Machine Learning Notes

Cherrot Luo

2011

目录

1 Introduction **4**
1.1 Supervised Learning . . . 4
1.2 Unsupervised Learning(Clustering) . . . 4

2 Linear Regression With One Variable **4**
2.1 Model Representation . . . 4
2.1.1 Training Set . . . 4
2.2 Cost Function . . . 4
2.3 Cost function intuition I . . . 4
2.4 Cost function intuition II . . . 5
2.5 Gradient descent -- 求最小 J(θ_0, θ_1) 的算法 . . . 6
2.6 Gradient descent intuition . . . 7
2.7 Gradient descent for linear regression . . . 8

3 Linear Algebra Review **8**
3.1 Matrices & Vectors . . . 8
3.2 加法 . . . 8
3.3 矩阵 X 向量 . . . 8
3.4 矩阵 X 矩阵 . . . 8
3.5 矩阵乘法的性质 . . . 8
3.6 矩阵求逆 Inverse 和转置 Transpose . . . 8
3.6.1 求逆: . . . 8
3.6.2 转置 . . . 9

4 Linear Regression with Multiple Variables **9**
4.1 Multiple Features . . . 9
4.2 Gradient descent for multiple variables . . . 9
4.3 Gradient descent in practice I:Feature Scaling . . . 9
4.3.1 Mean(平均值)normalization . . . 10
4.4 Gradient descent in practice II: Learning rate(α) . . . 10
4.5 Features and polynomial regression . . . 10
4.6 Normal equation 正规方程 . . . 10
4.7 Normal equation and non-invertibility(不可逆性) . . . 11
4.7.1 $X^T X$ 不可逆的原因? . . . 12

5 Octave Tutorial **12**
5.1 Basic operations . . . 12
5.2 Moving data around . . . 13
5.3 Computing on Data . . . 14
5.4 Plotting data . . . 15
5.5 For, While, If statements, and Functions . . . 15
5.5.1 For . . . 15
5.5.2 While . . . 16
5.5.3 If-else . . . 16
5.5.4 Funtions . . . 16

5.6 Vectorization 向量化 16

6 Logistic Regression **17**
6.1 Classification 17
6.2 Hypothesis Representation 17
6.2.1 Interpretation(解释)of Hypothesis Output 17
6.3 Decision Boundary 17
6.4 Cost function 18
6.5 Simplified cost function and gradient descent 19
6.5.1 Gradient Descent 20
6.6 Advanced Optimization 20
6.7 Multi-class classification: One-vs-All 21
6.7.1 One-vs-all(one-vs-rest) 21

7 Regularization **23**
7.1 The problem of overfitting 23
7.1.1 Addressing overfitting 23
7.2 Cost function 24
7.2.1 Intuition 24
7.2.2 Regularization 24
7.3 Regularized linear regression 25
7.3.1 Gradient descent 25
7.3.2 Normal equation 26
7.4 Regularized logistic regression 26
7.4.1 Gradient Descent 26
7.4.2 Advanced optimization 26

8 Neural Networks: Representation **26**
8.1 Non-linear Hypotheses 26
8.2 Neurons(神经元)and the Brain 27
8.3 Model representation I 27
8.3.1 Nuron in the brain 27
8.3.2 Neuron model: losgistic unit 29
8.3.3 Neural Network 29
8.4 Model representation II 29
8.4.1 Forward propagation: Vectorized implementation 前向传播:矢量化实现 29
8.4.2 Neural Network learning its own features 29
8.5 Examples and intuitions I 30
8.5.1 Simple example: AND 30
8.6 Examples and intuitions II 30
8.7 Multi-class classification 31
8.7.1 Multiple output units: One-vs-all 31

9 Neural Networks: Learning(the parameters) **32**
9.1 Cost Function 32
9.1.1 Neural Network (Classification) 32
9.1.2 Cost function 33
9.2 Backpropagation Algorithm (to compute derivation of cost function) 34
9.2.1 Gradient Comutation 34
9.2.2 Backpropagation algorithm 36
9.3 Implementation note: Unrolling parameters 36
9.4 Gradient Checking 37
9.5 Random Initialization 38
9.5.1 Initial value of Θ 38
9.5.2 Random initialization: Symmetry(对称)breaking 39
9.6 Putting It Together 39
9.6.1 Training a neural network 39

10 Advice for applying machine learning **40**
10.1 Deciding what to try next . . . 40
10.2 Evaluating a hypothesis . . . 40
10.2.1 Training/testing procedure for linear regression . . . 40
10.2.2 Training/testing procedure for logistic regression(classification) . . . 41
10.3 Model selection and training/validation/test sets . . . 41
10.3.1 Evaluating your hypothesis . . . 42
10.3.2 Train/validation/test error . . . 42
10.4 Diagnosing bias vs. variance . . . 44
10.5 Regularization and bias/variance . . . 45
10.5.1 Choosing the regularization parameter λ . . . 46
10.5.2 Bias/variance as a function of the regularization parameter λ . . . 47
10.6 Learning curves . . . 48
10.6.1 High bias . . . 48
10.6.2 High variance . . . 49
10.7 Deciding what to try next (revisited) . . . 49
10.7.1 Neural networks and overfitting . . . 49

11 Machine Learning System Design **50**
11.1 Prioritizing what to work on: Spam classification example . . . 50
11.2 Error analysis . . . 52
11.2.1 Recommended approach . . . 52
11.2.2 Error Analysis . . . 52
11.2.3 The importance of numerical evaluation . . . 53
11.3 Error metrics for skewed classes . . . 54
11.3.1 Cancer classification example . . . 54
11.3.2 Pricision/Recall . . . 54
11.4 Trading off precision and recall . . . 55
11.4.1 F_1 Score (F score) . . . 56
11.5 Data for Machine Learning . . . 58

1 Introduction

1.1 Supervised Learning

e.g. predict the price, predict cancer or not, etc.
We give the algorithm a data set in which the right answer was given.

Regression program Predict continuous valued output.

Classification Discrete(离散的)valued output.

1.2 Unsupervised Learning(Clustering)

e.g. Google News, Social network clustrering, 鸡尾酒派对问题 etc.

singular value decomposition 奇异值分解。

2 Linear Regression With One Variable

2.1 Model Representation

卖房子问题

Supervised learning Given the ``right answer'' for each example in the data.

Regression Problem Predict real-valued output.(Classification Problem 是用来预测离散值的)

2.1.1 Training Set

m Number of training examples.

x's ``input'' variable / features.(often also called purchase)

y's ``output'' variable / ``target'' variable

$\left(x^{(i)}, y^{(i)}\right)$ 通常的 (x_i, y_i)。

单变量(Univariate)线性回归的直观认识可见图2.1。

2.2 Cost Function

一元 Hypothesis 的公式 $h_\theta(x) = \theta_0 + \theta_1 x$, 其实就是 y=a*x+b。$\Theta_{i's}$ 称为这个模型的参数 (Parameters). 问题要解决的就是如何选择这两个参数,使数据拟合的误差最小:

$$\underset{\theta_0\theta_1}{Minimize} \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 \tag{2.1}$$

定义 cost function 为:

$$J(\theta_0, \theta_1) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right)^2 \tag{2.2}$$

(方差的二分之一)

2.3 Cost function intuition I

如果 $\theta_0 = 0$, 那么 cost function 是二维坐标系的弓形(一元二次方程)。

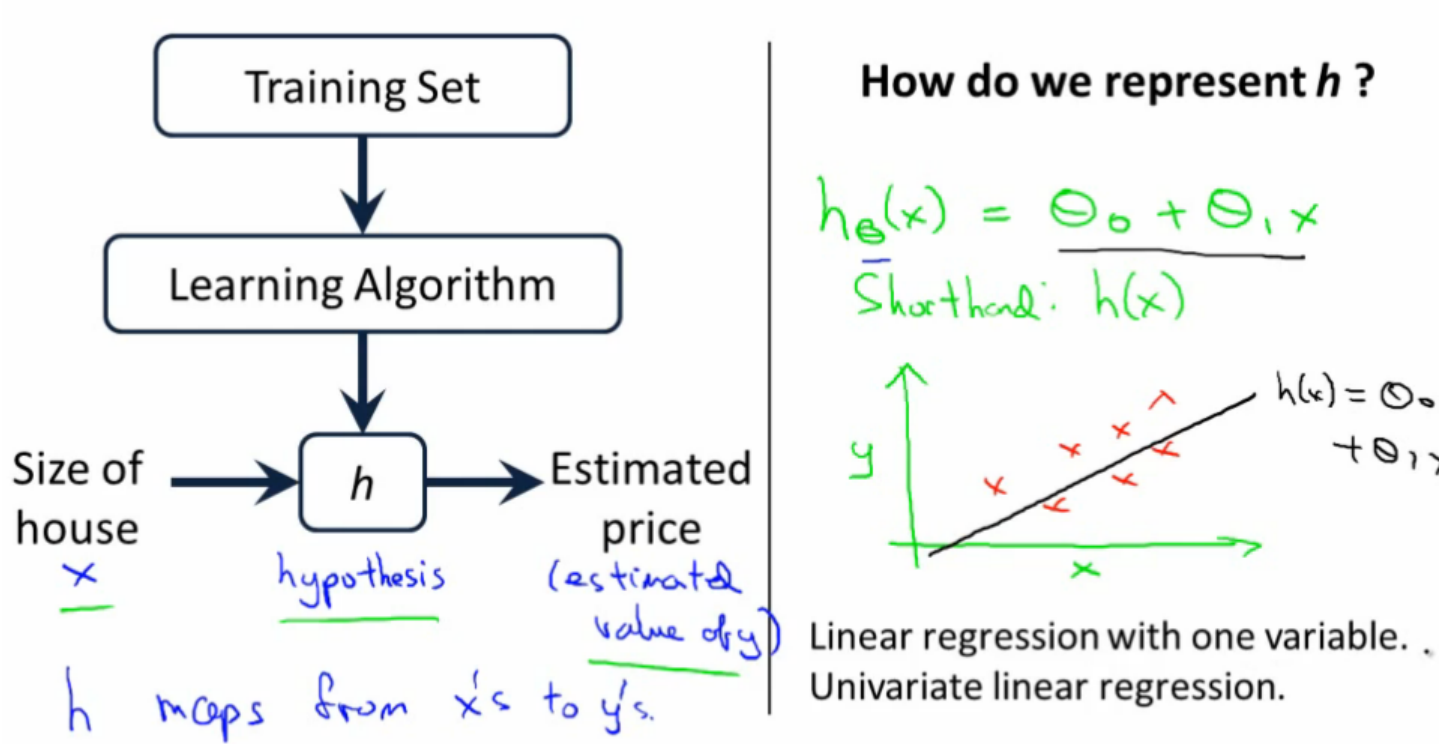


图 2.1: training set

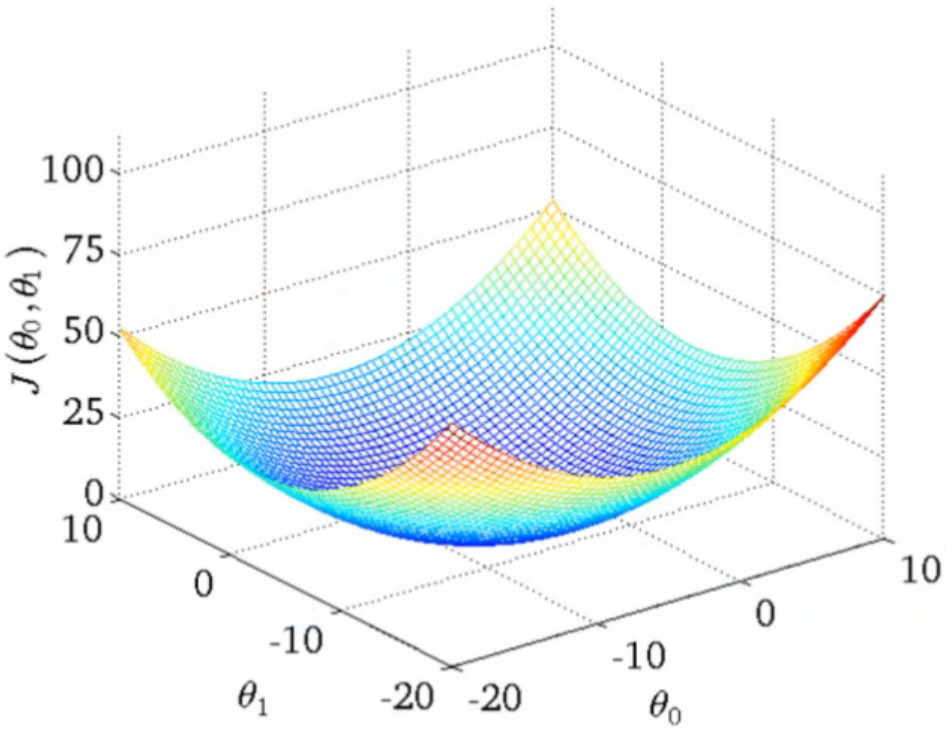


图 2.2: Cost Function 的直观表示

2.4 Cost function intuition II

cost function 如图2.2所示，自变量是 θ_0 和 θ_1(这种形状称为凸函数 convex funtion)：
contour plot(等值线图)可以更清楚直观的表示出 Cost Function 的收敛趋势，如图2.3所示。

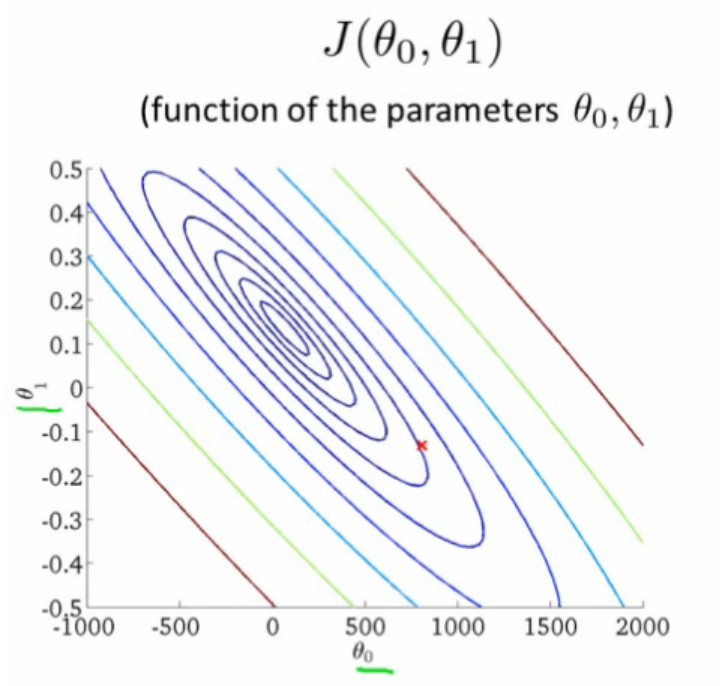


图 2.3: 等值线图 (Contour Plot)

2.5 Gradient descent -- 求最小 J(θ_0, θ_1) 的算法

θ 的初始值习惯上全部置为零(当然取别的值也无所谓)[1]。使用该算法得到的是局部最优解,不一定是全局最优解,如图2.4所示,如果起始点不同,那么可能会得到不同的局部最优解(local optimal)。

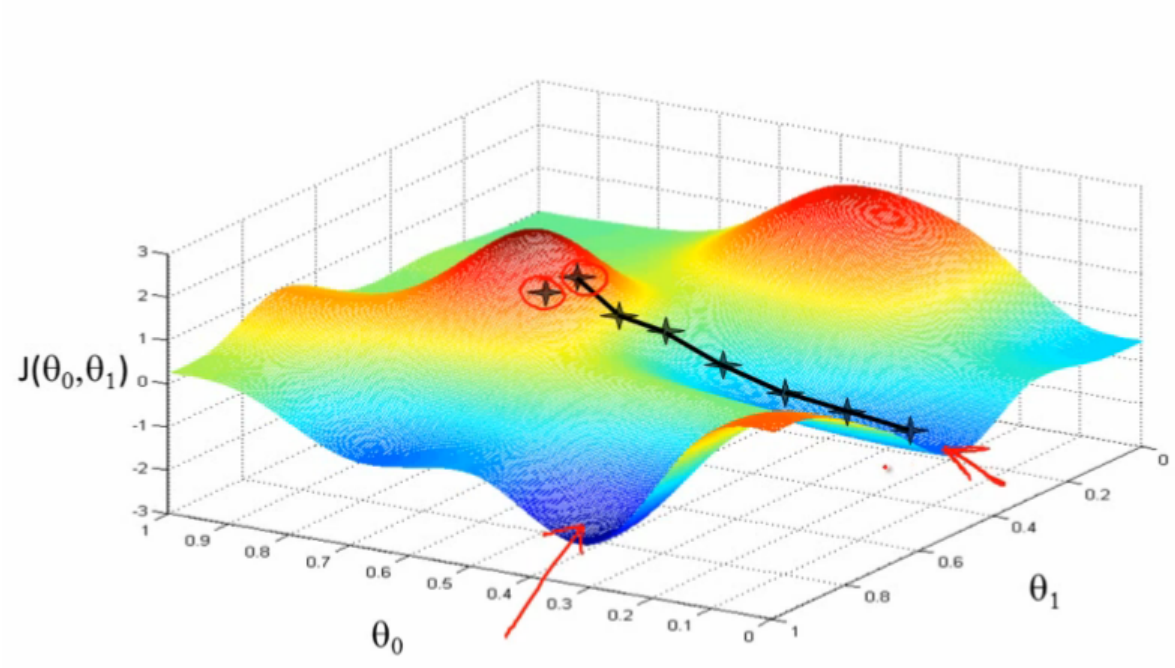


图 2.4: Gradient Descent 的过程

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

[1]在神经网络中, Θ 的初始值却不能全部为 0,详见9.5。

}

:= 赋值号

α learning rate. Controls how big a step we take downhill with gradient descent. 即梯度值,是个正数。

偏导数 (partial derivative term) 见下一节。偏导数和导数 (用 d 而不是 ∂ 表示) 的区别在于参数个数, 该题中有两个参数,只对一个求导,所以是偏导 (partial)。

注意,我们需要同时更新 (Simultaneous update)θ_0 和 θ_1(见图2.5)。

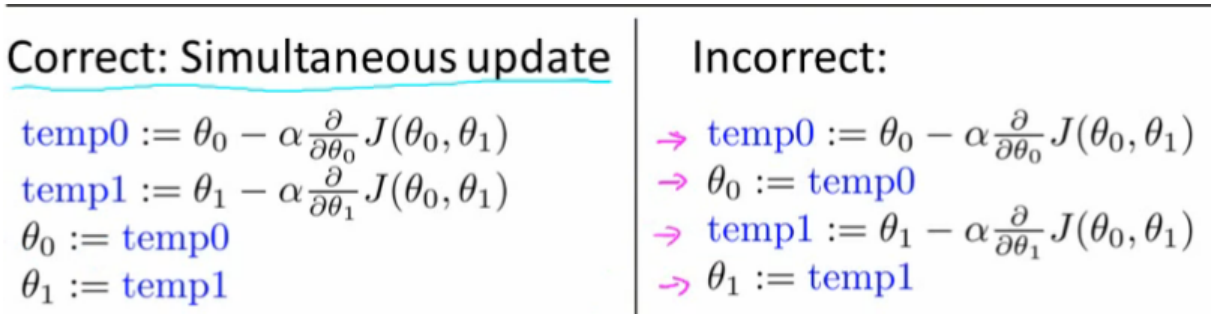


图 2.5: 同步更新

2.6 Gradient descent intuition

见图2.6,以 θ_1 为例,如果落点在最小点右边,那么函数偏导是负数, θ_1 的值会随之减小(向最小点靠近);如果落点在最小点左边,那么求导结果是负数,那么 θ_1 的值会随之增大。最终的结果就是使 θ_1 落在或逼近最小点。

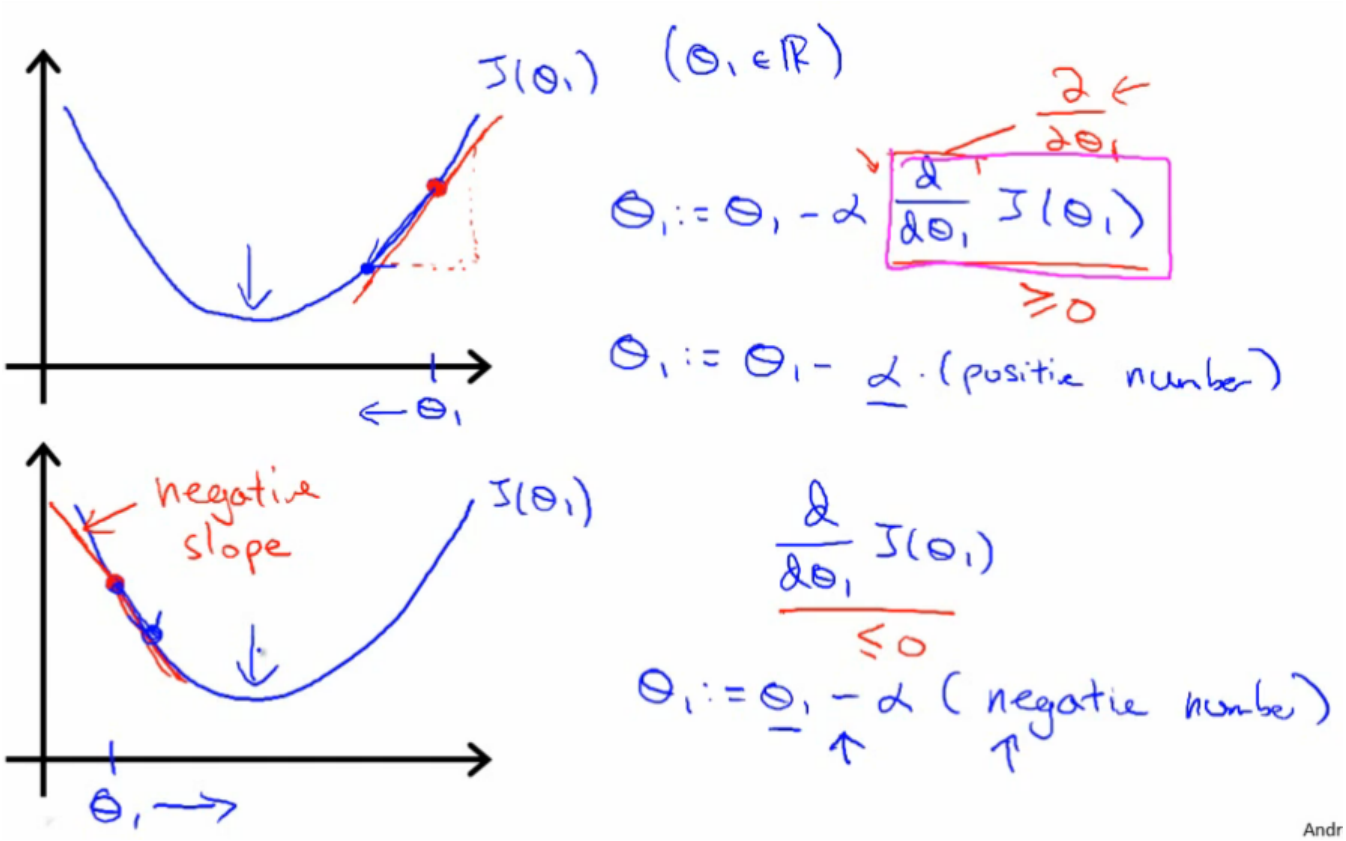


图 2.6: 直观理解 Gradient Descent

2.7 Gradient descent for linear regression

"Batch" Gradient Descent. Each step of gradient descent uses all the training examples.

3 Linear Algebra Review

3.1 Matrices & Vectors

$\mathbb{R}^{4\times 2}$ 表示 4 行 2 列的实数矩阵。

Vector $n \times 1$ 的矩阵。$\mathbb{R}^n$ 表示 n 维的实数向量。

3.2 加法

3.3 矩阵 X 向量

矩阵 X 向量,矩阵的列数 = 向量的行数,得到的是另一个向量。
一般的矩阵乘法规则如下:
简单来说就是行 X 列,用"**行列式**"来记忆它的运算顺序吧,恰好得到的结果是前面矩阵的行数 X 后面矩阵的列数的新矩阵。
做乘法必须保证前面矩阵的列数 = 后面矩阵的行数,可以用"**前列腺**"来记忆……、

3.4 矩阵 X 矩阵

很简单,矩阵和一个个的向量分别计算得到一个个的结果向量,然后组成新矩阵,可见图3.1

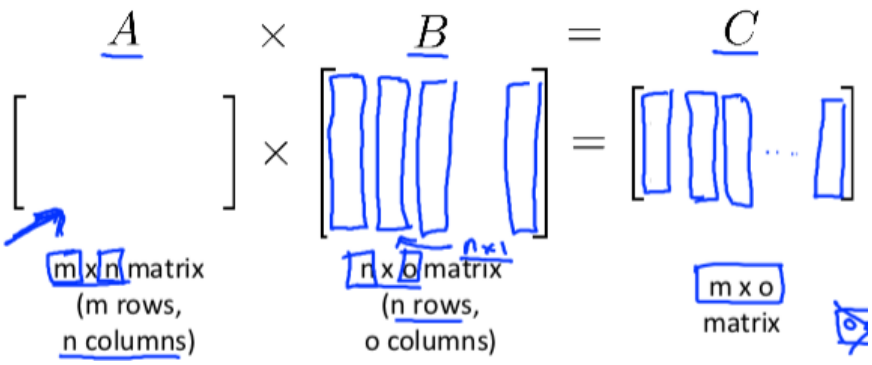


图 3.1: 矩阵相乘示意图

3.5 矩阵乘法的性质

1. Not Commutative 不符合交换律,如图3.2所示。
2. associative 符合结合律(也符合分配律)
3. Identity Matrix 单位矩阵
   $A \cdot I = I \cdot A = A$,这里 I 是单位矩阵,但注意两个 I 并不一定相同($m \times n$ 和 $n \times m$)。

3.6 矩阵求逆 Inverse 和转置 Transpose

3.6.1 求逆:

$$AA^{-1} = A^{-1}A = I \tag{3.1}$$

其中 A 为方阵(Square Matrix),只有方阵可以求逆。教材没有给出矩阵求逆的算法,而是用软件(Octave)实现的:pinv(A)

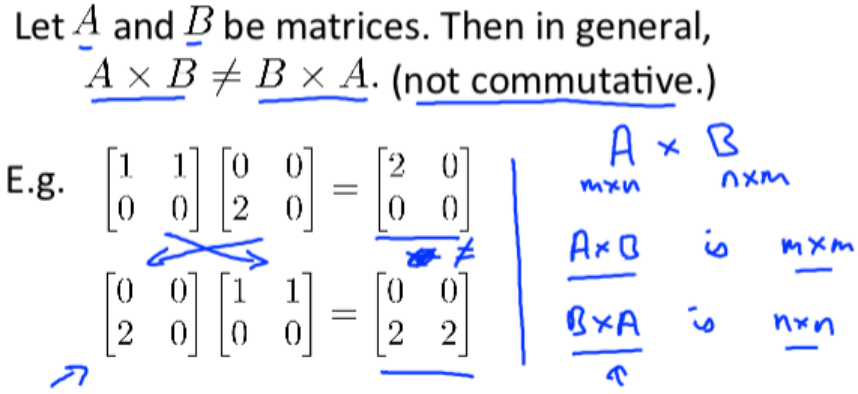


图 3.2: 矩阵相乘不符合交换律

3.6.2 转置

记矩阵 A 的转置为 A^T, $A_{ij} = A^T_{ji}$

$$(AB)^T = B^T A^T \tag{3.2}$$

4 Linear Regression with Multiple Variables

4.1 Multiple Features

Notation:
n = number of features
$x^{(i)}$ = input (features) of i^{th} training example. 对应训练集中的一条输入(一个向量)
$x^{(i)}_j$ = value of feature j in i^{th} training example. 向量 $x^{(i)}$ 中的第 j 项。
多个 feature 的 Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$
假设 $x_0 = 1$,且将 x 和 θ 均表示为向量,那么上式可写为:

$$h_\theta(x) = \theta^T x \tag{4.1}$$

通常被称作 multivariate linear regression(多元线性回归)

4.2 Gradient descent for multiple variables

Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$
Parameters: 将参数 θ 看作 n+1 维的向量 θ。
Cost function: $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$
Gradient descent:

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

多变量 Gradient Descent 如图4.1所示。

4.3 Gradient descent in practice I:Feature Scaling

Make sure features are on a similar scale。
推荐的方案是把 Feature 的值控制在 -1 到 1(或其他一个相近的)范围内,便于快速收敛。
如上例,两个 feature 相差 3 个数量级,这样就会导致 contour plot (等值线图)画出来非常的陡长(一个个很长的椭圆)。这样导致的后果便是,Gradient desent 需要很长的时间才能收敛到最优解。

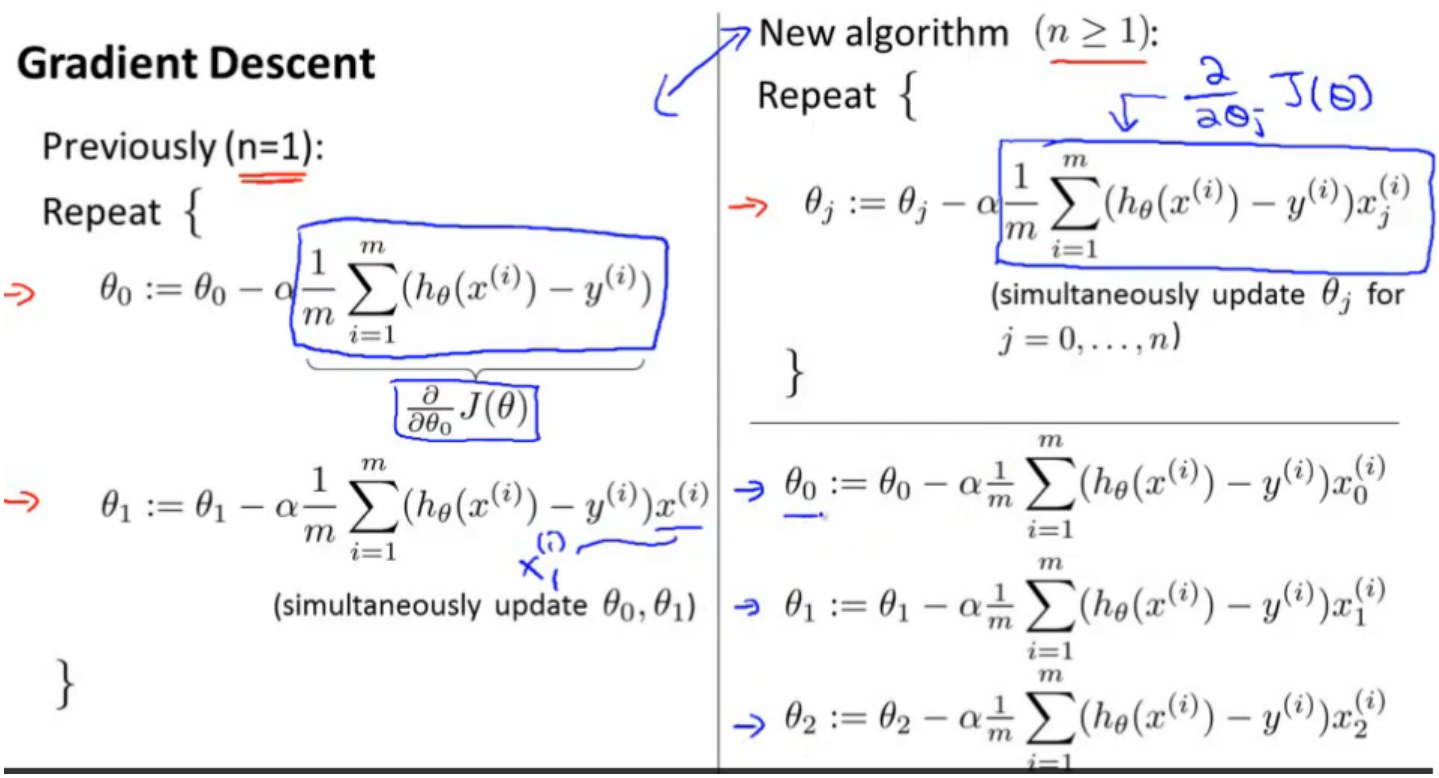


图 4.1: 多变量的 Gradient Descent

4.3.1 Mean(平均值)normalization

将 x_i 替换为 $x_i - \mu_i$ 使 features 的平均值接近于 0。
结合这两种方法,我们可以使 x_i 取如图4.3所示的值。

4.4 Gradient descent in practice II: Learning rate(α)

α 太大可能不收敛,太小可能收敛太慢。来回试吧 ~

4.5 Features and polynomial regression

可以根据已有的 feature 定义新的 feature 来简化模型。比如卖房子的例子,假定我们有房子的长和宽两个 feature,那么我可以定义房子的面积作为一个新的 feature,这样就只剩下一个 feature 就是房子的面积了。

另外可以使用多项式回归的方法,定义原 feature 的平方、立方等,增大数据拟合的精确度,如图4.4所示。

4.6 Normal equation 正规方程

Normal equation Method to solve for θ analytically.

将 features 和 y 表示成如图4.5所示的矩阵 (design matrix) 和向量的形式后,最小化 cost function 的 θ 值为(课程并没有给出这个公式的证明):

$$\theta = \left(X^T X\right)^{-1} X^T y \tag{4.2}$$

在 Octave 中表示为 pinv(x' *x) *x' *y。
使用 normal equation 时,feature scaling 和 mean normalization 没有必要。
图4.6表示的是 Gradient Descent 和 Normal Equation 的区别:
当 n 不是很大时,使用 normal equation 是个很好的选择。然而 n 很大时,normal equation 的代价相当昂贵。

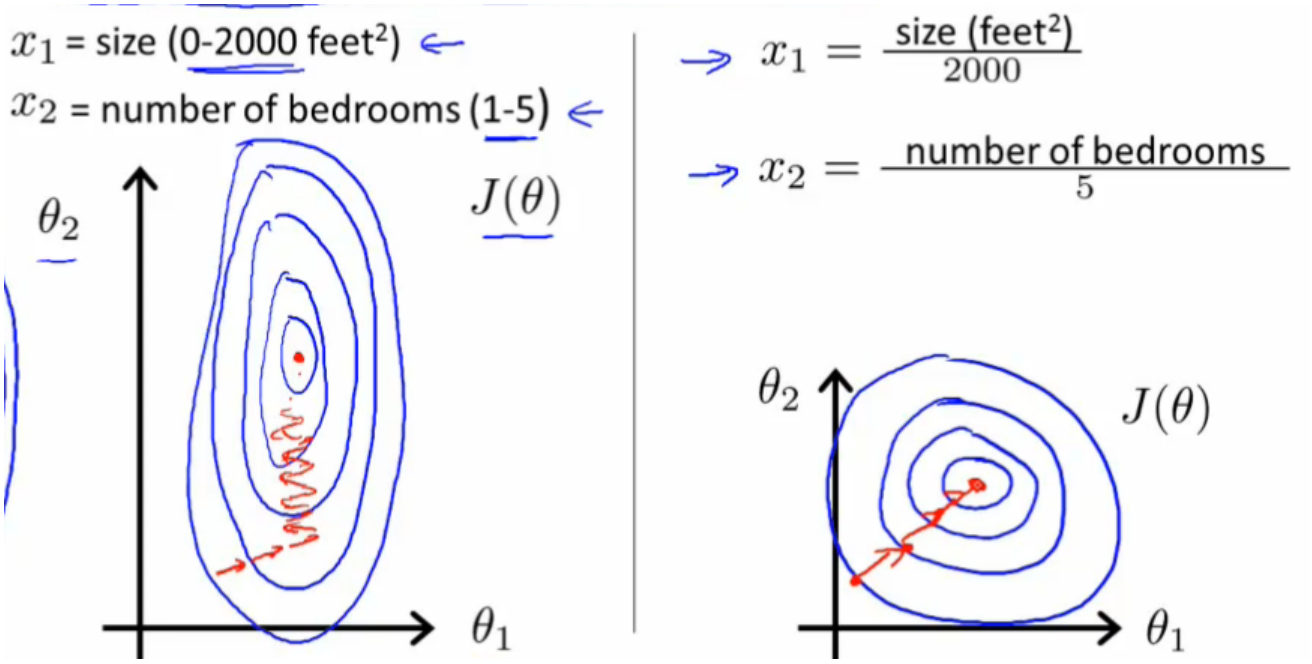


图 4.2: Feature Scaling

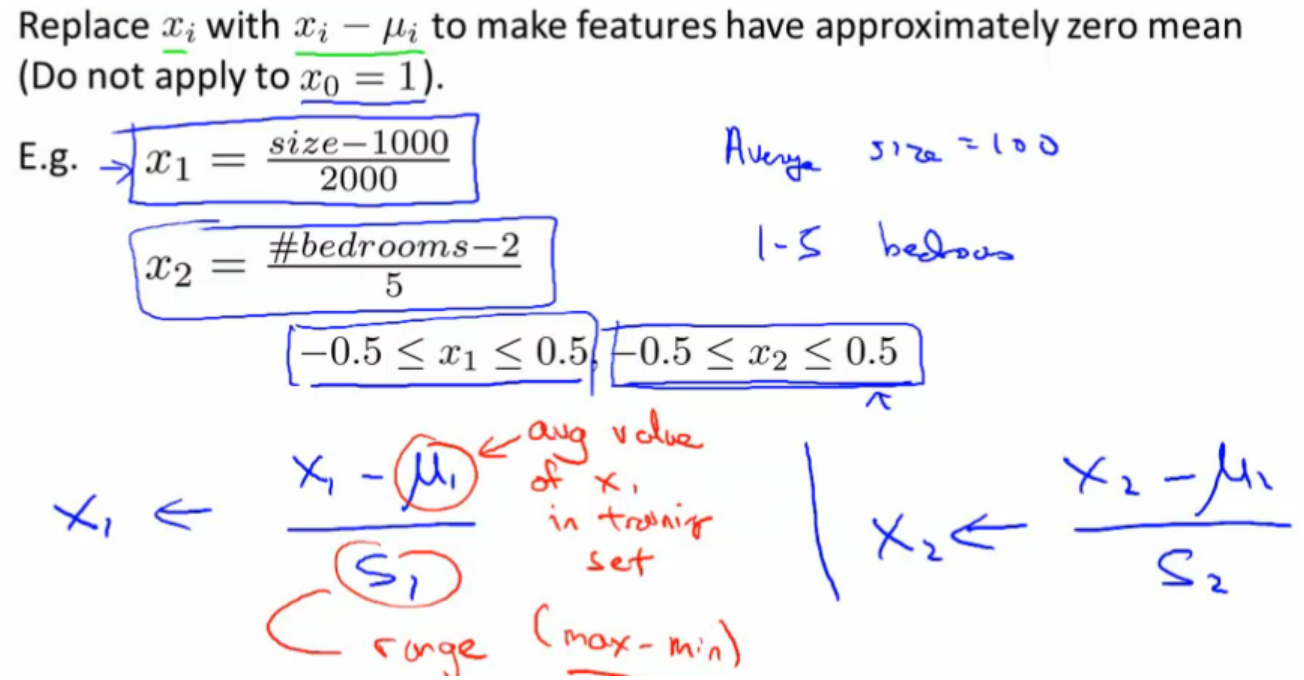


图 4.3: Mean Normalization

4.7 Normal equation and non-invertibility(不可逆性)

$$\theta = \left(X^T X\right)^{-1} X^T y \tag{4.3}$$

$X^T X$ 如果是不可逆的怎么办(虽然很少见)?(sigular/degenerate)[1]

Octave 中 pinv 和 inv 的区别:pinv 是 pseudo inverse, 伪求逆,即使矩阵不可逆他也可以求得我们需要的结果。

[1]矩阵 $A_{n\times n}$ 可逆的充分必要条件是 A 的行列式 $|A| \neq 0$

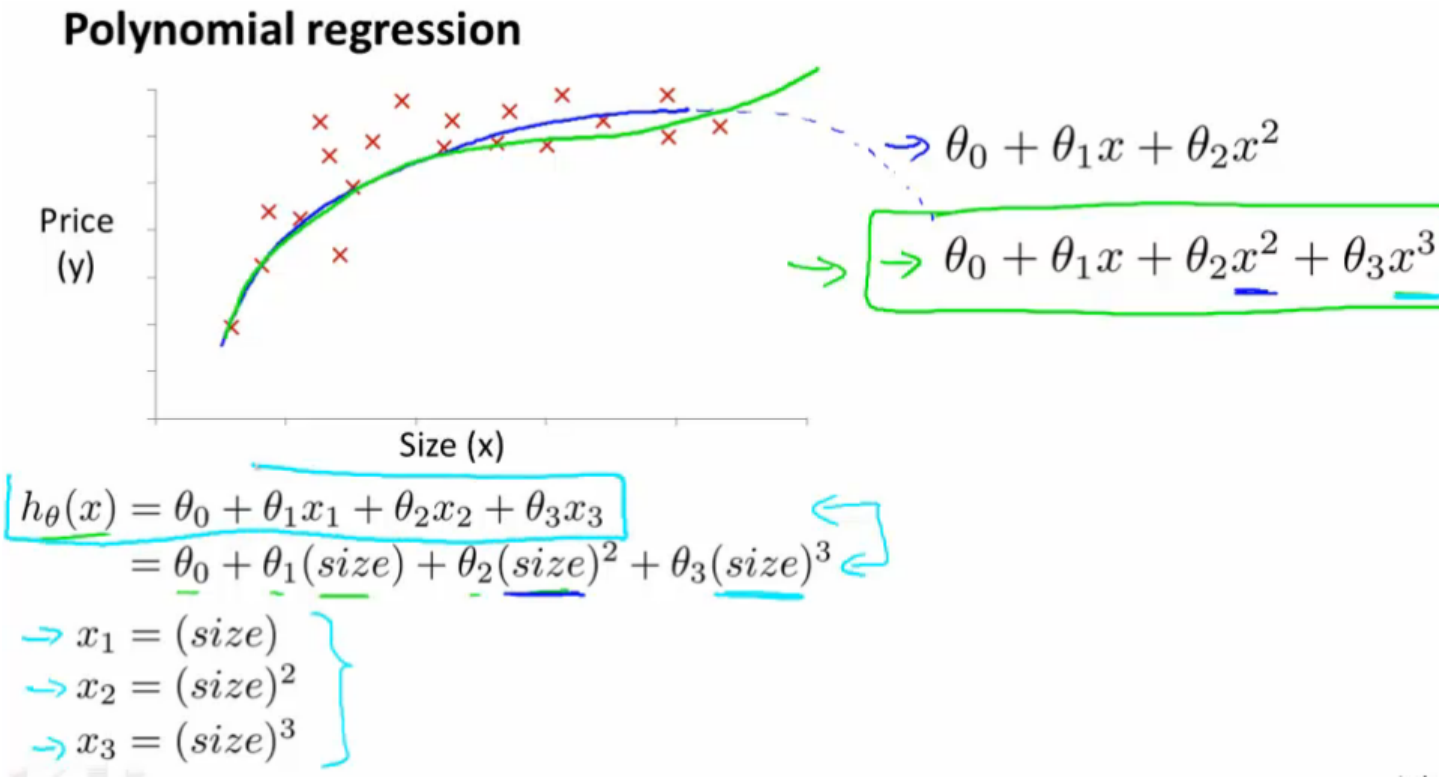


图 4.4: 多项式拟合

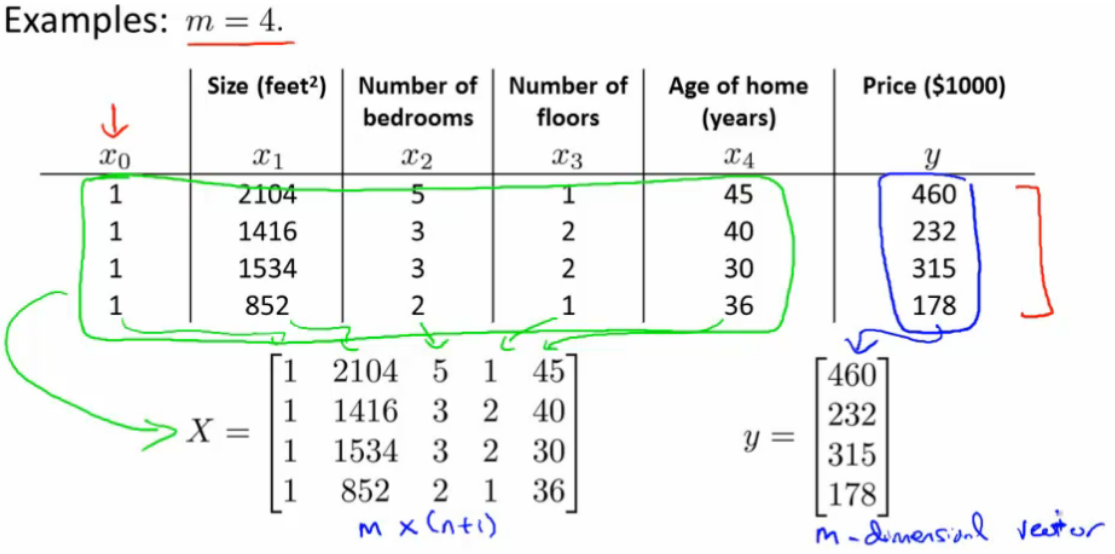
4.7.1 $X^T X$ 不可逆的原因？

- Redundant features(linearly dependenta 存在**线性**依赖).
  E.g. $x_1 = size\,in\,feet^2$, $x_2 = size\,in\,m^2$
- Too many features(e.g. $m \leq n$)
  Delete some features, or use regularization(talk later).

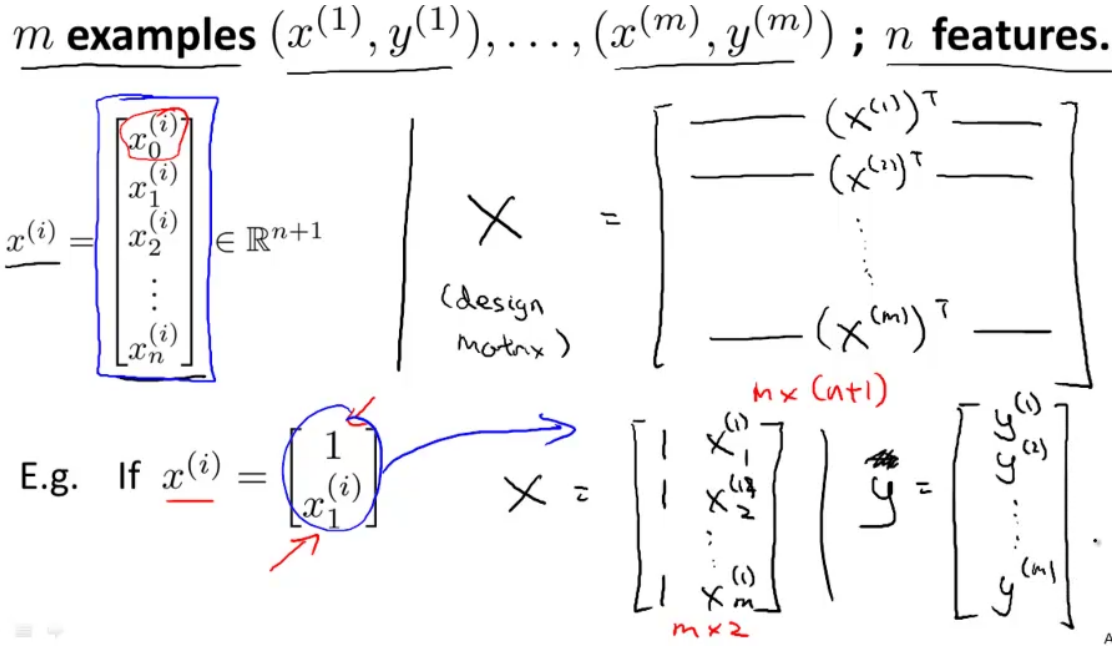
5 Octave Tutorial

5.1 Basic operations

- 声明矩阵:v=[1 2; 3 4]
- v= 1:0.2:2 的意思是,得到一个行向量(一行 N 列的矩阵),以 1 开始,到 2 结束,每次递增 0.2。
- v=1:6 得到 1 到 6 递增 1 的整数行向量。
- disp() 显示函数,使用它输出的结果不会带有 ans= 前缀。
- sprintf() 与 C 语言相似的格式输出函数。
- ones(2,3) 产生一个 2*3 的全是 1 的矩阵。
- zeros(3,5) 同理。
- rand(1,3) 得到一个 1*3 的介于 0 到 1 的随机数矩阵。(如果是方阵,只写一个参数即可)
- randn(1,3) Gaussian random variable。得到的随机数呈 0 对称的高斯(正态)分布
- hist(w) 绘制 w 的分布直方图 hist(w,50),显示 50 个统计条。
- eye(4) 产生 4 维的单位矩阵。eye 是 I 的同音。



(a) X 为 design matrix



(b) 构造 design matrix

图 4.5: 构造 Normal equation

5.2 Moving data around

- size(A),返回 A 的尺寸,如果 A 是矩阵,则返回 A 的行数和列数。比如 A 是 3*2 的矩阵,那么该命令返回一个矩阵 [3, 2]。如果执行 size(A, 1) 将返回 3,size(A, 2) 返回 2.
- length(A) 返回 A 较大的维数(可能是行数也可能是列数)。
- pwd 该变量保存 Octave 的当前工作目录(如/home/cherrot)。可以使用 cd 命令切换目录。
- who 用于显示当前环境中的所有变量。whos 给出细节
- load 文件名/ load(' 文件名') :加载数据文件。
- clear 变量名:清除变量。如果不跟变量则清除所有。
- v=priceY(1:10) :将 priceY 的前 10 个元素赋值给 v

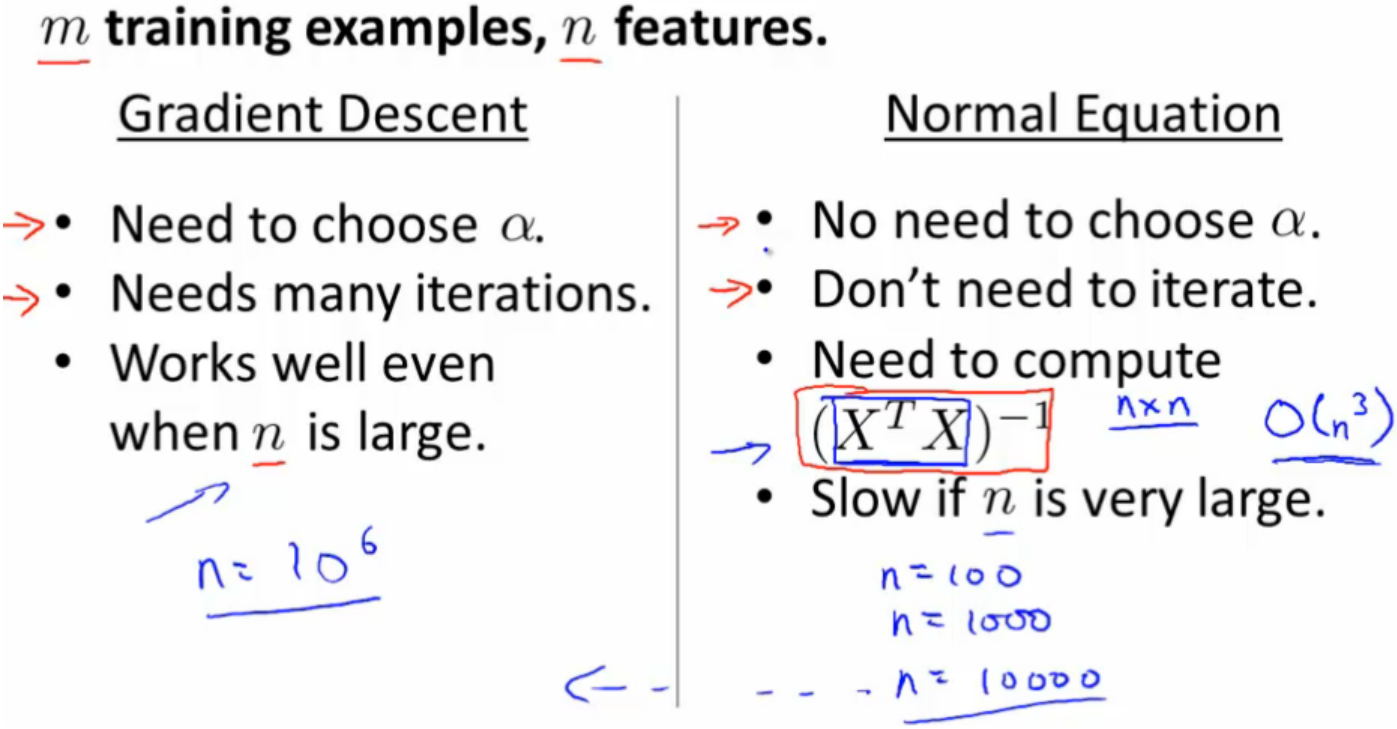


图 4.6: Gradient Descent 与 Normal Equation 的区别

- save hello.mat v :将 v 存到文件 hello.mat 中。默认为二进制形式。可以加参数 -ascii 以存为文本格式。
- A(3,2) :返回 $A_{3,2}$(从 1 开始,1-indexed)
  A(3,:) :返回 A 第三行的所有元素。":"代表该行/列的每一个元素
  A([1 3], :) :返回第一行和第三行的所有元素
- A=[A, [5; 6; 7]] :将向量 [5; 6; 7] 附加在 A 后面(右边)。
- A(:) 将 A 的所有元素打印成一个向量。
- C=[A B] :将矩阵 A、B 左右拼接(也可以用逗号分隔)。
- C=[A; B] :将矩阵 A、B 上下拼接。

5.3 Computing on Data

- A*B 矩阵乘法
- Element wise operations 比如 A .* B 将 A 中的**每个元素**与 B 中对应**元素**相乘构成新矩阵。
  点号在常被用于指示 element wise operations。比如 A .^2 的结果是 A 中每个元素都平方。
- 对矩阵应用 log() 或者 abs() 函数时,同样是 element wise operation。对矩阵取反、加/减一个常数值、大小比较都将看作是 element wise operation。
- A' :A 的转置。
- max(A):如果 A 是矩阵,则返回 A 中每一列的最大元素;如果 A 是向量或行向量,则返回 A 中最大元素,而且如果写作 [value, index] = max(A),将返回值和其 index。
- find(V<3):返回向量 V 中小于 3 的元素下标
  如果要 find 一个矩阵,可以写成 [i, j] = find(A)。对应的 i 和 j 分别为找到元素的行列 index。
- magic(n): 创建一个 n*n (n!=2) 的 magic 方阵。即每行、每列和两对角线上的元素和都为同一个数。
- sum() 求和。如果 sum(A,**1**) 则对 A 每一列求和,等效于 sum(A) 。
  sum(A,**2**) 对每一行求和。

- prod() 求积
- floor() 向下取整
- ceil() 向上取整
- max(A, [], **1**) :column wise maximum。找到每列的最大值,得到一个行向量。1 means to take max along the first dimension of A.
- max(A, [], **2**) :row wise maximum。找到每行的最大值,得到一个向量
- 要想找到矩阵 A 中的最大值,可以 max(max(A)), 或者 max(A(:))——即先将 A 表示成一个向量,在求最大值。
- flipud() :flip up & down,上下颠倒。故 flipud(eye(3)) 的效果就是把单位矩阵的对角线对换。

5.4 Plotting data

如果要在一个 Figure 面板中绘制两个曲线,那么需要调用 hold on 命令:

```
plot(t, y1);
hold on;
plot(t, y2, 'r'); % 'r' 表示本事件的颜色为红色。
xlabel('time')
ylabel('value')
legend('sin', 'cos') % 添加图例。
title('my plot')
print -dpng 'myPlot.png' % 保存为 png 文件
close % 关闭 plot
```

如果要绘制多于一个的图像时:

```
figure(1); plot(t, y1);
figure(2); plot(t, y2);
```

如果要在一个 figure 中绘制两个图,可以使用 subplot

```
subplot(1, 2, 1); %Divides plot a 1x2 grid, access first element
plot(t, y1);
subplot(1, 2, 2);
plot(t, y2);
axis([0.5 1 -1 1]); % 改变右边图片的 x y 坐标范围,x 从 0.5 到 1,y 从 -1 到 1。
clf; % 清空图像
```

如果要“绘制”一个矩阵:

```
A=magic(5) % 生成一个矩阵
imagesc(A)
```

或者改成灰度显示一个矩阵:

```
imagesc(A), colorbar, colormap gray; % 逗号可以使多个函数写在一行并且不屏蔽输出(异于分号)。
```

5.5 For, While, If statements, and Functions

5.5.1 For

e.g.1:

```
v = zeros(10,1);
for i=1:10,
    v(i) = 2^i;
end;
```

e.g.2:

```
indices = 1:10; %indices 初始化为一个行向量
for i = indices
    disp(i);
end;
```

5.5.2 While

e.g.1

```
i = 1;
while i <= 5,
    v(i) = 100;
    i = i +1;
end;
```

e.g.2 使用 break;

```
i = 1;
while true,
    v(i) = 999;
    i = i + 1;
    if i == 6,
       break;
    end;
end;
```

5.5.3 If-else

e.g.1

```
if v(1) == 1,
    disp('one');
elseif v(1) == 2,
    disp('two');
else % 这里没有逗号？
    disp('other');
end;
```

5.5.4 Funtions

Octave 定义函数的方式为把函数写四则算法在一个 function_name.m 文件中。文件内容范例：

```
function y = function_name(x)
y = x^2;
```

定义的函数文件必须在当前工作目录或者 Octave 的 search path 下。可以通过 addpath('new_path') 函数来增加 search path

Octave 的函数可以返回多值，比如返回 [y1,y2]。

5.6 Vectorization 向量化

把变量向量化使计算更加高效(Octave 擅长矩阵运算)，而且代码量更小(减少了循环等流程控制语句的使用)。

6 Logistic Regression

6.1 Classification

Classification 仍然属于 Supervised Learning，只不过是离散的值
应用场景：

- Email: Spam / Not Spam
- Online Transactions(交易): Fraudulent(欺诈)?
- Tumor(肿瘤); Malignant / Benign?

本节将讨论 binary classification，multi-class classification 将在以后讨论。
一种做法是继续使用线性回归 $h_\theta(x)$ 预测，然后定一个阈值 (Threshold Classifier)，比如取 0.5 作为阈值，如果 $h_\theta(x) \geq 0.5$，则预测 y=1，否则预测 y=0。但是这并不合适：

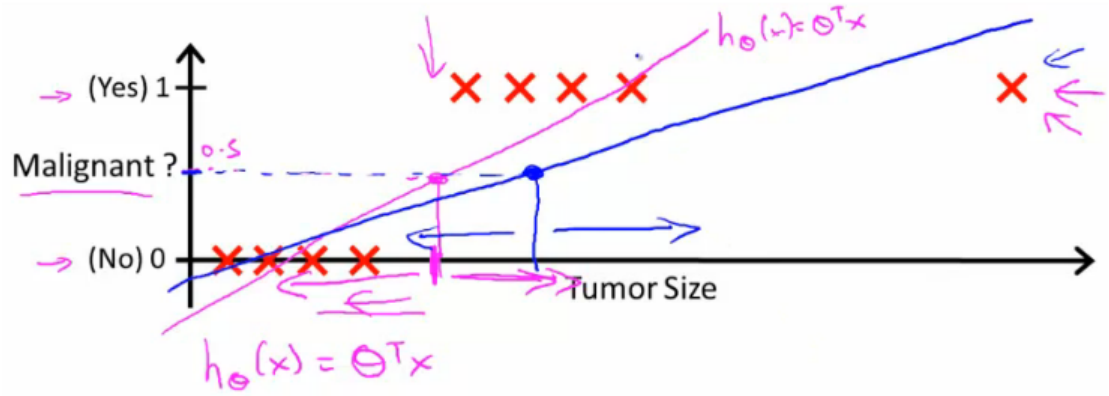


图 6.1: 使用线性回归处理 Classification 的问题

正如图6.1所示，假设没有最右边的值，回归曲线应该是洋红色的这条，那么我们取阈值为 0.5，预测的结果完全符合样本。然而，如果多了最右侧的这个取值时，回归曲线看起来应该像蓝色这条一样，当我们取阈值 0.5 时可以发现，预测值不再符合样本了。

Logistic Regression: 总能产生 $0 \leq h_\theta \leq 1$ 的 h_θ 的算法(在 y=0 或 y=1 的 binary classification 的情况下)。

6.2 Hypothesis Representation

为了得到 $0 \leq h_\theta(x) \leq 1$，则定义

$$h_\theta(x) = g\left(\theta^T x\right) \tag{6.1}$$

其中

$$g(z) = \frac{1}{1+e^{-z}} \tag{6.2}$$

上式称为 sigmoid function(S 形函数) 或 logistic function(逻辑函数)。$\theta^T x$ 可见公式4.1。公式6.2的图形如所示：

6.2.1 Interpretation(解释)of Hypothesis Output

$h_\theta(x) = P(y=1 \mid x;\theta)$. It **estimats** probability that y=1, given x, parameterized by θ.

6.3 Decision Boundary

只有当 $z \geq 0$ 时，$g(z) \geq 0.5$ 才成立，故只有 $\theta^T x \geq 0$ 时，$h_\theta(x) \geq 0.5$ 才成立。
Decision Boundary 就是使 $\theta^T x = 0$ 的直线，如图6.3所示：
图6.3中 $x_1 + x_2 = 3$ 这条线就是 Decision Boundary。

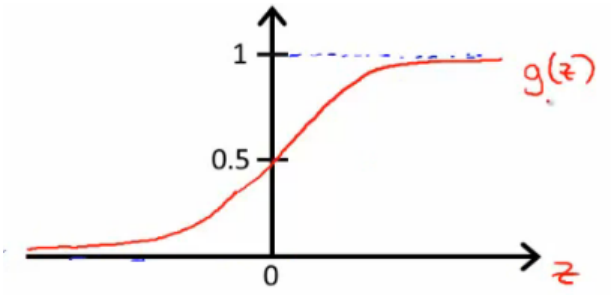


图 6.2: sigmoid function

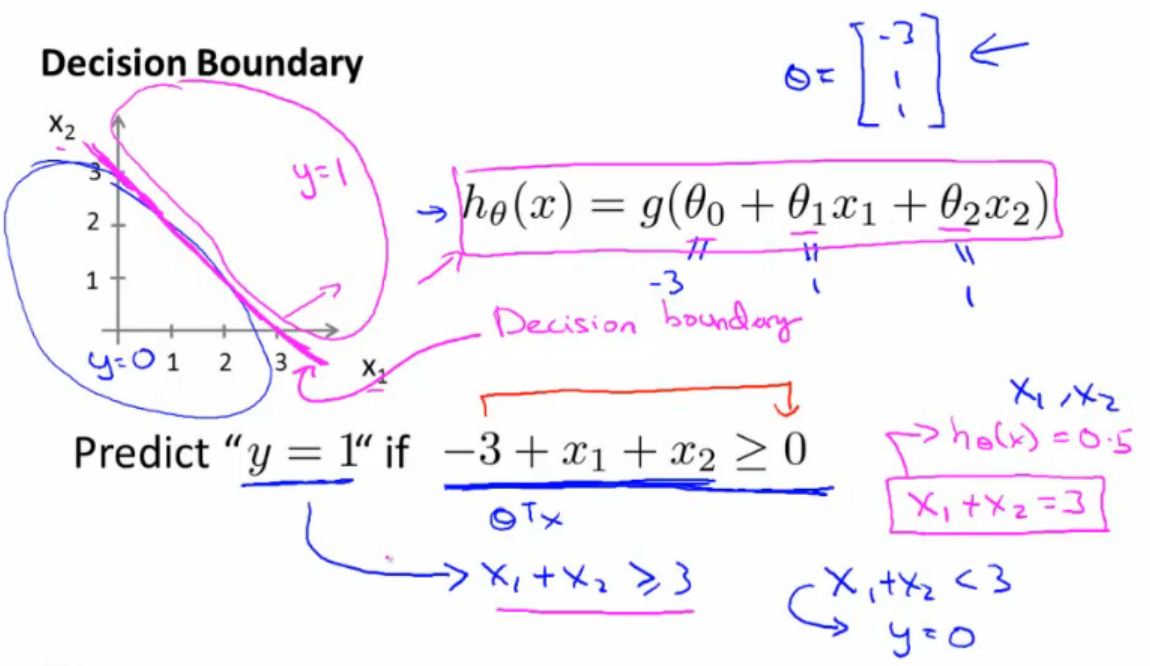


图 6.3: Decision Boundary

6.4 Cost function

如果将线性回归中的 cost function 用到这里的话,那么我们得到的将是一个 non-convex (非凸函数)的图形(因为 $h_\theta(x)$ 不是一个线性函数,而是一个平方代价函数 squre cost function),导致得不到全局最优解,如图6.4所示。

下面是 Classification 情况下的 Cost function(with single training example):

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) \tag{6.3}$$

其中,

$$Cost\left(h_\theta(x), y\right) = \begin{cases} -\log\left(h_\theta(x)\right) & if\ y = 1 \\ -\log\left(1 - h_\theta(x)\right) & if\ y = 0 \end{cases} \tag{6.4}$$

函数的图形可以直观的告诉我们上式的作用:

为方便计算,公式6.4可写为式6.5的形式。

$$Cost\left(h_\theta(x), y\right) = -y\log\left(h_\theta(x)\right) - (1-y)\log\left(1 - h_\theta(x)\right) \tag{6.5}$$

这里要注意当 y=1 时它的几个特性,如图6.6所示的。在 y=1 时,当 $h_\theta(x) = 1$ 时,Cost 为 0,也就是没有误差;相反,若 $h_\theta(x) \to 0$,那么则视为误差无限大,即 Cost 趋向于无穷。

The topic of convexity analysis is beyond the scope of this course.

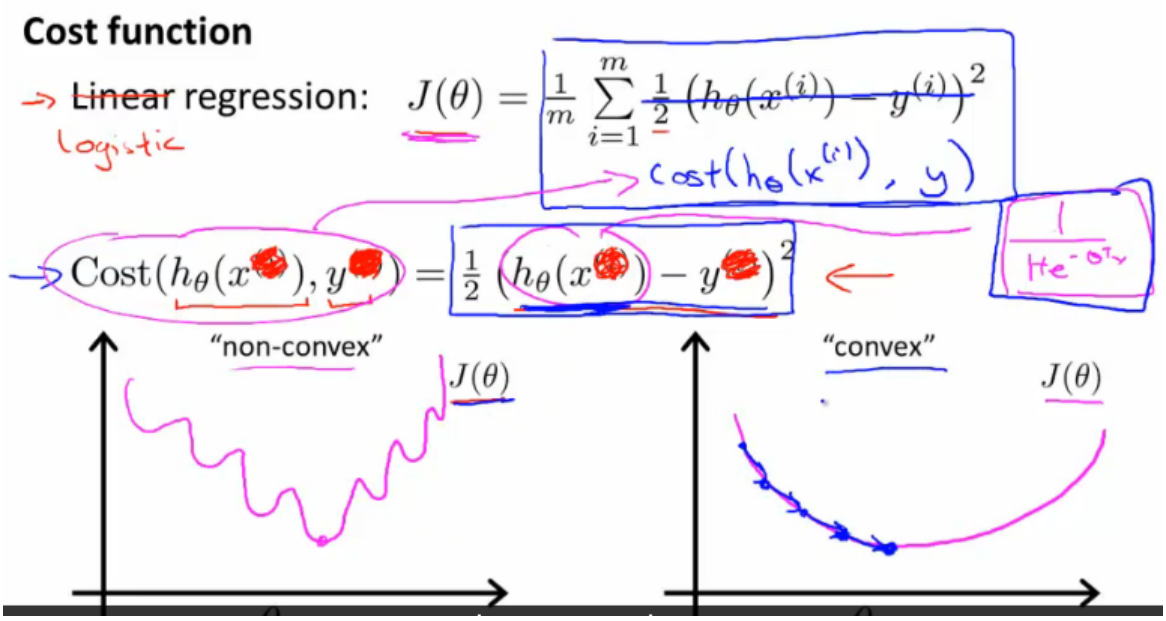


图 6.4: 直接使用线性回归的 cost function 导致结果为非凸函数

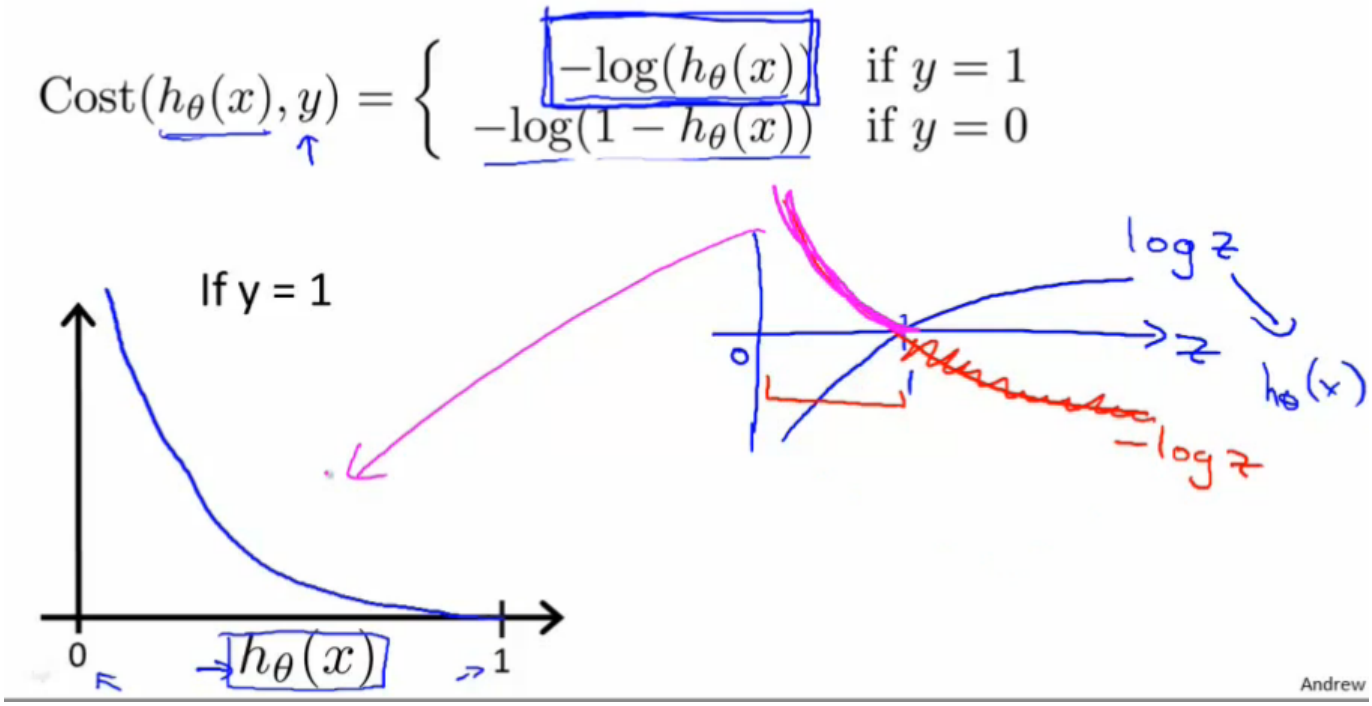


图 6.5: Plotting the Cost()

6.5 Simplified cost function and gradient descent

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right] \quad (6.6)$$

之所以使用这个公式作为 Cost Function 而不是使用其他的公式是因为 This cost function can be derived from statistics using the **principle of maximum likelihood estimation(最大似然估计)**, and it is convex. 更多的细节超出了本课程的讨论范围。

给出上式,我们需要寻找到合适的 θ 满足 $\min_\theta J(\theta)$,从而就可以计算 $h_\theta(x)$ 了。

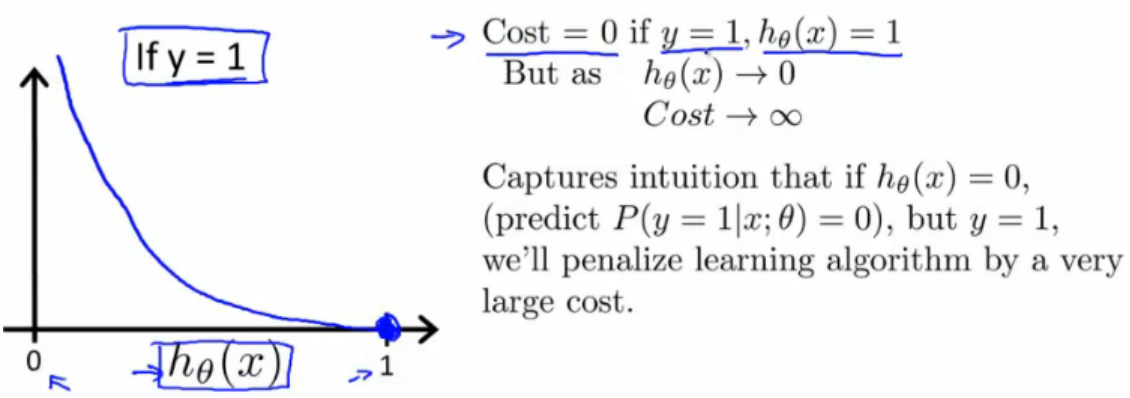


图 6.6: $Cost\left(h_\theta\left(x^{(i)}\right), y^{(i)}\right)$ 函数

6.5.1 Gradient Descent

Repeat{
$\theta_j := \theta_j - \alpha \frac{\delta}{\delta\theta_j} J(\theta)$
}(simultaneously update all θ_j)
其中 $\frac{\delta}{\delta\theta_j} J(\theta) = \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta\left(x^{(i)}\right) - y^{(i)}\right) x_j^{(i)}$,和线性回归的结果**相似**[1],见4.1。
该算法同样可以使用 feature scaling(见4.3节)来优化计算。

6.6 Advanced Optimization

Optimaization algorithms:

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

其他三个算法具体是怎么做的将不在本课程中涉及。这些算法的缺点是非常复杂,而优点是:

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

We can think of these algorithms as having a clever inter-loop. In fact the inter-loop is called the **line search** algorithm that automatically tries out different values for α and automatically picks a good learning rate so that it can even pick a different alpha for every interation.

本课程将不详细的讨论这些算法内部究竟是怎样工作的。作者使用这些算法好多年后才去探索它们是如何工作的 =.=

而且不建议自己写代码实现这些代码,有类库干吗不用?例如使用 octave 的自建函数 fminunc 更好的实现 gradient descent:(图6.7)

fminunc() 函数:function minimization unconstrained。是 Octave 中的内建函数,@ 表示一个指向函数的指针 (function handle)。fminunc 的作用就是计算使给定函数取值最小的参数值,如图6.8所示即为 Octave 的演示:

options 是存储我们需要的 option 的数据结构。'GradObj', 'on' sets the gradient objective parameter to on. It just means you are indeed going to provide a gradient to this algorithm. 其中 exitFlag=1 表示算法成功收敛 (converge) 了。optTheta 和 functionVal 都得到了期望的值(functionVal=0)。注意要使用这个函数,initialTheta 必须是不小于二维的向量。

- Octvave 语法:@(t) (costFunction(t, X, y)) . This creates a function, with argument t, which calls your costFunction.(实例见编程作业 2)

What we need to do is **write a function that returns the cost and returns the gradient** (见图6.9). 当然,我们甚至可以将这个算法使用在线性回归问题上。

[1] **怎么算的?** 我的猜想是,sigmoid 函数只是起函数域的限定作用,不改变原函数的增长性,所以为了计算简便,求导时不考虑 sigmoid 函数。求证。

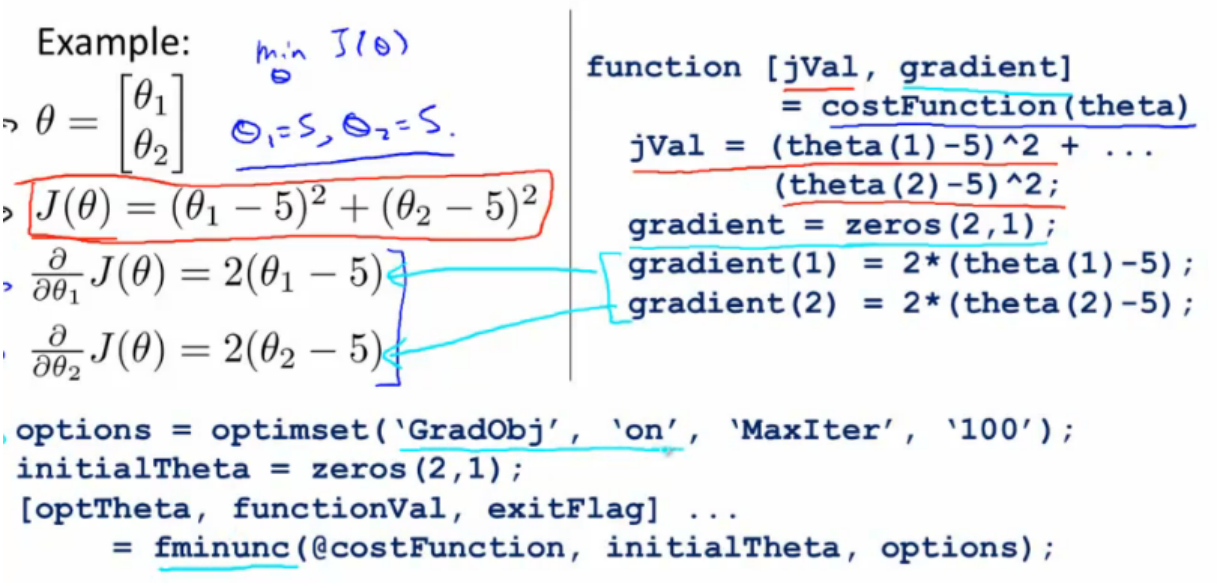


图 6.7: fminunc 函数的使用示例

```
octave:4> options = optimset ('GradObj', 'on', 'MaxIter', '100')
options =
{
  GradObj = on
  MaxIter = 100
}

octave:5> initialTheta = zeros (2,1)
initialTheta =

   0
   0

octave:6> [optTheta, functionVal, exitFlag]=fminunc(@testCostFunction, initialTh
eta, options)
optTheta =

   5.0000
   5.0000

functionVal =  1.5777e-30
exitFlag =  1
octave:7>
```

图 6.8: Octave 演示 fminunc 函数

6.7 Multi-class classification: One-vs-All

实例:

- 标记 E-mail 类型:Wor, Friends, Family, Hobby。
- 诊断:Not ill, Cold, Flu
- 天气:Sunny, Cloudy, Rain, Snow

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

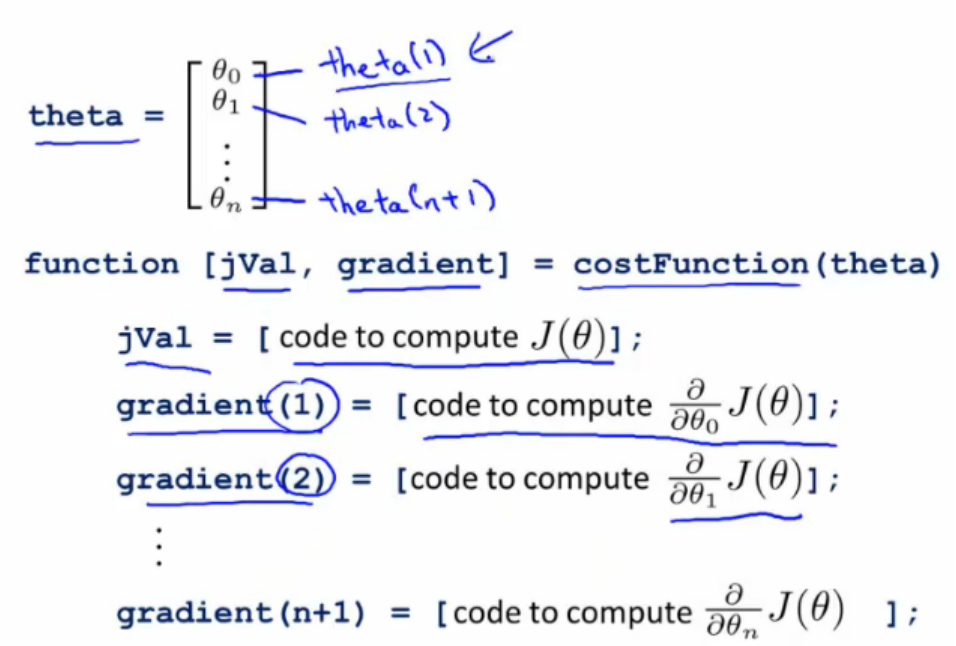


图 6.9: 使用 fminunc 时需要自定义的函数

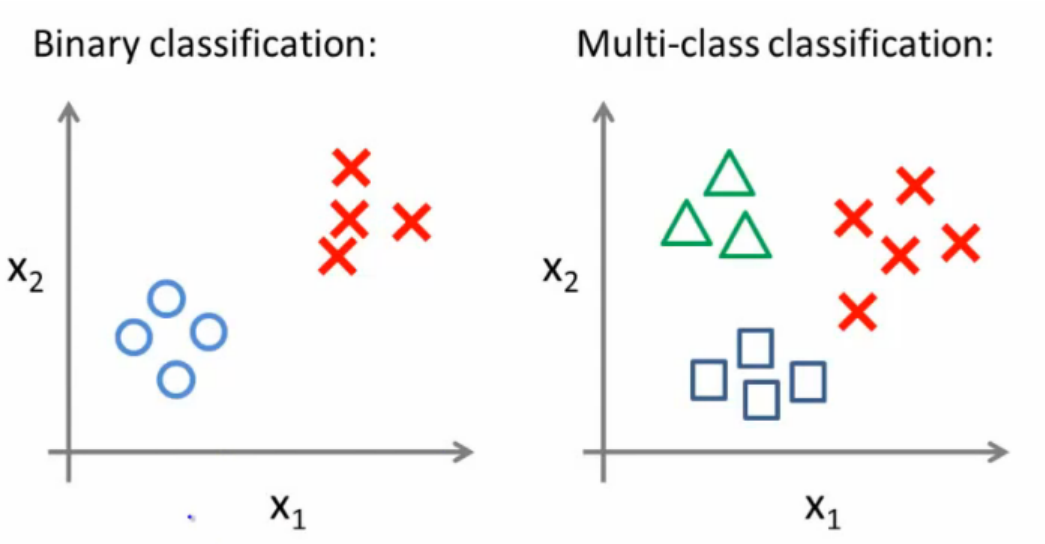


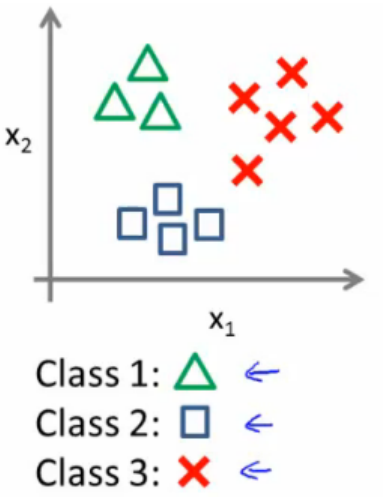
图 6.10: Multi-class classification 问题

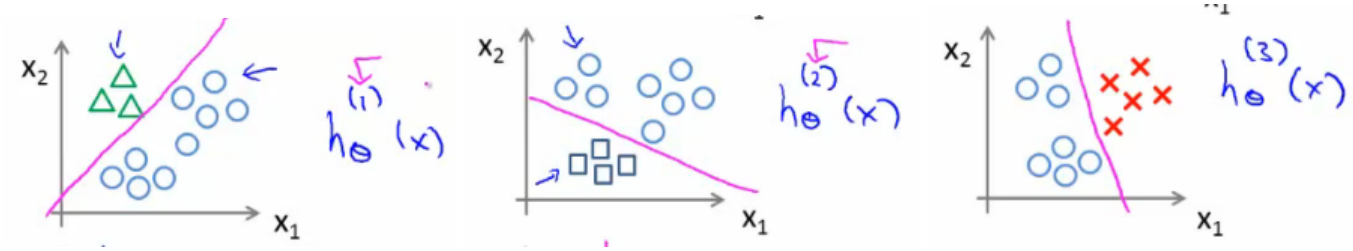
What we are going to do is take a training set, and turn this into three **separate binary classification problems:**(如图6.11所示)

$$h_\theta(x) = P(y = i \mid x; \theta), \quad (i = 1, 2, 3) \tag{6.7}$$

Train a logistic regression classifier $h_\theta^{(i)}(x)$ for each class i to predict the probability that $y = i$.

On a new input x, to make a prediction, pick the class i that maximizes $\max_i h_\theta^{(i)}(x)$.





(a) (b) (c)

图 6.11: Multi-class 的分解

7 Regularization

7.1 The problem of overfitting

features 太少导致 underfitting(high bias),而 features 太多就会导致 overfitting(high variants),正如图7.1和图7.2所示。

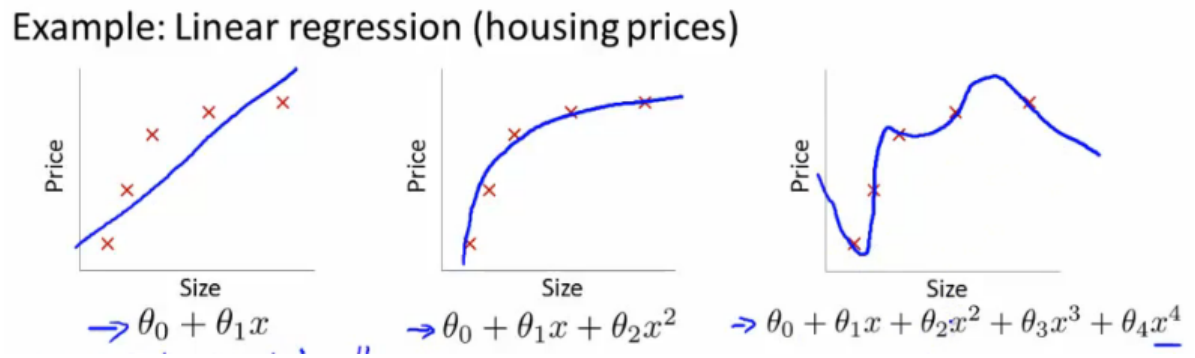


图 7.1: Underfitting and Overfitting in Linear Regression

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

7.1.1 Addressing overfitting

Options:

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 θ_j.(减小参数的权重)
   - Works well when we have a lot of features, each of which contributes a bit to predicting y.

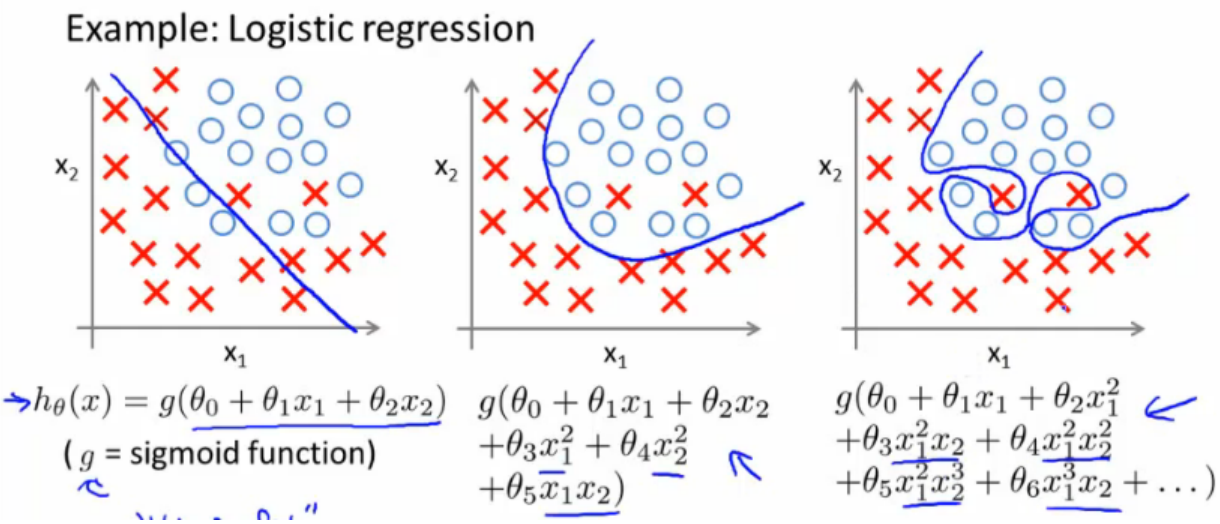


图 7.2: Underfitting and Overfitting in Logistic Regression

7.2 Cost function

7.2.1 Intuition

如图7.3所示,如果出现了过度拟合,我们可以使 θ_3 和 θ_4 变得非常小甚至接近于 0,从而降低这两个参数对整个图形的贡献。如图7.3所示,在原式后面加上 $1000\theta_3^2 + 1000\theta_4^2$ 两个因子(其中 1000 是随便指定的比较大的值),这样,如果要使原式取最小值,那我们必须使 θ_3 和 θ_4 取值很小,从而达到我们的目的。

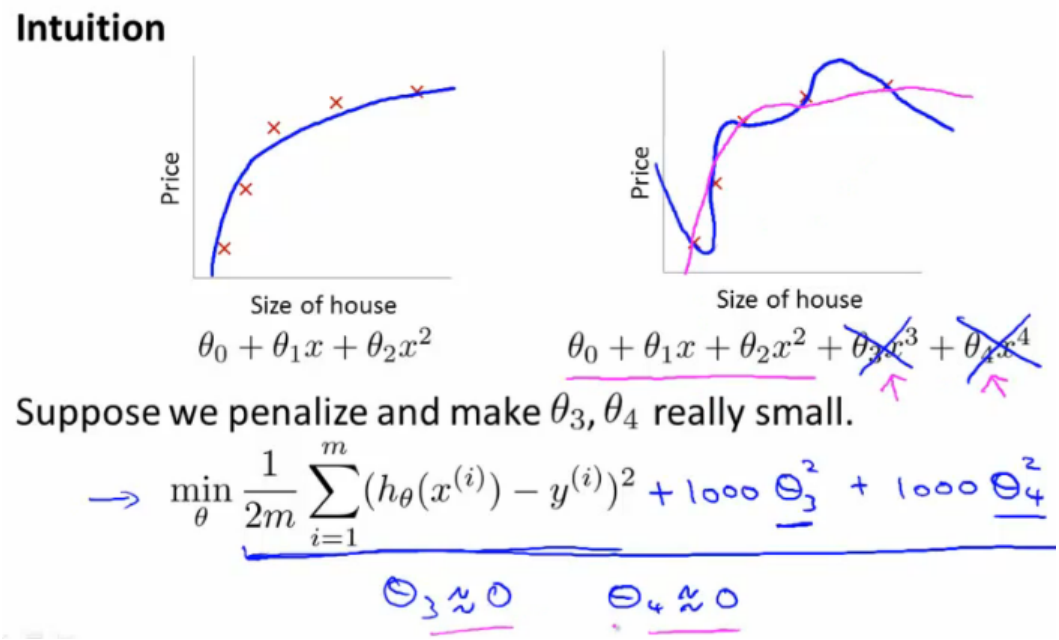


图 7.3: Penalize parameters instead of removing them

7.2.2 Regularization

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

- ``Simpler'' hypothesis
- Less prone(倾向)to overfitting.

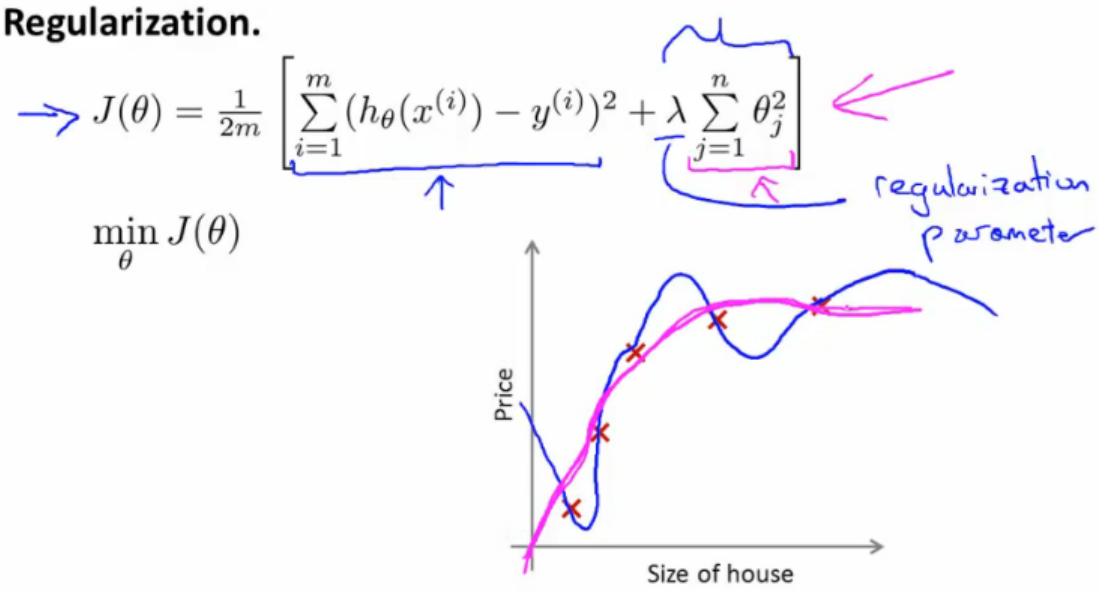


图 7.4: Regularization

如图7.4所示[1],Regularization 对每个参数 (θ) 都乘以一个惩罚因子 λ。What the regularization parameter does is it controls the tradeoff between the goal of fitting the data well and the goal of keeping the parameter small, and therefore keeping the hypothesis simple.

但是如果 λ 取值太大,则会造成 underfitting。道理很简单,right?

7.3 Regularized linear regression

7.3.1 Gradient descent

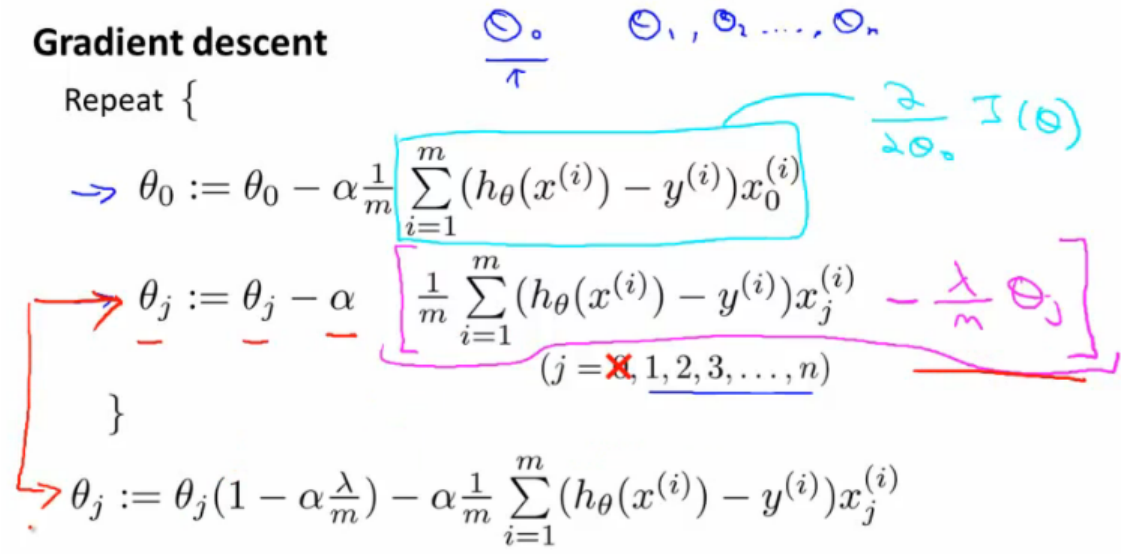


图 7.5: Gradient Descent with Regularization

注意 $1 - \alpha\frac{\lambda}{m}$ 是小于 1 的,这就相当于把 θ_j 缩小了,而后面的部分和之前的公式是一样的。中间公式中括号内的部分正是对 $J(\theta)$ 求导的结果。注意,这里有一处符号错误, $\frac{\lambda}{m}\theta_j$ 前应该是 +。

[1]注意 j 是从 1 开始取值的。这是个习惯上的做法,实际上从 0 开始或从 1 开始对结果不会有太大影响 (little difference)。

7.3.2 Normal equation

Gradient descent 只是两种线性回归拟合算法中的一个，另外一种是基于 normal equation 的，我们需要一个 design matrix $X_{m\times(n+1)}$ 和结果矢量 y。X 的每一行对应一个 training example。y 保存着每个 training example 的结果。关于算法的详细描述可参见4.6

$$\theta = \left(X^T X + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix} \right)^{-1} X^T y \tag{7.1}$$

上式中 λ 后面的矩阵是 $(n+1)\times(n+1)$ 的，如果去掉 λ 和其后的矩阵，那么就是之前学过的的 Normal Equation，这里是加上 Regularization。其实等号后的部分正是令 cost function 求导为 0 的结果：$\frac{\delta}{\delta\theta_j} J(\theta) \overset{set}{=} 0$。由于证明比较复杂，因此具体为什么是这个结果老师不作解释:D

我们知道，当 $m \le n$ 时，公式4.2中的 $X^T X$ 是不可逆的，然而可以证明，当使用 Regularization 后，加上 Regularization 因子后的矩阵一定是可逆的。

7.4 Regularized logistic regression

7.4.1 Gradient Descent

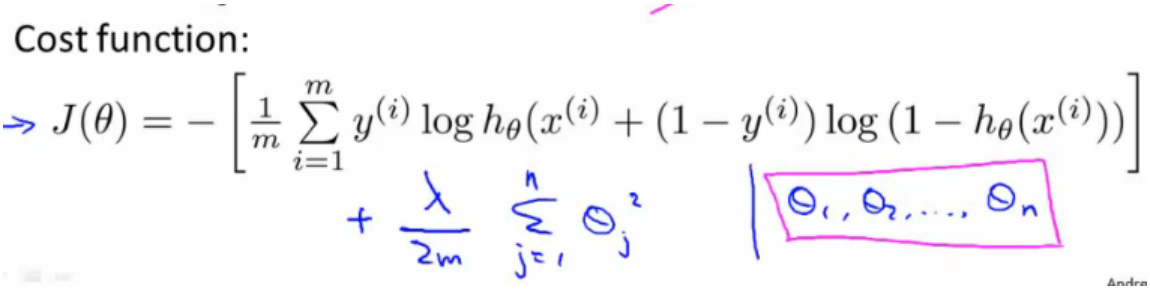


图 7.6: Cost function of logistic with reguarization

$$J(\theta) = -\left[\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)} \log h_\theta\left(x^{(i)}\right) + \left(1-y^{(i)}\right) \log\left(1-h_\theta\left(x^{(i)}\right)\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2 \tag{7.2}$$

在 logistic regression 中使用 Regularization 的 cost function 见图7.6和公式7.2，需要注意 j 仍然是从 1 开始的。

经过 Regularized 的 gradient descent 如图7.7所示，该图中仍有一处符号错误：$\frac{\lambda}{m}\theta_j$ 前应该是 +。注意，虽然看起来图7.7中的公式和线性回归中的类似，但该图中的 h_θ 和线性回归中的 h_θ 是完全不同的函数。

同上一节一样，中括号中的部分是 $J(\theta)$ 的偏导。

7.4.2 Advanced optimization

要想使用 regularized 的 advanced optimization，只需要将 cost function 的定义进行如图7.8所示的修改即可，注意图中仍然存在符号错误。

8 Neural Networks: Representation

8.1 Non-linear Hypotheses

当输入的 feature 集巨大时，Logistic Regression 将会力不从心。神经网络是用于计算复杂非线性回归问题 (complex non-linear hypotheses) 的好方法。

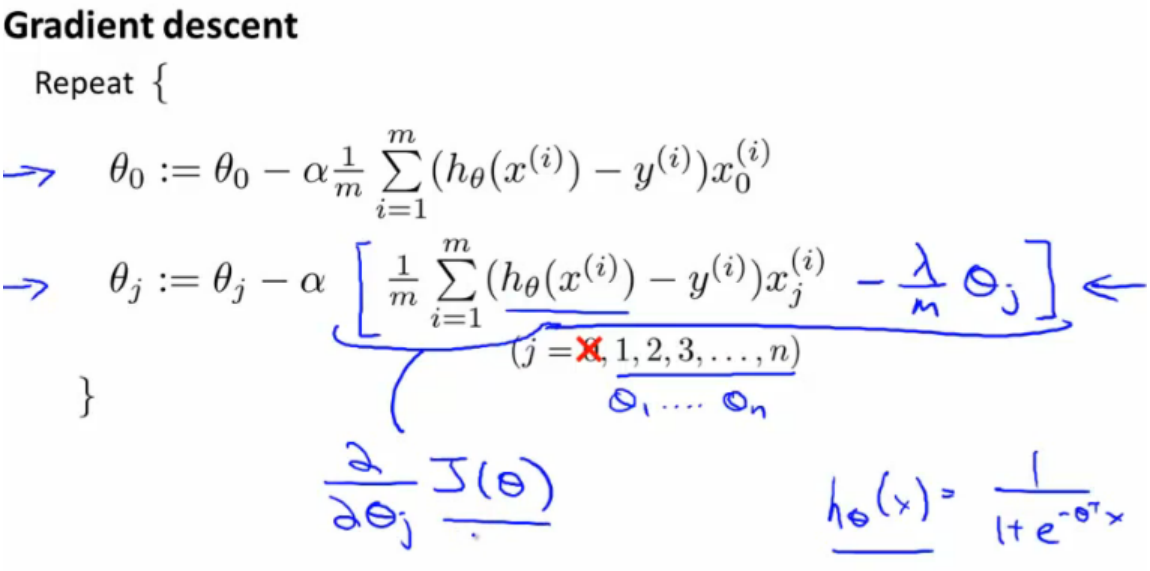


图 7.7: Gradient Descent of Logistic Regression with Regularization

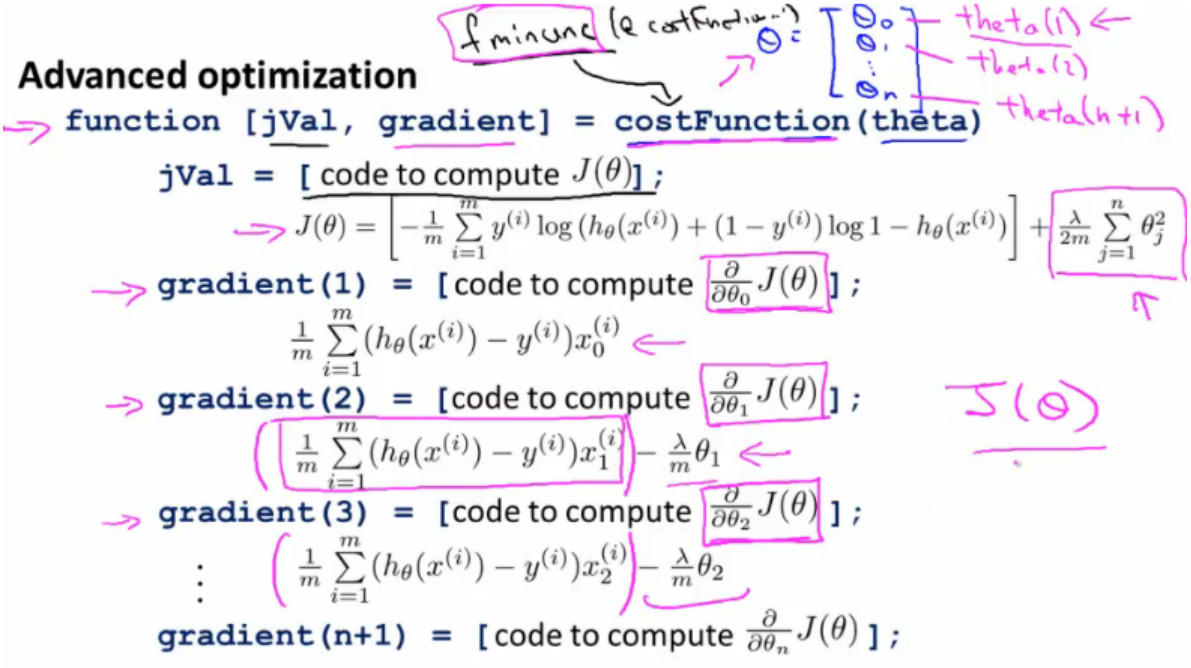


图 7.8: Advanced Optimization

8.2 Neurons(神经元)and the Brain

Neuro-rewiring experiments 比如把传入听觉中枢的神经 re-wire 到眼睛,那么听觉中枢会学习用眼睛看的功能。同一块大脑组织,其实可以处理听觉、视觉、触觉等等功能。看起来大脑有非常强大的学习算法:)

8.3 Model representation I

8.3.1 Nuron in the brain

大脑中的神经元结构如图8.2所示,图中 Dendrite 是树突,也就是输入端,Axon 是轴突,也就是输出端。

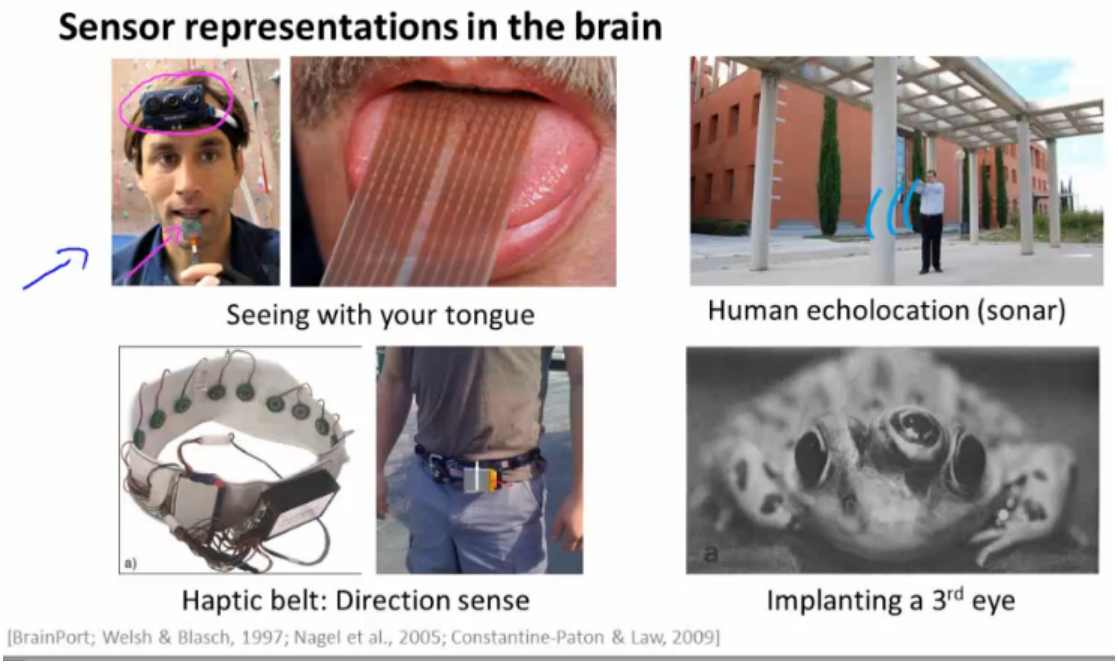


图 8.1: 大脑强大的学习能力

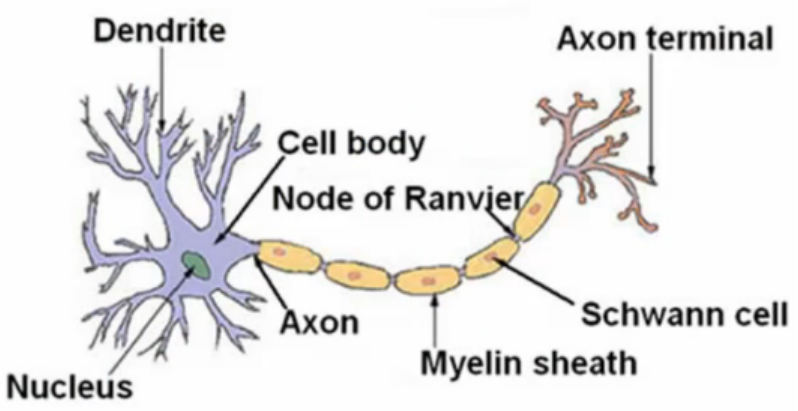


图 8.2: 大脑中的神经元结构

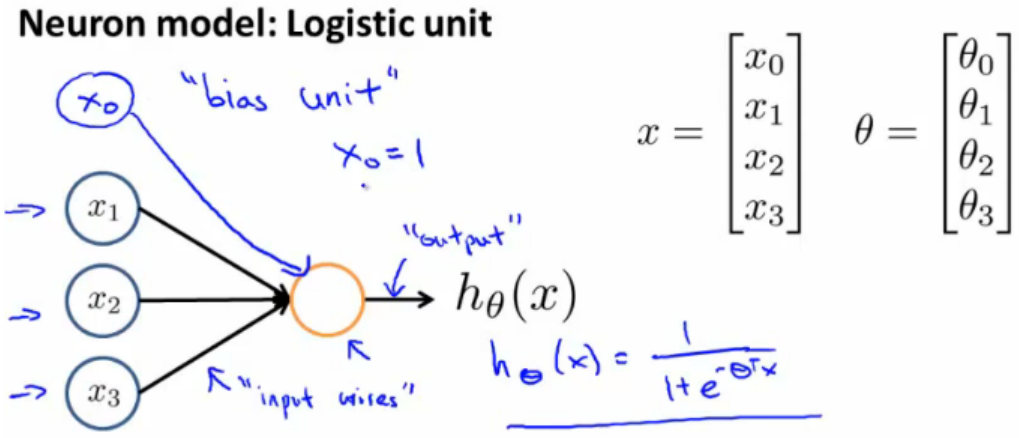


图 8.3: 人工神经元模型:逻辑单元

8.3.2 Neuron model: losgistic unit

我们将大脑中的神经元类比到人工智能网络中,一个简单的神经网络可见图8.3。

一般绘制神经网络图时会省略 x_0, $x_0 = 1$ 称为 bias unit 或 bias neuron。

When we talk about neural networks, sometimes we'll say that this is a neural network with a sigmoid(logistic) **activation function**(激活函数). This activation function in the neural network terminology is just another term for that non-linear function $g(z) = \frac{1}{1+e^{-z}}$.

回归模型中的参数(θ)在神经网络中有时被称为权重(weights)。

8.3.3 Neural Network

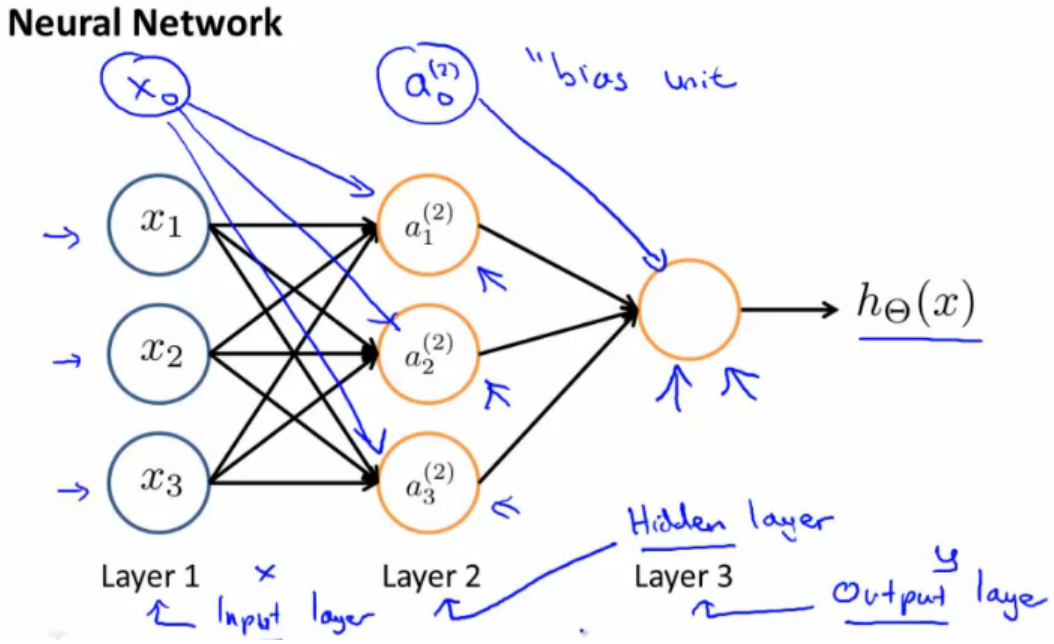


图 8.4: Neural Network

图8.4是一个三层的神经网络,其中 x_1,x_2,x_3 是输入层,The third layer outputs the value that the hypothesis $h(x)$ computes. 第一层经常称为输入层 (input layer),最后一层被称为输出层 (output layer),而位于中间的被称为隐藏层 (hidden layer)。关于这个神经网络是如何工作的,请参考图8.5。

需要注意图中的标号,正如图中所说,如果神经网络在第 j 层有 s_j 个单位,在 $j+1$ 层有 s_{j+1} 个单位,那么 $\Theta^{(j)}$ 将是一个 $s_{j+1} \times (s_j + 1)$ 的矩阵,如 h 图中 $\Theta^{(1)}$ 就是 3×4 的矩阵。

8.4 Model representation II

8.4.1 Forward propagation: Vectorized implementation 前向传播:矢量化实现

如图8.6所示,将括号中的部分表示为 $z_i^{(j)}$,那么计算过程可以使用图中右半部分的矩阵运算表示(输入层 x 使用 $a^{(1)}$ 表示)。We call this process of computing $h_\theta(x)$ **forward propagation** because that we start of with the **activations** of the input-units and then we **forward propagate** that to the hidden layer and compute the activations of the hidden layer and then we forward propagate that, and compute the activations of the output layer, this process of computing the activations from the input then the hidden then the output layer and that's also called forward propagation.

8.4.2 Neural Network learning its own features

如图8.7所示,如果将 input layer 遮住,那么计算 $h_\theta(x)$ 的公式将如图中所示,可见与 Logistic Regression 的公式(公式6.1)几乎相同(除了 θ 变成了大写 Θ 外 ~)。只不过在神经网络中,我们并不直接使用输入的 features -- x_1, x_2, x_3,而是使用 hidden layer 的 activations -- $a_1^{(2)}, a_2^{(2)}, a_3^{(2)}$。It learned its own features to apply logistic regression. 这将比使用原始的输入 features (或他们的多项式形式 $x_1^2, x_1x_2, x_2^2 \ldots$)来计算 logistic regression 产生更好的 hypotheses。

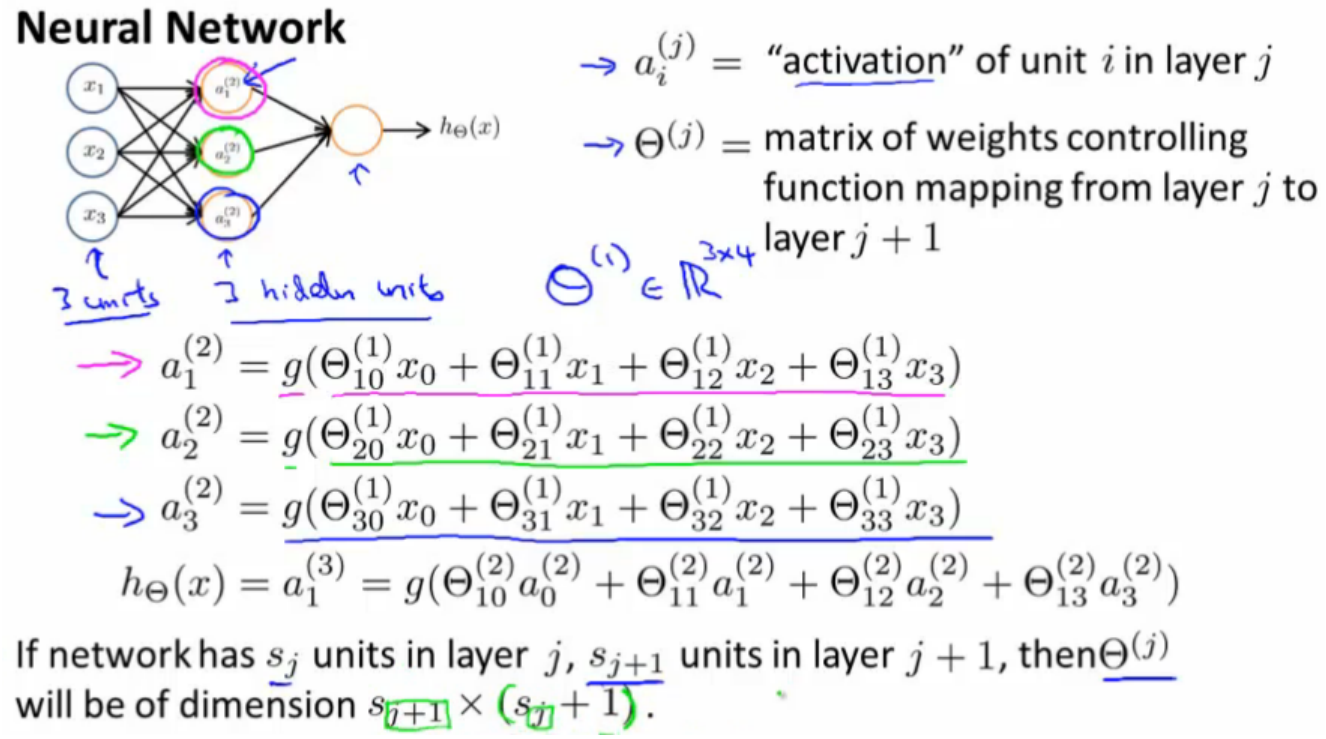


图 8.5: Neural Network(2)

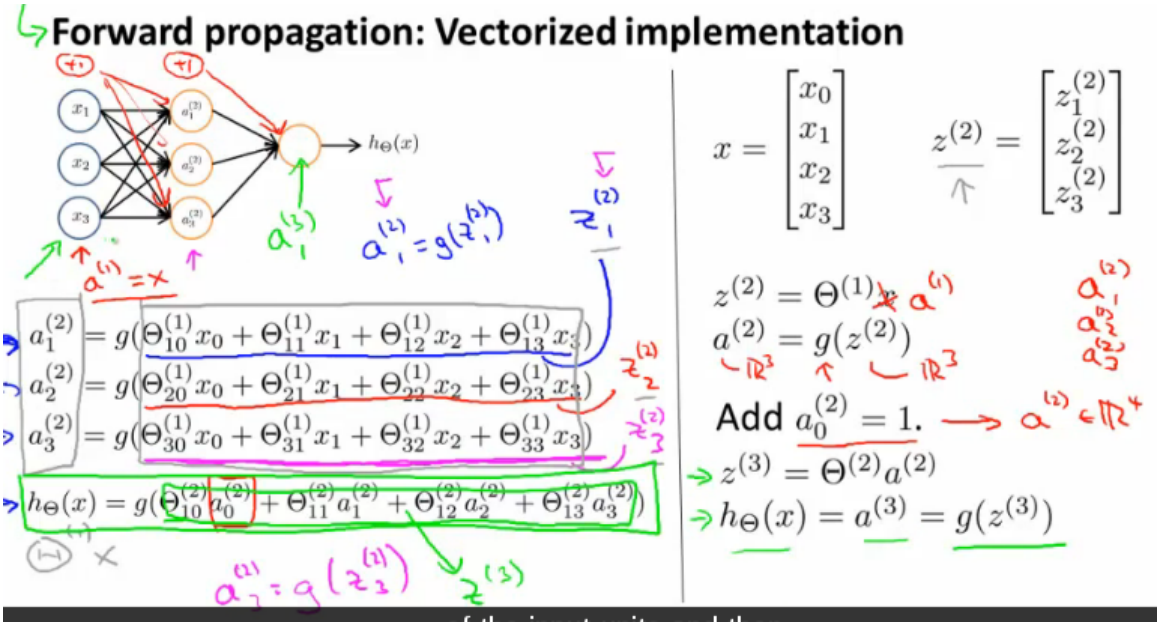


图 8.6: Forward propagation:Vectorized implementation

8.5 Examples and intuitions I

8.5.1 Simple example: AND

图8.8为神经网络实现的 AND 逻辑。

8.6 Examples and intuitions II

在本节将进行更为复杂的非线性问题的求解。
图8.9向我们展示了 XNOR 逻辑的具体实现过程。XNOR 的真值表见图右下部分，即“相同为真，相异为

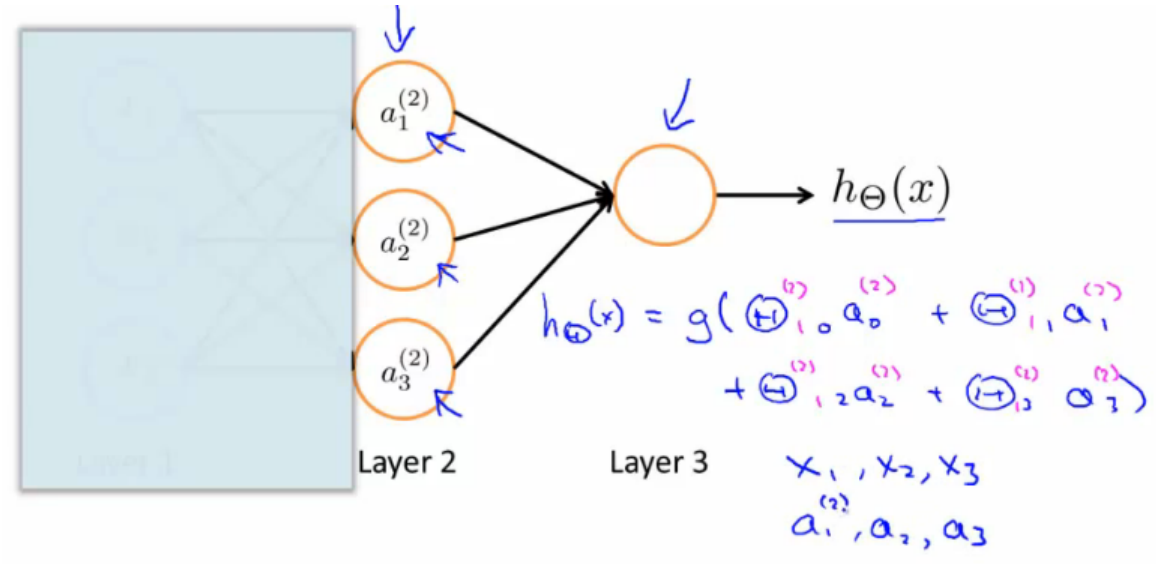


图 8.7: Neural network learning its own features

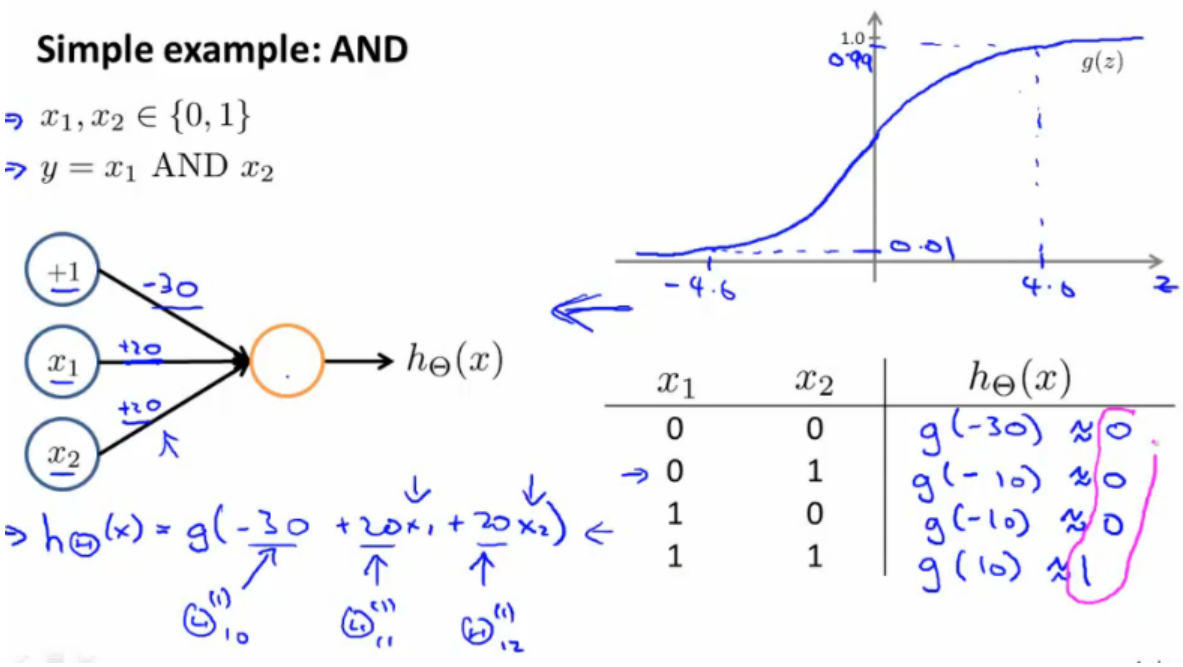


图 8.8: 神经网络中的 AND 逻辑

假”,若以 x_1 和 x_2 为坐标轴,那么我们需要确定图8.9上部所示坐标系的 decision boundary。通过组合图中上部的三个简单神经网络,可以构造出 XNOR 的神经网络,即中间一层使用 AND 和 NOT AND,输出层使用 OR,得到的结果可以使用真值表检验。

8.7 Multi-class classification

8.7.1 Multiple output units: One-vs-all

如图8.11,四个输出项分别标识了是否是步行、是否是轿车、是否时摩托车和是否是卡车。可见类似于之前的 One-vs-All 方法,这里有四个逻辑标识符 (logistic classifier)。注意图中用的是约等号。

课后习题:Suppose you have a multi-class classification problem with three classes, trained with a 3 layer network. Let $a_2^{(3)} = (h_\Theta(x))_2$ (注意 Θ 要用大写)be the activation of the first output unit, and similarly $a_2^{(3)} = (h_\Theta(x))2$ and $a_3^{(3)} = (h_\Theta(x))_3$. Then for any input x, it must be the case

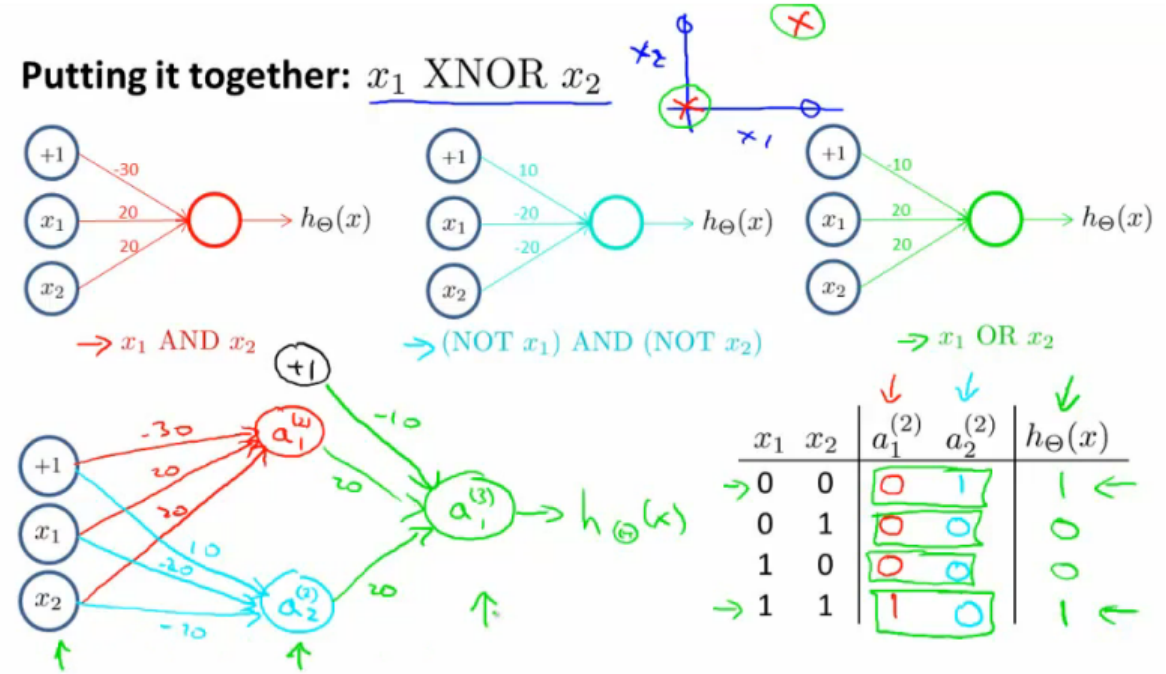


图 8.9: XNOR 逻辑的实现过程

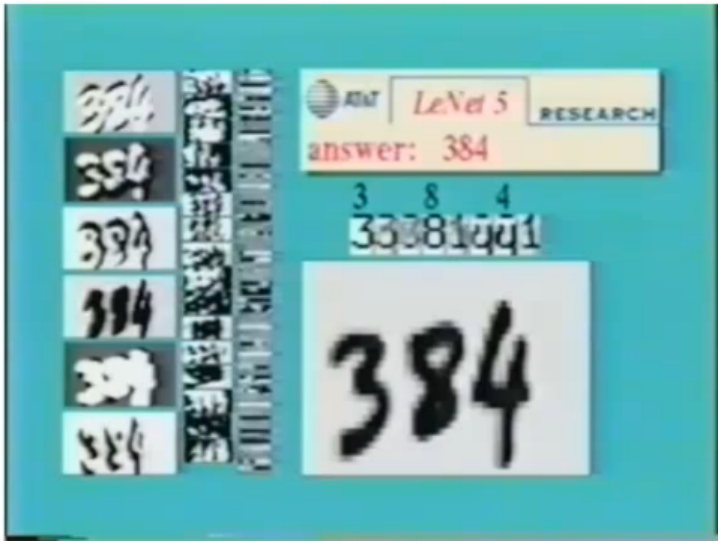


图 8.10: 神经网络的经典案例:数字识别

that $a_1^{(3)} + a_2^{(3)} + a_3^{(3)} = 1$。答案:错误,因为输出并非是概率,所以输出的总和并不一定是 1。

9 Neural Networks: Learning(the parameters)

9.1 Cost Function

9.1.1 Neural Network (Classification)

图9.1解释:
L: 网络层数。
s_l: 第 l 层的节点个数(不包括 bias 节点)。
k: output layer 的节点个数。
我们将考虑 binary classification 和 Muti-class classification($K \geqslant 3$, $K = 2$ 没意义) 两种情况。

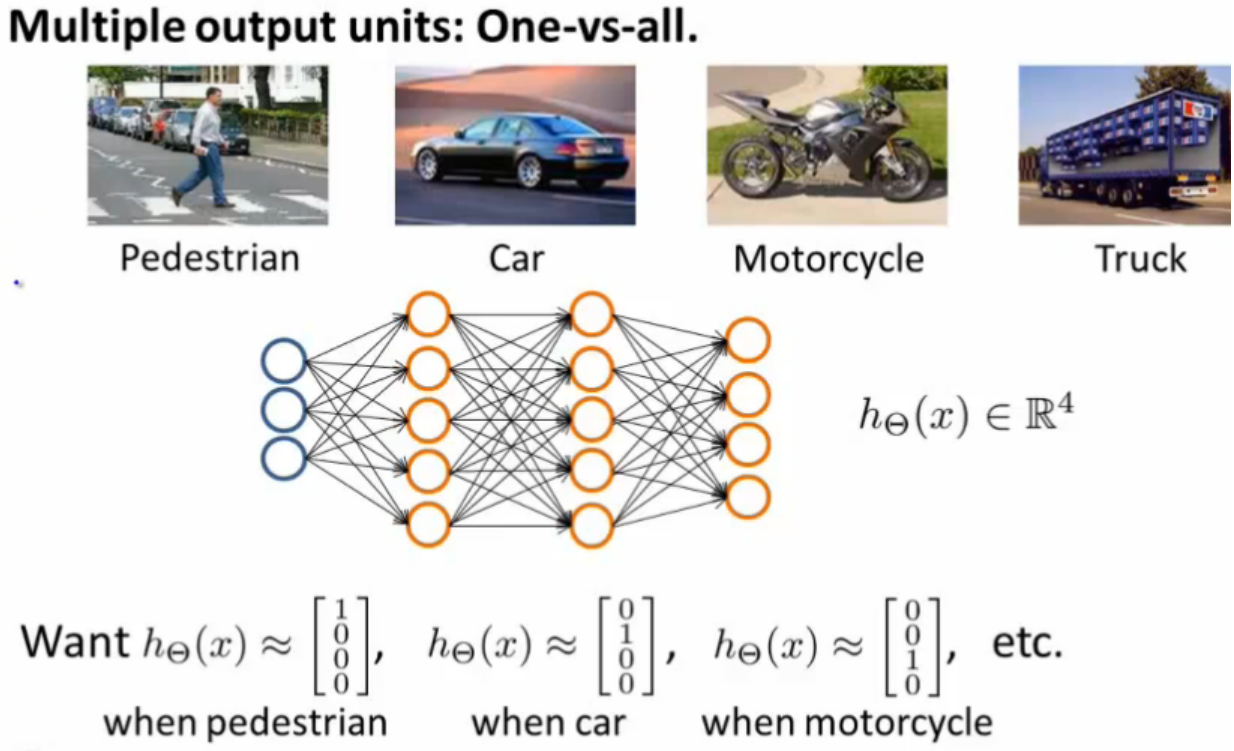


图 8.11: One vs All in Neural Network

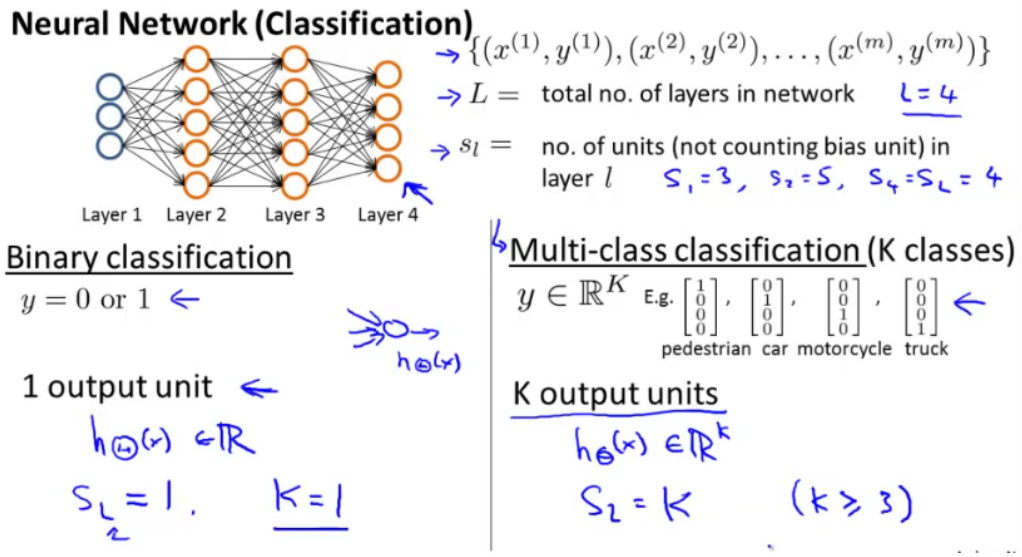


图 9.1: Neural Network (Classification)

9.1.2 Cost function

Logistic regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta\left(x^{(i)}\right) + \left(1 - y^{(i)}\right) \log\left(1 - h_\theta\left(x^{(i)}\right)\right)\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2 \tag{9.1}$$

The cost function we use for the neural network is going to be the generalization of the one that we use for logistic regression(见公式7.2和9.1).

Neural network:(有 K 个输出)

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

$$\begin{aligned} J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log\left(h_\Theta\left(x^{(i)}\right)\right)_k + \left(1-y_k^{(i)}\right)\log\left(1-\left(h_\Theta\left(x^{(i)}\right)\right)_k\right)\right] \\ +\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(\Theta_{ji}^{(l)}\right)^2 \end{aligned} \tag{9.2}$$

For logistic regression, we used to minimize the cost function j of theta that was shown above. For neural network, instead of having basically just one logistic regression output unit, we have **K of them**.

In the cost funtion, J of θ, we have a sum from k = 1~K. This is basically the sum over the K output unit. It's basically the logistic regression algorithm's cost function but **summing** that cost function over each of my output units in turn.

因此在公式9.2中,我们使用了 y_k 和 $h_\Theta(\ldots)_k$ 来表示针对每一个输出项的 y 和 hypothesis。虽然该公式的 regularization 项看起来非常复杂,但无非是在计算除了 bias unit(i=0)外所有的 $\left(\Theta_{ji}^{(l)}\right)^2$ 的总和而已,和 logistic regression 一样,从 1 开始(去除 bisa unit)而不是从 0 开始仅仅是个惯例,即使从 0 开始也不会有太大的不同。

9.2 Backpropagation Algorithm (to compute derivation of cost function)

9.2.1 Gradient Comutation

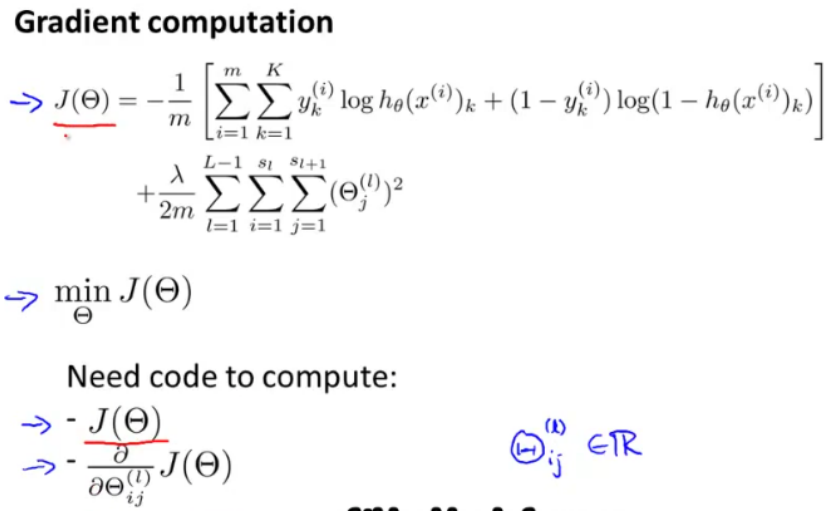


图 9.2: Gradient computation

如图9.2,What we want to do is to focus on how we can compute the **partial derivative** terms.

让我们从只有一个 training example (x, y) 的情况开始,如图9.3所示,逐步计算 cost function 的偏导。第一步为 forward propagation,用来计算 hypothesis 输出。[1]

貌似图中的 $a^{(1)}$ 应该加上 bias unit($a_0^{(1)}$)。

要计算偏导,下一步需要用到后向传播(backpropagation)算法。对该算法的直观理解就是对每个节点计算一个 $\delta_j^{(l)}$,表示节点 j 在第 l 层的“误差”(error)。比如要计算输出层节点的“误差”,如图9.4,那么只需计算每个节点 $a_j^{(4)}$ 和结果 y_i 的差距即可。

[1]在设计神经网络时,我们自然是没有现成的矩阵的(我们的任务也就是计算 Θ 矩阵),然而却需要先用前向传播算法计算 Hypothesis(这势必要用到 Θ)。虽然课程未做说明,但这里的 Θ 往往是随机生成的值,具体可以参考第9.5节和9.6节。这里只是提醒一下,以免看到相关内容时会困惑。

Given one training example (x, y):

Forward propagation:

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

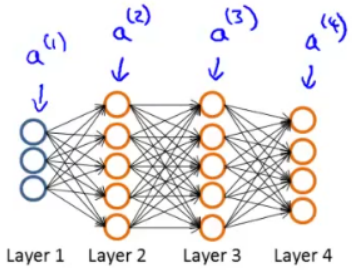


图 9.3: Forward propagation 过程

Intuition: $\delta_j^{(l)}$ = "error" of node j in layer l.

For each output unit (layer L = 4)

$$\delta_j^{(4)} = a_j^{(4)} - y_j$$

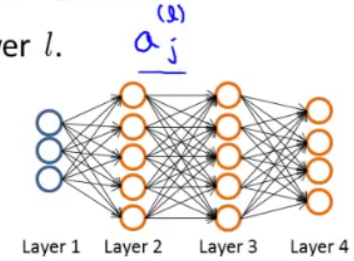


图 9.4: backpropagation intuition

注意图9.4中的算式可以矢量化为 $\delta^{(4)} = a^{(4)} - y = (h_\Theta(x)) - y$

下一步需要计算前面每一层的 δ,见式9.3、9.4。注意式9.3中 $\left(\Theta^{(3)}\right)^T \delta^{(4)}$ 是一个矢量, $g'\left(z^{(3)}\right)$ 也是一个矢量,他们之间使用的是点乘号(element wise),表示按元素进行乘法运算(两矢量的对应元素分别相乘)。[1]

$$\delta^{(3)} = \left(\Theta^{(3)}\right)^T \delta^{(4)} .* g'\left(z^{(3)}\right) \tag{9.3}$$

$$\delta^{(2)} = \left(\Theta^{(2)}\right)^T \delta^{(3)} .* g'\left(z^{(2)}\right) \tag{9.4}$$

这里, $g'\left(z^{(3)}\right)$ 是 activation function $g\left(z^{(3)}\right)$ 的导数:

$$g'\left(z^{(3)}\right) = a^{(3)} .* \left(1 - a^{(3)}\right) \tag{9.5}$$

这里不对该式的得出进行推导(我没看懂是怎么得出来的)。

注意没有 δ_1 这一说。

The name backpropagation comes from the fact that we start by computing the delta term for the output layer and then we **go back** a layer and compute the delta terms for that layer. So we are back propagating the errors from the current layer to the earlier layer.

最后得到的求偏导的公式非常复杂,可以证明(过程比这个公式还要复杂,故略掉),如果忽略 regularization($\lambda = 0$), 那么我们需要的偏导数是:

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = \delta_i^{(l+1)} a_j^{(l)} \tag{9.6}$$

[1] $z^{(3)}$ 是 5*1 的矢量,然而如果按之前 Θ 的定义,那么 $\left(\Theta^{(3)}\right)^T \delta^{(4)}$ 的结果应该是 6*1 的。因此这里的 Θ 应该不包括 bias unit,即 Θ 是 4*5 的,这样结果就是 5*1 的了。编程训练中也是这样处理的。

9.2.2 Backpropagation algorithm

如果 training example 不止一个,在这里设定有 m 个,如图9.5所示,首先设置 $\Delta_{ij}^{(l)} = 0$(for all l, i, j),这里的 Δ 是用来计算 cost function 的偏导的,而且正如即将看到的, Δ 其实是个累加数 (Accumulator)。针对每个 training example 执行图中所示循环,每次循环都更新一次 Δ。

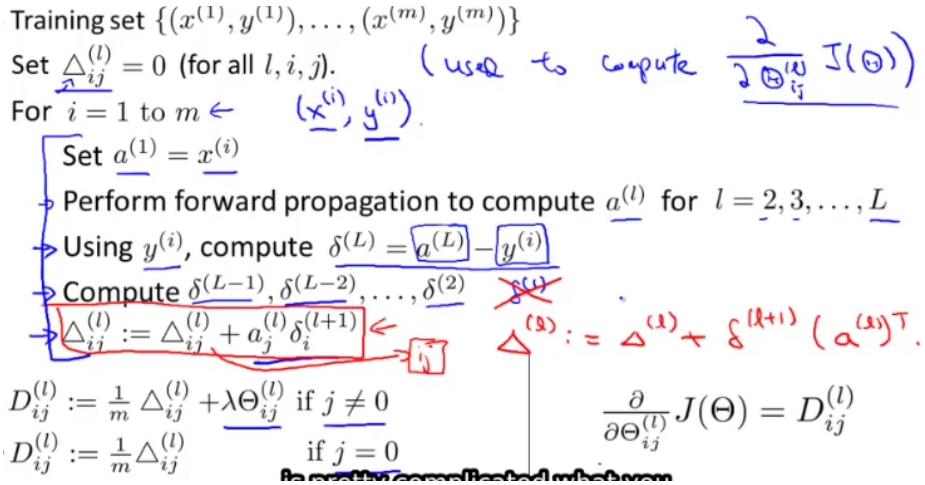


图 9.5: Backpropagation algorithm

[1]

也如图中所示, Δ 的计算可以矢量化为 $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} \left(a^{(l)}\right)^T$ 的形式。

在循环外计算 $D_{ij}^{(l)}$,分为 $j = 0$(bias unit)和 $j \neq 0$ 两种情况。

可以证明,最后求得的 $D_{ij}^{(l)}$ 就是 cost function 的偏导:

$$\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\Theta) = D_{ij}^{(l)} \tag{9.7}$$

课堂习题(图9.6)的答案就是求 gradient 过程的简单概括。

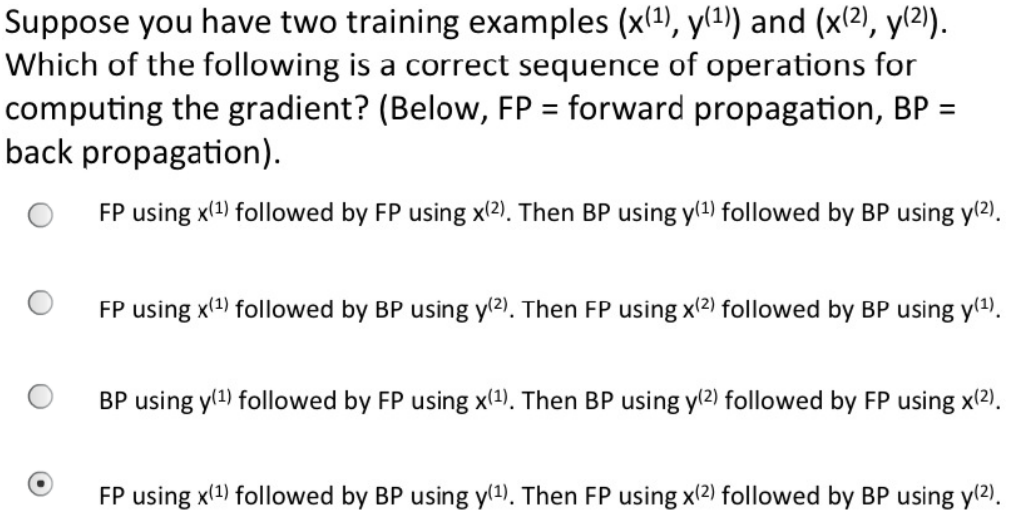


图 9.6: 课堂练习:计算 gradient 步骤

9.3 Implementation note: Unrolling parameters

我们需要用到的函数一般都将 θ 视为矢量,而神经网络中的 Θ 是矩阵,因此需要将矩阵转化为矢量后才能使用之前介绍过的函数(如 fminunc)进行计算(图9.7)。

[1] 图9.5中计算 $D_{ij}^{(l)}$ 的公式错误。应该为 $D_{ij}^{(l)} := \frac{1}{m}\left(\Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}\right) \quad if\ j \neq 0$

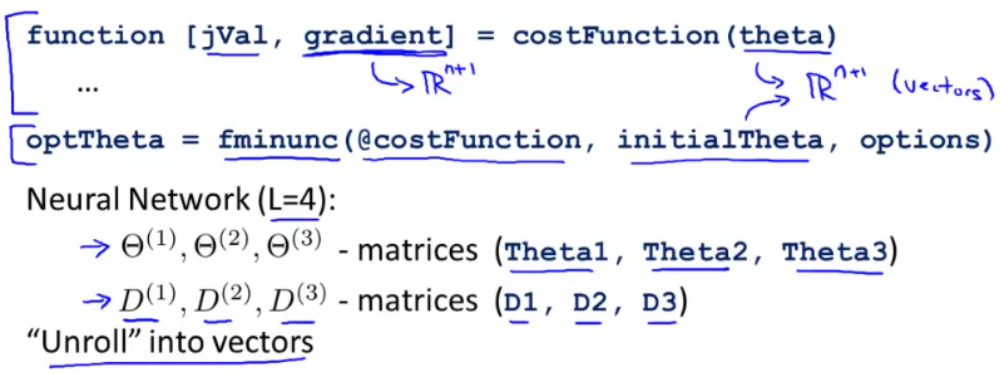


图 9.7: Need to unrol parameters

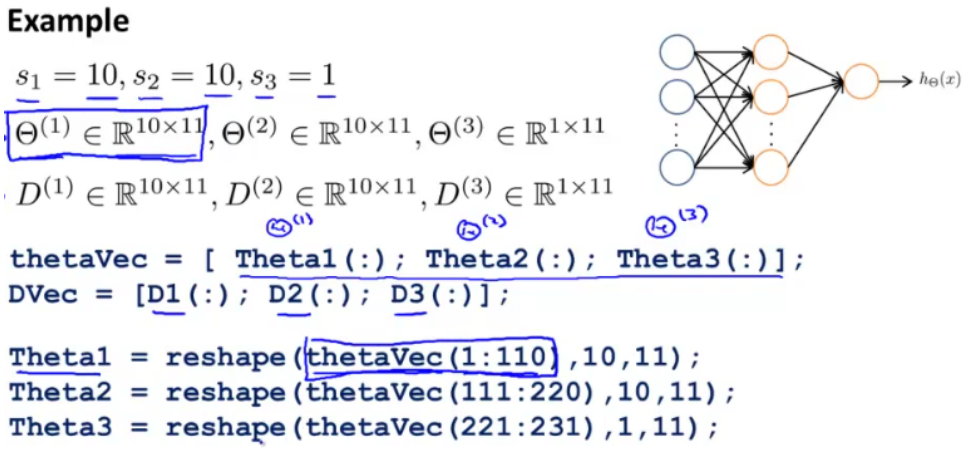


图 9.8: 矩阵和矢量的互转

图9.8所示便是矩阵到矢量和矢量到矩阵的转换方法。

9.4 Gradient Checking

神经网络的后向算法可能会出现很多微妙的错误,因此最好在投入运作之前检查一下算法是否工作正常。

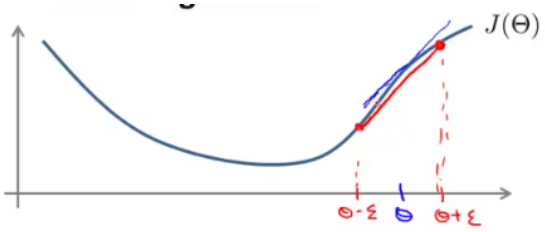


图 9.9: Gradient 错误检查

图9.9是假设 θ 是一个实数($\theta \in \mathbb{R}$)的情况,我们通过检查斜率和我们估计的斜率值的误差是否在容许范围之内来判断算法是否工作正常。当 θ 是矢量($\theta \in \mathbb{R}$)时,我们计算的近似斜率如图9.10:

在 Octave 中实现如图9.11:

实现提示:检查无误后,记得关掉 Gradient checking(time costing)。相比后向传播,Gradient 计算是非常耗时的(图9.12)。

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

$\theta = [\theta_1, \theta_2, \theta_3, \ldots, \theta_n]$

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

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

$$\vdots$$

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

图 9.10: 近似斜率的计算

```
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;
```

Check that gradApprox ≈ DVec

图 9.11: Octave 实现的近似斜率

What is the main reason that we use the backpropagation algorithm rather than the numerical gradient computation method during learning?

- ○ The numerical gradient computation method is much harder to implement.
- ◉ The numerical gradient algorithm is very slow.
- ○ Backpropagation does not require setting the parameter EPSILON.
- ○ None of the above.

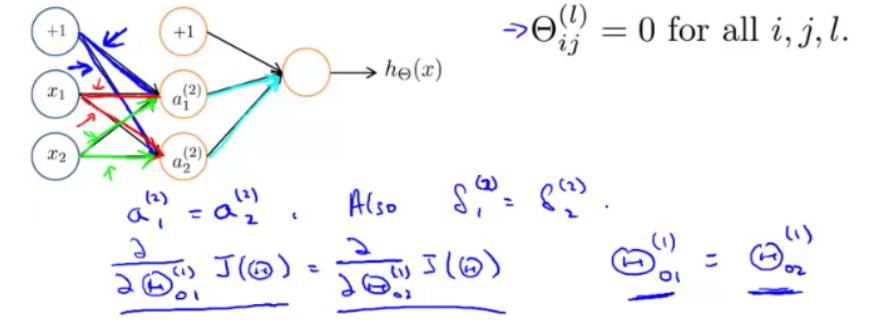
图 9.12: 使用后向传播而不是 Gradient descent 的原因

9.5 Random Initialization

9.5.1 Initial value of Θ

For gradient descent and advanced optimization method, need initial value for Θ.

把 Θ 初始化为零向量(在 logistic regression 中就是这么做的)在神经网络中是行不通的,见图9.13,在每次更新(gradient descent, etc)后,每个参数的权重永远是相同的。有时这被称作 The problem of symmetric weights.



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

图 9.13: 将 Θ 初始化为 0 将导致每一层元素全部相同

9.5.2 Random initialization: Symmetry(对称)breaking

将每一个 $\Theta_{ij}^{(l)}$ 初始化为 $[-\epsilon, \epsilon]$ 之间的随机值:

```
Theta1 = rand(10,11)*(2*INIT_EPSILON) - INIT_EPSILON;
```

9.6 Putting It Together

9.6.1 Training a neural network

1. The first thing you need to do is to pick some network architecture:
   - Number of input units: Dimension of features $x^{(i)}$.
   - Number of output units: Number of classes.
   - Reasonable default: 1 hidden layer, or if >1 hidden layer, have same number of hidden units in every layer.
   - Usually the number of hidden units in each layer will be comparable to the dimension of x. Same number or 3 or 4 times of that(usually the more the better).
2. Randomly initialize weights.
3. Implement forward propagation to get $h_\Theta\left(x^{(i)}\right)$ for any $x^{(i)}$.
4. Implement code to compute cost function $J(\Theta)$.
5. Implement backprop to compute partial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$.

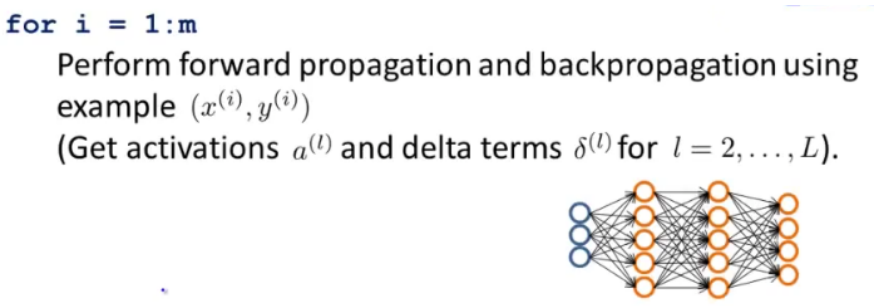


图 9.14: backpropagation contour

图9.14为后向传播算法概述,完整算法见 9.5。

6. 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.
7. Use gradient descent or advanced optimization method with backpropatation(to compute $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$) to try to minimize $J(\Theta)$ as a function of parameters Θ.

注意:在神经网络中, $J(\Theta)$ 是 non-convex 的。所以 s 理论上可能会得到局部最优解。不过在实践中这并不是一个大问题,通常得到的解已足够令人满意。

Gradient descent 的直观描述见图9.15。

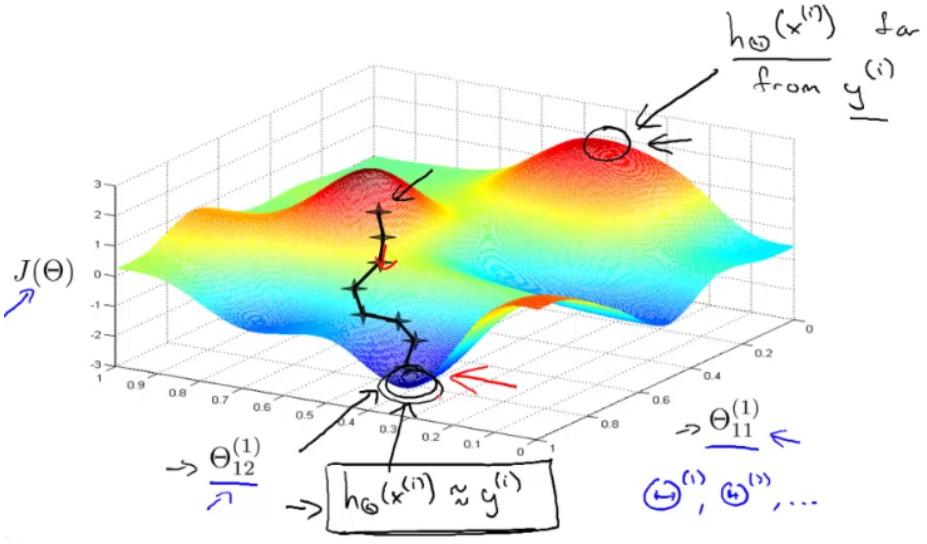


图 9.15: 回顾 Gradient descent

What gradient descent does is starting from some random initial point, and it will repeatedly go downhill. And so what backpropagation is doing is computing the direction of the gradient, and what gradient descent is doing is taking little steps downhill untill hopefully it gets to a pretty good local optimum.

When you implement backpropagation and use gradient descent or one of the advanced optimization methods, this picture sort of explaining what the algorithm is doing.

10 Advice for applying machine learning

10.1 Deciding what to try next

如图10.1,在之后的课程里,将会介绍 Machine learning diagnostic.

It's a test that you can run to gain insight what is/isn't working with a learning algorithm, and gain guidance as to how best to improve its performance.

10.2 Evaluating a hypothesis

我们可以把 hypothesis 画出来,判断是否出现 overfitting 或 underfitting. 可如果 features 数量很多的话,画图就不是什么好主意了。

一般化的方法是将 dataset 分割为两部分:第一部分用作 training set, 而第二部分用来做 test set。分割的比例大概是 7:3。在分割前先对 dataset 随机排序一下(如果其排列不是随机的话)。

10.2.1 Training/testing procedure for linear regression

见图10.2。

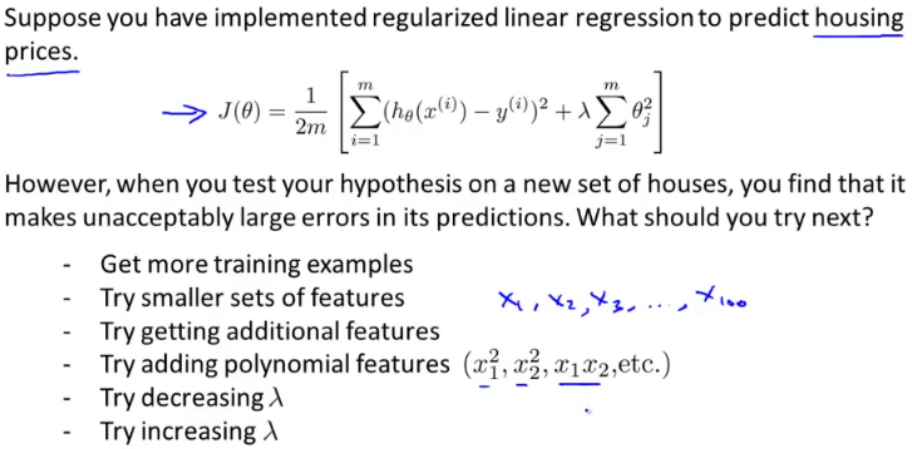


图 10.1: What to try next

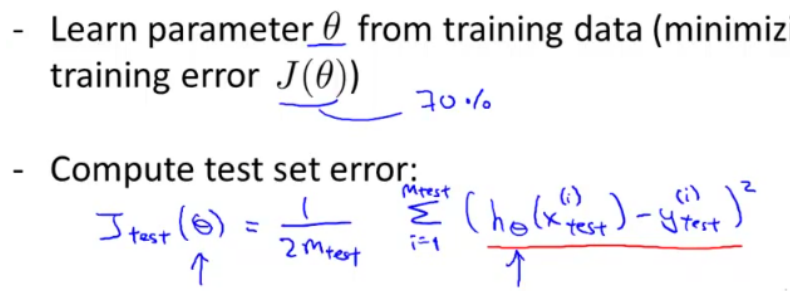


图 10.2: Training/testing procedure for linear regression

10.2.2 Training/testing procedure for logistic regression(classification)

图10.3给出了两种检验学习质量的方法,一种是使用 cost function,另一种称为 misclassification error。不论使用哪种方法,如果算出的误差偏大,那么说明机器学习的质量偏低。

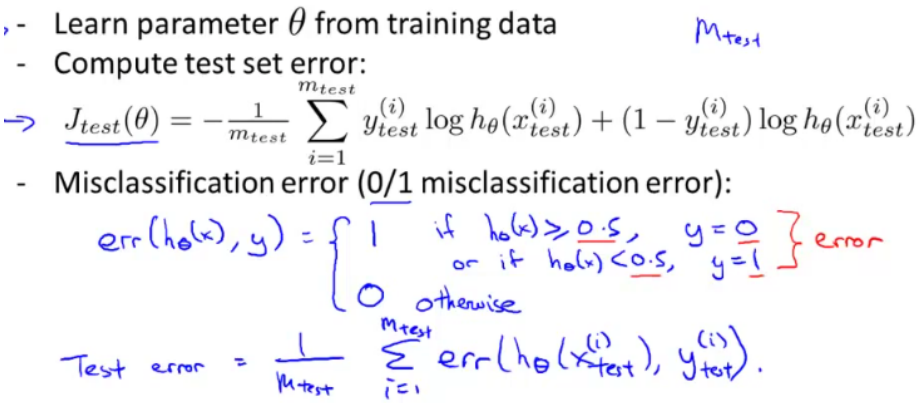


图 10.3: Training/testing procedure for classification

10.3 Model selection and training/validation/test sets

Suppose you'd like to decide what the degree of polynomial to fit to a data set, sort of what features to include to give you a learning algorithm. Or suppose you'd like to choose the regularization parameter λ for the learning algorithm. These are called **model selection problems**.

在 model selection 问题中,我们将把 data set 分成 trainig、validation 和 test 三部分,而不只是 training 和 test 两部分。

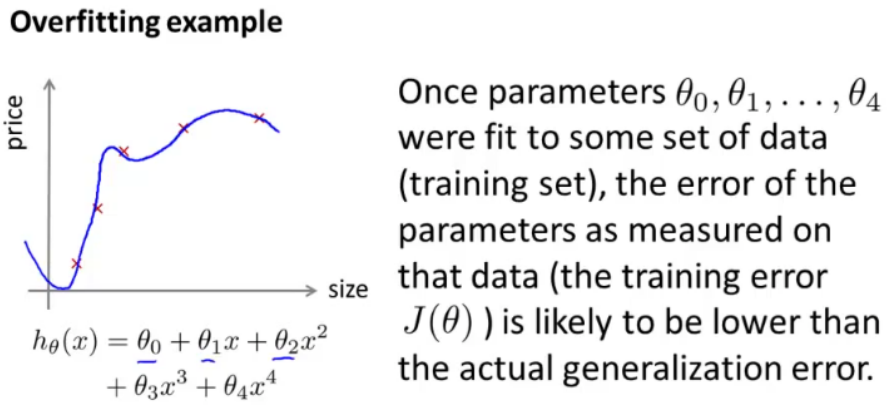


图 10.4: Overfitting example

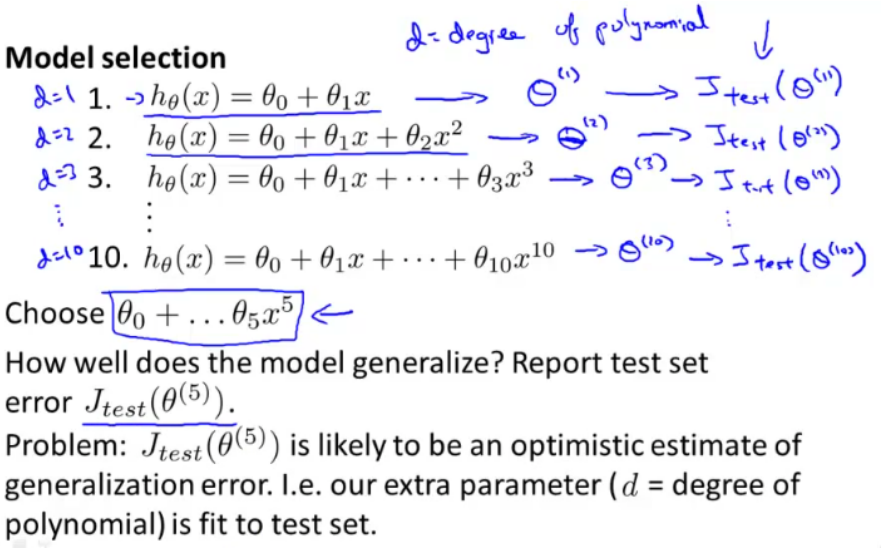


图 10.5: Model slection

如图10.5所示,假定我们列出 d=1 到 d=10(d 表示多项式的维度)10 个模型,然后计算每个模型的测试误差 (test error)$J_{test}(\theta)$,最后选择测试误差最小的模型。假设我们最终选择的模型是 d=5 的多项式,可以看到,使用 $J_{test}\left(\theta^{(5)}\right)$ 来估计模型的归纳质量是不公平的。因为 What we done is we fit the extra parameter d to the test set(即找出 d 的最小值),那么我们据此选择的模型得到的 Θ 可能过于乐观的估计了它的归纳效果。也就是说,建立在 test set 上的估计是不全面的。

现在我们将数据分成三部分:Training set, Cross validation set 和 Test set,大概是 6:2:2 的比例,见图10.6。

10.3.1 Evaluating your hypothesis

10.3.2 Train/validation/test error

Training error:

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

Cross Validation error:

$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}\left(h_\theta\left(x_{cv}^{(i)}\right) - y_{cv}^{(i)}\right)^2$$

Test error:

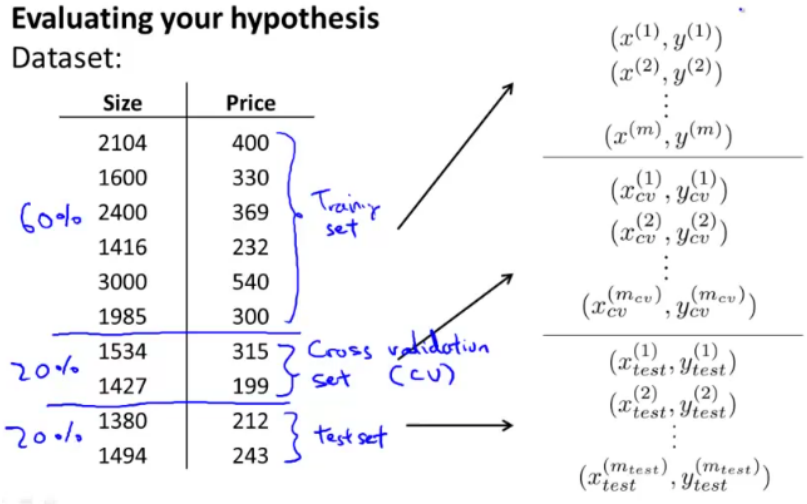
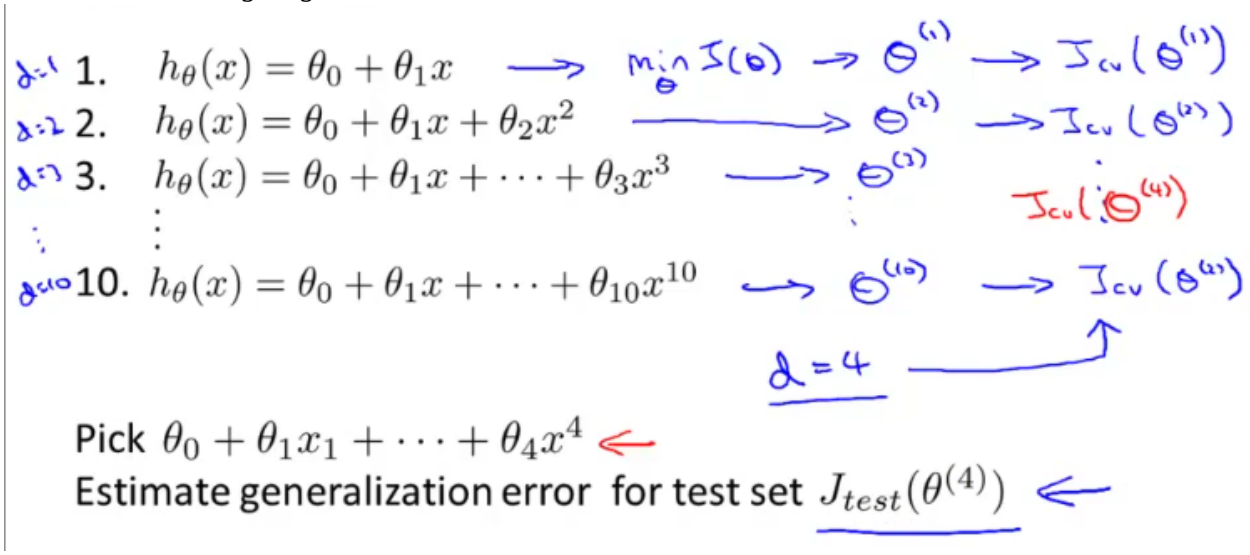


图 10.6: Evaluating hypothesis

$$J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} \left(h_\theta \left(x_{test}^{(i)} \right) - y_{test}^{(i)} \right)^2$$

So when we fits the model selection problem by this, instead of using the test set to select the model, we're instead going to use validation set or the cross-validation set to select the model.



Consider the model selection procedure where we choose the degree of polynomial using a cross validation set. For the final model (with paramet θ), we might generally expect $J_{CV}(\theta)$ To be lower than $J_{test}(\theta)$ because:

- (•) An extra parameter (d, the degree of the polynomial) has been fit to the cross validation set.
- ( ) An extra parameter (d, the degree of the polynomial) has been fit to the test set.
- ( ) The cross validation set is usually smaller than the test set.
- ( ) The cross validation set is usually larger than the test set.

Correct

Continue

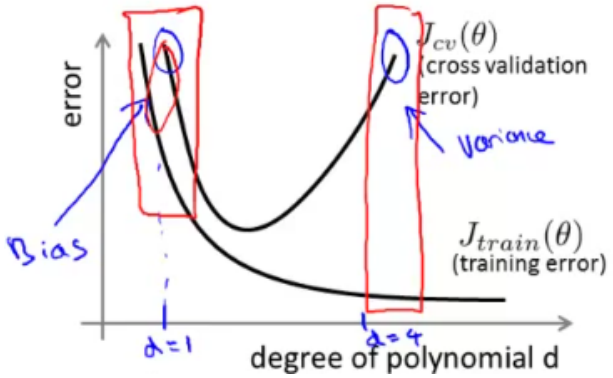
承认我没看懂……

Try evaluating the hypothesis on a cross validation set rather than the test set	0.00	A cross validation set is useful for choosing the optimal non-model parameters like the regularization parameter λ, but the train / test split is sufficient for debugging problems with the algorithm itself.

10.4 Diagnosing bias vs. variance

If you run the learning algorithm and it doesn't do as well as you're hoping, almost all the time it will be because you have either a high bias problem or a high variance problem. In other words they're either an underfitting problem or an overfitting problem.

Suppose your learning algorithm is performing less well than you were hoping. ($J_{cv}(\theta)$ or $J_{test}(\theta)$ is high.) Is it a bias problem or a variance problem?



Bias (underfit):

$J_{train}(\theta)$ will be high

$J_{cv}(\theta) \approx J_{train}(\theta)$

Variance (overfit):

$J_{train}(\theta)$ will be low

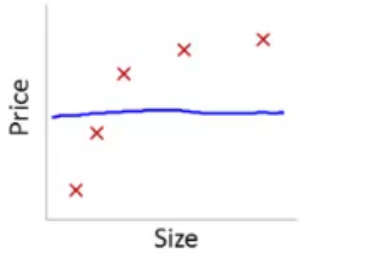
$J_{cv}(\theta) >> J_{train}(\theta)$

10.5 Regularization and bias/variance

Linear regression with regularization

Model: $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

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

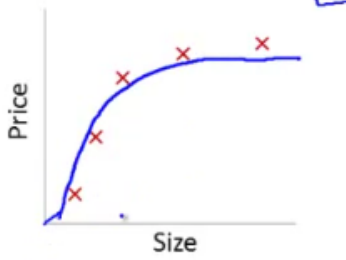


Large λ

High bias (underfit)

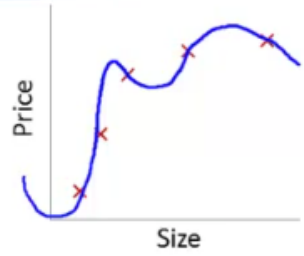
$\lambda = 10000.\ \theta_1 \approx 0, \theta_2 \approx 0, \ldots$

$h_\theta(x) \approx \theta_0$



Intermediate λ

"Just right"



Small λ

High variance (overfit)

$\lambda = 0$

And how can we automatically choose a good value for the regularization parameter λ ?

10.5.1 Choosing the regularization parameter λ

$$h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$$

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

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

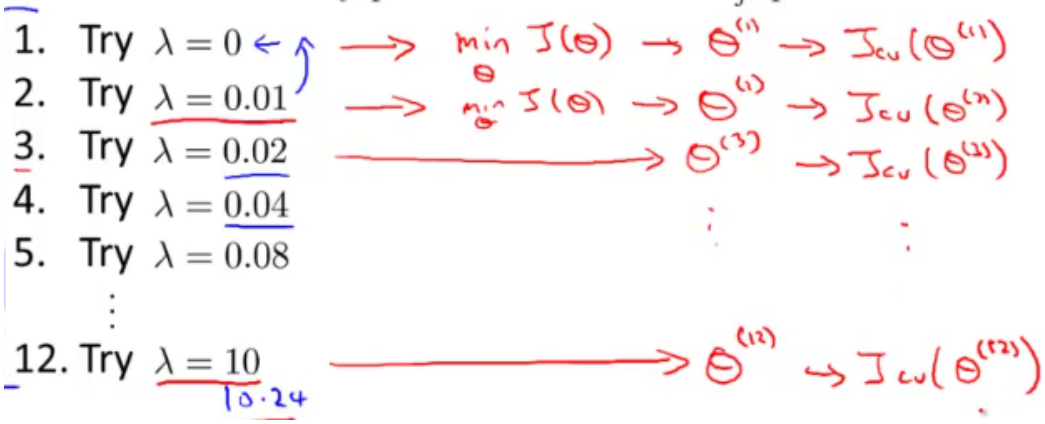
$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$

$$J_{test}(\theta) = \frac{1}{2m_{test}}\sum_{i=1}^{m_{test}}(h_\theta(x_{test}^{(i)}) - y_{test}^{(i)})^2$$

$J_{train}(\theta)$、$J_{cv}(\theta)$ 和 $J_{test}(\theta)$ 都没有 regularization。

Model: $h_\theta(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

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



选择一个使 Cross validation error 最小的 λ 即可。选定之后，计算该 θ 的 test error: $J_{test}(\Theta^{(?)})$ 来检查其在 test set 上的表现如何。

10.5.2 Bias/variance as a function of the regularization parameter λ

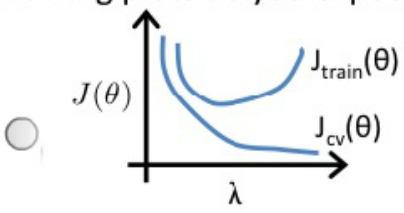
Consider regularized logistic regression. Let

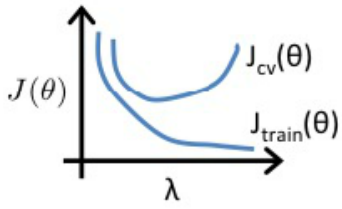
$$J(\theta) = \frac{1}{2m}\left[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

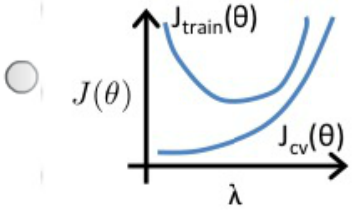
$$J_{\text{train}}(\theta) = \frac{1}{2m_{\text{train}}}\left[\sum_{i=1}^{m_{\text{train}}}(h_\theta(x_{\text{train}}^{(i)}) - y_{\text{train}}^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

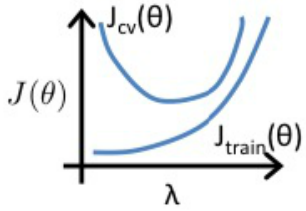
$$J_{\text{cv}}(\theta) = \frac{1}{2m_{\text{cv}}}\left[\sum_{i=1}^{m_{\text{cv}}}(h_\theta(x_{\text{cv}}^{(i)}) - y_{\text{cv}}^{(i)})^2 + \lambda\sum_{j=1}^{n}\theta_j^2\right]$$

Suppose you plot J_{train} and J_{cv} as a function of the regularization parameter λ. Which of the following plots do you expect to get?









Correct

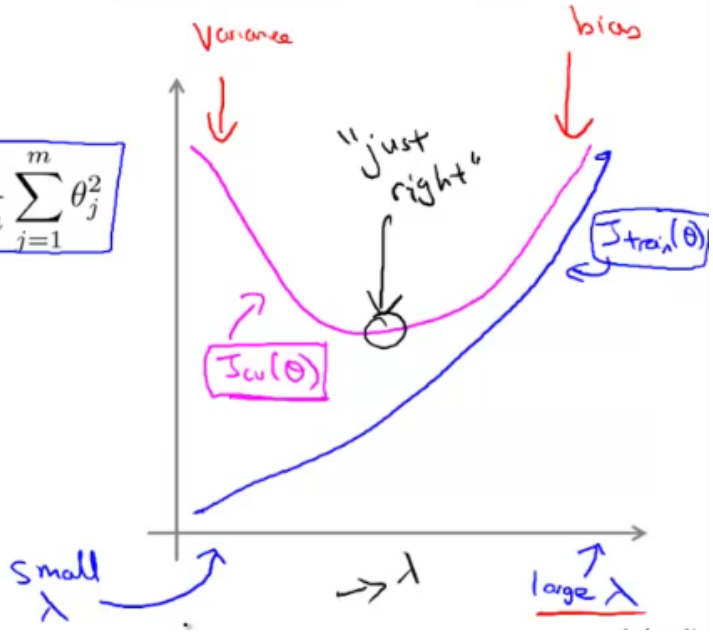
Continue

Bias/variance as a function of the regularization parameter λ

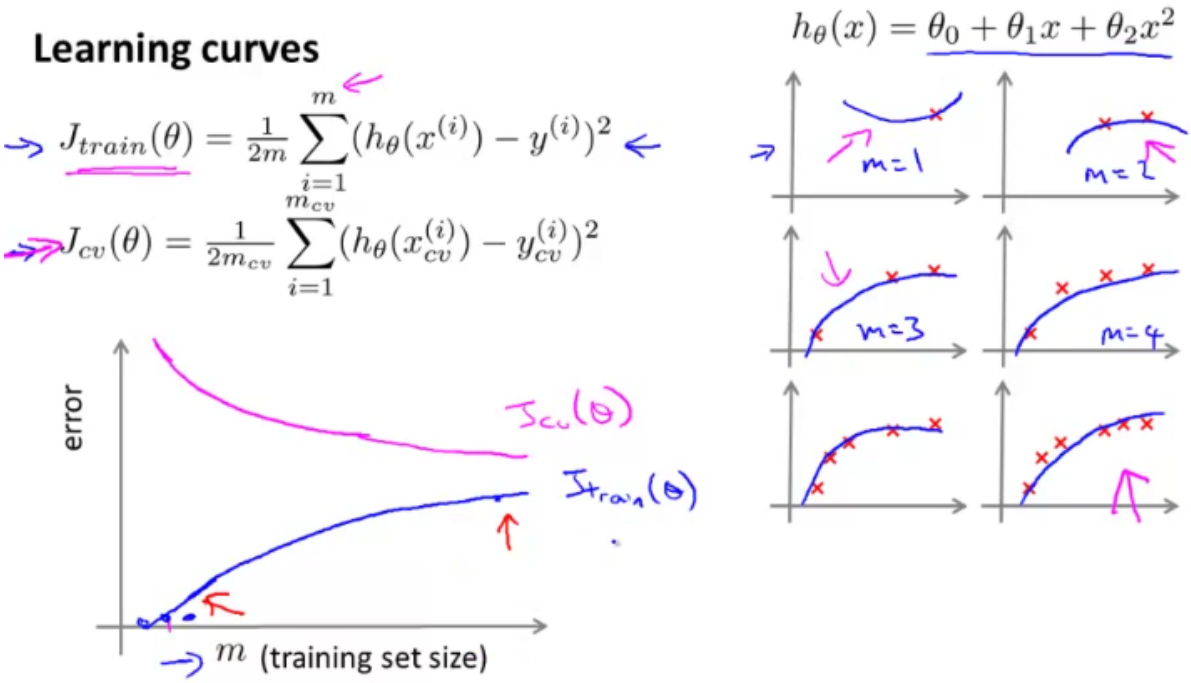
$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m}\sum_{j=1}^{m}\theta_j^2$$

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

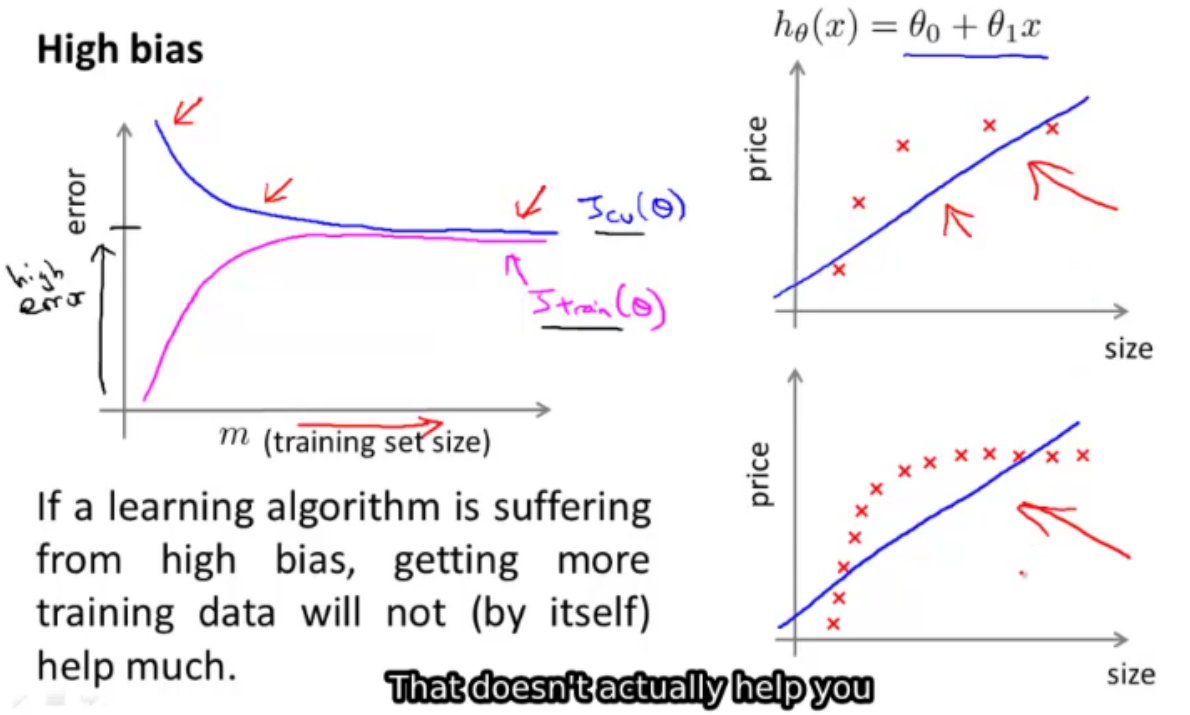
$$J_{cv}(\theta) = \frac{1}{2m_{cv}}\sum_{i=1}^{m_{cv}}(h_\theta(x_{cv}^{(i)}) - y_{cv}^{(i)})^2$$



10.6 Learning curves

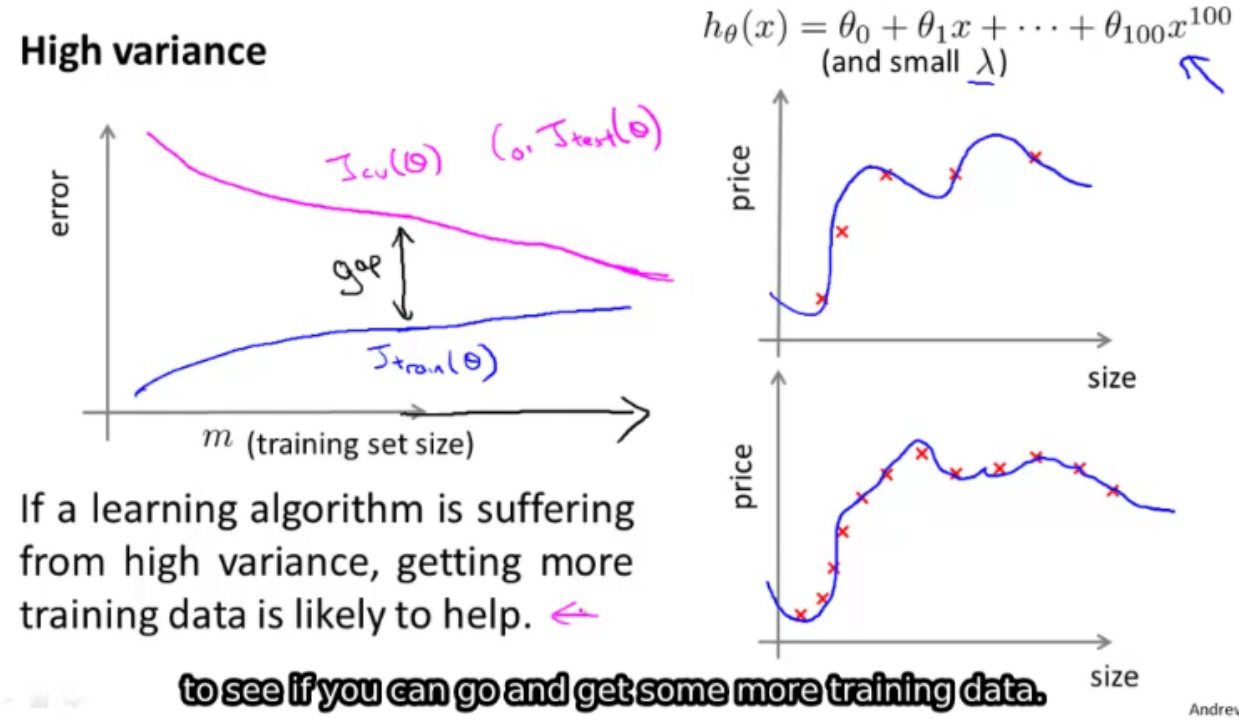


10.6.1 High bias



Both $J_{cv}(\Theta)$ and $J_{train}(\Theta)$ are high.

10.6.2 High variance



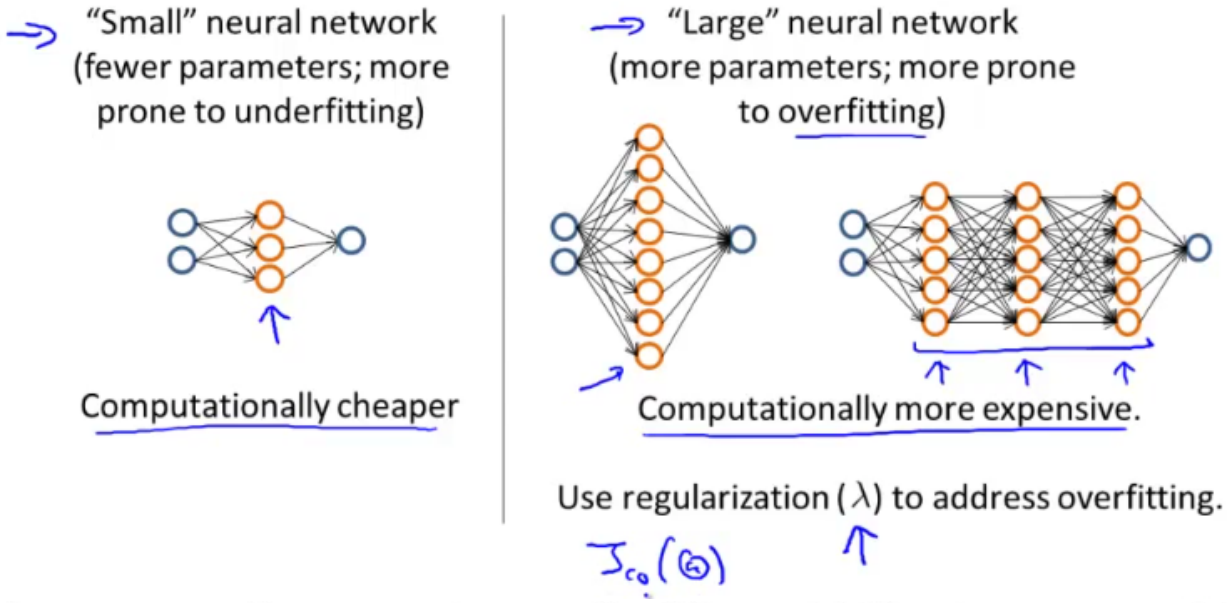
10.7 Deciding what to try next (revisited)

Debugging a learning algorithm:
Suppose you have implemented regularized linear regression to predict housing prices. However, when you test your hypothesis in a new set of houses, you find that it makes unacceptably large errors in its prediction. What should you try next?

- Get more training examples → fixes high variance
- Try smaller sets of features → fixes high variance
- Try getting additional features → fixes high bias
- Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, \text{etc})$ → fixes high bias.
- Try decreasing λ → fixes high bias
- Try increasing λ → fixes high variance

10.7.1 Neural networks and overfitting

如果要增加神经网络的层数，可以测试一下不同层数的 $J_{cv}(\Theta)$ 以找到最佳结果。



Suppose you fit a neural network with one hidden layer to a training set. You find that the cross validation error $J_{CV}(\theta)$ is much larger than the training error $J_{train}(\theta)$. Is increasing the number of hidden units likely to help?

- ○ Yes, because this increases the number of parameters and lets the network represent more complex functions.
- ○ Yes, because it is currently suffering from high bias.
- ○ No, because it is currently suffering from high bias, so adding hidden units is unlikely to help.
- ◉ No, because it is currently suffering from high variance, so adding hidden units is unlikely to help.

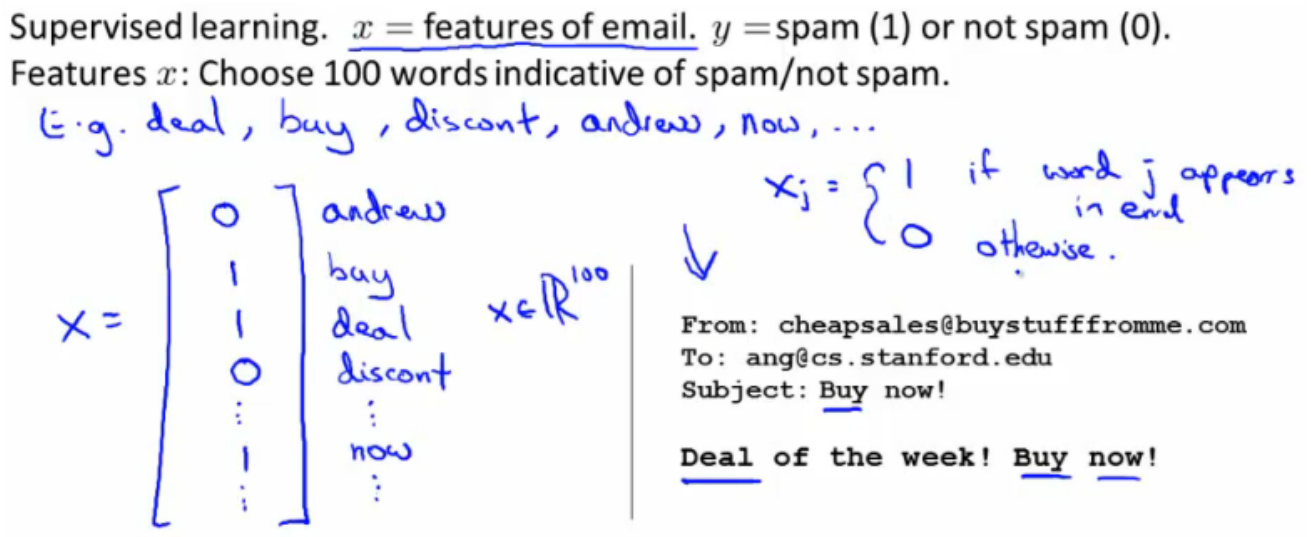
Correct

Continue

11 Machine Learning System Design

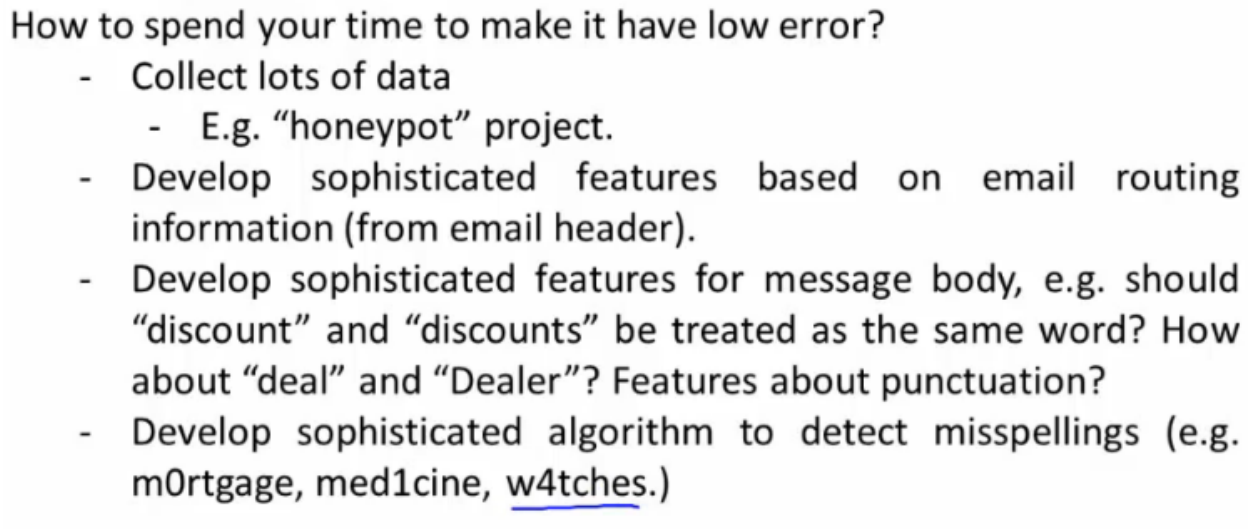
11.1 Prioritizing what to work on: Spam classification example

采用 Supervised leaning 来构建一个垃圾邮件分类器,问题就在于如何选择 email 的 features。而输出 y 则是 spam(1) 或 not spam(0) 两个值。



Note: In practice, take most frequently occurring n words (10,000 to 50,000) in training set, rather than manually pick 100 words.

How to spend your time to make it have low error?



- [x] For some learning applications, it is possible to imagine coming up with many different features (e.g. email body features, email routing features, etc.). But it can be hard to guess in advance which features will be the most helpful.
- [ ] For spam classification, algorithms to detect and correct deliberate misspellings will make a significant improvement in accuracy.
- [ ] Because spam classification uses very high dimensional feature vectors (e.g. n = 50,000 if the features capture the presence or absence of 50,000 different words), a significant effort to collect a massive training set will always be a good idea.
- [x] There are often many possible ideas for how to develop a high accuracy learning system; "gut feeling" is not a recommended way to choose among the alternatives.

11.2 Error analysis

11.2.1 Recommended approach

- Start with a simple algorithm that you can implement quickly. Implement it and test it on your cross-validation data.
- Plot learning curves to decide if more data, more features, etc. are likely to help.
- Error analysis: Manually examine the examples (in cross validation set) that your algorithm made errors on. See if you spot any systematic trend in what type of examples it is making errors on.

先用 24 小时的时间构建一个相当简单而且脏乱的系统。然后使用 cross-validation data 来测试。之后通过画学习曲线和误差分析来优化系统。

11.2.2 Error Analysis

m_{CV} = 500 examples in cross validation set
Algorithm misclassifies 100 emails.
Manually examine the 100 errors, and categorize them based on:
→ (i) What type of email it is — pharma, replica, steal passwords, ...
→ (ii) What cues (features) you think would have helped the algorithm classify them correctly.

Pharma: 12
Replica/fake: 4
→ Steal passwords: 53
Other: 31

→ Deliberate misspellings: 5
(m0rgage, med1cine, etc.)
→ Unusual email routing: 16
→ Unusual (spamming) punctuation: 32

11.2.3 The importance of numerical evaluation

Why is the recommended approach to perform error analysis using the cross validation data used to compute $J_{cv}(\theta)$ rather than the test data used to compute $J_{test}(\theta)$?

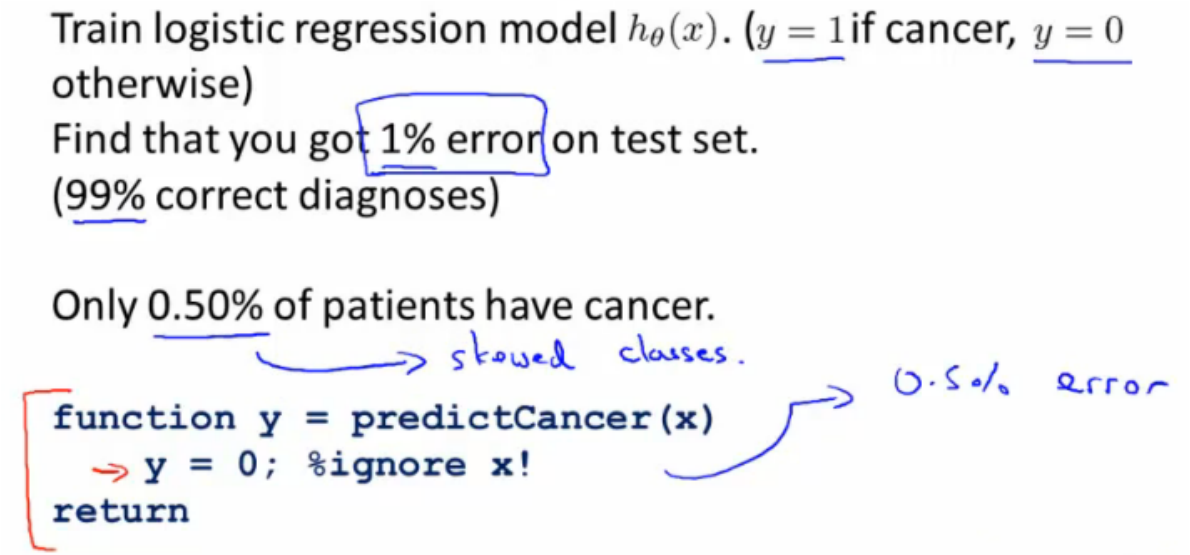
- ○ The cross validation data set is usually large.
- ○ This process will give a lower error on the test set.
- ◉ If we develop new features by examining the test set, then we may end up choosing features that work well specifically for the test set, so $J_{test}(\theta)$ is no longer a good estimate of how well we generalize to new examples.
- ○ Doing so is less likely to lead to choosing an excessive number of features.

Should discount/discounts/discounted/discounting be treated as the same word?
Can use "stemming" software (E.g. "Porter stemmer")
universe/university.
Error analysis may not be helpful for deciding if this is likely to improve performance. Only solution is to try it and see if it works.
Need numerical evaluation (e.g., cross validation error) of algorithm's performance with and without stemming.
Without stemming: With stemming:

强烈建议使用 cross validation error 而不是 test error 来进行误差分析。

11.3 Error metrics for skewed classes

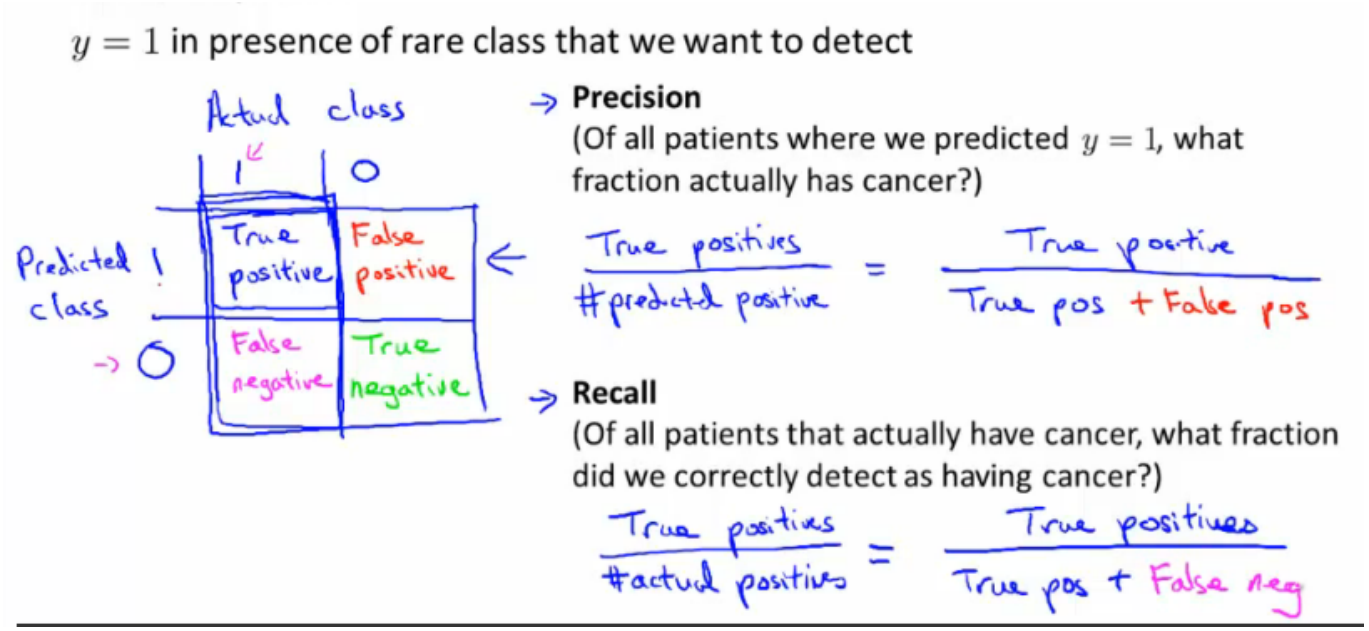
11.3.1 Cancer classification example



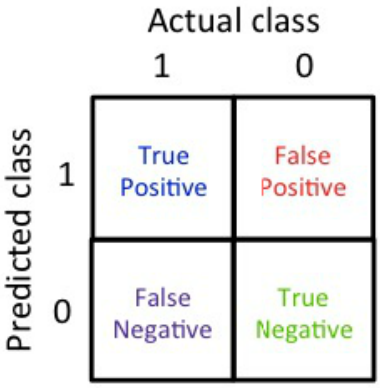
在 skewed class(样本中某一类别出现的频率非常之小)的情况下,使用分类准确率将变得非常困难(比如上例,将诊断正确率从 99.2% 提升到 99.5%,谁知道是不是算法更先进了还是单纯的预测出更多的 y=0?)

对于 skewed class 问题,我们可能需要别的 error metric / evaluation metric。

11.3.2 Pricision/Recall



Precision and recall are defined according to:



Precision:

$$\frac{\text{True positives}}{\#\text{ predicted as positive}} = \frac{\text{True positives}}{\text{True positives + False positiv}}$$

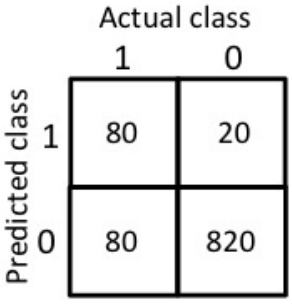
Recall:

$$\frac{\text{True positives}}{\#\text{ actual positives}} = \frac{\text{True positives}}{\text{True positives + False negatives}}$$

Your algorithm's performance on the test set is given to the right. What is the algorithm's precision and recall? Enter your answer as a real number (eg. 0.11, 0.5, etc.).

Precision: 0.8

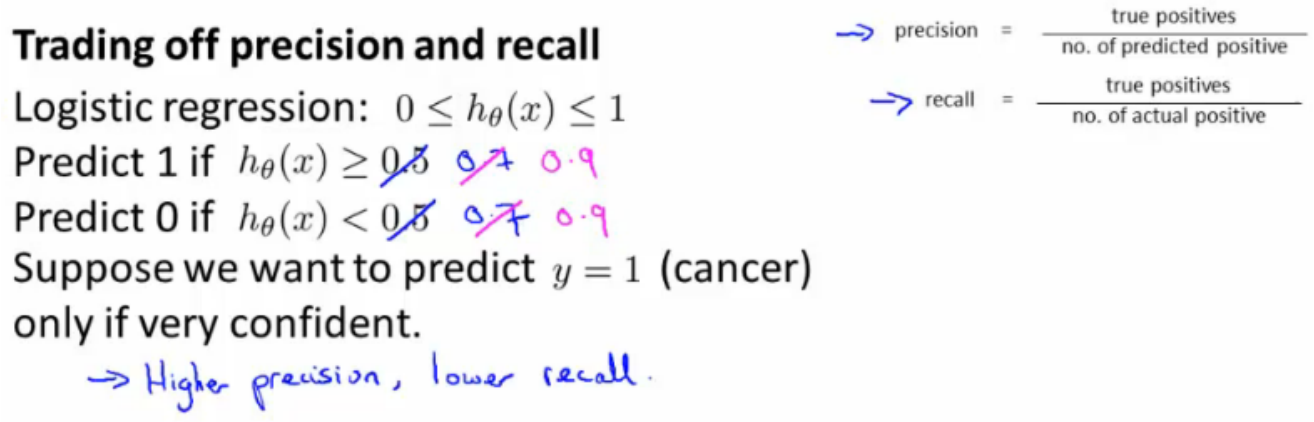
Recall: 0.5



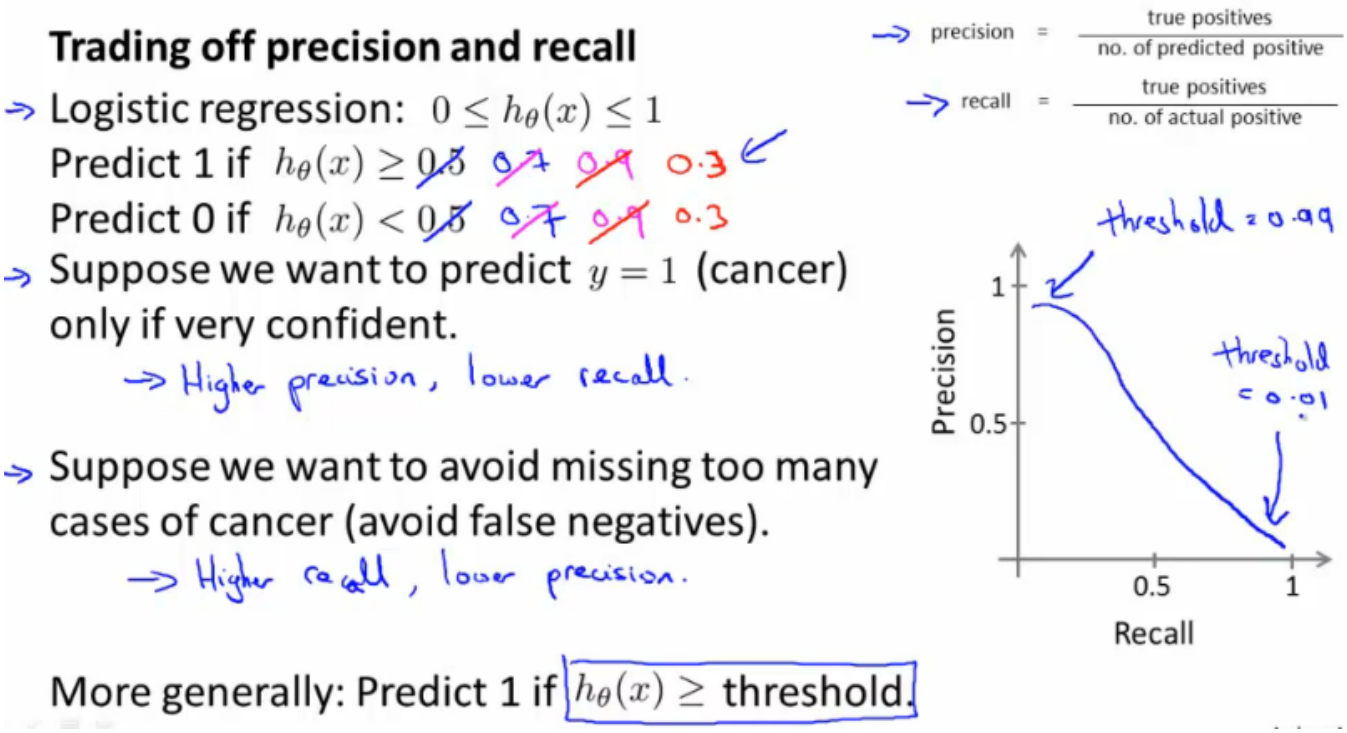
Precision 和 recall 的取值越高越有利。特别地，如果像上一张幻灯片那样所有诊断结果都为 0（良性），那么将得到为 0 的 precision 和 recall 值。

所以，如果我们得到了很高的 precision 和 recall ，那么我们可以相信我们的学习算法，即使样本是非常 skewed 的情况。

11.4 Trading off precision and recall

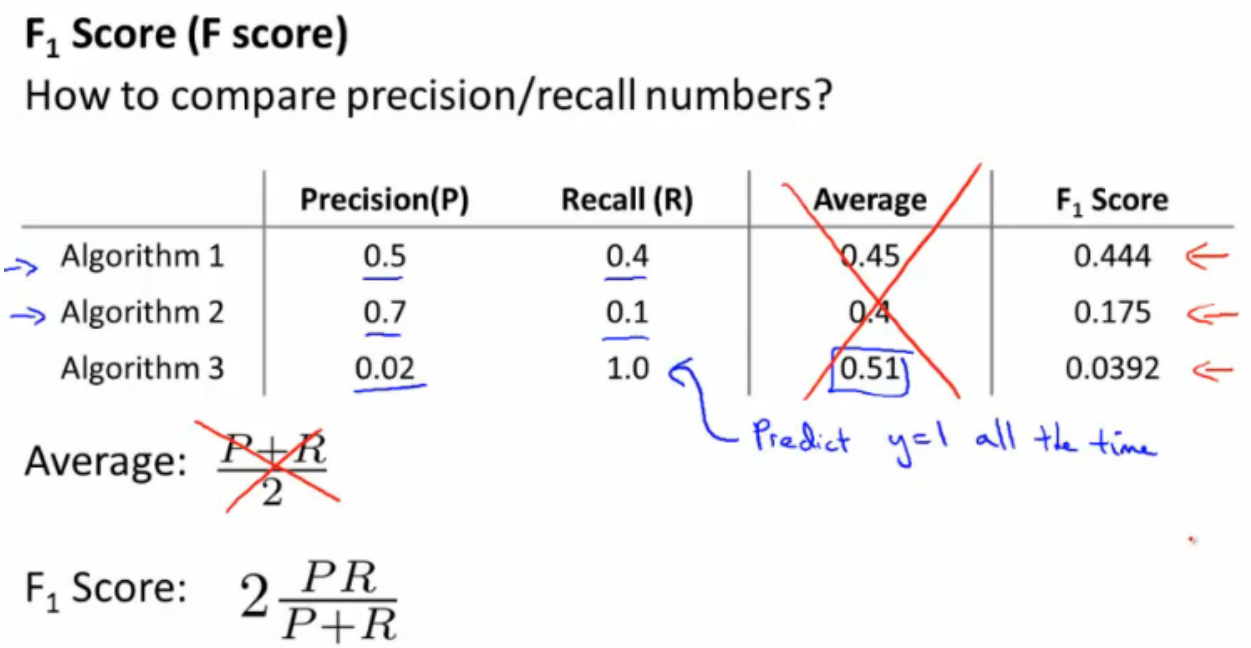


当我们将阈值从 0.5 改成更高的值，以达到只有当我们非常自信时才预测为癌症，那么这会带来更高的 precision，和更低的 recall。



如果我们宁愿错诊也不想让病人错过治疗，那么就需要调低阈值，这样就会带来更高的 recall 和更低的 precision。

11.4.1 F_1 Score (F score)



现在一个学习算法有两个变量来描述，我们希望使用一个单值来衡量一个学习算法。使用平均值是相当没用的方式。这里我们使用 F_1 Score 来描述算法的质量。

F_1 Score (F score)

How to compare precision/recall numbers?

	Precision(P)	Recall (R)	Average	F_1 Score
Algorithm 1	0.5	0.4	0.45	0.444
Algorithm 2	0.7	0.1	0.4	0.175
Algorithm 3	0.02	1.0	0.51	0.0392

Predict y=1 all the time

Average: $\frac{P+R}{2}$

F_1 Score: $2\frac{PR}{P+R}$

P=0 or R=0 ⇒ F-score = 0.

P=1 and R=1 ⇒ F-score = 1

那么,很明显:

You have trained a logistic regression classifier and plan to make predictions according to:

Predict y = 1 if $h_\theta(x) \geq$ threshold
Predict y = 0 if $h_\theta(x) <$ threshold

For different values of the threshold parameter, you get different values of precision (P) and recall (R). Which of of the following would be a reasonable way to pick the value to use for the threshold?

- Measure precision (P) and recall (R) on the **test set** and choose the value of threshold which maximizes $\frac{P+R}{2}$
- Measure precision (P) and recall (R) on the **test set** and choose the value of threshold which maximizes $2\frac{PR}{P+R}$
- Measure precision (P) and recall (R) on the **cross validation set** and choose the value of threshold which maximizes $\frac{P+R}{2}$
- (●) Measure precision (P) and recall (R) on the **cross validation set** and choose the value of threshold which maximizes $2\frac{PR}{P+R}$

11.5 Data for Machine Learning

Large data rationale

Assume feature $x \in \mathbb{R}^{n+1}$ has sufficient information to predict y accurately.

Example: For breakfast I ate ____ eggs.

Counterexample: Predict housing price from only size (feet^2) and no other features.

Useful test: Given the input x, can a human expert confidently predict y?

Large data rationale

→ Use a learning algorithm with many parameters (e.g. logistic regression/linear regression with many features; neural network with many hidden units).

→ $J_{train}(\theta)$ will be small.

Use a very large training set (unlikely to overfit)

→ $J_{train}(\theta) \approx J_{test}(\theta)$

$J_{test}(\theta)$ will be small