一文看尽深度学习中的各种损失函数

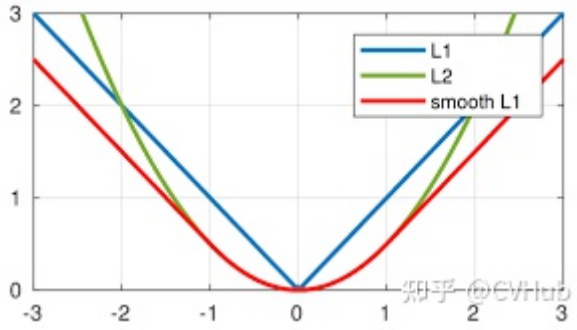
在机器学习中，损失函数是代价函数的一部分，而代价函数则是目标函数的一种类型[1]。

Loss function，即损失函数：用于定义单个训练样本与真实值之间的误差；

Cost function，即代价函数：用于定义单个批次/整个训练集样本与真实值之间的误差；

Objective function，即目标函数：泛指任意可以被优化的函数。

损失函数是用于衡量模型所作出的预测离真实值（**Ground Truth**）之间的偏离程度。 通常，我们都会最小化目标函数，最常用的算法便是“梯度下降法”（**Gradient Descent**）。俗话说，任何事情必然有它的两面性，因此，并没有一种万能的损失函数能够适用于所有的机器学习任务，所以在这里我们需要知道每一种损失函数的优点和局限性，才能更好的利用它们去解决实际的问题。损失函数大致可分为两种：回归损失（针对**连续型**变量）和分类损失（针对**离散型**变量）。



1. 回归损失（Regression Loss）

L1 Loss

也称为**Mean Absolute Error**，即平均绝对误差（**MAE**），它衡量的是预测值与真实值之间距离的平均误差幅度，作用范围为0到正无穷。

优点：对离群点（**Outliers**）或者异常值更具有鲁棒性。

缺点：由图可知其在0点处的导数不连续，使得求解效率低下，导致收敛速度慢；而对于较小的损失值，其梯度也同其他区间损失值的梯度一样大，所以不利于网络的学习。

L2 Loss

也称为**Mean Squred Error**，即均方差（**MSE**），它衡量的是预测值与真实值之间距离的平方和，作用范围同为0到正无穷。

优点：收敛速度快，能够对梯度给予合适的惩罚权重，而不是“一视同仁”，使梯度更新的方向可以更加精确。

缺点：对异常值十分敏感，梯度更新的方向很容易受离群点所主导，不具备鲁棒性。

> 对于L1范数和L2范数，如果异常值对于实际业务非常重要，我们可以使用MSE作为我们的损失函数；另一方面，如果异常值仅仅表示损坏的数据，那我们应该选择MAE作为损失函数。此外，考虑到收敛速度，在大多数的卷积神经网络中（CNN）中，我们通常会选择L2损失。但是，还存在这样一种情形，当你的业务数据中，存在95%的数据其真实值为1000，而剩下5%的数据其真实值为10时，如果你使用MAE去训练模型，则训练出来的模型会偏向于将所有输入数据预测成1000，因为MAE对离群点不敏感，趋向于取中值。而采用MSE去训练模型时，训练出来的模型会偏向于将大多数的输入数据预测成10，因为它对离群点异常敏感。因此，大多数情况这两种回归损失函数并不适用，能否有什么办法可以同时利用这两者的优点呢?

Smooth L1 Loss

即平滑的L1损失（SLL），出自Fast RCNN [7]。SLL通过综合L1和L2损失的优点，在0点处附近采用了L2损失中的平方函数，解决了L1损失在0点处梯度不可导的问题，使其更加平滑易于收敛。此外，在|x|>1的区间上，它又采用了L1损失中的线性函数，使得梯度能够快速下降。

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

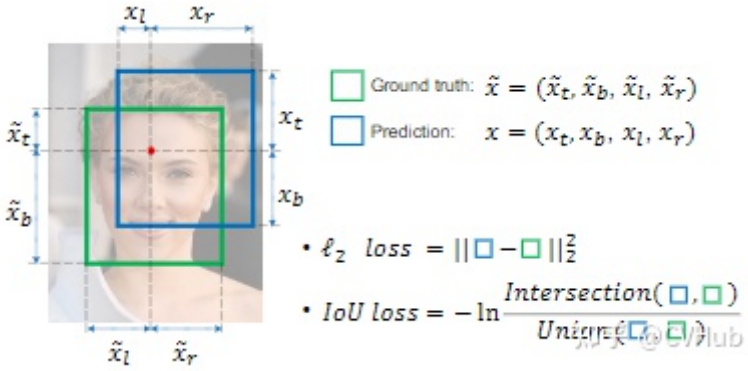
通过对这三个损失函数进行求导可以发现，L1损失的导数为常数，如果不及时调整学习率，那么当值过小时，会导致模型很难收敛到一个较高的精度，而是趋向于一个固定值附近波动。反过来，对于L2损失来说，由于在训练初期值较大时，其导数值也会相应较大，导致训练不稳定。最后，可以发现Smooth L1在训练初期输入数值较大时能够较为稳定在某一个数值，而在后期趋向于收敛时也能够加速梯度的回传，很好的解决了前面两者所存在的问题。

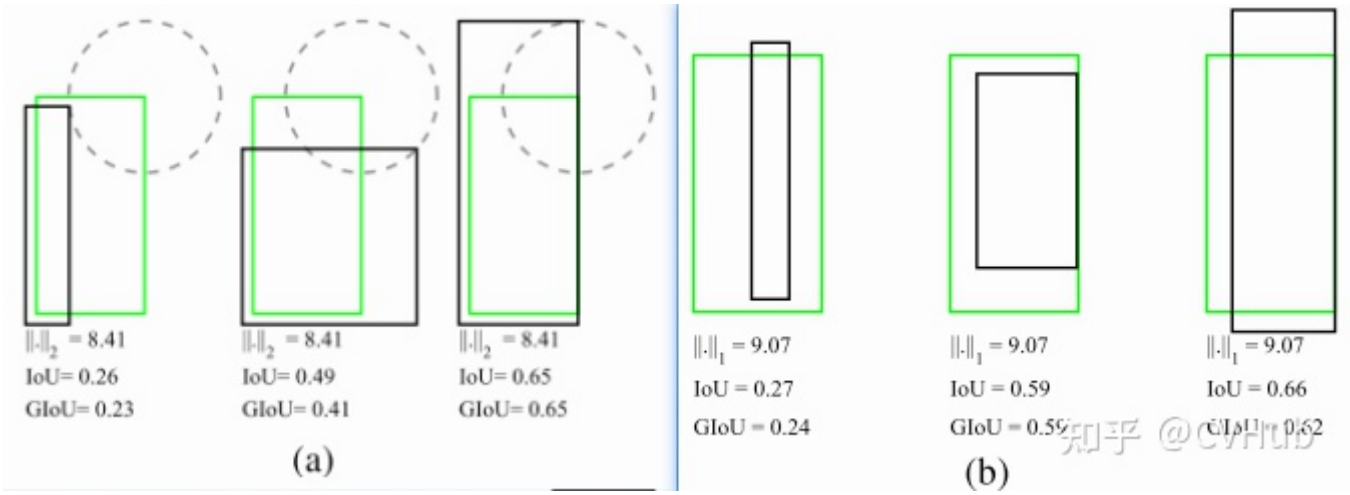
$$\frac{dL_1(x)}{x} = \begin{cases} 1 & if\ x \geq 0 \\ -1 & otherwise \end{cases} \qquad \frac{dL_2(x)}{x} = 2x \qquad \frac{dSmooth_{L_1}(x)}{x} = \begin{cases} x & if\ |x| < 1 \\ \pm 1 & otherwise \end{cases}$$

IoU Loss

即交并比损失，出自UnitBox [8]，由旷视科技于ACM2016首次提出。常规的Lx损失中，都是基于目标边界中的4个坐标点信息之间分别进行回归损失计算的。因此，这些边框信息之间是相互**独立**的。然而，直观上来看，这些边框信息之间必然是存在某种**相关性**的。如下图(a)-(b)分别所示，绿色框代表Ground Truth，黑色框代表Prediction，可以看出，同一个Lx分数，预测框与真实框之间的拟合/重叠程度并不相同，显然**重叠度**越高的预测框是越合理的。IoU损失将候选框的四个边界

信息作为一个**整体**进行回归，从而实现准确、高效的定位，具有很好的尺度不变性。为了解决IoU度量不可导的现象，引入了负Ln范数来间接计算IoU损失。





GIoU Loss

即泛化的IoU损失，全称为Generalized Intersection over Union，由斯坦福学者于CVPR2019年发表的这篇论文 [9]中首次提出。上面我们提到了IoU损失可以解决边界框坐标之间相互独立的问题，考虑这样一种情况，当预测框与真实框之间没有任何重叠时，两个边框的交集（分子）为0，此时IoU损失为0，因此IoU无法算出两者之间的距离（重叠度）。另一方面，由于IoU损失为零，意味着梯度无法有效地反向传播和更新，即出现梯度消失的现象，致使网络无法给出一个**优化的方向**。此外，如下图所示，IoU对**不同方向的边框对齐**也是一脸懵逼的，所计算出来的值都一样。

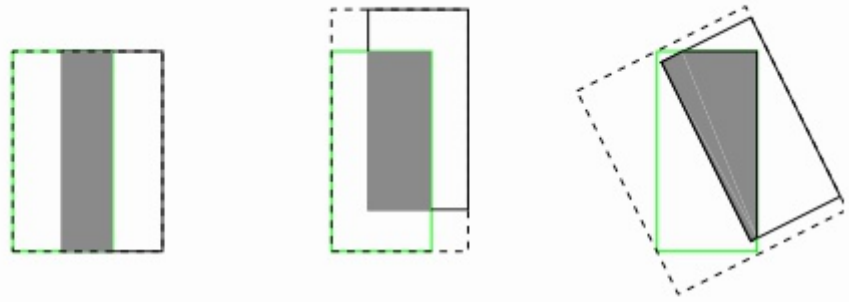


Figure 2. Three different ways of overlap between two rectangles with the exactly same IoU values, *i.e.* $IoU = 0.33$, but different $GIoU$ values, *i.e.* from the left to right $GIoU = 0.33, 0.24$ and -0.1 respectively. $GIoU$ value will be higher for the cases with better aligned orientation.

为了解决以上的问题，如下图公式所示，GIoU通过计算任意两个形状（这里以矩形框为例）A和B的一个**最小闭合凸面**C，然后再计算C中排除掉A和B后的面积占C原始面积的比值，最后再用原始的IoU减去这个比值得到泛化后的IoU值。

Algorithm 1: Generalized Intersection over Union

input : Two arbitrary convex shapes: $A, B \subseteq \mathbb{S} \in \mathbb{R}^n$

output: $GIoU$

1 For A and B, find the smallest enclosing convex object C, where $C \subseteq \mathbb{S} \in \mathbb{R}^n$

2 $$IoU = \frac{|A \cap B|}{|A \cup B|}$$

3 $$GIoU = IoU - \frac{|C \backslash (A \cup B)|}{|C|}$$

GIoU具有IoU所拥有的一切特性，如对称性，三角不等式等。GIoU ≤ IoU，特别地，0 ≤ IoU(A, B) ≤ -1, 而0 ≤ GIoU(A, B) ≤ -1；当两个边框的完全重叠时，此时GIoU = IoU = 1. 而当 |AUB| 与 最小闭合凸面C 的比值趋近为0时，即两个边框不相交的情况下，此时GIoU将渐渐收敛至 -1. 同样地，我们也可以通过一定的方式去计算出两个矩形框之间的GIoU损失，具体计算步骤也非常简单，详情参考原论文。

DIoU Loss

即距离IoU损失，全称为Distance-IoU loss，由天津大学数学学院研究人员于AAAI2020所发表的这篇论文 [10]中首次提出。上面我们谈到GIoU通过引入最小闭合凸面来解决IoU无法对不重叠边框的优化问题。但是，其仍然存在两大局限性：**边框回归还不够精确 & 收敛速度缓慢** 。考虑下图

这种情况，当目标框**完全包含**预测框时，此时GIoU**退化**为IoU。显然，我们希望的预测是最右边这种情况。因此，作者通过计算两个边框之间的**中心点归一化距离**，从而更好的优化这种情况。

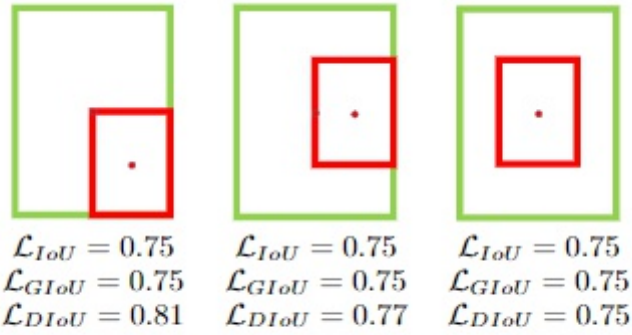
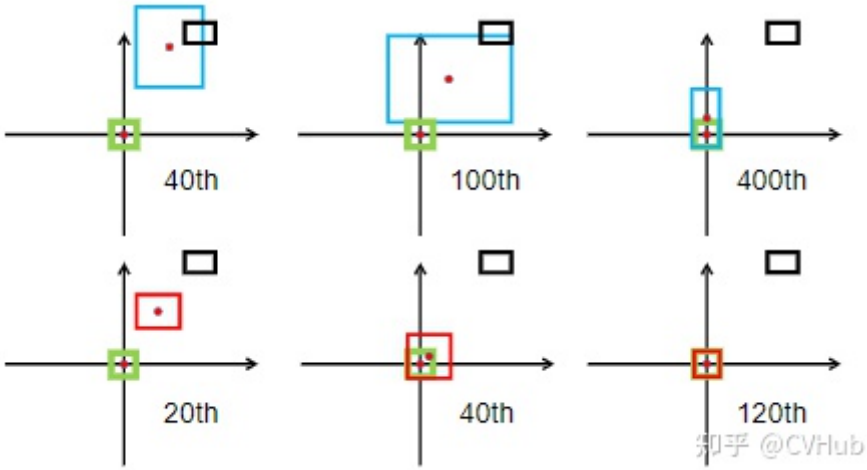


Figure 2: GIoU loss degrades to IoU loss for these cases, while our DIoU loss is still distinguishable. Green and red denote target box and predicted box respectively.

下图表示的是GIoU损失（第一行）和DIoU损失（第二行）的一个训练过程收敛情况。其中绿色框为目标边框，黑色框为锚框，蓝色框和红色框则分别表示使用GIoU损失和DIoU损失所得到的预测框。可以发现，GIoU损失一般会增加预测框的大小使其能和目标框重叠，而DIoU损失则直接使目标框和预测框之间的中心点归一化距离最小，即让预测框的中心快速的向目标中心收敛。

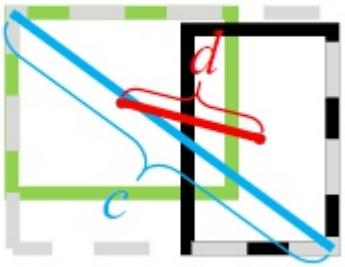


左图给出这三个IoU损失所对应的计算公式。对于DIoU来说，如图右所示，其惩罚项由两部分构成：分子为目标框和预测框中心点之间的欧式距离；分母为两个框最小外接矩形框的两个对角线距离。因此， 直接优化两个点之间的距离会使得模型**收敛得更快，同时又能够在两个边框不重叠的情况下给出一个优化的方向。**

$$\mathcal{L}_{IoU} = 1 - \frac{|B \cap B^{gt}|}{|B \cup B^{gt}|}.$$

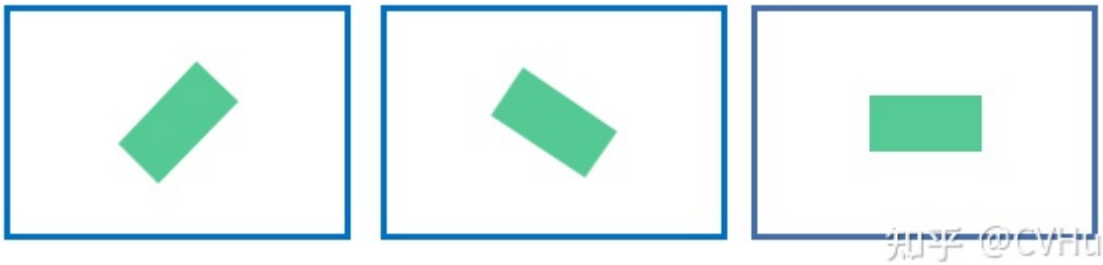
$$\mathcal{L}_{GIoU} = 1 - IoU + \frac{|C - B \cup B^{gt}|}{|C|},$$

$$\mathcal{L}_{DIoU} = 1 - IoU + \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2}.$$



CIoU Loss

即完整IoU损失，全称为Complete IoU loss，与DIoU出自同一篇论文。上面我们提到GIoU存在两个缺陷，DIoU的提出解决了其实一个缺陷，即收敛速度的问题。而一个好的边框回归损失应该同时考虑三个重要的几何因素，即重叠面积（**Overlap area**）、中心点距离（**Central point distance**）和高宽比（**Aspect ratio**）。GIoU考虑到了重叠面积的问题，DIoU考虑到了重叠面积和中心点距离的问题，CIoU则在此基础上进一步的考虑到了高宽比的问题。



CIoU的计算公式如下所示，可以看出，其在DIoU的基础上加多了一个惩罚项αv。其中α为权重为正数的重叠面积平衡因子，在回归中被赋与更高的优先级，特别是在两个边框不重叠的情况下；而v则用于测量宽高比的一致性。

$$\mathcal{L}_{CIoU} = 1 - IoU + \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2} + \alpha v.$$

$$\alpha = \frac{v}{(1 - IoU) + v},$$

$$v = \frac{4}{\pi^2}(arctan\frac{w^{gt}}{h^{gt}} - arctan\frac{w}{h})^2$$

F-EIoU Loss

Focal and Efficient IoU Loss是由华南理工大学学者最近提出的一篇关于目标检测损失函数的论文，文章主要的贡献是提升网络收敛速度和目标定位精度。目前检测任务的损失函数主要有两个缺

点：(1)无法有效地描述边界框回归的目标，导致收敛速度慢以及回归结果不准确（2）忽略了边界框回归中不平衡的问题。

F-EIou loss首先提出了一种有效的交并集（IOU）损失，它可以准确地测量边界框回归中的**重叠面积**、**中心点**和**边长**三个几何因素的差异：

$$L_{EIOU} = L_{IOU} + L_{dis} + L_{asp}$$
$$= 1 - IOU + \frac{\rho^2(\mathbf{b}, \mathbf{b^{gt}})}{c^2} + \frac{\rho^2(w, w^{gt})}{C_w^2} + \frac{\rho^2(h, h^{gt})}{C_h^2}$$

其次，基于对有效样本挖掘问题（EEM）的探讨，提出了Focal loss的回归版本，以使回归过程中专注于高质量的锚框：

$$L_f(x) = \begin{cases} -\frac{\alpha x^2(2\ln(\beta x)-1)}{4}, & 0 < x \leq 1; 1/e \leq \beta \leq 1 \\ -\alpha \ln(\beta) x + C, & x > 1; 1/e \leq \beta \leq 1, \end{cases}$$

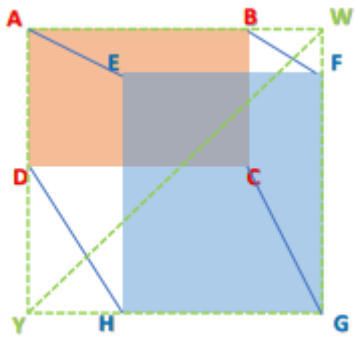
最后，将以上两个部分结合起来得到Focal-EIou Loss：

$$L_{\text{Focal-EIOU}} = \frac{\sum_{i=1}^{n} W_i \cdot L_{EIOU_i}}{\sum_{i=1}^{n} W_i}$$

其中，通过加入每个batch的权重和来避免网络在早期训练阶段收敛慢的问题。

CDIoU Loss

Control Distance IoU Loss是由同济大学学者提出的，文章的主要贡献是在几乎不增强计算量的前提下有效提升了边界框回归的精准度。目前检测领域主要两大问题：（1）SOTA算法虽然有效但计算成本高（2）边界框回归损失函数设计不够合理。



文章首先提出了一种对于Region Propasl（RP）和Ground Truth（GT）之间的新评估方式，即CDIoU。可以发现，它虽然没有直接中心点距离和长宽比，但最终的计算结果是有反应出RP和GT的差异。计算公式如下：

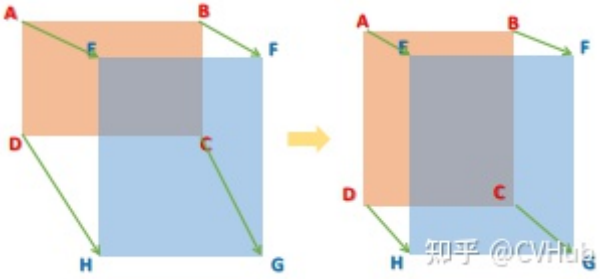
$$\begin{aligned} \text{diou} &= \frac{\|RP - GT\|_2}{4\text{MBR}'s \text{ diagonal}} \\ &= \frac{AE + BF + CG + DH}{4WY} \end{aligned}$$

$$CDIoU = IoU + \lambda(1 - \text{diou})$$

对比以往直接计算中心点距离或是形状相似性的损失函数，CDIoU能更合理地评估RP和GT的差异并且有效地降低计算成本。然后，根据上述的公式，CDIoU Loss可以定义为：

$$\mathcal{L}_{CDIoU} = \mathcal{L}_{IoU_s} + \text{diou}$$

通过观察这个公式，可以直观地感受到，在权重迭代过程中，模型不断地将RP的四个顶点拉向GT的四个顶点，直到它们重叠为止，如下图所示：



2. 分类损失

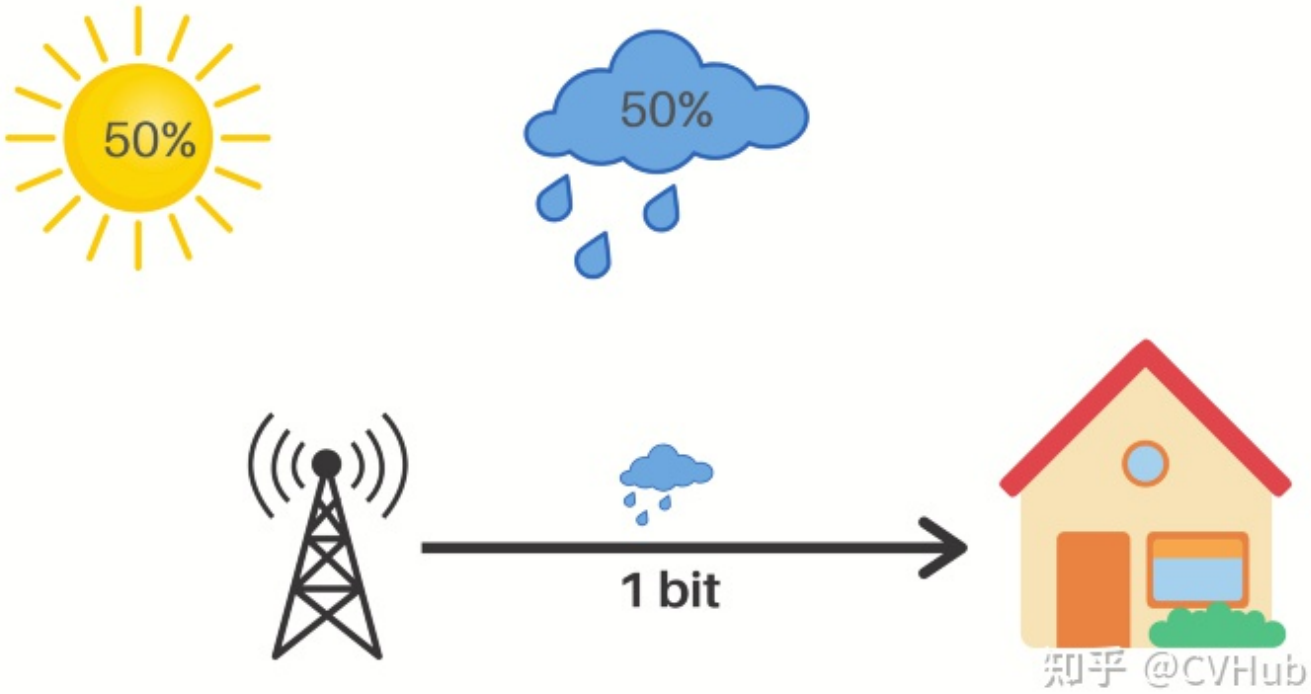
Entropy

即“熵”，**熵**的概念最早起源于物理学，用于度量一个热力学系统的**无序**程度。但更常见的，在信息论里面， 熵是用于描述对**不确定性**的度量。所以，这个概念可以延伸到深度神经网络中，比如我们的模型在做分类时，其实也是在做一个判断一个物体到底是不是属于某个类别。因此，在正式介绍分类损失函数时，我们必须先了解熵的概念。

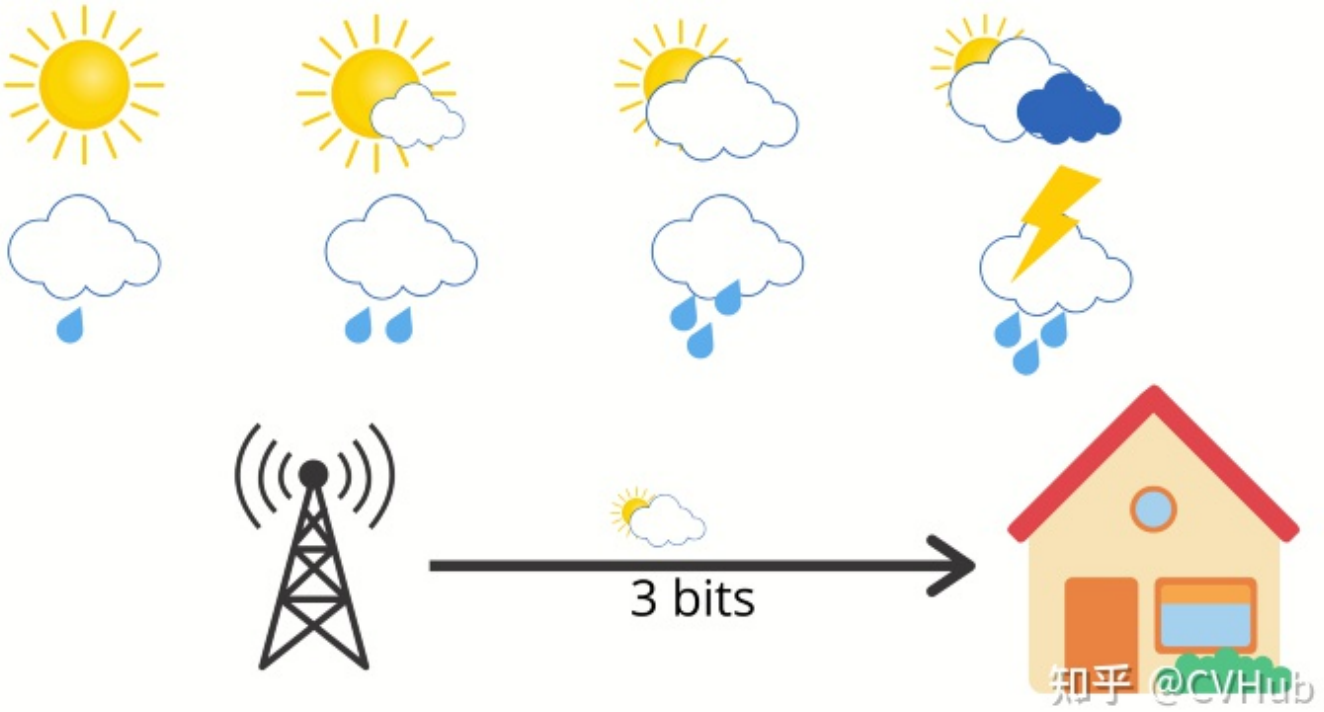
数字化时代，信息都是由Bits(0和1)组成的。在通信时，有些位是**有用（useful）**的信息，有些位则是**冗余（redundant）**的信息，有些位甚至是**错误（error）**的信息，等等。当我们传达信息时，我们希望尽可能多地向接收者传递有用的信息。

> 传输1比特的信息意味着将接收者的不确定性降低2倍。 —— 香浓

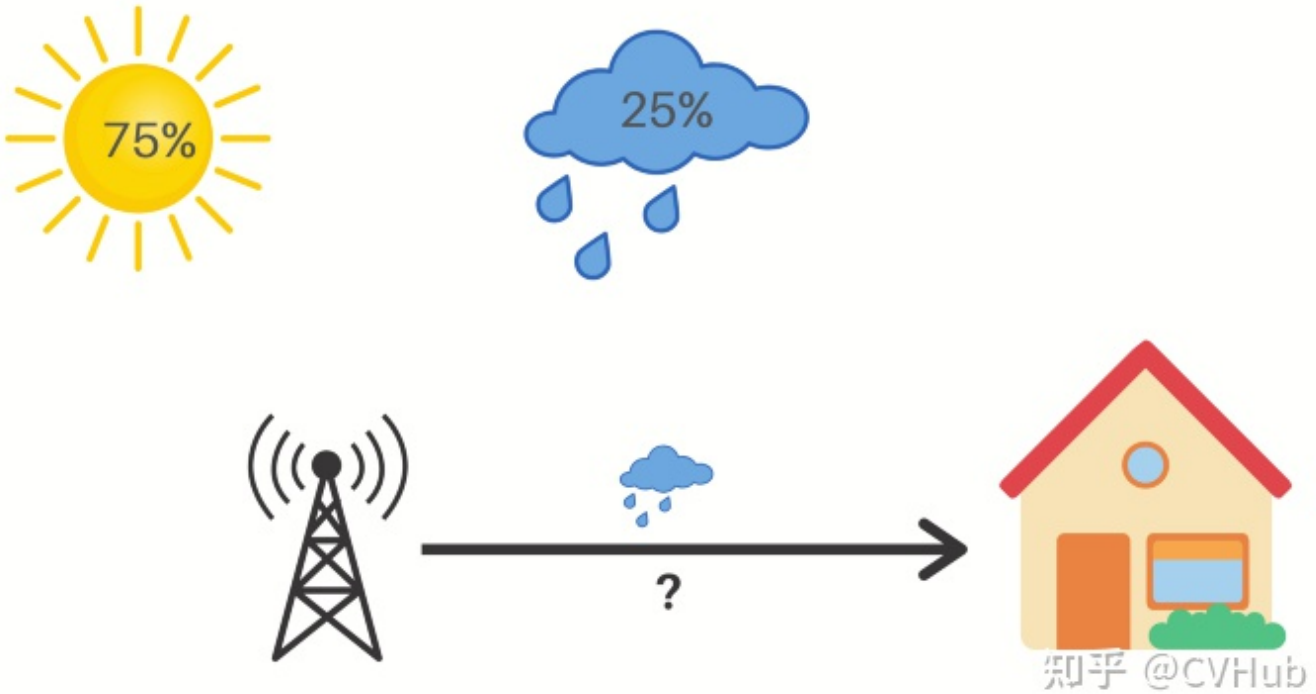
下面以一个天气预报的例子为例，形象化的讲解熵到底尤为何物？假设一个地方的天气是随机的，每天有**50%**的机会是晴天或雨天。



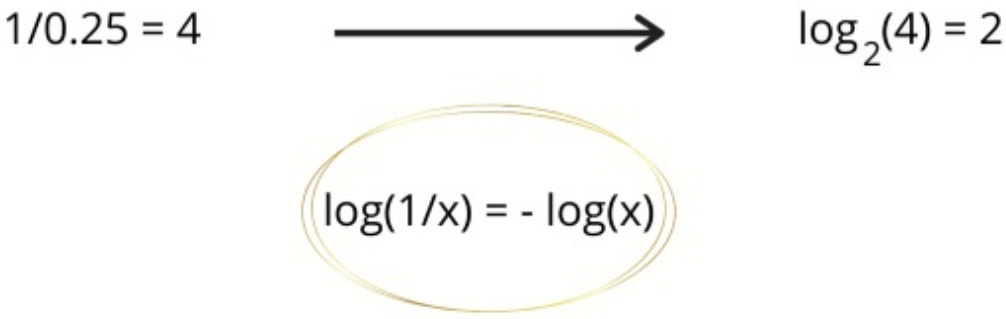
现在，如果气象站告诉您明天将要下雨，那么他们将**不确定性**降低了2倍。起初，有两种同样可能的可能性，但是在收到气象站的更新信息后，我们只有一种 。 在这里，气象站向我们发送了一点**有用**的信息，无论他们如何编码这些信息，这都是事实。即使发送的消息是雨天的，每个字符占一个字节，消息的总大小为40位，但它们仍然**只通信1位的有用信息。**现在，我们假设天气有8种可能状态，且都是等可能的。



那么，当气象站为您提供第二天的天气时，它们会将您的不确定性降低了**8**倍。由于每个事件的发生几率为1/8，因此**降低因子**为8。但如果这些可能性不是等概率的呢？比如，75%的机会是晴天，25%的机会是雨天。



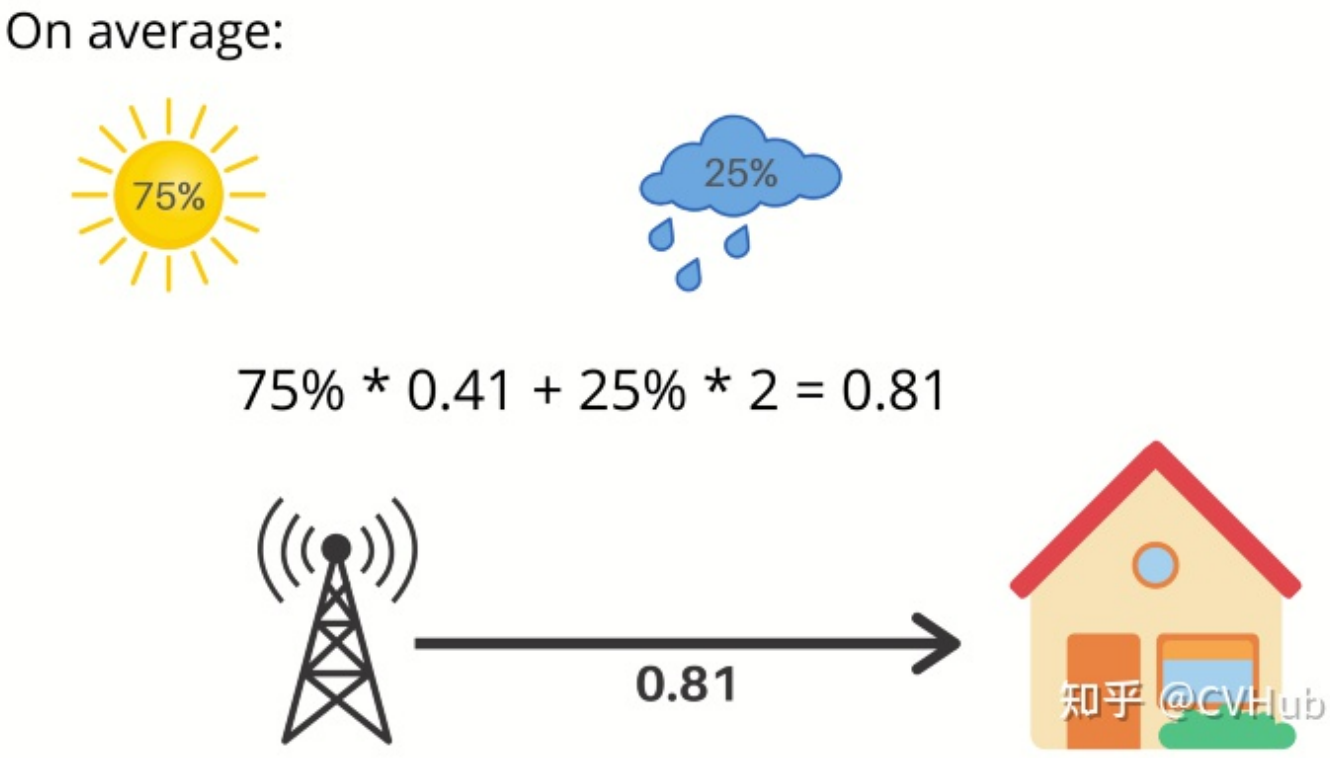
现在，如果气象台说第二天会下雨，那么你的不确定性就降低了**4**倍，也就是**2比特**的信息。**不确定性的减少就是事件概率的倒数。**在这种情况下，25%的倒数是4，log(4)以2为底得到2。因此，我们得到了**2位有用的信息。**



log(1/0.25) = - log(0.25) = 2

log(1/0.75) = - log(0.75) = 0.41

如果气象站说第二天是晴天，那么我们得到**0.41**比特的有用信息。那么，我们**平均**能从气象站得到多少信息呢？明天是晴天的概率是75%这就给了你0.41比特的信息而明天是雨天的概率是25%这就给了你2比特的信息，这就对应我们平均每天从气象站得到0.81比特的信息。



我们刚刚所计算出来的就叫做熵，它可以很好地描述事件的不确定性。它是由以下公式给出：

$$Entropy, H(p) = -\sum p(i) * \log(p(i))$$

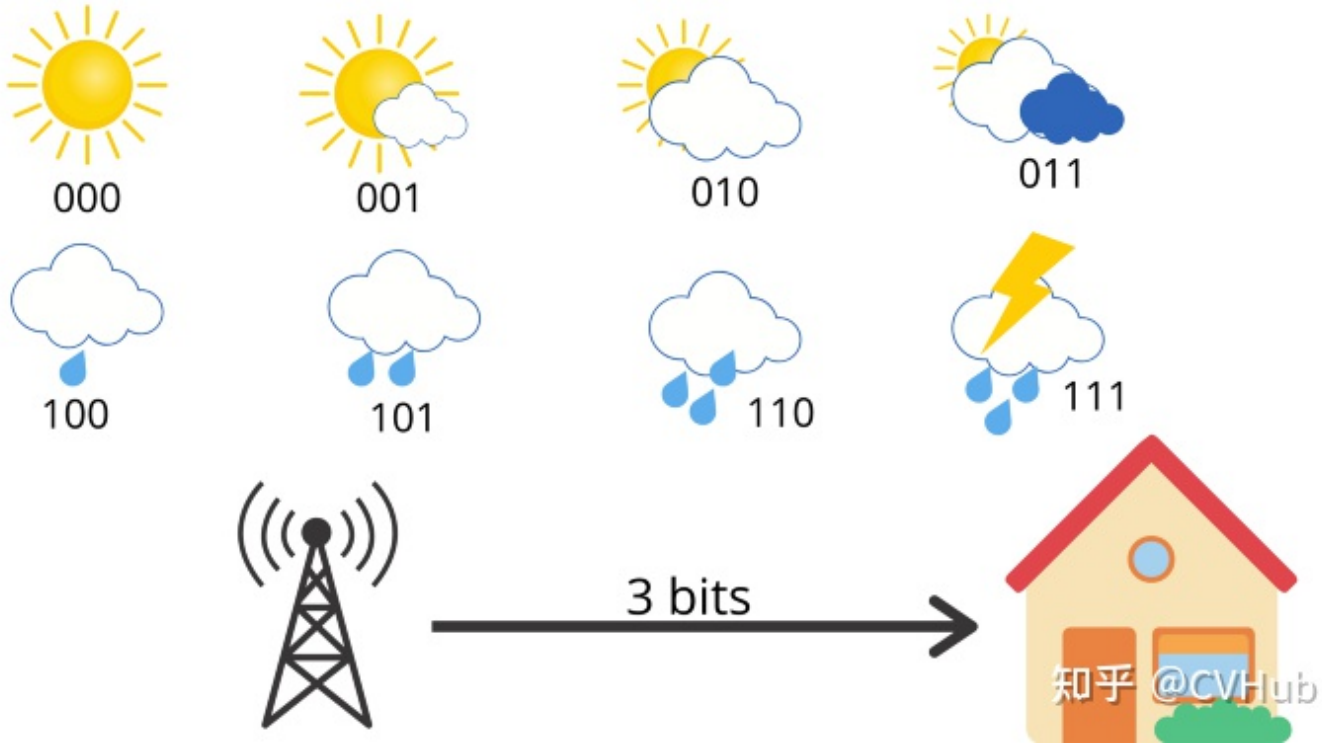
它衡量的是你每天了解天气情况时所得到的**平均信息量**。一般来说，它给出了给定**概率分布p**中样本值的平均信息量它告诉我们概率分布有**多不可预测**。如果我们住在沙漠中央，那里每天都是阳光

灿烂的，平均来说，我们不会每天从气象站得到很多信息。熵会接近于零。另一方面，如果天气变化很大，熵就会大得多。

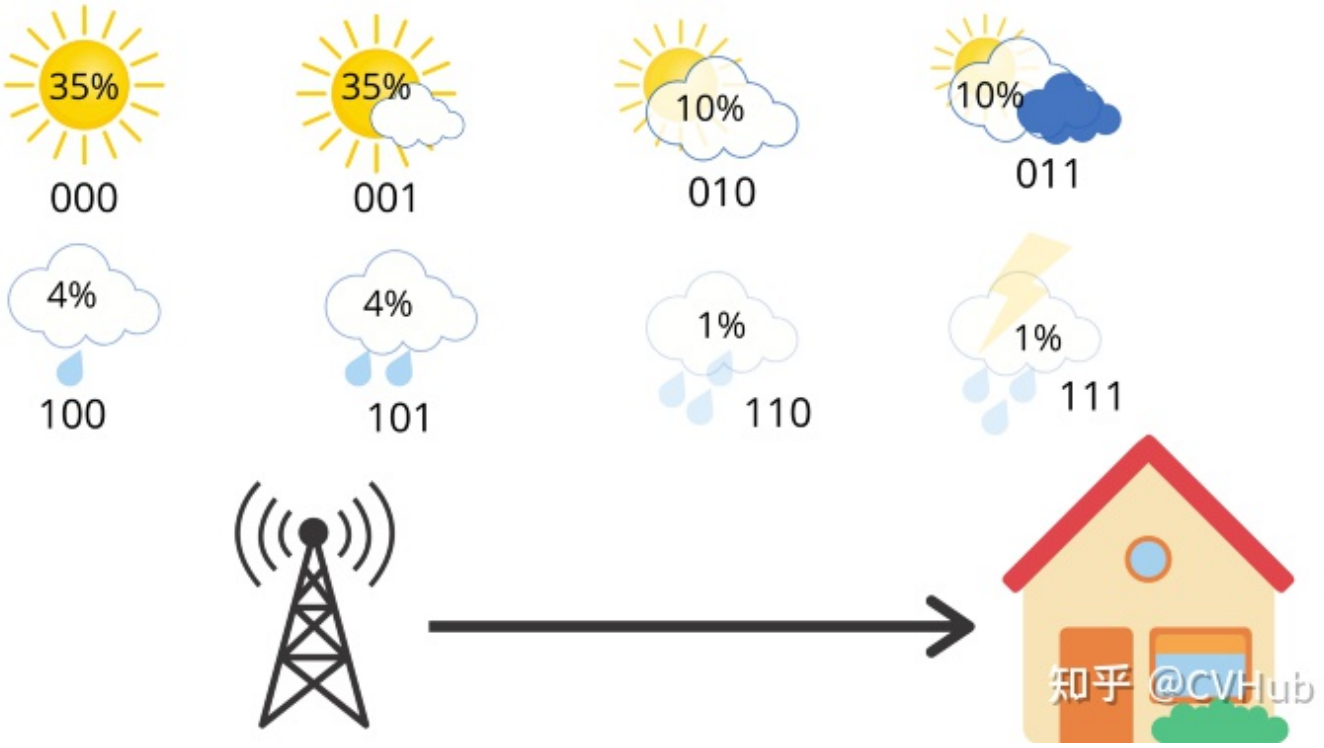
总的来说：一个事件的不确定性就越大，其信息量越大，它的熵值就越高。比如**CVHub**今日宣布上市。相反，如果一个时间的不确定性越小，其信息量越小，它的熵值就越低。比如**CVHub**今天又增加了一个读者。

Cross Entropy

现在，让我们讨论一下**交叉熵**。它只是**平均信息长度**。考虑同样的例子，8种可能的天气条件，所有都是等可能的，每一种都可以用3位编码 [2^3=8]。



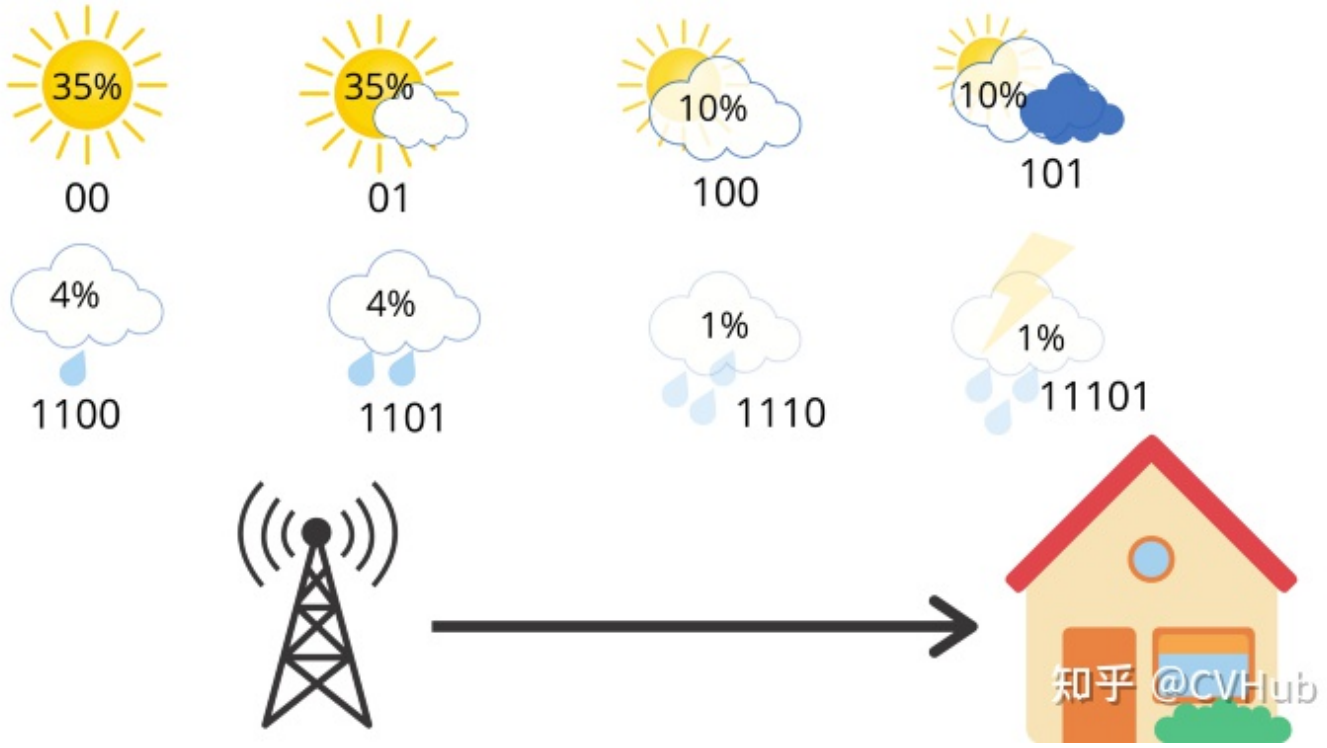
这里的**平均信息长度**是3，这就是**交叉熵**。但是现在，假设你住在一个阳光充足的地区，那里的天气概率分布是这样的：



即每天有35%的机会出现晴天，只有1%的机会出现雷雨。我们可以计算这个概率分布的熵，我们得到**2.23bits的熵**，具体计算公式如下：

$$\text{Entropy} = -(0.35 * \log(0.35) + 0.35 * \log(0.35) + 0.1 * \log(0.1) + 0.1 * \log(0.1) + 0.04 * \log(0.04) + 0.04 * \log(0.04) + 0.01 * \log(0.01) + 0.01 * \log(0.01))$$

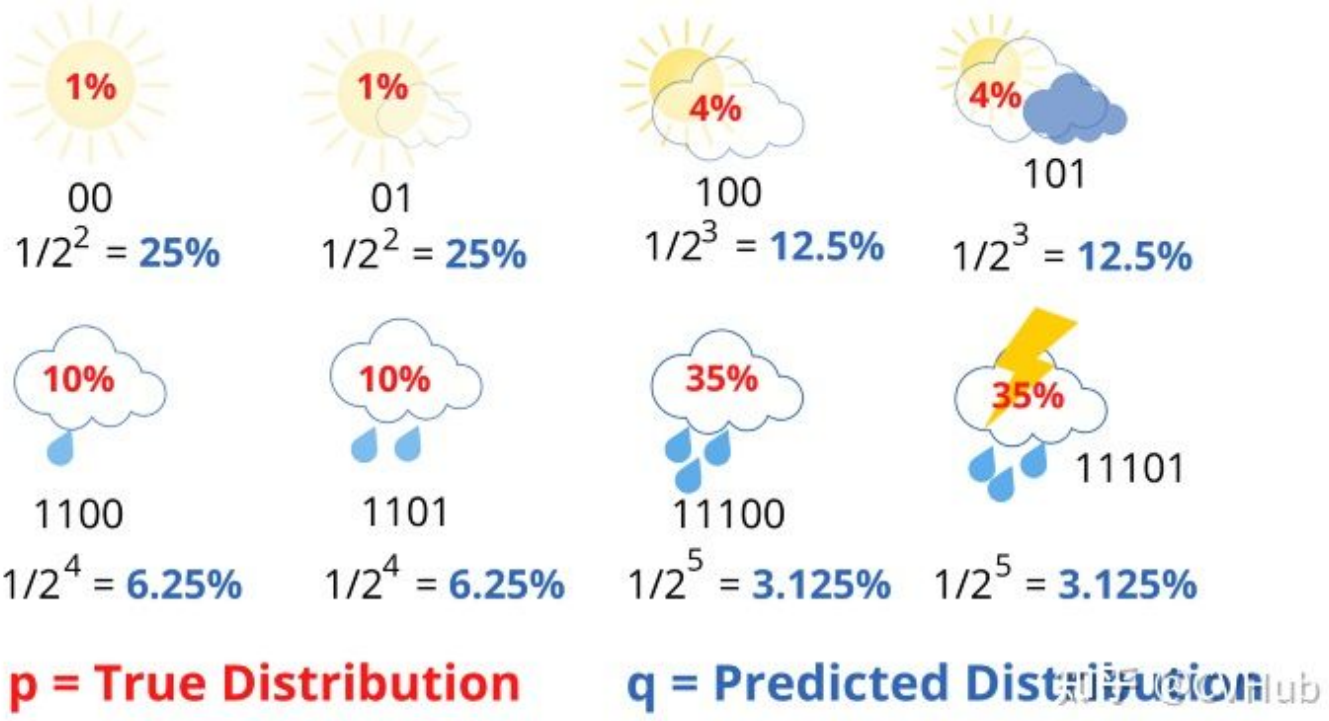
所以，平均来说，气象站发送了3个比特，但接收者只得到**2.23**个比特**有用**的信息。但是，我们可以做得更好。例如，让我们像这样更改编码方式：



现在，我们只使用2位用于表示晴天或部分晴天，使用3位用于多云和大部分多云，使用4位用于表示中雨和小雨，使用5位用于大雨和雷暴。天气的编码方式是明确的，并且如果你链接多条消息，则只有一种方法可以解释位的顺序。 例如，01100只能表示部分晴天（01），然后是小雨（100）。 因此，如果我们计算该站每天发送的平均比特数，则可以得出：

$$35\%*2 + 35\%*2 + 10\%*3 + 10\%*3 + 4\%*4 + 4\%*4 + 1\%*5 + 1\%*5 = 2.42 \text{ bits}$$

我们将得到4.58位。大约是熵的两倍。 平均而言，该站发送4.58位，但只有2.23位对接收者有用。 每条消息发送的信息量是必要信息的两倍。 这是因为我们使用的编码对天气分布做出了一些隐含的假设。 例如，当我们在晴天使用2位消息时，我们隐式地预测晴天的概率为25%。以同样的方式，我们计算所有天气情况。



分母中2的幂对应于用于传输消息的比特数。很明显，**预测分布q和真实分布p有很大不同。**现在我们可以把交叉熵表示成真实概率分布p的函数和预测概率分布q的函数：

$$CrossEntropy, H(p,q) = -\sum p(i)\log(q(i))$$

注意，这里对数的底数为2。

K-L Divergence

即KL散度。对于交叉熵损失，除了我们在这里使用**预测概率的对数（log(q(i))）**外，它看起来与上面**熵**的方程非常相似。如果我们的预测是完美的，那就是**预测分布等于真实分布**，此时**交叉熵就等于熵。** 但是，如果分布不同，则**交叉熵将比熵大一些位数。 交叉熵超过熵的量称为相对熵**，或更普遍地称为**库尔贝克-莱布里埃发散度（KL Divergence）**。总结如下：

$$CrossEntropy = Entropy + KL - Divergence$$

$$D_{KL}(p||q) = H(p,q) - H(p) = -\sum_i p_i \log(q_i) - (-\sum_i p_i \log(p_i))$$

$$D_{KL}(p||q) = -\sum_i p_i \log(q_i) + \sum_i p_i \log(p_i) = \sum_i p_i \log \frac{p_i}{q_i}$$

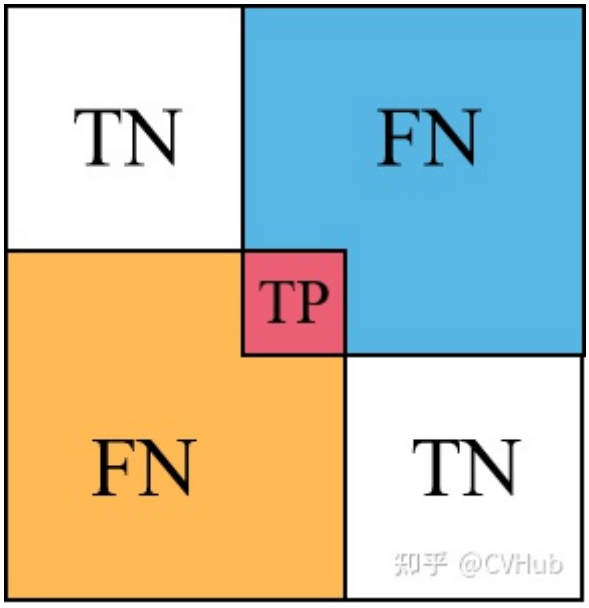
接上面的例子，我们便可以顺便算出： **KL散度 = 交叉熵 - 熵 = 4.58 - 2.23 = 2.35（Bits）**。通常来说，一般分类损失最常用的损失函数之一便是交叉熵损失。假设我们当前做一个3个类别的图像分类任务，如猫、狗、猪。给定一张输入图片其真实类别是猫，模型通过训练用Softmax分类后的输出结果为： {"cat": 0.3, "dog": 0.45, "pig": 0.25}，那么此时交叉熵为：-1 * log(0.3) = 1.203。当输出结果为： {"cat": 0.5, "dog": 0.3, "pig": 0.2}时，交叉熵为：-1 * log(0.5) = 0.301。可以发现，当真实类别的预测概率接近于0时，损失会变得非常大。但是当预测值接近真实值时，损失将接近0。

Dice Loss

即骰子损失，出自V-Net [3]，是一种用于评估两个样本之间相似性度量的函数，取值范围为0~1，值越大表示两个值的相似度越高，其基本定义（二分类）如下：

$$L_{dice} = 1 - \frac{2 \cdot |X \cap Y|}{|X| + |Y|} = 1 - \frac{2 \cdot TP}{2 \cdot TP + FP + FN}$$

其中，|X∩Y|表示X和Y之间的交集，|X|和|Y|分别表示集合X和Y中像素点的个数，分子乘于2保证域值范围在0~1之间，因为分母相加时会计算多一次重叠区间，如下图：



从右边公式也可以看出，其实Dice系数是等价于F1分数的，优化Dice等价于优化F1值。此外，为了防止分母项为0，一般我们会在分子和分母处同时加入一个很小的数作为平滑系数，也称为拉普拉斯平滑项。Dice损失由以下两个主要特性：

有益于正负样本不均衡的情况，侧重于对前景的挖掘；
训练过程中，在有较多小目标的情况下容易出现振荡；
极端情况下会出现梯度饱和的情况。

所以一般来说，我们都会结合交叉熵损失或者其他分类损失一同进行优化。

Focal Loss

焦点损失，出自何凯明的《Focal Loss for Dense Object Detection》[4]，出发点是解决目标检测领域中one-stage算法如YOLO系列算法准确率不高的问题。作者认为样本的**类别不均衡**（比如前景和背景）是导致这个问题的主要原因。比如在很多输入图片中，我们利用网格去划分小窗口，大多数的窗口是不包含目标的。如此一来，如果我们直接运用原始的交叉熵损失，那么负样本所占比例会非常大，主导梯度的优化方向，即网络会偏向于将前景预测为背景。即使我们可以使用OHEM（在线困难样本挖掘）算法来处理不均衡的问题，虽然其增加了误分类样本的权重，但也容易忽略掉易分类样本。而Focal loss则是聚焦于训练一个困难样本的稀疏集，通过直接在标准的交叉熵损失基础上做改进，引进了两个惩罚因子，来减少易分类样本的权重，使得模型在训练过程中更专注于困难样本。其基本定义如下：

$$\mathrm{FL}\left(p, \hat{p}\right) = -\left(\alpha(1-\hat{p})^{\gamma} p \log(\hat{p}) + (1-\alpha)\hat{p}^{\gamma}(1-p)\log(1-\hat{p})\right)$$

其中：

参数 α 和 (1-α) 分别用于控制正/负样本的比例，其取值范围为[0, 1]。α的取值一般可通过交叉验

证来选择合适的值。
参数 γ 称为聚焦参数，其取值范围为[0, +∞)，目的是通过**减少易分类**样本的权重，从而使模型在训练时更专注于困难样本。 当 γ = 0 时，Focal Loss就退化为交叉熵损失，γ 越大，对易分类样本的惩罚力度就越大。

实验中，作者取（α=0.25，γ=0.2）的效果最好，具体还需要根据任务的情况调整。由此可见，应用Focal-loss也会引入多了两个超参数需要调整，而一般来说很需要经验才能调好。

Tversky loss

Tversky loss，发表于CVPR 2018上的一篇《Tversky loss function for image segmentation using 3D fully convolutional deep networks》文章 [5]，是根据Tversky 等人于1997年发表的《Features of Similarity》文章 [6] 所提出的Tversky指数所改造的。Tversky系数主要用于描述两个特征（集合）之间的相似度，其定义如下：

$$T(A,B)=\frac{|A\bigcap B|}{|A\bigcap B|+\alpha|A-B|+\beta|B-A|}$$

由上可知，它是结合了Dice系数（F1-score）以及Jaccard系数（IoU）的一种广义形式，如：

当 α = β = 0.5时，此时Tversky loss便退化为Dice系数（分子分母同乘于2）
当 α = β = 1时，此时Tversky loss便退化为Jaccard系数（交并比）

因此，我们只需控制 α 和 β 便可以控制**假阴性**和**假阳性**之间的平衡。比如在医学领域我们要检测肿瘤时，更多时候我们是希望Recall值（查全率，也称为灵敏度或召回率）更高，因为我们不希望说将肿瘤检测为非肿瘤，即假阴性。因此，我们可以通过增大 β 的取值，来提高网络对肿瘤检测的灵敏度。其中，α + β 的取值我们一般会令其1。

总结

总的来说，损失函数的形式千变万化，但追究溯源还是万变不离其宗。其本质便是给出一个能较全面合理的描述两个特征或集合之间的相似性度量或距离度量，针对某些特定的情况，如类别不平衡等，给予适当的惩罚因子进行权重的加减。大多数的损失都是基于最原始的损失一步步改进的，或提出更一般的形式，或提出更加具体实例化的形式。