Learning to Optimize on SPD Manifolds
Supplementary Materials

Zhi Gao[1], Yuwei Wu[1]*, Yunde Jia[1], Mehrtash Harandi[2]
[1]Beijing Laboratory of Intelligent Information Technology
School of Computer Science, Beijing Institute of Technology, Beijing, China
[2]Department of Electrical and Computer Systems Eng., Monash University, and Data61, Australia
{gaozhi_2017,wuyuwei,jiayunde}@bit.edu.cn, mehrtash.harandi@monash.edu

1. Training Our Optimizer on Specific Tasks

Recall that optimization problems with SPD constraints can be formulated as minimizing a loss

$$\mathcal{L}(\boldsymbol{M}) \triangleq \frac{1}{n}\sum_{i=1}^{n} l\big(f(\boldsymbol{M},\boldsymbol{x}_i),\boldsymbol{y}_i\big), \tag{1}$$

with respect to $\boldsymbol{M}$, that is $\min_{\boldsymbol{M}\in\mathcal{S}^d_{++}}\mathcal{L}(\boldsymbol{M})$. In Eq. (1), $\boldsymbol{x}_i$ is the i-th training sample, and its corresponding target (*e.g.*, label) is $\boldsymbol{y}_i$. $f(\cdot)$ and $l(\cdot)$ represent the prediction and objective functions, respectively, and $\boldsymbol{M}\in\mathcal{S}^d_{++}$ encapsulates parameters of the optimization problem.

To solve Eq. (1), we parameter the SPD optimizer by a network, and ϕ is the parameter of the network, through which the SPD parameter is updated by

$$\boldsymbol{M}^{(t+1)} = \Gamma_{\boldsymbol{M}^{(t)}}\big(-g_\phi(\nabla^{(t)}_{\boldsymbol{M}},\boldsymbol{S}^{(t-1)})\big), \tag{2}$$

where $\Gamma_{\boldsymbol{M}^{(t)}}(\cdot)$ is the retraction operation, and $g_\phi(\nabla^{(t)}_{\boldsymbol{M}},\boldsymbol{S}^{(t-1)})$ is the update vector on the tangent space. We train the optimizer by minimizing the following meta-objective

$$\begin{aligned}\mathcal{J}(\phi) &= \frac{1}{m}\sum_t^T\sum_j^m \mathcal{L}(\boldsymbol{M}_j^{(t+1)})\\ &= \frac{1}{mn}\sum_t^T\sum_{j,i}^{m,n} l\left(f\left(\boldsymbol{M}_j^{(t+1)},\boldsymbol{x}_i\right),\boldsymbol{y}_i\right)\\ &= \frac{1}{mn}\sum_t^T\sum_{j,i}^{m,n} l\left(f\left(\Gamma_{\boldsymbol{M}_j^{(t)}}\left(-g_\phi(\nabla^{(t)}_{\boldsymbol{M}_j},\boldsymbol{S}_j^{(t-1)})\right),\boldsymbol{x}_i\right),\boldsymbol{y}_i\right),\end{aligned} \tag{3}$$

where m is the batchsize in the outer loop, n is the batchsize in the inner loop, and we consider T continuous steps in the inner loop once. Note that, forms of the projection function $f(\cdot)$ and the objective function $l(\cdot)$ differ per task. In this section, we will elaborate on specific forms of $f(\cdot)$ and $l(\cdot)$ on the metric nearness, clusters, and similarity learning tasks.

*Corresponding author

1.1. Metric Nearness

Metric nearness refers to the problem of optimally restoring metric properties to distance measurements that happen to be non-metric due to measurement errors or otherwise [1]. In the metric nearness task, given a set of vectors $\{\boldsymbol{x}_i\in\mathbb{R}^d\}_{i=1}^n$ and an SPD matrix $\boldsymbol{A}\in\mathcal{S}^d_{++}$, we expect that we can discover another SPD parameter $\boldsymbol{M}$ to project the feature $\boldsymbol{A}\boldsymbol{x}_i$ back to its original value $\boldsymbol{x}_i$ in the metric space. The prediction function is $f(\boldsymbol{M},\boldsymbol{x}_i)=\boldsymbol{M}\boldsymbol{A}\boldsymbol{x}_i$, and the objective function is $l\big(f(\boldsymbol{M},\boldsymbol{x}_i),\boldsymbol{x}_i\big)=\|f(\boldsymbol{M},\boldsymbol{x}_i)-\boldsymbol{x}_i\|_2^2$, where $\|\cdot\|_2$ is the 2-norm. Thus, the metric nearness task is formulated as minimizing the loss

$$\min_{\boldsymbol{M}\in\mathcal{S}^d_{++}}\mathcal{L}(\boldsymbol{M}) \triangleq \frac{1}{n}\sum_i^n\Big\|\boldsymbol{M}\boldsymbol{A}\boldsymbol{x}_i-\boldsymbol{x}_i\Big\|_2^2. \tag{4}$$

To train our optimizer, we build the meta-objective as

$$\begin{aligned}\min_\phi \mathcal{J}(\phi) &= \frac{1}{m}\sum_t^T\sum_j^m\mathcal{L}(\boldsymbol{M}_j^{(t+1)})\\ &= \frac{1}{mn}\sum_{t,j,i}^{T,m,n}\Big\|\boldsymbol{M}_j^{(t+1)}\boldsymbol{A}\boldsymbol{x}_i-\boldsymbol{x}_i\Big\|_2^2\\ &= \frac{1}{mn}\sum_{t,j,i}^{T,m,n}\Big\|\Gamma_{\boldsymbol{M}_j^{(t)}}\big(-g_\phi(\nabla^{(t)}_{\boldsymbol{M}_j},\boldsymbol{S}_j^{(t-1)})\big)\boldsymbol{A}\boldsymbol{x}_i-\boldsymbol{x}_i\Big\|_2^2.\end{aligned} \tag{5}$$

Updating ϕ. In the outer loop, we resort to the gradient descent to minimize Eq. (5), where the derivative $\frac{\partial\mathcal{J}(\phi)}{\partial\phi}$ is needed. We calculate $\frac{\partial\mathcal{J}(\phi)}{\partial\phi}$ by the truncated backpropagation through time (TBPTT) algorithm. As the meta-objective contains T steps in the inner loop, we denote the loss at step t as $\mathcal{J}^{(t)}(\phi)=\frac{1}{m}\sum_j^m\mathcal{L}(\boldsymbol{M}_j^{(t+1)})$, and $\mathcal{J}(\phi)=\sum_t^T\mathcal{J}^{(t)}(\phi)$. We first compute $\frac{\partial\mathcal{J}^{(t)}(\phi)}{\partial\phi}$ for each step t. Then we compute $\frac{\partial\mathcal{J}(\phi)}{\partial\phi}$ by $\frac{\partial\mathcal{J}(\phi)}{\partial\phi}=\sum_t^T\frac{\partial\mathcal{J}^{(t)}(\phi)}{\partial\phi}$. In this process, the derivatives of the update vector $\boldsymbol{P}^{(t)}$ and the SPD parameter $\boldsymbol{M}^{(t)}$ with respect to $\mathcal{J}^{(t)}(\phi)$ are non-trivial in the retraction operation $\Gamma_{\boldsymbol{M}_j^{(t)}}(\cdot)$, as $\Gamma_{\boldsymbol{M}_j^{(t)}}(\cdot)$ contains the matrix power and matrix exponential operations: $\boldsymbol{M}^{(t)\frac{1}{2}}$, $\boldsymbol{M}^{(t)\frac{-1}{2}}$, and $\mathrm{expm}\big(-\boldsymbol{M}^{(t)\frac{-1}{2}}\boldsymbol{P}^{(t)}\boldsymbol{M}^{(t)\frac{-1}{2}}\big)$. To solve this issue, we rewrite the retraction operation as

$$\begin{cases}\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^\top = \boldsymbol{M}^{(t)} \\ \boldsymbol{U}_Q\boldsymbol{\Sigma}_Q\boldsymbol{U}_Q^\top = \boldsymbol{Q} = -\boldsymbol{U}\boldsymbol{\Sigma}^{\frac{-1}{2}}\boldsymbol{U}^\top\boldsymbol{P}^{(t)}\boldsymbol{U}\boldsymbol{\Sigma}^{\frac{-1}{2}}\boldsymbol{U}^\top \\ \boldsymbol{M}^{(t+1)} = \big(\boldsymbol{U}\boldsymbol{\Sigma}^{\frac{1}{2}}\boldsymbol{U}^\top\big)\big(\boldsymbol{U}_Q\exp(\boldsymbol{\Sigma}_Q)\boldsymbol{U}_Q^\top\big)\big(\boldsymbol{U}\boldsymbol{\Sigma}^{\frac{1}{2}}\boldsymbol{U}^\top\big)\end{cases}, \tag{6}$$

where $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^\top = \boldsymbol{M}^{(t)}$ and $\boldsymbol{U}_Q\boldsymbol{\Sigma}_Q\boldsymbol{U}_Q^\top = \boldsymbol{Q}$ are the eigenvalue decomposition. After obtaining $\frac{\partial\mathcal{J}}{\partial\boldsymbol{U}_Q}$ and $\frac{\partial\mathcal{J}}{\partial\boldsymbol{\Sigma}_Q}$, we can compute $\frac{\partial\mathcal{J}}{\partial\boldsymbol{Q}}$ based on Proposition 1, and $\frac{\partial\mathcal{J}}{\partial\boldsymbol{P}^{(t)}}$ can be computed by $\frac{\partial\mathcal{J}}{\partial\boldsymbol{P}^{(t)}} = -\boldsymbol{U}\boldsymbol{\Sigma}^{\frac{-1}{2}}\boldsymbol{U}^\top\frac{\partial\mathcal{J}}{\partial\boldsymbol{Q}}\boldsymbol{U}\boldsymbol{\Sigma}^{\frac{-1}{2}}\boldsymbol{U}^\top$. Similarly, after obtaining $\frac{\partial\mathcal{J}}{\partial\boldsymbol{U}}$ and $\frac{\partial\mathcal{J}}{\partial\boldsymbol{\Sigma}}$, we can compute $\frac{\partial\mathcal{J}}{\partial\boldsymbol{M}^{(t)}}$. Thus, $\frac{\partial\mathcal{J}^{(t)}(\phi)}{\partial\phi}$ can be computed by the channel rule according to our optimizer architecture.

Algorithm 1 Trainging Process of Our Optimizer

Input: The randomly initialized optimizer parameter. The initial SPD parameter $\boldsymbol{M}^{(0)} = \mathbf{I}_d$. The initial state $\boldsymbol{S}^{(0)} = \mathbf{0}_d$. The initial experience pool $\Psi = \emptyset$.
Output: The optimizer parameter ϕ.
while *not reach the maximum iteration of the observation stage* **do**
    Compute the loss $\mathcal{L}$ of the base learner by Eq. (4);
    Compute the gradient $\nabla^{(t)}_{\boldsymbol{M}}$;
    Update the parameter $\boldsymbol{M}^{(t+1)}$ by Eq. (2);
    Push $\{\boldsymbol{M}^{(t+1)}, \boldsymbol{S}^{(t)}\}$ into Ψ;
end
while *not reach the maximum iteration of the learning stage* **do**
    Randomly select $\{(\boldsymbol{M}_j^{(t)}, \boldsymbol{S}_j^{(t-1)})\}_{j=1}^m$ from Ψ;
    while *not reach T* **do**
        Compute the loss $\mathcal{L}$ of the base learner by Eq. (4);
        Compute the gradient $\nabla^{(t)}_{\boldsymbol{M}}$;
        Update $\boldsymbol{M}^{(t+1)}$ by Eq. (2);
    end
    Compute the loss $\mathcal{J}$ of our optimizer by Eq. (5);
    while *not reach T* **do**
        Compute $\frac{\partial\mathcal{J}^{(t)}(\phi)}{\partial\phi}$;
    end
    Compute $\frac{\partial\mathcal{J}(\phi)}{\partial\phi} = \sum_t^T\frac{\partial\mathcal{J}^{(t)}(\phi)}{\partial\phi}$;
    Update ϕ via the ADAM algorithm;
    if $t + T > \tau$ **then**
        Push $\{(\boldsymbol{M}_j^{(0)} = \mathbf{I}_d, \boldsymbol{S}_j^{(0)} = \mathbf{0}_d\}_{j=1}^m$ into Ψ;
    else
        Push $\{(\boldsymbol{M}_j^{(t+T)}, \boldsymbol{S}_j^{(t+T-1)})\}_{j=1}^m$ into Ψ;
    end
end
Return the parameter ϕ of our optimizer.

Since the derivative $\frac{\partial\mathcal{J}(\phi)}{\partial\phi} = \sum_t^T\frac{\partial\mathcal{J}^{(t)}(\phi)}{\partial\phi}$ is obtained, we utilized the ADAM algorithm to update ϕ, and the learning rate was set as 0.001. A mroe detailed trainging process of our optimizer is shown in Algorithm 1. After obtaining the parameter ϕ of our optimizer, our optimizer can work well on the metric nearness task.

Proposition 1. *Let $\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^\top = \boldsymbol{M}$ be the eigenvalue decomposition of $\boldsymbol{M}$. For the loss function $\mathcal{J}(\boldsymbol{U},\boldsymbol{\Sigma})$, given the derivatives $\frac{\partial\mathcal{J}}{\partial\boldsymbol{U}}$ and $\frac{\partial\mathcal{J}}{\partial\boldsymbol{\Sigma}}$, the derivative $\frac{\partial\mathcal{J}}{\partial\boldsymbol{M}}$ is*

$$\frac{\partial\mathcal{J}}{\partial\boldsymbol{M}} = 2\boldsymbol{U}\big(\boldsymbol{R}^\top\otimes(\boldsymbol{U}^\top\frac{\partial\mathcal{J}}{\partial\boldsymbol{U}} + \frac{\partial\mathcal{J}}{\partial\boldsymbol{U}}^\top\boldsymbol{U})\big)\boldsymbol{U}^\top + \boldsymbol{U}(\frac{\partial\mathcal{J}}{\partial\boldsymbol{\Sigma}})_{diag}\boldsymbol{U}^\top, \tag{7}$$

where

$$\boldsymbol{R}_{ij} = \begin{cases}\frac{1}{\lambda_i-\lambda_j}, & if\ i\neq j \\ 0, & if\ i=j\end{cases},$$

λ_i is the i-th eigenvalue, and $\boldsymbol{X}_{diag}$ denotes $\boldsymbol{X}$ with all off-diagonal elements being 0.

1.2. Clustering

In our visual clustering task, each image is represented by a covariance descriptor [2, 3, 5], and we expect to divide all covariance descriptors into a number of clusters and find their centers. For each image, we resize it to 128×128 and divide the image into 1024 grids. The size of each grid is 4×4. At the position (u,v) of the grid, we extract a 5-D feature vector as

$$\boldsymbol{x}_{uv} = \left[I_{uv}, \left|\frac{\partial I}{\partial u}\right|, \left|\frac{\partial I}{\partial v}\right|, \left|\frac{\partial I^2}{\partial u^2}\right|, \left|\frac{\partial I^2}{\partial v^2}\right|\right]. \tag{8}$$

Based on extracted features, we compute a 5×5 covariance descriptor to represent the image:

$$\boldsymbol{X} = \sum_{uv}(\boldsymbol{x}_{uv}-\boldsymbol{\mu})(\boldsymbol{x}_{uv}-\boldsymbol{\mu})^\top, \tag{9}$$

where $\boldsymbol{\mu}$ is the mean of all feature vectors, and $\boldsymbol{X}\in\mathcal{S}^5_{++}$ is an SPD matrix. A set of covariance descriptors $\{\boldsymbol{X}_i\in\mathcal{S}^5_{++}\}_{i=1}^n$ is the training data, and centers are represented by $\{\boldsymbol{M}_j\in\mathcal{S}^5_{++}\}_{j=1}^m$. In the task, the prediction function $f(\cdot)$ is the affine invariant metric (AIM) [4], which computes distances between centers and descriptors. The distance between a center $\boldsymbol{M}_j$ and a descriptor $\boldsymbol{X}_i$ is

$$f(\boldsymbol{M}_j,\boldsymbol{X}_i) = d(\boldsymbol{M}_j,\boldsymbol{X}_i) = \left\|\mathrm{logm}(\boldsymbol{X}_i^{-\frac{1}{2}}\boldsymbol{M}_j\boldsymbol{X}_i^{-\frac{1}{2}})\right\|_F, \tag{10}$$

where $\|\cdot\|_F$ is the Frobenius-norm, and $\mathrm{logm}(\cdot)$ is the matrix logarithmic function. The objective function $l(\cdot)$ is $l\big(f(\boldsymbol{M}_j,\boldsymbol{X}_i)\big) = f(\boldsymbol{M}_j,\boldsymbol{X}_i)^2$. The optimizer discovers the centers via minimizing distances between each descriptor and the center that the descriptor belongs to, given by

$$\begin{aligned}\min_{\{\boldsymbol{M}_j\in\mathcal{S}^d_{++}\}} \mathcal{L}(\{\boldsymbol{M}_j\}) &\triangleq \frac{1}{n}\sum_i^n d(\boldsymbol{M}_{c^{(i)}},\boldsymbol{X}_i)^2 \\ &= \frac{1}{n}\sum_i^n\left\|\mathrm{logm}(\boldsymbol{X}_i^{-\frac{1}{2}}\boldsymbol{M}_{c^{(i)}}\boldsymbol{X}_i^{-\frac{1}{2}})\right\|_F^2,\end{aligned} \tag{11}$$

where $\boldsymbol{M}_{c(i)}$ denotes the center that $\boldsymbol{X}_i$ belongs to. Since we need to solve multiple centers in the clustering tasks, we do not consider to optimize multiple SPD parameters in the inner loop, and the number of centers is regarded as the batchsize of training our optimizer.

The meta-objective to learn our optimizer is

$$\begin{aligned}
&\min_{\phi} \mathcal{J}(\phi) = \sum_{t}^{T} \mathcal{L}(\{\boldsymbol{M}_j^{(t+1)}\}) \\
=&\frac{1}{n}\sum_{t,i}^{T,n} \left\| \operatorname{logm}(\boldsymbol{X}_i^{-\frac{1}{2}} \boldsymbol{M}_{c(i)}^{(t+1)} \boldsymbol{X}_i^{-\frac{1}{2}}) \right\|_F^2 \\
=&\frac{1}{n}\sum_{t,i}^{T,n} \\
&\left\| \operatorname{logm}\Big(\boldsymbol{X}_i^{-\frac{1}{2}} \Big(\Gamma_{\boldsymbol{M}_{c(i)}^{(t)}} \big(-g_\phi(\nabla^{(t)}_{\boldsymbol{M}_{c(i)}}, \boldsymbol{S}_j^{(t-1)})\big)\Big) \boldsymbol{X}_i^{-\frac{1}{2}} \Big) \right\|_F^2 .
\end{aligned} \tag{12}$$

Minimizing Eq. (12) and obtaining the optimal parameter ϕ for the clustering task are similar to those in the metric nearness task.

1.3. Similarity Learning

Similarity learning aims to learn a Mahalanobis metric, through which similar samples have small distances and dissimilar samples have large distances. In similarity learning experiments, we extract a feature vector $\boldsymbol{x}_i \in \mathbb{R}^d$ from each image, pairs of vectors $(\boldsymbol{x}_i, \boldsymbol{x}_{i'})$ are the training data, and their labels are $y_{ii'}$. $y_{ii'} = 1$ means $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i'}$ are similar; otherwise, $y_{ii'} = 0$. In this task, the prediction function $f(\cdot)$ is the Mahalanobis metric that computes the distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i'}$ by

$$\begin{aligned}
&f(\boldsymbol{M}, \boldsymbol{x}_i, \boldsymbol{x}_{i'}) = d_{\boldsymbol{M}}(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) \\
=&\sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_{i'})^\top \boldsymbol{M} (\boldsymbol{x}_i - \boldsymbol{x}_{i'})},
\end{aligned} \tag{13}$$

where $\boldsymbol{M} \in \mathbb{R}^{d\times d}$ is the metric matrix, and the non-negative condition of the metric requires $\boldsymbol{M}$ to be an SPD matrix, *i.e.*, $\boldsymbol{M} \in \mathcal{S}_{++}^d$. We use the contrastive loss as the objective function $l(\cdot)$, given by

$$\begin{aligned}
&l\big(f(\boldsymbol{M}, \boldsymbol{x}_i, \boldsymbol{x}_{i'}), y_{ii'}\big) \\
=&y_{ii'} \cdot \max\Big(f(\boldsymbol{M}, \boldsymbol{x}_i, \boldsymbol{x}_{i'}) - \zeta_s, 0\Big)^2 \\
&+(1-y_{ii'}) \cdot \max\Big(\zeta_d - f(\boldsymbol{M}, \boldsymbol{x}_i, \boldsymbol{x}_{i'}), 0\Big)^2 .
\end{aligned} \tag{14}$$

We expect that the distance between two similar samples is smaller than a threshold ζ_s, and the distance between two dissimilar samples is larger than a threshold ζ_d. The loss function of the similarity learning task is

$$\begin{aligned}
&\min_{\boldsymbol{M}\in\mathcal{S}_{++}^d} \mathcal{L}(\boldsymbol{M}) = \frac{1}{n}\sum_{i=1}^{n} l\big(f(\boldsymbol{M}, \boldsymbol{x}_i, \boldsymbol{x}_{i'}), y_{ii'}\big) \\
=&\frac{1}{|\mathcal{S}|}\sum_{i,i'\in\mathcal{S}} y_{ii'} \cdot \max\Big(d_{\boldsymbol{M}}(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) - \zeta_s, 0\Big)^2 \\
&+ \frac{1}{|\mathcal{D}|}\sum_{i,i'\in\mathcal{D}} (1-y_{ii'}) \cdot \max\Big(\zeta_d - d_{\boldsymbol{M}}(\boldsymbol{x}_i, \boldsymbol{x}_{i'}), 0\Big)^2 ,
\end{aligned} \tag{15}$$

where n is the number of pairs, $\mathcal{S}$ is the similar pair set, $\mathcal{D}$ is the dissimilar pair set, and $|\mathcal{S}|$ and $|\mathcal{D}|$ are numbers of pairs in $\mathcal{S}$ and $\mathcal{D}$, respectively.

We train our optimizer according to the following meta-objective

$$\begin{aligned}
&\min_{\phi} \mathcal{J}(\phi) = \frac{1}{m}\sum_{t}^{T}\sum_{j}^{m} \mathcal{L}(\boldsymbol{M}_j^{(t+1)}) \\
=&\frac{1}{m}\sum_{t,j}^{T,m} \Bigg(\frac{1}{|\mathcal{S}|}\sum_{i,i'\in\mathcal{S}} y_{ii'} \cdot \max\Big(d_{\boldsymbol{M}_j^{(t+1)}}(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) - \zeta_s, 0\Big)^2 \\
&+ \frac{1}{|\mathcal{D}|}\sum_{i,i'\in\mathcal{D}} (1-y_{ii'}) \cdot \max\Big(\zeta_d - d_{\boldsymbol{M}_j^{(t+1)}}(\boldsymbol{x}_i, \boldsymbol{x}_{i'}), 0\Big)^2 \Bigg) \\
=&\frac{1}{m}\sum_{t,j}^{T,m} \Bigg(\frac{1}{|\mathcal{S}|}\sum_{i,i'\in\mathcal{S}} y_{ii'} \\
&\cdot \max\Big(\sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_{i'})^\top \Gamma_{\boldsymbol{M}_j^{(t)}}\big(-g_\phi(\nabla^{(t)}_{\boldsymbol{M}}, \boldsymbol{S}_j^{(t-1)})\big)(\boldsymbol{x}_i - \boldsymbol{x}_{i'})} \\
&- \zeta_s, 0\Big)^2 \\
&+ \frac{1}{|\mathcal{D}|}\sum_{i,i'\in\mathcal{D}} (1-y_{ii'}) \cdot \max \\
&\Big(\zeta_d - \sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_{i'})^\top \Gamma_{\boldsymbol{M}_j^{(t)}}\big(-g_\phi(\nabla^{(t)}_{\boldsymbol{M}}, \boldsymbol{S}_j^{(t-1)})\big)(\boldsymbol{x}_i - \boldsymbol{x}_{i'})}, 0\Big)^2 \Bigg),
\end{aligned} \tag{16}$$

Minimizing Eq. (16) and obtaining the optimal parameter ϕ of our optimizer for the similarity learning task are similar to those in the metric nearness task.

References

[1] Justin Brickell, Inderjit S Dhillon, Suvrit Sra, and Joel A Tropp. The metric nearness problem. *SIAM Journal on Matrix Analysis and Applications*, 30(1):375–396, 2008.

[2] Zhiwu Huang, Ruiping Wang, Shiguang Shan, Xianqiu Li, and Xilin Chen. Log-euclidean metric learning on symmetric positive definite manifold with application to image set classification. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 720–729, 2015.

[3] Peihua Li, Jiangtao Xie, Qilong Wang, and Wangmeng Zuo. Is second-order information helpful for large-scale visual recognition? In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2070–2078, 2017.

[4] Xavier Pennec, Pierre Fillard, and Nicholas Ayache. A riemannian framework for tensor computing. *International Journal of Computer Vision (IJCV)*, 66(1):41–66, 2006.

[5] Lei Wang, Jianjia Zhang, Luping Zhou, Chang Tang, and Wanqing Li. Beyond covariance: Feature representation with nonlinear kernel matrices. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.