损失函数大总结：

https://blog.csdn.net/qq_14845119/article/details/80787753

损失函数（loss function)是用来估量模型的预测值f(x)与真实值Y的不一致程度，是一个非负实值函数，通常使用L(Y,f(x))来表示。损失函数越小，模型的鲁棒性越好。损失函数是**经验风险函数**的核心部分，也是**结构风险函数**重要组成部分。模型的结构风险函数包括了经验风险项和正则项，通常表示如下：

$$\theta^* = \arg\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i;\theta)) + \lambda \Phi(\theta)$$

其中，前面的均值函数表示的是经验风险函数，L代表的是损失函数，后面的Φ是正则化项（regularizer）或者叫惩罚项（penalty term），它可以是L1，也可以是L2，或者其他的正则函数。整个式子表示的意思是找到使目标函数最小时的 θ 值。下面主要列出几种常见的损失函数。

1. log对数损失函数（逻辑回归）

逻辑回归的损失函数并不是平方损失。平方损失函数可以通过线性回归在假设样本是高斯分布的条件下推导得到，而逻辑回归得到的并不是平方损失。在逻辑回归的推导中，它假设样本服从伯努利分布（0-1分布），然后求得满足该分布的似然函数，接着取对数求极值等等。而逻辑回归并没有求似然函数的极值，而是把极大化当做一种思想，进而推导出它的经验风险函数为：最小化负的似然函数（即max F(y, f(x)) —> min -F(y, f(x)) 。从损失函数的角度来看，它就成了log损失函数。

log损失函数的标准形式：

$L(Y,P(Y|X))=-\log P(Y|X)$

取对数是为了方便计算极大似然估计，因为在MLE中，直接求导比较困难，所以通常都是先取对数再求导找极值点。损失函数L(Y,P(Y|X))表达的是样本X在分类Y的情况下，使概率P(Y|X)达到最大值。（换言之，就是利用已知的样本分布，找到最有可能（即最大概率）导致这种分布的参数值；或者说什么样的参数才能使我们观测到目前这组数据的概率最大）。因为log函数是单调递增的，所以logP(Y|X)也会达到最大值，因此在前面加上负号之后，**最大化P(Y|X)就等价于最小化L了。**

逻辑回归的P(Y=y|x)表达式如下（为了将类别标签y统一为1和0，下面将表达式分开表示）：

$$P(Y=y|x)=\begin{cases} h_\theta(x)=g(f(x))=\frac{1}{1+exp\{-f(x)\}} & ,y=1 \\ 1-h_\theta(x)=1-g(f(x))=\frac{1}{1+exp\{f(x)\}} & ,y=0 \end{cases}$$

将它带入到上式，通过推导可以得到logistic的损失函数表达式，如下：

$$L(y,P(Y=y|x))=\begin{cases} \log(1+exp\{-f(x)\}) & ,y=1 \\ \log(1+exp\{f(x)\}) & ,y=0 \end{cases}$$

逻辑回归最后得到的目标式子如下：

$$J(\theta)=-\frac{1}{m}\sum_{i=1}^{m}\left[y^{(i)}\log h_\theta(x^{(i)})+(1-y^{(i)})\log(1-h_\theta(x^{(i)}))\right]$$

之所以有人认为逻辑回归是平方损失，是因为在使用梯度下降来求最优解的时候，它的迭代式子与平方损失求导后的式子几乎一样，从而给人一种直观上的错觉。

> **平方损失梯度下降：**

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

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

2. 平方损失函数（最小二乘法,Ordinary Least Squares)

最小二乘法是线性回归的一种，OLS将问题转化成了一个凸优化问题。在线性回归中，它假设样本和噪声都服从高斯分布（为什么假设成高斯分布呢？隐藏了一个小知识点，中心极限定理，参考https://blog.csdn.net/u012284960/article/details/52207886））），最后通过极大似然估计（MLE)可以推导出最小二乘式子。最小二乘的基本原则是：最优拟合直线应该是使各点到回归直线的距离和最小的直线，即平方和最小。换言之，OLS是基于距离的，而这个距离就是我们用的最多的欧几里得距离。为什么它会选择使用欧式距离作为误差度量呢（即Mean squared error， MSE），主要有以下几个原因：

- 简单，计算方便；
- 欧氏距离是一种很好的相似性度量标准；
- 在不同的表示域变换后特征性质不变。

平方损失的标准形式如下：

$$L(Y, f(X)) = (Y - f(X))2$$

当样本个数为n时，此时的损失函数变为：

$$L(Y, f(X)) = \sum_{i=1}^{n} (Y - f(X))^2$$

整个式子表示的是残差的平方和，而我们的目的就是最小化这个目标函数值（没有加入正则化）

在实际应用中，经常会使用均方差（MSE)作为一项衡量指标，公式如下：

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (\tilde{Y}_i - Y_i)^2$$

3.指数损失函数（Adaboost)

指数损失函数的标准形式：

$$L(Y|f(X)) = exp[-yf(x)]$$

Adaboost算法是前向分步加法算法的特例，是一个加和模型，损失函数是指数函数。在Adaboost中，经过m次迭代之后，可以得到fm(x)：

$$f_m(x) = f_{m-1}(x) + \alpha_m G_m(x)$$

Adaboost每次迭代时的目的是为了找到最小化下列式子时的参数α和G：

$$\arg\min_{\alpha,G} = \sum_{i=1}^{N} exp[-y_i(f_{m-1}(x_i) + \alpha G(x_i))]$$

而指数损失函数(exp-loss) 的标准形式如下

$$L(y, f(x)) = \exp[-yf(x)]$$

可以看出，Adaboost的目标式子就是指数损失，在给定n个样本的情况下，Adaboost的损失函数为：

$$L(y, f(x)) = \frac{1}{n}\sum_{i=1}^{n} \exp[-y_i f(x_i)]$$

李航P145有推导。

4.Hinge损失函数（SVM)

在SVM中，最优化问题可以等价于下列式子：

$$\min_{w,b} \sum_{i}^{N} [1 - y_i(w \cdot x_i + b)]_+ + \lambda||w||^2$$

下面对式子变形。令：

$$[1 - y_i(w \cdot x_i + b)]_+ = \xi_i$$

原式变成：

$$\min_{w,b} \sum_{i}^{N} \xi_i + \lambda||w||^2$$

如果取

$\lambda = \frac{1}{2C}$

$$\min_{w,b} \frac{1}{C}\left(\frac{1}{2}||w||^2 + C\sum_{i}^{N}\xi_i\right)$$

可以看出，该式子与下式非常相似：

$$\frac{1}{m}\sum_{i=1}^{m} l(w \cdot x_i + b, y_i) + ||w||^2$$

前半部分中的l就是hinge损失函数，而后面相当于L2正则项。

Hinge 损失函数的标准形式

$$L(y) = \max(0, 1 - y\tilde{y}), y = \pm 1$$

可以看出，当|y|>=1时，L(y)=0。

5.其他损失函数

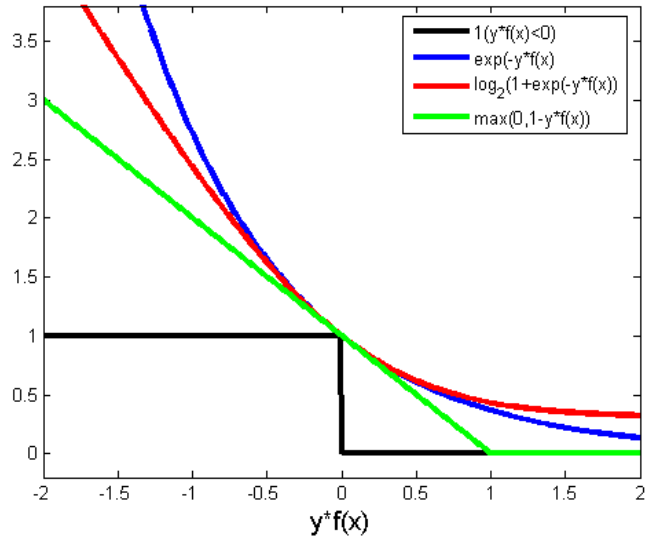
5.1 0-1损失函数

感知机用的0-1损失函数

$$L(Y, f(X)) = \begin{cases} 1, & Y \neq f(X) \\ 0, & y = f(X) \end{cases}$$

5.2 绝对值损失函数

$$L(Y, f(X)) = |Y - f(X)|$$



6.损失函数改进之Large-Margin Softmax Loss（基于最大间隔）

最近几年网络效果的提升除了改变网络结构外，还有一群人在研究损失层的改进，这篇博文要介绍的就是较为新颖的Large-Margin softmax loss（L-softmax loss）。

Large-Margin softmax loss来自ICML2016的论文：Large-Margin Softmax Loss for Convolutional Neural Networks

论文链接：http://proceedings.mlr.press/v48/liud16.pdf，

MxNet代码链接：https://github.com/luoyetx/mx-lsoftmax

先来直观看下Large-Margin Softmax Loss的效果，也是论文的核心：

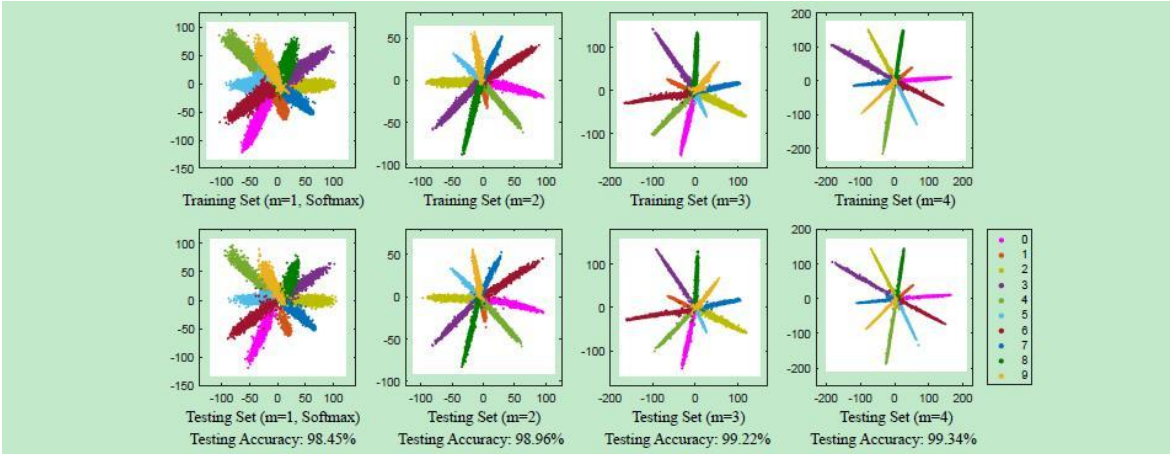


Figure 2. CNN-leanrned features visualization (Softmax Loss (m=1) vs. L-Softmax loss (m=2,3,4)) in MNIST dataset. Specifically, we set the feature (input of the L-Softmax loss) dimension as 2, and then plot them by class. We omit the constant term in the fully connected layer, since it just complicates our analysis and nearly does not affect the performance. Note that, the reason why the testing accuracy is not as good as in Table. 2 is that we only use 2D features to classify the digits here.

什么意思呢？上面一行表示training set，下面一行表示testing set。每一行的第一个都是传统的softmax，后面3个是不同参数的L-softmax，**看看类间和类内距离的差距！**

因为L-softmax loss是在原来传统的softmax loss上改进的，因此还是先从softmax loss讲起：

softmax loss的公式如下：

$$L = \frac{1}{N}\sum_i L_i = \frac{1}{N}\sum_i -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$$

fj表示class score f向量的第j个元素。N表示训练数据的数量。**log函数的括号里面的内容就是softmax，（如果不是很理解softmax，softmax loss，可以参看这篇博文：softmax，softmax-loss，BP的解释）**简单讲就是属于各个类别的概率，因此fyi就可以理解为全连接层的输出，也就是：

$$f_{y_i} = W_{y_i}^T x_i$$

上面这个式子就是W和x的内积，因此可以写成下面这样：

$$f_j = \|W_j\|\|x_i\|\cos(\theta_j)$$

因此Li就变成下式：

$$L_i = -\log\left(\frac{e^{\|W_{y_i}\|\|x_i\|\cos(\theta_{y_i})}}{\sum_j e^{\|W_j\|\|x_i\|\cos(\theta_j)}}\right)$$

那么Large Margin Softmax Loss是什么意思？
假设一个2分类问题，x属于类别1，那么原来的softmax肯定是希望：

$$W_1^T x > W_2^T x$$

也就是属于类别1的概率大于类别2的概率，这个式子和下式是等效的：

$$\|W_1\|\|x\|\cos(\theta_1) > \|W_2\|\|x\|\cos(\theta_2))$$

那么Large Margin Softmax就是将上面不等式替换成下式：

sion margin. So we instead require $\|W_1\|\|x\|\cos(m\theta_1) > \|W_2\|\|x\|\cos(\theta_2)$ $(0 \le \theta_1 \le \frac{\pi}{m})$ where m is a positive integer. Because the following inequality holds:

因为m是正整数，cos函数在0到π范围又是单调递减的，所以cos(mx)要小于cos(x)。m值越大则学习的难度也越大，这也就是最开始Figure2中那几个图代表不同m值的意思。因此通过这种方式定义损失会逼得模型学到类间距离更大的，类内距离更小的特征。不过个人认为可能需要迭代次数多点才能看到效果。

这样L-softmax loss的Li式子就可以在原来的softmax loss的Li式子上修改得到：

Following the notation in the preliminaries, the L-Softmax loss is defined as

$$L_i = -\log\left(\frac{e^{\|W_{y_i}\|\|x_i\|\psi(\theta_{y_i})}}{e^{\|W_{y_i}\|\|x_i\|\psi(\theta_{y_i})} + \sum_{j\neq y_i} e^{\|W_j\|\|x_i\|\cos(\theta_j)}}\right) \tag{4}$$

in which we generally require

$$\psi(\theta) = \begin{cases} \cos(m\theta), & 0 \le \theta \le \frac{\pi}{m} \\ \mathcal{D}(\theta), & \frac{\pi}{m} < \theta \le \pi \end{cases} \tag{5}$$

where m is a integer that is closely related to the classification margin. With larger m, the classification margin becomes larger and the learning objective also becomes harder. Meanwhile, $\mathcal{D}(\theta)$ is required to be a monotonically decreasing function and $\mathcal{D}(\frac{\pi}{m})$ should equal $\cos(\frac{\pi}{m})$.

因此L-softmax loss的思想简单讲就是加大了原来softmax loss的学习难度。借用SVM的思想来理解的话，如果原来的softmax loss是只要支持向量和分类面的距离

大于h就算分类效果比较好了，那么L-softmax loss就是需要距离达到mh（m是正整数）才算分类效果比较好了。

7.loss改进之 带权重的loss

> 2 weighted softmax loss【1】

这第一个改进就是weighted，怎么说？假如我们是一个分类问题，只有两类，但是两类的样本数目差距非常之大。比如边缘检测问题，这个时候，明显边缘像素的重要性是比非边缘像素大的，此时可以针对性的对样本进行加权。

$$l(y,z) = -\sum_{k=0}^{C} w_c y_c log(f(z_c))$$

wc就是这个权重，像刚才所说，c=0代表边缘像素，c=1代表非边缘像素，则我们可以令w0=1，w1=0.001，即加大边缘像素的权重。

当然，这个权重，我们还可以动态地计算让其自适应。

具体的反向公式推导，就跟上面差不多不再赘述，详细的实现我会在git中给出代码。【1】中用了weighted sigmoid_cross_entropy_loss，原理类似。

固定权重的fixed softmax loss和自适应权重的adapted softmax loss大家可以去git中自行查看。

Loss: Loss function based on interclass distance

- **Loss function**

$$loss(x,y) = weight_y\left(-\log\left(\frac{\exp(x[y])}{\sum_j \exp(x[j])}\right)\right)$$

$$weight_y = \left(\left|\arg\max(x) - y\right| + 1\right) / M, \quad y \in [0, C-1]$$

$$M = \sum_{i=0}^{C-1}\left(\left|y - i\right| + 1\right)$$

$$x = (x_0, x_1, \ldots, x_{C-1})$$

Model	Score
SENet	0.9411
SENet + Custom loss	**0.9444**

8、交叉熵

交叉熵损失的计算分为两个部分

8.1 softmax分类器

交叉熵损失是基于softmax计算来的，softmax将网络最后输出z通过指数转变成概率形式。首先看一下softmax计算公式：

$$p_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

其中， 分子e^{z_i} 是要计算的类别 i 的网络输出的指数；分母是所有类别网络输出的指数和，共k个类别。这样就得到了类别i的输出概率 p_i 。

> 这里说点题外话，实际上，softmax是由逻辑斯的回归模型（用于二分类）推广得到的多项逻辑斯蒂回归模型（用于多分类）。具体可以参考李航大神的《统计学方法》第六章

8.2 交叉熵损失

公式定义如下： $J = -\frac{1}{N}\sum_{1}^{N}\sum_{i=1}^{k} y_i \cdot log(p_i)$

其中， y_i 是类别 i 的真实标签； p_i 是上面softmax计算出的类别 i 的概率值；k是类别数，N是样本总数。

两种形式

- $-t_j \log(y_j)$ ， t_j说明样本的ground-truth是第j类。
- $-\sum_i t_i \log(y_i) + (1 - t_i)\log(1 - y_i)$

这两个都是交叉熵损失函数，但是看起来长的却有天壤之别。为什么同是交叉熵损失函数，长的却不一样呢?

因为这两个交叉熵损失函数对应不同的最后一层的输出：
第一个对应的最后一层是softmax，第二个对应的最后一层是sigmoid。

9.Focal loss

目标识别有两大经典结构: **第一类是以Faster RCNN为代表的两级识别方法**，这种结构的第一级专注于proposal的提取，第二级则对提取出的proposal进行分类和精确坐标回归。**两级结构准确度较高**，但因为第二级需要单独对每个proposal进行分类/回归，速度就打了折扣；目标识别的**第二类结构是以YOLO和SSD为代表的单级结构**，它们摒弃了提取proposal的过程，只用一级就完成了识别/回归，**虽然速度较快但准确率远远比不上两级结构**。那有没有办法在单级结构中也能实现较高的准确度呢？Focal Loss就是要解决这个问题。

一、为什么单级结构的识别准确度低

作者认为单级结构准确度低是由类别失衡**(class imbalance)**引起的。在深入理解这个概念前我们先来强化下“类别”这个概念：计算Loss的bbox可以分为positive和negative两类。当bbox(由anchor加上偏移量得到)与ground truth间的IOU大于上门限时(一般是0.5)，会认为该bbox属于positive example，如果IOU小于下门限就认为该bbox属于negative example。在一张输入image中，目标占的比例一般都远小于背景占的比例，**所以两类example中以negative为主，这引发了两个问题：**

1 、**negative example过多造成它的loss太大，以至于把positive的loss都淹没掉了，不利于目标的收敛；**

2 、**大多negative example不在前景和背景的过渡区域上**，分类很明确(这种易分类的negative称为**easy negative)**，训练时对应的背景类score会很大，换个角度看就是单个example的loss很小，反向计算时梯度小。梯

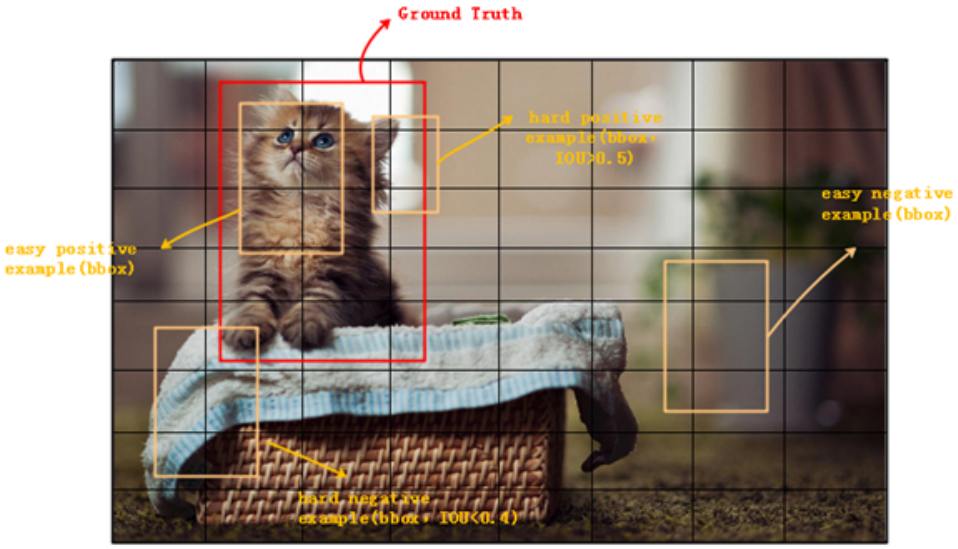
度小造成easy negative example对参数的收敛作用很有限，我们更需要loss大的对参数收敛影响也更大的example，即**hard positive/negative example**。

这里要注意的是前一点我们说了negative的loss很大，是因为negative的绝对数量多，所以总loss大；后一点说easy negative的loss小，是针对单个example而言。

Faster RCNN的两级结构可以很好的规避上述两个问题。具体来说它有两大法宝：1、会根据前景score的高低过滤出最有可能是前景的example (1K~2K个)，因为依据的是前景概率的高低，就能把大量背景概率高的**easy negative给过滤掉，这就解决了前面的第2个问题**；2、**会根据IOU的大小来调整positive和negative example的比例，比如设置成1：3，这样防止了negative过多的情况**(同时防止了easy negative和hard negative)，就解决了前面的第1个问题。所以Faster RCNN的准确率高。

OHEM是近年兴起的另一种筛选example的方法，**它通过对loss排序，选出loss最大的example来进行训练，这样就能保证训练的区域都是hard example。这个方法有个缺陷，它把所有的easy example都去除掉了，造成easy positive example无法进一步提升训练的精度。**

图1是hard positvie、hard negative、easy positive、easy negative四种example的示意图，可以直观的感受到easy negativa占了大多数。



二、**Focal Loss的解决方法**

$$\mathrm{FL}(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t).$$

Focal Loss通过调整loss的计算公式使单级结构达到和Faster RCNN一样的准确度，公式1是Focal Loss的计算方法。p_t是不同类别的分类概率，r是个大于0的值，a_t是个[0，1]间的小数，r和a_t都是固定值，不参与训练。从表达式可以看出：

1、无论是前景类还是背景类，p_t越大，权重$(1-p_t)^r$就越小。也就是说easy example可以通过权重进行抑制；

2、a_t用于调节positive和negative的比例，前景类别使用a_t时，对应的背景类别使用$1-a_t$；

3、r和a_t的最优值是相互影响的，所以在评估准确度时需要把两者组合起来调节。作者在论文中给出r=2、a_t=0.25时，ResNet-101+FPN作为backbone的结构有最优的性能。

10.smooth L1 loss

训练RPN时，只对两种anchor给予正标签：和gt_box有着最高的IoU && IoU超过0.7。如果对于

所有的gt_box，其IoU都小于0.3，则标记为负。损失函数定义如下：

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*). \quad (1)$$

其中i为一个mini-batch中某anchor的索引，p_i表示其为目标的预测概率,p_i^*表示gt_box（正为1，否则为0）。

t_i和t_i^*分别表示预测框的位置和gt_box框的位置。L_{reg}如下：

$$L_{\text{loc}}(t^u, v) = \sum_{i\in\{\text{x,y,w,h}\}} \text{smooth}_{L_1}(t_i^u - v_i), \quad (2)$$

in which

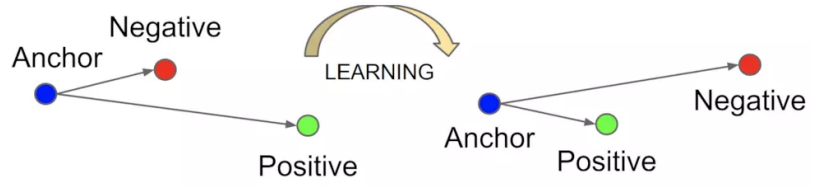
$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases} \quad (3)$$

为什么用smooth L1?

对于边框的预测是一个回归问题，通常可以选择平方损失函数（L2损失），f(x)=x^2.但这个损失对于比较大的误差（离群点）惩罚比较高，对噪声点比较敏感，所以采用稍微缓和一点的l1损失，f(x)=|x|，它随着误差线性增长而不是平方增长。但是这个函数在0点处导数不唯一，因此可能会影响收敛，所以在0点附近使用平方函数使得它更加平滑。

11. triplet loss

Triplet loss用于训练差异性较小的样本，如人脸等，输入数据包括Anchor positive negative。通过优化锚示例与正示例的距离小于锚示例与负示例的距离，实现样本的相似性计算。



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

这里距离用欧式距离度量，+表示[]内的值大于零的时候，取该值为损失，小于零的时候，损失为零。

12.center loss

center loss来自ECCV2016的一篇论文：A Discriminative Feature Learning Approach for Deep Face Recognition。

论文链接：http://ydwen.github.io/papers/WenECCV16.pdf

代码链接：https://github.com/pangyupo/mxnet_center_loss

12.1 softmax loss

下面公式1中log函数的输入就是softmax的结果（是概率），而Ls表示的是softmax loss的结果（是损失）。wx+b是全连接层的输出，因此log的输入就表示xi属于类别yi的概率。

$$\mathcal{L}_S = -\sum_{i=1}^{m} \log \frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^{n} e^{W_j^T x_i + b_j}} \tag{1}$$

In Equation 1, $\boldsymbol{x}_i \in \mathbb{R}^d$ denotes the ith deep feature, belonging to the y_ith class. d is the feature dimension. $W_j \in \mathbb{R}^d$ denotes the jth column of the weights $W \in \mathbb{R}^{d \times n}$ in the last fully connected layer and $\boldsymbol{b} \in \mathbb{R}^n$ is the bias term. The size of mini-batch and the number of class is m and n, respectively. We omit

12.2 center loss

$$\mathcal{L}_C = \frac{1}{2}\sum_{i=1}^{m} \|\boldsymbol{x}_i - \boldsymbol{c}_{y_i}\|_2^2$$

cyi表示第yi个类别的特征中心，xi表示全连接层之前的特征。后面会讲到实际使用的时候，m表示mini-batch的大小。因此这个公式就是希望一个batch中的每个样本的feature离feature的中心的距离的平方和要越小越好，也就是类内距离要越小越好。这就是center loss。

关于LC的梯度和cyi的更新公式如下：

The gradients of $\mathcal{L}_C$ with respect to $\boldsymbol{x}_i$ and update equation of $\boldsymbol{c}_{y_i}$ are computed as:

$$\frac{\partial \mathcal{L}_C}{\partial \boldsymbol{x}_i} = \boldsymbol{x}_i - \boldsymbol{c}_{y_i} \tag{3}$$

$$\Delta \boldsymbol{c}_j = \frac{\sum_{i=1}^{m} \delta(y_i = j) \cdot (\boldsymbol{c}_j - \boldsymbol{x}_i)}{1 + \sum_{i=1}^{m} \delta(y_i = j)} \tag{4}$$

这个公式里面有个条件表达式如下式，这里当condition满足的时候，下面这个式子等于1，当不满足的时候，下面这个式子等于0.

$$\delta(condition)$$

因此上面关于cyi的更新的公式中，当yi（表示yi类别）和cj的类别j不一样的时候，cj是不需要更新的，只有当yi和j一样才需要更新。

作者文中用的损失L的包含softmax loss和center loss，用参数lambda控制二者的比重，如下式所示。这里的m表示mini-batch的包含的样本数量，n表示类别数。

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

Algorithm 1 The discriminative feature learning algorithm

Input: Training data $\{\boldsymbol{x}_i\}$. Initialized parameters θ_C in convolution layers. Parameters W and $\{\boldsymbol{c}_j|j=1,2,...,n\}$ in loss layers, respectively. Hyperparameter λ, α and learning rate μ^t. The number of iteration $t \leftarrow 0$.

Output: The parameters θ_C.

1: **while** not converge **do**
2: $t \leftarrow t+1$.
3: Compute the joint loss by $\mathcal{L}^t = \mathcal{L}_S^t + \mathcal{L}_C^t$.
4: Compute the backpropagation error $\frac{\partial \mathcal{L}^t}{\partial \boldsymbol{x}_i^t}$ for each i by $\frac{\partial \mathcal{L}^t}{\partial \boldsymbol{x}_i^t} = \frac{\partial \mathcal{L}_S^t}{\partial \boldsymbol{x}_i^t} + \lambda \cdot \frac{\partial \mathcal{L}_C^t}{\partial \boldsymbol{x}_i^t}$.
5: Update the parameters W by $W^{t+1} = W^t - \mu^t \cdot \frac{\partial \mathcal{L}^t}{\partial W^t} = W^t - \mu^t \cdot \frac{\partial \mathcal{L}_S^t}{\partial W^t}$.
6: Update the parameters $\boldsymbol{c}_j$ for each j by $\boldsymbol{c}_j^{t+1} = \boldsymbol{c}_j^t - \alpha \cdot \Delta \boldsymbol{c}_j^t$.
7: Update the parameters θ_C by $\theta_C^{t+1} = \theta_C^t - \mu^t \sum_i^m \frac{\partial \mathcal{L}^t}{\partial \boldsymbol{x}_i^t} \cdot \frac{\partial \boldsymbol{x}_i^t}{\partial \theta_C^t}$.
8: **end while**