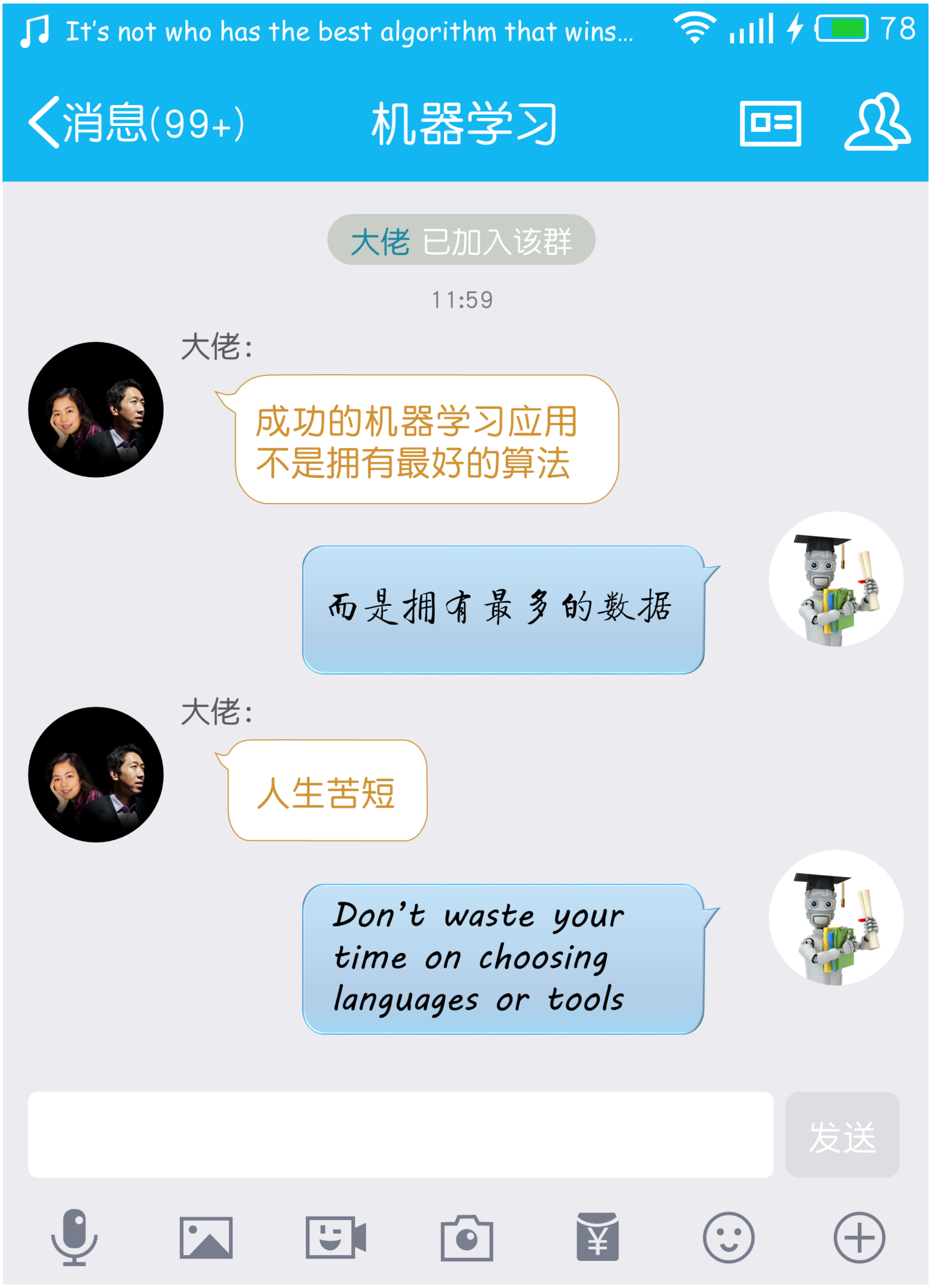
$$\frac{\partial \vec{x}^T \mathbf{A}}{\partial \vec{x}} = \mathbf{A}^T \quad \left(\mathbf{A}_{n\times m}\text{不是}\vec{\boldsymbol{x}}_{n\times 1}\text{的函数}\right)$$



It's not who has the best algorithm that wins... 78

<消息(99+) 机器学习

大佬 已加入该群

11:59

大佬：

成功的机器学习应用
不是拥有最好的算法

而是拥有最多的数据

大佬：

人生苦短

Don't waste your
time on choosing
languages or tools

发送

# 机器学习笔记

## 一、线性代数相关

### **1.**矩阵的求导

| 表达形式 | 公式 |
|---|---|
| $\frac{\partial 向量}{\partial 标量}$ | $\vec{y}=\begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_m \end{pmatrix} \qquad \frac{\partial \vec{y}}{\partial x}=\begin{pmatrix} \frac{\partial y_1}{\partial x} \\ \frac{\partial y_2}{\partial x} \\ \cdots \\ \frac{\partial y_m}{\partial x} \end{pmatrix}$ |
| $\frac{\partial 标量}{\partial 向量}$ | $\vec{x}=\begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{pmatrix} \qquad \frac{\partial y}{\partial \vec{x}}=\begin{pmatrix} \frac{\partial y}{\partial x_1} & \frac{\partial y}{\partial x_2} & \cdots & \frac{\partial y}{\partial x_n} \end{pmatrix}$ |
| 标量函数关于空间向量的方向导数 | $\nabla_{\vec{u}} f(x) \triangleq \nabla f(x)\cdot\vec{u}=\frac{\partial f(x)}{\partial x}\cdot\vec{u}$ |
| $\frac{\partial 向量}{\partial 向量}$ | $\vec{y}=\begin{pmatrix} y_1 \\ y_2 \\ \cdots \\ y_m \end{pmatrix} \quad \vec{x}=\begin{pmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{pmatrix}$ $\frac{\partial \vec{y}}{\partial \vec{x}}=\begin{pmatrix} \frac{\partial y_1}{\partial \vec{x}} \\ \frac{\partial y_2}{\partial \vec{x}} \\ \cdots \\ \frac{\partial y_m}{\partial \vec{x}} \end{pmatrix}=\begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \cdots & \frac{\partial y_2}{\partial x_n} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial y_m}{\partial x_1} & \frac{\partial y_m}{\partial x_2} & \cdots & \frac{\partial y_m}{\partial x_n} \end{pmatrix}_{m\times n}$ |
| $\frac{\partial 矩阵}{\partial 标量}$ | $\frac{\partial \mathbf{Y_{m\times n}}}{\partial x}=\begin{pmatrix} \frac{\partial y_{11}}{\partial x} & \frac{\partial y_{12}}{\partial x} & \cdots & \frac{\partial y_{1n}}{\partial x} \\ \frac{\partial y_{21}}{\partial x} & \frac{\partial y_{22}}{\partial x} & \cdots & \frac{\partial y_{2n}}{\partial x} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial y_{m1}}{\partial x} & \frac{\partial y_{m2}}{\partial x} & \cdots & \frac{\partial y_{mn}}{\partial x} \end{pmatrix}_{m\times n}$ |
| $\frac{\partial 标量}{\partial 矩阵}$ | $\frac{\partial y}{\partial \mathbf{X_{p\times q}}}=\begin{pmatrix} \frac{\partial y}{\partial x_{11}} & \frac{\partial y}{\partial x_{21}} & \cdots & \frac{\partial y}{\partial x_{p1}} \\ \frac{\partial y}{\partial x_{12}} & \frac{\partial y}{\partial x_{22}} & \cdots & \frac{\partial y}{\partial x_{p2}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial y}{\partial x_{1q}} & \frac{\partial y}{\partial x_{2q}} & \cdots & \frac{\partial y}{\partial x_{pq}} \end{pmatrix}_{q\times p}$ 通常写作 $\nabla_{\mathbf{X}} y(\mathbf{X})=\frac{\partial \mathbf{y}(\mathbf{X})}{\partial \mathbf{X}}$ |
| 标量函数关于矩阵的方向导数 | $\nabla_{\mathbf{Y}} f=tr\left(\frac{\partial f}{\partial \mathbf{X}}\mathbf{Y}\right)$ |

| 表达形式 | 公式 |
|---|---|
| $\frac{\partial 矩阵}{\partial 矩阵}$ | $\frac{\partial \mathbf{F_{p\times q}}}{\partial \mathbf{X_{n\times m}}} = \begin{pmatrix} \frac{\partial \mathbf{F}}{\partial \mathbf{X_{1,1}}} & \frac{\partial \mathbf{F}}{\partial \mathbf{X_{1,2}}} & \cdots & \frac{\partial \mathbf{F}}{\partial \mathbf{X_{1,n}}} \\ \frac{\partial \mathbf{F}}{\partial \mathbf{X_{2,1}}} & \frac{\partial \mathbf{F}}{\partial \mathbf{X_{2,2}}} & \cdots & \frac{\partial \mathbf{F}}{\partial \mathbf{X_{2,n}}} \\ \cdots & \cdots & \cdots & \cdots \\ \frac{\partial \mathbf{F}}{\partial \mathbf{X_{m,1}}} & \frac{\partial \mathbf{F}}{\partial \mathbf{X_{m,2}}} & \cdots & \frac{\partial \mathbf{F}}{\partial \mathbf{X_{m,n}}} \end{pmatrix}_{mp\times nq}$ <br> 其中每一项$\frac{\partial \mathbf{F}}{\partial \mathbf{X_{i,j}}}$都是$p\times q$的矩阵 |
| 矩阵函数关于矩阵的方向导数 | $\nabla_{\mathbf{Y}}\mathbf{F} = \mathbf{tr}\left(\frac{\partial \mathbf{F}}{\partial \mathbf{X}}\mathbf{Y}\right)$ |

- 常用公式

($\mathbf{X}$、$\mathbf{Y}$均为矩阵)

> $\frac{\partial \mathbf{X}}{\partial \mathbf{X}} = \mathbf{I}$
>
> $\frac{\partial a\mathbf{Y}}{\partial \mathbf{X}} = a\frac{\partial \mathbf{Y}}{\partial \mathbf{X}}$ ($a$不是$\mathbf{X}$的函数)
>
> $\frac{\partial \vec{x}^T\mathbf{A}\vec{x}}{\partial \vec{x}} = (\mathbf{A}+\mathbf{A}^T)\vec{x}$ ($\mathbf{A}$不是$\mathbf{X}$的函数，$\vec{x}^T\mathbf{A}\vec{x}$为标量)
>
> $\frac{\partial^2 \vec{x}^T\mathbf{A}\vec{x}}{\partial \vec{x}^2} = \mathbf{A}+\mathbf{A}^T$ ($\mathbf{A}$不是$\mathbf{X}$的函数，$\vec{x}^T\mathbf{A}\vec{x}$为标量)
>
> $\frac{\partial \vec{u}\vec{v}}{\partial \mathbf{X}} = \vec{u}\frac{\partial \vec{v}}{\partial \mathbf{X}} + \vec{v}\frac{\partial \vec{u}}{\partial \mathbf{X}}$ ($\vec{u}\vec{v}$是标量)?
>
> $\frac{\partial \vec{u}^T}{\partial x} = \left(\frac{\partial \vec{u}}{\partial x}\right)^T$

| 分子布局(**Numerator Layout**)<br>(对分母转置，分子不转置) | 分母布局(**Denominator Layout**)<br>(对分子转置，分母不转置) |
|---|---|
| $\frac{\partial \vec{x}^T\mathbf{A}}{\partial \vec{x}} = \mathbf{A}^T$ ($\mathbf{A}_{n\times m}$不是$\vec{x}_{n\times 1}$的函数) | $\frac{\partial \vec{x}^T\mathbf{A}}{\partial \vec{x}} = \mathbf{A}$ ($\mathbf{A}_{n\times m}$不是$\vec{x}_{n\times 1}$的函数) |
| $\frac{\partial \vec{x}^T\mathbf{A}\vec{x}}{\partial \vec{x}} = (\mathbf{A}+\mathbf{A}^T)\vec{x}$ ($\mathbf{A}_{n\times n}$不是$\vec{x}_{n\times 1}$的函数) | $\frac{\partial \vec{x}^T\mathbf{A}\vec{x}}{\partial \vec{x}} = (\mathbf{A}+\mathbf{A}^T)\vec{x}$ ($\mathbf{A}_{n\times n}$不是$\vec{x}_{n\times 1}$的函数) |
| | |
| | |
| | |

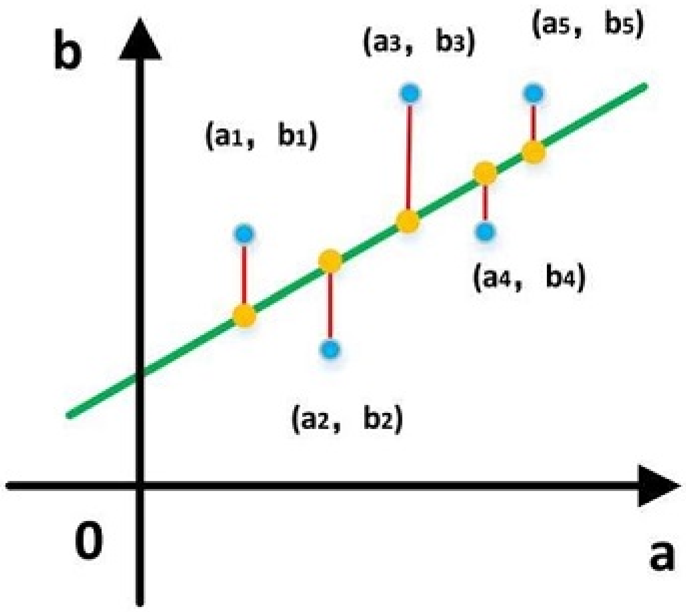
## 二、线性回归(Linear Regression)

### 1. 模型表示

假设函数

$h_\theta(x) = \theta_0 + \theta_1 x$ 单个特征

$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_n x_n$ 多个特征



### 2. 代价函数(Cost function)

$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{n} (h_\theta(x^{(i)})^2 - y^{(i)})^2$ 最小二乘代价函数

- 为什么线性回归使用最小二乘代价函数$J(\theta_0, \theta_1)$?

### 3. 最小二乘法(LSE)

$\hat{y} = \hat{a} + \hat{b}x$

$$\hat{b} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^{n}(x_i - \bar{x})^2} = \frac{\sum_{i=1}^{n} x_i y_i - n\bar{x}\bar{y}}{\sum_{i=1}^{n} x_i^2 - n\bar{x}^2}$$

$\hat{a} = \bar{y} - \hat{b}\bar{x}$

其中$\bar{x}$、$\bar{y}$为$x_i$、$y_i$的均值

- 求回归直线方程的推导过程
- 最小二乘法？为神马不是差的绝对值
- 求解全局最优
- 数据量很大时，计算量大

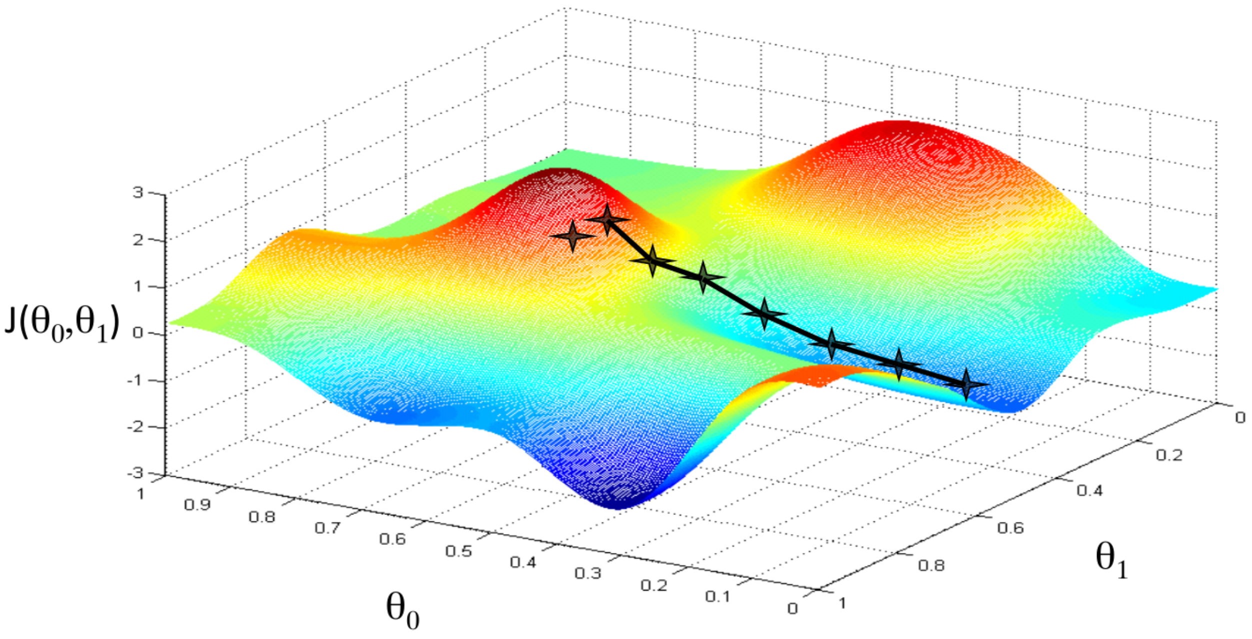
## 4. 梯度下降(Gradient Descent)

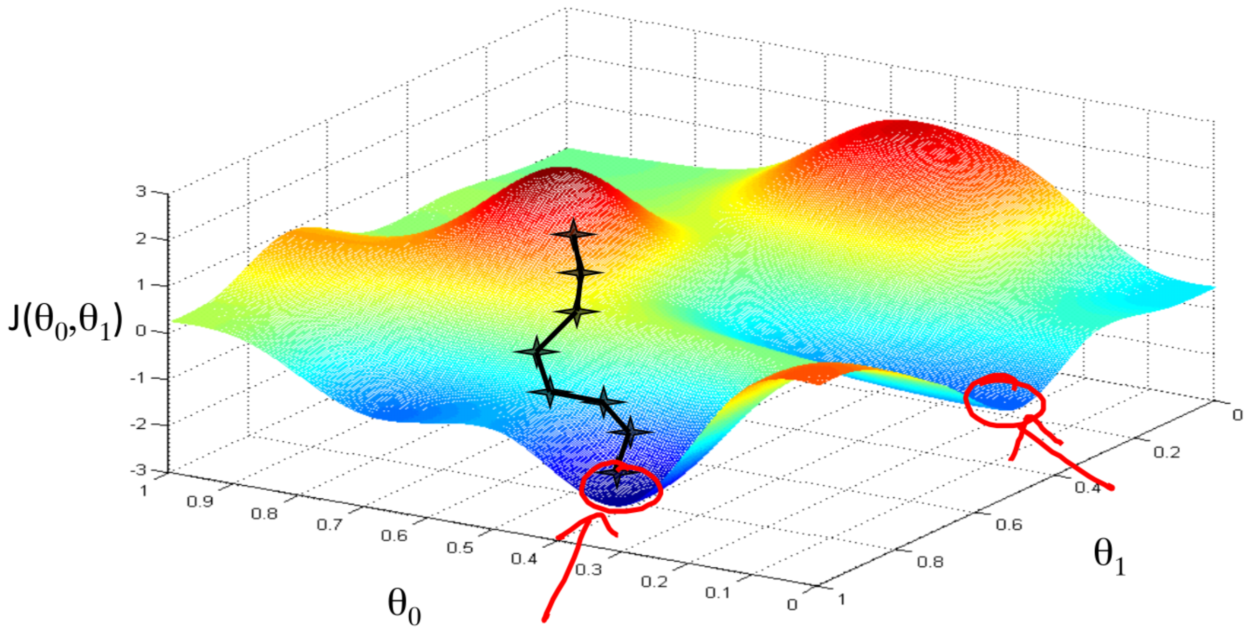
- 目标：$\underset{\theta_0,\theta_1}{minimize}\, J(\theta_0,\theta_1)$
- 迭代法
  - $\theta_j := \theta_j - \alpha\frac{\partial}{\partial\theta_j}J(\theta_0,\theta_1)$ α为学习率

$$\xrightarrow{\text{线性回归}}\begin{cases}\theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}-y^{(i)}))\\ \theta_1 := \theta_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}-y^{(i)}))\cdot x^{(i)}\end{cases}$$

同时更新θ₀和θ₁

- 每一步更新未知量，逐渐逼近局部最优解





- 如何确保梯度下降能正常工作
  - J(θ)应随着迭代不断减小

    - 使用较小的学习率$\alpha$
- 如何选取学习率$\alpha$
    - 尝试：

      ..., 0.001, 0.003, 0.01, 0.03, 0.1, 0.3, 1, ...
    - 取值范围

      $0 < \alpha < 2(|X^T X|^{-1})$
- 分类
    - Batch Gradient Descent

        每一步都使用**整个**训练集的数据计算梯度
    - Stochastic Gradient Descent

      $\Downarrow$ 每一步仅用训练集中的**单个**计算梯度
    - Mini-batch Gradient Descent

        每一步仅用训练集中的**一小部分**（32/64/128/256）计算梯度

      > The way to go is therefore to use some acceptable (but not necessarily optimal) values for the other hyper-parameters, and then trial a number of different mini-batch sizes, scaling α as above. Plot the validation accuracy versus time (as in, real elapsed time, not epoch!), and choose whichever mini-batch size gives you the most rapid improvement in performance. With the mini-batch size chosen you can then proceed to optimize the other hyper-parameters.

        - 适用于存在较多局部最优解的情况，能跳出局部最优解
        - 计算量更小，计算速度更快
        - BGD与SGD的不同

      > The applicability of batch or stochastic gradient descent really depends on the error manifold expected.
      >
      > Batch gradient descent computes the gradient using the whole dataset. This is great for convex, or relatively smooth error manifolds. In this case, we move somewhat directly towards an optimum solution, either local or global. Additionally, batch gradient descent, given an annealed learning rate, will eventually find the minimum located in it's basin of attraction.
      >
      > Stochastic gradient descent (SGD) computes the gradient using a single sample. Most applications of SGD actually use a minibatch of several samples, for reasons that will be explained a bit later. SGD works well (Not well, I suppose, but better than batch gradient descent) for error manifolds that have lots of local maxima/minima. In this case, the somewhat noisier gradient calculated using the reduced number of samples tends to jerk the model out of local minima into a region that hopefully is more optimal. Single samples are really noisy, while minibatches tend to average a little of the noise out. Thus, the amount of jerk is reduced when using minibatches. A good balance is struck when the minibatch size is small enough to avoid some of the poor local minima, but large enough that it doesn't avoid the global minima or better-performing local minima. (Incidently, this assumes that the best minima have a larger and deeper basin of attraction, and are therefore easier to fall into.)

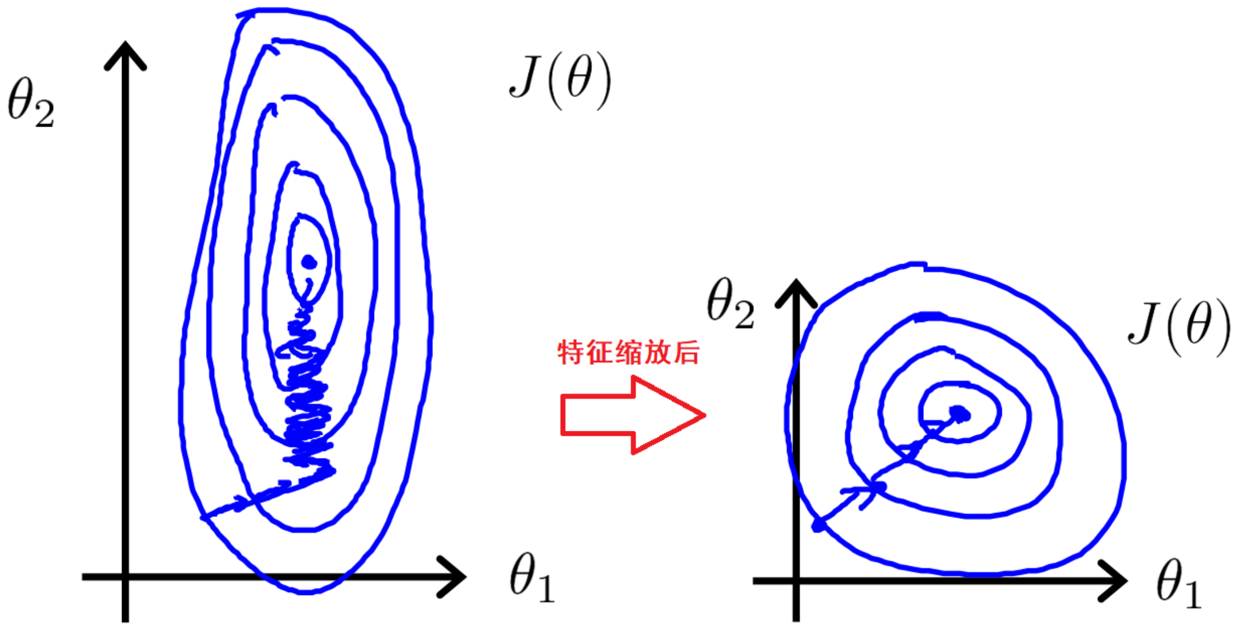
> One benefit of SGD is that it's computationally a whole lot faster. Large datasets often can't be held in RAM, which makes vectorization much less efficient. Rather, each sample or batch of samples must be loaded, worked with, the results stored, and so on. Minibatch SGD, on the other hand, is usually intentionally made small enough to be computationally tractable.
>
> Usually, this computational advantage is leveraged by performing many more iterations of SGD, making many more steps than conventional batch gradient descent. This usually results in a model that is very close to that which would be found via batch gradient descent, or better.
>
> The way I like to think of how SGD works is to imagine that I have one point that represents my input distribution. My model is attempting to learn that input distribution. Surrounding the input distribution is a shaded area that represents the input distributions of all of the possible minibatches I could sample. It's usually a fair assumption that the minibatch input distributions are close in proximity to the true input distribution. Batch gradient descent, at all steps, takes the steepest route to reach the true input distribution. SGD, on the other hand, chooses a random point within the shaded area, and takes the steepest route towards this point. At each iteration, though, it chooses a new point. The average of all of these steps will approximate the true input distribution, usually quite well.

## 5. 向量化

| 变量 | 向量表示（其中$m$为训练样本数，$n$为特征数） |
|---|---|
| feature | $X=\begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_n^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{pmatrix}_{m\times(n+1)}$ 人为地添加一列全1，即$x_0$ |
| output | $y=\begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(m)} \end{pmatrix}_{m\times 1}$ |
| θ | $\theta=\begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \dots \\ \theta_n \end{pmatrix}_{(n+1)\times 1}$ |
| hypothesis | $h=X\theta=\begin{pmatrix} \sum_{i=0}^{n}\theta_i x_i^{(1)} \\ \sum_{i=0}^{n}\theta_i x_i^{(2)} \\ \dots \\ \sum_{i=0}^{n}\theta_i x_i^{(m)} \end{pmatrix}_{m\times 1}=\begin{pmatrix} \hat{y}^{(1)} \\ \hat{y}(2) \\ \dots \\ \hat{y}(m) \end{pmatrix}_{m\times 1}$ |
| cost function | $J=\frac{1}{2m}(h-y)^T(h-y)$ |
| Gradient descent | $\theta:=\theta-\frac{\alpha}{m}X^T(h-y)$ |

## 6. 特征缩放(Feature Scaling)

- 确保所有特征在相似的规模上（$-1\le x_i \le 1$）
- 提升梯度下降速度



- 方法
  - **mean normalization**

    $$x_i' = \frac{x_i - \mu_i}{max(x) - min(x)}$$

    其中$\mu_i$为$x$的均值
  - **Standardization**

    $$x_i' = \frac{x - \bar{x}}{\sigma}$$

    其中$\sigma$为$x$的标准差

## 6. 正规方程(Normal Equation)

- 公式

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

  推导过程
- 直接求解参数$\theta$的最优值，不需要多次迭代 `解析方法`
- 不需要特征缩放
- 不需要选择学习率$\alpha$

> `问题`
>
> A. 特征数很大时计算量大（n≥10000） `求解逆矩阵的时间复杂度O(n³)`
>
> B. 只适用于线性模型
>
> C. $X^T X$可能不可逆
>
> - 冗余特征：存在线性相关的特征
> - 太多特征：特征数>样本数，导致过拟合
>
> $$\xrightarrow{处理方法} \begin{cases} 删除部分特征 \begin{cases} 手动选择 \\ 模型选择算法 \end{cases} \\ 正则化(\text{Regularization}) \end{cases}$$

## 7. 正则化(Normalization)

保留所有特征，但减小参数$\theta_j$的量级/值

引入正则化参数λ

特征较多时效果较好

- 公式

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

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

$$\Longrightarrow \begin{cases} \theta_0 := \theta_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)} - y^{(i)})) \\ \theta_j := \theta_j - \alpha[\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)} - y^{(i)}))\cdot x_j^{(i)} + \frac{\lambda}{m}\theta_j] \qquad (j = 1, 2, 3, \dots, n) \end{cases}$$

同时更新所有θ

$$\Downarrow$$

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

[注] $1 - \alpha\frac{\lambda}{m} < 1$

正规方程

$$\theta = (X^TX + \lambda\begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix}_{(n+1)\times(n+1)} )^{-1}X^Ty$$

[注] 当$\lambda > 0$时，可以证明$X^TX + \lambda\begin{pmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & 1 \end{pmatrix}$是可逆的

> 建议一开始将正则项系数λ设置为0，先确定一个比较好的learning rate。然后固定该learning rate，给λ一个值（比如1.0），然后根据validation accuracy，将λ增大或者减小10倍（增减10倍是粗调节，当确定了λ的合适的数量级后，比如λ = 0.01,再进一步地细调节，比如调节为0.02，0.03，0.009之类。）

## 8. 牛顿法

-

## 9. 最速下降法

-

## 10. 局部加权线性回归 非参数学习方法

I Fit $\theta$ to minimize $\sum_i w^{(i)}(y^{(i)} - \theta^T x^{(i)})^2$

II Output $\theta^T x$

$w^{(i)} = e^{-\frac{(x^{(i)}-x)^2}{2\tau^2}}$，其中τ为常数，可调节$w^{(i)}$钟形曲线的宽度

- 当$|x^{(i)} - x| \approx 0$时，$w^{(i)} \approx 1$，即预测样本举例训练样本越近，权值越大
- 当$|x^{(i)} - x| \approx +\infty$时，$w^{(i)} \approx 0$，即预测样本举例训练样本越远，权值越小

> 问题
>
> A. 当数据规模比较大时，计算量很大，学习效率很低
>
> B. 不一定能避免欠拟合

# 三、逻辑回归(Logistic Regression)

## 1. 模型表示

> `分类`
>
> $y = 0 \text{ or } 1$
>
> $h_\theta(x) \text{ can be } > 1 \text{ or } < 0$
>
> ⇓ 逻辑函数
>
> `逻辑回归`
>
> $0 \le h_\theta(x) \le 1$

假设函数$h_\theta(x) = g(\theta^T x) = \dfrac{1}{1+e^{-\theta^T x}}$ `在线性回归模型的基础上加了一个Sigmoid函数`

其中$g(z) = \dfrac{1}{1+e^{-z}}$ `sigmoid函数（S型曲线）`

$$g'(z) = \frac{e^{(-z)}}{(1+e^{(-z)})^2} = \frac{1}{1+e^{(-z)}}(1-\frac{1}{1+e^{(-z)}}) = g(z)(1-g(z))$$

## 2. 代价函数(Cost function)

> 线性回归的代价函数在逻辑回归模型中是非凸函数("'non-convex'")，不容易收敛到全局最优点，故引入新的代价函数

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

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

## 3. 梯度下降(Gradient Descent)

- 目标：$\underset{\theta}{minimize}\, J(\theta)$
- 公式

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

- 优化算法

  `不需要手动选择学习率α`

  `通常比梯度下降更快`

  `但算法更复杂`

  - Conjugate gradient
  - BFGS

  - L-BFGS

## 4. 向量化

| 变量 | 向量表示（其中$m$为训练样本数，$n$为特征数） |
|---|---|
| feature | $X = \begin{pmatrix} 1 & x_1^{(1)} & x_2^{(1)} & \dots & x_n^{(1)} \\ 1 & x_1^{(2)} & x_2^{(2)} & \dots & x_n^{(2)} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_n^{(m)} & x_2^{(m)} & \dots & x_n^{(m)} \end{pmatrix}_{m\times(n+1)}$ `人为地添加一列全1，即x₀` |
| output | $y = \begin{pmatrix} y^{(1)} \\ y^{(2)} \\ \dots \\ y^{(m)} \end{pmatrix}_{m\times1}$ `每一项取值0或1` |
| θ | $\theta = \begin{pmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \dots \\ \theta_n \end{pmatrix}_{(n+1)\times1}$ |
| hypothesis | $h = g(X\theta) = \begin{pmatrix} g(\sum_{i=0}^{n} \theta_i x_i^{(1)}) \\ g(\sum_{i=0}^{n} \theta_i x_i^{(2)}) \\ \dots \\ g(\sum_{i=0}^{n} \theta_i x_i^{(m)}) \end{pmatrix}_{m\times1} = \begin{pmatrix} \hat{y}^{(1)} \\ \hat{y}(2) \\ \dots \\ \hat{y}(m) \end{pmatrix}_{m\times1}$ |
| cost function | $J = -\frac{1}{m}[y^T log(h) + (1-y)^T log(1-h)]$ |
| Gradient descent | $\theta := \theta - \frac{\alpha}{m} X^T(h-y)$ |

## 5. 正则化(Normalization)

保留所有特征，但减小参数$\theta_j$的量级/值

`引入正则化参数λ`

`特征较多时效果较好`

- 公式

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

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

$$\Longrightarrow \begin{cases} \theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)} - y^{(i)})) \\ \theta_j := \theta_j - \alpha[\frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)} - y^{(i)})) \cdot x_j^{(i)} + \frac{\lambda}{m}\theta_j] \qquad (j = 1, 2, 3, \dots, n) \end{cases}$$

`同时更新所有θ`

- 向量化

$J = -\frac{1}{m}[y^T log(h) + (1-y)^T log(1-h)] + \frac{\lambda}{2m}(\theta^T\theta - \theta_0^2)$

$\theta := \theta - \frac{\alpha}{m}[X^T(h-y) + \lambda\theta]$

## * 范数(norm)

- L0范数：向量中非0的元素的个数
  - L0范数可以实现稀疏
  - L1范数：向量中各个元素绝对值之和\quad$ `稀疏规则算子`
  - L1范数会使权值稀疏
  - L2范数：向量各元素的平方和然后求平方根 `权值衰减`
  - L2范数可以防止过拟合，提升模型的泛化能力

  L1正则化：$\lambda\|\theta\|$ `套索回归Lasso`

  L2正则化：$\lambda\|\theta\|^2$ `岭回归Ridge`

## 6. 多元分类

> $y \in 0, 1, 2, \dots, n$
>
> $$\begin{cases} h_\theta^{(0)}(x) = P(y=0|x;\theta) \\ h_\theta^{(1)}(x) = P(y=1|x;\theta) \\ \dots \\ h_\theta^{(n)}(x) = P(y=n|x;\theta) \end{cases}$$

$\text{prediction} = \max(h_\theta^{(i)}(x))$

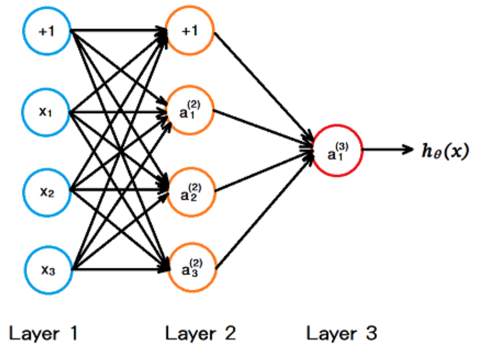
# 四、神经网络

## 1. 模型表示

- $L=$ 网络总层数 $=$ 输入层 $+$ 隐藏层 $+$ 输出层
- $s_l=$ 第$l$层中的单元数（不包含偏置单元）
- $K=$ 输出单元个数/类别数 $\overset{\text{二分类}}{\Longrightarrow}$ $K=1$
- $(h_\Theta(x^{(i)}))_k=$ 第$k$个输出
- $a^{(j)}=$ 第$j$层的激励(activation) $\overset{\text{维度}}{\Longrightarrow}$ $(s_j+1)\times 1$

  $a_i^{(j)}=$ 第$j$层第$i$个单元的激励
- $\Theta^{(j)}=$ 从第$j$层映射到第$j+1$层的权重矩阵 $\overset{\text{维度}}{\Longrightarrow}$ $s_{j+1}\times(s_j+1)$

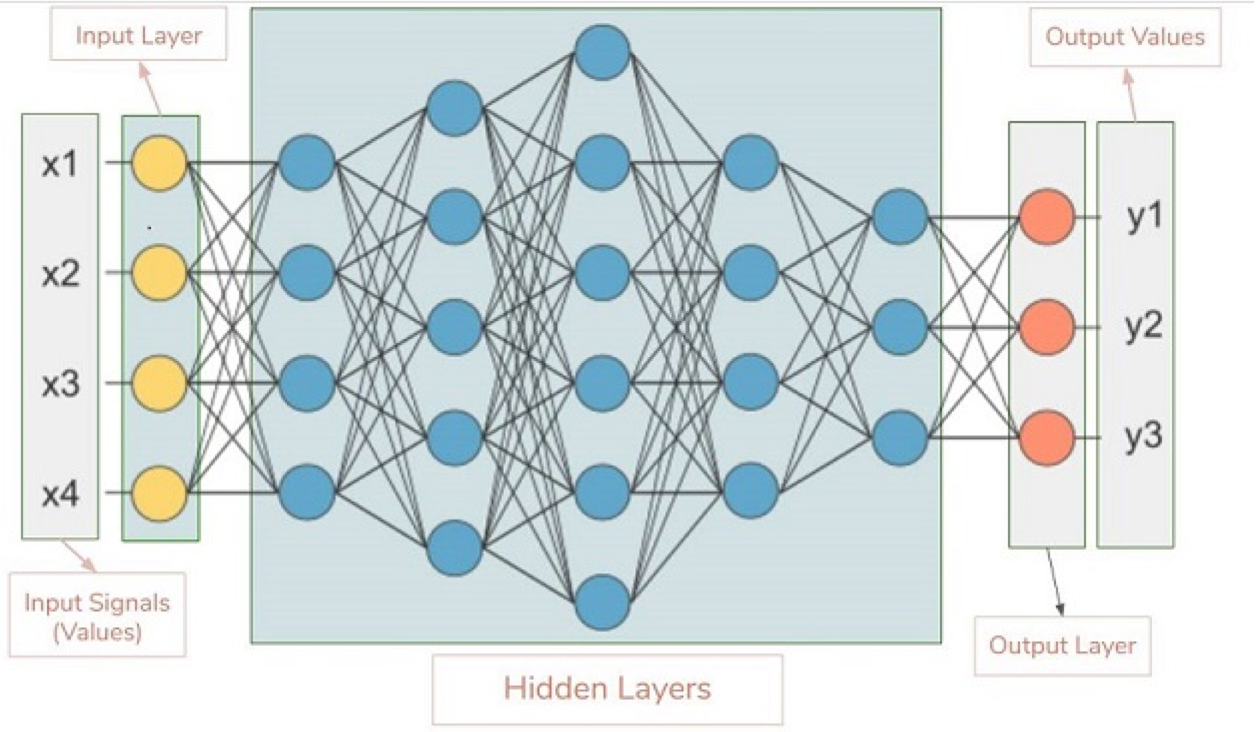
  $\Theta^{(j)}_{kn}=$ 从第$j$层映射到第$j+1$层第$k$个单元的第$n$个参数(权重)



$$a_1^{(2)}=g(\theta_{10}^{(1)}x_0+\theta_{11}^{(1)}x_1+\theta_{12}^{(1)}x_2+\theta_{13}^{(1)}x_3)$$
$$a_2^{(2)}=g(\theta_{20}^{(1)}x_0+\theta_{21}^{(1)}x_1+\theta_{22}^{(1)}x_2+\theta_{23}^{(1)}x_3)$$
$$a_3^{(2)}=g(\theta_{30}^{(1)}x_0+\theta_{31}^{(1)}x_1+\theta_{32}^{(1)}x_2+\theta_{33}^{(1)}x_3)$$
$$h_\theta(x)=a_1^{(3)}=g(\theta_{10}^{(2)}a_0^{(2)}+\theta_{11}^{(2)}a_1^{(2)}+\theta_{12}^{(2)}a_2^{(2)}+\theta_{13}^{(2)}a_3^{(2)})$$

### 网络架构

- 输入层→隐藏层→输出层

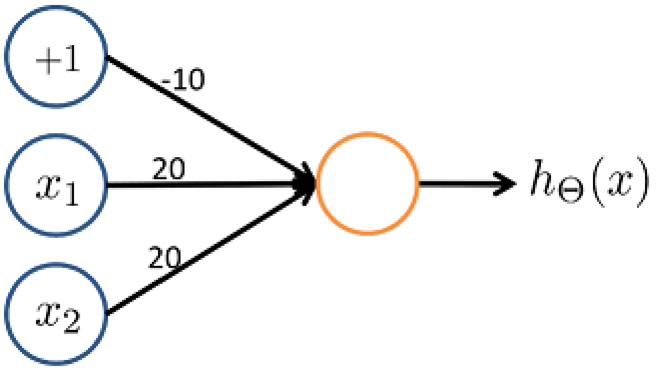


## 2. 非线性分类：逻辑运算

`激活函数取sigmoid函数`

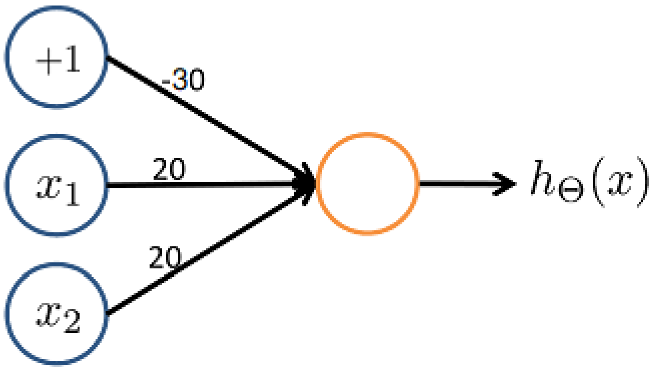
- or运算

$$h_\Theta(x) = g(-10 + 20x_1 + 20x_2) \Longrightarrow \begin{cases} \approx 1 & x_1 = 1, x_2 = 1 \\ \approx 1 & x_1 = 0, x_2 = 1 \\ \approx 1 & x_1 = 1, x_2 = 0 \\ \approx 0 & x_1 = 0, x_2 = 0 \end{cases}$$



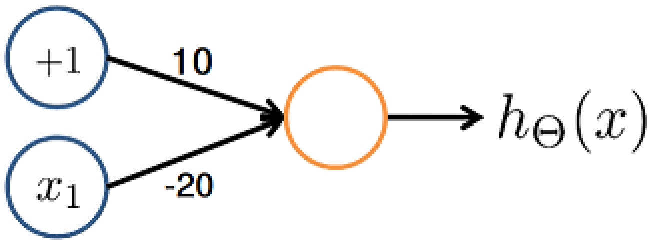
- and运算

$$h_\Theta(x) = g(-30 + 20x_1 + 20x_2) \Longrightarrow \begin{cases} \approx 1 & x_1 = 1, x_2 = 1 \\ \approx 0 & x_1 = 0, x_2 = 1 \\ \approx 0 & x_1 = 1, x_2 = 0 \\ \approx 0 & x_1 = 0, x_2 = 0 \end{cases}$$



- not运算

$$h_\Theta(x) = g(10 - 20x) \Longrightarrow \begin{cases} \approx 1 & x = 0 \\ \approx 0 & x = 1 \end{cases}$$

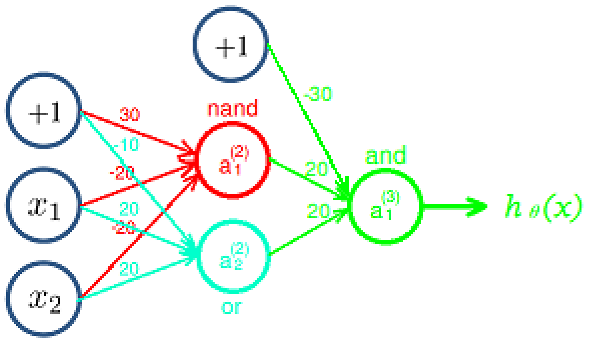


- xor运算

$$h_\Theta(x) = a_1^{(3)} = g(-30 + 20a_1^{(2)} + 20a_2^{(2)}) \Longrightarrow \begin{cases} \approx 0 & x_1 = 1, x_2 = 1 \\ \approx 1 & x_1 = 0, x_2 = 1 \\ \approx 1 & x_1 = 1, x_2 = 0 \\ \approx 0 & x_1 = 0, x_2 = 0 \end{cases}$$

$$a_1^{(2)} = g(30 - 20x_1 - 20x_2) \Longrightarrow \begin{cases} \approx 0 & x_1 = 1, x_2 = 1 \\ \approx 1 & x_1 = 0, x_2 = 1 \\ \approx 1 & x_1 = 1, x_2 = 0 \\ \approx 1 & x_1 = 0, x_2 = 0 \end{cases}$$

$$a_2^{(2)} = g(-10 + 20x_1 + 20x_2) \Longrightarrow \begin{cases} \approx 1 & x_1 = 1, x_2 = 1 \\ \approx 1 & x_1 = 0, x_2 = 1 \\ \approx 1 & x_1 = 1, x_2 = 0 \\ \approx 0 & x_1 = 0, x_2 = 0 \end{cases}$$

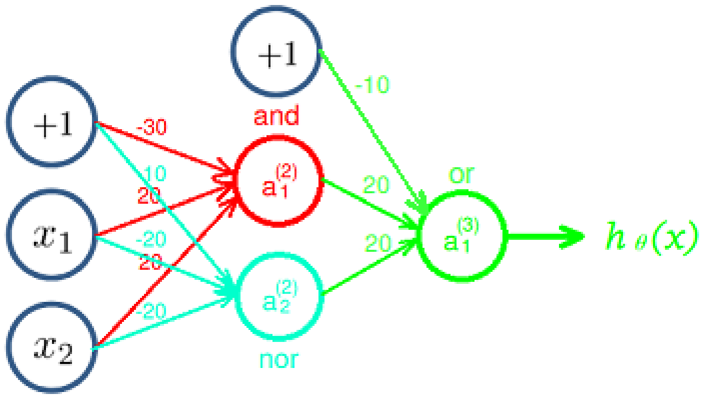


- xnor运算

$$h_\Theta(x) = a_1^{(3)} = g(-10 + 20a_1^{(2)} + 20a_2^{(2)}) \Longrightarrow \begin{cases} \approx 1 & x_1 = 1, x_2 = 1 \\ \approx 0 & x_1 = 0, x_2 = 1 \\ \approx 0 & x_1 = 1, x_2 = 0 \\ \approx 1 & x_1 = 0, x_2 = 0 \end{cases}$$

$$a_1^{(2)} = g(-30 + 20x_1 + 20x_2) \Longrightarrow \begin{cases} \approx 1 & x_1 = 1, x_2 = 1 \\ \approx 0 & x_1 = 0, x_2 = 1 \\ \approx 0 & x_1 = 1, x_2 = 0 \\ \approx 0 & x_1 = 0, x_2 = 0 \end{cases}$$

$$a_2^{(2)} = g(10 - 20x_1 - 20x_2) \Longrightarrow \begin{cases} \approx 0 & x_1 = 1, x_2 = 1 \\ \approx 0 & x_1 = 0, x_2 = 1 \\ \approx 0 & x_1 = 1, x_2 = 0 \\ \approx 1 & x_1 = 0, x_2 = 0 \end{cases}$$



## 3. 分类

· 二分类

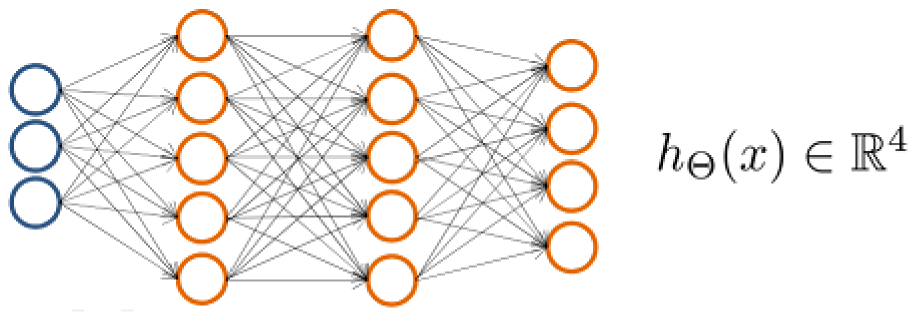
$y = 0 \ or \ 1$

· 感知机(perception)

假设空间是特征空间中的所有线性分类模型，即找到一个线性方程$\boldsymbol{x} \cdot \boldsymbol{w} + b = 0$，它对应于特征空间的一个超平面，能够把对应的特征空间分为两部分，位于两部分中的点分别是正负两类。

· 多元分类

$$y^{(i)} \in one \ of \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$



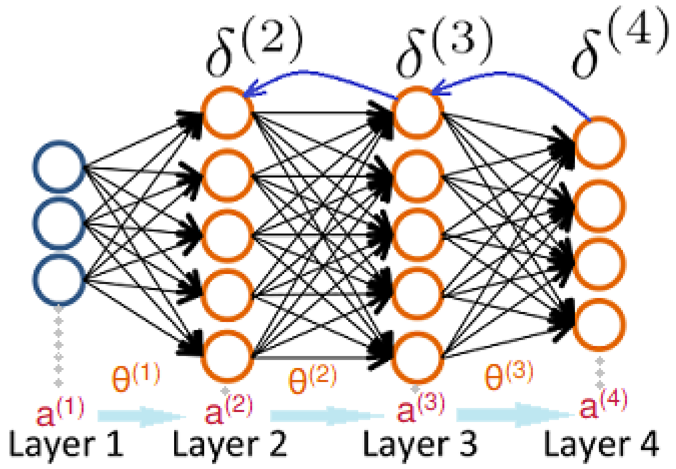
## 4. 代价函数(Cost function)

逻辑回归中代价函数的一般形式

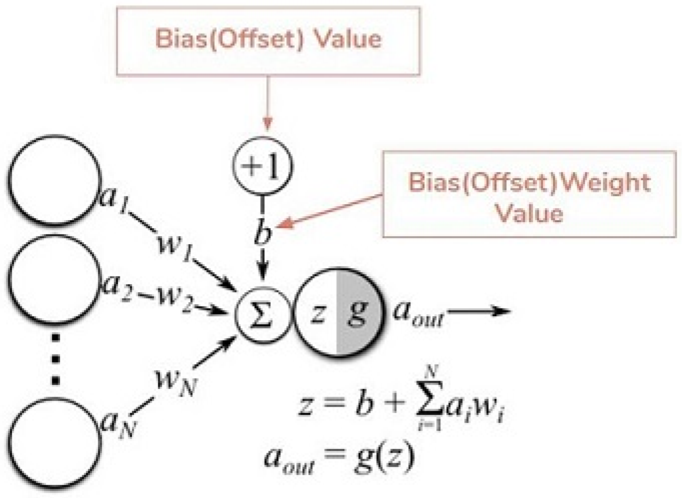
$$J(\Theta) = -\frac{1}{m}[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} log(h_\Theta(x^{(i)}))_k + (1 - y_k^{(i)}) log(1 - h_\Theta(x^{(i)}))_k] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{j+1}}(\Theta_{ji}^{(l)})^2$$

## 5. 反向传播算法(Backpropagation algorithm)

梯度下降 迭代法



- I 前向传播 计算各层输出



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

- II　反向传播　`误差逆传播`

> 若输入层的实际输出$h_\Theta(x)$与期望的输出$y$不符，则进行 误差的反向传播
>
> 直到网络输出的误差减少到了可以接受的 程度（或 进行到预先设定的学习次数为止）

`偏置单元可以计入δ，也可不计入`

$$\nabla_{\Theta^{(l)}} J(\Theta) = \frac{\partial J(\Theta)}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial \Theta^{(l)}} = \delta^{(l+1)} (a^{(l)})^T$$

$$\delta^{(l)} = \frac{\partial J}{\partial z^{(l)}} = \frac{\partial J}{\partial z^{(l+1)}} \frac{\partial z^{(l+1)}}{\partial a^{(l)}} \frac{\partial a^{(l)}}{\partial z^{(l)}} = (\Theta^{(l)})^T \delta^{(l+1)} \cdot g'(z^{(l)}) = (\Theta^{(l)})^T \delta^{(l+1)} \cdot a^{(l)} (1 - a^{(l)})$$

$$\delta_j^{(l)} = \text{第}\, l \text{层第}\, j \text{个单元的误差} = \begin{cases} a_j^{(l)} - y_j = (h_\Theta(x))_j - y_j & \text{输出层} \quad (l = L) \\ a_j^{(l)} \cdot (1 - a_j^{(l)}) & \text{隐藏层} \quad (1 < l < L) \end{cases}$$

$$\xrightarrow{\text{忽略正则项}\lambda} \frac{\partial}{\partial \Theta_{ji}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\xrightarrow{\text{二分类}} \begin{cases} cost(t) = y^{(t)} \ \log(h_\Theta(x^{(t)})) + (1 - y^{(t)}) \ \log(1 - h_\Theta(x^{(t)})) \\ \delta_j^{(l)} = \dfrac{\partial}{\partial z_j^{(l)}} cost(t) \\ \dfrac{\partial}{\partial \Theta_{ji}^{(l)}} J(\Theta) = a_j^{(l)} \delta_i^{(l+1)} \end{cases}$$

## 6. 向量化

| 变量 | 向量表示（其中$m$为训练样本数，$K$为类别数，$L$为网络层数，$s_l$为第$l$层单元数） |
|---|---|
| feature | $x^{(i)} = \begin{pmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \dots \\ x_{s_L}^{(i)} \end{pmatrix}_{s_L \times 1} \qquad (1 \le i \le m)$ |
| output | $y^{(i)} = \begin{pmatrix} y_1^{(i)} \\ y_2^{(i)} \\ \dots \\ y_{s_L}^{(i)} \end{pmatrix}_{s_L \times 1} \qquad (1 \le i \le m)$ `每一项取值0或1` |
| $\Theta^{(l)}$ | $\Theta^{(l)} = \begin{pmatrix} \theta_{10}^{(l)} & \theta_{11}^{(l)} & \theta_{12}^{(l)} & \dots & \theta_{1s_l}^{(l)} \\ \theta_{20}^{(l)} & \theta_{21}^{(1)} & \theta_{22}^{(l)} & \dots & \theta_{2s_l}^{(l)} \\ \theta_{30}^{(l)} & \theta_{31}^{(l)} & \theta_{32}^{(l)} & \dots & \theta_{3s_l}^{(l)} \\ \dots & \dots & \dots & \dots & \dots \\ \theta_{s_{l+1}0}^{(l)} & \theta_{s_{l+1}1}^{(l)} & \theta_{s_{l+1}2}^{(l)} & \dots & \theta_{s_{l+1}s_l}^{(l)} \end{pmatrix}_{s_{l+1} \times (s_l+1)} = \begin{pmatrix} (\vec{\theta}_1^{(l)})^T \\ (\vec{\theta}_2^{(l)})^T \\ (\vec{\theta}_3^{(l)})^T \\ \dots \\ (\vec{\theta}_{s_{l+1}}^{(l)})^T \end{pmatrix}$ |
| activation | $a^{(1)} = x \qquad (l = 1)$ <br> $a^{(l)} = g(\Theta^{(l-1)} a^{(l-1)}) = \begin{pmatrix} 1 \\ g(\sum_{k=0}^{K} \Theta_{1k}^{(l-1)} a_k^{(l-1)}) \\ g(\sum_{k=0}^{K} \Theta_{2k}^{(l-1)} a_k^{(l-1)}) \\ \dots \\ g(\sum_{k=0}^{K} \Theta_{s_l k}^{(l-1)} a_k^{(l-1)}) \end{pmatrix}_{(s_l+1) \times 1} = \begin{pmatrix} a_0^{(l)} \\ a_1^{(l)} \\ a_2^{(l)} \\ \dots \\ a_{s_l}^{(l)} \end{pmatrix} \qquad (2 \le l \le L)$ |
| error | $\delta^{(L)} = a^{(L)} - y = h_\Theta(x) - y \qquad (l = L)$ <br> $\delta^{(l)} = ((\Theta^{(l)})^T \delta^{(l+1)}) .* g'(z^{(l)}) = ((\Theta^{(l)})^T \delta^{(l+1)}) .* a^{(l)} .* (1 - a^{(l)})$ <br> $= \begin{pmatrix} \sum_{i=1}^{s_{l+1}} \Theta_{i1}^{(l)} \delta_i^{(i+1)} \cdot a_1^{(l)} \cdot (1 - a_1^{(l)}) \\ \sum_{i=1}^{s_{l+1}} \Theta_{i2}^{(l)} \delta_i^{(i+1)} \cdot a_2^{(l)} \cdot (1 - a_2^{(l)}) \\ \sum_{i=1}^{s_{l+1}} \Theta_{i3}^{(l)} \delta_i^{(i+1)} \cdot a_3^{(l)} \cdot (1 - a_3^{(l)}) \\ \dots \\ \sum_{i=1}^{s_{l+1}} \Theta_{is_l}^{(l)} \delta_i^{(i+1)} \cdot a_{s_l}{}^{(l)} \cdot (1 - a_{s_l}^{(l)}) \end{pmatrix}_{s_l \times 1} \qquad (2 \le l \le L-1)$ <br> (未计入偏置单元，故公式中$\Theta^{(l)}$的维度应为$s_{l+1} \times s_l$，$a^{(l)}$的维度应为$s_l \times 1$) |

| 变量 | 向量表示（其中$m$为训练样本数，$K$为类别数，$L$为网络层数，$s_l$为第$l$层单元数） |
|---|---|
| cost function(例) | $J = -\frac{1}{m}[y^T log(h) + (1-y)^T log(1-h)]$ |
| Gradient descent | $\theta := \theta - \frac{\alpha}{m}X^T(h-y)$ |

## 7. 梯度检查

Suppose you are using gradient descent together with backpropagation to try to minimize $J(\Theta)$ as a function of $\Theta$. Which of the following would be a useful step for verifying that the learning algorithm is running correctly?

Plot $J(\Theta)$ as a function of $\Theta$, to make sure gradient descent is going downhill.

Plot $J(\Theta)$ as a function of the number of iterations and make sure it is increasing (or at least non-decreasing) with every iteration.

[✓ Plot $J(\Theta)$ as a function of the number of iterations and make sure it is decreasing (or at least non-increasing) with every iteration.]

正确

Plot $J(\Theta)$ as a function of the number of iterations to make sure the parameter values are improving in classification accuracy.

## 8. 随机初始化

## 9. 训练神经网络的步骤

1. 选择一个网络架构 `神经元之间的连接样式`
    - 输入单元的数量：特征$x^{(i)}$的维度
    - 输出单元的数量：分类数

    > 合理的默认选择：1个隐藏层，或若有多个隐藏层，则在每一层中有相同数量的隐藏单元（通常越多越好）

2. 训练过程
    1. 随机初始化权重(参数)
    2. 利用正向传播方法计算所有输入$x^{(i)}$的$h_\Theta(x^{(i)})$
    3. 编写计算代价函数$J(\Theta)$ 的代码
    4. 利用反向传播方法计算偏导数$\frac{\partial}{\partial\Theta_{jk}^{(l)}}J(\Theta)$
    5. 利用数值检验方法比较用反向传播计算的和用$J(\Theta)$的梯度的数值估计得到的$\frac{\partial}{\partial\Theta_{jk}^{(l)}}J(\Theta)$

> > 然后禁用梯度检查的代码，因为梯度检查速度很慢

6. 使用梯度下降或优化算法与反向传播算法相结合，来使$J(\Theta)$最小化

> $J(\Theta)$是非凸函数，因此梯度下降可能得到一个局部最优点

## * Geoffrey Hinton关于BP的看法