# 机器学习算法总结3

## 11 EM算法

参考自

- 《统计学习方法》
- 机器学习常见算法个人总结（面试用）
- 从最大似然到EM算法浅解
- （EM算法）The EM Algorithm

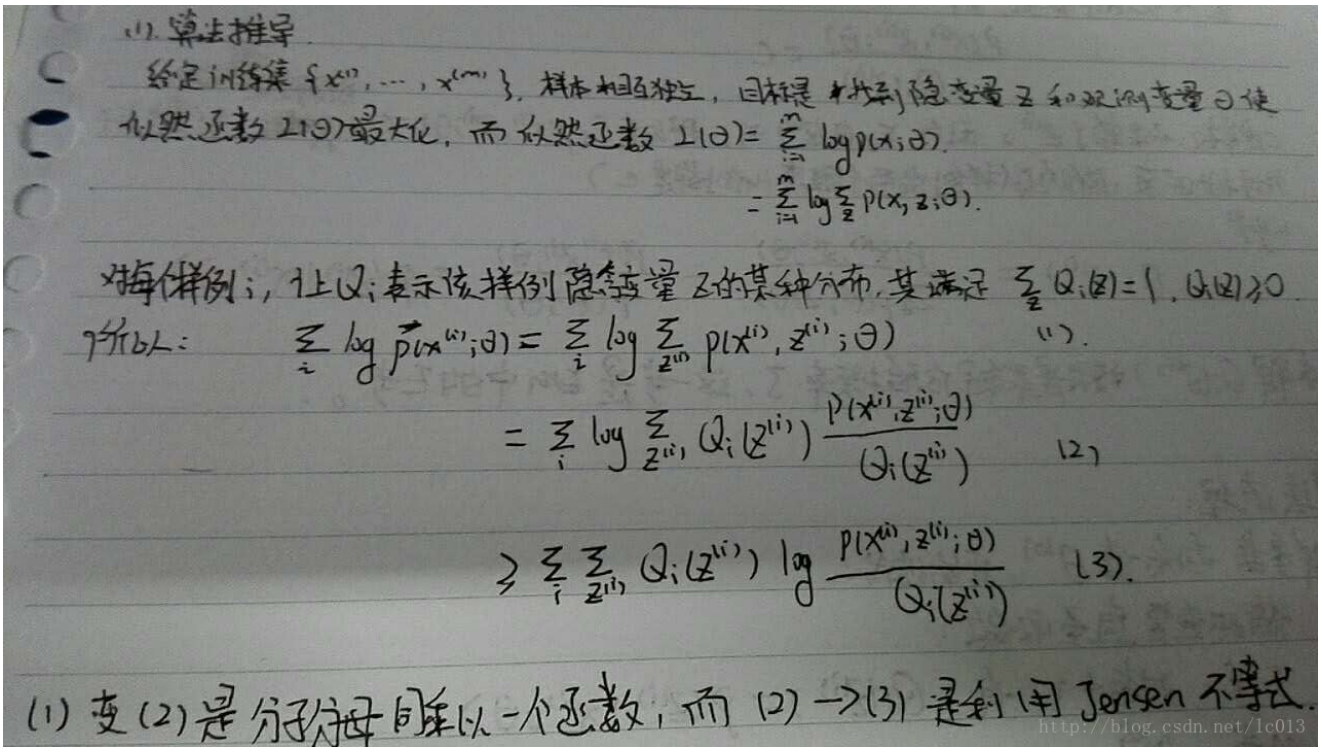
### 简介

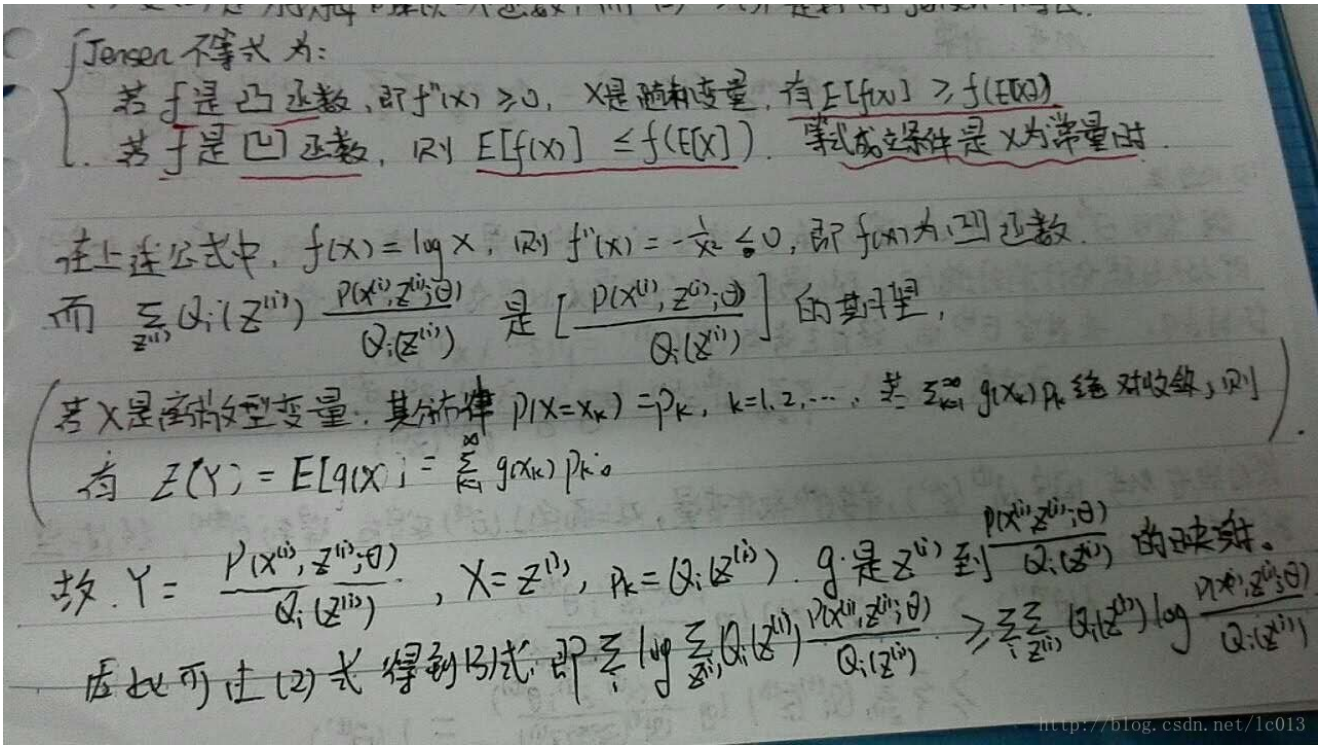
> EM算法，即期望极大算法，用于含有隐变量的概率模型的极大似然估计或极大后验概率估计，它一般分为两步：第一步求期望**(E)**,第二步求极大**(M)**。

如果概率模型的变量都是观测变量，那么给定数据之后就可以直接使用极大似然法或者贝叶斯估计模型参数。
但是当模型含有隐含变量的时候就不能简单的用这些方法来估计，EM就是一种含有隐含变量的概率模型参数的极大似然估计法。

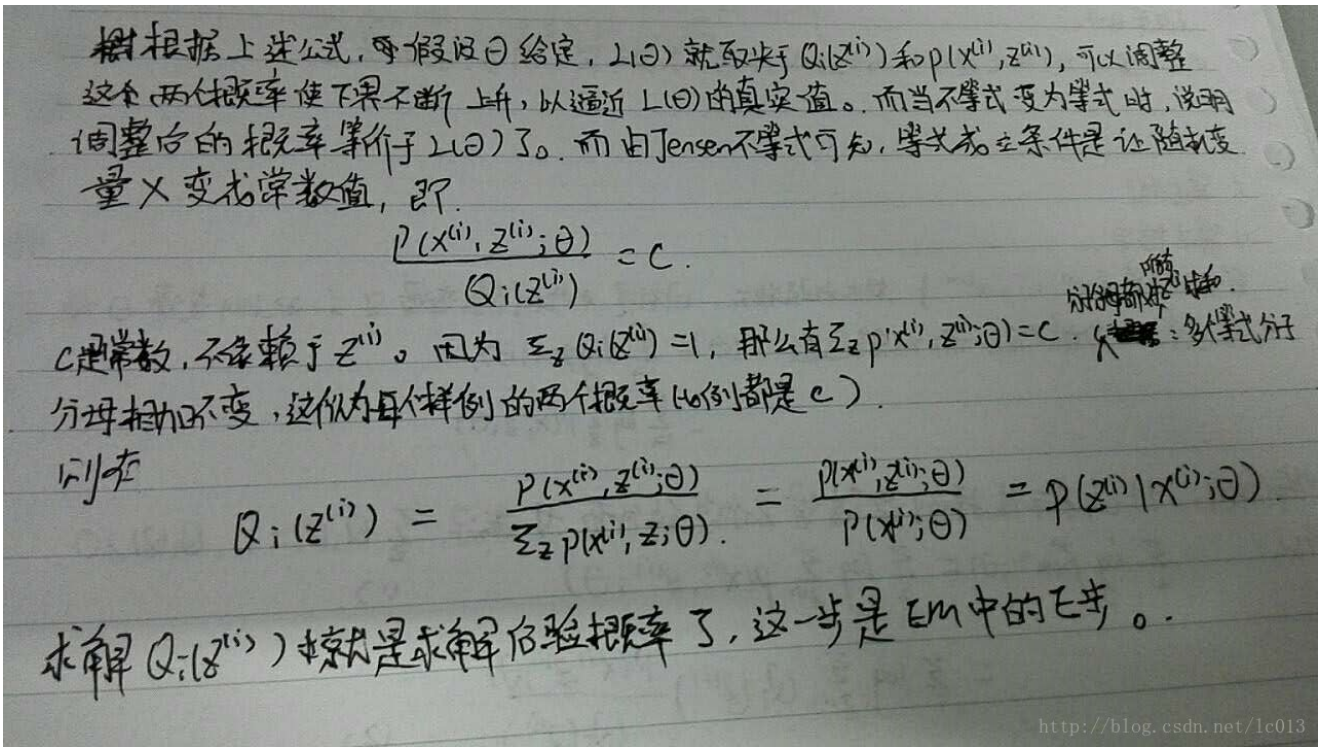
应用到的地方：混合高斯模型、混合朴素贝叶斯模型、因子分析模型

### 算法推导



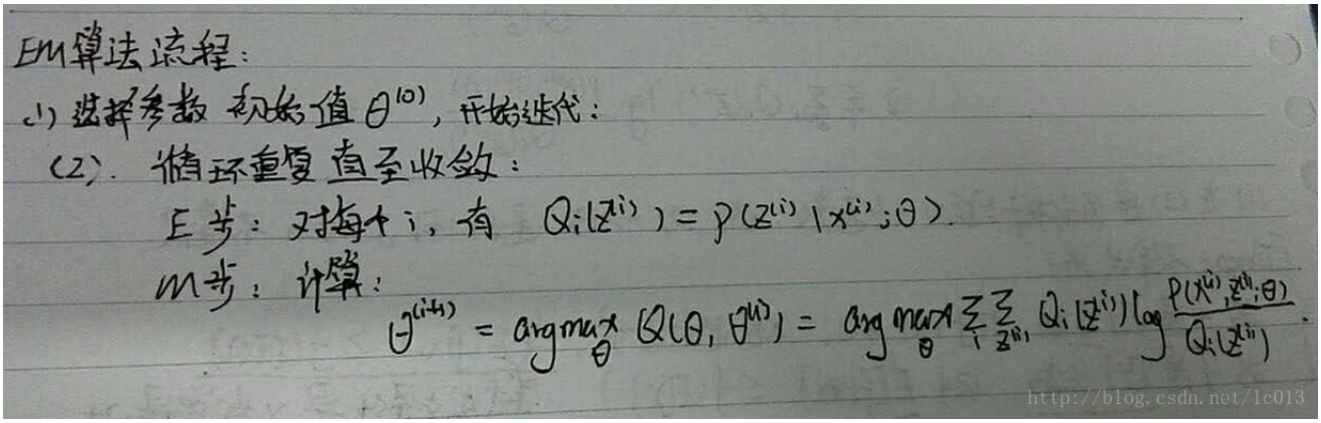



上述公式相当于决定了$L(\theta)$的下界，而**EM**算法实际上就是通过不断求解下界的极大化来逼近对数似然函数极大化的算法。




## 算法流程

算法流程如下所示：




## 收敛性

收敛性部分可以主要看（EM算法）The EM Algorithm的推导，最终可以推导得到如下公式：

$$
\begin{aligned}
L(\theta^{(t+1)}) &\geq \sum_i \sum_{z^i} Q_i^{(t)}(z^{(i)}) log \frac{p(x^{(i)}, z^{(i)}; \theta^{(t+1)})}{Q_i^{(t)}(z^{(i)})} \\
&\geq \sum_i \sum_{z^i} Q_i^{(t)}(z^{(i)}) log \frac{p(x^{(i)}, z^{(i)}; \theta^{(t)})}{Q_i^{(t)}(z^{(i)})} \\
&= L(\theta^{(t)})
\end{aligned}
$$

## 特点

1. 最大优点是简单性和普适性
2. **EM**算法不能保证找到全局最优点，在应用中，通常选取几个不同的初值进行迭代，然后对得到的几个估计值进行比较，从中选择最好的
3. **EM**算法对初值是敏感的，不同初值会得到不同的参数估计值

## 使用例子

**EM**算法一个常见的例子就是**GMM**模型，即高斯混合模型。而高斯混合模型的定义如下：

> 高斯混合模型是指具有如下形式的概率分布模型：
>
> $$P(y|\theta) = \sum_{k=1}^{K} \alpha_k \phi(y|\theta_k)$$
>
> 其中，$\alpha_k$是系数，$\alpha_k \geq 0, \sum_{k=1}^{K} \alpha_k = 1; \phi(y|\theta_k)$是高斯分布密度，$\theta_k = (\mu_k, \sigma_k^2)$，
>
> $$\phi(y|\theta_k) = \frac{1}{\sqrt{2\pi}\sigma_k} exp(-\frac{(y-\mu_k)^2}{2\sigma_k^2})$$

$\phi(y|\theta_k)$称为第$k$个分模型。

每个样本都有可能由$k$个高斯产生，只不过由每个高斯产生的概率不同而已，因此每个样本都有对应的高斯分布（$k$个中的某一个），此时的隐含变量就是每个样本对应的某个高斯分布。

GMM的E步公式如下（计算每个样本对应每个高斯的概率）：

(E-step) For each $i, j$, set

$$w_j^{(i)} := p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma)$$

更具体的计算公式为：

$$p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma) = \frac{p(x^{(i)} | z^{(i)} = j; \mu, \Sigma) p(z^{(i)} = j; \phi)}{\sum_{l=1}^{k} p(x^{(i)} | z^{(i)} = l; \mu, \Sigma) p(z^{(i)} = l; \phi)}$$

M步公式如下（计算每个高斯的比重，均值，方差这3个参数）：

(M-step) Update the parameters:

$$\begin{aligned}
\phi_j &:= \frac{1}{m} \sum_{i=1}^{m} w_j^{(i)}, \\
\mu_j &:= \frac{\sum_{i=1}^{m} w_j^{(i)} x^{(i)}}{\sum_{i=1}^{m} w_j^{(i)}}, \\
\Sigma_j &:= \frac{\sum_{i=1}^{m} w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^{m} w_j^{(i)}}
\end{aligned}$$



# 12 优化算法

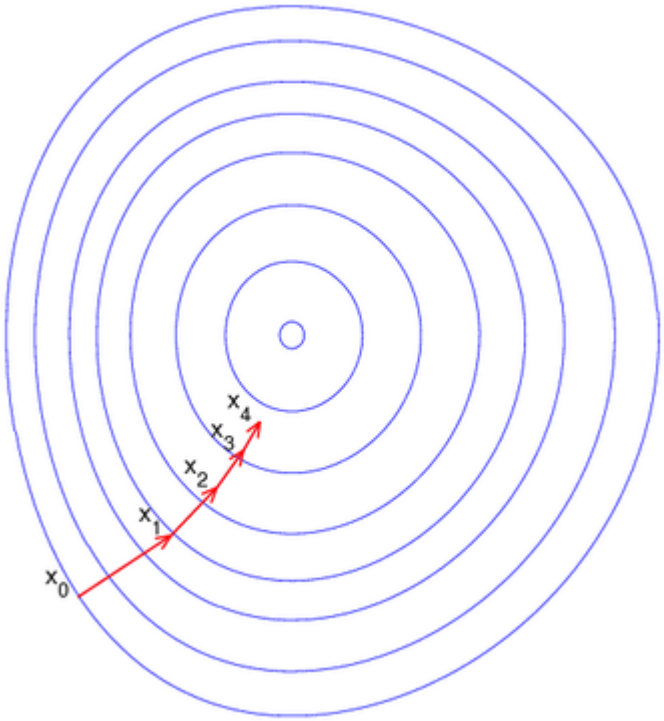
参考自

- 常见的几种最优化方法

常见的最优化方法有梯度下降法、牛顿法和拟牛顿法、共轭梯度法等等

## 梯度下降法

梯度下降法是最早最简单，也是最为常用的最优化方法。梯度下降法实现简单，当目标函数是凸函数时，梯度下降法的解是全局解。一般情况下，其解不保证是全局最优解，梯度下降法的速度也未必是最快的。梯度下降法的优化思想是用当前位置负梯度方向作为搜索方向，因为该方向为当前位置的最快下降方向，所以也被称为是”最速下降法“。最速下降法越接近目标值，步长越小，前进越慢。梯度下降法的搜索迭代示意图如下图所示：

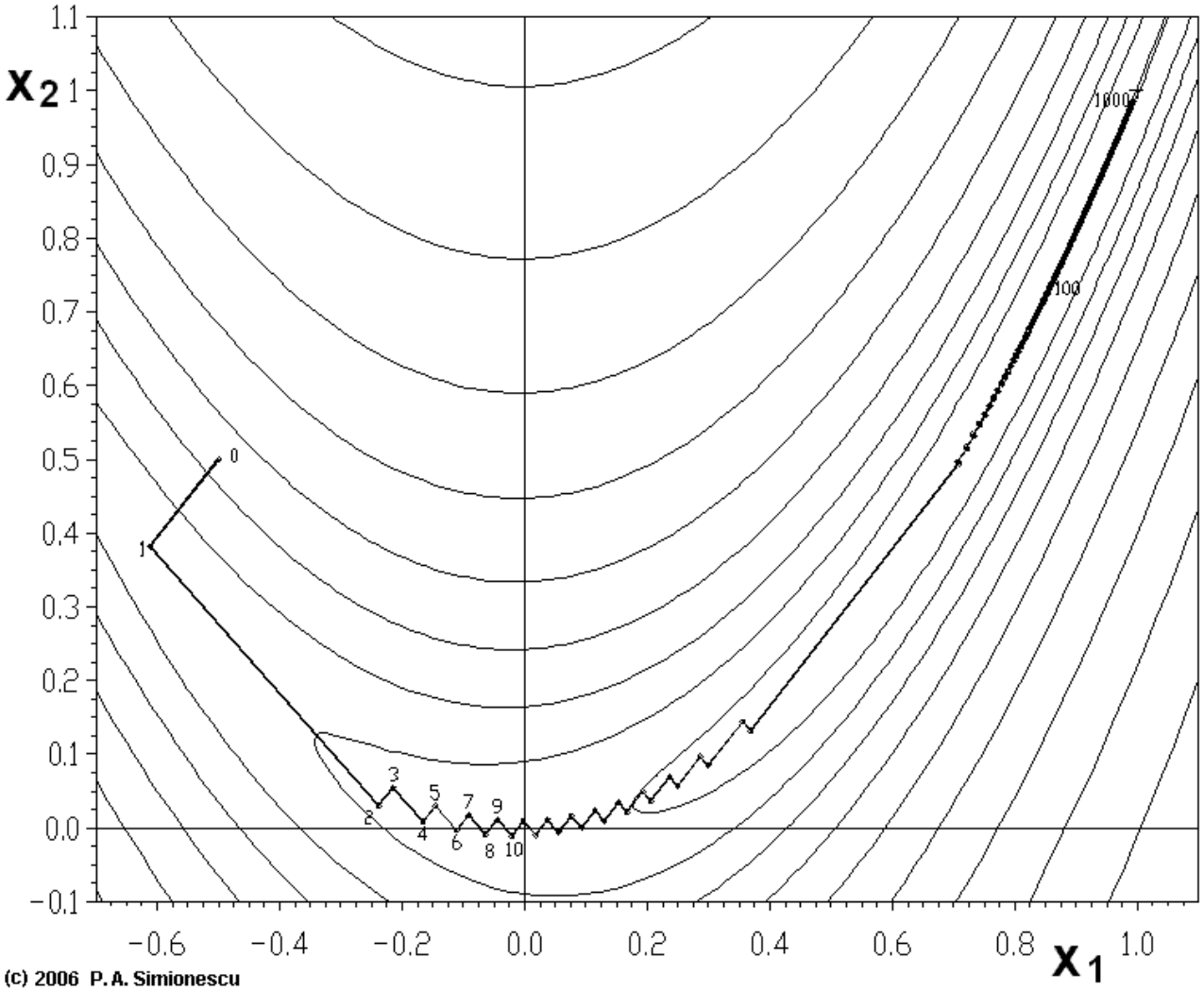



其缺点是：

（**1**）靠近极小值时收敛速度减慢，如下图所示；

（**2**）直线搜索时可能会产生一些问题；

（**3**）可能会“之字形”地下降。




从上图可以看出，梯度下降法在接近最优解的区域收敛速度明显变慢，利用梯度下降法求解需要很多次的迭代。

在机器学习中，基于基本的梯度下降法发展了两种梯度下降方法，分别为随机梯度下降法和批量梯度下降法。

比如对一个线性回归模型，假设$h(x)$是要拟合的函数，$J(\theta)$是损失函数，而$\theta$是参数，要迭代求解的值，然后$m$是训练集样本个数，$n$是特征个数，则有：

$$h(\theta) = \sum_{j=0}^{n} \theta_j x_j$$

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

## 批量梯度下降法(Batch Gradient Descent, BGD)

首先是令损失函数对参数求偏导，即$\frac{\partial J(\theta)}{\partial \theta_j} = -\frac{1}{m} \sum_{i=1}^{m} (y^i - h_\theta(x^i)) x_j^i$。然后需要最小化损失函数，所以每个参数会按照其梯度负方向更新，即有：

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

从上述公式可知，其求解的是一个全局最优解，因为每次更新参数的时候，都需要使用整个训练集的样本，所以如果$m$很大，那么每次迭代的速度就会很慢，这也是为何有随机梯度下降方法出现的原因。

对于这种方法，每次迭代的计算量是$O(m*n^2)$，$m,n$分别是样本总数量和每个样本的特征维度。

## 随机梯度下降(Stochastic Gradient Descent, SGD)

上述使用过的损失函数可以写成如下形式：

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

$$cost(\theta,(x^i,y^i))=\frac{1}{2}(y^i-h_\theta(x^i))^2$$

而函数更新公式如下：

$$\theta_j^{'}=\theta_j+(y^i-h_\theta(x^i))x_j^i$$

所以SGD是通过每个样本来迭代更新一次，当样本数量很大的时候，那么可能只需要使用其中一部分即可找到最优解。相比BGD方法，迭代速度加快了很多。但是SGD的一个问题是噪音会更多，这使得它每次迭代并不是朝着最优解的方向。

随机梯度下降每次迭代只使用一个样本，迭代一次计算量为$O(n^2)$，当样本个数$m$很大的时候，随机梯度下降迭代一次的速度要远高于批量梯度下降方法。两者的关系可以这样理解：随机梯度下降方法以损失很小的一部分精确度和增加一定数量的迭代次数为代价，换取了总体的优化效率的提升。增加的迭代次数远远小于样本的数量。

对批量梯度下降法和随机梯度下降法的总结：

- 批量梯度下降---最小化所有训练样本的损失函数，使得最终求解的是全局的最优解，即求解的参数是使得风险函数最小，但是对于大规模样本问题效率低下。
- 随机梯度下降---最小化每条样本的损失函数，虽然不是每次迭代得到的损失函数都向着全局最优方向， 但是大的整体的方向是向全局最优解的，最终的结果往往是在全局最优解附近，适用于大规模训练样本情况。

# 牛顿法

牛顿法是一种在实数域和复数域上近似求解方程的方法。方法使用函数$f(x)$的泰勒级数的前面几项来寻找方程$f(x)=0$的根。它是二阶算法，它使用了Hessian矩阵求权重的二阶偏导数，目标是采用损失函数的二阶偏导数寻找更好的训练方向。牛顿法最大的特点就在于它的收敛速度很快。

具体步骤如下：

首先，选择一个接近函数$f(x)$零点的$x_0$，计算相应的$f(x_0)$和切线斜率$f\prime(x_0)$($f\prime$表示函数$f$的导数)。然后我们计算穿过点$(x_0,f(x_0))$并且斜率是$f\prime(x_0)$的直线和$x$轴的交点的$x$坐标，也就是求下列方程的解：

$$x*f\prime(x_0)+f(x_0)-x_0*f\prime(x_0)=0$$

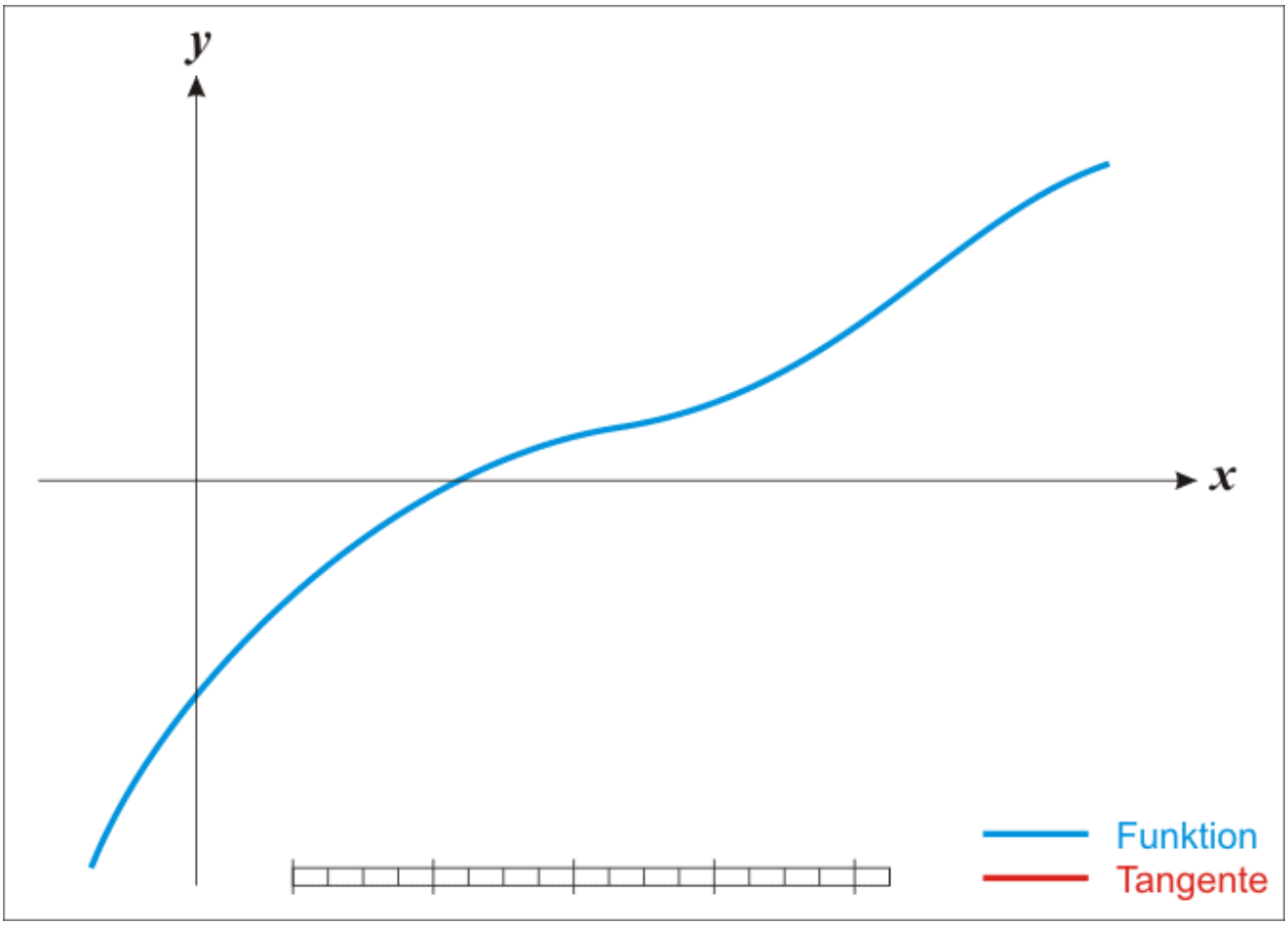
我们将求得的新点的$x$坐标命名为$x_1$，通常$x_1$会比$x_0$更接近方程$f(x)=0$的解。因此，我们可以利用$x_1$开始下一轮迭代。迭代公式可化简为如下所示：

$$x_{n+1}=x_n-\frac{f(x_n)}{f\prime(x_n)}$$

已经证明，如果$f\prime$是连续的，并且待求的零点$x$是孤立的，那么在零点$x$周围存在一个区域，只要初始值$x_0$位于这个邻近区域内，那么牛顿法必定收敛。 并且，如果$f\prime(x)$不为0, 那么牛顿法将具有平方收敛的性能. 粗略的说，这意味着每迭代一次，牛顿法结果的有效数字将增加一倍。

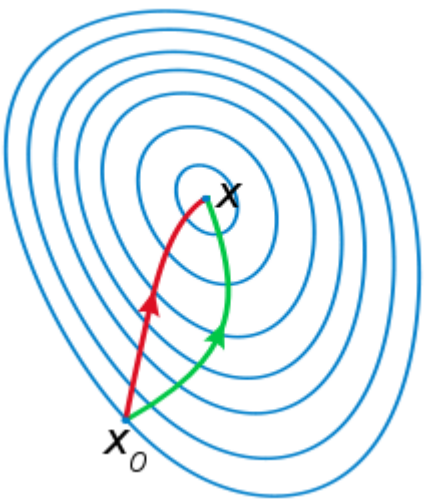
由于牛顿法是基于当前位置的切线来确定下一次的位置，所以牛顿法又被很形象地称为是"切线法"。牛顿法的搜索路径（二维情况）如下图所示：

牛顿法搜索动态示例图：




关于牛顿法和梯度下降法的效率对比：

- 从本质上去看，牛顿法是二阶收敛，梯度下降是一阶收敛，所以牛顿法就更快。如果更通俗地说的话，比如你想找一条最短的路径走到一个盆地的最底部，梯度下降法每次只从你当前所处位置选一个坡度最大的方向走一步，牛顿法在选择方向时，不仅会考虑坡度是否够大，还会考虑你走了一步之后，坡度是否会变得更大。所以，可以说牛顿法比梯度下降法看得更远一点，能更快地走到最底部。（牛顿法目光更加长远，所以少走弯路；相对而言，梯度下降法只考虑了局部的最优，没有全局思想。）
- 根据wiki上的解释，从几何上说，牛顿法就是用一个二次曲面去拟合当前所处位置的局部曲面，而梯度下降法是用一个平面去拟合当前的局部曲面，通常情况下，二次曲面的拟合会比平面更好，所以牛顿法选择的下降路径会更符合真实的最优下降路径。




注：红色的牛顿法的迭代路径，绿色的是梯度下降法的迭代路径。

### 优缺点

优点

二阶收敛，收敛速度快；

缺点

1. **Hessian**矩阵（海森矩阵的逆）计算量较大，当问题规模较大时，不仅计算量大而且需要的存储空间也多，因此牛顿法在面对海量数据时由于每一步迭代的开销巨大而变得不适用；
2. 牛顿法在每次迭代时不能总是保证海森矩阵是正定的，一旦海森矩阵不是正定的，优化方向就会“跑偏”，从而使得牛顿法失效，也说明了牛顿法的鲁棒性较差。

## 拟牛顿法

拟牛顿法的本质思想是改善牛顿法每次需要求解复杂的**Hessian**矩阵的逆矩阵的缺陷，它使用正定矩阵来近似**Hessian**矩阵的逆，从而简化了运算的复杂度。拟牛顿法和最速下降法一样只要求每一步迭代时知道目标函数的梯度。通过测量梯度的变化，构造一个目标函数的模型使之足以产生超线性收敛性。这类方法大大优于最速下降法，尤其对于困难的问题。另外，因为拟牛顿法不需要二阶导数的信息，而是在每次迭代的时候计算一个矩阵，其逼近海塞矩阵的逆。最重要的是，该逼近值只是使用损失函数的一阶偏导来计算，所以有时比牛顿法更为有效。如今，优化软件中包含了大量的拟牛顿算法用来解决无约束，约束，和大规模的优化问题。

具体步骤如下：

首先，构造目标函数在当前迭代$x_k$的二次模型：

$$m_k(p) = f(x_k) + \nabla f(x_k)^T p + \frac{p^T B_k p}{2}$$
$$p_k = -B_k^{-1} \nabla f(x_k)$$

这里$B_k$是一个对称正定矩阵，我们取这个二次模型的最优解作为搜索方向，并且得到新的迭代点：

$x_{k+1} = x_k + a_k p_k$

其中要求步长$a_k$满足Wolfe条件。这样的迭代与牛顿法类似，区别就在于用近似的**Hessian**矩阵$B_k$代替真正的**Hessian**矩阵。所以拟牛顿法最关键的地方就是每次迭代中矩阵$B_k$的更新。现在假设得到一个新的迭代$x_{k+1}$，并得到一个新的二次模型：

$$m_{k+1}(p) = f(x_{k+1}) + \nabla f(x_{k+1})^T p + \frac{p^T B_{k+1} p}{2}$$

我们尽可能利用上一步的信息来选取$B_k$，具体地，我们要求$\nabla f(x_{k+1}) - \nabla f(x_k) = a_k B_{k+1} p_k$

从而得到$B_{k+1}(x_{k+1} - x_k) = \nabla f(x_{k+1}) - \nabla f(x_k)$

这个公式被称为割线方程。常用的拟牛顿法有DFP算法和BFGS算法。

# 共轭梯度法(Conjugate Gradient)

共轭梯度法是介于最速下降法与牛顿法之间的一个方法，它仅需利用一阶导数信息，但克服了最速下降法收敛慢的缺点，又避免了牛顿法需要存储和计算**Hesse**矩阵并求逆的缺点，共轭梯度法不仅是解决大型线性方程组最有用的方法之一，也是解大型非线性最优化最有效的算法之一。在各种优化算法中，共轭梯度法是非常重要的一种。其优点是所需存储量小，具有收敛快，稳定性高，而且不需要任何外来参数。

在共轭梯度训练算法中，因为是沿着共轭方向（**conjugate directions**）执行搜索的，所以通常该算法要比沿着梯度下降方向优化收敛得更迅速。共轭梯度法的训练方向是与海塞矩阵共轭的。

共轭梯度法已经证实其在神经网络中要比梯度下降法有效得多。并且由于共轭梯度法并没有要求使用海塞矩阵，所以在大规模神经网络中其还是可以做到很好的性能。

具体的实现步骤请参加[wiki百科共轭梯度法](#)。

# 启发式优化方法

启发式方法指人在解决问题时所采取的一种根据经验规则进行发现的方法。其特点是在解决问题时,利用过去的经验,选择已经行之有效的方法，而不是系统地、以确定的步骤去寻求答案。启发式优化方法种类繁多，包括经典的模拟退火方法、遗传算法、蚁群算法以及粒子群算法等等。

还有一种特殊的优化算法被称之多目标优化算法，它主要针对同时优化多个目标（两个及两个以上）的优化问题，这方面比较经典的算法有NSGAII算法、MOEA/D算法以及人工免疫算法等。

具体可以参考文章[Evolutionary Algorithm] 进化算法简介

# 解决约束优化问题--拉格朗日乘数法

这个方法可以参考文章[拉格朗日乘数法](#)

**Levenberg-Marquardt** 算法

Levenberg-Marquardt 算法，也称之为衰减最小二乘法（damped least-squares method），该算法的损失函数采用平方误差和的形式。该算法的执行也不需要计算具体的海塞矩阵，它仅仅只是使用梯度向量和雅可比矩阵（**Jacobian matrix**）。

该算法的损失函数使用如下所示的方程，即平方误差和的形式：

$$f = \sum_{i=0}^{m} e_i^2$$

其中参数$m$是数据集样本的数量。

我们可以定义损失函数的雅克比矩阵是以误差对参数的偏导数为元素的，即

$J_{i,j} f(w) = \frac{de_i}{dw_j} \quad i = 1, 2..., m \quad j = 1, 2, \ldots, n$

$n$是神经网络的参数数量，所以雅克比矩阵就是$m * n$的矩阵。

损失函数的梯度向量就是$\nabla f = 2J^T e$，$e$在这里表示所有误差项的向量。

所以，我们最后可以使用下列表达式来逼近Hessian矩阵：

$$H(f) \approx 2J^T J + \lambda I$$

其中$\lambda$是衰减因子，它确保了Hessian矩阵的正定性，$I$是单位矩阵。

所以，参数的更新和优化如下公式所示：

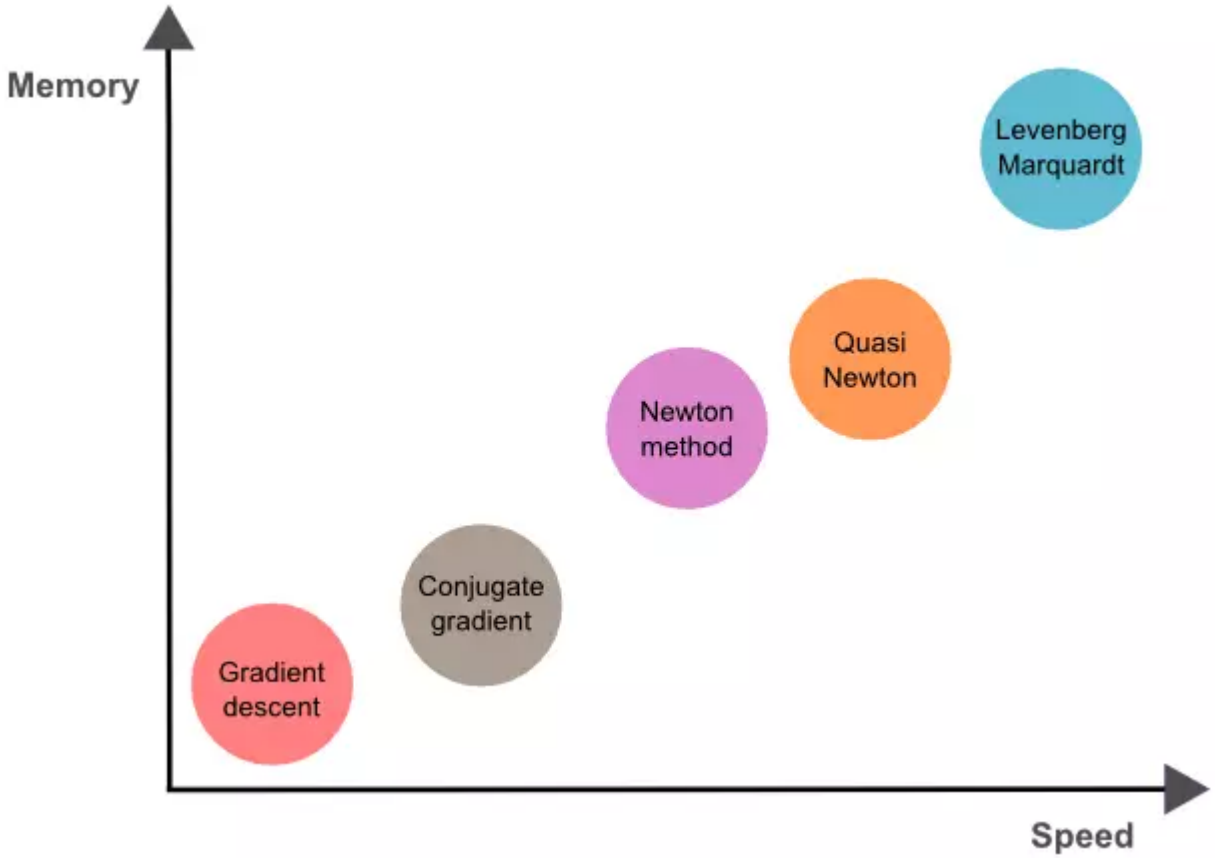
$$w_{i+1} = w_i - (J_i^T J_i + \lambda I)^{-1}(2J_i^T e_i), \quad i = 0, 1, \ldots$$

当$\lambda = 0$，该算法就是使用Hessian矩阵逼近值的牛顿法，而当$\lambda$很大时，该算法就近似于采用很小学习率的梯度下降法。如果进行迭代导致了损失函数上升，衰减因子$\lambda$就会增加。如果损失函数下降，那么$\lambda$就会下降，从而 **Levenberg-Marquardt** 算法更接近于牛顿法。该过程经常用于加速收敛到极小值点。

Levenberg-Marquardt 算法是为平方误差和函数所定制的。这就让使用这种误差度量的神经网络训练地十分迅速。然而 Levenberg-Marquardt 算法还有一些缺点，第一就是其不能用于平方根误差或交叉熵误差（**cross entropy error**）等函数，此外该算法还和正则项不兼容。最后，对于大型数据集或神经网络，雅可比矩阵会变得十分巨大，因此也需要大量的内存。所以我们在大型数据集或神经网络中并不推荐采用 Levenberg-Marquardt 算法。

## 内存与收敛速度的比较

下图展示了所有上文所讨论的算法，及其收敛速度和内存需求。其中收敛速度最慢的是梯度下降算法，但该算法同时也只要求最少的内存。相反，Levenberg-Marquardt 算法可能是收敛速度最快的，但其同时也要求最多的内存。比较折衷方法是拟牛顿法。




总而言之，如果我们的神经网络有数万参数，为了节约内存，我们可以使用梯度下降或共轭梯度法。如果我们需要训练多个神经网络，并且每个神经网络都只有数百参数、数千样本，那么我们可以考虑 Levenberg-Marquardt 算法。而其余的情况，拟牛顿法都能很好地应对。