Loss Function in ReID and Facial Recognition

Yujia Zhu

ReID aims at matching or retrieving objects with the same ID from different cameras, while facial recognition aims at matching a human face from a digital image or a video frame against a database of faces. Two tasks are very similar to each other.

Problem Definition

We can mainly devide the problem to **closed-set problem** and **open-set problem**. The data we get is the training pairs $\{(x_i, y_i)\}$, where x_i is the image with a matrix representation and y_i is the ID of the image x_i. Images with the same ID represent the same object or person respectively in re-id and facial recognition. For closed-set problem, we know all the IDs in advance, so we only need to predict which ID or class the image belongs to. This means all the ID appear in the test set will definitely appear in the training set. But for open-set problem, some identities in the test set will not appear in the training set. This increases the difficulty of dealing problems but is more common in the daily problem.

The solution for closed-set problem is quiet easy. We can directly view it as a classification problem where the number of categories equals to the number of IDs. For open-set problem, the most frequently-used solution under deep learning is to learn a representation vector via neural networks. In the stage of testing/matching, we allocate the image to the ID with the minimum distance. To learn the open-set problem better, we need to learn large-margin features through metric learning rather than a separable hyperplane in closed-set problem. But from my point of view, some large-margin features are learned from a classification loss with some tricks. So, metric learning problems are essentially the same as classifaction problems under some cases.

For open-set problem, problem-solving procedure is firstly to learn a representation mapping in the training stage, namely learn a mapping function f from the image to a low-dimension space $f : X \to R^d$ and then make decision via the distance among representation vectors in the test stage. The core is learning the mapping function f. For deep learning, f is mainly determined by **the design of the loss function** and **the design of network**. We will focus on the design of the loss function since then.

Classifaction Loss Function

Softmax Loss

For a closed-set problem, we can treat it as a classification problem. Softmax loss function is the most commonly-used loss function in the classification problem. Suppose the input of the last hidden layer is x_i and the weight and bais of the layer are w_i and b_i corresponding to the ID y_i. The softmax loss function is defined as

$$L_i = -\log\left(\frac{\exp(w_{y_i} x_i + b_i)}{\sum_{j=1}^{n} \exp(w_j x_i + b_j)}\right)$$

For open-set problem, we can still use this loss function. This means we actually treats it as a closed-set problem but it sometimes still works when we use the output of the last hidden layer as the representation vector. This is beacuse of the strong feature-extracting ability of the neural network. But, since softmax loss function only requires separablity, so the distance among different IDs are not very far from each other. When adding some unseen IDs in the test stage, some misclassification cases will emerge.

L-softmax Loss

As we have seen, the softmax loss function of the i-th instance is

$$L_i = -\log\left(\frac{\exp(w_{y_i}x_i + b_i)}{\sum_j \exp(w_j x_i + b_j)}\right)$$

Suppose the bias $b_i = 0$, then we can transform the linear transformation to the inner product $w_{y_i}x_i = \|w_{y_i}\|\|x_i\|\cos\theta_{y_i}$, where θ_j is the angle between w_j and x_i. So, the loss function can be changed to

$$L_i = -\log\left(\frac{\exp(\|w_{y_i}\|\|x_i\|\cos(\theta_{y_i}))}{\sum_j \exp(\|w_j\|\|x_i\|\cos(\theta_j))}\right)$$

Let us move back to a binary classifaction problem and we sample x from class 1. Then we will correctly classify x under softmax if $w_1x > w_2x$. This condition equals to $\|w_1\|\|x\|\cos\theta_1 > \|w_2\|\|x\|\cos\theta_2$. For L-softmax loss function, we require a more rigorous condition to classify the sample x to class 1 if

$$\|w_1\|\|x\|\cos(m\theta_1) > \|w_2\|\|x\|\cos\theta_2$$

This means the angle between x and w_1 needs to be much larger than the angle between x and w_2, so the distance between class 1 and class 2 will be larger thus predicting well (So L-softmax function is also called Large-margin softmax loss function). In addition, when $0 \le \theta \le \frac{\pi}{m}$, we have $\|w_1\|\|x\|\cos(\theta_1) > \|w_1\|\|x\|\cos(m\theta_1) > \|w_2\|\|x\|\cos\theta_2$. So if an instance is classified correctly under L-softmax loss function, it will also be classified correctly under softmax function. But, we need to mention that the requirements of L-softmax loss function is more strict. For $\theta_1 \ge \frac{\pi}{m}$, we can generalize the loss function to

$$L_i = -\log\left(\frac{\exp\left(\|w_{y_i}\|\|x_i\|\phi(\theta_{y_i})\right)}{\exp\left(\|w_{y_i}\|\|x_i\|\phi(\theta_{y_i})\right) + \sum_{j\neq y_i}\exp\left(\|w_j\|\|x_i\|\cos\theta_j\right)}\right)$$

where we require $\phi(.)$ to be a continous and monotonically decreasing function with the following form

$$\phi(\theta) = \begin{cases}\cos(m\theta) & ,0 \le \theta \le \frac{\pi}{m}\\ D(\theta) & ,\frac{\pi}{m} \le \theta \le \pi\end{cases}$$

To be simple, we can design $\phi(\theta)$ as

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

m is a hyperparameter which controls the margin among different IDs. For $m = 1$, the L-softmax loss function will degrade to softmax loss function. Empirical results verify that with L-softmax loss function, the distance among different IDs is much larger.

A-softmax Loss

From L-softmax loss, we can see that the loss function of the i-th training sample is

$$L_i = -\log\left(\frac{\exp\left(w_{y_i}x_i + b_{y_i}\right)}{\sum_j \exp\left(w_j x_i + b_j\right)}\right) = -\log\left(\frac{\exp\left(\|w_{y_i}\|\|x_i\|\cos(\theta_{y_i,i}) + b_{y_i}\right)}{\sum_j \exp\left(\|w_j\|\|x_i\|\cos(\theta_{j,i}) + b_j\right)}\right)$$

We can assume that $b_j = 0$ and $\|w_j\| = 1$. Note that these can be achieved easily by removing bias parameters and adding a L2-norm layer respectively. So, the loss function will be transformed to

$$L_i = -\log\left(\frac{\exp(\|x_i\|\cos\left(\theta_{y_i,i}\right))}{\sum_j \exp\left(\| x_j\|\cos(\theta_{j,i})\right)}\right)$$

Simialr to L-softmax loss, we can design the A-softmax loss function as

$$L_i = -\log\left(\frac{\exp\left(\|x_i\|\phi(\theta_{y_i,i})\right)}{\exp\left(\|x_i\|\phi(\theta_{y_i,i})\right) + \sum_{j\neq y_i}\exp\left(\| x_j \|\cos(\theta_{j,i})\right)}\right)$$

where $\phi(\theta) = (-1)^k\cos(m\theta) - 2k$,$\theta \in [\frac{k\pi}{m}, \frac{(k+1)\pi}{m}]$ and $k \in [0, m-1]$. m is a hyperparameter which control the size of angular margin. It can be proved that with larger m, the larger angular margin A-softmax loss constrains. But with too large m, the learning task will be too difficult. We only need a large m which satisfies the maximal intra-class angular distance to be smaller than the minimal inter-class angular distance. For binary classification problem, m should be larger than $2+\sqrt{3}$ and for multi-class classification, m should be larger than 3. Experiments show that $m = 4$ is enough.

CosFace Loss (AM-softmax Loss)

As is shown in the past, the softmax loss function have the form of

$$L_i = -\log\left(\frac{\exp(\|w_{y_i}\|\|x_i\|\cos(\theta_{y_i}) + b_{y_i})}{\sum_j \exp(\|w_j\|\|x_i\|\cos(\theta_j) + b_j)}\right)$$

In cosface loss, some restrictions will be put on, including $b_j = 0$, $\|w_j\| = 1$ and $\|x_i\| = s$. So, the loss has the form

$$L_i = -\log\frac{\exp(s\cos(\theta_{y_i,i}))}{\sum_j \exp(s\cos(\theta_{j,i}))}$$

For this loss function, in a binary classification problem, an instance will be classified to class 1 rather than class 2 if $\cos(\theta_1) > \cos(\theta_2)$. To have a classifier with a large margin, cosface loss requires $\cos(\theta_1) - m > \cos(\theta_2)$ or $\cos(\theta_2) - m > \cos(\theta_1)$ instead. m is a hyperparameter which control the magnitude of cosine margin. The cosface loss function has the formula

$$L_i = -\log\frac{\exp\left(s(\cos(\theta_{y_i,i}) - m)\right)}{\exp\left(s(\cos(\theta_{y_i,i}) - m)\right) + \sum_{j\neq y_i}\exp(s\cos(\theta_{j,i}))}$$

Arcface Loss

Arcface loss also makes some changes on the loss function

$$L_i = -\log\frac{\exp(s\cos(\theta_{y_i,i}))}{\sum_j \exp(s\cos(\theta_{j,i}))}$$

Unlike cosface loss which requires a large magnitude on cosine margin, arcface loss directly increase the magnitude of angle margin. So, the loss has the form

$$L_i = -\log\frac{\exp\left(s\cos(\theta_{y_i,i} + m)\right)}{\exp\left(s\cos(\theta_{y_i,i} + m)\right) + \sum_{j\neq y_i}\exp(s\cos(\theta_{j,i}))}$$

The loss function is called arcface loss is because after getting the dot product of normalized w and normalized x, we need to to a arccos operation first.

Summary

The decision boudary of classification loss function is listed in the following table (assume $b_j = 0$).

Loss Function	Decision Boundary
Softmax Loss	$\|w_1\|\cos(\theta_1) > \|w_2\|\cos(\theta_2)$
L-Softmax Loss	$\|w_1\|\cos(m\theta_1) > \|w_2\|\cos(\theta_2)$
A-Softmax Loss	$\cos(m\theta_1) > \cos(\theta_2)$
Cosface Loss	$\cos(\theta_1) - m > \cos(\theta_2)$
Arcface Loss	$\cos(\theta_1 + m) > \cos(\theta_2)$

Instinctly, we can merge several loss function together and create a boudary like $\cos(m_1\theta_1+m_2)-m_3 > \cos(\theta_2)$.

Metric Learning Loss Function

Classification loss only gives a separable hyperplane , so the location of different IDs may be very near to each other. We have seen some refinements to make the margin larger. Another way is directly learning a big margin. For classification problem, the input is an image and the output is the ID. For metric learning, the inputs are similar images and the output measures the simialrity among the images.

Contrastive Loss

For contrastive loss, we input two images. Two images will go through the same network and get the representation vectors of two images. Suppose the representation vectors are x_1 and x_2 and the relative IDs are y_1 and y_2. If $y_1 = y_2$, then we need to make the distance between x_1 and x_2 lower. And if $y_1 \neq y_2$, then we need to make the distance between x_1 and x_2 bigger. So, the contrastive loss is defined as

$$L(x_i, x_j) = \begin{cases} \|x_i - x_j\|_2^2 & , y_i = y_j \\ \max\big(0, (m - \|x_i - x_j\|_2)^2\big) & , y_i \neq y_j \end{cases}$$

We can see that contrastive loss uses euclidean distance rather cosine distance to measure similarity. The loss function also has the following form

$$L(x_i, x_j) = I(y_i = y_j)\|x_i - x_j\|_2^2 + I(y_i \neq y_j)\max\big(0, (m - \|x_i - x_j\|_2)^2\big)$$

Triplet Loss

Triplet loss considers three images in one iteration. Three images are anchor image, positive image (which has the same ID of anchor image) and negative image (which has the different ID of anchor image). Suppose the representation vectors is x_a, x_p and x_n, then we want three vectors have the property: $\|x_a - x_p\|_2^2 + \alpha < \|x_a - x_n\|_2^2$ for $\forall(x_a, x_p, x_n) \in T$. This means the distance between same ID is at least α smaller than that of different ID. So, large margin will be learned and triplet loss is defined as

$$L(x_a, x_p, x_n) = \big[\|x_a - x_p\|_2^2 + \alpha - \|x_a - x_n\|_2^2\big]_+$$

Another problem of triplet loss is how to choose the triplets. Once given the anchor image x_a, the positive image is easy to select but the selection of the negative image needs some tricks. Negative images can be divided into three groups:

- Easy triplets: $d(x_a, x_p) + \alpha < d(x_a, x_n)$. This case need not optimize.
- Hard triplets: $d(x_a, x_p) > d(x_a, x_n)$. This case will lead to big loss.
- Semi-hard triplets: $d(x_a, x_p) < d(x_a, x_n) < d(x_a, x_p) + \alpha$. This case has small loss.

Training with hard triplets is theoretically the best choice because it has the biggest loss. But in practice semi-hard triplets has the best performance because hard triplets are sometimes hard to optimize. Triplet loss also has some shortcomings. The biggest of these is that there are too many triplets in the dataset so the loss need many iterations to converge.

N-pair Loss

Instead of only choosing one negative vector, we can choose N-1 negative vectors to accelerate the training process. Suppose x is the anchor vector, x_+ is the positive vector and the negative vectors are $x_1, x_2, ..., x_{N-1}$. Use this (N+1)-tuple can build a cross entropy loss via cosine distance

$$l = -\log \frac{\exp(x \cdot x_+)}{\exp(x \cdot x_+) + \sum_{i=1}^{N-1} \exp(x \cdot x_i)} = \log \left(1 + \sum_{i=1}^{N-1} \exp(x \cdot x_i - x \cdot x_+) \right)$$

But directly using (N+1)-tuple in SGD optimization will cause some problems. Suppose the batch size is M, then we altogether need $M(N+1)$ examples to do train the embedding function. So the number of examples we need is quadratic to M and N, then it's very time-consuming and impratical. N-pair loss change the way of getting negative samples, so we only need $2N$ examples at a time.

Let $\{(x_1, x_1^+), (x_2, x_2^+), ..., (x_N, x_N^+)\}$ be N pairs of examples under N different IDs. Then for x_j, x_j^+ is the positive vector and $x_i^+, i \neq j$ are negative vectors. We can create N (N+1)-tuple pairs and the average loss is

$$l(\{x_i, x_i^+\}_{i=1}^N) = \frac{1}{N} \sum_{i=1}^{N} \log \left(1 + \sum_{j \neq i} \exp(x_i \cdot x_i^+ - x_i \cdot x_j^+) \right)$$

Summary

Loss function is one of the most important settings in the training process of ReID and Facial Recognition. There are diverse loss functions but what loss functions do is the same: **decrease inter-class distance and increase intra-class distance**.