# 0-1损失函数(zero-one loss)

0-1损失是指预测值和目标值不相等为1， 否则为0:

特点：

(1)0-1损失函数直接对应分类判断错误的个数，但是它是一个非凸函数，不太适用.

(2)感知机就是用的这种损失函数。但是相等这个条件太过严格，因此可以放宽条件，即满足时认为相等，

# 绝对值损失函数

绝对值损失函数是计算预测值与目标值的差的绝对值：

# log对数损失函数

log对数损失函数的标准形式如下：

特点：

(1) log对数损失函数能非常好的表征概率分布，在很多场景尤其是多分类，如果需要知道结果属于每个类别的置信度，那它非常适合。

(2)健壮性不强，相比于hinge loss对噪声更敏感。

(3)逻辑回归的损失函数就是log对数损失函数。

# 平方损失函数

平方损失函数标准形式如下：

特点：

(1)经常应用与回归问题

# 指数损失函数（exponential loss）

指数损失函数的标准形式如下：

特点：

(1)对离群点、噪声非常敏感。经常用在AdaBoost算法中。

# Hinge 损失函数

Hinge损失函数标准形式如下：

特点：

(1)hinge损失函数表示如果被分类正确，损失为0，否则损失就为。SVM就是使用这个损失函数。

(2)一般的是预测值，在-1到1之间，是目标值(-1或1)。其含义是，的值在-1和+1之间就可以了，并不鼓励，即并不鼓励分类器过度自信，让某个正确分类的样本距离分割线超过1并不会有任何奖励，从而使分类器可以更专注于整体的误差。

(3) 健壮性相对较高，对异常点、噪声不敏感，但它没太好的概率解释。

# 感知损失(perceptron loss)函数

感知损失函数的标准形式如下：

特点：

(1)是Hinge损失函数的一个变种，Hinge loss对判定边界附近的点(正确端)惩罚力度很高。而perceptron loss只要样本的判定类别正确的话，它就满意，不管其判定边界的距离。它比Hinge loss简单，因为不是max-margin boundary，所以模型的泛化能力没 hinge loss强。

# 交叉熵损失函数 (Cross-entropy loss function)

交叉熵损失函数的标准形式如下:

注意公式中表示样本，表示实际的标签，表示预测的输出，表示样本总数量。

特点：

(1)本质上也是一种对数似然函数，可用于二分类和多分类任务中。

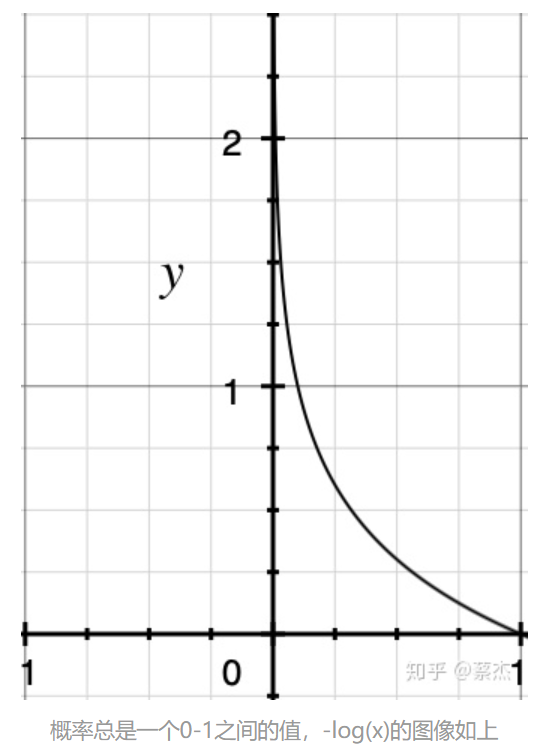
二分类问题中的loss函数（输入数据是softmax或者sigmoid函数的输出）：

多分类问题中的loss函数（输入数据是softmax或者sigmoid函数的输出）：

(2)当使用sigmoid作为激活函数的时候，常用交叉熵损失函数而不用均方误差损失函数，因为它可以完美解决平方损失函数权重更新过慢的问题，具有“误差大的时候，权重更新快；误差小的时候，权重更新慢”的良好性质。

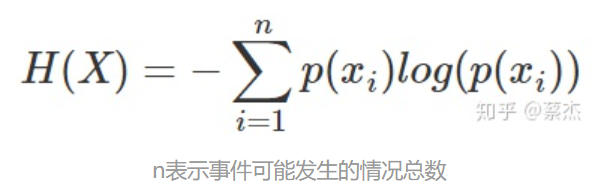
# 信息量

一条信息的信息量大小和它的不确定性有很大的关系。一句话如果需要很多外部信息才能确定，我们就称这句话的信息量比较大。

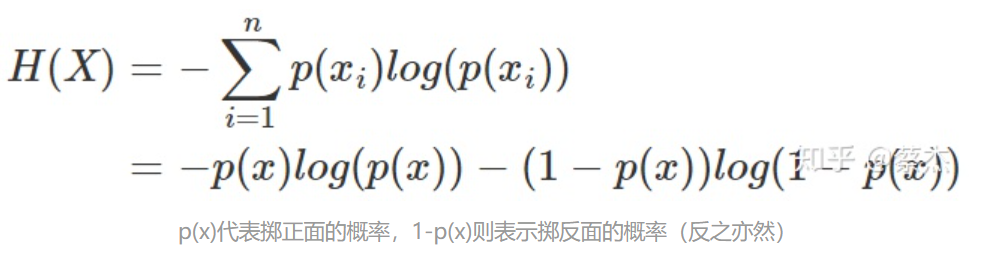


# 熵

信息量是对于单个事件来说的，但是实际情况一件事有很多种发生的可能。**熵是表示随机变量不确定的度量，是对所有可能发生的事件产生的信息量的期望**。

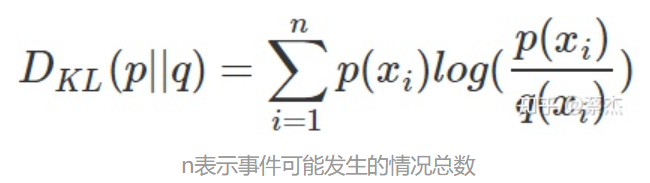


其中一种比较特殊的情况就是掷硬币，只有正、反两种情况，该种情况（二项分布或者0-1分布）熵的计算可以简化如下

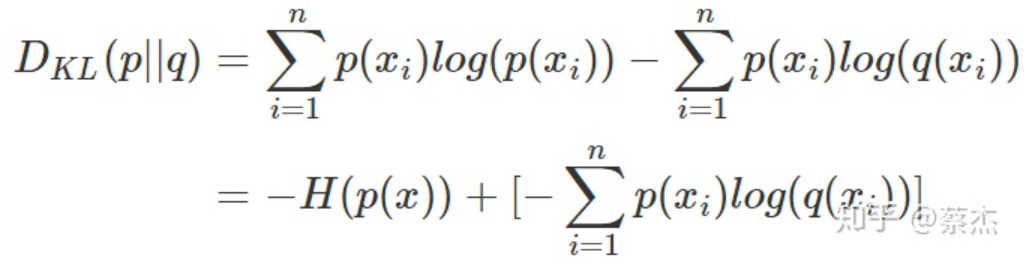


# 相对熵

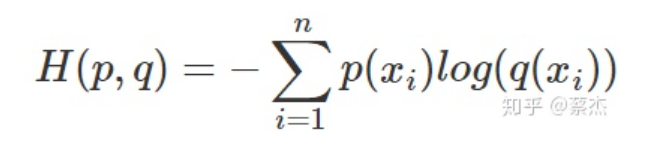
相对熵又称KL散度，用于衡量对于同一个随机变量x的两个分布p(x)和q(x)之间的差异。在机器学习中，p(x)常用于描述样本的真实分布，例如[1,0,0,0]表示样本属于第一类，而q(x)则常常用于表示预测的分布，例如[0.7,0.1,0.1,0.1]。显然使用q(x)来描述样本不如p(x)准确，q(x)需要不断地学习来拟合准确的分布p(x)。



# 交叉熵

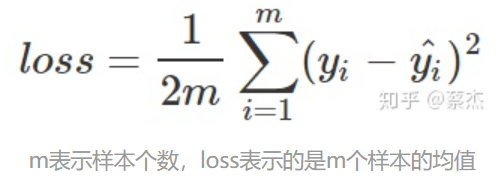


前半部分就是p(x)的熵，后半部分就是我们的交叉熵：

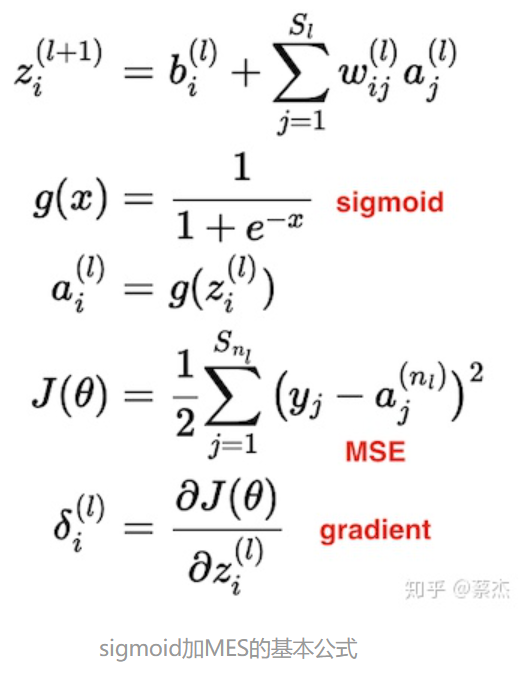


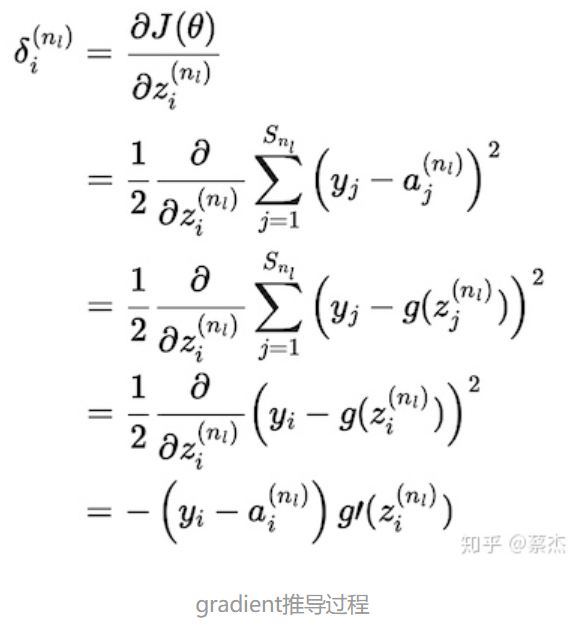
机器学习中，我们常常使用KL散度来评估predict和label之间的差别，但是由于KL散度的前半部分是一个常量，所以我们常常将后半部分的交叉熵作为损失函数，其实二者是一样的。

在回归问题中，我们常常使用均方误差（MSE）作为损失函数，其公式如下：

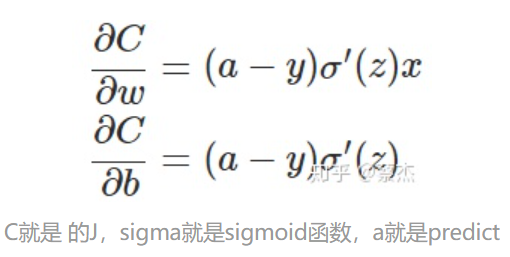


其实这里也比较好理解，因为回归问题要求拟合实际的值，通过MSE衡量预测值和实际值之间的误差，可以通过梯度下降的方法来优化。而不像分类问题，需要一系列的激活函数（sigmoid、softmax）来将预测值映射到0-1之间，这时候再使用MSE的时候就要好好掂量一下了，为啥这么说，请继续看：

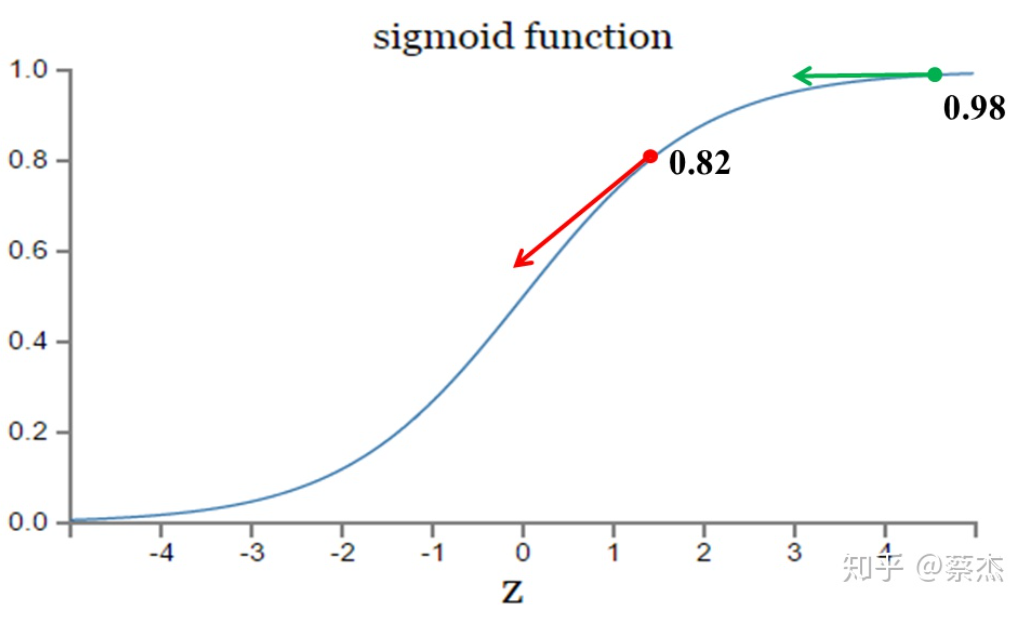




上面复杂的推导过程，其实结论就是下面一张图：

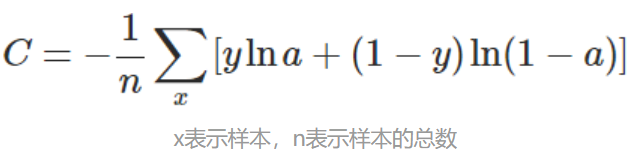


从以上公式可以看出，w和b的梯度跟激活函数的梯度成正比，激活函数的梯度越大，w和b的大小调整得越快，训练收敛得就越快。而我们都知道sigmoid函数长这样：

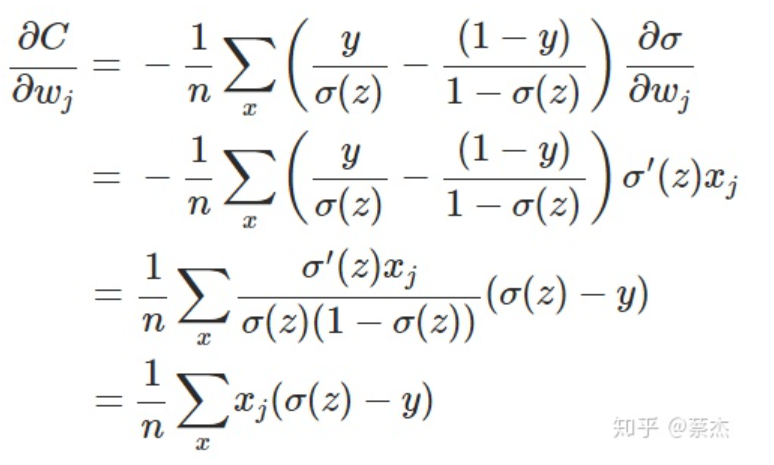


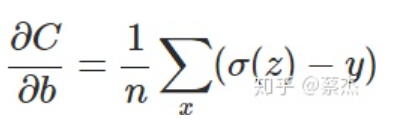
在上图的绿色部分，初始值是0.98，红色部分初始值是0.82，假如真实值是0。直观来看那么0.82下降的速度明显高于0.98，但是明明0.98的误差更大，这就导致了神经网络不能像人一样，误差越大，学习的越快。

但是如果我们把MSE换成交叉熵会怎么样呢？

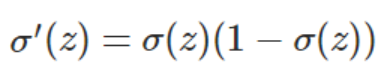


重新计算梯度：





另外sigmoid有一个很好的性质：



我们从结果可以看出梯度中不再含有sigmoid的导数，有的是sigmoid的值和实际值之间的差，也就满足了我们之前所说的错误越大，下降的越快。

这也就是在分类问题中常用cross entropy 而不是 MSE的原因了。