PAPER

Adversarial Metric Learning with Naive Similarity Discriminator

Yi-ze LE[†,††], ***Student Member***, **Yong FENG**[†,††a)], **Da-jiang LIU**[†,††], ***and*** **Bao-hua QIANG**[†††,††††], ***Nonmembers***

SUMMARY Metric learning aims to generate similarity-preserved low dimensional feature vectors from input images. Most existing supervised deep metric learning methods usually define a carefully-designed loss function to make a constraint on relative position between samples in projected lower dimensional space. In this paper, we propose a novel architecture called Naive Similarity Discriminator (NSD) to learn the distribution of easy samples and predict their probability of being similar. Our purpose lies on encouraging generator network to generate vectors in fitting positions whose similarity can be distinguished by our discriminator. Adequate comparison experiments was performed to demonstrate the ability of our proposed model on retrieval and clustering tasks, with precision within specific radius, normalized mutual information and F_1 score as evaluation metrics.

key words: *metric learning, adversarial learning, naive similarity discriminator*

1. Introduction

Metric learning is a fundamental problem in a variety of computer visual tasks, including image retrieval [1], [2], person re-identification [3]–[5], face recognition [6], [7] and image classification [8]–[10]. The goal of metric learning is to learn a distance metric that reflects the similarity of given data pair, i.e. images with a common label should be adjacent to each other and vice versa.

Most existing deep learning approaches deal the metric learning task in two step. The first is designing loss function to constraint the relative position between positive pairs and negative pairs, the second pays attention on mining beneficial samples for faster convergence and stronger performance.

Loss functions used in metric learning can be generally formulated as

$$L = h(\|x_a - x_p\|_2^2, \|x_a - x_n\|_2^2, \alpha) \quad (1)$$

where $\|\cdot\|_2$ represents l2-normalization. x_a, x_p and x_n indicate anchor, positive and negative sample respectively. Anchor sample shares a common label with positive sample while have a different label to negative sample. The parameter α determines the dispersion between anchor-positive pairs and anchor-negative pairs. Specifically designed function h aims to shorten the distance between x_a and x_p meanwhile enlarge distance between x_a and x_n, with the constraint of α. Classical methods build pairwise [11] or triplet samples [6], [12] in loss functions to control their relations.

Sample mining intends to search for samples that profit training most to achieve a faster convergence and higher performance. Generally, we are more likely to select samples that most against our objective, which are called “hard samples”. For instance, close pair with different labels and distant pair with same label are harder to optimized. Relatively, those suitable points that rarely used in optimization are called “easy samples”. Existing approaches with sample mining seek for information with more and harder classes [1] or harder samples [2] to benefit the training. Many recent works [13], [14] focus on applying weights on samples which aims to reflect their importance on optimization.

The spirit of adversarial learning comes from the Generative Adversarial Network (GAN) [15]. Adversarial learning generally composed of two components called generator and discriminator. Generator aims to generate features under specific requirements, while discriminator tries to beat against it to encourage generator to achieve a better performance. There also exists GAN based method [16], [17] which generate hard samples to augment dataset.

Existing works illustrate relations among samples with well-designed loss functions, but they rarely use the distribution of given samples and may have restriction on describing their relations. In this work, we replace these conventional loss function with a neural network called Naive Similarity Discriminator (NSD) and the standard binary cross entropy loss. Meanwhile, we exploit adversarial learning to encourage our generator to provide similarity-preserved samples. The main contribution of this work can be described as following:

- Learning the distribution of existing easy samples using a carefully-designed neural network as the discriminator, which exploit the learning ability of this deep network to illustrate the state of similarity-preserved samples.

Manuscript received October 13, 2019.
Manuscript revised January 20, 2020.
Manuscript publicized March 10, 2020.

[†]The authors are with the College of Computer Science, Chongqing University, Chongqing 400030, China.
[††]The authors are with the Key Laboratory of Dependable Service Computing in Cyber Physical Society, Ministry of Education, Chongqing University, Chongqing 400030, China.
[†††]The author is with the Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin 541004, China.
[††††]The author is with the Guangxi Key Laboratory of Optoelectronic Information Processing, Guilin University of Electronic Technology, Guilin 541004, China.
a) E-mail: fengyong@cqu.edu.cn

DOI: 10.1587/transinf.2019EDP7278

- Utilizing adversarial learning by mining hard samples and feed them to discriminators. Huge loss will be produced and the generator are forced to push hard samples to easier positions, which exhibiting a novel learning strategy for metric learning.
- Alternatively applying three individual stages for continuously encourage the generator network to achieve a better performance.

2. Related Works

Existing metric learning methods mainly focus on loss functions formulations or sample mining strategies. The well-designed loss functions describe the relation among samples. These loss functions build the optimization target to meet the goal of enlarging distance between samples in different class meanwhile shorten the distance between samples from the same class. Sample mining approaches can be generally categorized into hard mining methods, hard sample synthesis methods and weighted methods.

2.1 Loss Function Formulations

Loss function plays a critical role in metric learning task. Many existing deep metric learning methods pay more attention on building a proper loss function to illustrate relation between pairwise samples or among triplet samples.

Contrastive Loss [11] is trained on pairwise samples with a margin parameter α, which give penalty to those negative pairs with distances smaller than α:

$$L = \frac{1}{n}\sum_{i\neq j}^{n} s_{ij}\|x_i - x_j\|_2^2 + (1 - s_{ij})[\alpha - \|x_i - x_j\|_2^2]_+ \quad (2)$$

where $s_{ij} = 1$ if x_i share a common label with sample x_j otherwise $s_{ij} = 0$. $[\cdot]_+$ is the hinge function.

Semi-hard Triplet Loss [6] further utilizes the relation among samples by constructing triplets and controls their discrepancy. Penalty is delivered to those triplet with discrepancy smaller than α:

$$L = \frac{1}{n}\sum_{(a,p,n)\in T} [\|x_a - x_p\|_2^2 - \|x_a - x_n\|_2^2 + \alpha]_+ \quad (3)$$

where T comprises all triplets of a given batch of data.

Instead of comparing samples in Euclidean space, Angular Loss [12] measure the distances between samples with angle relationships with a angle constraint $\tan\alpha$, which introduce scale invariance and capture additional local structure:

$$L = \frac{1}{n}\sum_{(a,p,c,n)\in T} [\|x_a - x_p\|_2^2 - 4\tan^2\alpha\|x_n - x_c\|_2^2]_+ \quad (4)$$

where x_c is the middle point of x_a and x_p.

2.2 Sample Mining Strategies

Appropriate sample mining can bring great benefits in metric learning tasks. Various of methods have been contributed on hard sample mining, weighted methods and hard sample sythesis methods.

2.2.1 Hard Sample Mining

N-pair Loss [1] considers more classes to be involved in training. They firstly choose C classes and pass one or two examples of theses class to extract embedding vectors. Then select one class and greedily add new class that mostly violates triplet constraint until meet the required number of classes. Finally they draw two examples from chosen classes to finish the N-pair construction.

Lifted Structured Loss [2] emphasizes the hardest negative for each sample from all pair wise edges within the batch and push negative data points further than the margin. They further optimize the smooth upper bound loss function to avoid converging to local optimum caused by mining single hardest negative sample.

These methods try to dig data that have higher demand to be optimized, but they may be restricted by the lack of hard samples.

2.2.2 Hard Sample Synthesis

To address the problem of inadequacy hard samples, sample synthesis methods have been made great contributions. Hardness-Aware Deep Metric Learning [17] synthesizes new hard sample by exploiting existing samples on the edge of manifold, and generates a closer sample on manifold with adaptive hardness controlled by a label-and-hardness-preserving generator. Deep Adversarial Metric Learning [16] utilizes a generator with adversarial loss to synthesize hard samples for training. With a different orientation to these works, our ambition is to force generator to provide suitable embedding vectors using adversarial learning.

2.2.3 Weighted Methods

Many recent approaches focus on determining the importance of samples in training procedure, which can be categorized as weighted methods. These methods can be generally formulated as

$$\begin{aligned} L &= \frac{1}{n}\sum_{(a,p,n)\in T} w_{ap}\|x_a - x_p\|_2^2 - w_{an}\|x_a - x_n\|_2^2 \\ &w.r.t \quad \|x_a - x_p\|_2^2 - \|x_a - x_n\|_2^2 + \alpha > 0 \end{aligned} \quad (5)$$

where w_* indicates the weight of a given pair, which is calculated by the relations of distances from a set of sample pairs. Ranked List Loss [13] chooses samples violate constraint and set weight on each anchor-negative pair through their distances. Multi-Similarity Loss [14] collects informative pairs and weight these pairs through their own and relative similarities.

In this work, we introduce a novel training strategy for metric learning. Instead of constructing a handcraft loss

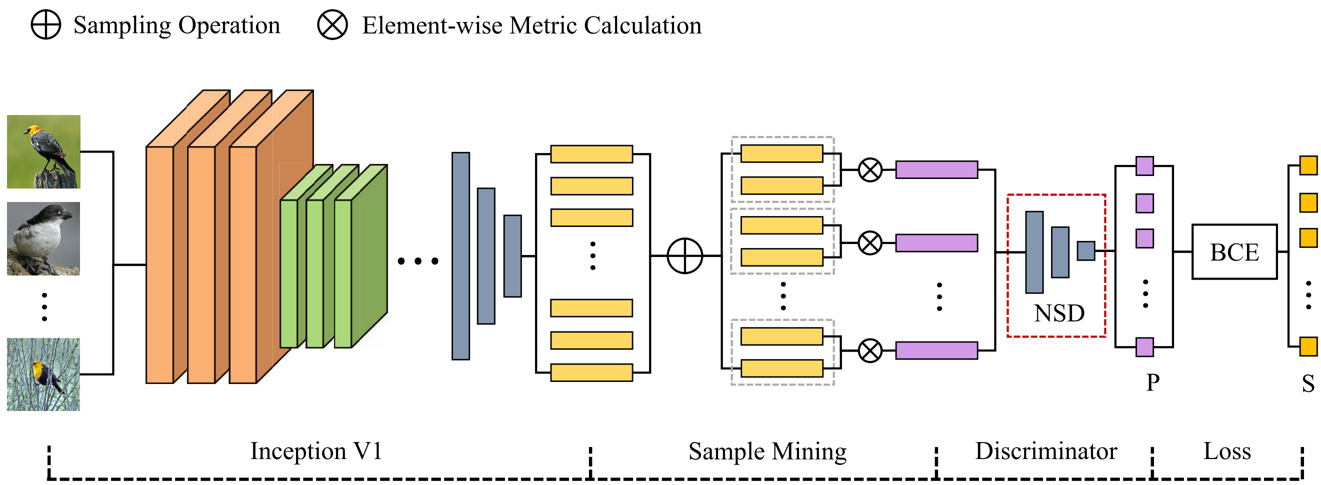


Fig. 1 An end-to-end architecture of entire model. Our model takes a batch of images as input and extract features through Inception, followed by sample mining procedure to pick suitable pairwise samples and apply element-wise distance metric for Naive Similarity Discriminator (NSD), which predict the probability of being similar of given pairs. A standard binary cross entropy (BCE) loss are applied to train our model.

function to constraint relative position between samples, we propose a discriminator that learns the distribution of easy samples. Our discriminator takes a pair of feature representations as input, which extracted from two individual images through the generator network, and outputs the probability of being similar for the input pair. After training with easy samples, the discriminator is supposed to accurately classify easy samples, while contribute large loss when feeding hard samples. These loss will feedback to the generator network and enhance its performance on metric learning task. It is worth noticing that our discriminator should not be too robust for hard samples to guarantee the sufficient loss for improving generator network.

3. Proposed Method

In this section, we present our model with a feature generator as metric function, and a similarity discriminator learnt with a prior element-wise distance metric and labeled similarity. An adversarial learning strategy is applied to encourage generator to provide samples that retain the similarity relations.

3.1 Problem Formulation

Let $X = \{x_i\}_{i=1}^N$ be the input images and $Y = \{y_i\}_{i=1}^N$ be the corresponding labels where $y_i \in \{1 \dots C\}$. We define $S = \{s_{ij}\}_{i,j=1}^N$ as a similarity matrix between input instances, with $s_{ij} = 1$ for x_i and x_j that from the same class and $s_{ij} = 0$ otherwise. Our ambition is to learn a metric function f that properly measures the similarity of input pair

$$D(x_i, x_j) = f(\theta; x_i, x_j), \tag{6}$$

where D represents distance between input pairwise samples under learnt metric and θ is the parameters of function f.

3.2 Model Architecture

Our architecture is shown in Fig. 1. The model accepts a batch of images as input and go through the following pipeline:

1. A typical convolutional neural network as generator to extract meaningful image representations.
2. A sample mining process that selecting similar and dissimilar pairs with different strategy for three individual stages.
3. A simple but carefully-designed neural network as discriminator to predict probability of being similar of given pairs.

3.2.1 Generator

We exploit Inception V1 [18] which pretrained on ImageNet [19] as our backbone network for fairly comparison to existing deep metric learning approaches. An additional fully connected layer was added to the Inception V1 network to project feature embeddings to proper size. We apply l2-normalization on each output vector z_i with m-dimension as following:

$$\hat{z}_i = \frac{z_i}{\sqrt{\sum_{j=1}^{m} (z_i^{(j)})^2}} \tag{7}$$

where $z_i^{(j)}$ indicates the j-th element of vector z_i. Thus all the output vectors become unit vectors with a specific direction in the learnt metric space.

Our purpose is to train a generator as metric function f that well preserve similarity between input samples, i.e. instances with the same class label are closer than those in different classes.

3.2.2 Discriminator

Our similarity discriminator is constructed by a simple multilayer perceptron which is randomly initialized. We firstly select several similar and dissimilar pairs $P = \{(\hat{z}_i, \hat{z}_j)\}_{(i,j)\in I}$ from generated features, where I comprises indices of selected embeddings. After that, we apply a prior element-wise distance metric for each pair to construct a m-dimension vector v_{ij} as the input of the discriminator to obtain the predicted p_{ij}, which indicates the probability of being similar. Formulaically, for k-th element of $\hat{z}_i$, $\hat{z}_j$ and v_{ij}, we have

$$v_{ij}^{(k)} = d_e(\hat{z}_i^{(k)}, \hat{z}_j^{(k)}) = (\hat{z}_i^{(k)} - \hat{z}_j^{(k)})^2 \tag{8}$$

$$p_{ij} = g(v_{ij}) \tag{9}$$

where d_e is an euclidean-like prior element-wise distance metric and g represents the discriminator which predict the probability of $\hat{z}_i$ being similar to $\hat{z}_j$. We adopt typical binary cross-entropy loss to train our generator and discriminators as

$$L(p_{ij}, s_{ij}) = -(s_{ij} \log p_{ij} + (1 - s_{ij}) \log(1 - p_{ij})) \tag{10}$$

3.3 Adversarial Metric Learning

As mentioned above, it is clear that hard samples have higher demand to be optimized as easy samples have already in a fitting position. In turn, if the generator always provide easy samples, the model may be in a good condition for generating similarity-preserved samples, which meet the goal of metric learning.

Our motivation of designing a similarity discriminator is to encourage generator to provide easy pairs whose similarity can be easily distinguished by any simple classifier. To this end, inspired by the spirit of adversarial learning, we train our generator and discriminator by selecting samples in disparate hardness.

Concretely, for training discriminators, we choose easiest samples, i.e. the closest points in the same class and the furthest points in different classes. We call this discriminator trained with easy samples Naive Similarity Discriminator (NSD).

NSD is trained to learn the distribution of easy pairs and can perfectly distinguish the similarity of easy pairs. But NSD has a low accuracy on determining hard pairs due to the lack of knowledge of hard samples. Thus, we train generator with hard samples and fixed NSD in the purpose of encouraging our model to produce easier samples by optimizing our similarity classification loss. Those hard pairs that can not be distinguished by NSD will be misclassified and deliver large loss, thus the generator are intend to push them to a easier place. It is worth noticing that our model is an end-to-end architecture and both training procedure are employed under the guidance of standard binary cross-entropy loss described in Eq. (10).

Specifically, we implement three stages in training procedure for different components of our model:

- **Warm-up stage** to train the randomly initialized additional layer and the discriminator with easy samples.
- **Generator stage** to train the generator network with hard samples.
- **Discriminator stage** to train the discriminator with easy samples.

It is worth noticing that the only difference between warm-up stage and discriminator stage is the optimization on the additional layer. The goal of warm-up stage is to optimize all the randomly initialized parameter at the beginning of training procedure.

Concretely, we firstly adopt warm-up stage to train all the randomly initialized parameters. Then we alternately employ generator stage and discriminator stage to train these two components in turn.

3.4 Naive Similarity Discriminator

The effect of NSD is to learn the distribution information of easy pairs which can be exploited to make a classification on similarity of these samples. The ideal state for metric learning is all pairs constructed from dataset ought to be easy pairs. Our destination is to leave NSD to learn this ideal state on warm-up and discriminators stages, and make judgement and punishment on generator stage, which encourage all samples in dataset to get closer to this ideal state.

Considering the role of NSD played in different stage, it is supposed to design a discriminator that obeying the following rules:

- Converged within a few epochs for distinguish easy pairs with nearly perfect accuracy, which enabling NSD to learn current distribution of easy samples on every alternation of stages.
- Relative weak in predicting hard samples to guarantee adequate misclassified samples and sufficient loss for training generator.

To achieve these goals, two key requirements must be met when designing the architecture of NSD.

1. Must not too simple to unsuccessfully reach a satisfied performance on easy pairs judgement.
2. Not too complicated to guarantee the large number of misclassified samples and enough loss is provided, with a relative weak ability on determining hard pairs.

To make a balance between these two requirements, an unusual composition of multilayer perceptron was employed as NSD. Concretely, we build a three fully connected layers network with batch normalization [20]. But the difference to general neural network is that our model do not have any activation function in hidden layers.

Theoretically, a three-layer structure with batch normalization has a higher speed on convergence than a single-layer structure, though they can be transformed to each

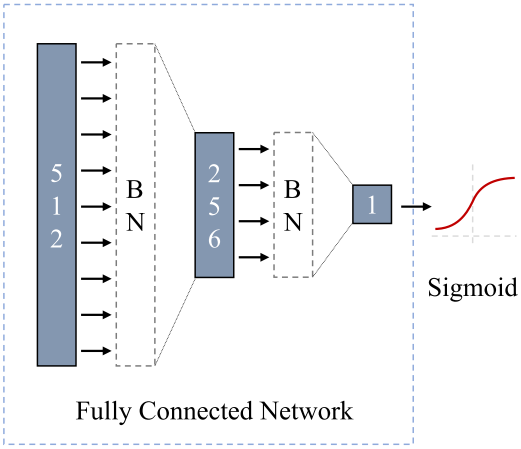


Fig. 2 Architecture of Naive Similarity Discriminator, which is composed of three fully connected layers with 512, 256 and 1 neurons respectively, which are shown as the gray square marked with number of neurons. The first two layers are followed a by batch normalization (BN) layer respectively, while a sigmoid function is applied after the output layer to come up with a probability.

other through matrix decomposition and matrix multiplication technology. The main principle of speeding up in convergence is that batch normalization guarantees the stability on distribution for each hidden layer inside discriminators.

Our concern of removing activation on hidden layers is to decline the ability of discriminator on determining similarity between hard pair for providing adequate misclassified samples and sufficient loss to reinforce capability of producing similarity preserved feature of generator. Complete architecture of NSD is shown in Fig. 2.

Noticing that a critical destination of performing a NSD is to continuously reinforce generator through misclassified hard samples, rather than accurately distinguish those pairs. The ideal state on generator stage is that the input hard samples yield a large classification loss and rarely distinguish similarities. Concretely, if the accuracy on determining hard samples stays around 0.1, the model will continuously improve performance on all experimental metrics. We observed from experiments that once the NSD fail to provide adequate misclassified samples and sufficient loss on discriminator stage, especially when the accuracy reach the uncertain stage, i.e. around 0.5, the improvement of performance would stand still or even drop down.

3.5 Implementation Details

We utilized PyTorch framework with 32GB memory and NVIDIA 1080Ti through the experiment. We resized input images into 256×256 followed by standard randomly resized crop to 224×224 and horizontal flipping for data augmentation. Our backbone Inception was pretrained on ImageNet ILSVRC [19] dataset with a randomly initialized additional fully connected layer, which was optimized with 10 times learning rate compared to other layers.

For NSD, we built a network with 512, 256 and 1 neurons for three layers respectively and the learning rate was set to 0.1. We applied standard stochastic gradient descent (SGD) with momentum set to 0.9. We performed learning rate decay as 0.8 on NSD and Inception with different frequency. Concretely for NSD, we applied learning rate decay on each stage alternation, while for Inception, we decay the learning rate five times in each generator stage. We optimized warm-up stage and discriminator stage for 3 epochs, followed by 70 epochs on generator stage. We fixed the embedding size to 512 as a fair comparison to other approaches. We set 20000 iterations on training with batch size as 120.

We apply the same sampling strategy for each dataset. Concretely for each batch, we randomly select 40 classes from training set, and randomly choose 3 images for each class to comprise the batch. We calculate pairwise distance within a batch. Formulaically for class c_i, we select the closest pair within c_i, and find the furthest pair consists of instances from c_i and other classes $\{c_j\}_{j \neq i}$ as easy samples. Oppositely, hard samples are selected from furthest pair within the class and the closest pair to other classes.

4. Experiments

We implemented our experiment on two well-known datasets for metric learning on both retrieval and clustering task to verify the impact of our adversarial learning with NSD and make a comparison to existing approaches.

4.1 Datasets and Evaluation

Our experiments was employed on widely-used CUB-200-2011 [21] dataset and Cars196 [22] dataset. We followed the dataset configuration of existing approaches [1], [2], [6], [12], [13] for fairly comparison as following:

- CUB-200-2011 [21] dataset contains 11,788 bird images in 200 categories. We exploited the first 100 categories with 5,684 images as training set and the rest for testing.
- Cars196 [22] dataset includes 16,185 car images with 196 categories. We used the first 98 categories with 8,054 images for training and save the rest for building test set.

The purpose of dividing dataset based on categories is to evaluate the robustness to unseen classes.

We evaluated proposed method and existing methods on both retrieval and clustering task.

For retrieval task, we calculated percentage of retrieved samples with the same label to the query image in K nearest neighbors, where $K \in \{1, 2, 4, 8\}$, marked as R@K.

For clustering task, we employed standard K-means algorithm in test set, which evaluated with normalized mutual information (NMI) and F_1 score. NMI consists of the ratio of mutual information divided by the average entropy of clusters and the average entropy of labels. F_1 score computes the harmonic mean of precision and recall on determining whether sample attribution to a specific cluster.

Table 1 Experimental results on CUB-200-2011 dataset

Method	R@1	R@2	R@4	R@8	NMI	F_1
Semi-hard	0.3621	0.4693	0.5930	0.7108	0.4880	0.2807
N-pair	0.4147	0.5418	0.6634	0.7714	0.5509	0.3475
Lifted	0.4932	0.6120	0.7265	0.8256	0.5691	0.3902
Angular(N-pair)	0.4799	0.6041	0.7268	0.8210	0.5827	0.4136
Ranked List	0.5712	0.6877	0.7873	0.8661	0.6302	0.4647
NSD(Ours)	0.5833	0.6987	0.7984	0.8754	0.6559	0.5162

Table 2 Experimental results on Cars196 dataset

Method	R@1	R@2	R@4	R@8	NMI	F_1
Semi-hard	0.2399	0.3507	0.4859	0.6205	0.4297	0.2226
N-pair	0.3924	0.5288	0.6498	0.7606	0.5159	0.3086
Lifted	0.5630	0.6606	0.7470	0.8249	0.4585	0.2654
Angular(N-pair)	0.6729	0.7368	0.8542	0.8917	0.5773	0.4306
Ranked List	0.7544	0.8385	0.8975	0.9382	0.6174	0.4740
NSD(Ours)	0.7303	0.8320	0.8845	0.9246	0.5829	0.4434

4.2 Comparisons

We compared our adversarial metric learning with NSD to some famous deep metric learning approaches. We implemented the famous Semi-hard triplet loss [6] and Angular loss [12], the classical sample mining method N-pair loss [1], Lifted Structured loss [2], and the recent weighted method Ranked List loss [13]. We followed the settings mentioned in these original papers through the comparison experiments. Noticing that we applied Angular loss with the N-pair sampling strategy, as they mentioned in their works.

Table 1 and Table 2 shows the result of NSD compared to other baseline methods in two benchmark dataset respectively. Compared to methods with standard handcraft-designed loss functions [1], [2], [6], [12], we outperformed them on both datasets with a large margin on both retrieval and clustering task. Rather than set a specific margin and optimize objective with constraint, we leave our discriminator to decide what kind of pairs should be optimized, which break the restriction of these loss functions. For the state-of-the-art Ranked List loss, we reach a higher performance on the smaller CUB-200-2011 dataset, but failed to achieve their performance on the larger Cars196 with a small margin. We analyse that with the increase of samples in each class, we fail to collect sufficient information within a fix sampling number, while setting weight for each individual sample provided more information.

4.3 Impact of Naive Similarity Discriminator

The architecture of our NSD has a critical impact on the performance in our model. As mentioned above, the ideal model should converge fast for easy samples and provide sufficient loss for hard samples. We take various of architecture into consideration including

- A three fully connected layers network with only batch normalization on hidden layers, which we finally applied on our model.
- Standard three-layers fully connected neural network with batch normalization and ReLU function on hidden layers, marked as NSD-A.
- Two layers architecture variant of original NSD, with 256 units in hidden layer, marked as NSD-T.
- A single node layer composited with sigmoid function, i.e. logistic regression, marked as NSD-L.
- Three layer neural network with no activation function and batch normalization on hidden layers, which is theoretically equal to NSD-L if sufficiently trained, marked as NSD-B.

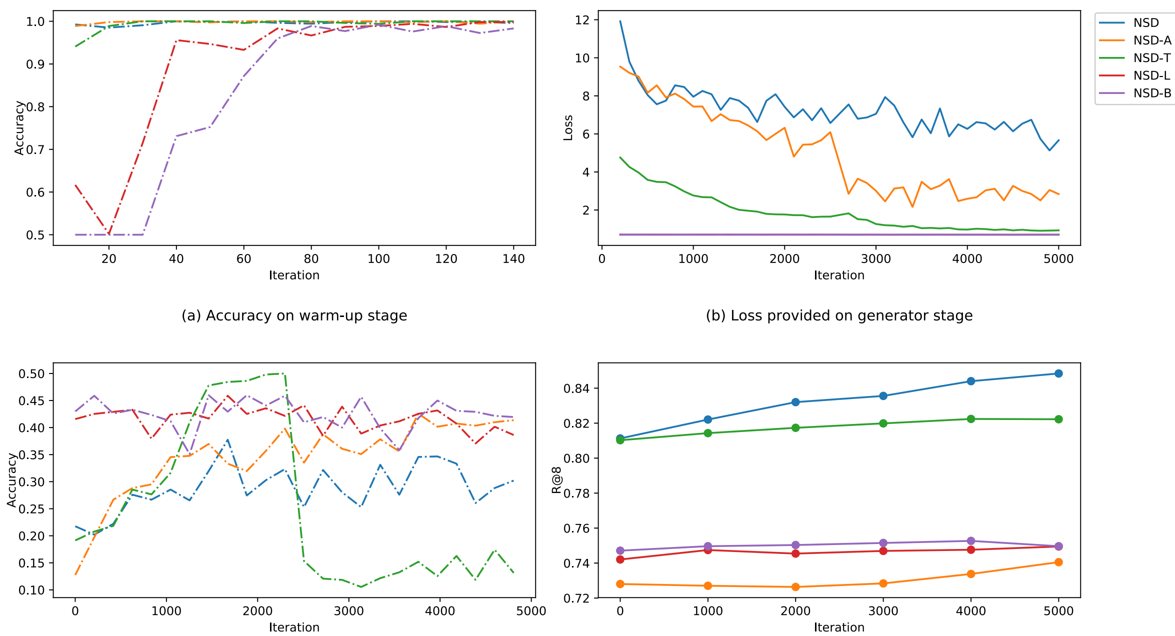
We evaluated the effect of these variants with four metrics on CUB-200-2011 dataset, including

1. Convergence speed and accuracy with easy samples on warm-up stage.
2. Loss provided on generator stage.
3. Accuracy with hard samples on generator stage, which reflects the percentage of correctly classified samples within a batch.
4. Percentage of samples with the same label in 8 nearest neighbors, i.e. R@8, during the first 5000 iteration.

Figure 3 shows the experiment result of four variants and original NSD on three metrics mentioned above, respectively.

Separately discussing, for networks without batch normalization, i.e. NSD-L and NSD-B, failed to achieve a perfect prediction at the very beginning while other variants early gained the ability of judging easy pairs. As a high generator loss with a low accuracy on distinguishing hard pairs leads to continuously reinforce the performance, we can see from Fig. 3 (b) and (c) that these two variants with nearly zero loss and an nearly uncertain judgement, i.e. with accuracy as 0.5, and failed to improve performance on precision metric.

Specifically for NSD-A, though it provided adequate loss, but the accuracy on predicting hard pairs early reach an uncertain level, which reflected that it may determine the input pair in a random state and rarely feed beneficial information to the generator, resulting the bad performance.



(a) Accuracy on warm-up stage

(b) Loss provided on generator stage

(c) Accuracy on generator stage

(d) Trend on R@8

Fig. 3 Result of comparison experiments on three variants of NSD on (a) accuracy with easy pairs on warm-up stage, (b) loss provided on generator stage, (c) accuracy with hard pairs on generator stage and (d) percentage of samples with same label in 8 nearest neighbors. Noticing that in (b), NSD-L and NSD-B are overlapping around a rare small value.

It is worth discussing a special phenomenon performed by the two layer structured variant NSD-T, which reach a good performance but slightly weaker than original NSD. NSD-T provided relative lower loss on generator stage with an unstable accuracy on determining hard pairs. We can see that the accuracy went up at the first 2000 iterations and suddenly dropped down around 2500 iteration. This sudden change resulted from the alternation of stage, and the fresh randomly initialized NSD assisted this model out of the puzzle zone of distinguishing hard pairs, which guarantee the number of misclassified samples and leads to the stable performance.

From the analysis above, we can conclude that high loss with low accuracy on generator stage can frequently boosting the ability of generating easy pairs of generator, which leads to a better performance on adversarial metric learning with NSD.

5. Conclusion

In this paper, we have proposed a novel training strategy on metric learning task by building a carefully-designed neural network called Naive Similarity Discriminator, with the spirit of adversarial learning. The main principle of out model is to encourage generator network to generate easier samples that can be distinguished by simple classifier by exploiting the amount of misclassified samples with large loss provided by NSD. Experimental results on CUB-200-2011 and Cars196 indicate strong performance of our NSD model with adversarial learning. While the essentially principle on influence of architecture of NSD and the strategy of sampling more information are needed further study on our future works.

Acknowledgments

Supported by National Key R&D Program of China (No. 2017YFB1402400), National Nature Science Foundation of China (No. 61762025), Guangxi Key Laboratory of Trusted Software (No. kx201701), Guangxi Key Laboratory of Optoelectroric Information Processing (No. GD18202), and Frontier and Application Foundation Research Program of CQ CSTC (No. cstc2017jcyjAX0340).

References

[1] K. Sohn, "Improved deep metric learning with multi-class n-pair loss objective," Advances in Neural Information Processing Systems, pp.1857–1865, 2016.
[2] H.O. Song, Y. Xiang, S. Jegelka, and S. Savarese, "Deep metric learning via lifted structured feature embedding," Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp.4004–4012, 2016.
[3] J. Zhou, P. Yu, W. Tang, and Y. Wu, "Efficient online local metric adaptation via negative samples for person re-identification," Proc. IEEE International Conference on Computer Vision, pp.2420–2428, 2017.
[4] H.-X. Yu, A. Wu, and W.-S. Zheng, "Cross-view asymmetric metric learning for unsupervised person re-identification," Proc. IEEE International Conference on Computer Vision, pp.994–1002, 2017.
[5] Z. Liu, D. Wang, and H. Lu, "Stepwise metric promotion for unsupervised video person re-identification," Proc. IEEE International Conference on Computer Vision, pp.2429–2438, 2017.

[6] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," Proc. IEEE conference on computer vision and pattern recognition, pp.815–823, 2015.
[7] W. Deng, J. Hu, Z. Wu, and J. Guo, "From one to many: Pose-aware metric learning for single-sample face recognition," Pattern Recognition, vol.77, pp.426–437, 2018.
[8] T. Mensink, J. Verbeek, F. Perronnin, and G. Csurka, "Metric learning for large scale image classification: Generalizing to new classes at near-zero cost," European Conference on Computer Vision, vol.7573, pp.488–501, Springer, 2012.
[9] G. Cheng, C. Yang, X. Yao, L. Guo, and J. Han, "When deep learning meets metric learning: Remote sensing image scene classification via learning discriminative cnns," IEEE Trans. Geosci. Remote Sens., vol.56, no.5, pp.2811–2821, 2018.
[10] Z. Wang, Y. Li, R. Hong, and X. Tian, "Eigenvector-based distance metric learning for image classification and retrieval," ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), vol.15, no.3, p.84, 2019.
[11] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), pp.1735–1742, IEEE, 2006.
[12] J. Wang, F. Zhou, S. Wen, X. Liu, and Y. Lin, "Deep metric learning with angular loss," Proc. IEEE International Conference on Computer Vision, pp.2593–2601, 2017.
[13] X. Wang, Y. Hua, E. Kodirov, G. Hu, R. Garnier, and N.M. Robertson, "Ranked list loss for deep metric learning," Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp.5207–5216, 2019.
[14] X. Wang, X. Han, W. Huang, D. Dong, and M.R. Scott, "Multi-similarity loss with general pair weighting for deep metric learning," Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp.5022–5030, 2019.
[15] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," Advances in Neural Information Processing Systems, pp.2672–2680, 2014.
[16] Y. Duan, W. Zheng, X. Lin, J. Lu, and J. Zhou, "Deep adversarial metric learning," Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp.2780–2789, 2018.
[17] W. Zheng, Z. Chen, J. Lu, and J. Zhou, "Hardness-aware deep metric learning," Proc. IEEE Conference on Computer Vision and Pattern Recognition, pp.72–81, 2019.
[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," Proc. IEEE conference on computer vision and pattern recognition, pp.1–9, 2015.
[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," 2009 IEEE conference on computer vision and pattern recognition, pp.248–255, Ieee, 2009.
[20] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," International Conference on Machine Learning, pp.448–456, 2015.
[21] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The caltech-ucsd birds-200-2011 dataset," Tech. Rep. CNS-TR-2011-001, California Institute of Technology, 2011.
[22] J. Krause, M. Stark, J. Deng, and L. Fei-Fei, "3d object representations for fine-grained categorization," 4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13), Sydney, Australia, 2013.

Yi-ze Le is a master at the College of Computer Science, Chongqing University. His research interest is Deep Learning and Big Data Retrieval.

Yong Feng is a Professor at the College of Computer Science, Chongqing University. His research interest covers Big Data Analysis and Data Mining, Artificial Intelligence and Big Data Processing, Deep Learning and Big Data Retrieval. Corresponding author of this paper.

Da-jiang Liu is a Lecturer at the College of Computer Science, Chongqing University. His research interest covers Deep Reinforcement Learning, Software Definition Hardware, and Deep Learning Compilation.

Bao-hua Qiang is a Professor at the Guangxi Cooperative Innovation Center of cloud computing and Big Data, Guilin University of Electronic Technology. His research interest is Big Data Processing and Information Retrieval.