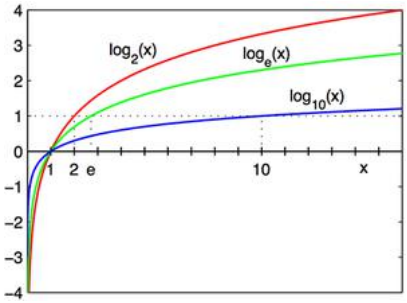
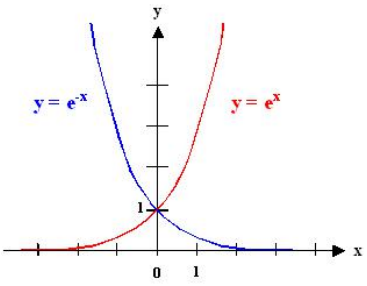
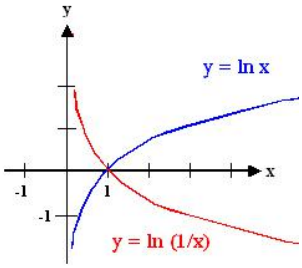
损失函数







距离 Loss

L1, L2, Smooth L1 Loss

均方误差 MSE(L2 Loss)

$$MSE = \frac{\sum_{i=1}^{n}(f_{xi} - y_i)^2}{n}$$

平均绝对误差 MAE(L1 Loss)

$$MAE = \frac{\sum_{n=1}^{n} | f(x_i) - y_i|}{n}$$

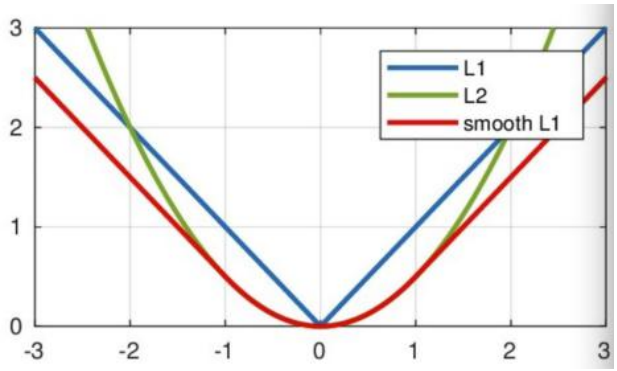
Smooth L1 Loss

$$\text{Smooth}L_1(x) = \begin{cases} 0.5x^2 & if \mid x \mid < 1 \\ \mid x \mid -0.5 & otherwise \end{cases}$$

$$\frac{\partial L_2(x)}{\partial x} = 2x$$

$$\frac{\partial L_1(x)}{\partial x} = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

$$\frac{\partial smooth_{L_1}(x)}{\partial x} = \begin{cases} x & if \mid x \mid < 1 \\ \pm 1 & otherwise \end{cases}$$



L1 loss 在底部需要精修的时候，无法改变梯度；L2 loss 在一开始梯度太大，容易导致梯度爆炸。Smooth L1 结合两者。

面积 Loss

IoU

L1，L2，Smooth L1 loss 都不适合 bbox。因为这三个指标无法反映出坐标点之间空间的关系。L2 loss 相同的情况下，可以有无数种坐标的分布方式，因此学习出来的方框极不稳定。而 IOU 比较好的解决了这一点。IOU 不受两个物体尺度大小的影响。

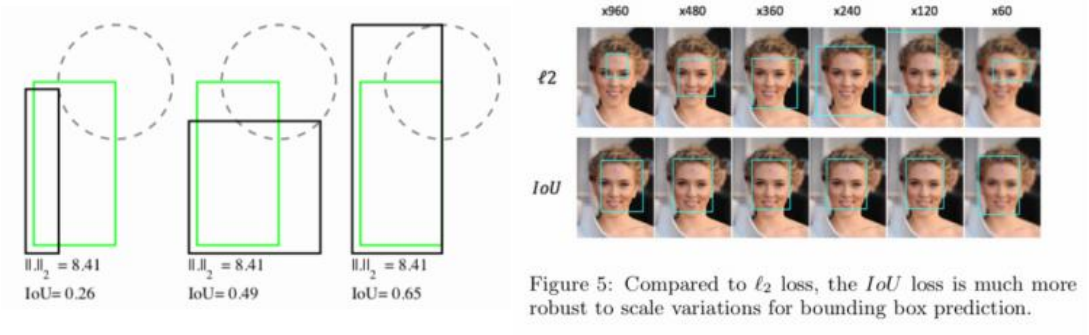
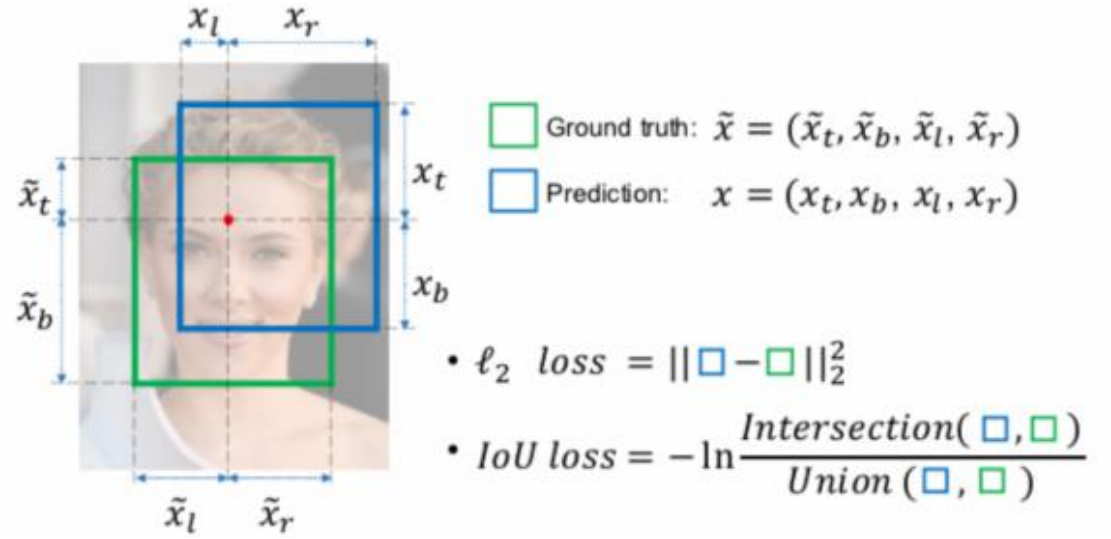


Figure 5: Compared to ℓ_2 loss, the IoU loss is much more robust to scale variations for bounding box prediction.

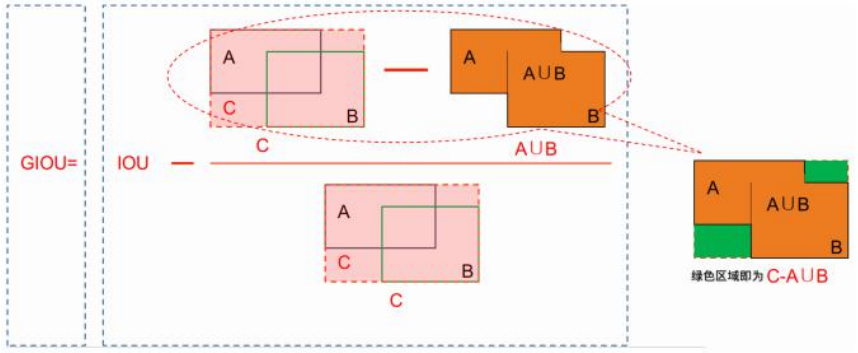


Iou=两个 bbox 的交除以并。

但是当两个物体不相交的时候，IOU 没有回传梯度。

GIOU

IOU 不相交时，没有梯度，无法衡量两框是相邻还是甚远，因此 GIOU 出现了。



当两个 bbox 围起来的面积减去两个 bbox 的交集，除以两个 bbox 围起来的面积。这样两个 bbox 离得越远，分子就越大，Loss 也越大。

DIOU

当一个目标框包含另一个目标框时，无法判断哪个更好

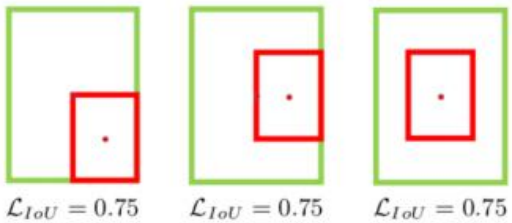
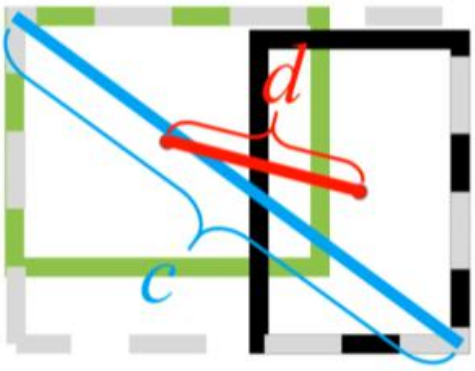


Figure 2: GIoU loss degrades to IoU loss for these cases, while our DIoU loss is still distinguishable. Green and red denote target box and predicted box respectively.



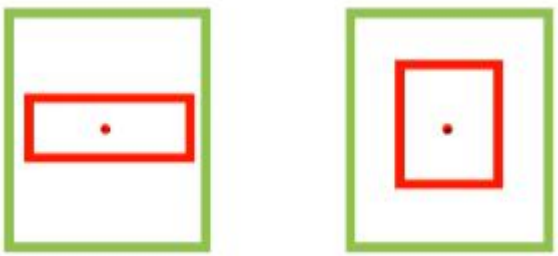
$$\mathcal{R}_{DIoU} = \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2} \quad \text{(Penalty Term)}$$

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

计算中心点距离和斜边距离的比值值域是[0，2)。DIoU loss 会使 IoU 趋近 1，使这个距离比趋近 0。

CIOU

Diou 没有考虑到框的长宽比。



$$\mathcal{R}_{CIoU} = \frac{\rho^2(\mathbf{b}, \mathbf{b}^{gt})}{c^2} + \alpha v$$

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

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

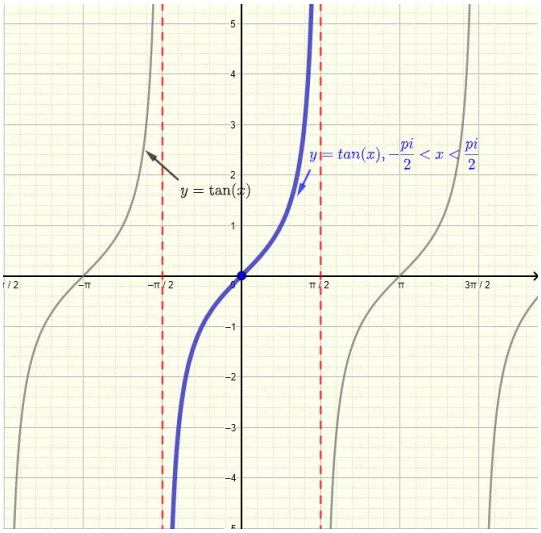
$$\alpha = \frac{v}{(1 - IoU) + v}$$ The more IoU is, the

在 DIoU 的基础上加上 αV。V 是 ground truth 的宽高比减去 bbox 的宽高比，为什么加上 arctan，是为了限制长宽比。不让它飚了。Arctan 值在-pi/2 和 pi/2 之间。为什么平方又

乘以 4/pi^2，为了归一化到 1.

Alpha 是为了平衡 IoU 和 V，当 IoU 小的时候，这时候首先考虑增大 IoU，V 不是那么重要。

当 bbox 逐渐靠拢 gt，alpha 会偏向 V，这时候就开始考虑宽高比了。



Dice Loss

用于计算两个样本之间的像素之间的相似度

Dice $s = \frac{2|X \cap Y|}{|X| + |Y|} = \frac{2TP}{2TP + FN + FP}$ Dice loss $s = 1 - \frac{2|X \cap Y|}{|X| + |Y|}$

分子中之所以有一个系数 2 是因为分母中有重复计算 X 和 Y 的原因，s 的取值范围是[0,1]。

缺点是对小目标十分不利，因为在只有前景和背景的情况下，小目标一旦有部分像素预测错误，那么就会导致 Dice 大幅度的变动，从而导致梯度变化剧烈，训练不稳定。

Dice 和 IoU 的区别：$IoU = \frac{Dice}{2 - Dice}$

$$Dice = \frac{2TP}{2TP + FP + FN} = \frac{2|X \cap Y|}{|X| + |Y|}$$

$$IoU = \frac{TP}{TP + FP + FN} = \frac{|X \cap Y|}{|X| + |Y| - |X \cap Y|}$$

分布 Loss

Binary Cross-Entropy

二分类交叉熵损失函数

交叉熵定义为对给定随机变量或事件集的两个概率分布之间的差异的度量。它被广泛用于分类任务，并且由于分割是像素级分类，因此效果很好。在多分类任务中， 经常采用 softmax 激活函数+交叉熵损失函数，因为交叉熵描述了两个概率分布的差异，然而神经网络输出的是向量，并不是概率分布的形式。所以需要 softmax 激活函数将一个向量进行“归一化”成概率分布的形式，再采用交叉熵损失函数计算 loss。

$$L = -\sum_{i=1}^{N}[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)]$$

yi 表示样本 i 的 label，正类为 1，负类为 0。y^表示为正类的预测值。

在图像分割任务中，需要分割前景和背景。当前景像素的数量远远小于背景像素的数量时，即 y=0 的数量远大于 y=1 的数量，损失函数会倾向于将所有像素判断为背景。

Weighted Binary Cross-Entropy

加权交叉熵损失函数只是在交叉熵 Loss 的基础上为每一个类别添加了一个权重参数为正样本加权。如果前景少，将 β >1，提升正样本的权重。缺点是 β 需要人为调整。

$$WCE(p, \hat{p}) = -[\beta p \log \hat{p} + (1 - p) \log(1 - \hat{p})]$$

Balanced Cross-Entropy

$$L_{BCE}(y, \hat{y}) = -(\beta * ylog(\hat{y}) + (1 - \beta) * (1 - y)log(1 - \hat{(}y)))$$

Here, β is defined as $1 - \frac{y}{H*W}$

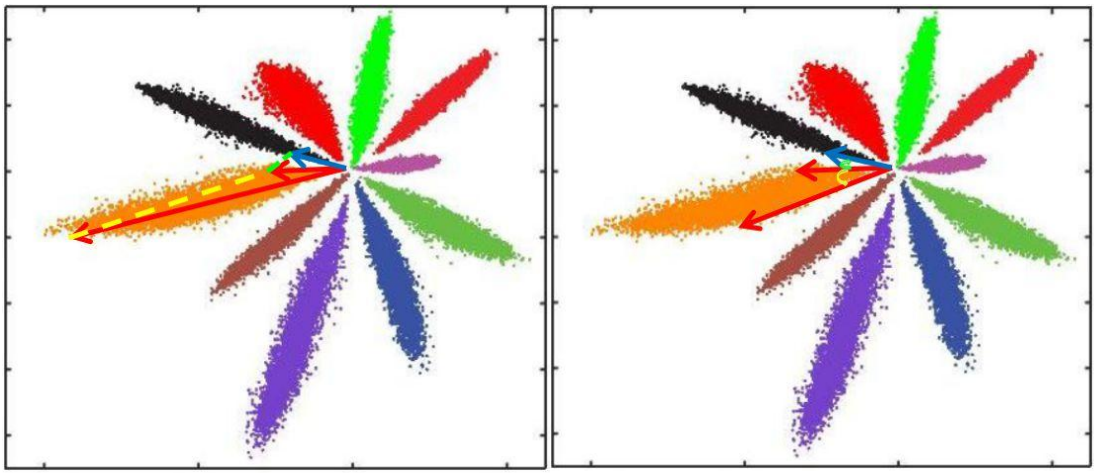
Focal Loss

Focal loss 是在目标检测领域提出来的。其目的是关注难例（也就是给难分类的样本较大的权重）。对于正样本，使预测概率大的样本（简单样本）得到的 loss 变小，而预测概率小的样本（难例）loss 变得大，从而加强对难例的关注度。缺点是：引入了额外参数，增加了调参难度。

$$FL(p, \hat{p}) = -[\alpha(1 - \hat{p})^{\gamma} p \log \hat{p} + (1 - \alpha)\hat{p}^{\gamma}(1 - p) \log(1 - \hat{p})]$$

特性向量 Loss

在人脸识别等类似的需要判断向量相似度的任务中，我们发现，以前的 softmax loss（softmax+交叉熵）虽然可以分类，但是类内太散不够聚合，造成类内的 L2 距离往往比类间还大，Cos 距离也是同样的道理。



对用 softmax loss 分类后的 MNIST 数据集进行可视化后发现，比如黄色的类，对于 L2 距离，同一类之间的两点的距离可以大于黄色到黑色类之间的距离。对于 cos 距离，也有同样情况，在图 2 中可以清晰的看到。当计算两个特征向量的时候，这就会造成分类错误。

所以需要改造 Softmax，除了保证可分性外，还要做到特征向量类内尽可能紧凑，类间尽可能分离。

Softmax Loss

Softmax 是 soft（软化）的 max，是将每个输出 x 非线性放大到 exp(x)。

超多分类的 Softmax，人脸或者商品，训练时，分类都上千或者过万。

通过非线性放大拉开了最大值和其它值之间的距离，降低了训练的难度。

但这也带来一个问题，就是 softmax 鼓励类别之间有差距，但是并不鼓励这个差距拉的很大。

如下图（5，1，1，1）就能带来不错的分类效果，就没有必要追求（10，1，1，1）了。

feature	**1**	1	1	1	**2**	1	1	1
$p=\frac{x_i}{\sum_j x_j}$	**25%**	25%	25%	25%	**40%**	20%	20%	20%
$p=\frac{e^{x_i}}{\sum_j e^{x_j}}$	**25%**	25%	25%	25%	**47.54%**	17.49%	17.49%	17.49%
feature	**5**	1	1	1	**10**	1	1	1
$p=\frac{x_i}{\sum_j x_j}$	**62.5%**	12.5%	12.5%	12.5%	**76.92%**	7.69%	7.69%	7.69%
$p=\frac{e^{x_i}}{\sum_j e^{x_j}}$	**94.79%**	1.74%	1.74%	1.74%	**99.96%**	0.012%	0.012%	0.012%

$$\text{softmax: } S_j = \frac{e^{a_j}}{\sum_{k=1}^{T} e^{a_k}}$$

$$a_k = W_j^T x_i + b_j$$

$$Loss = -\sum_{j=1}^{T} y_j \, logS_j$$

简化后：$Loss = -logS_j$

$$L_1 = -\frac{1}{N}\sum_{i=1}^{N} \log \frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^{n} e^{W_j^T x_i + b_j}},$$

Contrastive Loss

$$\text{Verif}(f_i, f_j, y_{ij}, \theta_{ve}) = \begin{cases} \frac{1}{2}\|f_i - f_j\|_2^2 & \text{if } y_{ij} = 1 \\ \frac{1}{2}\max\left(0, m - \|f_i - f_j\|_2\right)^2 & \text{if } y_{ij} = -1 \end{cases}$$

DeepID2, DeepID2+, DeepID3 系列采用的 Softmax+Contrastive Loss。类内距离 Loss 等于两向量之间的 L2 距离，而类间距离 Loss 若是小于 margin，等于 margin 减去距离的平方，若是大于 margin，就等于 0。这样网络就会向着尽可能减少类内距离和让类间距离尽可能大于 margin 的方向努力。

Triple Loss

$$\sum_i^N \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha\right]_+$$

以三元组（a, p, n）形式进行优化（Anchor,Positive,Negative），不同类特征的 L2 距离要比同类特征的 L2 距离大 margin m，同时获得类内紧凑和类间分离。三元组选择极具技巧性，复现非常困难。

加约束 Loss

Center Loss

为每个类别学习一个中心，并将每个类别的所有特征向量拉向对应类别中心，联合 Softmax 一起使用。

$$\begin{aligned} \mathcal{L} &= \mathcal{L}_S + \lambda \mathcal{L}_C \\ &= -\sum_{i=1}^{m} \log \frac{e^{W_{y_i}^T \boldsymbol{x}_i + b_{y_i}}}{\sum_{j=1}^{n} e^{W_j^T \boldsymbol{x}_i + b_j}} + \frac{\lambda}{2} \sum_{i=1}^{m} \|\boldsymbol{x}_i - \boldsymbol{c}_{y_i}\|_2^2 \end{aligned}$$

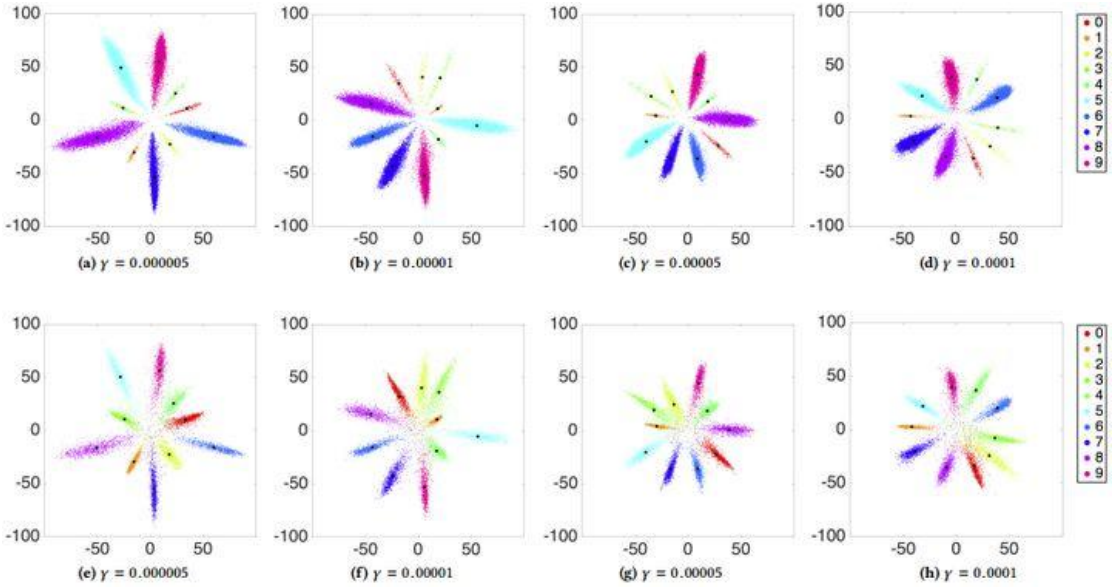
Center Loss 在 Softmax 的基础上，**仅显式约束类内紧凑**。但 Center Loss 为每个类别需要保留一个类别中心，当类别数量很多（>10000）时，这个内存消耗非常可观，对 GPU 的内存要求较高。

Center Invariant Loss

首先证明了训练样本多的类别，Softmax 训练后特征区域会更大，这就是训练集类别不均衡导致的分类倾向问题，Center invariant loss 联合 Softmax + Center loss 一起使用：

$$L = L_s + \gamma L_I + \lambda L_c$$
$$= -\log\left(\frac{e^{\mathbf{w}_y^T \mathbf{x}_i + b_y}}{\sum_{j=1}^{m} e^{\mathbf{w}_j^T \mathbf{x}_i + b_j}}\right) + \frac{\gamma}{4}\left(||\mathbf{c}_y||_2^2 - \frac{1}{m}\sum_{k=1}^{m}||\mathbf{c}_k||_2^2\right)^2$$
$$+ \frac{\lambda}{2}||\mathbf{x}_i - \mathbf{c}_y||^2,$$

除了 Center loss 每个类都拉向类别中心，额外约束每个类的类别中心都拉向一个固定半径的超球上，这个半径是所有类别中心的模均值，减轻类别不均衡带来的特征区域差异。



这个方法了解一下思想就行。

Range Loss

首先证明了训练集的长尾分布(Long tail distribution)会影响训练模型的性能，然后针对训练集的长尾分布提出，类似 Contrastive Loss 提出了 Range loss 与 Softmax 一起使用：

$$\mathcal{L} = \mathcal{L}_M + \lambda \mathcal{L}_R = -\sum_{i=1}^{M} \log \frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^{n} e^{W_j^T x_i + b_j}} + \lambda \mathcal{L}_R$$

$$\mathcal{L}_R = \alpha \mathcal{L}_{R_{intra}} + \beta \mathcal{L}_{R_{inter}}$$

$$\mathcal{L}_{R_{intra}} = \sum_{i \subseteq I} \mathcal{L}_{R_{intra}}^{i} = \sum_{i \subseteq I} \frac{k}{\sum_{j=1}^{k} \frac{1}{\mathcal{D}_j}}$$

$$\mathcal{L}_{R_{inter}} = \max(m - \mathcal{D}_{Center}, 0)$$
$$= \max(m - \|\overline{x}_{\mathcal{Q}} - \overline{x}_{\mathcal{R}}\|_2^2, 0)$$

Range loss 同时约束类内紧凑类间分离，类内紧凑约束为每个类最小化两个最大类内距离，类间分离约束为每次都计算每个类别中心，并使类中心距离最小的两个类别距离大于 margin m。这个方法也是了解一下思想就行。

Ring Loss

Ring loss 将所有特征向量都拉向半径为 R 的超球上，需要联合 Softmax 或 SphereFace 一起使用。

$$L_R = \frac{\lambda}{2m}\sum_{i=1}^{m}(\|\mathcal{F}(\mathbf{x}_i)\|_2 - R)^2$$

Ring loss 将特征向量的模长约束到固定值 R 附近。想法非常简单，效果也非常简单：相比 SphereFace，SphereFace+Ring loss 并没有表现出任何优势。这个方法也是了解一下思想就行。

Largin Angular margin 系列算法

L-Softmax

L-Softmax 是 Large-Margin Softmax Loss，是 large margin 系列的开创算法。首先联合 FC+Softmax+Cross-entropy 重新给出 Softmax loss 的表达式：

$$L_i = -\log\left(\frac{e^{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|\cos(\theta_{y_i})}}{\sum_j e^{\|\boldsymbol{W}_j\|\|\boldsymbol{x}_i\|\cos(\theta_j)}}\right)$$

然后加强分类条件，强制让对应类别的 W 和 x 夹角增加到原来的 m 倍，下面看到的长得比较复杂的 $\psi(\theta)$ 是 $cos(m\theta)$ 的单调函数版本：

$$L_i = -\log\left(\frac{e^{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|\psi(\theta_{y_i})}}{e^{\|\boldsymbol{W}_{y_i}\|\|\boldsymbol{x}_i\|\psi(\theta_{y_i})} + \sum_{j\neq y_i} e^{\|\boldsymbol{W}_j\|\|\boldsymbol{x}_i\|\cos(\theta_j)}}\right)$$

$$\psi(\theta) = (-1)^k \cos(m\theta) - 2k, \quad \theta \in [\frac{k\pi}{m}, \frac{(k+1)\pi}{m}]$$

原来类内的夹角是θ，现在原θ=m*θ，将原来的类内的夹角缩小m倍，也可以说类内夹角不变而把类间的夹角放大了m倍。**L-Softmax 仅显式约束类间分离**。

SphereFace

SphereFace 是 L-Softmax 的改进。归一化了权值 W，让训练更加集中在优化深度特征映射和特征向量角度上，降低样本数量不均衡问题，提出了 A-Softmax（angular softmax）：

$$L_{\text{ang}} = \frac{1}{N} \sum_i -\log \left(\frac{e^{\|\boldsymbol{x}_i\| \psi(\theta_{y_i,i})}}{e^{\|\boldsymbol{x}_i\| \psi(\theta_{y_i,i})} + \sum_{j \neq y_i} e^{\|\boldsymbol{x}_i\| \cos(\theta_{j,i})}} \right)$$

$$\psi(\theta) = (-1)^k \cos(m\theta) - 2k, \quad \theta \in [\frac{k\pi}{m}, \frac{(k+1)\pi}{m}]$$

SphereFace 依然仅显式约束类间分离。

L-Softmax 和 SphereFace 都采用乘性 margin 使不同类别更加分离，特征相似度都采用 cos 距离，需要注意这两个 loss 直接训练很难收敛，实际训练中都用到了退火方法：

$$f_{y_i} = \frac{\lambda \|\boldsymbol{x}_i\| \cos(\theta_{y_i}) + \|\boldsymbol{x}_i\| \psi(\theta_{y_i})}{1+\lambda}$$

从 Softmax 逐渐退火到 L-Softmax 或 A-Softmax，难以训练可能是因为这个乘性 margin 太难了。因为 SphereFace 中 m=4，即夹角要增大到原来的四倍，难度太大导致很难收敛，而采用退火方法后，最终等价于 m=1.5，相当于降低了训练难度。

Feature Normalization 特性归一化

COCO Loss

SphereFace 仅将权值归一化了，COCO(congenerous cosine)loss 进一步将特征也归一化，并乘以尺度因子α：

$$\mathcal{L}^{COCO}(\boldsymbol{f}^{(i)}, \boldsymbol{c}_k) = -\sum_{i \in \mathcal{B}, k} t_k^{(i)} \log p_k^{(i)} = -\sum_{i \in \mathcal{B}} \log p_{l_i}^{(i)}$$

$$\hat{\boldsymbol{c}}_k = \frac{\boldsymbol{c}_k}{\|\boldsymbol{c}_k\|}, \ \hat{\boldsymbol{f}}^{(i)} = \frac{\alpha \boldsymbol{f}^{(i)}}{\|\boldsymbol{f}^{(i)}\|}, \ p_k^{(i)} = \frac{\exp(\hat{\boldsymbol{c}}_k^T \cdot \hat{\boldsymbol{f}}^{(i)})}{\sum_m \exp(\hat{\boldsymbol{c}}_m^T \cdot \hat{\boldsymbol{f}}^{(i)})}$$

L2-Softmax

在 Softmax 的 w*x 基础上，将特征向量 x 做归一化，并乘尺度因子进行放大：

$$p = \frac{e^{x_i}}{\sum_j e^{x_j}} \qquad \mathbf{y} = \frac{\mathbf{x}}{\|\mathbf{x}\|_2}$$

$$\mathbf{z} = \alpha \cdot \mathbf{y}$$

尺度因子 α 可以是固定值，也可以自适应训练，建议用固定值 α=50。

NormFace

归一化了特征，同样加了尺度因子 s：

$$\tilde{\mathbf{x}} = \frac{\mathbf{x}}{\|\mathbf{x}\|_2} = \frac{\mathbf{x}}{\sqrt{\sum_i \mathbf{x}_i^2 + \epsilon}}. \quad \mathcal{L}_S = -\frac{1}{m}\sum_{i=1}^{m} log\frac{e^{s\tilde{W}_{y_i}^T \tilde{\mathbf{f}}_i}}{\sum_{j=1}^{n} e^{s\tilde{W}_j^T \tilde{\mathbf{f}}_i}}$$

AM-Softmax/CosFace

在 SphereFace 的基础上，乘性 margin 改成了加性 margin，即 cos(mθ)变成了 cos(θ)-m，在权值 W 归一化的基础上对特征 f 也做了归一化,采用固定尺度因子 s=30,相比 SphereFace 性能有提升，最重要的是训练难度大幅降低，不需要退火优化。

ArcFace

在 SphereFace 的基础上，乘性 margin 改成了加性 margin，即 cos(mθ)变成了 cos(θ+m)，权值 W 归一化，特征 f 也做了归一化，采用固定尺度因子 s=64，ArcFace 的特点是大训练集加大网络，也做了细致的训练集和测试集清理。

特征归一化的重要性，权值 W 和特征 f(或 x)归一化已经成为了标配，而且都给归一化特征乘以尺度因子 s 进行放大，目前主流都采用固定尺度因子 s 的方法；

权值和特征归一化使得 CNN 更加集中在优化夹角上，得到的深度人脸特征更加分离；

特征归一化后，特征向量都固定映射到半径为 1 的超球上，便于理解和优化；但这样也会压缩特征表达的空间；乘尺度因子 s，相当于将超球的半径放大到 s，超球变大，特征表达的空间也更大（简单理解：半径越大球的表面积越大）；

特征归一化后，人脸识别计算特征向量相似度，L2 距离和 cos 距离意义等价，计算量也相同，我们再也不用纠结到底用 L2 距离还会用 cos 距离，后面有解释为什么归一化后，L2 距

离和 cos 距离意义等价；

为什么仅特征归一化无法收敛，而必须乘固定尺度因子呢？以四分类为例：仅特征归一化时，输出{x1, x2, x3, x4}等价于{cos θ , x2, x3, x4}（x1=wx=|w||x|cos θ =cos θ ），理想的情况下，类内 θ 最小化，为 0，{x2, x3, x4}都为 0，得到{1，0，0，0}。过 softmax 得到的概率时{47.54%, 17.49%, 17.49%, 17.49%}，远远达不到收敛的要求，所以仅归一化是不能训练的。而乘以放大因子 α 以后，比如 α =60，得到的值就是{60，0，0，0}，过 softmax 得到{100%, 0%, 0%, 0%}，完全可以达到收敛要求。所以特征归一化必须乘尺度因子。

Large Margin 到底优化了什么，前面提到 large margin 显式约束了类间分离，看可视化结果好像也是这样，但其实这种说法是不对的。large margin 优化的核心-夹角 θ 是权值 W 和特征 f 之间的夹角，并不是不同类别之间的夹角。Loss 函数也完全没有涉及不同类别特征向量之间的夹角约束。从公式 $W \cdot f = |W| \cdot |f| \cdot cos\theta$ 可以看出，归一化后，训练目标从 $cos(\theta)$ 变成 $cos(m\theta)$ ， $cos\theta - m$ 或 $cos(\theta + m)$ ，要使概率达到 100%，就需要优化出更小的夹角 θ，也就是说 **large margin 的优化目标是让权值向量 W 和特征向量 f 之间的夹角更小**。如果一个类有 1000 张图片，CNN 特征映射后得到 1000 个特征向量，而 W 权值向量只有一个。large margin loss 要求这 1000 个特征向量和这 1 个权值向量的夹角非常小，也就是说，**优化让 1000 个特征向量都向权值向量 W 的方向靠拢**。因此 **large magin 是显式的类内夹角约束，目标是让同一类的所有特征向量都拉向该类别的权值向量。**

总结，L-Softmax 重构了 Softmax，输出 x 变成 $W \cdot f = |W| \cdot |f| \cdot cos\theta$ ，SphereFace 归一化权值 W，变成 $W \cdot f = |f| \cdot cos\theta$ ；最新 AM-Softmax 和 ArcFace 继续归一化特征乘尺度因子，变成 $W \cdot f = s \cdot cos\theta$ 。所以这里我们简化问题，默认归一化权值 W 和特征 f，即 $e^x = e^{s \cdot cos\theta}$ ，仅考虑 $cos(\theta)$ 这一项变动对分类任务的影响。Softmax 相比 hardmax 放大了输出，减小了训练难度，使分类问题更容易收敛，但是也造成了训练提前收工，类内不够聚合，类间不够分散的问题。因此 $cos(m\theta)$ ， $cos\theta - m$ 和 $cos(\theta + m)$ 都非线性的减小了输出，增加了训练难度，使得训练得到的特征映射更好。在这几个 loss 中，乘性 margin 的 SphereFace 训练难度最大，难以收敛，而加性 margin 的 ArcFace 训练难度稍加，所以更容易收敛。

余弦相似度与欧式距离

两向量的内积除以两向量的 L2 范数。

最小化

$$\cos(\theta) = \frac{\sum_{i=1}^{n}(x_i \times y_i)}{\sqrt{\sum_{i=1}^{n}(x_i)^2} \times \sqrt{\sum_{i=1}^{n}(y_i)^2}}$$

$$= \frac{a \bullet b}{||a|| \times ||b||}$$ 余弦相似度

$$d(x,y) := \sqrt{(x_1-y_1)^2+(x_2-y_2)^2+\cdots+(x_n-y_n)^2} = \sqrt{\sum_{i=1}^{n}(x_i-y_i)^2}.$$ 欧式距离

余弦相似度与欧式距离的区别，相似度不能代替距离，这两个不是等价的关系，它们一个是角度的衡量，一个是距离的衡量。

将欧式距离和余弦相似度结合起来：

假设二维空间两个点， $A(x_1,y_1), B(x_2,y_2)$

然后归一化为**单位向量**， $A(\frac{x_1}{\sqrt{x_1^2+y_1^2}}, \frac{y_1}{\sqrt{x_1^2+y_1^2}}), B(\frac{x_2}{\sqrt{x_2^2+y_2^2}}, \frac{y_2}{\sqrt{x_2^2+y_2^2}})$

那么余弦相似度就是： $cos = \frac{x_1}{\sqrt{x_1^2+y_1^2}} \times \frac{x_2}{\sqrt{x_2^2+y_2^2}} + \frac{y_1}{\sqrt{x_1^2+y_1^2}} \times \frac{y_2}{\sqrt{x_2^2+y_2^2}}$

(分母是1，省略了)

欧式距离就是： $euc = \sqrt{(\frac{x_1}{\sqrt{x_1^2+y_1^2}} - \frac{x_2}{\sqrt{x_2^2+y_2^2}})^2 + (\frac{y_1}{\sqrt{x_1^2+y_1^2}} - \frac{y_2}{\sqrt{x_2^2+y_2^2}})^2}$

化简后就是： $euc = \sqrt{2-2\times cos}$

上面的公式可以再推一步：

$$euc = \sqrt{2(1-cos(A,B))}$$

cos(A, B)为余弦相似度，1-cos(A,B)为余弦距离。

pytorch 中的损失函数

- class torch.nn.L1Loss(size_average=True)
- class torch.nn.MSELoss(size_average=True)
- class torch.nn.CrossEntropyLoss(weight=None, size_average=True)
- class torch.nn.NLLLoss(weight=None, size_average=True)
- class torch.nn.NLLLoss2d(weight=None, size_average=True)
- class torch.nn.KLDivLoss(weight=None, size_average=True)
- class torch.nn.BCELoss(weight=None, size_average=True)
- class torch.nn.MarginRankingLoss(margin=0, size_average=True)
- class torch.nn.HingeEmbeddingLoss(size_average=True)
- class torch.nn.MultiLabelMarginLoss(size_average=True)
- class torch.nn.SmoothL1Loss(size_average=True)
- class torch.nn.SoftMarginLoss(size_average=True)
- class torch.nn.MultiLabelSoftMarginLoss(weight=None, size_average=True)
- class torch.nn.CosineEmbeddingLoss(margin=0, size_average=True)
- class torch.nn.MultiMarginLoss(p=1, margin=1, weight=None, size_average=True)

L1Loss

L1 loss

MSELoss

L2 loss

CrossEntropyLoss

softmax+log+交叉熵

NLLLoss

交叉熵

NLLLoss2d

多了几个维度(input, (N, C, H, W)，target, (N, H, W))，用于图片，最后图片的每个点都会预测一个类别标签。可以用于语义识别。

KLDivLoss

KL 散度计算，KLDivLoss=Yn*(logYn - Xn)，所以输入 Yn 是原标签 softmax 后的值，Xn 是输出经过 logsoftmax 后的值。

BCELoss

是多分类的交叉熵，把输出过 sigmoid 之后再输入 BCELoss

BCEWithLogitsLoss

是把 sigmoid 和 BCELoss 合成一步，比如输出经过 sigmoid 以后是[0.39,0.22,0.64]，标签是[0,1,1]，loss=(0*ln0.39+1*(1-0.39) + 1*ln0.22+0*(1-0.22) + 1*ln0.64+0*(1-0.64)) / 3;

MarginRankingLoss

待更

HingeEmbeddingLoss

待更

MultiLabelMarginLoss

待更

SmoothL1Loss

SmoothL1 Loss

SoftMarginLoss

待更

MultiLabelSoftMarginLoss

待更

CosineEmbeddingLoss

待更

MultiMarginLoss

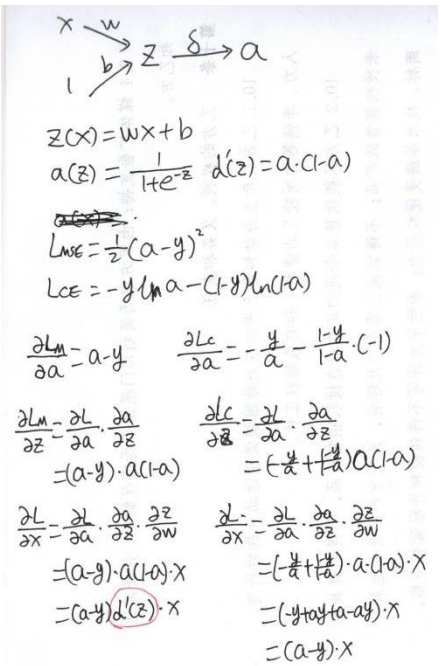
Q&A

为什么分类问题损失函数用交叉熵不用 L2

交叉熵处理概率分布问题，离散问题。L2 处理距离问题，连续函数问题。

分类问题是要找出正确的分类，而不是要求函数在分类的每个值上都要错误最小。这样也容易过拟合。

L2 的导数是 sigmoid 导数的函数，而交叉熵的导数是 x 的函数，计算简单。



什么时候用 sigmoid，什么时候用 softmax

若分类可选多个类别多标签，那么用 sigmoid（电影可以是美国，喜剧）

若分类是互斥的类别，用 softmax（男，女）

zi: [-0.5, 1.2, -0.1, 2.4]

Sigmoid函数输出：

$$S(x) = \frac{1}{1+e^{-x}}$$

$\sigma(zi)$: $[0.37, 0.77, 0.48, 0.91]$

Softmax函数形式:

$$S(x_j) = \frac{e^{xj}}{\sum_{k=1}^{K} e^{xk}}, j = 1, 2, \ldots, K$$

$\sigma(zi)$: $[0.04, 0.21, 0.05, 0.70]$

- **Sigmoid交叉熵：**

Loss (y_pred, y_true) = - y_true * ln(sigmoid(y_pred)) – (1 – y_true) * ln(1 – sigmoid(y_pred))

- **Softmax交叉熵：**

Loss (y_pred, y_true) = ∑ –y_true * ln(softmax(y_pred))