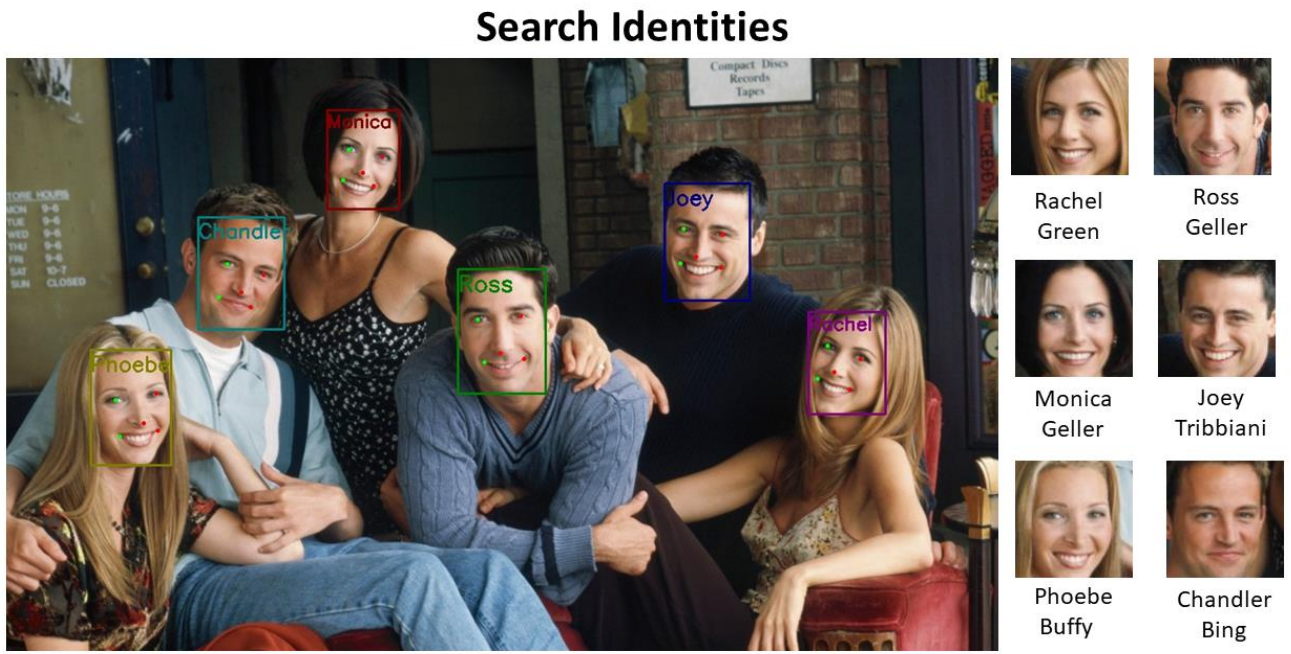
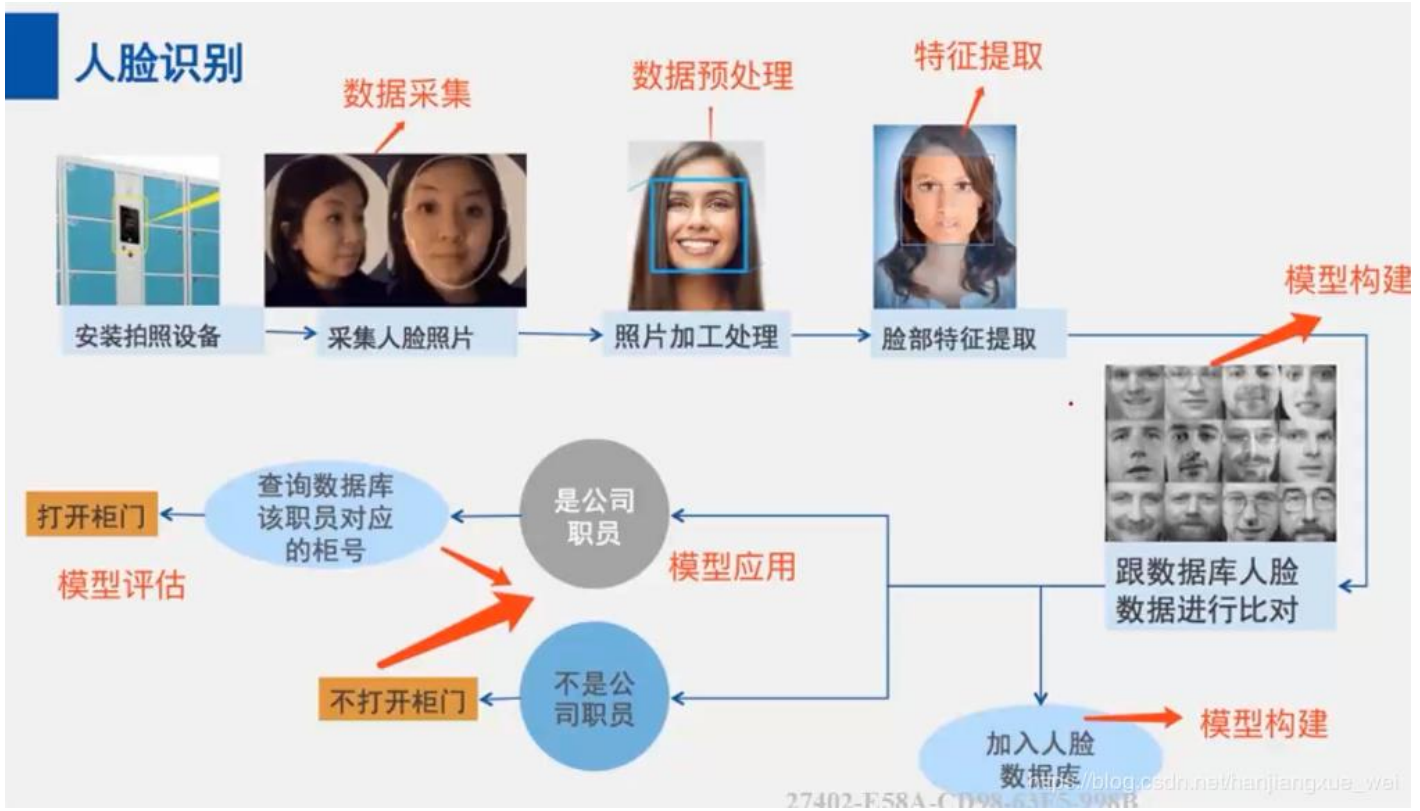
人脸识别

Additive Angular Margin Loss

人脸识别，是基于人的脸部特征信息进行身份识别的一种生物识别技术。用摄像机或摄像头采集含有人脸的图像或视频流，并自动在图像中检测和跟踪人脸，进而对检测到的人脸进行脸部识别的一系列相关技术，通常也叫做人像识别、面部识别。



人脸识别常分为四个过程:人脸检测，人脸对齐，特征提取，特征匹配。首先检测图片中的人脸位置，对其做预处理（例如对齐），再进行特征提取，提取后对比两张人脸的特征，计算其距离，设定一阈值，小于该阈值则判断不是同一个人，大于则认为是同一个人。



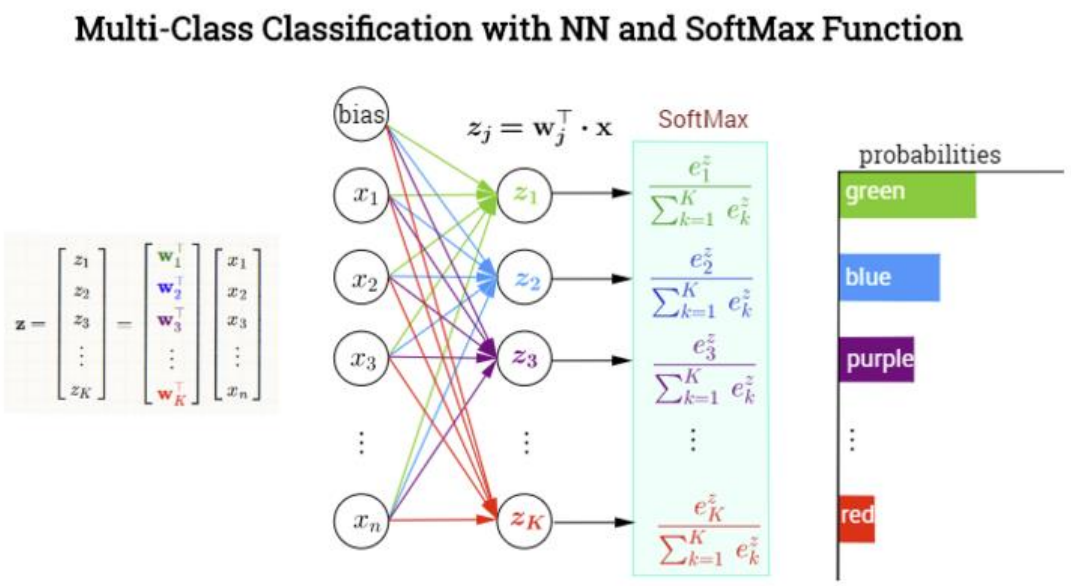
特征提取通常可以认为是人脸识别最关键的步骤，我们希望提取到的特征更偏向于该人脸“独有”的特征。我们的网络和模型承担着提取特征的重任，优秀的网络和训练策略使模型更加健壮。

在ResNet在2015年被提出后，越来越多优秀的网络基于ResNet进行优化更新也已取得卓越的成就，而在网络结构进一步升级优化有困难的情况下，研究者逐步将目光转向损失函数。

关于Loss对于网络的影响，最直观的就是训练中通过计算Loss反传梯度来实现对模型参数的更新。因此不同的Loss可以使模型更加侧重于学习到数据某一方面的特性，并在之后能够更好地提取到这一“独有”的特征，Loss对于网络优化有导向性的作用。

文章[ArcFace:Additive Angular Margin Loss for Deep Face Recognition]的作者提出了Additive Angular Margin Loss。在继SoftmaxLoss、Center Loss、A-Softmax Loss、Cosine Margin Loss之后有好的表现。

Loss函数：神经网络模型的效果及优化的目标是通过损失函数来定义的。通常解决多分类问题最常用的方法是设置n个输出节点，其中n为类别的个数。每一个输出节点对应一个类别。在理想情况下，如果一个样本属于类别k，那么这个类别所对应的输出节点的输出值应该为1，其他输出节点的输出值为0。例如对识别数字来说，以识别1为例，模型的输出结果越接近[0,1,0,0,0,0,0,0,0,0]越好。

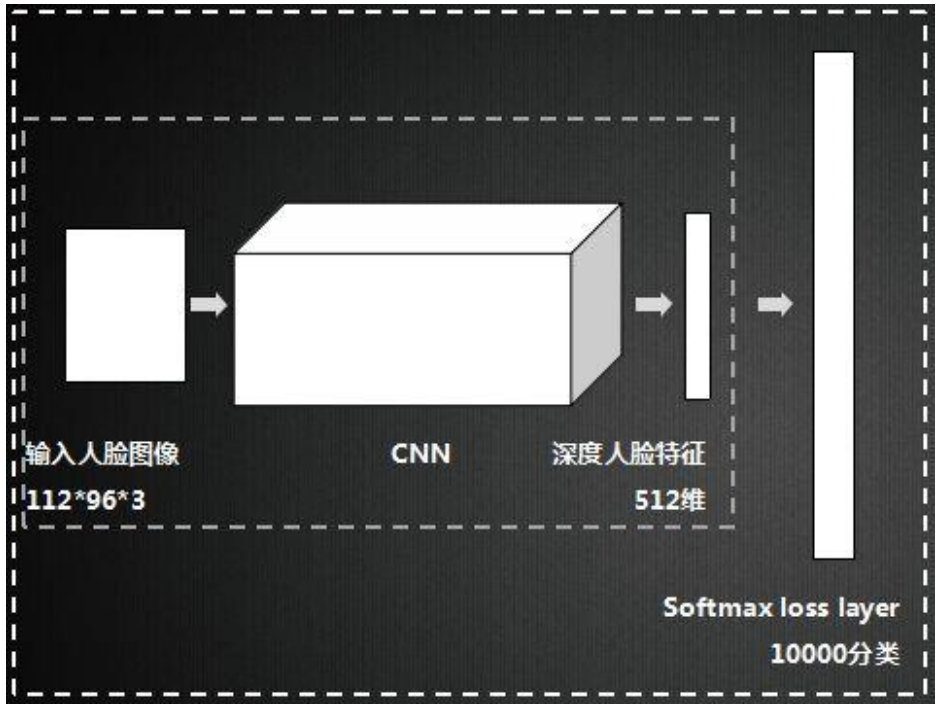


假设原始的神经网络输出为y1,y2,...,yn,那么经过softmax回归处理后的输出为：

$$\text{soft max}(y)_i = y_i^{'} = \frac{e^{yi}}{\sum_{j=1}^{n} e^{yi}}$$

处理后输出课看作一个概率分布，使用交叉熵计算与预想输出的距离。

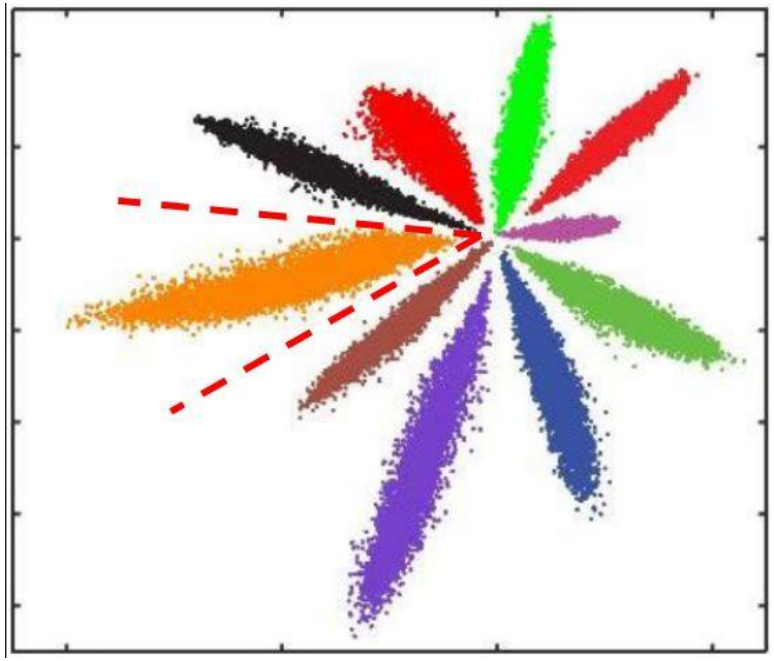
常用的softmax loss:



$$L_S = -\frac{1}{m}\sum_{i=1}^{m}\log\left(\frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^{n} e^{W_j^T x_i + b_j}}\right)$$

这种方式主要保证训练集的样本是否能正确分类，这在例如MNIST数字识别之类的任务中取得了非常好的效果。但是人脸识别问题有所不同，人脸识别属于开集分类问题(open-set problem)，在测试使用时所输入的人脸类别是在训练集中没有出现过的。使用Softmax Loss就会出现一些问题。

在2016年由Y.Wen等人完成的论文 *A discriminative feature learning approach for deep face recognition* 中，做了一个基于MNIST的实验，将数据提取为两维特征，可视化后得到下面的图。

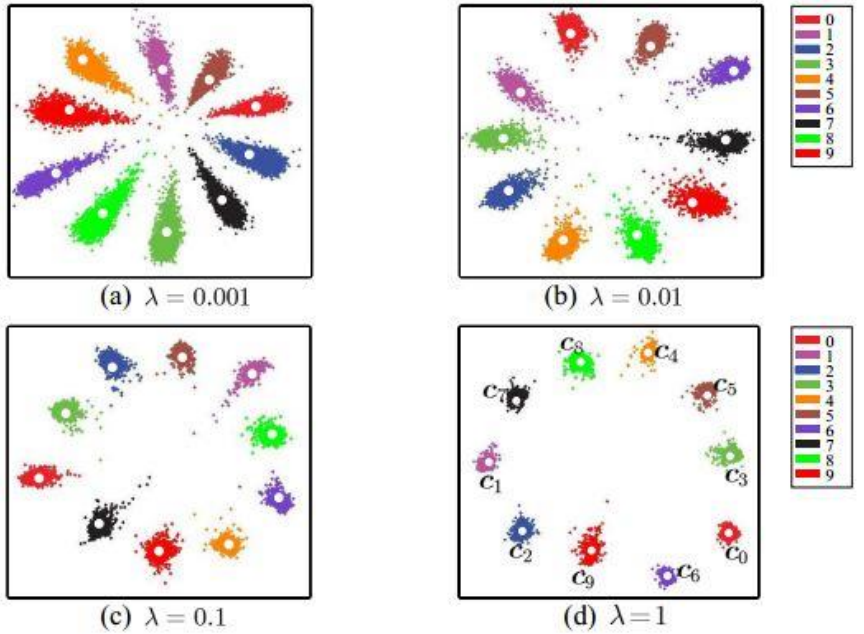


可以看到，类之间确实有清晰的划分，但是没有强调增大类间距离，减小类内距离。

改进的Center Loss:

$$\mathcal{L} = \mathcal{L}_S + \lambda \mathcal{L}_C$$

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



Center Loss的整体思想是希望一个batch中的每个样本的feature离这个类的feature 的中心的距离要越小越好，也就是类内距离要越小越好。

之后，研究者们将W，特征f归一化，并去掉偏置项，特征层输出则可以视为cosθ。

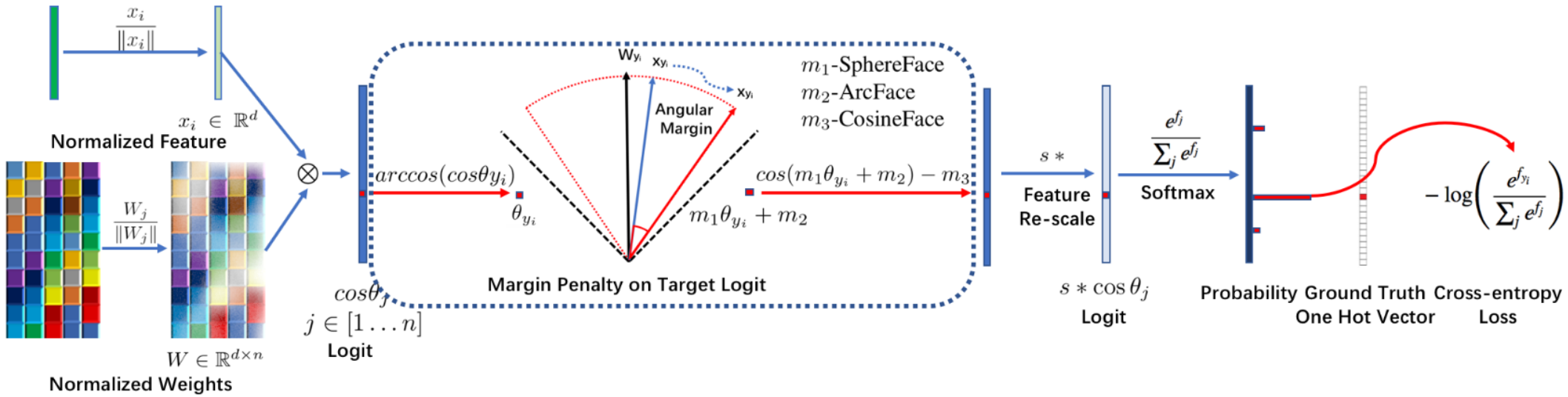
A-Softmax Loss(SphereFace):θ乘以决策余量m，进行权重归一化，并将偏置项归零（||W_i||=1，b_i=0）

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

cosine Margin Loss：使用尺度系数s,并使用cosθ-t.

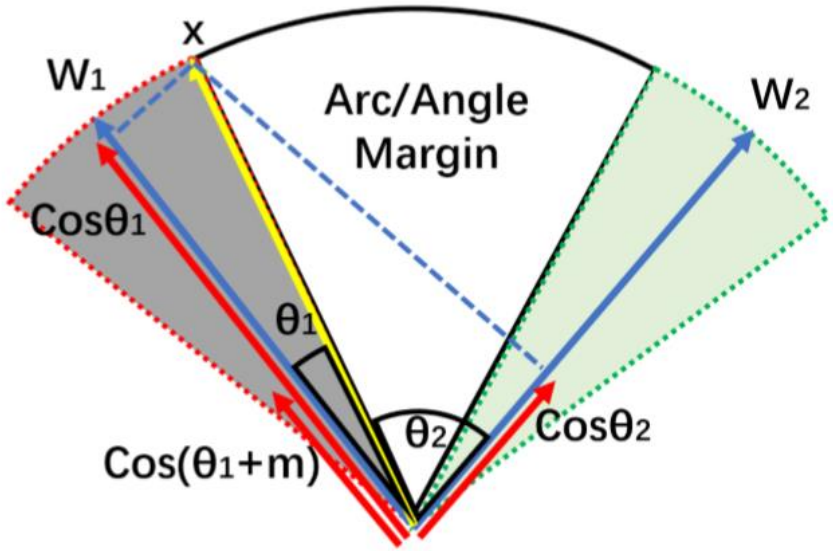
$$L_{Cosine} = -\frac{1}{m}\sum_{i=1}^{m}\log\left(\frac{e^{s\cdot(cos(\theta_{y_i})-t)}}{e^{s\cdot(cos(\theta_{y_i})-t)} + \sum\limits_{j=1,j=y_i}^{n} e^{s\cdot cos\theta_j}}\right)$$

如下是ArcFace的输入到输出流程：resnet model输出特征，archead将特征与权重间加上角度间隔后，再输出预测标签，softmax loss求预测标签和实际的误差。



DCNN特征和最后一个完全连接层之间的点积等于特征和权重归一化后的余弦距离。我们利用arc-cosine函数来计算当前特征和目标权重之间的角度。然后，在目标角上加上一个附加的角度间隔，用余弦函数重新计算逻辑回归的反向传播过程。然后，我们用一个固定的特征范数重新缩放所有的逻辑，随后的步骤与Softmax loss 中的步骤完全相同。

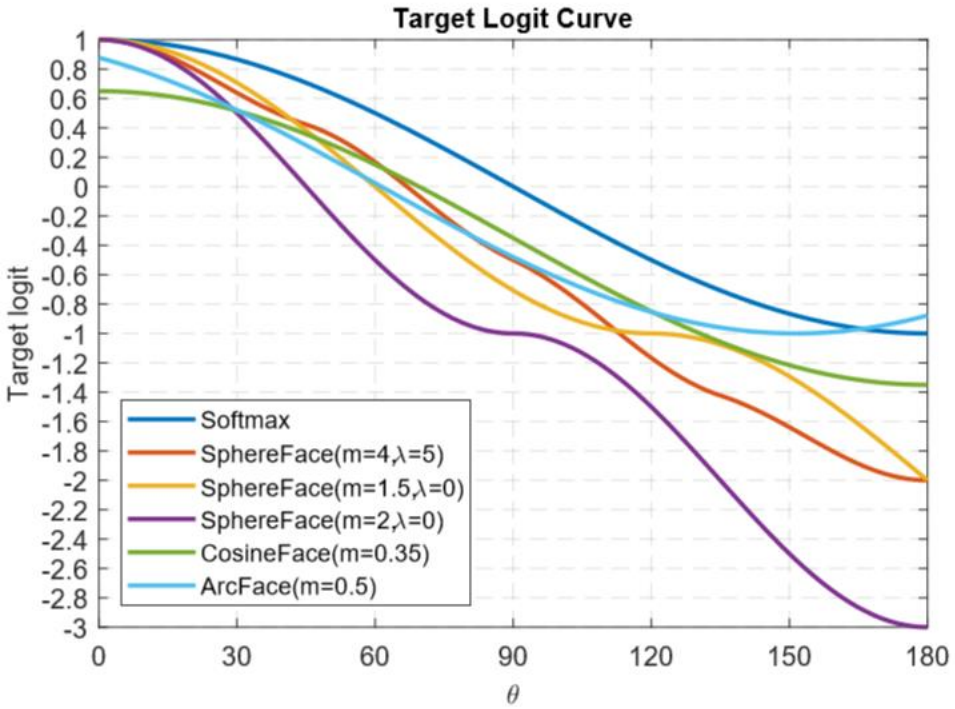
假设一特征向量没使用arcface，输出为 $(\cos\theta_1, \cos\theta_2, \cos\theta_3, \cos\theta_4, \cos\theta_5)$,对应类为3，变换后则为 $(\cos\theta_1, \cos\theta_2, \cos(\theta_3+\mathrm{m}), \cos\theta_4, \cos\theta_5)$
假如把特征向量看作分布在一个超球上，通过对θ+m的操作，可以使得原本的对应输出更小，空间的角度变大，从而增加训练难度，增加向类中心的聚集。



因此作者提出了Angular Margin Loss,其损失公式为

$$L_{Arcface} = -\frac{1}{m}\sum_{i=1}^{m}\log\left(\frac{e^{s\cdot(cos(\theta_{y_i}+t))}}{e^{s\cdot(cos(\theta_{y_i}+t))}+\sum_{j=1,j\neq y_i}^{n}e^{s\cdot cos\theta_j}}\right)$$

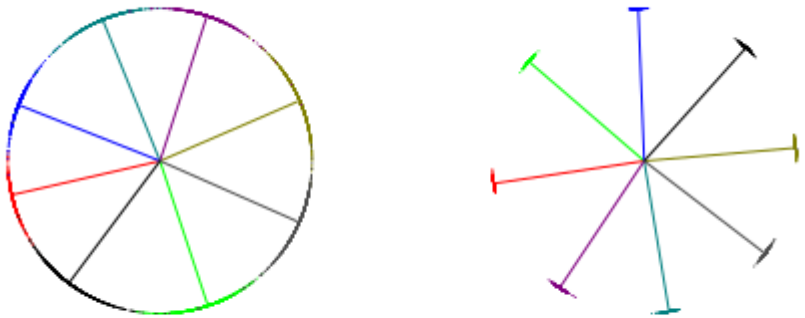
作者还对比了不同loss，可以看到SphereFace的惩罚显得过大，训练中难以收敛，而Arcface则下降较少容易收敛。



(a) Target Logit Curves

```python
def arcface_loss(embedding, labels, out_num, w_init=None, s=64., m=0.5):
    '''
    :param embedding: the input embedding vectors
    :param labels:  the input labels, the shape should be eg: (batch_size, 1)
    :param s: scalar value default is 64
    :param out_num: output class num
    :param m: the margin value, default is 0.5
    :return: the final cacualted output, this output is send into the tf.nn.softmax directly
    '''
    cos_m = math.cos(m)
    sin_m = math.sin(m)
    mm = sin_m * m  # issue 1
    threshold = math.cos(math.pi - m)
    with tf.variable_scope('arcface_loss'):
        # inputs and weights norm
        embedding_norm = tf.norm(embedding, axis=1, keep_dims=True)
        embedding = tf.div(embedding, embedding_norm, name='norm_embedding')
        weights = tf.get_variable(name='embedding_weights', shape=(embedding.get_shape().as_list()[-1], out_num),
                                  initializer=w_init, dtype=tf.float32)
        weights_norm = tf.norm(weights, axis=0, keep_dims=True)
        weights = tf.div(weights, weights_norm, name='norm_weights')
        # cos(theta+m)
        cos_t = tf.matmul(embedding, weights, name='cos_t')
        cos_t2 = tf.square(cos_t, name='cos_2')
        sin_t2 = tf.subtract(1., cos_t2, name='sin_2')
        sin_t = tf.sqrt(sin_t2, name='sin_t') # sint=(1-cost)^2再开方
        cos_mt = s * tf.subtract(tf.multiply(cos_t, cos_m), tf.multiply(sin_t, sin_m), name='cos_mt')

        # this condition controls the theta+m should in range [0, pi]
        #      0<=theta+m<=pi
        #     -m<=theta<=pi-m
        cond_v = cos_t - threshold
        cond = tf.cast(tf.nn.relu(cond_v, name='if_else'), dtype=tf.bool)

        keep_val = s*(cos_t - mm)
        cos_mt_temp = tf.where(cond, cos_mt, keep_val)

        mask = tf.one_hot(labels, depth=out_num, name='one_hot_mask')
        # mask = tf.squeeze(mask, 1)
        inv_mask = tf.subtract(1., mask, name='inverse_mask')

        s_cos_t = tf.multiply(s, cos_t, name='scalar_cos_t')

        output = tf.add(tf.multiply(s_cos_t, inv_mask), tf.multiply(cos_mt_temp, mask), name='arcface_loss_output')
    return output
```

作者还做了一个小实验，使用8个包含足够样本的不同身份人脸，分别训练具有Softmax loss和ArcFace loss 的网络。如下图所示，Softmaxloss提供了可分离的特征，但在边界会产生模糊性。而ArcFace loss显然在类之间产生明显的差距。



(a) Softmax (b) ArcFace

MegaFace作为百万规模级别的面部识别算法的测试基准，作者以LResNet100E-IR作为网络结构，以MS1M作为训练集，分别对比了不同损失函数在MegaFace上的表现，其中最后两列括号中有“R”是代表清洗过的MegaFace，可以看到ArcFace的结果达到了state-of-the-art。

Methods	Rank1@10^6	VR@FAR10^{-6}	Rank1@10^6 (R)	VR@FAR10^{-6} (R)
Softmax	78.89	94.95	91.43	94.95
Softmax-pretrain,Triplet-finetune	80.6	94.65	94.08	95.03
Softmax-pretrain@VGG2, Triplet-finetune	78.87	95.43	93.96	95.07
SphereFace(m=4, λ=5)	82.95	97.66	97.43	97.66
CosineFace(m=0.35)	82.75	98.41	98.33	98.41
ArcFace(m=0.4)	82.29	98.20	98.10	97.83
ArcFace(m=0.5)	**83.27**	**98.48**	**98.36**	**98.48**

引用参考文献：

J.Deng,J.Guo and N.Xue.ArcFace:Additive Angular Margin Loss for Deep Face Recognition.
Y.Wen,K.Zhang,Z.Li and Y.Qiao.A Discriminative Feature Learning Approach for Deep Face Recognition.
W.Liu,Y.Wen,Z.Yu,M.Li,B.Raj and L.Song.SphereFace: Deep Hypersphere Embedding for Face Recognition

视频源码来源：

https://github.com/auroua/InsightFace_TF

谢谢