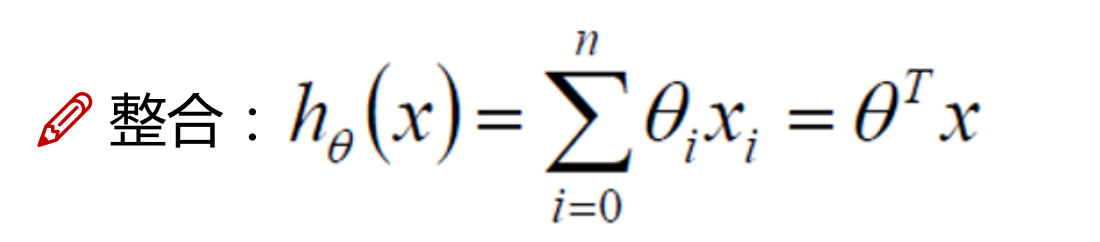
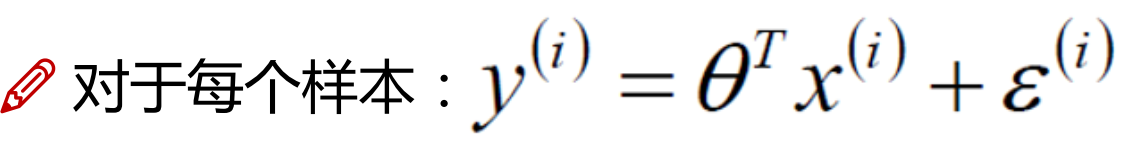
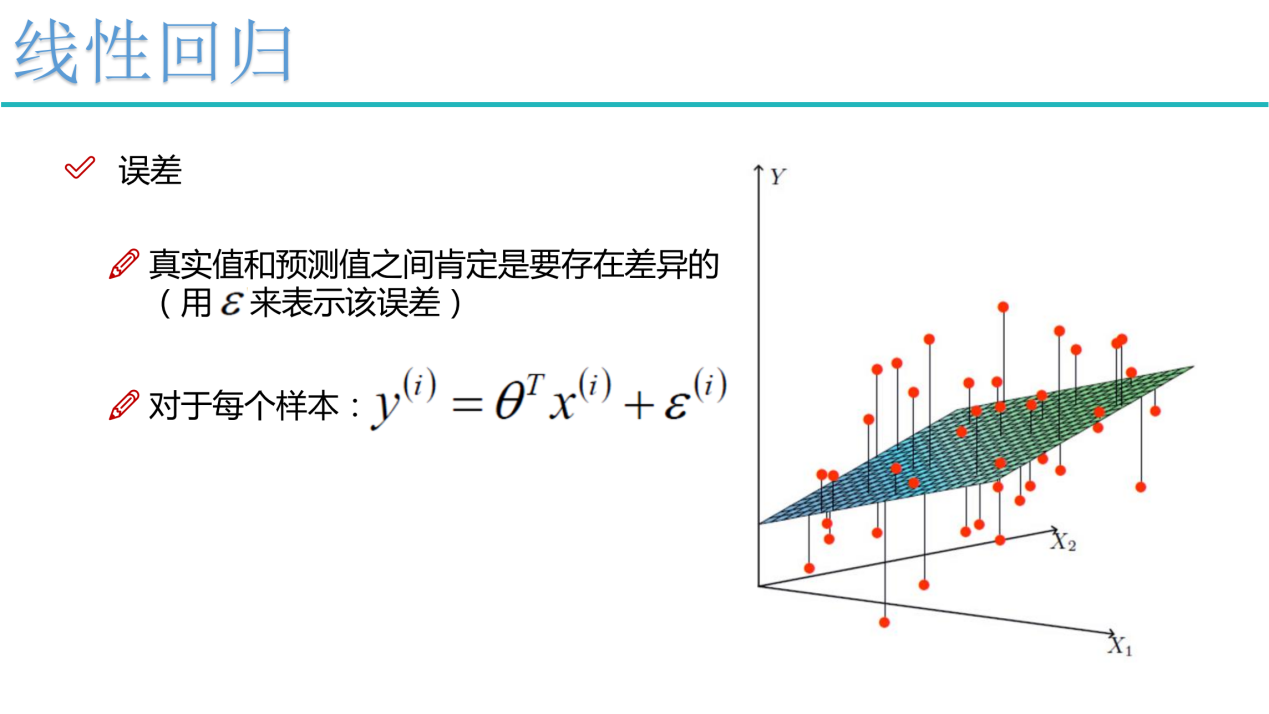


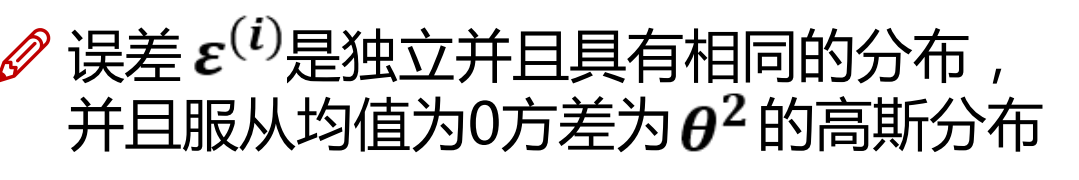
某一样本的预测结果可以表示为

对于所有样本我们可以用，这里的i和上面的i不同，

上面的i是维度，这里的i是代表样本

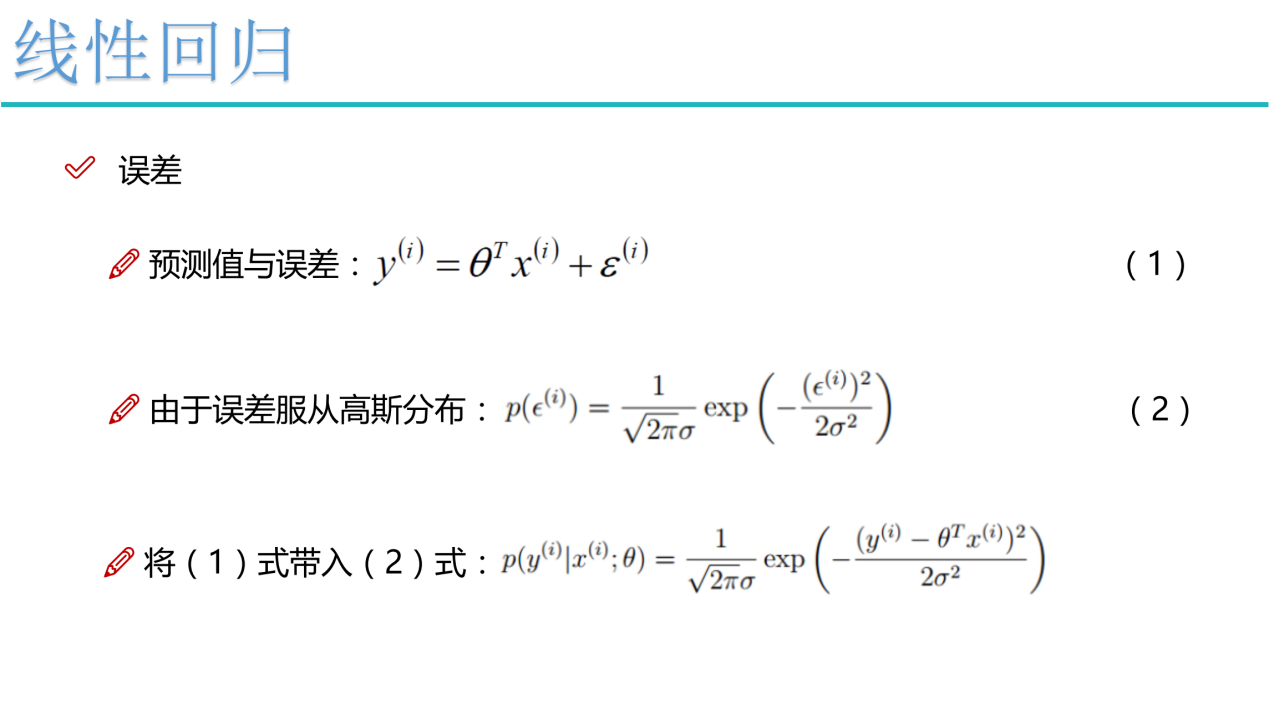






因为我们已经知道这种误差是服从某一分布的，我们就可以用似然估计来预估参数，这个参数就是分布的方差啊期望啊，

我们知道误差服从高斯分布，但是我们不知道具体的分布参数，这个需要估计

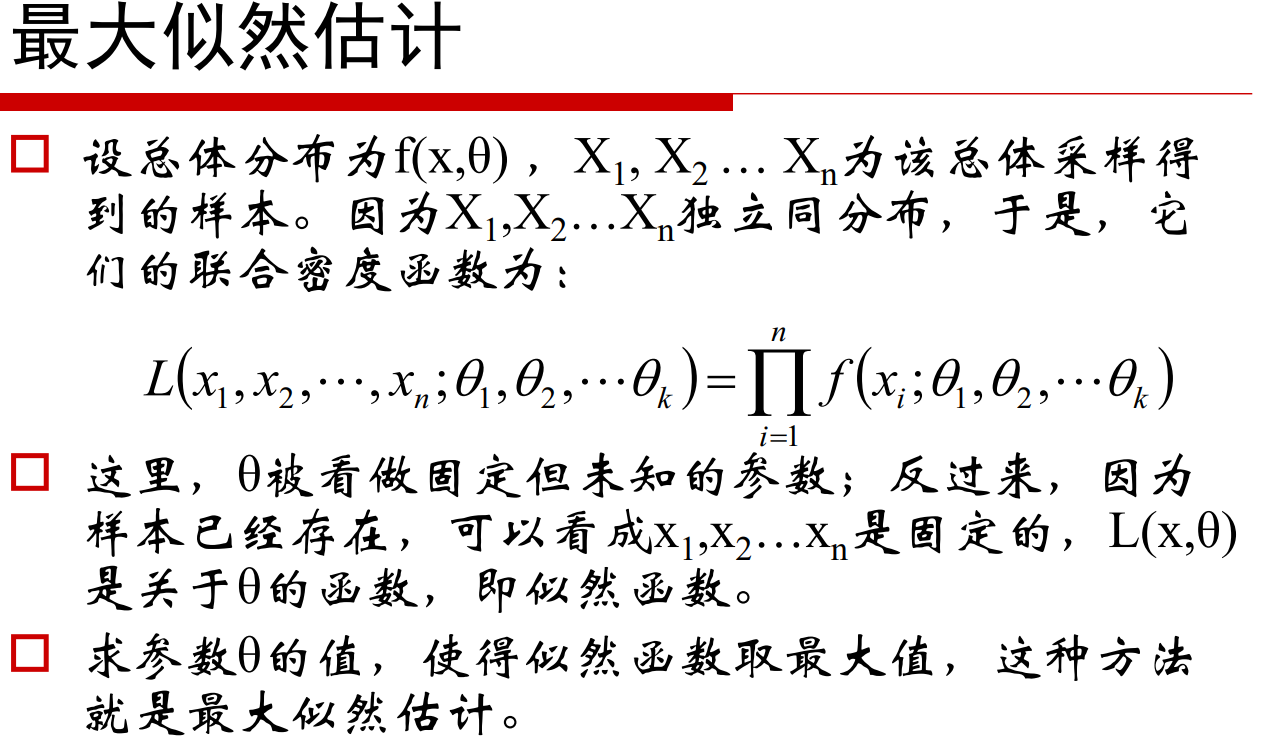


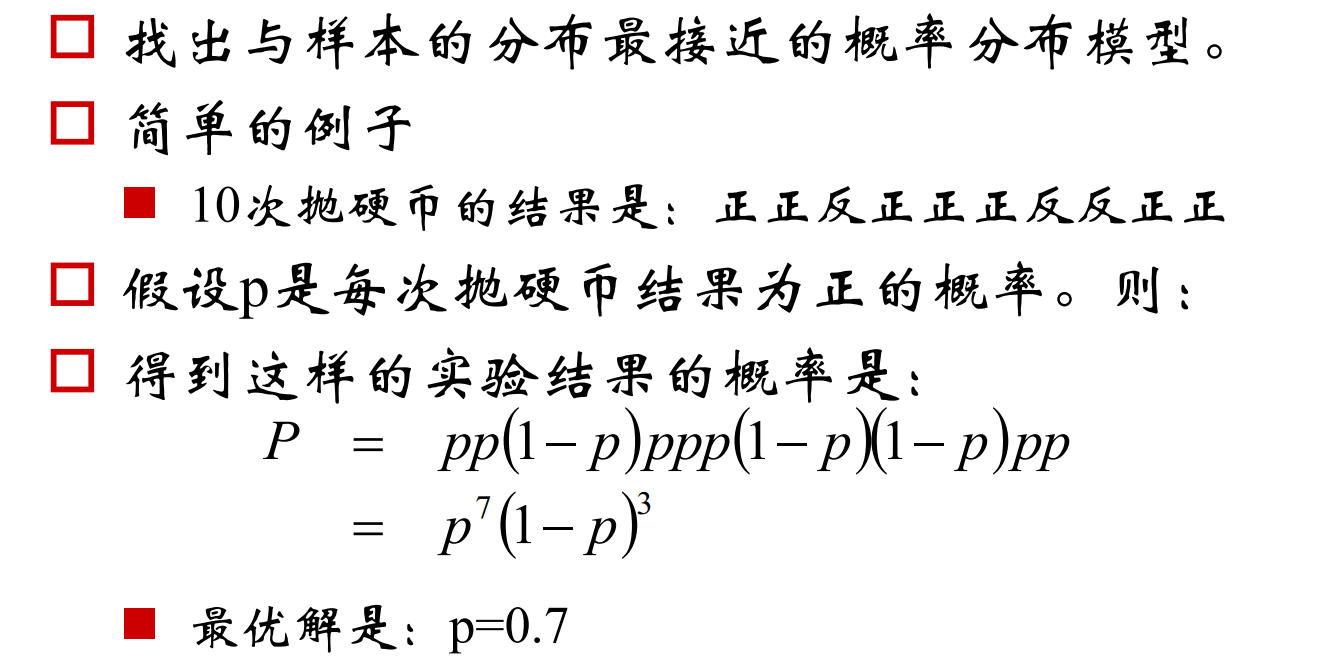
为什么用最大似然估计呢？

当我们有一个样本分布数据，我们是可以去估计某一个事件发生的概率。

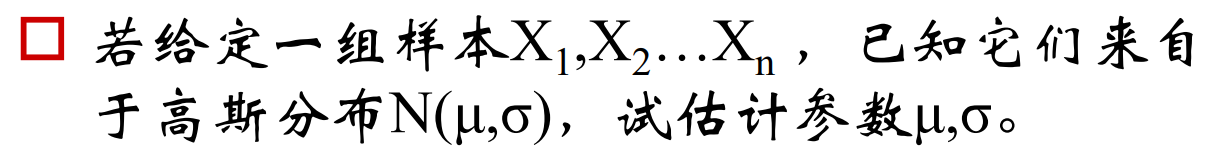
最大似然估计的通俗理解就是使当前样本最有可能发生的情况下，这个分布的参数是啥，

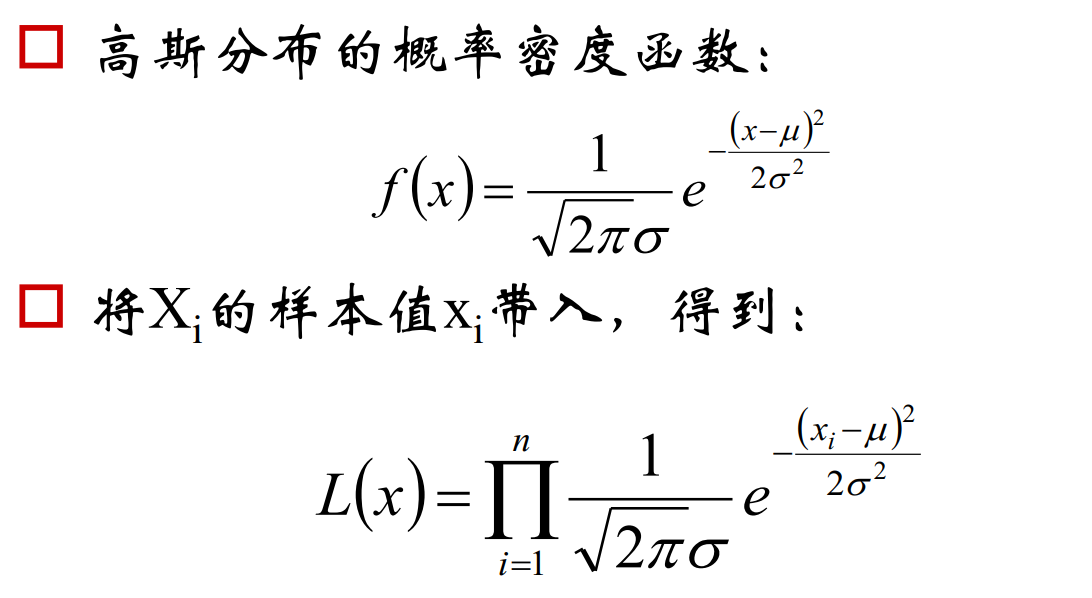
这里的参数并不是说一个样本有一个参数，而是说这个分布可能有多个参数，比如说高斯分布



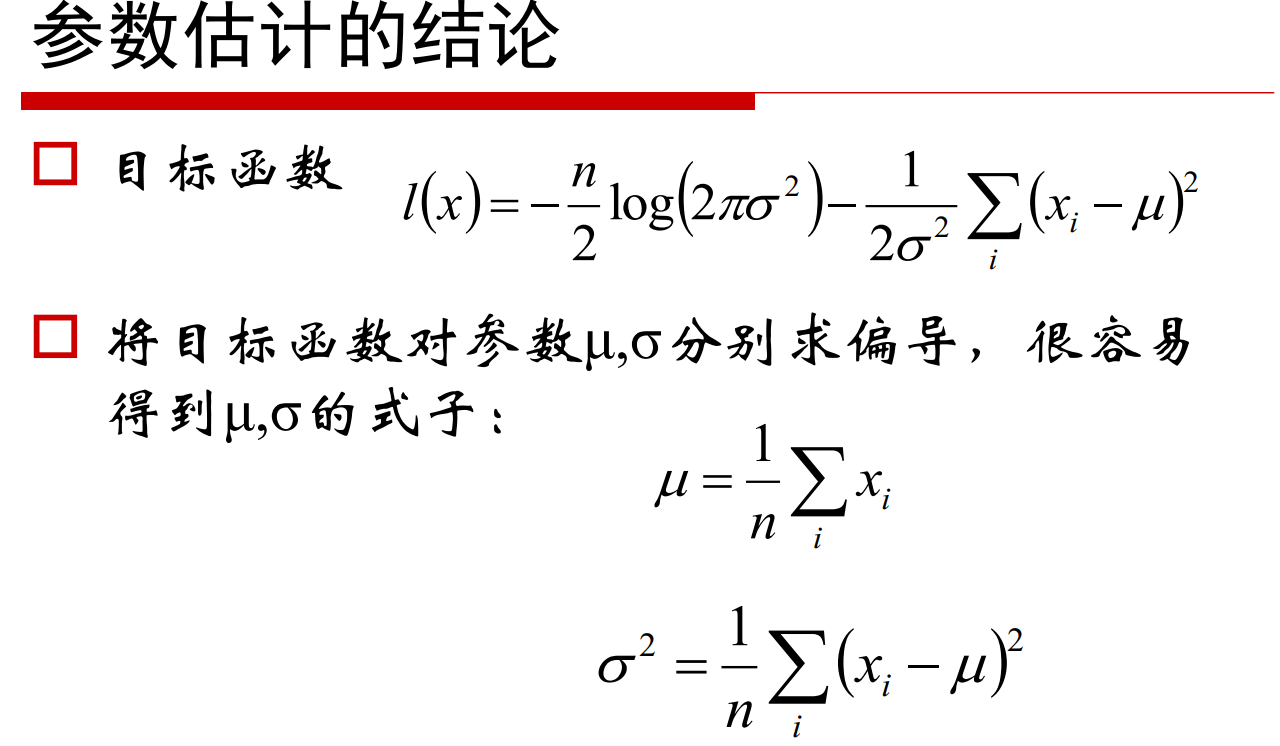


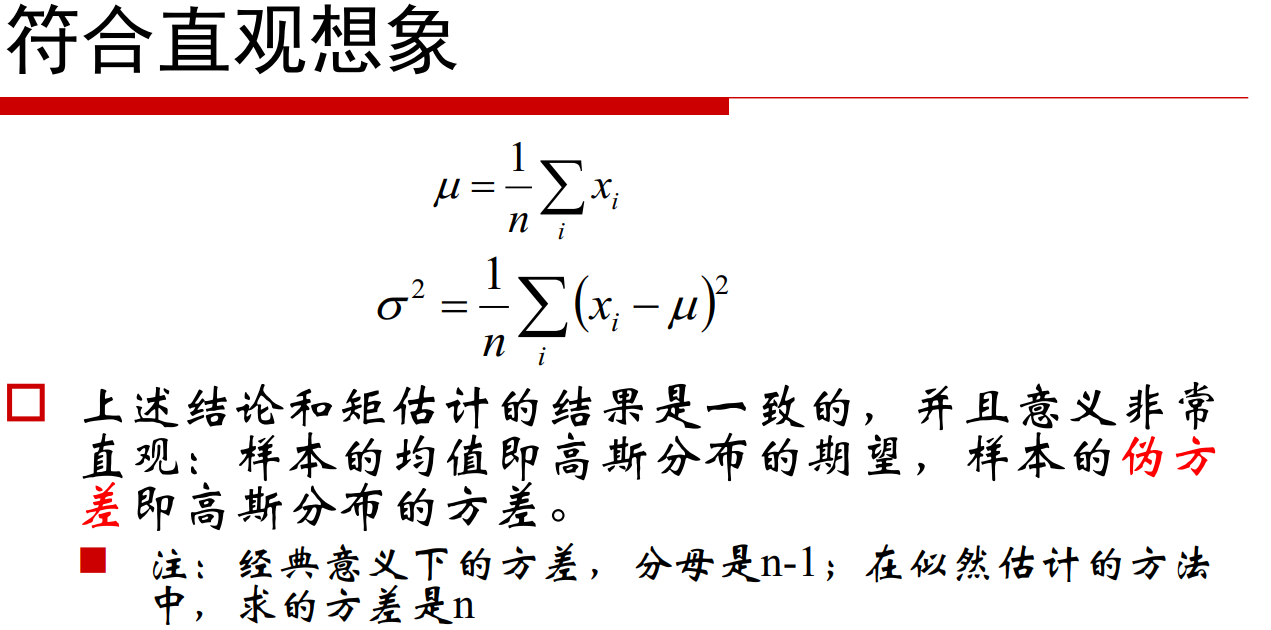
对于上例，抛硬币来说，我们得到了一个分布，虽然我们不知道这个结果服从什么已知的分布，但是我们可以用联合密度函数来预估某一事件发生的概率，也就是说p=0.7时最有可能是上述样本情况发生





F（x）代表x发生的概率，L（x）代表这些独立同分布样本连续发生的概率，最终可以用一个函数表示，这里自变量已经变成了这些参数



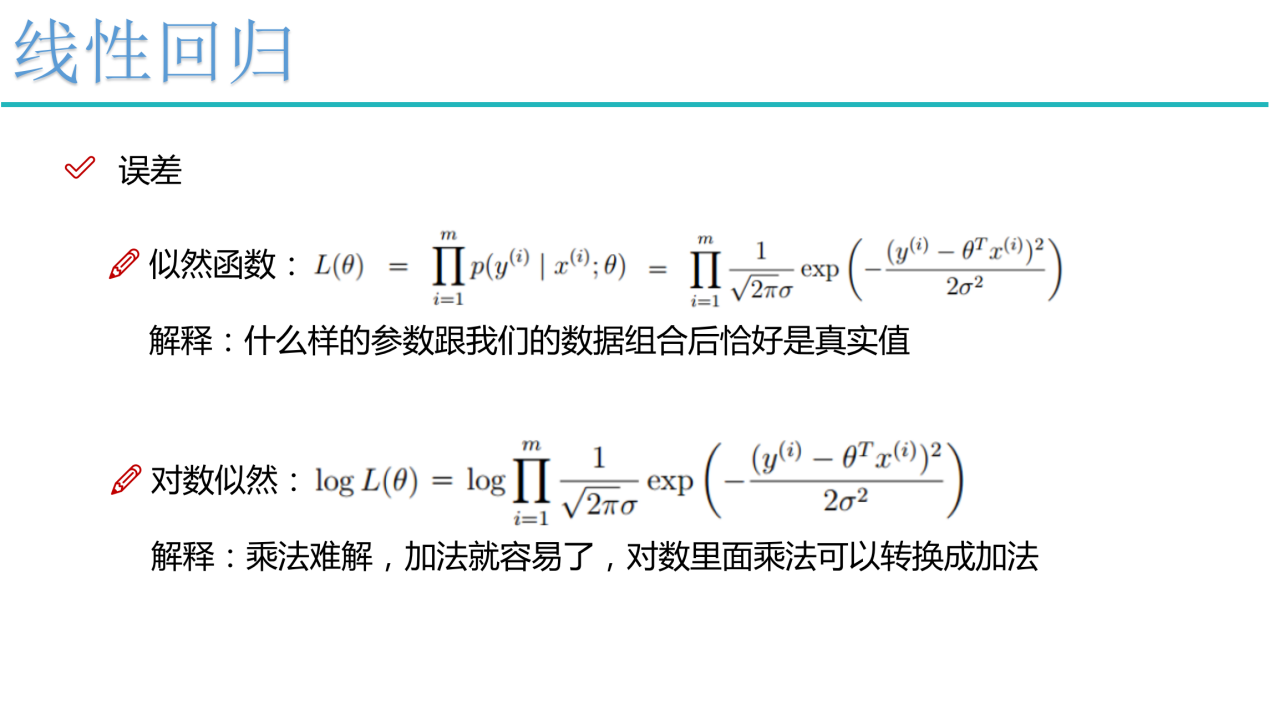


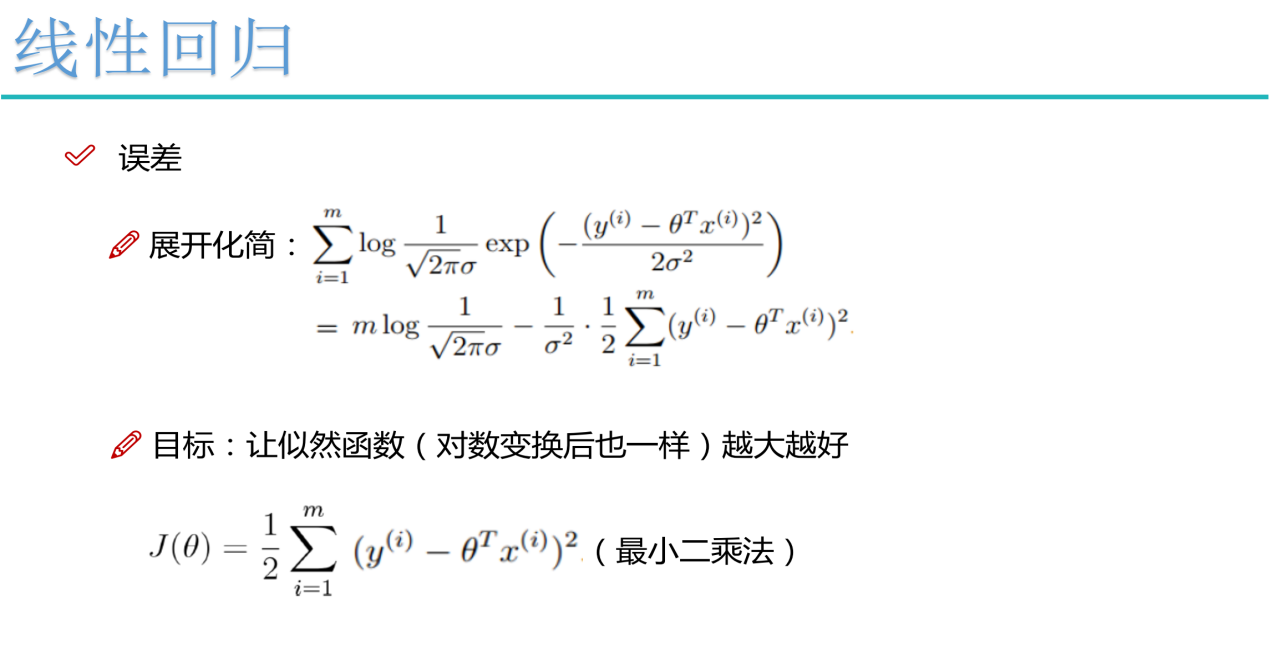
上述推导可以看出最大似然估计受到样本量的影响较大

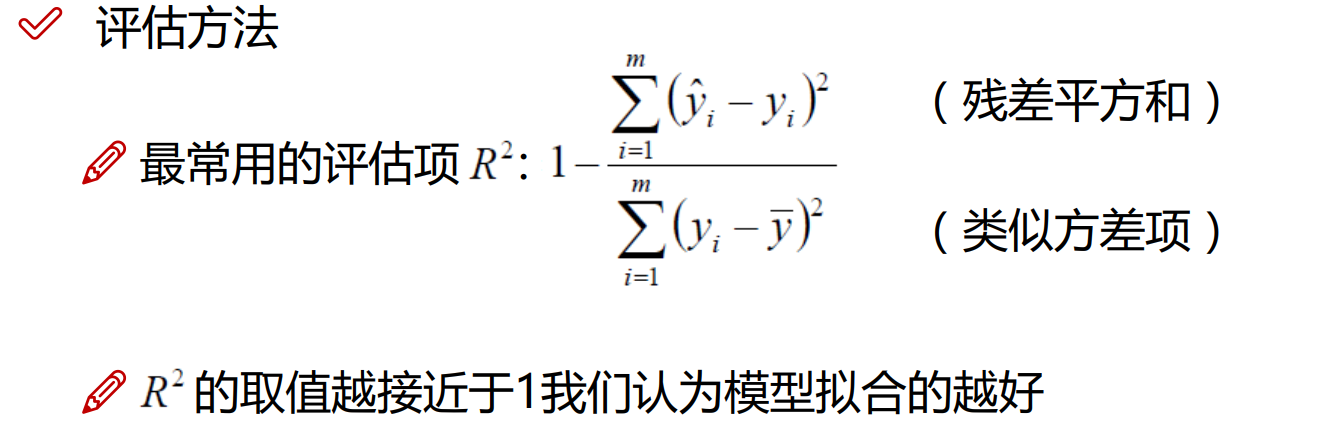
当我们知道某一组数据是服从已知分布的，那我们就可以进行估计

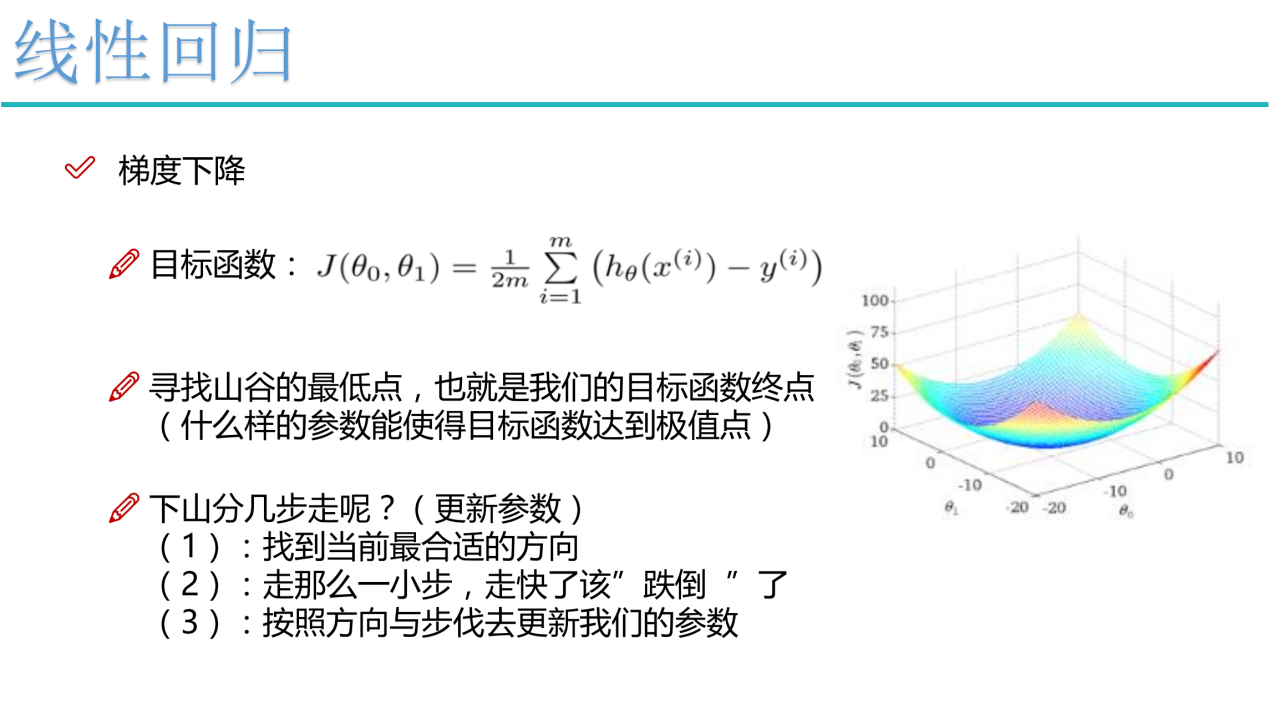
这里的似然函数本质上来理解的话

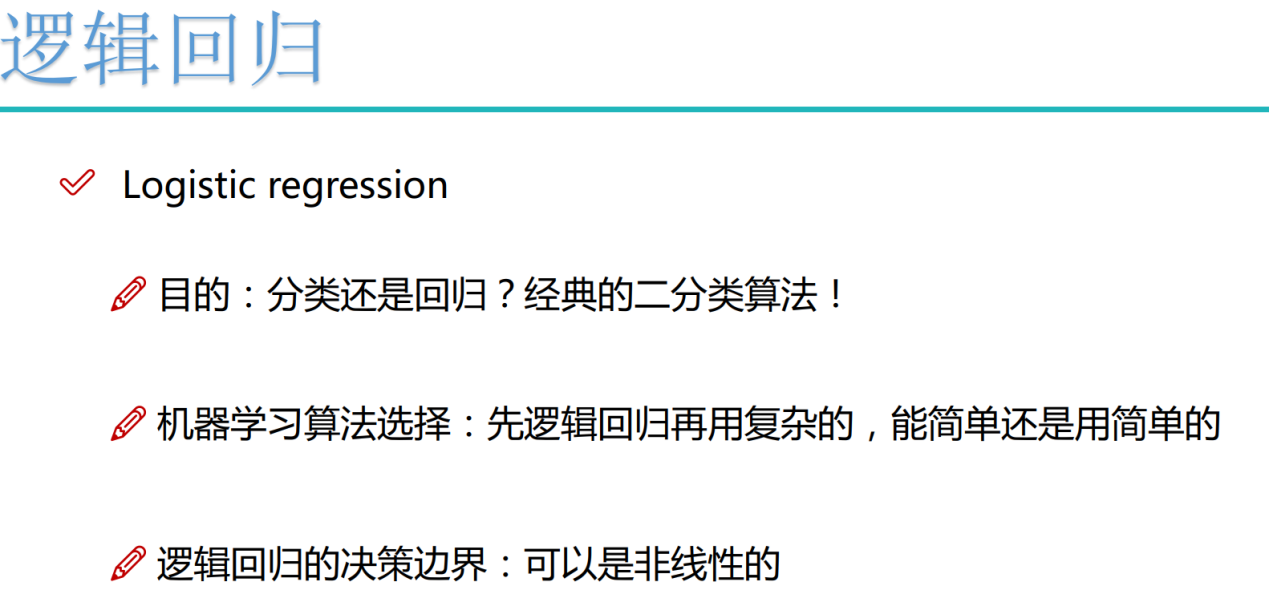
比如我们已经得到了一组预测值与真实值之间的误差值，我们知道其服从正态分布，但不知道具体参数，这里似然函数就是，不同误差值发生的概率值相乘得到的一个函数。我们要找到这个高斯分布的两个参数，这两个参数要让误差分布最接近我们真实的误差样本值，直观上理解就是，找到这样的两个参数要使得我们观测的误差样本最有可能发生就是，所有样本都发的的概率最大，就是所有样本误差发生的概率的积最大，如果每一个样本发生的概率都是0.1和每一样本发生的概率都是0.5肯定是不样的。



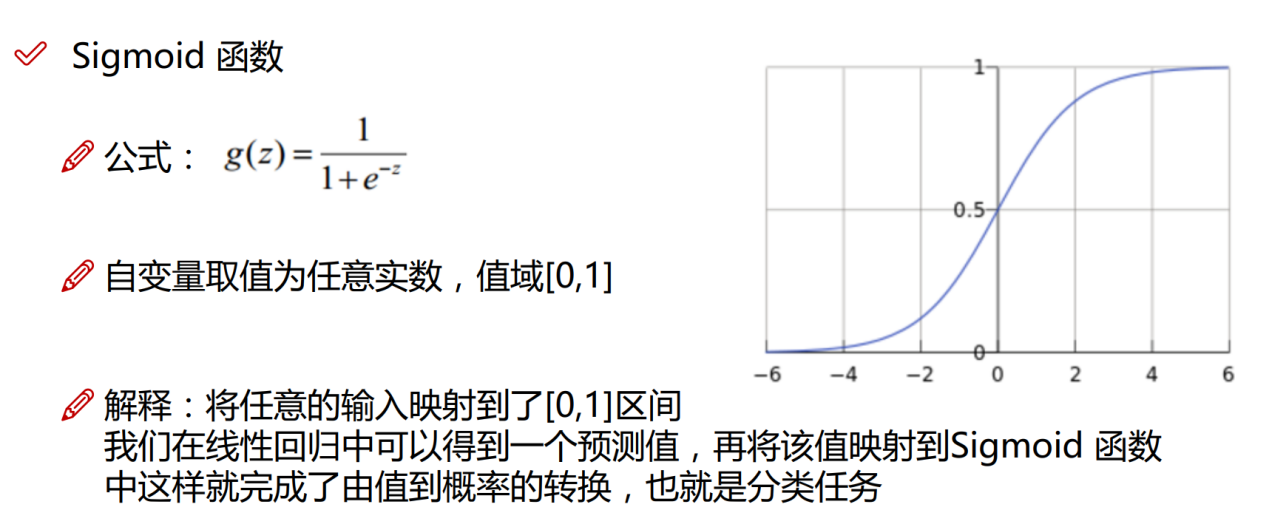








线性回归和逻辑回归的不同

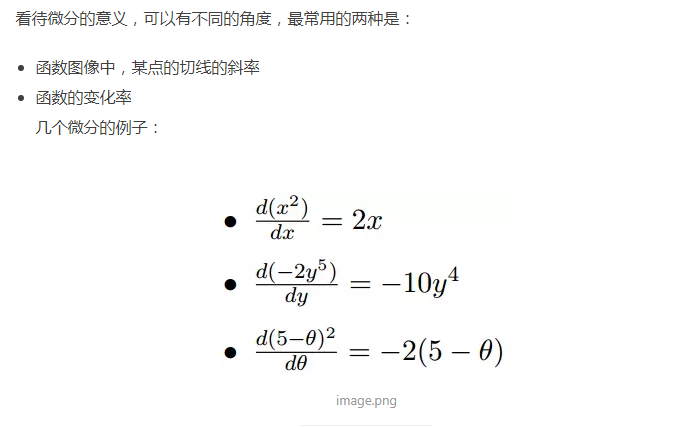


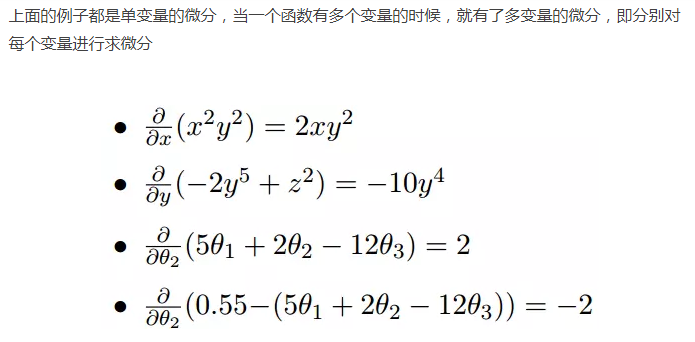
梯度下降

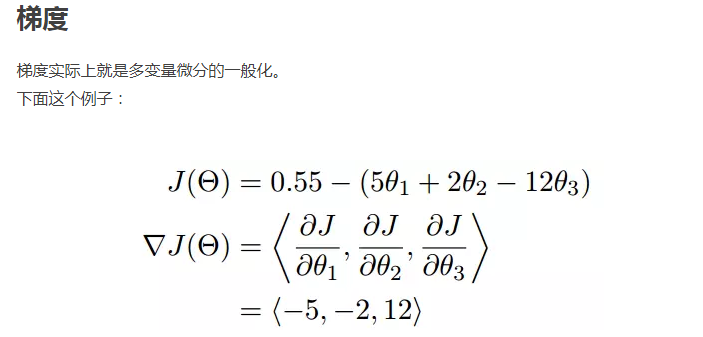
<https://www.jianshu.com/p/c7e642877b0e>

梯度下降的基本过程就和下山的场景很类似。首先，我们有一个可[微分](https://link.jianshu.com?t=https://en.wikipedia.org/wiki/Differentiable_function)的函数。这个函数就代表着一座山。我们的目标就是找到这个函数的最小值，也就是山底。根据之前的场景假设，最快的下山的方式就是找到当前位置最陡峭的方向，然后沿着此方向向下走，对应到函数中，就是找到给定点的[梯度](https://link.jianshu.com?t=https://en.wikipedia.org/wiki/Gradient) ，然后朝着梯度相反的方向，就能让函数值下降的最快！因为梯度的方向就是函数之变化最快的方向(在后面会详细解释)  
所以，我们重复利用这个方法，反复求取梯度，最后就能到达局部的最小值，这就类似于我们下山的过程。而求取梯度就确定了最陡峭的方向，也就是场景中测量方向的手段。

1微分







* 在单变量的函数中，梯度其实就是函数的微分，代表着函数在某个给定点的切线的斜率
* 在多变量函数中，梯度是一个向量，向量有方向，梯度的方向就指出了函数在给定点的上升最快的方向

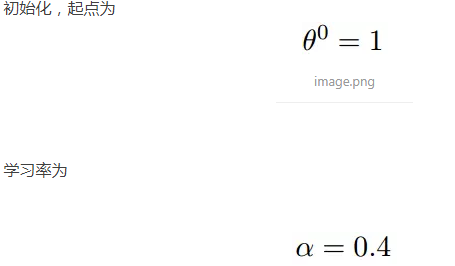


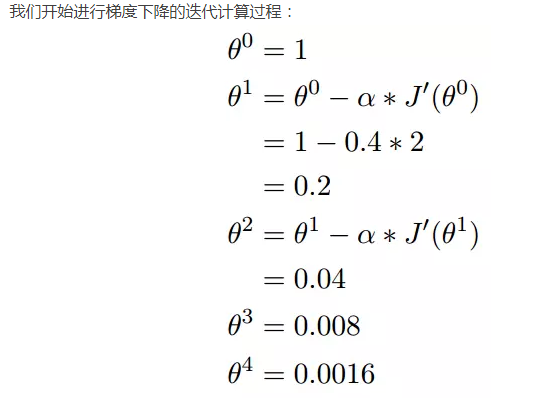
α是什么含义？  
α在梯度下降算法中被称作为**学习率**或者**步长**，意味着我们可以通过α来控制每一步走的距离，以保证不要步子跨的太大扯着蛋，哈哈，其实就是不要走太快，错过了最低点。同时也要保证不要走的太慢，导致太阳下山了，还没有走到山下。所以α的选择在梯度下降法中往往是很重要的！α不能太大也不能太小，太小的话，可能导致迟迟走不到最低点，太大的话，会导致错过最低点！

为什么要梯度要乘以一个负号？  
梯度前加一个负号，就意味着朝着梯度相反的方向前进！我们在前文提到，梯度的方向实际就是函数在此点上升最快的方向！而我们需要朝着下降最快的方向走，自然就是负的梯度的方向，所以此处需要加上负号

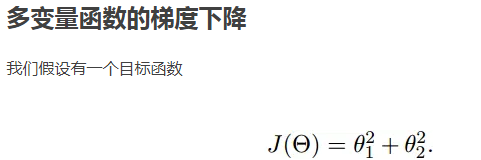
## 单变量函数的梯度下降

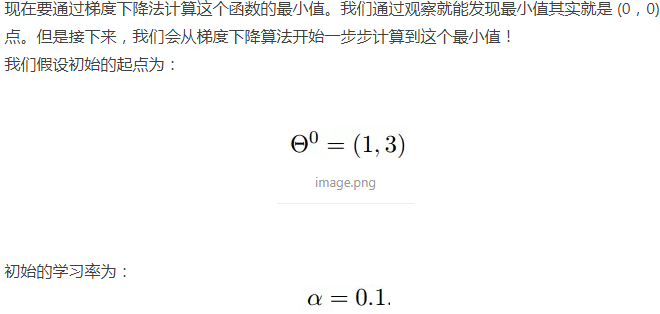




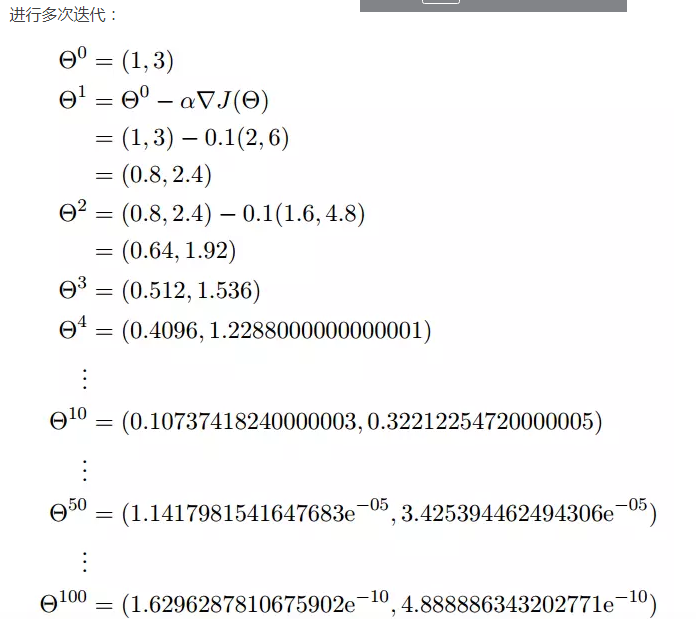


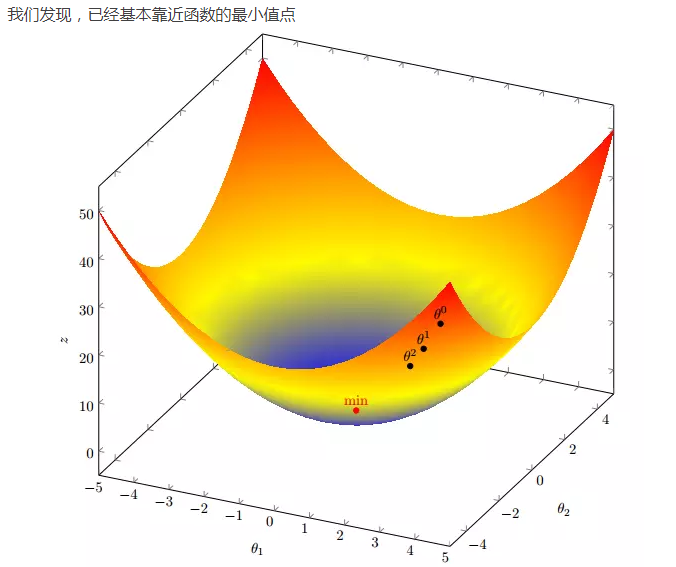
对于自变量O来说，只有给定一个O值，才能求出对应的负梯度值，这个值就是自变量变化的方向与大小



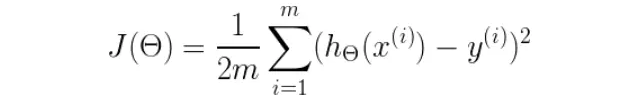


函数的梯度为





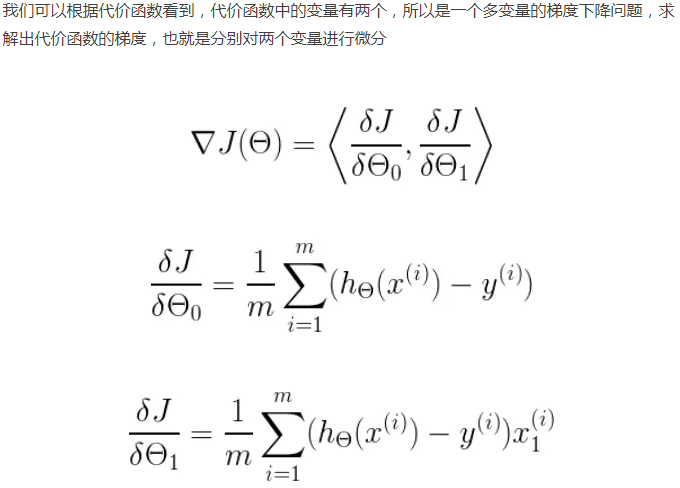
拿线性回归来举例子



这里自变量Xi 是特征值，O是权重，整个过程都是再调整权重O。

假设只有一维变量

有两个参数需要学习



权值完全可以给定初始随机值，然后开始调整

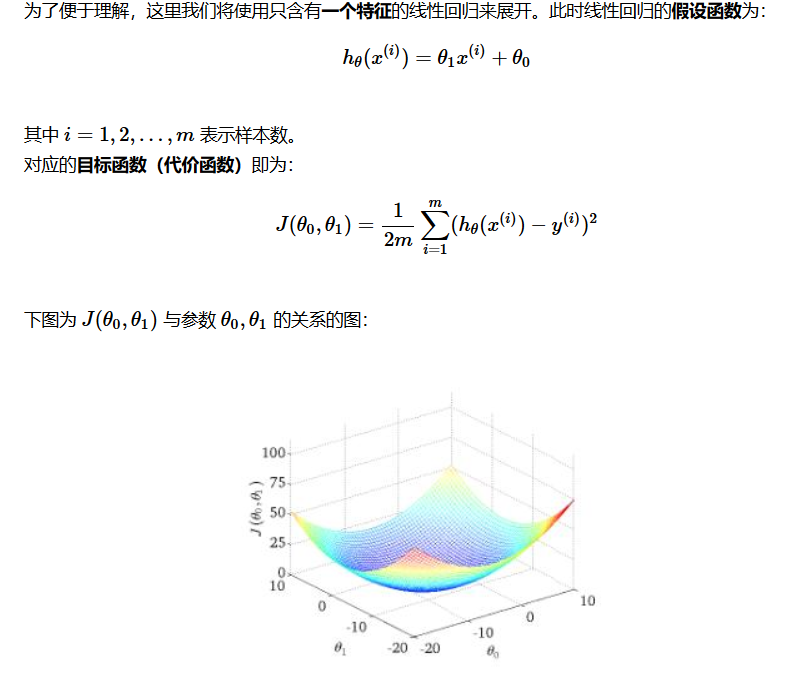
梯度下降法作为机器学习中较常使用的优化算法，其有着三种不同的形式：

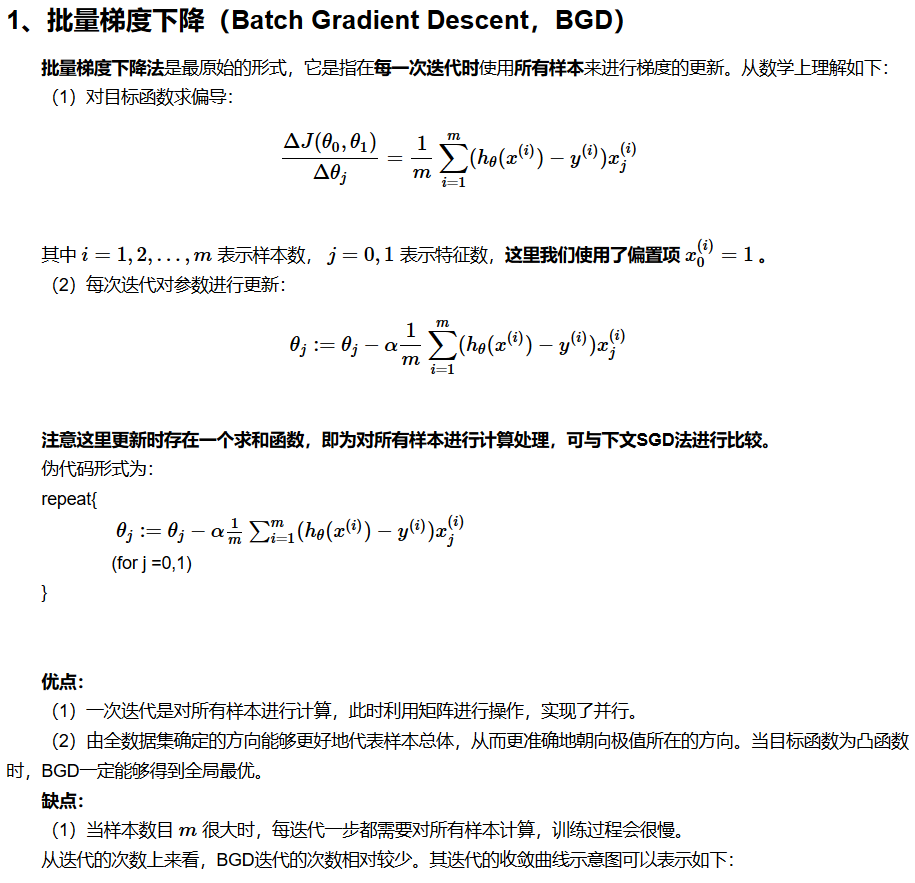
https://www.cnblogs.com/lliuye/p/9451903.html

**批量梯度下降（Batch Gradient Descent）、**

**随机梯度下降（Stochastic Gradient Descent）以及**

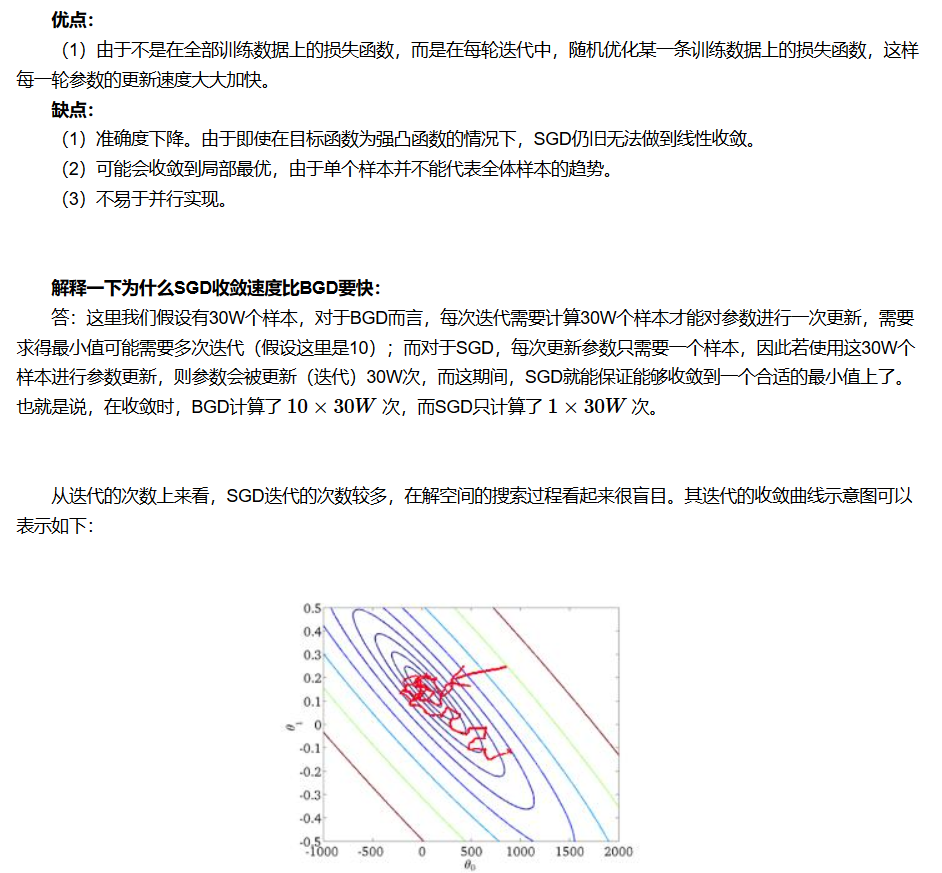
**小批量梯度下降（Mini-Batch Gradient Descent）**。其中小批量梯度下降法也常用在深度学习中进行模型的训练。接下来，我们将对这三种不同的梯度下降法进行理解。





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

**随机梯度下降法**不同于批量梯度下降，随机梯度下降是**每次迭代**使用**一个样本**来对参数进行更新。使得训练速度加快。



## 3、小批量梯度下降（Mini-Batch Gradient Descent, MBGD）

**小批量梯度下降**，是对批量梯度下降以及随机梯度下降的一个折中办法。其思想是：**每次迭代** 使用 \*\* batch\_size\*\* 个样本来对参数进行更新。  
  这里我们假设 *batchsize*=10样本数 *m*=1000 。

**优点：**  
  （1）通过矩阵运算，每次在一个batch上优化神经网络参数并不会比单个数据慢太多。  
  （2）每次使用一个batch可以大大减小收敛所需要的迭代次数，同时可以使收敛到的结果更加接近梯度下降的效果。(比如上例中的30W，设置batch\_size=100时，需要迭代3000次，远小于SGD的30W次)  
  （3）可实现并行化。  
  **缺点：**  
  （1）batch\_size的不当选择可能会带来一些问题。  
  
  
  **batcha\_size的选择带来的影响：**  
  （1）在合理地范围内，增大batch\_size的好处：  
    a. 内存利用率提高了，大矩阵乘法的并行化效率提高。  
    b. 跑完一次 epoch（全数据集）所需的迭代次数减少，对于相同数据量的处理速度进一步加快。  
    c. 在一定范围内，一般来说 Batch\_Size 越大，其确定的下降方向越准，引起训练震荡越小。  
  （2）盲目增大batch\_size的坏处：  
    a. 内存利用率提高了，但是内存容量可能撑不住了。  
    b. 跑完一次 epoch（全数据集）所需的迭代次数减少，要想达到相同的精度，其所花费的时间大大增加了，从而对参数的修正也就显得更加缓慢。  
    c. Batch\_Size 增大到一定程度，其确定的下降方向已经基本不再变化。

