Learning to Optimize on SPD Manifolds
Supplementary Materials

Zhi Gao1, Yuwei Wu1,∗, Yunde Jia1, Mehrtash Harandi2
1Beijing Laboratory of Intelligent Information Technology
School of Computer Science, Beijing Institute of Technology, Beijing, China
2Department 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
\[ \mathcal{L}(M) \triangleq \frac{1}{n} \sum_{i=1}^{n} l(f(M, x_i), y_i), \] (1)
with respect to \( M \), that is \( \min_{M \in S_{++}^d} \mathcal{L}(M) \). In Eq. (1), \( x_i \) is the \( i \)-th training sample, and its corresponding target (e.g., label) is \( y_i \). \( f(\cdot) \) and \( l(\cdot) \) represent the prediction and objective functions, respectively, and \( M \in S_{++}^d \) encapsulate parameters of the optimization problem.

To solve Eq. (1), we parameter the SPD optimizer by a network, and \( \phi \) is the parameter of the network, through which the SPD parameter is updated by
\[ M^{(t+1)} = \Gamma^{(t)}(\phi) \left( -g_\phi(\nabla^{(t)}_M, S^{(t-1)}) \right), \] (2)
where \( \Gamma^{(t)}(\cdot) \) is the retraction operation, and \( g_\phi(\nabla^{(t)}_M, S^{(t-1)}) \) is the update vector on the tangent space. We train the optimizer by minimizing the following meta-objective
\[ \mathcal{J}(\phi) = \frac{1}{m} \sum_{t} \sum_{j} \mathcal{L}(M^{(t)}_j) \]
\[ = \frac{1}{mn} \sum_{t} \sum_{j,i} \left( f\left(M^{(t)}_j, x_i\right), y_i \right) \]
\[ = \frac{1}{mn} \sum_{t} \sum_{j,i} \left( f\left(\Gamma^{(t)}_j(\phi) - g_\phi(\nabla^{(t)}_M, S^{(t-1)}), x_i\right), y_i \right), \] (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 \( \{x_i \in \mathbb{R}^d\}_{i=1}^n \) and an SPD matrix \( A \in S_{++}^d \), we expect that we can discover another SPD parameter \( M \) to project the feature \( Ax_i \) back to its original value \( x_i \) in the metric space. The prediction function is \( f(M, x_i) = MAx_i \), and the objective function is \( l(f(M, x_i), x_i) = \|f(M, x_i) - x_i\|_2^2 \), where \( \| \cdot \|_2 \) is the 2-norm. Thus, the metric nearness task is formulated as minimizing the loss
\[ \min_{M \in S_{++}^d} \mathcal{L}(M) \triangleq \frac{1}{n} \sum_{i} \|MAx_i - x_i\|_2^2. \] (4)
To train our optimizer, we build the meta-objective as
\[ \min_{\phi} \mathcal{J}(\phi) = \frac{1}{m} \sum_{t} \sum_{j} \mathcal{L}(M^{(t+1)}_j) \]
\[ = \frac{1}{mn} \sum_{t} \sum_{j,i} \left( M^{(t+1)}_j Ax_i - x_i \right)^2 \]
\[ = \frac{1}{mn} \sum_{t} \sum_{j,i} \left( \Gamma^{(t)}_j(\phi) - g_\phi(\nabla^{(t)}_M, S^{(t-1)})) Ax_i - x_i \right)^2. \] (5)

Updating \( \phi \). 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 \( \mathcal{J}^{(t)}(\phi) = \frac{1}{m} \sum_{j} \mathcal{L}(M^{(t+1)}_j) \), and \( \mathcal{J}(\phi) = \sum_{t} \mathcal{J}^{(t)}(\phi) \). We first compute \( \frac{\partial \mathcal{J}^{(t)}(\phi)}{\partial \phi} \) for each step \( t \). Then we compute \( \frac{\partial \mathcal{J}(\phi)}{\partial \phi} = \sum_{t} \frac{\partial \mathcal{J}^{(t)}(\phi)}{\partial \phi} \). In this process, the derivatives of the update vector \( P^{(t)} \) and
the SPD parameter $M^{(t)}$ with respect to $\mathcal{J}^{(t)}(\phi)$ are non-trivial in the retraction operation $\Gamma_{M^{(t)}}(\cdot)$, as $\Gamma_{M^{(t)}}(\cdot)$ contains the matrix power and matrix exponential operations: $M^{(t)} \frac{\partial}{\partial t} M^{(t)}$, and $\exp_m(-M^{(t)} P^{(t)} M^{(t)} \frac{1}{2})$. To solve this issue, we rewrite the retraction operation as

$$
\begin{align*}
\frac{\partial}{\partial M} & \begin{bmatrix} U \Sigma U^T = M^{(t)} \\
U_Q \Sigma_Q U_Q^T = Q \\
M^{(t+1)} = (U \Sigma^{-\frac{1}{2}} U^T)(U_Q \exp(\Sigma_Q) U_Q^T)(U \Sigma^{-\frac{1}{2}} U^T)
\end{bmatrix}
\end{align*}
$$

where $U \Sigma U^T = M^{(t)}$ and $U_Q \Sigma_Q U_Q^T = Q$ are the eigenvalue decomposition. After obtaining $\frac{\partial \mathcal{J}}{\partial U}$ and $\frac{\partial \mathcal{J}}{\partial \Sigma}$, we can compute $\frac{\partial \mathcal{J}}{\partial M}$ based on Proposition 1 and $\frac{\partial \mathcal{J}}{\partial \Sigma}$ can be computed by

$$
\frac{\partial \mathcal{J}^{(t)}}{\partial M} = -U \Sigma^{-\frac{1}{2}} U^T \frac{\partial \mathcal{J}}{\partial Q} U \Sigma^{-\frac{1}{2}} U^T.
$$

Similarly, after obtaining $\frac{\partial \mathcal{J}}{\partial U}$ and $\frac{\partial \mathcal{J}}{\partial \Sigma}$, we can compute $\frac{\partial \mathcal{J}^{(t)}}{\partial \Sigma}$, which can be computed by the channel rule according to our optimizer architecture.

**Algorithm 1** Training Process of Our Optimizer

**Input:** The randomly initialized optimizer parameter. The initial SPD parameter $M^{(0)} = I_d$. The initial state $S^{(0)} = 0_d$.

**Output:** The optimizer parameter $\phi$.

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 $\nabla \mathcal{J}^{(t)}$:

Update the parameter $M^{(t+1)}$ by Eq. (2):

Push $\{M^{(t+0)}, S^{(t)}\}$ into $\Psi$;

end

while not reach the maximum iteration of the learning stage do

Randomly select $\{\{(M^{(t)}, S^{(t+1)})\}\}_{t=1}^{n}$ from $\Psi$;

while not reach $T$ do

Compute the loss $\mathcal{J}$ of the optimizer by Eq. (5):

Compute the gradient $\nabla \mathcal{J}^{(t)}$:

Update $M^{(t+1)}$ by Eq. (2):

end

Compute the loss $\mathcal{J}$ of our optimizer by Eq. (5):

while not reach $T$ do

Compute $\frac{\partial \mathcal{J}^{(t)}}{\partial \phi}$:

Update $\phi$ via the ADAM algorithm;

if $t + T > \tau$ then

Push $\{(M^{(0)}, \Sigma^{(0)})\}$ into $\Psi$;

else

Push $\{(M^{(t+T)}, S^{(1+T)})\}_{t=1}^{n}$ into $\Psi$;

end

end

Return the parameter $\phi$ of our optimizer.

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

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

$$
\frac{\partial \mathcal{J}}{\partial M} = 2U(R^T \otimes (U^T \frac{\partial \mathcal{J}}{\partial U} U + U^T \frac{\partial \mathcal{J}}{\partial U} U))U^T + U(\frac{\partial \mathcal{J}}{\partial \Sigma})_{\text{diag}} U^T,
$$

where

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

$\lambda_i$ is the $i$-th eigenvalue, and $X_{\text{diag}}$ denotes $X$ 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 $128 \times 128$ and divide the image into $1024$ grids. The size of each grid is $4 \times 4$. At the position $(u, v)$ of the grid, we extract a 5-D feature vector as

$$
x_{uv} = \left[ \frac{\partial I}{\partial u}, \frac{\partial I}{\partial v}, \frac{\partial I^2}{\partial u}, \frac{\partial I^2}{\partial v}, \frac{\partial I^{2}}{\partial u^2} \right].
$$

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

$$
X = \sum_{uv} (x_{uv} - \mu)(x_{uv} - \mu)^T,
$$

where $\mu$ is the mean of all feature vectors, and $X \in S^{5}_{++}$ is an SPD matrix. A set of covariance descriptors $\{X_i \in S^{5}_{++}: i = 1, \ldots, n\}$ is the training data, and centers are represented by $\{M_j \in S^{5}_{++}: j = 1, \ldots, m\}$. 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 $M_j$ and a descriptor $X_i$ is

$$
f(M_j, X_i) = d(M_j, X_i) = \| \logm(X_i^{-\frac{1}{2}} M_j X_i^{-\frac{1}{2}}) \|_F^2,
$$

where $\cdot$ denotes the Frobenius-norm, and $\logm(\cdot)$ is the matrix logarithmic function. The objective function $l(\cdot)$ is $l(f(M_j, X_i)) = f(M_j, X_i)^2$. The optimizer discovers the centers via minimizing distances between each descriptor and the center that the descriptor belongs to, given by

$$
\min_{\{M_j \in S^{5}_{++}\}} \mathcal{L}(\{M_j\}) \equiv \frac{1}{n} \sum_{i} d(M_{c(i)}, X_i)^2 = \frac{1}{n} \sum_{i} \| \logm(X_i^{-\frac{1}{2}} M_{c(i)} X_i^{-\frac{1}{2}}) \|_F^2.
$$
where $M_{c(i)}$ denotes the center that $X_i$ 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

\[
\min_{\phi} \mathcal{J}(\phi) = \frac{1}{n} \sum_{i,t} L(\{M_{c(i)}^{(t+1)}\})
\]

\[
= \frac{1}{n} \sum_{i,t} \left\| \log(\mathbf{X}_i^{-\frac{1}{2}} M_{c(i)}^{(t+1)} \mathbf{X}_i^{-\frac{1}{2}}) \right\|^2_F
\]

\[
= \frac{1}{n} \sum_{i,t} \left\| \log(\mathbf{X}_i^{-\frac{1}{2}} (\Gamma_{M_{c(i)}} (-g_\phi(\nabla_{M_{c(i)}^{(t)}}, S_j^{(t-1)}))) \mathbf{X}_i^{-\frac{1}{2}}) \right\|^2_F.
\]

Minimizing Eq. (12) and obtaining the optimal parameter $\phi$ 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 $x_i \in \mathbb{R}^d$ from each image, pairs of vectors $(x_i, x_{i'})$ are the training data, and their labels are $y_{ii'}$. $y_{ii'} = 1$ means $x_i$ and $x_{i'}$ are similar; otherwise, $y_{ii'} = 0$. In this task, the prediction function $f(\cdot)$ is the Mahalanobis metric that computes the distance between $x_i$ and $x_{i'}$ by

\[
f(M, x_i, x_{i'}) = d_M(x_i, x_{i'}) = \sqrt{(x_i - x_{i'})^\top M (x_i - x_{i'})},
\]

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

\[
l(f(M, x_i, x_{i'}), y_{ii'}) = y_{ii'} \cdot \max \left( f(M, x_i, x_{i'}) - \zeta_s, 0 \right)^2
\]

\[+(1 - y_{ii'}) \cdot \max \left( \zeta_d - f(M, x_i, x_{i'}), 0 \right)^2.
\]

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

\[
\min_{M \in S_+^d} \mathcal{L}(M) = \frac{1}{n} \sum_{i=1}^n l(f(M, x_i, x_{i'}), y_{ii'})
\]

\[= \frac{1}{|S|} \sum_{i,i' \in S} y_{ii'} \cdot \max \left( d_M(x_i, x_{i'}) - \zeta_s, 0 \right)^2
\]

\[+ \frac{1}{|D|} \sum_{i,i' \in D} (1 - y_{ii'}) \cdot \max \left( \zeta_d - d_M(x_i, x_{i'}), 0 \right)^2,
\]

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

We train our optimizer according to the following meta-objective

\[
\min_{\phi} \mathcal{J}(\phi) = \frac{1}{m} \sum_{t} \sum_{j} L(M_{c(i)}^{(t+1)})
\]

\[= \frac{1}{m} \sum_{t,j} \left( \sum_{i,i' \in S} y_{ii'} \cdot \max \left( d_{M_{c(i)}^{(t+1)}}(x_i, x_{i'}) - \zeta_s, 0 \right)^2
\]

\[+ \frac{1}{|D|} \sum_{i,i' \in D} (1 - y_{ii'}) \cdot \max \left( \zeta_d - d_{M_{c(i)}^{(t+1)}}(x_i, x_{i'}), 0 \right)^2
\]

\[= \frac{1}{m} \sum_{t,j} \left( \frac{1}{|S|} \sum_{i,i' \in S} y_{ii'}
\cdot \max \left( \sqrt{(x_i - x_{i'})^\top \Gamma_{M_{c(i)}^{(t)}} (-g_\phi(\nabla M_{c(i)}^{(t)}, S_j^{(t-1)}))) (x_i - x_{i'})
\right.
\right.
\]

\[+ \left. \zeta_s, 0 \right)^2
\]

\[+ \frac{1}{|D|} \sum_{i,i' \in D} (1 - y_{ii'}) \cdot \max \left( \zeta_d - \sqrt{(x_i - x_{i'})^\top \Gamma_{M_{c(i)}^{(t)}} (-g_\phi(\nabla M_{c(i)}^{(t)}, S_j^{(t-1)}))) (x_i - x_{i'}), 0 \right)^2,
\]

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

References


