ArcFace

Arc Face

Besides the architecture, the evaluation method is also of great importance to achieve a good performance. Softmax Cross Entropy is often used in classification tasks, including face classification. Evaluation is like a supervisor to the neural network and pushes the network to reach the criterion. So, we can also change the evaluation method to make improvement because higher and harder criterion can make perfect!

Softmax

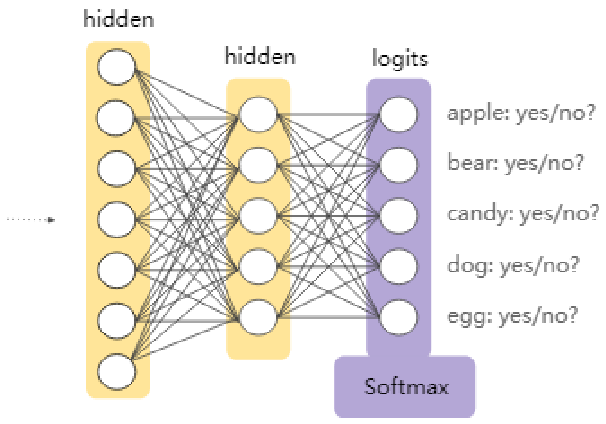

Figure 1

Softmax is one of the most useful and common functions used in classification. It is also known as normalized exponential function. The softmax takes as input a vector z , and normalizes it into a probability distribution portional to the exponentials of the input numbers. Prior to applying softmax, some vector components could be any number but after going through softmax, each component will be in the interval [0,1], and the components will add up to 1, so that they can be interpreted as probabilities. **In a word, softmax transfer the regression result to probability.**

Here is the function of softmax:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

However, we are familiar with the exponential function that if z is huge, it will cause overflow in computer, like $\sigma(1000) = inf$ or if z is small, $\sigma(-1000) = 0$. When all the z is small enough, the denominator will be zero and it is underflow.

To prevent overflow and underflow, we can subtract each element by the maximum element:

$$\sigma(z)_i = \frac{e^{z_i - z_{max}}}{\sum_{j=1}^{K} e^{z_j - z_{max}}}$$

Softmax Cross Entropy Loss

Once we get the probability, we should compute the loss. Here we introduce a loss named Cross Entropy Loss. It metrics the difference between the probability and the ground truth, which is a one-hot vector. Like $\sigma(z) = [0.2, 0.7, 0.1]$ and $gt = [0, 1, 0]$. The equation of CE loss:

$$CE = \sum_{i=1}^{K} -gt_i log(\sigma(z)_i)$$

Because gt is one-hot vector, we can simplify the equation to:

$$CE = -log(\sigma(z)_{gt})$$

ArcFace

https://arxiv.org/pdf/1801.07698.pdf

If we hope to improve the performance of classification starting from the aspect of evaluation, we should process the distribution of each sample.

- [ ] Enhance the intra-class compactness
- [ ] Enhance the inter-class discrepancy

Let's look at the most common evaluation the Softmax Cross Entropy Loss:

$$L = -log(\sigma_{gt}) = -log(\frac{e^{z_{gt}}}{\sum_{i=1}^{K} e^{z_i}})$$

$$= -log(\frac{e^{W^T x + b}}{\sum_{i=1}^{K} e^{W_i^T x + b_i}})$$

$$= -log(\frac{e^{||W_{gt}^T||*||x||*cos(\theta_{W_{gt},x})+b_{gt}}}{\sum_{i=1}^{K} e^{||W_i^T||*||x||*cos(\theta_{W_i,x})+b_i}})$$

$\theta_{i,j} \in [0, \pi]$ represents the angle between two vectors i and j. If we normalize W_i, set bias term b to zero and fix the embedding feature $||x_i||$ by l_2 normalization and re-scale it to s , which means $||W_i|| = 1$, $b_i = 0$ and $||x_i|| = s$, then we have:

$$L_{modified} = -log(\frac{e^{s*cos(\theta_{W_{gt},x})}}{\sum_{i=1}^{K} e^{s*cos(\theta_{W_i,x})}})$$

The normalization step on features and weights makes the predictions only depend on the angle between the feature and the weight. The learned embedding features are thus distributed on a hypersphere with a radius of s.

$$L_{modified} = -log(\frac{e^{s*cos(\theta_{W_{gt},x})}}{e_{s*cos(\theta_{W_{gt},x})} + \sum_{i=1,i\neq gt}^{K} e^{s*cos(\theta_{W_i,x})}})$$

As the embedding features are distributed around each feature center on the hypersphere, we add an additive angular margin penalty m between x_i and W_{gt} to simultaneously enhance the intra-class compactness and inter-class discrepancy.

$$L_{arc} = -log(\frac{e^{s*cos(\theta_{W_{gt},x}+m)}}{e_{s*cos(\theta_{W_{gt},x}+m)} + \sum_{i=1,i\neq gt}^{K} e^{s*cos(\theta_{W_i,x})}})$$

Here is a diagram about the whole procedure:

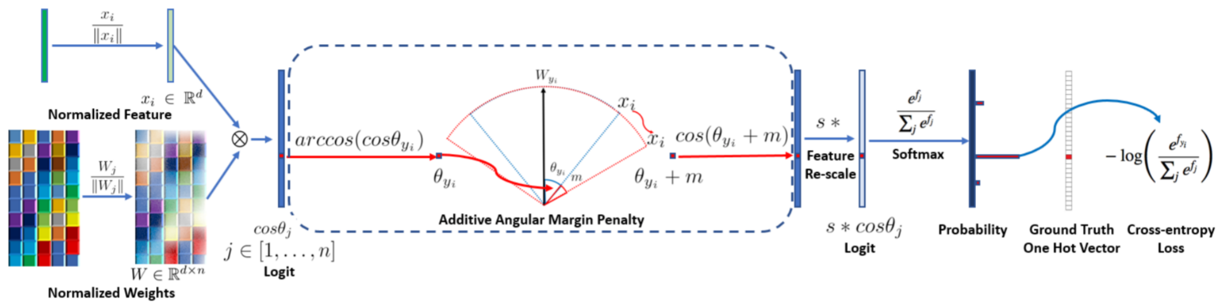

Figure 2

The authors in the paper select face images from 8 different identities containing enough samples (around 1,500 images/class) to train 2-D feature embedding networks with the softmax and ArcFace loss, respectively. As illustrated in Figure 3, the softmax loss provides roughly separable feature embedding but produces noticeable ambiguity in decision boundaries, while the proposed ArcFace loss can obvious more evident gap between the nearest classes.

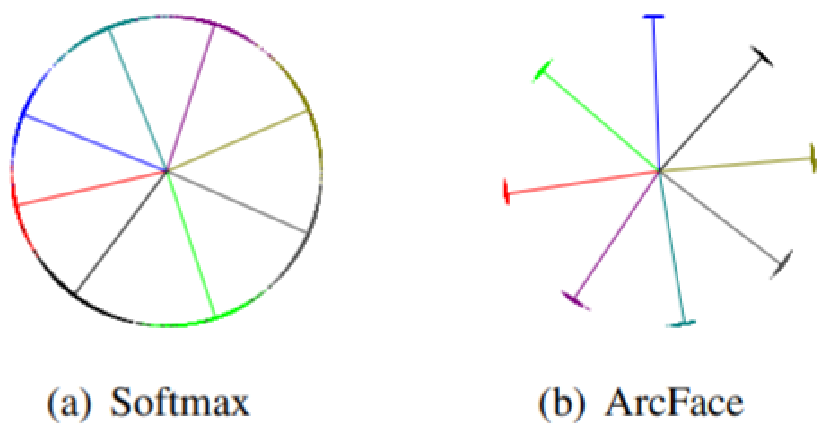

Figure 3

Dots indicate samples and lines refer to the center direction of each identity. The geodesic distance gap between closest classes becomes evident as the additive angular margin penalty is incorporated.

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.nn import Parameter
import math


# ArcFace
class ArcMarginProduct(nn.Module):
    """
    Implement of large margin arc distance (ArcFace)
    Args:
        in_features: size of each input sample
        out_features: size of each output sample
        s: norm of input feature
        m: margin

        cos(theta+m)
    """

    def __init__(self, in_features, out_features, s=30.0, m = 0.50, easy_margin=False):
        super(ArcMarginProduct, self).__init__()
        self.in_features = in_features
        self.out_features = out_features
        self.s = s
        self.m = m
        self.weight = Parameter(torch.FloatTensor(out_features, in_features))
        nn.init.xavier_normal_(self.weight)
        self.easy_margin = easy_margin
        self.cos_m = math.cos(m)
        self.sin_m = math.sin(m)
        self.th = math.cos(math.pi - m)
        self.mm = math.sin(math.pi - m) * m

    def forward(self, input, label):
        # ---------------------cos(theta) &  phi(theta) ----------------------
        # torch.nn.functional.linear(input, weight, bias=None)
        # y = x * W^T + b
        cosine = F.linear(F.normalize(input), F.normalize((self.weight)))
        sine = torch.sqrt(1.0 - torch.pow(cosine, 2))
        # cos(a + b) = cos(a) * cos(b) - sine(a) * sine(b)
        phi = cosine * self.cos_m - sine * self.sin_m
        if self.easy_margin:
            # torch.where(condition, x, y) -> Tensor
            # condition(ByteTensor) - When True (nonzero), yield x, otherwise yield y
            # x (Tensor) - values selected at indices where condition is True
            # y (Tensor) - values selected at indices where condition is False
            # return:
            # A tensor of shape equal to the broadcasted shape of condition, x, y
            # cosine > 0 means two class is similar, thus use the phi which make it
            phi = torch.where(cosine > 0, phi, cosine)
        else:
            phi = torch.where(cosine > self.th, phi, cosine - self.mm)
        # --------------------- convert label to one-hot ----------------------
        # one-hot = torch.zeros(cosine.size(), requires_grad=True, device='cuda')
        # update cos(theta+m) to tensor
        one_hot = torch.zeros_like(cosine, device='cuda')
        # scatter_(dim, index, src)
        one_hot.scatter_(1, label.view(-1, 1).long(), 1)
        # -------------------- torch.where(out_i = x_i if condition_i else y_i) ---------------------
        output = (one_hot * phi) + ((1.0 - one_hot) * cosine)
        output *= self.s

        return output
```

SphereFace and CosFace

https://arxiv.org/pdf/1704.08063.pdf

https://arxiv.org/pdf/1801.09414.pdf

There are two loss function prior to ArcFace and they all proposed different kinds of margin penalty to the $\theta_{i,j}$.

Here are the loss functions of SphereFace and CosFace and you can find more interesting details in the papers.

$$L_{sphere} = -log(\frac{e^{s*cos(m*\theta_{W_{gt},x})}}{e^{s*cos(m*\theta_{W_{gt},x})} + \sum_{i=1,i\neq gt}^{K} e^{s*cos(\theta_{W_i,x})}})$$

$$L_{cos} = -log(\frac{e^{s*(cos(\theta_{W_{gt},x})-m)}}{e^{s*(cos(\theta_{W_{gt},x})-m)} + \sum_{i=1,i\neq gt}^{K} e^{s*cos(\theta_{W_i,x})}})$$

We can find out that ArcFace and SphereFace tries to maximize the decision boundaries in angular space, while CosFace considers the cosine space. What's different is that ArcFace and CosFaec are both additive margin and SphereFace is multiplicative Margin.

As illustrated in Figure 5, compare the decision boundaries under the binary classification case. The proposed ArcFace has a constant linear angular margin throughout the whole interval. By contrast, SphereFace and CosFace only have a nonlinear angular margin

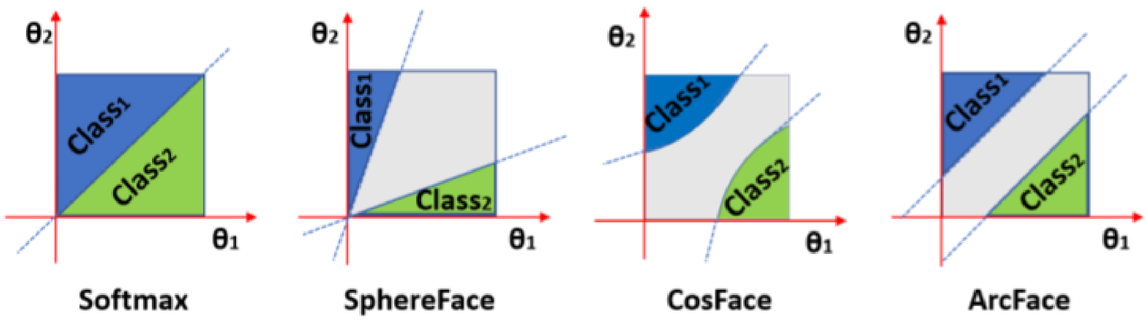

Figure 5

If you are interested in these two loss function, you can try to implement it by yourself!