)}}}\right) \\
&= -y^{(i)} \log\left(1+e^{-\theta^T x^{(i)}}\right) - (1-y^{(i)}) \log\left(1+e^{\theta^T x^{(i)}}\right)
\end{aligned}$$

$$\begin{aligned}
\text{所以: } \frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \left[-\frac{1}{m} \sum_{i=1}^m [-y^{(i)} \log\left(1+e^{-\theta^T x^{(i)}}\right) - (1-y^{(i)}) \log\left(1+e^{\theta^T x^{(i)}}\right)] \right] \\
&= -\frac{1}{m} \sum_{i=1}^m \left[-y^{(i)} \frac{-x_j^{(i)} e^{-\theta^T x^{(i)}}}{1+e^{-\theta^T x^{(i)}}} - (1-y^{(i)}) \frac{x_j^{(i)} e^{\theta^T x^{(i)}}}{1+e^{\theta^T x^{(i)}}} \right] = -\frac{1}{m} \sum_{i=1}^m y^{(i)} \frac{x_j^{(i)}}{1+e^{\theta^T x^{(i)}}} - (1-y^{(i)}) \frac{x_j^{(i)} e^{\theta^T x^{(i)}}}{1+e^{\theta^T x^{(i)}}} \\
&= -\frac{1}{m} \sum_{i=1}^m \frac{y^{(i)} x_j^{(i)} - x_j^{(i)} e^{\theta^T x^{(i)}} + y^{(i)} x_j^{(i)} e^{\theta^T x^{(i)}}}{1+e^{\theta^T x^{(i)}}} = -\frac{1}{m} \sum_{i=1}^m \frac{y^{(i)} (1+e^{\theta^T x^{(i)}}) - e^{\theta^T x^{(i)}}}{1+e^{\theta^T x^{(i)}}} x_j^{(i)} = -\frac{1}{m} \sum_{i=1}^m (y^{(i)} - \frac{e^{\theta^T x^{(i)}}}{1+e^{\theta^T x^{(i)}}}) x_j^{(i)} \\
&= -\frac{1}{m} \sum_{i=1}^m (y^{(i)} - \frac{1}{1+e^{-\theta^T x^{(i)}}}) x_j^{(i)} = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} - h_\theta(x^{(i)})] x_j^{(i)} = \frac{1}{m} \sum_{i=1}^m [h_\theta(x^{(i)}) - y^{(i)}] x_j^{(i)}
\end{aligned}$$

注: 虽然得到的梯度下降算法表面上看上去与线性回归的梯度下降算法一样, 但是这里的 $h_\theta(x) = g(\theta^T X)$ 与线性回归中不同, 所以实际上是不一样的。另外, 在运行梯度下降算法之前, 进行特征缩放依旧是非常必要的。

6. 高级优化

- 优化算法
- 除了梯度下降算法以外, 还有一些常被用来令代价函数最小的算法, 这些算法更加复杂和优越, 而且通常不需要人工选择学习率, 通常比梯度下降算法要更加快速。这些算法有: 共轭梯度 (Conjugate Gradient), 局部优化法(Broyden fletcher goldfarb shann,BFGS)和有限内存局部优化法(LBFGS), fminunc是 matlab 和 octave 中都带的一个最小值优化函数, 使用时我们需要提供代价函数和每个参数的求导, 下面是 octave 中使用 fminunc 函数的代码示例:

```

1 function [jVal, gradient] = costFunction(theta)
2     jVal = [...code to compute J(theta)...];
3     gradient = [...code to compute derivative of J(theta)...];
4 end
5 options = optimset('GradObj', 'on', 'MaxIter', '100');
6 initialTheta = zeros(2,1);
7 [optTheta, functionVal, exitFlag] = fminunc(@costFunction,
initialTheta, options);

```

- Pasted from: <http://www.ai-start.com/ml2014/html/week3.html>

Optimization algorithm

Given θ , we have code that can compute

$$\begin{aligned} & - J(\theta) \\ & - \frac{\partial}{\partial \theta_j} J(\theta) \quad (for j = 0, 1, \dots, n) \end{aligned}$$

Optimization algorithms:

- - Gradient descent
- [- Conjugate gradient
- BFGS
- L-BFGS

Advantages:

- No need to manually pick α
- Often faster than gradient descent.

Disadvantages:

- More complex

○ 简单举例

Example: $\min_{\theta} J(\theta)$

$$\rightarrow \theta = \begin{bmatrix} \theta_1 \\ \theta_2 \end{bmatrix} \quad \theta_1 = 5, \theta_2 = 5.$$

$$\rightarrow J(\theta) = (\theta_1 - 5)^2 + (\theta_2 - 5)^2$$

$$\rightarrow \frac{\partial}{\partial \theta_1} J(\theta) = 2(\theta_1 - 5)$$

$$\rightarrow \frac{\partial}{\partial \theta_2} J(\theta) = 2(\theta_2 - 5)$$

```
function [jVal, gradient] = costFunction(theta)
    jVal = (theta(1)-5)^2 + ... (theta(2)-5)^2;
    gradient = zeros(2,1);
    gradient(1) = 2*(theta(1)-5);
    gradient(2) = 2*(theta(2)-5);

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

○ 复杂示例

Andrew Ng

$$\underline{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix} \quad \begin{array}{l} \text{theta}(1) \\ \text{theta}(2) \\ \vdots \\ \text{theta}(n+1) \end{array}$$

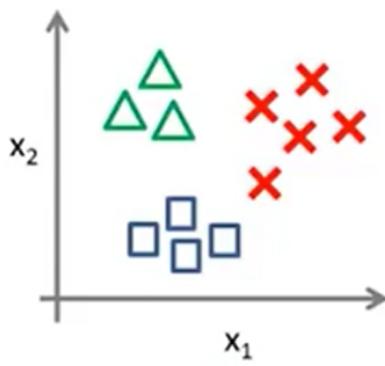
```

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  $\frac{\partial}{\partial \theta_n} J(\theta)$  ];

```

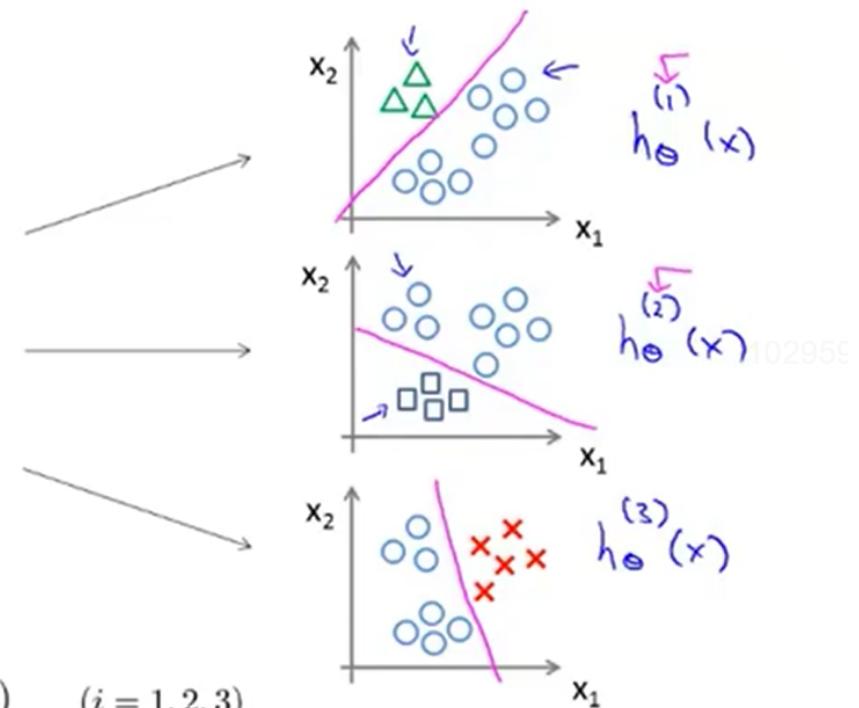
7. 多元分类

One-vs-all (one-vs-rest):



- Class 1: \triangle \leftarrow
- Class 2: \square \leftarrow
- Class 3: \times \leftarrow

$$h_{\theta}^{(i)}(x) = P(y = i|x; \theta) \quad (i = 1, 2, 3)$$



Andrew Ng

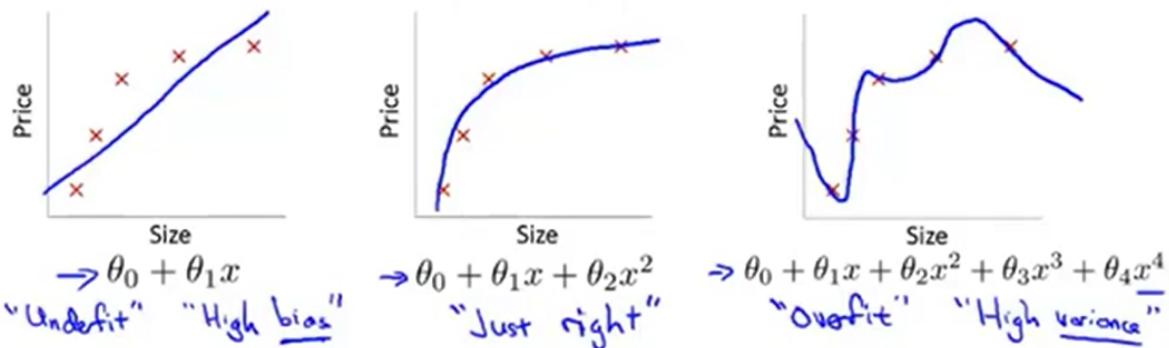
8. 正则化

1. 过渡拟合的坏处

- 使曲线过于复杂，失去预测作用

- 线性回归过度拟合

Example: Linear regression (housing prices)

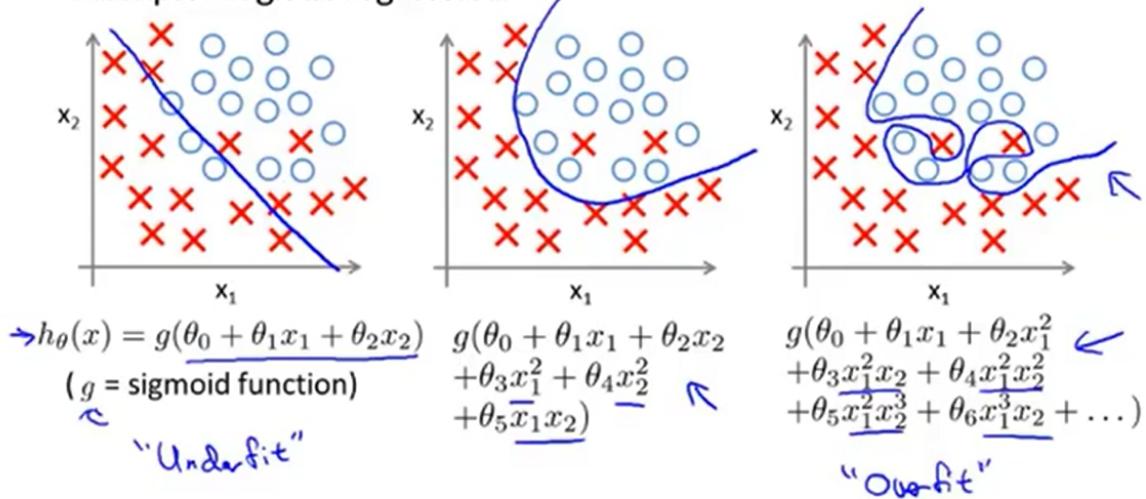


Overfitting: If we have too many features, the learned hypothesis may fit the training set very well ($J(\theta) = \frac{1}{2m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2 \approx 0$), but fail to generalize to new examples (predict prices on new examples).

Andrew Ng

- logistic回归过度拟合

Example: Logistic regression



Andrew Ng

- 如何避免过度拟合

Addressing overfitting:

Options:

1. Reduce number of features.
 - Manually select which features to keep.
 - Model selection algorithm (later in course).
2. Regularization.
 - Keep all the features, but reduce magnitude/values of parameters θ_j .
 - Works well when we have a lot of features, each of which contributes a bit to predicting y .

10295

Andrew Ng

2. 代价函数

- 正则化的目的
 - 减小 θ 预测值
 - 减少过度拟合

Regularization.

Small values for parameters $\theta_0, \theta_1, \dots, \theta_n$

- “Simpler” hypothesis
- Less prone to overfitting

$$\theta_2, \theta_4 \approx 0$$

Housing:

- Features: x_1, x_2, \dots, x_{100}
- Parameters: $\theta_0, \theta_1, \theta_2, \dots, \theta_{100}$

1029599063

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

~~$\theta_1, \theta_2, \theta_3, \dots, \theta_{100}$~~

Andrew Ng

- 加入后面一项
 - 增加代价函数，代价函数参数 λ 越小越好

In regularized linear regression, we choose θ to minimize

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

3. 线性回归的正则化

- 对于梯度下降算法的正则化

- 综上，在 θ 的右边加入了正则化一项
- 理解：
 - 对于 θ_0 不变
 - 对于 θ_j 而言，正则化项的数学作用是降低了 θ_j 的值，
 - θ_j 值减小的目的，就是降低x高阶项，（即使x项很多，通过让他们乘以较小的 θ ）就不会过渡拟合啦（放置拟合成过度S形状）

Gradient descent

$$\begin{aligned} & \text{Repeat } \{ \quad \theta_0, \theta_1, \dots, \theta_n \\ & \quad \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)} \quad \frac{\partial}{\partial \theta_0} J(\theta) \\ & \quad \theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right] \quad (j = 1, 2, 3, \dots, n) \\ & \quad \theta_j := \theta_j \left(1 - \alpha \frac{\lambda}{m}\right) - \alpha \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad 1029599063 \\ & \quad \left| 1 - \alpha \frac{\lambda}{m} \right| < 0.99 \quad \theta_j \times 0.99 \end{aligned}$$

Andrew Ng

- 举例说明

- 可见，正则化之后，对于 $\min(J(\theta))$, θ 需要满足的条件，有如下的变化。
- 这里也提示了， θ 的表达式就是对 J 求导=0化简得到的

Normal equation

$$X = \begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix} \leftarrow \text{m} \times (n+1)$$

$$y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m$$

$$\rightarrow \min_{\theta} J(\theta)$$

$$\rightarrow \theta = \left(X^T X + \lambda \underbrace{\begin{bmatrix} 0 & 1 & 0 & \dots & 0 \\ 1 & 0 & 1 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}}_{(n+1) \times (n+1)} \right)^{-1} X^T y$$

例 e.g. n=2 $\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

- 正则化对可逆性的影响，可以保证θ表达式中的逆矩阵一定是非奇异矩阵（可以求逆）

Non-invertibility (optional/advanced).

10295990

Suppose $m \leq n$, \leftarrow
 (#examples) (#features)

$$\theta = \frac{(X^T X)^{-1} X^T y}{\text{non-invertible / singular}} \quad \text{pinv} \quad \frac{\text{inv}}{\kappa}$$

If $\lambda > 0$,

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & 1 & & & \\ 1 & 0 & 1 & & \\ & 1 & 0 & \ddots & \\ & & \ddots & \ddots & 1 \end{bmatrix} \right)^{-1} X^T y$$

invertible.

4. Logistic 回归的正则化

- 总结：
 - 正则化的目的是降低过度拟合，
 - 两种寻找Min(J)的方法
 - 梯度下降
 - 高级优化（求导=0）
- Logistic和线性回归是相对的，只是 $h(\theta)$ 的取值方法有区别，以下面的Logistic回归梯度下降法为

例。

Gradient descent

Repeat {

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

$$\rightarrow \theta_j := \theta_j - \alpha \left[\frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} + \frac{\lambda}{m} \theta_j \right] \leftarrow$$

$\underbrace{\quad}_{(j = 1, 2, 3, \dots, n)}$
 $\underbrace{\quad}_{\theta_1, \dots, \theta_n}$

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

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

Andrew Ng

- 高级优化法的Logistic回归，也需要加一项



网易云课堂

Advanced optimization

$J(\theta)$ minimize (2 cost function)

θ_0 , $\theta_1, \dots, \theta_n$

```
function [jVal, gradient] = costFunction(theta)
    jVal = [code to compute  $J(\theta)$ ];
     $\Rightarrow J(\theta) = \left[ -\frac{1}{m} \sum_{i=1}^m y^{(i)} \log(h_\theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{j=1}^n \theta_j^2 \right]$ 
    gradient(1) = [code to compute  $\frac{\partial}{\partial \theta_0} J(\theta)$ ];
     $\frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_0^{(i)}$ 
    gradient(2) = [code to compute  $\frac{\partial}{\partial \theta_1} J(\theta)$ ];
     $\left( \frac{1}{m} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)}) x_1^{(i)} + \frac{\lambda}{m} \theta_1 \right)$ 
    gradient(3) = [code to compute  $\frac{\partial}{\partial \theta_2} J(\theta)$ ];
     $\vdots$ 
    gradient(n+1) = [code to compute  $\frac{\partial}{\partial \theta_n} J(\theta)$ ];
```

Andrew Ng

正则化逻辑回归的解 logistic regression with regularization.



```
1 import numpy as np
2 def costReg(theta, X, y, learningRate):
```

```

theta = np.matrix(theta)
3
X = np.matrix(X)
4
y = np.matrix(y)
5
first = np.multiply(-y, np.log(sigmoid(X*theta.T)))
6
second = np.multiply((1 - y), np.log(1 - sigmoid(X*theta.T)))
7
reg = (learningRate / (2 * len(X)) * np.sum(np.power(theta[:,1:theta.shape[1]], 2)))
8
return np.sum(first - second) / (len(X)) + reg
9

```

9. 神经网络学习

1. 非线性假设

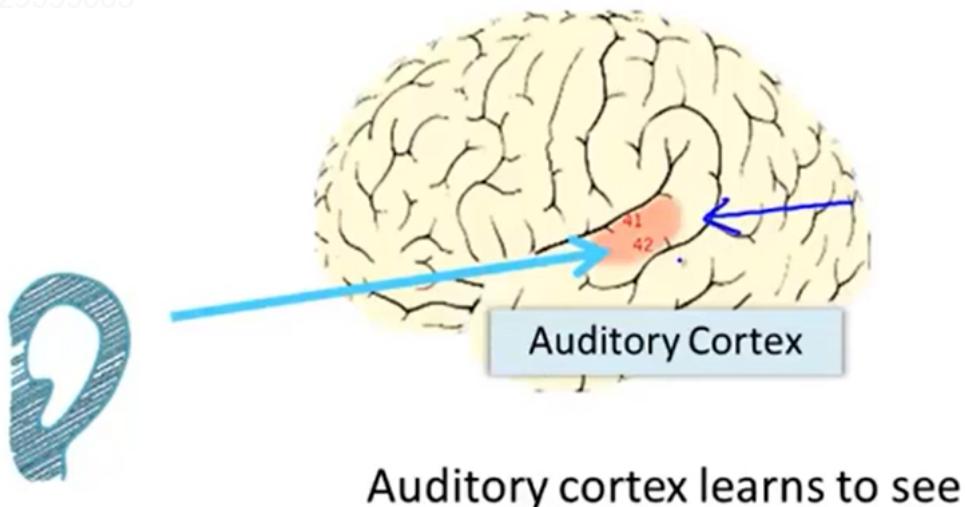
- 如果对所有的feature都进行线性回归的话，这样 x , $x \cdot x$ ……，等肯定会过度拟合

2. 神经元与大脑

- 神经网络算法的起源是上世纪八九十年代
- 人们想要模拟大脑思考的方式来解决实际问题
- 关于学习算法，像人类一样识别问题，解决问题

The “one learning algorithm” hypothesis

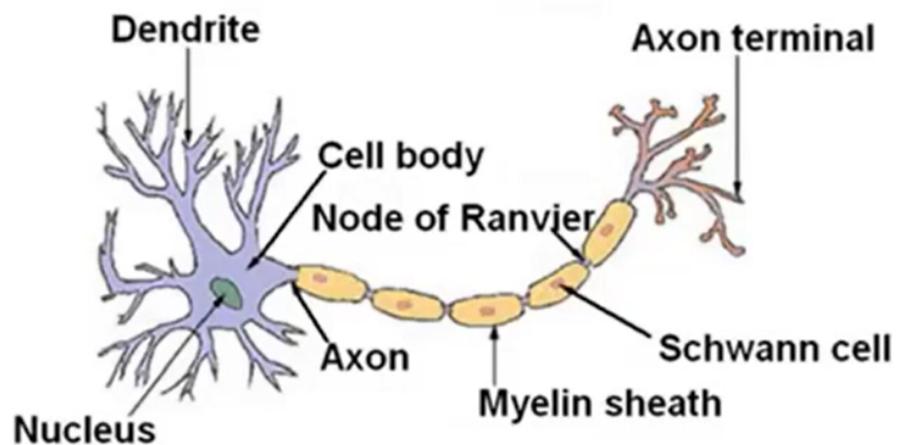
1029599063



3. 模型展示I

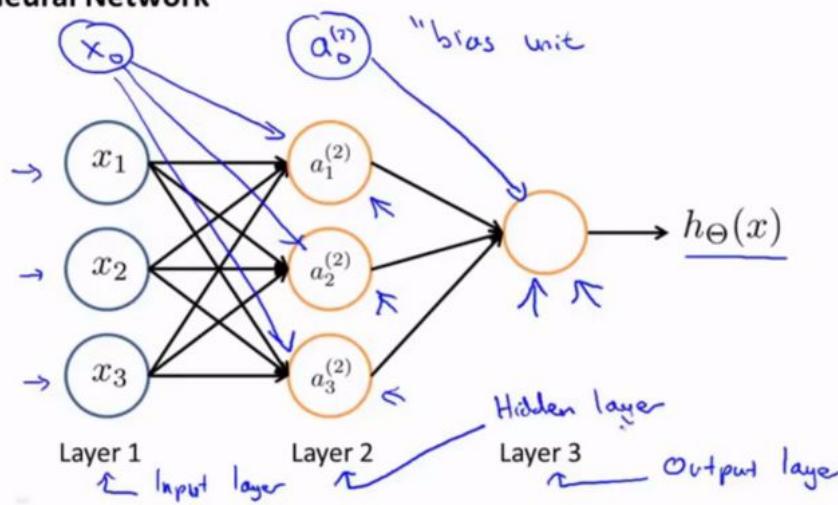
- 大脑中的神经网络

Neuron in the brain



网易云课堂

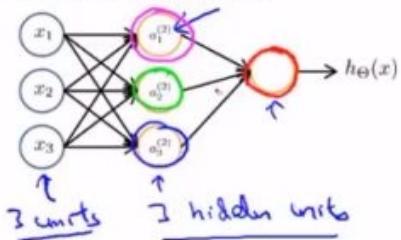
Neural Network



Andrew Ng

隐藏层不是特别好的术语
The term hidden layer isn't a great terminology,

Neural Network



$\rightarrow a_i^{(j)}$ = "activation" of unit i in layer j

$\rightarrow \Theta^{(j)}$ = matrix of weights controlling function mapping from layer j to layer $j + 1$

$$\Theta^{(1)} \in \mathbb{R}^{3 \times 4}$$

$$\rightarrow a_1^{(2)} = g(\Theta_{10}^{(1)} x_0 + \Theta_{11}^{(1)} x_1 + \Theta_{12}^{(1)} x_2 + \Theta_{13}^{(1)} x_3)$$

$$\rightarrow a_2^{(2)} = g(\Theta_{20}^{(1)} x_0 + \Theta_{21}^{(1)} x_1 + \Theta_{22}^{(1)} x_2 + \Theta_{23}^{(1)} x_3)$$

$$\rightarrow a_3^{(2)} = g(\Theta_{30}^{(1)} x_0 + \Theta_{31}^{(1)} x_1 + \Theta_{32}^{(1)} x_2 + \Theta_{33}^{(1)} x_3)$$

$$h_\Theta(x) = a_1^{(3)} = g(\Theta_{10}^{(2)} a_0^{(2)} + \Theta_{11}^{(2)} a_1^{(2)} + \Theta_{12}^{(2)} a_2^{(2)} + \Theta_{13}^{(2)} a_3^{(2)})$$

\Rightarrow If network has s_j units in layer j , s_{j+1} units in layer $j + 1$, then $\Theta^{(j)}$ will be of dimension $s_{j+1} \times (s_j + 1)$.

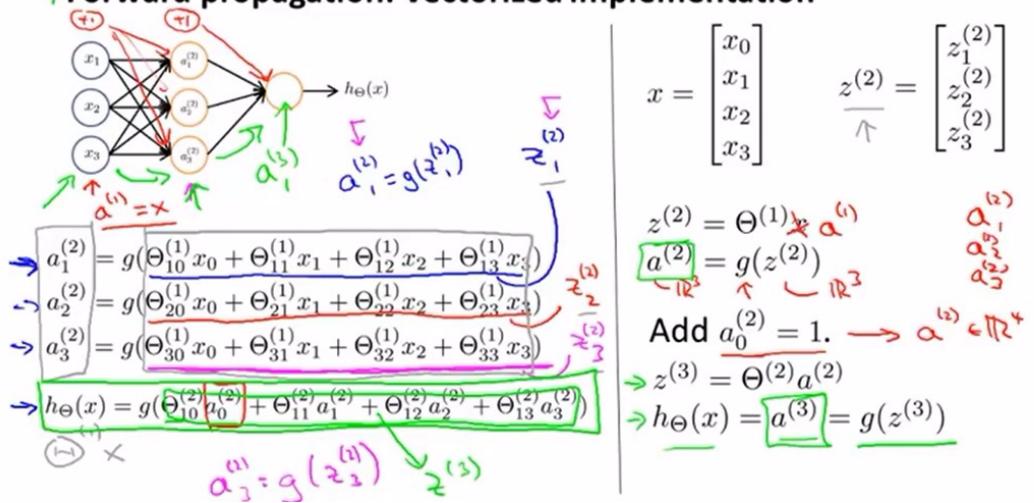
Andrew Ng

4. 模型展示II

- 前向传播：神经网络的向量化

- Θ_{10} 中，表示 x_0 到 a_1 的权重， Θ_{32} 表示 x_2 到 a_3 的权重， Θ_{21} 表示 x_1 到 a_2 的权重(第一个数字为本层第几行，第二个数字指前一层第几行)
- $\Theta^{(1)}$ 中，括号中的 1 表示它是放在第一列 input 值前的系数
- $g()$ 代指 σ 函数 (Logistic 回归)
- 右边： $z(2)$ 表示的是系数矩阵， 3×1 ，说明有三列输入 (x) 数据，括号中 $z(2)$ 表示其指向的是第二列 $a(2)$ ，同理 $z(3)$ 就指向第三列 (h)。
- z 中每列 z_1 还包括 4 个 Θ 系数， 4×1

Forward propagation: Vectorized implementation



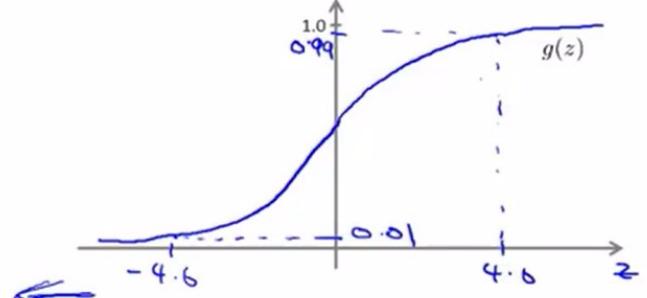
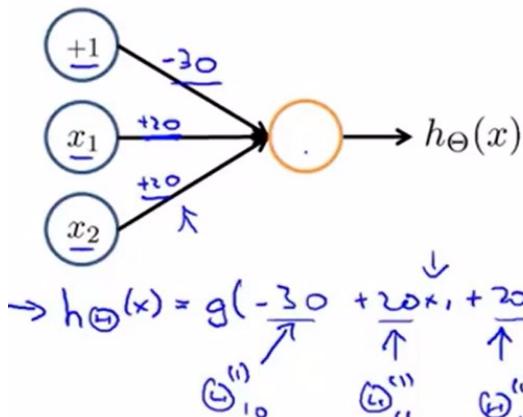
5. 例子与直觉理解I

- AND

Simple example: AND

$$\rightarrow x_1, x_2 \in \{0, 1\}$$

$$\rightarrow y = x_1 \text{ AND } x_2$$



x_1	x_2	$h_\Theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

$h_\Theta(x) \approx x_1 \text{ AND } x_2$

Andrew Ng

当且仅当 x_1 和 x_2 都等于 1 时输出为 1

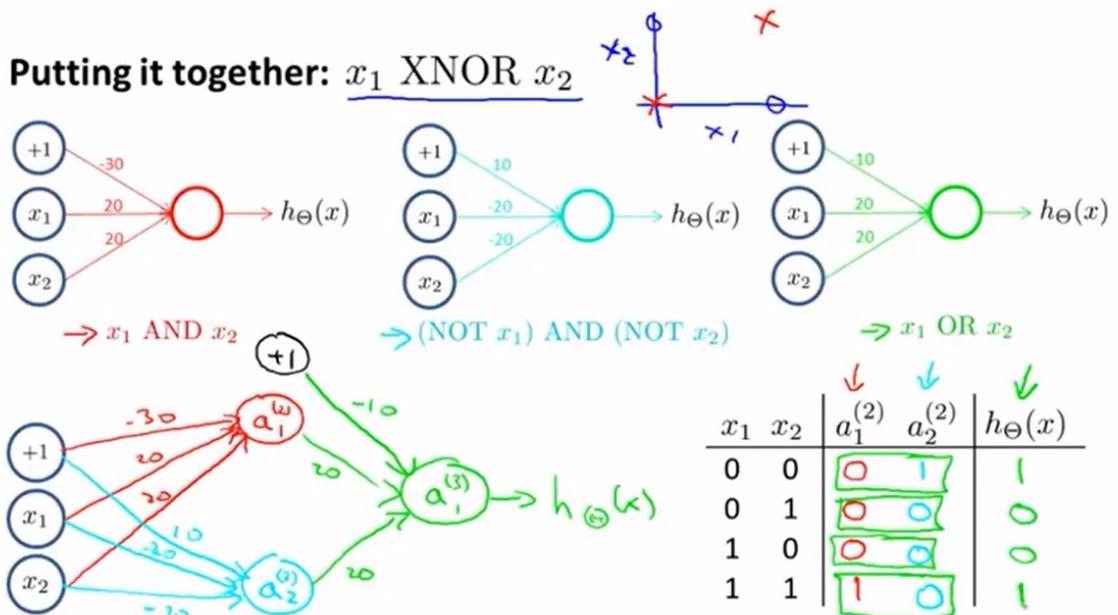
- 其中 $-30, 20, 10$ 为权重 Θ
- $g()$ 为 logistic 回归函数
- 可以看出，只有在 x_1, x_2 都等于 1 的时候结果为 1

- OR

- x_1, x_2 有一个为 1，结果为 1

6. 例子与直觉理解II

- 神经网络方法，将三种逻辑门合并



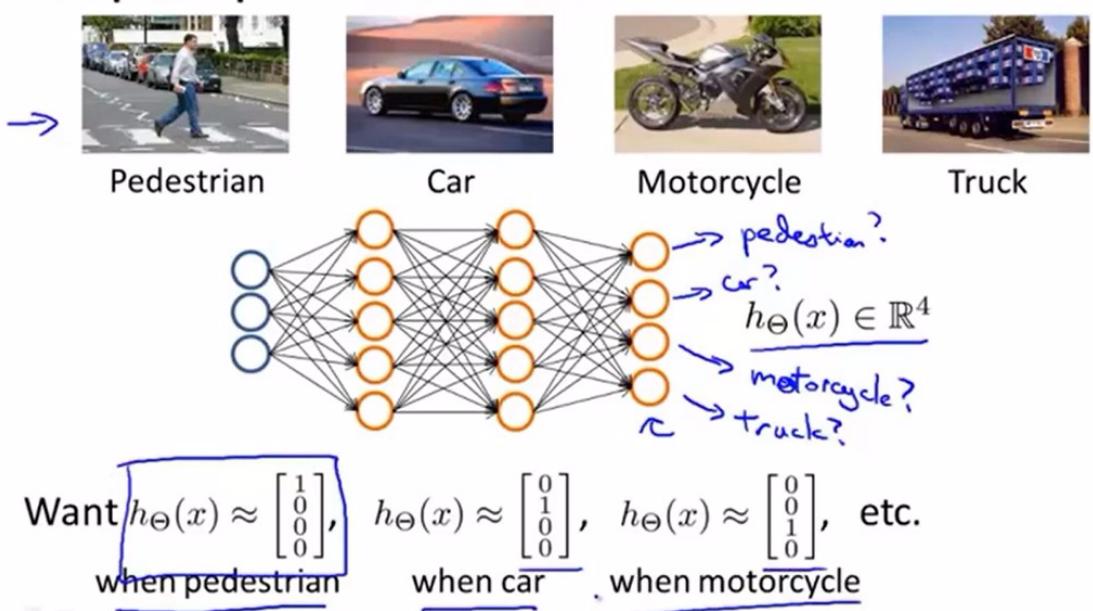
Andrew Ng

- 神经网络计算，不同层的logistic计算，并将计算结果传输到下一隐藏层，并最终得出输出结果

7. 多元分类（举例）

- 多类输出
 - 识别图片为行人，汽车，摩托，还是客车

Multiple output units: One-vs-all.



10. 神经网络参数的反向传播算法

1. 代价函数

Cost function

Logistic regression:

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

Neural network:

$$\rightarrow h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$\rightarrow J(\Theta) = -\frac{1}{m} \left[\sum_{i=1}^m \sum_{k=1}^K y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) \log(1 - (h_\Theta(x^{(i)}))_k) \right]$$

$$\boxed{\frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2}$$

Andrew Ng

2. 反向传播算法

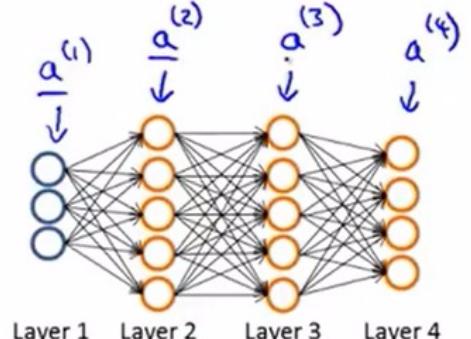
- 梯度计算

Gradient computation

Given one training example (x, y) :

Forward propagation:

$$\begin{aligned} a^{(1)} &= x \\ \rightarrow z^{(2)} &= \Theta^{(1)} a^{(1)} \\ \rightarrow a^{(2)} &= g(z^{(2)}) \quad (\text{add } a_0^{(2)}) \\ \rightarrow z^{(3)} &= \Theta^{(2)} a^{(2)} \\ \rightarrow a^{(3)} &= g(z^{(3)}) \quad (\text{add } a_0^{(3)}) \\ \rightarrow z^{(4)} &= \Theta^{(3)} a^{(3)} \\ \rightarrow a^{(4)} &= h_\Theta(x) = g(z^{(4)}) \end{aligned}$$



- 反向传播算法

- for i to m
- $a(1) = x(i)$ 即输入

Backpropagation algorithm

→ Training set $\{(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})\}$

Set $\Delta_{ij}^{(l)} = 0$ (for all l, i, j). (use to compute $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta)$)

For $i = 1$ to $m \leftarrow (\underline{x}^{(i)}, \underline{y}^{(i)})$.

- Set $a^{(1)} = \underline{x}^{(i)}$
- Perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \dots, L$
- Using $y^{(i)}$, compute $\delta^{(L)} = \underline{a}^{(L)} - \underline{y}^{(i)}$
- Compute $\delta^{(L-1)}, \delta^{(L-2)}, \dots, \delta^{(2)}$
- $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$
- $\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + \delta^{(l+1)} (a^{(l)})^T$
- $D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}$ if $j \neq 0$
- $D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$ if $j = 0$

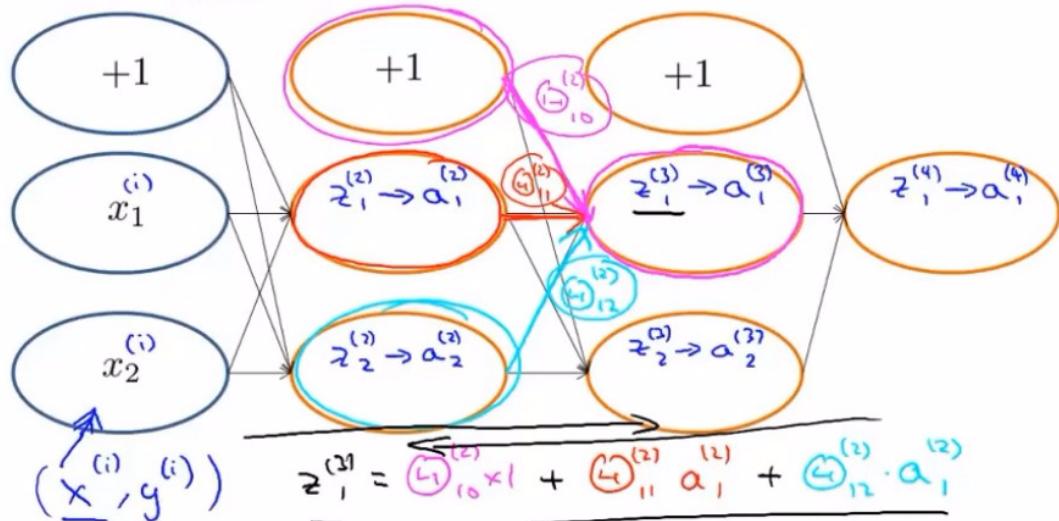
$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)}$

3. 反向传播算法实例（如何理解反向传播算法）

- 回顾，正向传播算法

- 首先输入层 x (或 $a^{(1)}$)，隐藏层 $a^{(2)}$ ，隐藏层 $a^{(3)}$ ，输出层 $a^{(4)}$ 。共计4层
- 正向算法即从左向右依次传递
- 第一层： $a^{(1)} = x$,
- 第二层： $z^{(2)} = \Theta^{(1)} a^{(1)}, a^{(2)} = g(z^{(2)})$
- 第三层： $z^{(3)} = \Theta^{(2)} a^{(2)}, a^{(3)} = g(z^{(3)})$

Forward Propagation



Andrew Ng

- 反向传播法

- 什么是反相传播？

- 代价函数的表示方法（对数形式，logistic回归）为 $x(i), y(i)$ 的函数。
- （忽略正则项）为了简单理解此处的代价函数，可以将求和号中的 $\text{cost}(i)$ 等价看成是

$$\text{cost}(i) \approx (h_{\Theta}(x^{(i)}) - y^{(i)})^2$$
 形式。表示预测值 h 与观测值 y 的误差大小。

What is backpropagation doing?

$$J(\Theta) = -\frac{1}{m} \left[\sum_{i=1}^m y^{(i)} \log(h_{\Theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\Theta}(x^{(i)})) \right] + \frac{\lambda}{2m} \sum_{l=1}^{L-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (\Theta_{ji}^{(l)})^2$$

$(x^{(i)}, y^{(i)})$

Focusing on a single example $x^{(i)}, y^{(i)}$, the case of 1 output unit, and ignoring regularization ($\lambda = 0$),

$$\text{cost}(i) = y^{(i)} \log h_{\Theta}(x^{(i)}) + (1 - y^{(i)}) \log h_{\Theta}(x^{(i)})$$

(Think of $\text{cost}(i) \approx (h_{\Theta}(x^{(i)}) - y^{(i)})^2$)

i.e. how well is the network doing on example i?

Andrew Ng

- 反向传播法是如何工作的
 - +1相为偏置单元，通常为1常数
 - $\delta_j^{(l)}$, 中j表示第几行, (l)表示第l层。
 - 从微积分的角度理解： $\delta_j^{(l)}$ 就是等于第cost函数对l层第j行z的偏导。
 - 如何求得 δ_j 呢： 和正向传播类似的！

- $\delta_1^{(4)}$ 是已知的= $y^{(4)} - a_1^{(4)}$

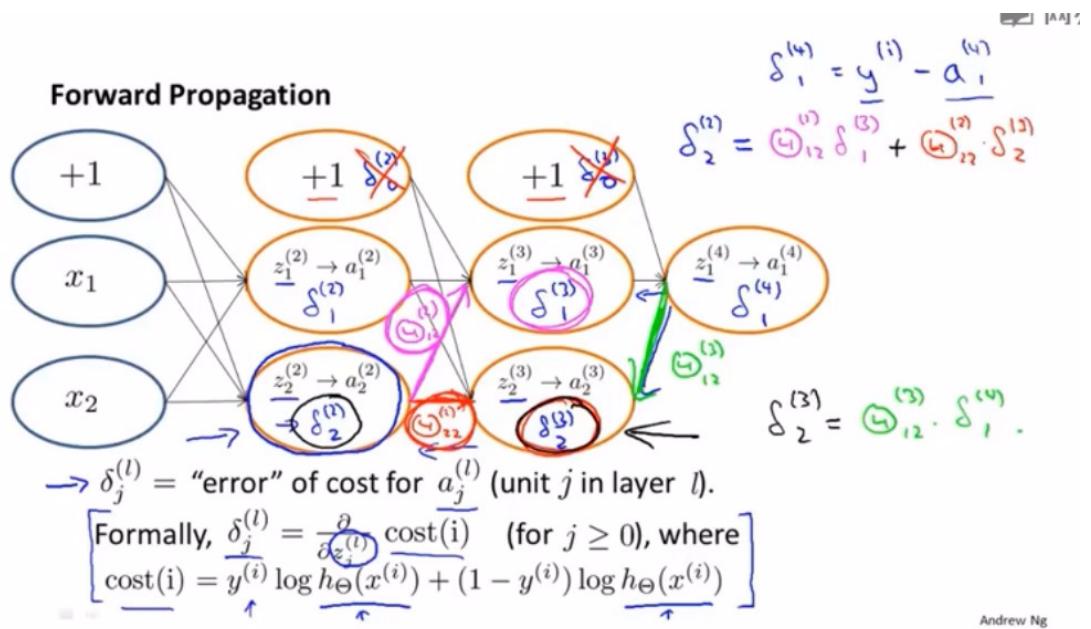
■ 那么 $\delta_2^{(3)}$ 就等于 =

$$\delta_2^{(3)} = \odot_{12}^{(3)} \cdot \delta_1^{(4)}$$

■ $\delta_2^{(2)}$ 就等于 =

$$\delta_2^{(2)} = \odot_{12}^{(2)} \delta_1^{(3)} + \odot_{22}^{(2)} \delta_2^{(3)}$$

- 无非就是反向乘以权重



Andrew Ng

- 使用注意：展开参数，在神经网络中，gradient, initialTheta, D, Θ都是矩阵形式

- Example 中介绍如何将这些矩阵“解压Unroll”成向量

Advanced optimization

```

function [jVal, gradient] = costFunction(theta)
    ...
optTheta = fminunc(@costFunction, initialTheta, options)

```

Neural Network (L=4):

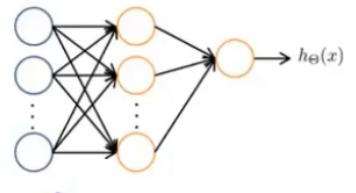
- $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ - matrices (Theta1, Theta2, Theta3)
- $D^{(1)}, D^{(2)}, D^{(3)}$ - matrices (D1, D2, D3)
- “Unroll” into vectors

- 举例：s1表示第一层有十个单元，s2表示第二层有十个单元(Unit)
- 解释：为什么 $\Theta^{(1)}$ 是 10×11 维矩阵：因为第一个数字10表示第二层有10个单元（层），11表示第一层有11个单元（包括偏置单元1）。D同理
- 解释：可以将Theta1(:,), Theta2(:,)展开成一个大向量thetaVec(1×231)，同理如果想将thetaVec

拆开成矩阵，就要用最下面的方法1:110,111:220,的拆分方法。

Example

$s_1 = 10, s_2 = 10, s_3 = 1$
 $\Theta^{(1)} \in \mathbb{R}^{10 \times 11}, \Theta^{(2)} \in \mathbb{R}^{10 \times 11}, \Theta^{(3)} \in \mathbb{R}^{1 \times 11}$
 $D^{(1)} \in \mathbb{R}^{10 \times 11}, D^{(2)} \in \mathbb{R}^{10 \times 11}, D^{(3)} \in \mathbb{R}^{1 \times 11}$
 $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$
 $\rightarrow \text{thetaVec} = [\Theta^{(1)}(:); \Theta^{(2)}(:); \Theta^{(3)}(:)];$
 $\rightarrow \text{DVec} = [D^{(1)}(:); D^{(2)}(:); D^{(3)}(:)];$
 $\Theta^{(1)} = \text{reshape}(\text{thetaVec}(1:110), 10, 11);$
 $\Theta^{(2)} = \text{reshape}(\text{thetaVec}(111:220), 10, 11);$
 $\Theta^{(3)} = \text{reshape}(\text{thetaVec}(221:231), 1, 11);$



- 将上述拆分组合方法应用于学习算法：
- 初始参数 $\Theta^{(1)}, \dots, \Theta^{(3)}$
- 将上述初始参数解压成超级向量(初始向量 initialTheta)
- 传入fminunc优化算法，可以返回costFunction值
- 梯度算法可以得到 $\Theta^{(1)}, \dots, \Theta^{(3)}$ 的优化值，再次形成一个超级向量(thetaVec)
- 从thetaVec中就可以得到 $\Theta^{(1)}$ 的值
- 用正/反向传播算法计算D，解压D得到gradientVec，又可以循环得到优化的 $\Theta^{(1)}, \dots, \Theta^{(3)}$
- 为什么要这个矩阵向量来回转化？因为有些算法需要长向量。

Learning Algorithm

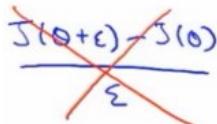
\rightarrow Have initial parameters $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$.
 \rightarrow Unroll to get initialTheta to pass to
 \rightarrow fminunc(@costFunction, initialTheta, options)

function [jval, gradientVec] = costFunction(thetaVec)
 \rightarrow From thetaVec, get $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ reshape
 \rightarrow Use forward prop/back prop to compute $D^{(1)}, D^{(2)}, D^{(3)}$ and $J(\Theta)$
Unroll $D^{(1)}, D^{(2)}, D^{(3)}$ to get gradientVec.

某些时候，反向梯度算法可能会在有bug的存在下良好运行，虽然经过优化Costfunction不断减小，但是bug的影响却会越来越大。所以此时需要运算梯度检测消除bug。

梯度检测可以100%使正向，反向传播算法正确运行！

a. 下图为假设J的曲线，对一段进行求斜率



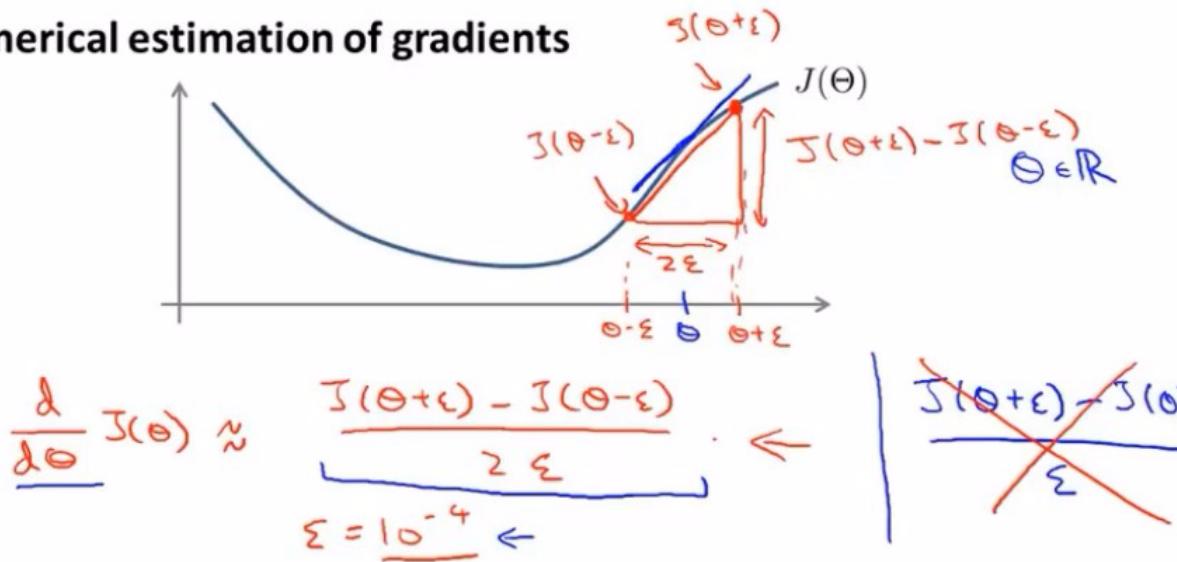
单侧差分：

$$\frac{\partial}{\partial \theta} J(\theta) \approx \frac{J(\theta + \epsilon) - J(\theta - \epsilon)}{2\epsilon}$$

(代码见下图底)

双层差分要更加准确的表示 θ 点的斜率。

Numerical estimation of gradients



Implement: gradApprox = $(J(\text{theta} + \text{EPSILON}) - J(\text{theta} - \text{EPSILON})) / (2 * \text{EPSILON})$

Andrew Ng

b. 利用上述的双侧差分方法，对下列进行求偏导数。

Parameter vector θ

$$\rightarrow \theta \in \mathbb{R}^n \quad (\text{E.g. } \theta \text{ is "unrolled" version of } \underline{\Theta^{(1)}}, \underline{\Theta^{(2)}}, \underline{\Theta^{(3)}})$$

$$\rightarrow \theta = [\theta_1, \theta_2, \theta_3, \dots, \theta_n]$$

$$\rightarrow \frac{\partial}{\partial \theta_1} J(\theta) \approx \frac{J(\theta_1 + \epsilon, \theta_2, \theta_3, \dots, \theta_n) - J(\theta_1 - \epsilon, \theta_2, \theta_3, \dots, \theta_n)}{2\epsilon}$$

$$\rightarrow \frac{\partial}{\partial \theta_2} J(\theta) \approx \frac{J(\theta_1, \theta_2 + \epsilon, \theta_3, \dots, \theta_n) - J(\theta_1, \theta_2 - \epsilon, \theta_3, \dots, \theta_n)}{2\epsilon}$$

$$\vdots$$

$$\rightarrow \frac{\partial}{\partial \theta_n} J(\theta) \approx \frac{J(\theta_1, \theta_2, \theta_3, \dots, \theta_n + \epsilon) - J(\theta_1, \theta_2, \theta_3, \dots, \theta_n - \epsilon)}{2\epsilon}$$

Andrew Ng

c. 将上述偏导数，写成Octave代码，

thetaPlus, thetaMinus分别为+epsilon, -epsilon。

最后计算出gradApprox和反向传播算法中的DVec是否接近！如果等于或十分接近就说明正向，反向传播算法是正确的！

```

for i = 1:n,
    thetaPlus = theta;
    thetaPlus(i) = thetaPlus(i) + EPSILON;
    thetaMinus = theta;
    thetaMinus(i) = thetaMinus(i) - EPSILON;
    gradApprox(i) = (J(thetaPlus) - J(thetaMinus)) / (2*EPSILON);
end;

```

■ Andrew Ng

d. 最后注意

1. 运用反向传播算法计算出Dvec (长向量形式)
2. 运用梯度检验计算gradApprox的值
3. 确保Dvec and gradApprox的值非常相近
4. 确认之后，在继续运行反向传播算法的时候，请关掉梯度检验，因为梯度检验是一种非常耗时复

杂的计算方法。而反向传播算法是高性能的计算方法。

重要: 在确认梯度检验正确之后，确保关掉梯度检验。否则程序运行会非常缓慢。

Implementation Note:

- - Implement backprop to compute DVec (unrolled $D^{(1)}, D^{(2)}, D^{(3)}$).
- - Implement numerical gradient check to compute gradApprox.
- - Make sure they give similar values.
- Turn off gradient checking. Using backprop code for learning.

$\downarrow \quad \downarrow \quad \downarrow$
 $DVec$
 $\delta^{(1)}, \delta^{(2)}, \delta^{(3)}$

Important:

- Be sure to disable your gradient checking code before training your classifier. If you run numerical gradient computation on every iteration of gradient descent (or in the inner loop of `costFunction(...)`) your code will be very slow.

Andrew Ng

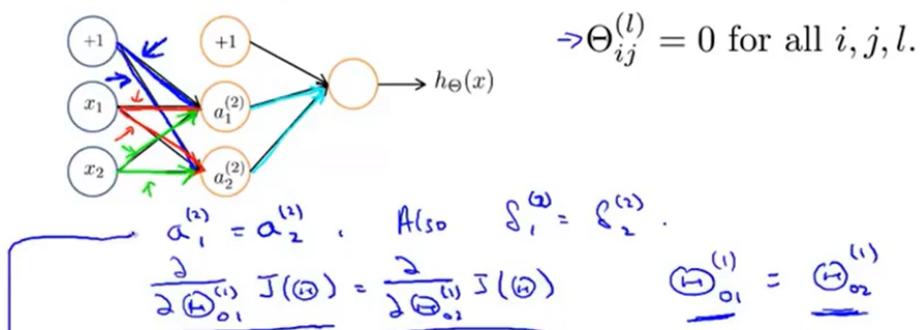
5. 随机初始化

◦ Θ 的初始值都为0的情况

- 这样的话，每一层的 a_1, a_2 都是相等的（因为 $\Theta=0$ ）
- 梯度下降后， Θ 同样一起更新，还是相等，红色的两条，绿色的两条 Θ 都保持相等

← → ← →

Zero initialization



After each update, parameters corresponding to inputs going into each of two hidden units are identical.

$$\underline{\underline{\Theta_{01}}} = \underline{\underline{\Theta_{02}}}$$

Andrew Ng

◦ 所以要对初始值随机化

- 控制 Θ 值在 ϵ , $-\epsilon$ 之间，方法如下所示

Random initialization: Symmetry breaking

→ Initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\epsilon, \epsilon]$
 (i.e. $-\epsilon \leq \Theta_{ij}^{(l)} \leq \epsilon$)

E.g.

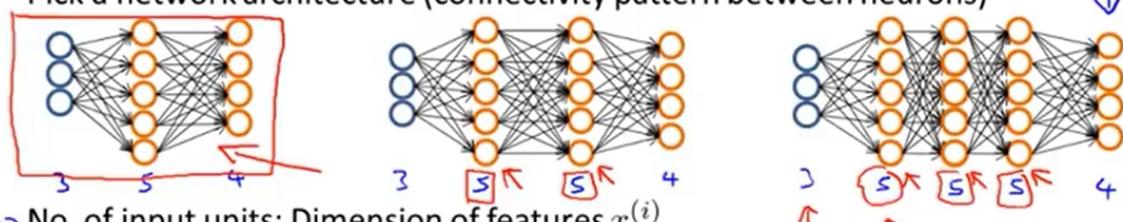
→ Theta1 = $\frac{\text{rand}(10, 11) * (2 * \text{INIT_EPSILON})}{\text{INIT_EPSILON}}$
 $[-\epsilon, \epsilon]$

Theta2 = $\frac{\text{rand}(1, 11) * (2 * \text{INIT_EPSILON})}{\text{INIT_EPSILON}}$

6. 组合到一起

Training a neural network

Pick a network architecture (connectivity pattern between neurons)



→ No. of input units: Dimension of features $x^{(i)}$

→ No. output units: Number of classes

Reasonable default: 1 hidden layer, or if >1 hidden layer, have same no. of hidden units in every layer (usually the more the better)

$$y \in \{1, 2, 3, \dots, 10\}$$

~~$y \neq 5$~~

$$y = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \text{ or } \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \leftarrow \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

Andrew Ng

-

- 训练神经网络

- 步骤一：随机化权重，尽量接近0，但不等于0

- 步骤二：采用正向传播法，求得 $h(x)$

- 步骤三：利用程序计算出 $J(\Theta)$

- 步骤四：采用反向传播法，对 J 求偏导

- 对于第一次求反向传播算法建议用for循环来完成

\rightarrow 4. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$
 \rightarrow for $i = 1:m$ { $(x^{(i)}, y^{(i)})$, $(x^{(i)}, y^{(i)})$, ..., $(x^{(i)}, y^{(i)})$ }
 → Perform forward propagation and backpropagation using example $(x^{(i)}, y^{(i)})$
 (Get activations $a^{(l)}$ and delta terms $\delta^{(l)}$ for $l = 2, \dots, L$).
 $\rightarrow \Delta^{(l)} := \Delta^{(l)} + \delta^{(l)}(a^{(l)})^T$
 ...
 Compute $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$.

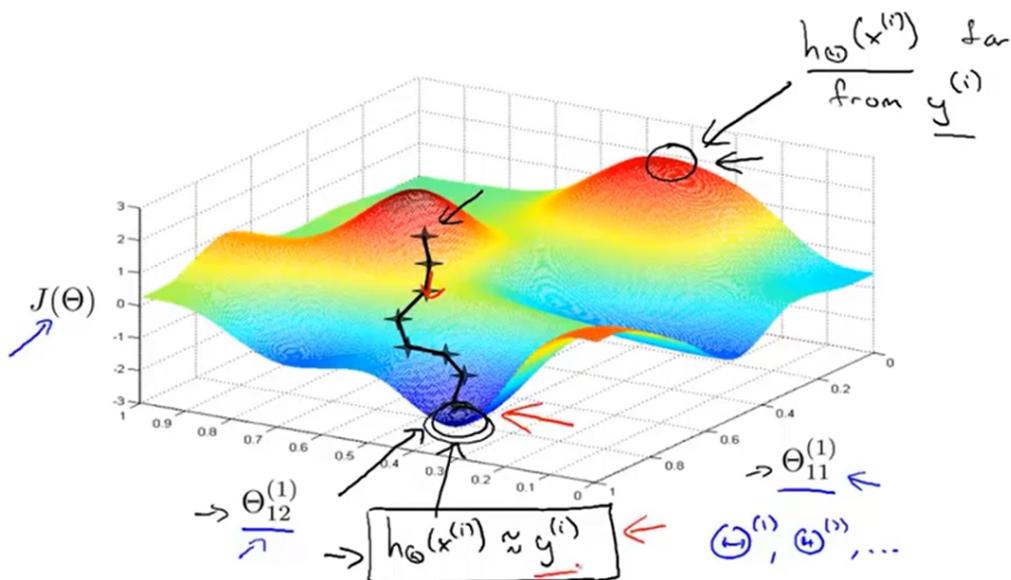
Andrew Ng

- 步骤五：利用梯度检查方法测试计算是否正确
- 利用梯度下降或者高级优化算法和反向传播算法得到最小的 J (cost函数)

Training a neural network

- \rightarrow 5. Use gradient checking to compare $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ computed using backpropagation vs. using numerical estimate of gradient of $J(\Theta)$.
 → Then disable gradient checking code.
 \rightarrow 6. Use gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ
- $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$ ←
 $J(\Theta)$ — non-convex.

- 举例说明：反向传播算法是最成功的深度学习算法，但是难于理解。



Andrew Ng

■