Additive Margin Softmax for Face Verification

Feng Wang
UESTC
feng.wff@gmail.com

Weiyang Liu
Georgia Tech
wyliu@gatech.edu

Haijun Liu
UESTC
haijun_liu@126.com

Jian Cheng
UESTC
chengjian@uestc.edu.cn

Abstract

In this paper, we propose a conceptually simple and geometrically interpretable objective function, i.e. additive margin Softmax (AM-Softmax), for deep face verification. In general, the face verification task can be viewed as a metric learning problem, so learning large-margin face features whose intra-class variation is small and inter-class difference is large is of great importance in order to achieve good performance. Recently, Large-margin Softmax [10] and Angular Softmax [9] have been proposed to incorporate the angular margin in a multiplicative manner. In this work, we introduce a novel additive angular margin for the Softmax loss, which is intuitively appealing and more interpretable than the existing works. We also emphasize and discuss the importance of feature normalization in the paper. Most importantly, our experiments on LFW and MegaFace show that our additive margin softmax loss consistently performs better than the current state-of-the-art methods using the same network architecture and training dataset. Our code has also been made available[1].

1. Introduction

Face verification is widely used for identity authentication in enormous areas such as finance, military, public security and so on. Nowadays, most face verification models are built upon Deep Convolutional Neural Networks and supervised by classification loss functions [18, 20, 19, 9], metric learning loss functions [16] or both [17, 13]. Metric learning loss functions such as contrastive loss [17] or triplet loss [16] usually require carefully designed sample mining strategies and the final performance is very sensitive to these strategies, so increasingly more researchers shift their attentions to building deep face verification models based on improved classification loss functions [20, 19, 9].

Current prevailing classification loss functions for deep face recognition are mostly based on the widely-used softmax loss. The softmax loss is typically good at optimizing the inter-class difference (i.e., separating different classes), but not good at reducing the intra-class variation (i.e., making features of the same class compact). To address this, lots of new loss functions are proposed to minimize the intra-class variation. [20] proposed to add a regularization term to penalize the feature-to-center distances. In [19, 12, 15], researchers proposed to use a scale parameter to control the "temperature" [2] of the softmax loss, producing higher gradients to the well-separated samples to further shrink the intra-class variance. In [9, 10], the authors introduced an conceptually appealing angular margin to push the classification boundary closer to the weight vector of each class. [9] also provided a theoretical guidance of training a deep model for metric learning tasks using the classification loss functions. [6, 12, 15] also improved the softmax loss by incorporating differnet kinds of margins.

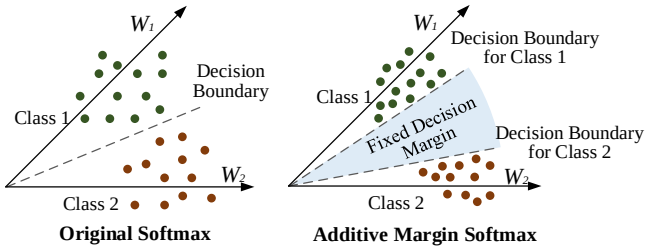

Figure 1. Comparison between the original softmax loss and the additive margin softmax loss. Note that, the angular softmax [9] can only impose unfixed angular margin, while the additive margin softmax incorporates the fixed hard angular margin.

In this work, we propose a novel and more interpretable way to import the angular margin into the softmax loss. We formulate an additive margin via $\cos\theta - m$, which is simpler than [9] and yields better performance. From Equation (3), we can see that m is multiplied to the target angle θ_{y_i} in [9], so this type of margin is incorporated in a multiplicative manner. Since our margin is a scalar subtracted from $cos\theta$, we call our loss function Additive Margin Softmax (AM-Softmax).

Experiments on LFW BLUFR protocol [7] and MegaFace [5] show that our loss function with the same network architecture achieves better results than the current state-of-the-art approaches.

[1] https://github.com/happynear/AMSoftmax

2. Preliminaries

To better understand the proposed AM-Softmax loss, we will first give a brief review of the original softmax and the A-softmax loss [9]. The formulation of the original softmax loss is given by

$$\begin{aligned}\mathcal{L}_S &= -\frac{1}{n}\sum_{i=1}^{n} log\frac{e^{W_{y_i}^T \boldsymbol{f}_i}}{\sum_{j=1}^{c} e^{W_j^T \boldsymbol{f}_i}} \\ &= -\frac{1}{n}\sum_{i=1}^{n} log\frac{e^{\|W_{y_i}\|\|\boldsymbol{f}_i\|cos(\theta_{y_i})}}{\sum_{j=1}^{c} e^{\|W_j\|\|\boldsymbol{f}_i\|cos(\theta_j)}},\end{aligned} \tag{1}$$

where $\boldsymbol{f}$ is the input of the last fully connected layer ($\boldsymbol{f}_i$ denotes the the i-th sample), W_j is the j-th column of the last fully connected layer. The $W_{y_i}^T \boldsymbol{f}_i$ is also called as the target logit [14] of the i-th sample.

In the A-softmax loss, the authors proposed to normalize the weight vectors (making $\|W_i\|$ to be 1) and generalize the target logit from $\|\boldsymbol{f}_i\|cos(\theta_{y_i})$ to $\|\boldsymbol{f}_i\|\psi(\theta_{y_i})$,

$$\mathcal{L}_{AS} = -\frac{1}{n}\sum_{i=1}^{n} log\frac{e^{\|\boldsymbol{f}_i\|\psi(\theta_{y_i})}}{e^{\|\boldsymbol{f}_i\|\psi(\theta_{y_i})} + \sum_{j=1,j\neq y_i}^{c} e^{\|\boldsymbol{f}_i\|cos(\theta_j)}}, \tag{2}$$

where the $\psi(\theta)$ is usually a piece-wise function defined as

$$\begin{aligned}\psi(\theta) &= \frac{(-1)^k \cos(m\theta) - 2k + \lambda cos(\theta)}{1+\lambda}, \\ &\theta \in [\frac{k\pi}{m}, \frac{(k+1)\pi}{m}],\end{aligned} \tag{3}$$

where m is usually an integer larger than 1 and λ is a hyper-parameter to control how hard the classification boundary should be pushed. During training, the λ is annealing from $1,000$ to a small value to make the angular space of each class become more and more compact. In their experiments, they set the minimum value of λ to be 5 and $m = 4$, which is approximately equivalent to $m = 1.5$ (Figure 2).

3. Additive Margin Softmax

In this section, we will first describe the definition of the proposed loss function. Then we will discuss about the intuition and interpretation of the loss function.

3.1. Definition

[10] defines a general function $\psi(\theta)$ to introduce the large margin property. Motivated by that, we further propose a specific $\psi(\theta)$ that introduces an additive margin to the softmax loss function. The formulation is given by

$$\psi(\theta) = cos\theta - m. \tag{4}$$

Compared to the $\psi(\theta)$ defined in L-Softmax [10] and A-softmax [9] (Equation (3)), our definition is more simple and intuitive. During implementation, the input after normalizing both the feature and the weight is actually $x = cos\theta_{y_i} = \frac{W_{y_i}^T f_i}{\|W_{y_i}\|\|f_i\|}$, so in the forward propagation we only need to compute

$$\Psi(x) = x - m. \tag{5}$$

In this margin scheme, we don't need to calculate the gradient for back-propagation because $\Psi'(x) = 1$. It is much easier to implement compared with SphereFace [9].

Since we use cosine as the similarity to compare two face features, we follow [19, 11, 12] to apply both feature normalization and weight normalization to the inner product layer in order to build a cosine layer. Then we scale the cosine values using a hyper-parameter s as suggested in [19, 11, 12]. Finally, the loss function becomes

$$\begin{aligned}\mathcal{L}_{AMS} &= -\frac{1}{n}\sum_{i=1}^{n} log\frac{e^{s\cdot(cos\theta_{y_i}-m)}}{e^{s\cdot(cos\theta_{y_i}-m)} + \sum_{j=1,j\neq y_i}^{c} e^{s\cdot cos\theta_j}} \\ &= -\frac{1}{n}\sum_{i=1}^{n} log\frac{e^{s\cdot(W_{y_i}^T \boldsymbol{f}_i-m)}}{e^{s\cdot(W_{y_i}^T \boldsymbol{f}_i-m)} + \sum_{j=1,j\neq y_i}^{c} e^{sW_j^T \boldsymbol{f}_i}}.\end{aligned} \tag{6}$$

In this paper, we assume that the norm of both W_i and $\boldsymbol{f}$ are normalized to 1 if not specified. In [19], the authors propose to let the scaling factor s to be learned through back-propagation. However, after the margin is introduced into the loss function, we find that the s will not increase and the network converges very slowly if we let s to be learned.

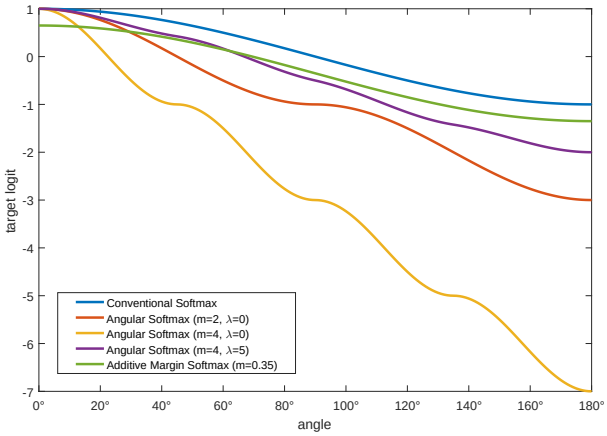

Figure 2. $\psi(\theta)$ for conventional Softmax, Angular Softmax [9] and our proposed Hard Margin Softmax. For Angular Softmax, we plot the logit curve for three parameter sets. From the curves we can infer that $m = 4, \lambda = 5$ lies between conventional Softmax and Angular Softmax with $m = 2, \lambda = 0$, which means it is approximately $m = 1.5$. Our proposed Additive Margin Softmax with optimized parameter $m = 0.35$ is also plotted and we can observe that it is similar with Angular Softmax with $m = 4, \lambda = 5$ in the range $[0°, 90°]$, in which most of the real-world θs lie.

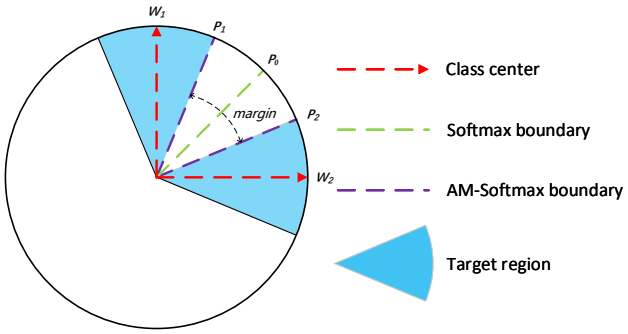

Figure 3. Conventional Softmax's decision boundary and Additive Margin Softmax's decision boundary. For conventional softmax, the decision boundary is at P_0, where $W_1^T P_0 = W_2^T P_0$. For AM-Softmax, the decision boundary for class 1 is at P_1, where $W_1^T P_1 - m = W_2^T P_1 = W_1^T P_2$. Note that the distance marked on this figure doesn't represent the real distances. The real distance is a function of the cosine of the angle, while in this figure we use the angle as the distance for better visualization effect. Here we use the word "center" to represent the weight vector of the corresponding class in the last inner-product layer, even though they may not be exactly the mean vector of the features in the class. The relationship between the weight vector ("agent") and the features' mean vector ("center") is described in Figure 6 of [19].

Thus, we fix s to be a large enough value, e.g. 30, to accelerate and stablize the optimization.

As described in Section 2, [10, 9] propose to use an annealing strategy to set the hyper-parameter λ to avoid network divergence. However, to set the annealing curve of λ, lots of extra parameters are introduced, which are more or less confusing for starters. Although properly tuning those hyper-parameters for λ could lead to impressive results, the hyper-parameters are still quite difficult to tune. With our margin scheme, we find that we no longer need the help of the annealing strategy. The network can converge flexibly even if we fix the hyper-parameter m from scratch. Compared to SphereFace [9], our additive margin scheme is more friendly to those who are not familiar with the effects of the hyper-parameters. Another recently proposed additive margin is also described in [6]. Our AM-Softmax is different than [6] in the sense that our feature and weight are normalized to a predefined constant s. The normalization is the key to the angular margin property. Without the normalization, the margin m does not necessarily lead to large angular margin.

3.2. Discussion

3.2.1 Geometric Interpretation

Our additive margin scheme has a clear geometric interpretation on the hypersphere manifold. In Figure 3, we draw a schematic diagram to show the decision boundary of both conventional softmax loss and our AM-Softmax. For example, in Figure 3, the features are of 2 dimensions. After normalization, the features are on a circle and the decision boundary of the traditional softmax loss is denoted as the vector P_0. In this case, we have $W_1^T P_0 = W_2^T P_0$ at the decision boundary P_0.

For our AM-Softmax, the boundary becomes a marginal region instead of a single vector. At the new boundary P_1 for class 1, we have $W_1^T P_1 - m = W_2^T P_1$, which gives $m = (W_1 - W_2)^T P_1 = \cos(\theta_{W_1,P_1}) - \cos(\theta_{W_2,P_1})$. If we further assume that all the classes have the same intra-class variance and the boundary for class 2 is at P_2, we can get $\cos(\theta_{W_2,P_1}) = \cos(\theta_{W_1,P_2})$ (Fig. 3). Thus, $m = \cos(\theta_{W_1,P_1}) - \cos(\theta_{W_1,P_2})$, which is the difference of the cosine scores for class 1 between the two sides of the margin region.

3.2.2 Angular Margin or Cosine Margin

In SphereFace [9], the margin m is multiplied to θ, so the angular margin is incorporated into the loss in a multiplicative way. In our proposed loss function, the margin is enforced by subtracting m from $\cos\theta$, so our margin is incorporated into the loss in an additive way, which is one of the most significant differences than [9]. It is also worth mentioning that despite the difference of enforcing margin, these two types of margin formulations are also different in the base values. Specifically, one is θ and the other is $\cos\theta$. Although usually the cosine margin has an one-to-one mapping to the angular margin, there will still be some difference while optimizing them due to the non-linearity induced by the cosine function.

Whether we should use the cosine margin depends on which similarity measurement (or distance) the final loss function is optimizing. Obviously, our modified softmax loss function is optimizing the cosine similarity, not the angle. This may not be a problem if we are using the conventional softmax loss because the decision boundaries are the same in these two forms ($\cos\theta_1 = \cos\theta_2 \Rightarrow \theta_1 = \theta_2$). However, when we are trying to push the boundary, we will face a problem that these two similarities (distances) have different densities. Cosine values are more dense when the angles are near 0 or π. If we want to optimize the angle, an $\arccos$ operation may be required after the value of the inner product $W^T \boldsymbol{f}$ is obtained. It will potentially be more computationally expensive.

In general, angular margin is conceptually better than the cosine margin, but considering the computational cost, cosine margin is more appealing in the sense that it could achieve the same goal with less efforts.

3.2.3 Feature Normalization

In the SphereFace model [9], the authors added the weight normalization based on Large Margin Softmax [10], leaving

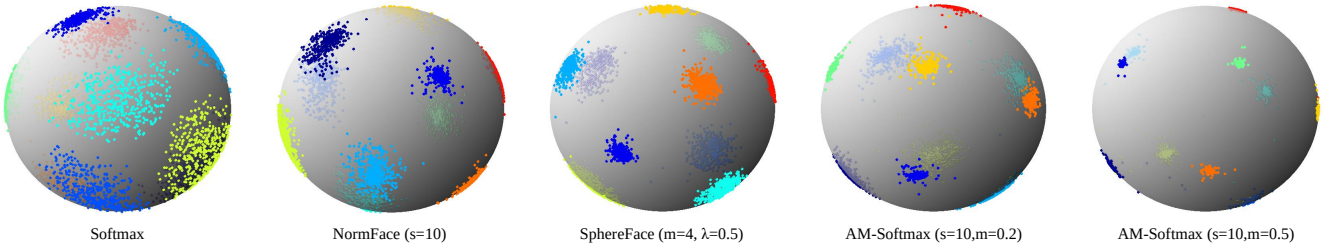

Figure 4. Feature distribution visualization of several loss functions. Each point on the sphere represent one normalized feature. Different colors denote different classes. For SphereFace [9], we have already tried to use the best hyper-parameters we could find.

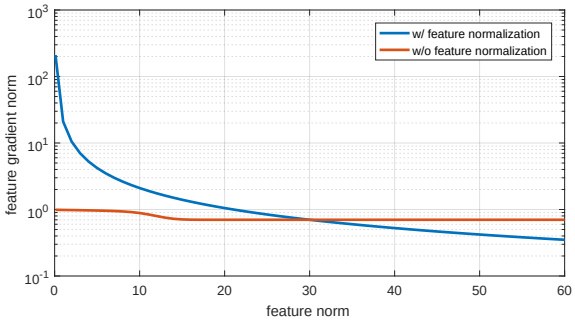

Figure 5. The feature gradient norm w.r.t. the feature norm for softmax loss with and without feature normalization. The gradients are calculated using the weights from a converged network. The feature direction is selected as the mean vector of one selected target center and one nearest class center. Note that the y-axis is in logarithmic scale for better visualization. For softmax loss with feature normalization, we set $s = 30$. That is why the intersection of these two curves is at 30.

the feature still not normalized. Our loss function, following [19, 12, 15], applies feature normalization and uses a global scale factor s to replace the sample-dependent feature norm in SphereFace [9]. One question arises: when should we add the feature normalization?

Our answer is that it depends on the image quality. In [15]'s Figure 1, we can see that the feature norm is highly correlated with the quality of the image. Note that back propagation has a property that,

$$y = \frac{x}{\alpha} \quad \Rightarrow \quad \frac{dy}{dx} = \frac{1}{\alpha}. \tag{7}$$

Thus, after normalization, features with small norms will get much bigger gradient compared with features that have big norms (Figure 5). By back-propagation, the network will pay more attention to the low-quality face images, which usually have small norms. Its effect is very similar with hard sample mining [16, 8]. The advantages of feature normalization are also revealed in [11]. As a conclusion, feature normalization is most suitable for tasks whose image quality is very low.

From Figure 5 we can see that the gradient norm may be extremely big when the feature norm is very small. This potentially increases the risk of gradient explosion, even though we may not come across many samples with very small feature norm. Maybe some re-weighting strategy whose feature-gradient norm curve is between the two curves in Figure 5 could potentially work better. This is an interesting topic to be studied in the future.

3.2.4 Feature Distribution Visualization

To better understand the effect of our loss function, we designed a toy experiment to visualize the feature distributions trained by several loss functions. We used Fashion MNIST [21] (10 classes) to train several 7-layer CNN models which output 3-dimensional features. These networks are supervised by different loss functions. After we obtain the 3-dimensional features, we normalize and plot them on a hypersphere (ball) in the 3 dimensional space (Figure 4).

From the visualization, we can empirically show that our AM-Softmax performs similarly with the best SphereFace [9] (A-Softmax) model when we set $s = 10, m = 0.2$. Moreover, our loss function can further shrink the intra-class variance by setting a larger m, while A-Softmax starts to degrade when $\lambda < 0.5$. Compared to A-Softmax [9], the AM-Softmax loss also converges easier with proper scaling factor s. The visualized 3D features well demonstrates that AM-Softmax could bring the large margin property to the features without tuning too many hyper-parameters.

4. On Automatic Hyper-Parameter Tuning

In our AM-Sotmax loss, we introduce two hyper-parameters into the conventional softmax loss function. However, tuning hyper-parameters is usually very tricky and time-consuming. Similar problems also appear in SphereFace [9]. In this section, we will provide some preliminary experiments and discussions about how to automatically tuning the hyper-parameters m and s. We hope that our observation could provide some useful insights and may inspire future works.

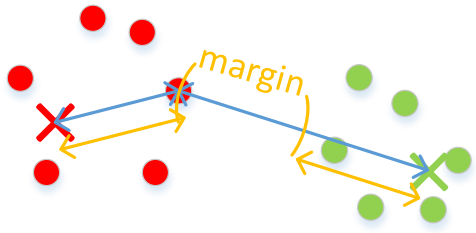

Figure 6. Illustration of the latent margin for a single sample. Note that the weight vectors ($\times$) are not at the center of the clusters. They will shift to be far away from other feature vectors. We call these weight vectors as the agents of their corresponding class (See [19] for details).

4.0.1 Latent Margin

To make things clear, we call the modified loss function proposed in [19] as *Cosine Softmax Loss* because it is optimizing the cosine similarity instead of the traditional inner-product similarity. The following analysis is based on the Cosine Softmax Loss. Following our convention, we assume that all the W_i and $\mathbf{f}$ are normalized to 1.

In the two-class (labeled as "0" and "1") scenario, the Cosine Softmax Loss for one sample is defined as,

$$\mathcal{L}_{CS2} = -log\frac{e^{s\cdot W_1^T\mathbf{f}}}{e^{s\cdot W_1^T\mathbf{f}} + e^{s\cdot W_2^T\mathbf{f}}}. \tag{8}$$

Here we assume that the ground truth label for this sample is 1. Through optimization, this loss function will push $W_1^T\mathbf{f}$ higher and make $W_2^T\mathbf{f}$ lower, so it will create a margin between $W_1^T\mathbf{f}$ and $W_2^T\mathbf{f}$ (Figure 6). Since the margin is not directly corresponding to Euclidean space or angular space, we call this margin as *latent margin*,

$$m_{latent} = W_1^T\mathbf{f} - W_2^T\mathbf{f}. \tag{9}$$

By collecting all the target and non-target logits, we can compute the empirical distribution of latent margin for a binary classification model. The latent margin empirical distribution reflects the discriminative capacity of the features learned by this model. The latent margin may not necessarily improve the accuracy for a classification task. But for metric learning tasks (e.g., face verification), the margin is very crucial for the improving the performance (Refer to Property 2 in [9]).

4.0.2 The LogSumExp Function

For the case of multiple classes, it is more complicated to analyze the latent margin because we will have $C-1$ non-target logits for every single sample. To simplify the analysis, we propose to use an equivalent form of the Cosine Softmax loss function. Therefore, we introduce the LogSumExp function[2]:

$$LSE(\mathbf{f}; s) = log(\sum_{j=1, j\neq y}^{c} e^{sW_j^T\mathbf{f}}), \tag{10}$$

to replace the $C-1$ non-target terms in the denominator of softmax loss. In this way, the softmax operation becomes,

$$\frac{e^{sW_{y_i}^T\mathbf{f}}}{e^{sW_{y_i}^T\mathbf{f}} + \sum_{j=1,j\neq y_i}^{c} e^{sW_j^T\mathbf{f}}} = \frac{e^{sW_{y_i}^T\mathbf{f}}}{e^{sW_{y_i}^T\mathbf{f}} + e^{LSE(\mathbf{f};s)}}. \tag{11}$$

Comparing this formulation with Equation 8, we can know that softmax loss is actually making a "binary" classification between $sW_y^T\mathbf{f}$ and $LSE(\mathbf{f}; s)$, i.e. softmax loss will push $sW_y^T\mathbf{f}$ higher and $LSE(\mathbf{f}; s)$ lower. With the help of the LogSumExp function, we transfer the multi-class softmax into a "binary" classifier, so that we can utilize some properties we have discussed before. Similar to the two-class scenario, we can also get a latent margin distribution by aggregating all the

$$m_{latent} = W_y^T\mathbf{f} - \frac{LSE(\mathbf{f}; s)}{s}. \tag{12}$$

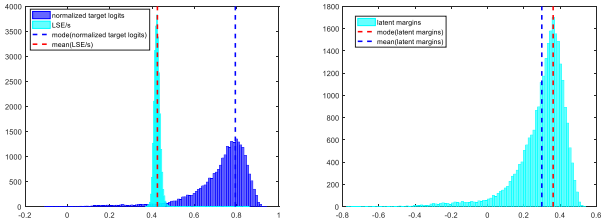

Figure 7. *Left:* The distribution of both normalized target logits and $\frac{LSE(\mathbf{f};\|\mathbf{f}\|)}{\|\mathbf{f}\|}$ for AM-Softmax loss with $s = 30, m = 0.35$. *Right:* The distribution of m_{latent}. The mean value (blue dashed line) of m_{latent} is about 0.299 and the mode value (red dashed line) is about 0.359.

4.0.3 Auto-tuning m

In Figure 7, we plot the distributions of $W_y^T\mathbf{f}$, $\frac{LSE(\mathbf{f};s)}{s}$ and the latent margins. As we empirically observe, m_{latent} follows a right-skewed distribution with a long-tail in the left, which cannot be precisely fit by a Gaussian distribution. The mean value is usually biased from the peak of the distribution. Therefore, we choose to use the mode instead of mean value to represent the distribution. The mode value can be obtained by Mean Shift algorithm with window size of 0.1.

For the weight normalized softmax loss [9] (without any angular margin), s should be replaced by $\|\mathbf{f}\|$. We have trained such a model with the same settings as Section 5.1.

[2] https://en.wikipedia.org/wiki/LogSumExp.

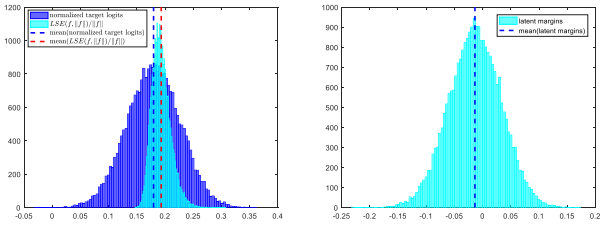

Figure 8. *Left:* The distribution of both normalized target logits and $\frac{LSE(\mathbf{f};\|\mathbf{f}\|)}{\|\mathbf{f}\|}$ for weight normalized softmax loss. *Right:* The distribution of m_{latent} for weight normalized softmax loss. The mean value of m_{latent} is about -0.0133.

We find that its mean latent margin is -0.0133, which indicates that the weight normalized softmax loss, as a classification loss, cannot create significant margin between $W_y^T\mathbf{f}$ and $\frac{LSE(\mathbf{f};\|\mathbf{f}\|)}{\|\mathbf{f}\|}$ (Figure 8).

When the features and weights are both normalized as [19] does, the loss function could force the model to create small latent margins between $W_y^T\mathbf{f}$ and $\frac{LSE(\mathbf{f};s)}{s}$. This is mainly because the Cosine Softmax Loss is optimizing the angle between features and weights, while both the original softmax loss and the weight normalized softmax loss mainly optimize the feature norm rather than the angle[19, 23, 10] after one sample is correctly classified.

After introducing the hyper-parameter m into Cosine Softmax Loss, the latent margin will continue to increase. In Figure 9, we show the relationship between the mode of latent margins and the manually set m. It can be observed that when $m = 0$, the model has a latent margin of about 0.1, which is a small but not ignorable value. Such margin is introduce due to the normalization of both feature and weight, as discussed in [19]. Moreover, we observe that there exists a point in range $(0.35, 0.4)$ where $m = m_{margin}$. Surprisingly, it can perfectly match the best m we found in experiments (See Table 5). This observation leads to an automatic hyper-parameter tuning for m. Specifically, we set $m = mode(m_{latent})$ on-the-fly during training. In the experiment section, we will verify such small trick can avoid tedious hyper-parameter tuning. The experiments show that the m finally converges to 0.378 and the model yields comparable performance with the other manually tuned m.

Even though this method is based on several observations and assumptions, it may still reveal some interesting properties of the softmax loss. This technical report is just a preliminary study for the automatical hyper-parameter tuning, and more in-depth study will be left for future works.

4.0.4 Auto-tuning s

As we said before in Section 3.1, the network converges very slowly if we let s to be learned. This was because we

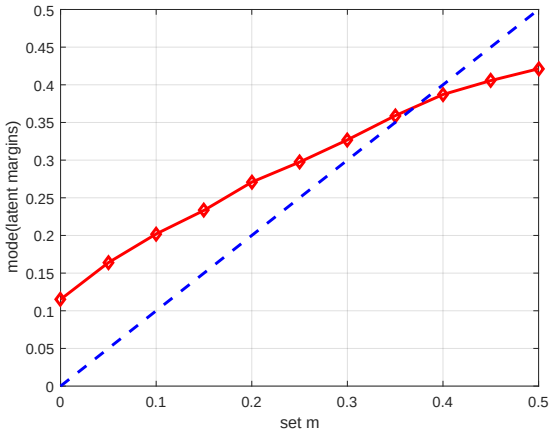

Figure 9. The mode of latent margins as a function of the manually set m.

fixed the parameter m, which will make the task defined by the loss too difficult during the early iterations of optimization. Now that we dynamically set the m according to the discriminative capacity of the model, we can let the s to be learned through back-propagation again as [19] does. Interestingly, the finaly s is about 38, which is just slightly higher than the previous best value (30) we set in experiments.

Learning the hyper-parameter s is just one way to avoid the hyper-parameter tuning. It may not necessarily be the best way to set the hyper-parameter s. Finding better ways to automatically tune s is one of our important future works.

5. Experiment

In this section, we will firstly describe the experimental settings. Then we will discuss the overlapping problem of the modern in-the-wild face datasets. Finally we will compare the performance of our loss function with several previous state-of-the-art loss functions.

5.1. Implementation Details

Our loss function is implemented using Caffe framework [4]. We follow all the experimental settings from [9], including the image resolution, preprocessing method and the network structure. Specifically speaking, we use MTCNN [24] to detect faces and facial landmarks in images. Then the faces are aligned according to the detected landmarks. The aligned face images are of size 112×96, and are normalized by subtracting 128 and dividing 128. Our network structure follows [9], which is a modified ResNet [1] with 20 layers that is adapted to face recognition.

All the networks are trained from scratch. We set the weight decay parameter to be $5e{-}4$. The batch size is 256 and the learning rate begins with 0.1 and is divided by 10 at the 16K, 24K and 28K iterations. The training is finished at

Loss Function	m	LFW[3] 6,000 pairs	LFW BLUFR[7] VR@FAR=0.01%	LFW BLUFR[7] VR@FAR=0.1%	LFW BLUFR[7] DIR@FAR=1%	MegaFace[5] Rank1@1e6	MegaFace[5] VR@FAR=1e−6
Softmax	-	97.08%	60.26%	78.26%	50.85%	45.26%	50.12%
Softmax+75% dropout	-	98.62%	77.64%	90.91%	63.72%	57.32%	65.58%
Center Loss [20]	-	99.00%	83.30%	94.50%	65.46%	63.38%	75.68%
NormFace [19]	-	98.98%	88.15%	96.16%	75.22%	65.03%	75.88%
A-Softmax [9]	∼1.5	99.08%	91.26%	97.06%	81.93%	67.41%	78.19%
AM-Softmax	0.25	99.13%	91.97%	97.13%	81.42%	70.81%	83.01%
AM-Softmax	0.3	99.08%	93.18%	97.56%	84.02%	72.01%	83.29%
AM-Softmax	0.35	98.98%	93.51%	97.69%	84.82%	**72.47%**	**84.44%**
AM-Softmax	0.4	99.17%	93.60%	97.71%	84.51%	72.44%	83.50%
AM-Softmax	0.45	99.03%	93.44%	97.60%	84.59%	72.22%	83.00%
AM-Softmax	0.5	99.10%	92.33%	97.28%	83.38%	71.56%	82.49%
AM-Softmax auto	∼0.38	99.05%	93.47%	97.51%	84.93%	**72.57%**	**84.17%**
AM-Softmax w/o FN	0.35	99.08%	93.86%	97.63%	**87.58%**	70.71%	82.66%
AM-Softmax w/o FN	0.4	99.12%	**94.48%**	**97.96%**	87.31%	70.96%	83.11%

Table 1. Performance on modified ResNet-20 with various loss functions. Note that, for Center Loss [20] and NormFace [19], we used modified ResNet-28 [20] because we failed to train a model using Center Loss on modified ResNet-20 [9] and the NormFace model was fine-tuned based on the Center Loss model.

30K iterations. During training, we only use image mirror to augment the dataset.

In testing phase, We feed both frontal face images and mirror face images and extract the features from the output of the first inner-product layer. Then the two features are summed together as the representation of the face image. When comparing two face images, cosine similarity is utilized as the measurement.

5.2. Dataset Overlap Removal

The dataset we use for training is CASIA-Webface [22], which contains 494,414 training images from 10,575 identities. To perform open-set evaluations, we carefully remove the overlapped identities between training dataset (CASIA-Webface [22]) and testing datasets (LFW[3] and MegaFace [5]). Finally, we find 17 overlapped identities between CASIA-Webface and LFW, and 42 overlapped identities between CASIA-Webface and MegaFace set1. Note that there are only 80 identities in MegaFace set1, i.e. over half of the identities are already in the training dataset. The effect of overlap removal is remarkable for MegaFace (Table 5.2). To be rigorous, all the experiments in this paper are based on the cleaned dataset. We have made our overlap checking code publicly available[3] to encourage researchers to clean their training datasets before experiments.

Loss Function	Overlap Removal?	MegaFace Rank1	MegaFace VR
AM-Softmax	No	75.23%	87.06%
AM-Softmax	Yes	72.47%	84.44%

Table 2. Effect of Overlap Removal on modified ResNet-20

[3] https://github.com/happynear/FaceDatasets

In our paper, we re-train some of the previous loss functions on the cleaned dataset as the baselines for comparison. Note that, we make our experiments fair by using the same network architecture and training dataset for every compared methods.

5.3. Effect of Hyper-parameter m

There are two hyper-parameters in our proposed loss function, one is the scale s and another is the margin m. The scale s has already been discussed sufficiently in several previous works [19, 12, 15]. In this paper, we directly fixed it to 30 and will not discuss its effect anymore.

The main hyper-parameter in our loss function is the margin m. In Table 5, we list the performance of our proposed AM-Softmax loss function with m varies from 0.25 to 0.5. From the table we can see that from $m = 0.25$ to 0.3, the performance improves significantly, and the performance become the best when $m = 0.35$ to $m = 0.4$.

We also provide the result for the loss function without feature normalization (noted as w/o FN) and the scale s. As we explained before, feature normalization performs better on low quality images like MegaFace[5], and using the original feature norm performs better on high quality images like LFW [3].

In Figure 10, we draw both of the CMC curves to evaluate the performance of identification and ROC curves to evaluate the performance of verification. From this figure, we can show that our loss function performs much better than the other loss functions when the rank or false positive rate is very low.

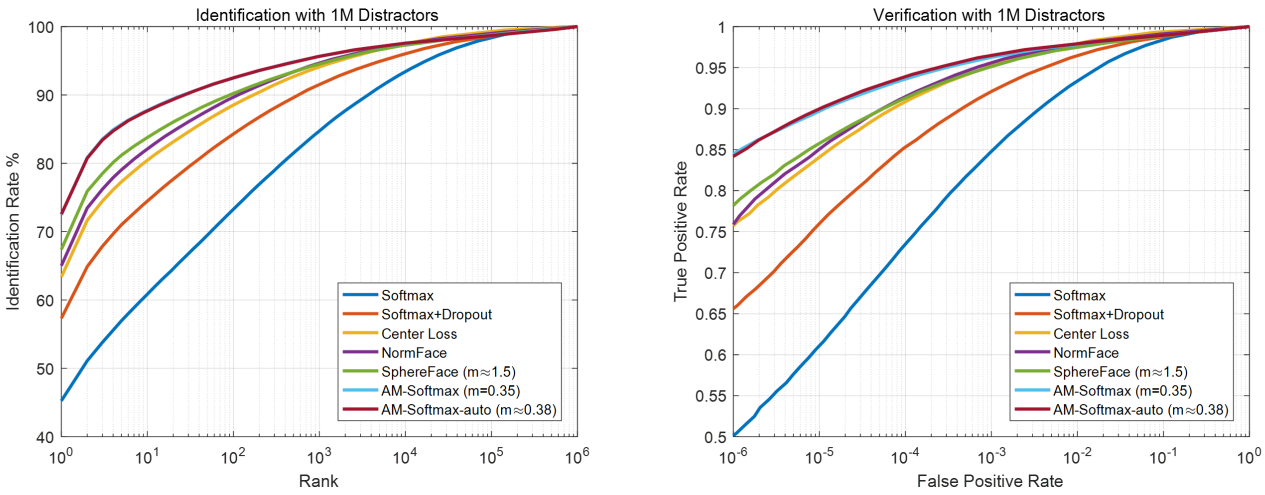

Figure 10. *Left:* CMC curves of different loss functions with 1M distractors on MegaFace[5] Set 1. *Right:* ROC curves of different loss functions with 1M distractors on MegaFace[5] Set 1. Note that for Center Loss and NormFace, the backend network is ResNet-28[20], while others are based on ResNet-20[9]. Even though the curves of the Center Loss model and the NormFace model is close to the SphereFace model, please keep in mind that part of the performance comes from the bigger network structure.

6. Conclusion and Future Work

In this paper, we propose to impose an additive margin strategy to the target logit of softmax loss with feature and weights normalized. Our loss function is built upon the previous margin schemes[9, 10], but it is more simple and interpretable. Comprehensive experiments show that our loss function performs better than A-Softmax [9] on LFW BLUFR [7] and MegaFace [5].

There is still lots of potentials for the research of the large margin strategies. There could be more creative way of specifying the function $\psi(\theta)$ other than multiplication and addition. In our AM-Softmax loss, the margin is a manually tuned global hyper-parameter. How to automatically determine the margin and how to incorporate class-specific or sample-specific margins remain open questions and are worth studying.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 6

[2] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. 1

[3] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical report, Technical Report 07-49, University of Massachusetts, Amherst, 2007. 7

[4] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014. 6

[5] I. Kemelmacher-Shlizerman, S. M. Seitz, D. Miller, and E. Brossard. The megaface benchmark: 1 million faces for recognition at scale. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4873–4882, 2016. 1, 7, 8

[6] X. Liang, X. Wang, Z. Lei, S. Liao, and S. Z. Li. Soft-margin softmax for deep classification. *24th International Conference on Neural Information Processing*, pages 413–421, 2017. 1, 3

[7] S. Liao, Z. Lei, D. Yi, and S. Z. Li. A benchmark study of large-scale unconstrained face recognition. In *IEEE International Joint Conference on Biometrics*, pages 1–8. IEEE, 2014. 1, 7, 8

[8] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002*, 2017. 4

[9] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song. Sphereface: Deep hypersphere embedding for face recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017. 1, 2, 3, 4, 5, 6, 7, 8

[10] W. Liu, Y. Wen, Z. Yu, and M. Yang. Large-margin softmax loss for convolutional neural networks. In *International Conference on Machine Learning*, pages 507–516, 2016. 1, 2, 3, 6, 8

[11] W. Liu, Y.-M. Zhang, X. Li, Z. Yu, B. Dai, T. Zhao, and L. Song. Deep hyperspherical learning. In *Advances in Neural Information Processing Systems*, pages 3953–3963, 2017. 2, 4

[12] Y. Liu, H. Li, and X. Wang. Rethinking feature discrimination and polymerization for large-scale recognition. *arXiv preprint arXiv:1710.00870*, 2017. 1, 2, 4, 7

[13] O. M. Parkhi, A. Vedaldi, and A. Zisserman. Deep face recognition. In *BMVC*, volume 1, page 6, 2015. 1

[14] G. Pereyra, G. Tucker, J. Chorowski, Ł. Kaiser, and G. Hinton. Regularizing neural networks by penalizing confident output distributions. *arXiv preprint arXiv:1701.06548*, 2017. 2

[15] R. Ranjan, C. D. Castillo, and R. Chellappa. L2-constrained softmax loss for discriminative face verification. *arXiv preprint arXiv:1703.09507*, 2017. 1, 4, 7

[16] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015. 1, 4

[17] Y. Sun, Y. Chen, X. Wang, and X. Tang. Deep learning face representation by joint identification-verification. In *Advances in neural information processing systems*, pages 1988–1996, 2014. 1

[18] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014. 1

[19] F. Wang, X. Xiang, J. Cheng, and A. L. Yuille. Normface: L2 hypersphere embedding for face verification. In *Proceedings of the 25th ACM international conference on Multimedia*. ACM, 2017. 1, 2, 3, 4, 5, 6, 7

[20] Y. Wen, K. Zhang, Z. Li, and Y. Qiao. A discriminative feature learning approach for deep face recognition. In *European Conference on Computer Vision*, pages 499–515. Springer, 2016. 1, 7, 8

[21] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017. 4

[22] D. Yi, Z. Lei, S. Liao, and S. Z. Li. Learning face representation from scratch. *arXiv preprint arXiv:1411.7923*, 2014. 7

[23] Y. Yuan, K. Yang, and C. Zhang. Feature incay for representation regularization. *arXiv preprint arXiv:1705.10284*, 2017. 6

[24] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, 2016. 6