Machine Learning

1-1 概念 Introduction

A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measure by P, improves with experience E.

常常使用 { supervised learning
unsupervised learning

others. Reinforcement learning, recommender systems.

1-2 Supervised learning

"right answers" given → Regression (回归): Predict continuous valued output. (like "房子价格")

Classification (分类): Discrete valued outputs (0 or 1) (不一定只有两类)

在监督学习中，对于数据集中的每个样本我们想要算法预测，并给出"正确答案"

(like "肿瘤恶性/良性?")

1-3 unsupervised learning

- 所用数据没有任何标签 (相同或没有)
- 可以将数据分成不同的簇 (聚类算法) ← UL中一种

应用: Organize computing clusters, Social network analysis, Market segmentation, Astronomical data analysis, etc.

- 「鸡尾酒会算法」找出数据结构. → 处理/分离音频

```
[W,S,V] = svd((repmat(sum(x.*x,1),size(x,1),1).*x)*x');
```

svd: 奇异值分解

02 - Linear regression with one variable

01 模型描述 · Linear regression with one variable

Training set of housing prices

Size in feet² (x)	Price in ($000's) (y)
2104	460
1416	232
1534	315
852	178
⋮	⋮

m = number of training examples $\quad m = 47$

$(x^{(i)}, y^{(i)})$ → ith training example

x's = "input" variable / feature

y's = "output" variable / "target" variable.

Training Set → Learning Algorithm → h (hypothesis 假设函数)

size of house x → h → estimated price y

线性回归模型

$h(x) = \theta_0 + \theta_1 x$

02 代价函数 为了让实际和预测更加拟合

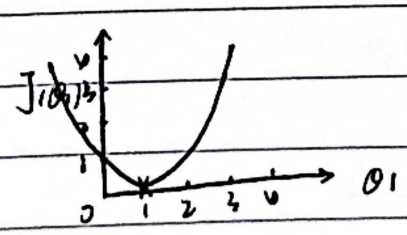
Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$ Parameters: θ_0, θ_1

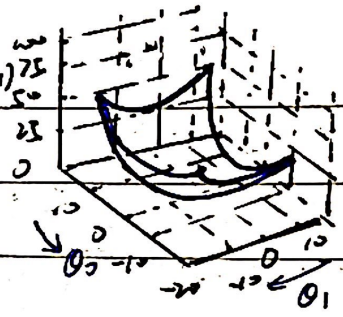
最小二乘法 Cost function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

Goal → $\underset{(\theta_0, \theta_1)}{\text{minimize}}\ J(\theta_0, \theta_1)$

⇓

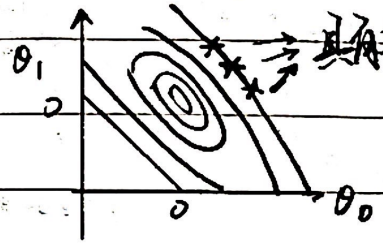
Simplified: $\theta_0 = 0 \Rightarrow J(\theta_1)$ & $\underset{\theta_1}{\text{minimize}}\ J(\theta_1)$





→ 保留 θ_1 和 θ_0，绘制则为 → 三维曲面图

→ 往往为等高线图



→ 具有相同的 $J(\theta_0,\theta_1)$ 值

2. 梯度下降 Outline:

- Start with some θ_0, θ_1
- Keeping changing θ_0, θ_1 to reduce $J(\theta_0,\theta_1)$ until we hopefully end up at a minimum.

- Gradient descent algorithm

repeat until convergence { → 学习率

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0,\theta_1) \quad (\text{for } j=0 \text{ and } j=1)$$

}

:= 赋值运算符

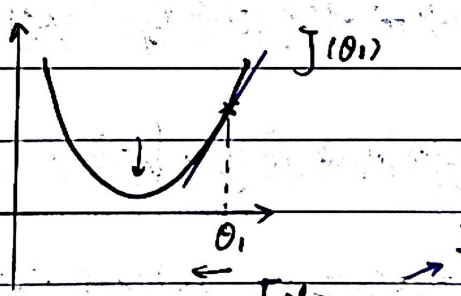
Simultaneous update (实现对更新 θ_0 和 θ_1)

$$temp0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} J(\theta_0,\theta_1)$$

$$temp1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_0,\theta_1)$$

$$\theta_0 := temp0$$

$$\theta_1 := temp1$$



逐渐减小

$$\theta_1 := \theta_1 - \alpha \underbrace{\frac{d}{d\theta_1} J(\theta_1)}_{\geq 0}$$

$$\theta_1 := \theta_1 - \alpha \cdot (\text{positive number})$$

线性回归用梯度下降

$$\frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1) = \frac{\partial}{\partial \theta_j} \cdot \frac{1}{2m} \sum_{i=1}^{m} (h(x^{(i)}) - y^{(i)})^2$$

$$= \frac{\partial}{\partial \theta_j} \cdot \frac{1}{2m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x^{(i)} - y^{(i)})^2$$

⇒ repeat until convergence (收敛) {

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

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

}

"Batch" Gradient Descent

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

4 Linear Regression with multiple variables

4-1 Multiple features

Size x_1	number of bedroom x_2	number of floors x_3	Age of home x_4	Price y
2104	5	1	65	460
1416	3	2	60	232
1534	3	2	30	315
852	2	1	36	178
...	...	...	...	...

$x_1 \sim x_4$

$n = 4$ — number of features

$m = 4$

$x^{(i)}$ = input of i^{th} training example → $x^{(2)} = \begin{bmatrix} 1416 \\ 3 \\ 2 \\ 60 \end{bmatrix}$

$x_j^{(i)}$ = value of feature j in i^{th} training example → $x_3^{(2)} = 2$

⇒ Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + \theta_4 x_4$

线性回归用假设函数形式

define $x_0 = 1$. ⇒ $h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \cdots + \theta_n x_n$

$$x = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad \theta = \begin{bmatrix} \theta_0 \\ \vdots \\ \theta_n \end{bmatrix} \Rightarrow h_\theta(x) = \theta^T x$$

"multivariate linear regression"

多元线性回归

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

4.2 多元梯度下降法 New algorithm ($n \geq 1$):

Repeat {

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

(Simultaneously update θ_j for $j = 0, \cdots n$)

}

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

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

...

4.3 Gradient descent in Practice I: Feature Scaling 特征缩放

Idea: Make sure features are on a similar scale

→ 梯度下降法就能更快收敛

E.g. x_1 = size ($0 - 2000\,m^2$)

x_2 = number of bedrooms ($1 - 5$) ⟹ 范围大，收敛较慢

$$\Rightarrow \quad x_1 = \frac{\text{size}}{2000}, \quad x_2 = \frac{\#\text{ of bedrooms}}{5}$$

So, $0 \leq x_i \leq 1$

- Feature Scaling

  Get every feature into approximately (A) a $-1 \leq x_i \leq 1$ range.

- Mean normalization 均值归一化

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

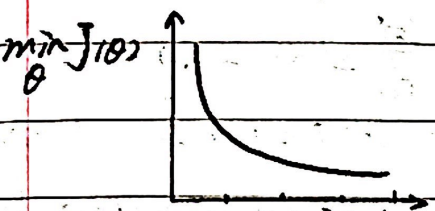
E.g. $x_1 = \frac{\text{size} - 1000}{2000}$ → $-0.5 \leq x_1 \leq 0.5$

$x_2 = \frac{\#\text{bedroom} - 2}{5}$ → $-0.5 \leq x_2 \leq 0.5$

2.4 Gradient descent in practice II : Learning rate

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

- Making sure gradient descent is working correctly



→ Example automatic convergence test:
Declare convergence if $J(\theta)$ decreases by less than ε (e.g. 10^{-3}) in one iteration

△ Gradient descent not working → Use smaller α.

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

2.5 Features and polynomial regression

- 选择 features 并如何使用？ ⇒ 定义新的特征来获得更好的模型

Idea I

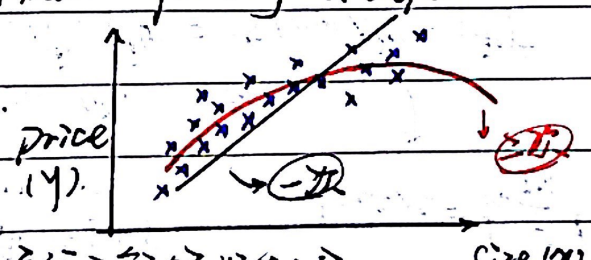
E.g. Housing prices Prediction

$h_\theta(x) = \theta_0 + \theta_1 \times \text{frontage} + \theta_2 \times \text{depth}$ （纵向深度）

（临街宽度）

也可以选择将两特征合并，即 Area = frontage × depth

⇒ $h_\theta(x) = \theta_0 + \theta_1 x$ ↘ Area.



Idea II

线型回归 ↔ 非线型回归

E.g. 对于 size ↔ price. 如果直线不能很好拟合

→ 考虑选择二次函数（但会出现下降趋势）

⇒ 考虑三次函数

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 = \theta_0 + \theta_1 (\text{size}) + \theta_2 (\text{size})^2 + \theta_3 (\text{size})^3$

x_1 x_2 x_3

（要使用特征缩放来 梯度下降）

or $h_\theta(x) = \theta_0 + \theta_1 (\text{size}) + \theta_2 \sqrt{\text{size}}$

2-6 Normal equation 正规方程（区别于迭代方法的直接解法）

- 求θ解析解，一步得到
- 原理：

$\theta \in \mathbb{R}^{n+1}$ $J(\theta_0, \cdots, \theta_m) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2$

$\frac{\partial}{\partial \theta_j} J(\theta) = \cdots = 0$ (for every j)

Solve for $\theta_0, \cdots, \theta_m$.

- Example

	x_0	x_1	x_2	x_3	x_4	y
房价预测	1	2104	5	1	45	460
	1	1416	3	2	40	232
特征变量	1	1534	3	2	30	315
	1	852	2	1	36	178

$m \times (n+1)$ 维

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix}$$

$$y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$ m 维

(矩阵分析中没有一次求解试试)

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

设置不同的

- for m example, n features

$$x^{(i)} = \begin{bmatrix} x_0^{(i)} \\ x_1^{(i)} \\ \vdots \\ x_n^{(i)} \end{bmatrix} \rightarrow X = \begin{bmatrix} -(x^{(1)})^T- \\ \vdots \\ -(x^{(m)})^T- \end{bmatrix}$$ (design matrix) $m \times (n+1)$

Gradient Descent	Normal Equation
· Need to choose α	· No need to choose α
· Needs many iterations	· No need to iterate
· Works well even when n is large	· Need to compute $(X^T X)^{-1}$ → Slow if n is very large

4] Normal equation and non-invertibility

What if X^TX is non-invertible?

- Redundant features (linearly dependent)

  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

6 Logistic Regression

6-1 Classification 分类 因变量 $y \in \{0, 1\}$ 二分类

使用线性回归(不推荐)

→ Threshold classifier output $h_\theta(x)$ at 0.5.

If $h_\theta(x) \leq 0.5$, predict "$y=0$"

If $h_\theta(x) > 0.5$, predict "$y=1$"

⇒ 新算法 logistic Regression

特点：预测值一直介于0和1之间 $(0 \leq h_\theta(x) \leq 1)$

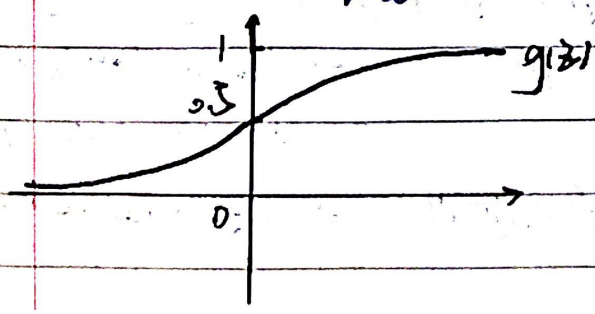
6-2 Hypothesis Representation 假设表示

Logistic Regression Model

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

$h_\theta(x) = g(\theta^T x)$ → $g(z) = \frac{1}{1+e^{-z}}$ } Sigmoid function / logistic function

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



· Interpretation of Hypothesis Output

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

Example

if $x = \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 1 \\ \text{tumorSize} \end{bmatrix}$

$h_\theta(x) = 0.7$ → 特征为 x，$y=1$ 时的概率

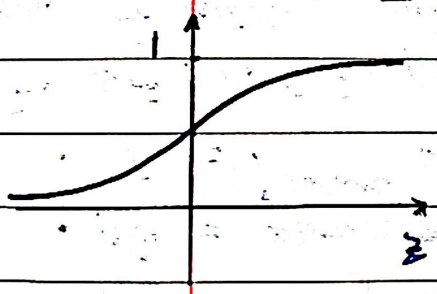
→ Tell patient that 70% chance of tumor being maligent

→ (假设函数的一个属性)

4-3 Decision boundary 决策边界

$h_\theta(x) = g(\theta^T x)$ → $P(y=1 \mid x, \theta)$ ← 给定参数 x 和 θ，$y=1$ 概率估计

$z = \theta^T x$



Suppose predict

"$y=1$" if $h_\theta(x) \geq 0.5$ → $\theta^T x \geq 0$

"$y=0$" if $h_\theta(x) < 0.5$ → $\theta^T x < 0$

4-4 Cost function

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

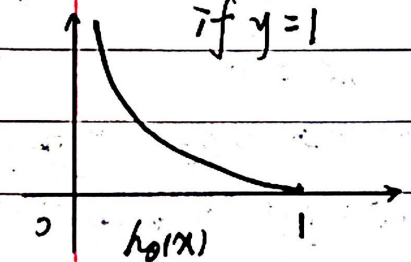
→ $\text{Cost}(h_\theta(x) - y) = \frac{1}{2}(h_\theta(x) - y)^2$

对于 logistic regression

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

"最大似然估计"

代价函数



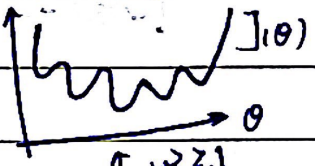
Cost = 0 if $y=1$, $h_\theta(x)=1$

But as $h_\theta(x) \to 0$

Cost → ∞

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

"non-convex"

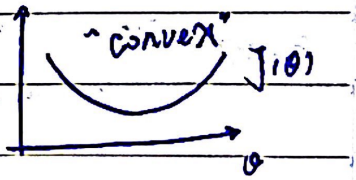


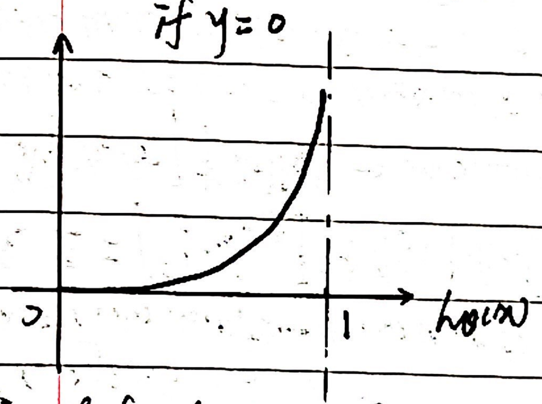
↑ 简化

若用"最小二乘"

↓

"convex"





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

4-5 Simplified cost function and gradient descent

logistic regression cost function (也叫交叉熵损失函数 Cross-entropy

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)})$$ 本质上也是一种对数似然函数)

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

note: y=0 or 1 always

★ Loss (error) function ↕(区别)

损失函数 ⟹ $-y\log(h_\theta(x)) - (1-y)\log(1-h_\theta(x))$ ⟹ Cost$(h_\theta(x), y)$

⟹

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

To fit parameter θ:

$\min_\theta J(\theta)$ → Get θ

To make a prediction given new x:

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

- Gradient Descent

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

Repeat {

$\theta_j := \theta_j - \alpha\left[\frac{\partial}{\partial\theta_j}J(\theta)\right]$ (simultaneously update all θ_j)

}

↓

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

(注 log <-> ln)

和 linear regression 结果一致
只"ㄋ=2" 实际上完全不同

11-6 Advanced optimization 高级优化

Given θ, we have code that can compute:

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

Optimization algorithms

- Conjugate gradient
- BFGS
- L-BFGS

} 共轭梯度法

Advantages:

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

Disadvantages:

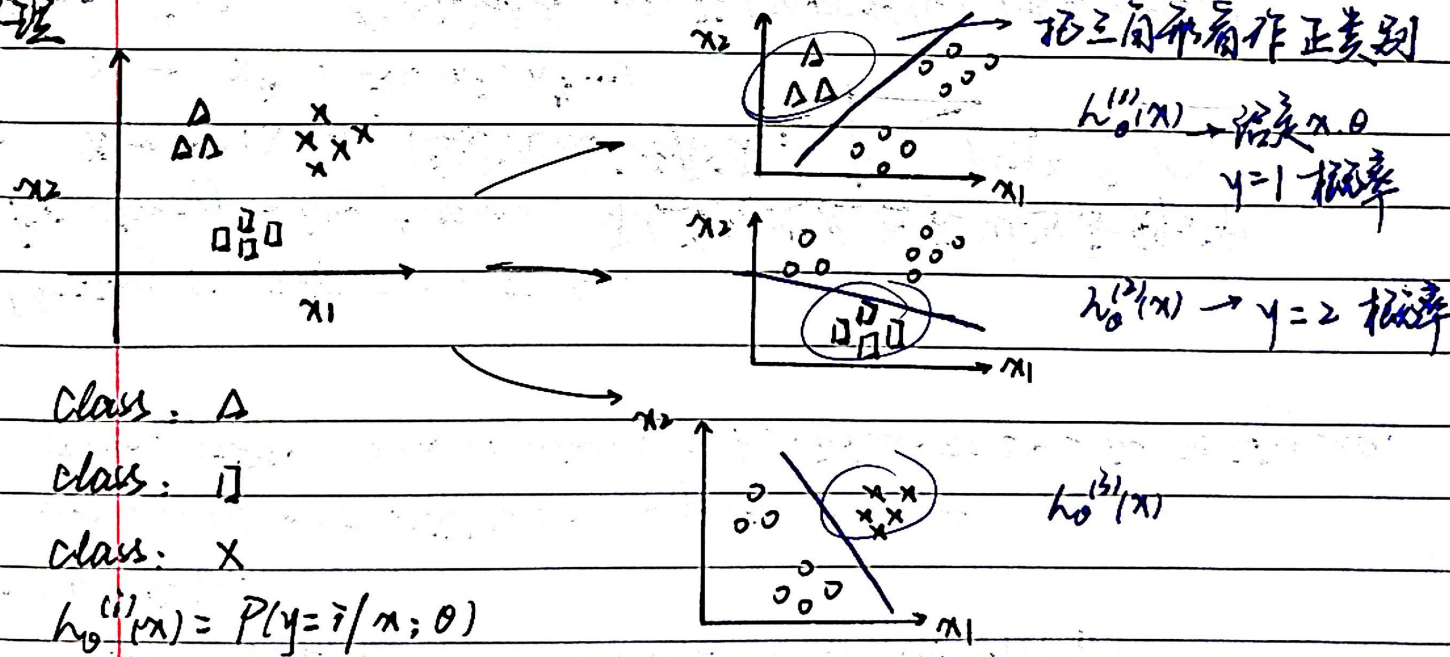
- More complex

11-7 Multi-class classification: One-vs-all. 多类分类

E.g. Weather: Sunny, Cloudy, Rain, Snow

$y=1$ $y=2$ $y=3$ $y=4$

原理:



Class: △

class: □

class: X

$h_\theta^{(i)}(x) = P(y=i|x;\theta)$

拟合出三个分类器

总结:

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 $\max_i h_\theta^{(i)}(x)$.

5 Regularization

5-1 The problem of overfitting 过拟合问题

- 没有很好拟合，具有高偏差 → 欠拟合
- 假设函数h学能拟合所有数据 → 函数太过庞大，变量太多，无约束 → 拟合例所集，导致无法泛化到新样本中 → 过拟合

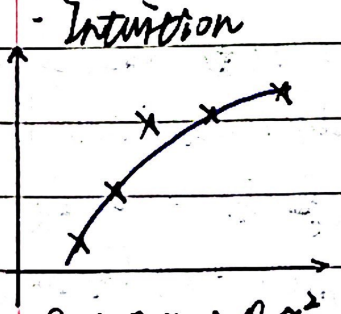
具有"高方差"

通常在变量太多时 (too many feature)

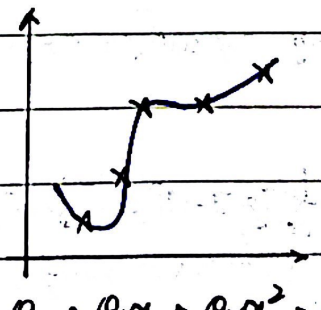
- 解决方式
  - Reduce the number of feature
    - 人工检查变量清单 → 重要 → 保留
    - 模型选择算法 → 缺点：舍弃变量也意味舍弃了信息
  - Regularization 正则化
    - keep all the features, but reduce magnitude/values of parameters θ_j
    - Works well when we have a lot of features, each of which contributes a bit to predicting y.

5-2 Cost function

- Intuition



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



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

- Suppose we penalize and make θ_3, θ_4 very small.

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

为了让上式合理 → θ_3, θ_4 趋于0.

正则化思想 - Small values for parameters $\theta_0, \theta_1, \cdots, \theta_n$

- 'Simpler' hypothesis
- Less prone to overfitting

↙ 函数更平滑.

如果给 $\theta_0 \sim \theta_n$ 都加上惩罚项 → 简化函数

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

因为不知道 θ_0 哪些参数，添加一个额外正则化项 → 每个参数值

→ 如 θ_0 与否影响不大，一般不对 θ_0 正则化

- 将 $J(\theta)$ 分为二个目标：
  - 第一项：为了更好拟合训练集
  - 第二项：保持参数尽可能小

λ：正则化参数，控制以上两项间平衡关系

- λ 过大，会导致 $\theta_1 \sim \theta_n$ 均 $\to 0$，只剩 θ_0（直线）

5-3 Regularized linear regression 线性回归的正则化

- Gradient descent

Repeat {

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

$\theta_j := \theta_j - \alpha\left[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \frac{\lambda}{m}\theta_j\right]$ $j=(1,2,3,\cdots,n)$

}

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

由于 α 偏小 $\Rightarrow \alpha\frac{\lambda}{m}$ 会是一个很小的数

$\Rightarrow (1 - \alpha\frac{\lambda}{m})$ 略小于 1

也就是每次迭代时都将 θ_j 乘以一个比 1 略小的数，然后进行和之前一样的更新操作

$x^{(m)} = [x_0^{(m)}\ x_1^{(m)} \dots x_n^{(m)}]$

- Normal equation

$$X = \begin{bmatrix} (x^{(1)})^T \\ \vdots \\ (x^{(m)})^T \end{bmatrix}_{m \times n+1} \qquad y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in R^m$$

$$\text{求 } \min_\theta J(\theta) \rightarrow \theta = (X^TX + \lambda \begin{bmatrix} 0 & & & \\ & 1 & & \\ & & \ddots & \\ & & & 1 \end{bmatrix})^{-1} X^T y$$

Regularized

5-4 logistic regression 逻辑回归的正则化

- Cost function

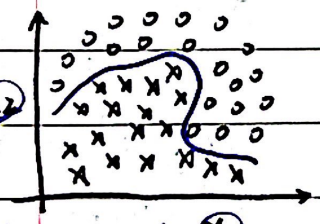
$$J(\theta) = -\left[\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log (1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

- 实现形式和 5-3 中 Gradient descent 一致，区别 $h_\theta(x)$ 表达式不同

6 Neural Networks: Representation (表示)

6-1 Non-linear hypotheses 非线性假设

- Non-linear classification



→ 可构造包含很多非线性项的 logistic 回归函数

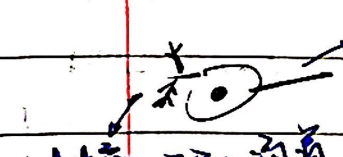
$$g(\theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_1 x_2 + \theta_4 x_1^2 x_2 + \theta_5 x_1^3 x_2 + \theta_6 x_1 x_2^2 + \dots)$$

- 但当初始特征个数 n 很大时，将高阶多项式项数包括到特征里，会使特征空间急剧膨胀

6-2 Neurons and brain 神经元和大脑

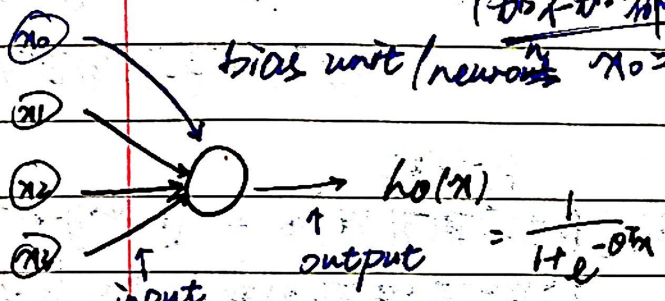
- Origins: Algorithms that try to mimic the brain

6-4 模型表示-1 Model representation



动作电位 ← 阈值

- Logistic unit

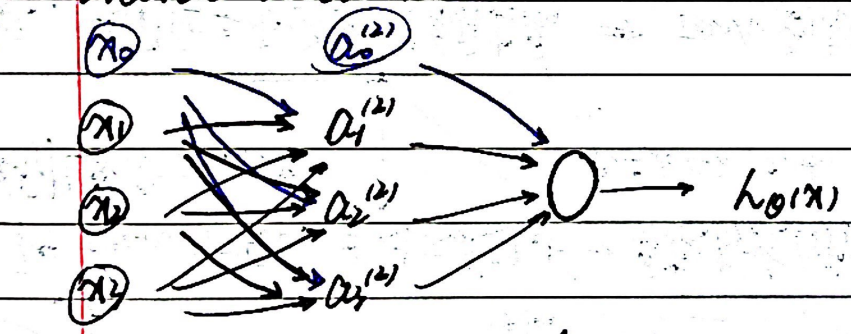


具体情况具体分析

权重 "weights" or 参数 parameters

- 等价 Sigmoid (logistic) 激活函数的人工神经元

- Neural Network



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

激活项：由一个具体神经元计算并输出的值

$\Theta^{(j)}$ = 权重矩阵，控制着 layer j 到 layer $j+1$ 的映射

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

$$\vdots$$

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

如果 S_j units in layer j，S_{j+1} units in layer $j+1$，则 $\Theta^{(j)} \to S_{j+1} \times (S_j + 1)$

6-4 模型展示-II

def $a_1^{(2)} = g(z_1^{(2)})$, $a_2^{(2)} = g(z_2^{(2)})$, $a_3^{(2)} = g(z_3^{(2)})$

$x = \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix}$, $z^{(2)} = \begin{bmatrix} z_1^{(2)} \\ z_2^{(2)} \\ z_3^{(2)} \end{bmatrix}$ $z^{(2)} = \Theta^{(1)} x$

$\rightarrow a^{(2)} = g(z^{(2)}) \rightarrow z^{(3)} = \Theta^{(2)} a^{(2)} \rightarrow h_\Theta(x) = a^{(3)} = g(z^{(3)})$

- 上述过程称为前向传播

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

和 logistic regression 类似，但 Neural Network 没有用输入特征 x_1, x_2, x_3 来训练逻辑回归，而是自己训练逻辑回归时输入 a_1, a_2, a_3

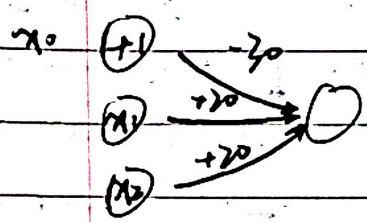
$\Rightarrow$ 为 $\Theta^{(1)}$ 选择不同参数，可以得到一个更好的假设函数

相比于直接使用原始特征 x_1, x_2, x_3.

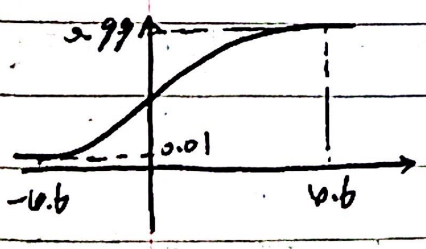
6-5 Examples and intuitions 例子和直观理解 I (单个神经元如何被用来计算逻辑函数)

- 异或 XOR：同为0(假)，异为1(真)
- 同或 ~~XNOR~~ XN：同真异假

- AND $x_1, x_2 \in \{0,1\}$, $y = x_1$ AND x_2



$h_\Theta(x) = g(-30 + 20x_1 + 20x_2)$



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	$g(10) \approx 1$

- OR $g(-10 + 20x_1 + 20x_2)$

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

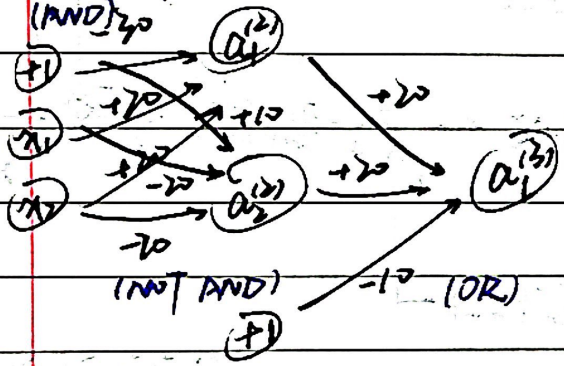
- NOT $g(10 - 20x_1)$

x_1	$h_\theta(x)$
0	$g(10) \approx 1$
1	$g(-10) \approx 0$

- (NOT x_1) AND (NOT x_2) $\rightarrow$ $g(10 - 20x_1 - 20x_2)$

6-6 例子和直觉理解II

- 构建 x_1 XNOR x_2 (putting AND, OR, (NOT AND) together)



x_1	x_2	$a_1^{(2)}$	$a_2^{(2)}$	$h_\theta(x)$
0	0	0	1	1
0	1	0	0	0
1	0	0	0	0
1	1	1	0	1

6-7 多元分类

- Multiple output units : one-vs-all

$\boxed{\text{Na} \cdots \text{Nec}} \rightarrow h_\theta(x) \in \mathbb{R}^4$

Want $h_\theta(x) \approx \begin{bmatrix}1\\0\\0\\0\end{bmatrix}$, $h_\theta(x) \approx \begin{bmatrix}0\\1\\0\\0\end{bmatrix}$, $h_\theta(x) \approx \begin{bmatrix}0\\0\\1\\0\end{bmatrix}$, etc.

when pedestrian, when car, when motorcycle

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

$y^{(i)}$ one of $\begin{bmatrix}1\\0\\0\\0\end{bmatrix}, \begin{bmatrix}0\\1\\0\\0\end{bmatrix}, \begin{bmatrix}0\\0\\1\\0\end{bmatrix}, \begin{bmatrix}0\\0\\0\\1\end{bmatrix}$ $\Rightarrow h_\theta(x^{(i)}) \approx y^{(i)}$

pedestrian, car, motor, truck

L7: Neural Networks: Learning

7-1 Cost function

- Neural Network (Classification)

$\{(x^{(1)}, y^{(1)}) \cdots (x^{(m)}, y^{(m)})\}$. $\begin{cases} L = \text{total no. of layers} \\ S_l = \text{no. of units (no bias) in layer } l \end{cases}$

- Binary classification

$y = 0$ or 1 . 1 output unit . $h_\Theta(x) \in \mathbb{R}$. $S_L = 1$ ($k=1$)

- Multi-class classification (K class)

$y \in \mathbb{R}^K$ (K维向量) . K output unit . $h_\Theta(x) \in \mathbb{R}^K$. $S_L = K$.

- Cost function

  - logistic regression:

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

  - Neural network:

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

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

- 不对 bias, 即 $i=0$ 求和

- 最后一层 K 个输出单元

~~[~~ [后一层一个单元和前一层所有单元连接]

7-2 Backpropagation algorithm 反向传播算法

- Need to compute $\begin{cases} \to J(\theta) \\ \to \frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\theta) \end{cases}$

- forward propagation (e.g. layer = 4)

$a^{(1)} = x$. $z^{(2)} = \Theta^{(1)} a^{(1)}$

$a^{(2)} = g(z^{(2)})$ (add $a_0^{(2)}$) . $z^{(3)} = \Theta^{(2)} a^{(2)}$

$a^{(3)} = g(z^{(3)})$ (add $a_0^{(3)}$) . $z^{(4)} = \Theta^{(3)} a^{(3)}$

$a^{(4)} = h_\Theta(x) = g(z^{(4)})$

引入"误差"的概念，然后通过"误差"来得到梯度

- Backpropagation

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

for each output unit (layer $L=4$)

$\delta_j^{(4)} = a_j^{(4)} - y_j$ → 向量表示 $\delta^{(4)} = a^{(4)} - y$

→ $\delta^{(3)} = (\Theta^{(3)})^T \delta^{(4)} .* g'(z^{(3)})$

→ $\delta^{(2)} = (\Theta^{(2)})^T \delta^{(3)} .* g'(z^{(2)})$ → $a^{(2)} .* (1-a^{(2)})$

激活函数偏导数

No $\delta^{(1)}$

→ 输入层(layer 1)不存在误差

⇒ 可简单证明 $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\theta) \Rightarrow a_j^{(l)} \delta_i^{(l+1)}$ (ignore λ)

- Backpropagation algorithm

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

Set $\Delta_{ij}^{(l)} = 0$ (for all l, i, j)

for $i = 1$ to m

Set $a^{(1)} = x^{(i)}$

perform forward propagation to compute $a^{(l)}$ for $l = 2, 3, \cdots, L$

using $y^{(i)}$, compute $\delta^{(L)} = a^{(L)} - y^{(i)}$

Compute $\delta^{(L-1)}, \delta^{(L-2)}, \cdots, \delta^{(2)}$

$\Delta_{ij}^{(l)} := \Delta_{ij}^{(l)} + a_j^{(l)} \delta_i^{(l+1)}$ (with all i, j 等和？误差和)

向量表示 $\Delta^{(l)} := \Delta^{(l)} + \delta^{(l+1)} (a^{(l)})^T$

计算梯度矩阵

$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)} + \lambda \Theta_{ij}^{(l)}$ if $j \neq 0$

$D_{ij}^{(l)} := \frac{1}{m} \Delta_{ij}^{(l)}$ if $j = 0$

⇒ ☆ $\frac{\partial}{\partial \Theta_{ij}^{(l)}} J(\theta) = D_{ij}^{(l)}$

↓

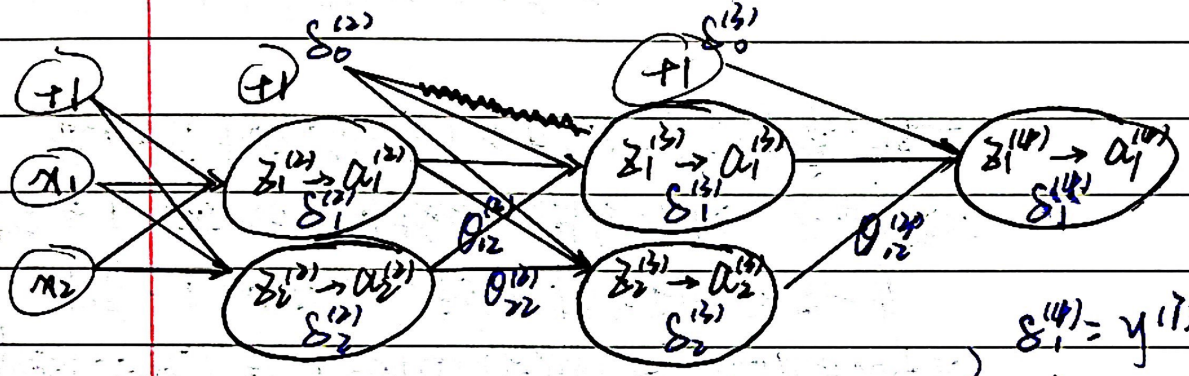
然后可以使用梯度下降或其它算法.

7-3 Backpropagation intuition 理解反向传播

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

$$Cost(i) = y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log h_\theta(x^{(i)})$$

(Think of $Cost(i) \approx (h_\theta(x^{(i)}) - y^{(i)})^2$)



$$\delta_1^{(4)} = y^{(i)} - a_1^{(4)}$$

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

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

Formally, $\delta_j^{(l)} = \frac{\partial}{\partial z_j^{(l)}} Cost(i)$ (for $j \geq 0$)

7-4 使用注意：展开参数 Implementation note: Unrolling parameters

- Advanced optimization 中输入的 theta/initial theta 为 n/n+1 维向量
  gradient 也为 n/n+1 维 vector.

- Neural Network ($L = 4$) → 为矩阵形式

参数矩阵 $\Theta^{(1)}, \Theta^{(2)}, \Theta^{(3)}$ - matrices (Theta1, Theta2, Theta3)

(返回值) 梯度矩阵 $D^{(1)}, D^{(2)}, D^{(3)}$ - matrices (D1, D2, D3)

如何展开为向量？

(unroll)

7-5 Gradient checking 梯度检测 → (避免算法产生 bug)

验证反向传播

- 原理 (双侧差分)

$$\frac{d}{d\theta}J(\theta) \approx \frac{J(\theta+\varepsilon)-J(\theta-\varepsilon)}{2\varepsilon}$$

ε 足够小 - e.g. $\varepsilon = 10^{-4}$

上述 θ 为实数

- Parameter vector θ, $\theta \in \mathbb{R}^n$ (e.g. θ 为 "unrolled" version of $\theta^{[1]}, \theta^{[2]}, \theta^{[3]}$)

$\theta = [\theta_1, \cdots, \theta_n]$

$$\frac{\partial}{\partial\theta_1}J(\theta) \approx \frac{J(\theta_1+\varepsilon, \theta_2, \cdots, \theta_n)-J(\theta_1-\varepsilon, \theta_2, \cdots, \theta_n)}{2\varepsilon}$$

$$\vdots$$

$$\frac{\partial}{\partial\theta_n}J(\theta) \approx \frac{J(\theta_1, \theta_2, \cdots, \theta_n+\varepsilon)-J(\theta_1, \theta_2, \cdots, \theta_n-\varepsilon)}{2\varepsilon}$$

在算法中可利用 for 循环遍历上述所有参数偏导数 gradApprox

然后与反向传播中得到的梯度进行比较 Dvec

一定确定 gradApprox 和 Dvec 近似相等. 在开始学习/训练网络前关闭"梯度检测".

→ · "gradApprox" 计算量大, 运算速度慢

· "Dvec" 高效

7-6 Random initialization 随机初始化

(可参考 Deep Learning 笔记 1-11 节)

- 如何对 θ 设初始值?

· 假设 zero initialization, 即 $\theta_{ij}^{(l)} = 0$ for all l, i, j

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

→ 对称权重问题, 即所有权重都一样

只有一个特征时设网络无法学习

{ Solution

random initialization

- Random initialization : Symmetry breaking 对称性破坏

Initialize each $\Theta_{ij}^{(l)}$ to a random value in $[-\varepsilon, \varepsilon]$

Summarize，为了训练神经网络，应首先将权重随机初始化为一个接近0的，范围在$-\varepsilon$到ε之间的数，然后进行反向传播，再梯度检测，最后使用梯度下降(或其它高级优化算法)

7-1 Putting it together

- Training a neural network

I. - pick a network architecture (connectivity pattern between neurons)

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

→ No. of output unit: Number of classes → like $y = \begin{bmatrix}1\\0\\\vdots\\0\end{bmatrix}$ or $\begin{bmatrix}0\\1\\\vdots\\0\end{bmatrix}$ or $\begin{bmatrix}0\\0\\\vdots\\1\end{bmatrix}$

- Reasonable default:

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

And hidden units no. ≥ No. of input units

II. 步骤

1. randomly initialize weights (见上述Summarize)
2. Implement forward propagation to get $h_\Theta(x^{(i)})$ for any $x^{(i)}$
3. Implement code to compute cost function $J(\Theta)$
4. Implement backprop to compute practial derivatives $\frac{\partial}{\partial \Theta_{jk}^{(l)}} J(\Theta)$
5. Using gradient checking (见7-5)
6. Using gradient descent or advanced optimization method with backpropagation to try to minimize $J(\Theta)$ as a function of parameters Θ.

注：由于$J(\Theta)$是非凸函数 → 理论上都可能收敛于局部最小值