Learning to Optimize on SPD Manifolds

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

Abstract

Many tasks in computer vision and machine learning are modeled as optimization problems with constraints in the form of Symmetric Positive Definite (SPD) matrices. Solving such optimization problems is challenging due to the non-linearity of the SPD manifold, making optimization with SPD constraints heavily relying on expert knowledge and human involvement. In this paper, we propose a meta-learning method to automatically learn an iterative optimizer on SPD manifolds. Specifically, we introduce a novel recurrent model that takes into account the structure of input gradients and identifies the updating scheme of optimization. We parameterize the optimizer by the recurrent model and utilize Riemannian operations to ensure that our method is faithful to the geometry of SPD manifolds. Compared with existing SPD optimizers, our optimizer effectively exploits the underlying data distribution and learns a better optimization trajectory in a data-driven manner. Extensive experiments on various computer vision tasks including metric nearness, clustering, and similarity learning demonstrate that our optimizer outperforms existing state-of-the-art methods consistently.

1. Introduction

Optimization problems with constraints in the form of Symmetric Positive Definite (SPD) matrices are pervasive in the computer vision and machine learning communities with a variety of applications. For example, in similarity learning, the non-negativity of the Mahalanobis distance requires the metric to be an SPD matrix [13, 22, 34]. To fit a Gaussian mixture model to data, the SPD constraint is required to obtain the covariance of each model [18]. Other applications range from coding with higher-order statistics [7, 15], to visual classification [8, 9, 10, 12, 14, 41, 42], and clustering [11, 21].

*Corresponding author

In general, the SPD constraint makes the optimization problems challenging and non-trivial to address [35]. Directly applying gradient-based optimization algorithms in the Euclidean space (*e.g.*, Stochastic Gradient Descent (SGD)) to such optimization problems is not fruitful and lead to results that do not comply with the geometry of the SPD manifold. To alleviate this issue, one needs to resort to the Riemannian form of gradient-based optimization algorithms [1]. Such optimizers view optimization problems with manifold constraints as unconstrained problems and utilize Riemannian operators to move along the manifold (SPD manifold in our case) in the quest for the solution.

Prior to our work, all the efforts and the focus of the community were directed towards designing Riemannian optimizers by hand [5, 23, 24, 25, 38, 46]. For example, Bonnabel [5] developed the SGD algorithm on Riemannian manifolds, and Roy *et al.* [25] generalized the momentum SGD to the Riemannian setting. The design of the optimizer, evaluating its performance, and probably updating the optimization scheme requires intuitions and expert knowledge about manifolds to achieve a satisfactory result. That is, the hand-designed optimizer relies heavily on expert knowledge and human involvements. Besides, existing SPD optimizers are task-independent. Since the underlying data distribution differs among various tasks [2, 32], an efficient optimizer should be tailored to the task in hand. Thus, a natural question arises: can we automatically design task-specific optimizers on SPD manifolds?

Our work is inspired by advances in meta-learning that offers a new perspective to optimization techniques. The meta-learning can automatically acquire knowledge to learn how to learn or quickly adapt to new information [27, 37]. In particular, some methods show that the optimization knowledge can be acquired by parameterizing the optimizer with a neural network, however, the network is trained to optimize in the Euclidean space [2, 17, 28, 43]. To the best of our knowledge, existing works have never attempted to learn the optimizer on SPD manifolds. A possible reason is that, directly applying existing meta-learning methods to SPD manifolds do not preserve the non-Euclidean geometry. This motivates us to study how to leverage meta-learning to automate the design of SPD optimizers.

To achieve this goal, two challenges need to be overcome. (1) The Riemannian gradient on SPD manifolds is identified by symmetric matrices. Naively processing the gradient inevitably destroys the symmetric structure that is indispensable during optimization. We should consider how to preserve the symmetric property of gradient matrices. (2) On a non-linear space such as an SPD manifold, operations like additions of two gradients need to be faithful to the intrinsic geometry of the manifold.

In this paper, we present a meta-learning method to automatically learn an optimizer on SPD manifolds. Specifically, we introduce a matrix LSTM (mLSTM) that can preserve the symmetric property of gradients and automatically generate an update step. We parameterize the optimizer by mLSTM and utilize Riemannian operations to ensure that the resulting design will comply with the geometry of the SPD manifolds. By training the optimizer to minimize the objective of the base learner, one effectively exploits the underlying data distribution and learns a better optimization trajectory in a data-driven manner. We conduct experiments on various computer vision tasks including the metric nearness, clustering, and similarity learning. Experimental results demonstrate that our optimizer converges faster and has a better optima than existing SPD optimizers. The code is available at `https://github.com/ZhiGaomcislab/Learning-to-optimize-on-SPD-manifolds`.

In summary, our main contributions are two-fold.

(1) We propose the first ever learnable optimizer on non-Euclidean SPD manifolds. The resulting optimizer learns how to optimize on SPD manifolds.

(2) We parameterize the optimizer by a novel structure, the mLSTM model and make use of Riemmanian operations to ensure that the overall method is faithful to the geometry of SPD manifolds.

2. Related Work

2.1. Optimization on SPD Manifolds

Optimization problems with SPD constraints can be solved by gradient-based Riemannian optimization algorithms. Luenberger [31] proposed the Riemannian gradient descent (RGD) algorithm, where the optimizer operates on all samples at each iteration. Bonnabel [5] presented the Riemannian stochastic gradient descent (RSGD) to reduce the computational load. RSGD is widely used on SPD manifolds, despite suffering from the variance problem [46], and the learning rate for RSGD acts equally for all coordinates. To solve the gradient variance issue, Roy *et al.* [25] utilized the momentum technique and designed the optimizer to consider previous solutions, Zhang *et al.* [46] and Sato *et al.* [38] exploited the finite-sum of the loss function and proposed the Riemannian stochastic variance reduced gradient (RSVRG) algorithm. Kasai *et al.* [24] used recursive gradients [33] and presented the Riemannian stochastic recursive gradient (RSRG) algorithm. Furthermore, some works studied adaptive optimization, where different coordinates have different learning rates. For example, Roy *et al.* [25] computed Euclidean-style adaptive weights for different coordinates. Kasai *et al.* [23] proposed adapting the row and column subspaces of gradients for different coordinates. Compared with existing works [5, 23, 24, 25, 31, 38, 46] where optimizers are all designed by hand, we introduce a meta-learning method to automatically design an SPD optimizer. In addition, our optimizer is task-specific and can effectively exploit underlying data distribution to learn a better optimization trajectory in a data-driven manner.

2.2. Learning to Optimize

Learning to optimize has a long history in learning theory and is dated back to the seminal work of Schmidhuber [39] that presented networks with the capability of modifying their own weights. Bengio *et al.* [4] introduced learning update rules for parameters to avoid back-propagation. Andrychowicz *et al.* [2], Ravi and Larochelle [36], Wichrowska *et al.* [43], and Rusu *et al.* [37] utilized a recurrent model as the optimizer and trained the optimizer by gradient information. Moreover, Bello *et al.* [3], and Li and Malik [28] made use of reinforcement learning to learn the recurrent model for optimization. Unfortunately, all these seminal methods cannot be applied to optimization problems with constraints. Recently, Xie *et al.* [45] proposed a differentiable linearized ADMM method to learn to solve problems with linear constraints. Nevertheless, the developed ADMM is still limited to the Euclidean space. In contrast, we focus on learning to optimize on non-linear spaces and in particular on SPD manifolds.

3. Preliminaries

Before introducing our method, we provide a brief review of the geometry and operations on SPD manifolds. Note that throughout this paper, vectors are represented by bold lower-case letters (*e.g.*, $\boldsymbol{x}$), and matrices are denoted by bold upper-case letters (*e.g.*, $\boldsymbol{M}$). We use $\mathbf{I}_d$ and $\mathbf{0}_d$ to denote the $d \times d$ identity matrix and a $d \times d$ matrix whose elements are all '0's, respectively.

The SPD Manifold. The SPD manifold $\mathcal{S}_{++}^d$ is composed of all $d \times d$ SPD matrices:

$$\begin{aligned}\mathcal{S}_{++}^d = \{\boldsymbol{M} \in \mathbb{R}^{d\times d} :& \boldsymbol{M} = \boldsymbol{M}^\top, \\ & \boldsymbol{x}^\top \boldsymbol{M}\boldsymbol{x} > 0, \forall \boldsymbol{x} \in \mathbb{R}^d \backslash \{\mathbf{0}_d\}\},\end{aligned} \tag{1}$$

where $\top$ is the matrix transpose operation, and $\boldsymbol{x}$ is any non-zero vector. For a point $\boldsymbol{M}$ on an SPD manifold $\mathcal{S}_{++}^d$, its tangent space is represented by $\mathcal{T}_{\boldsymbol{M}}\mathcal{S}_{++}^d$. The tangent space is a Euclidean space containing all vectors that are tangent to $\mathcal{S}_{++}^d$ at $\boldsymbol{M}$.

Matrix logarithm function. Given an SPD matrix $\boldsymbol{M} \in \mathcal{S}_{++}^d$, the matrix logarithm function $\operatorname{logm}(\boldsymbol{M}) : \mathcal{S}_{++}^d \rightarrow \operatorname{sym}(d)$ is

$$\operatorname{logm}(\boldsymbol{M}) = \boldsymbol{U}\log(\boldsymbol{\Sigma})\boldsymbol{U}^\top, \tag{2}$$

where $\operatorname{sym}(d)$ denotes the space of $d \times d$ symmetric matrices. $\boldsymbol{U\Sigma U}^\top = \boldsymbol{M}$ is the eigenvalue decomposition, $\boldsymbol{\Sigma}$ is the diagonal eigenvalue matrix, $\boldsymbol{U}$ is the eigenvector matrix, and $\log(\boldsymbol{\Sigma})$ denotes the logarithm operation on diagonal elements of $\boldsymbol{\Sigma}$.

Matrix exponential function. Given a symmetric matrix $\boldsymbol{N} \in \operatorname{sym}(d)$, the matrix exponential function $\operatorname{expm}(\boldsymbol{N}) : \operatorname{sym}(d) \rightarrow \mathcal{S}_{++}^d$ is computed by

$$\operatorname{expm}(\boldsymbol{N}) = \boldsymbol{U}\exp(\boldsymbol{\Sigma})\boldsymbol{U}^\top, \tag{3}$$

where $\boldsymbol{U\Sigma U}^\top = \boldsymbol{N}$ is the eigenvalue decomposition, and $\exp(\boldsymbol{\Sigma})$ denotes the exponential operation on diagonal elements of $\boldsymbol{\Sigma}$.

Retraction operation on SPD Manifolds. The retraction on Riemannian manifolds is a smooth mapping from the tangent space onto the manifold with a local rigidity condition ([1], Section 4.1). On SPD manifolds, the exponential map is often used as the retraction operation $\Gamma_{\boldsymbol{M}}(\cdot) : \mathcal{T}_M\mathcal{S}_{++}^d \rightarrow \mathcal{S}_{++}^d$, given by

$$\Gamma_{\boldsymbol{M}}(\boldsymbol{\zeta}) = \boldsymbol{M}^{\frac{1}{2}}\operatorname{expm}\big(\boldsymbol{M}^{-\frac{1}{2}}\boldsymbol{\zeta}\boldsymbol{M}^{-\frac{1}{2}}\big)\boldsymbol{M}^{\frac{1}{2}}, \tag{4}$$

where $\boldsymbol{\zeta} \in \mathbb{R}^{d\times d}$ is a point on the tangent space, and the tangent space is at $\boldsymbol{M}$.

Orthogonal projection on SPD Manifolds. The orthogonal projection on Riemannian manifolds transforms an arbitrary gradient at point $\boldsymbol{M}$ into the Riemannian gradient that is on the tangent space. On SPD manifolds, the orthogonal projection $\pi_{\boldsymbol{M}}(\cdot) : \mathbb{R}^{d\times d} \rightarrow \mathcal{T}_M\mathcal{S}_{++}^d$ is defined by

$$\pi_{\boldsymbol{M}}(\nabla_{\boldsymbol{M}}) = \boldsymbol{M}\frac{1}{2}(\nabla_{\boldsymbol{M}} + \nabla_{\boldsymbol{M}}^\top)\boldsymbol{M}, \tag{5}$$

where $\boldsymbol{M} \in \mathcal{S}_{++}^d$, and $\nabla_{\boldsymbol{M}} \in \mathbb{R}^{d\times d}$ is an arbitrary gradient at point $\boldsymbol{M}$.

4. Learning an Optimizer on SPD Manifolds

4.1. Optimization Framework

In general, optimization problems with SPD constraints can be formulated as minimizing a loss

$$\mathcal{L}(\boldsymbol{M}) \triangleq \frac{1}{n}\sum_{i=1}^{n} l\big(f(\boldsymbol{M}, \boldsymbol{x}_i), \boldsymbol{y}_i\big), \tag{6}$$

with respect to $\boldsymbol{M}$, that is $\min_{\boldsymbol{M}\in\mathcal{S}_{++}^d} \mathcal{L}(\boldsymbol{M})$. In Eq. (6), $\boldsymbol{x}_i \in \mathbb{R}^d$ is the i-th training sample, and its corresponding

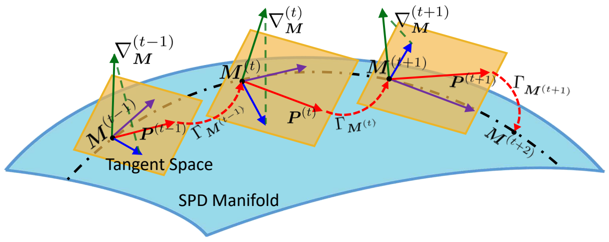
$\nabla_{\boldsymbol{M}}^{(t-1)}$
$\nabla_{\boldsymbol{M}}^{(t)}$
$\nabla_{\boldsymbol{M}}^{(t+1)}$
$\boldsymbol{M}^{(t-1)}$
$\boldsymbol{M}^{(t)}$
$\boldsymbol{M}^{(t+1)}$
$\boldsymbol{M}^{(t+2)}$
$\boldsymbol{P}^{(t-1)}$
$\boldsymbol{P}^{(t)}$
$\boldsymbol{P}^{(t+1)}$
$\Gamma_{\boldsymbol{M}^{(t-1)}}$
$\Gamma_{\boldsymbol{M}^{(t)}}$
$\Gamma_{\boldsymbol{M}^{(t+1)}}$
Tangent Space
SPD Manifold

Figure 1. The illustration of optimization on SPD manifolds. The black dotted line shows a geodesic. Green solid lines, blue solid lines, purple solid lines, and red solid lines denote arbitrary gradients, Riemannian gradients on tangent spaces, previous optimization state, and update vectors, respectively. Red curved lines indicate the retraction, and green dotted lines denote orthogonal projection.

target (*e.g.*, label) is $\boldsymbol{y}_i \in \mathbb{R}^k$. $f(\cdot) : \mathcal{S}_{++}^d \times \mathbb{R}^d \rightarrow \mathbb{R}^k$ and $l(\cdot) : \mathbb{R}^k \times \mathbb{R}^k \rightarrow \mathbb{R}_+$ represent the prediction and objective functions, respectively, and $\boldsymbol{M} \in \mathcal{S}_{++}^d$ encapsulates parameters of the optimization problem. Solving problem (6) is challenging, due to the non-Euclidean geometry. Directly using a gradient descent optimizer in the Euclidean space, that is

$$\boldsymbol{M}^{(t+1)} = \boldsymbol{M}^{(t)} - \eta^{(t)}\nabla_{\boldsymbol{M}}^{(t)}, \tag{7}$$

with step-size $\eta^{(t)}$ and $\nabla_{\boldsymbol{M}}^{(t)} \in \operatorname{sym}(d)$ being the gradient of the loss function $\mathcal{L}$ with respect to $\boldsymbol{M}^{(t)}$ (calculated at time t), will not comply with the manifold structure.

To alleviate this issue, a gradient descent optimizer that considers the Riemannian setting [31] is formulated by

$$\begin{cases} \boldsymbol{M}^{(t+1)} = \Gamma_{\boldsymbol{M}^{(t)}}(-\boldsymbol{P}^{(t)}) \\ \boldsymbol{P}^{(t)} = \eta^{(t)}\pi_{\boldsymbol{M}^{(t)}}(\nabla_{\boldsymbol{M}}^{(t)}) \end{cases}, \tag{8}$$

where $\boldsymbol{P}^{(t)}$ is an update vector on the tangent space. $\pi_{\boldsymbol{M}^{(t)}}(\cdot)$ and $\Gamma_{\boldsymbol{M}^{(t)}}(\cdot)$ are the orthogonal projection and retraction operators, respectively. The Riemannian gradient $\pi_{\boldsymbol{M}^{(t)}}(\nabla_{\boldsymbol{M}}^{(t)})$ is a search direction on the tangent space. The connection of Eq. (8) to Eq. (7) can be established by noting that the retraction operation $\Gamma_{\boldsymbol{M}^{(t)}}(\boldsymbol{P}^{(t)})$ is equivalent to $\boldsymbol{M}^{(t)} - \boldsymbol{P}^{(t)}$ and the orthogonal projection $\pi_{\boldsymbol{M}^{(t)}}(\nabla_{\boldsymbol{M}}^{(t)})$ is indeed an identity mapping on $\nabla_{\boldsymbol{M}}^{(t)}$ in the Euclidean space. More advanced works [25, 38, 46] take extra consideration of the previous update vector $\boldsymbol{P}^{(t-1)}$ to compute the update vector $\boldsymbol{P}^{(t)}$, which can speed up the convergence. For example, in a method using momentum [25], $\boldsymbol{P}^{(t)}$ is computed by

$$\boldsymbol{P}^{(t)} = \eta^{(t)}\pi_{\boldsymbol{M}^{(t)}}(\nabla_{\boldsymbol{M}}^{(t)}) + \gamma^{(t)}\tau_{\boldsymbol{M}^{(t-1)}\rightarrow\boldsymbol{M}^{(t)}}(\boldsymbol{P}^{(t-1)}), \tag{9}$$

where $\gamma^{(t)}$ is a trade-off hyperparameter. In Eq. (9), the update vector $\boldsymbol{P}^{(t)}$ is determined by adding the current gradient $\nabla_{\boldsymbol{M}}^{(t)}$ and the previous update vector $\boldsymbol{P}^{(t-1)}$, where one needs to resort to the complex parallel transport on SPD manifolds: $\tau_{\boldsymbol{M}^{(t-1)} \rightarrow \boldsymbol{M}^{(t)}} : \mathcal{T}_{\boldsymbol{M}^{(t-1)}} \mathcal{S}^{d}_{++} \rightarrow \mathcal{T}_{\boldsymbol{M}^{(t)}} \mathcal{S}^{d}_{++}$ [1]. This operation conveys previous update vector $\boldsymbol{P}^{(t-1)}$ from the previous tangent space $\mathcal{T}_{\boldsymbol{M}^{(t-1)}} \mathcal{S}^{d}_{++}$ to the current tangent space $\mathcal{T}_{\boldsymbol{M}^{(t)}} \mathcal{S}^{d}_{++}$. We stress that the strategy to compute $\boldsymbol{P}^{(t)}$ is task-independent and data agnostic.

In this paper, we propose a meta-learning method to automatically design an optimizer. In particular, we propose to make use of neural network to learn a non-linear function for $\boldsymbol{P}^{(t)}$, which considers both the current gradient and the previous optimization state. That is,

$$\boldsymbol{P}^{(t)} = g_{\phi}(\nabla^{(t)}_{\boldsymbol{M}}, \boldsymbol{S}^{(t-1)}). \tag{10}$$

Here, ϕ is the parameter of the network and the state $\boldsymbol{S}^{(t-1)}$ conveys information about the previous optimization. The optimization framework is shown in Figure 1. Below, we will elaborate on our optimizer and its training procedure.

4.2. Our SPD Optimizer

Several methods have shown that LSTMs are capable of learning to optimize in the Euclidean space [2, 36]. However, these methods cannot be directly applied to SPD manifolds. Gradients of SPD parameters are symmetric matrices. Conventional LSTMs flatten the gradients to vectors, which inevitably destroys the structure of symmetric matrices. As the very first consideration, we should preserve the symmetric property of gradient matrices during optimization. In addition, implementing arithmetic operations like additions of two gradient matrices on SPD manifolds takes more complicated forms than that in the Euclidean space and should be faithful to the geometry of the manifold.

Matrix LSTM. We propose a matrix LSTM (mLSTM) model to preserve the symmetric property of gradient matrices. In particular, we expect to map a symmetric matrix into another symmetric one. To this end, a bilinear projection is used in mLSTM, which has a symmetric form of $\boldsymbol{W}^{\top}\boldsymbol{X}\boldsymbol{W}$ with trainable parameters $\boldsymbol{W}$. The bilinear projection not only requires fewer parameters but also has more robust performance [16]. Our mLSTM is formulated as

$$\begin{cases} \boldsymbol{F}^{(t)} = \sigma(\boldsymbol{W}_{f1}^{\top}\boldsymbol{X}^{(t)}\boldsymbol{W}_{f1} + \boldsymbol{W}_{f2}^{\top}\boldsymbol{H}^{(t-1)}\boldsymbol{W}_{f2}) \\ \boldsymbol{I}^{(t)} = \sigma(\boldsymbol{W}_{i1}^{\top}\boldsymbol{X}^{(t)}\boldsymbol{W}_{i1} + \boldsymbol{W}_{i2}^{\top}\boldsymbol{H}^{(t-1)}\boldsymbol{W}_{i2}) \\ \boldsymbol{O}^{(t)} = \sigma(\boldsymbol{W}_{o1}^{\top}\boldsymbol{X}^{(t)}\boldsymbol{W}_{o1} + \boldsymbol{W}_{o2}^{\top}\boldsymbol{H}^{(t-1)}\boldsymbol{W}_{oh}) \\ \widehat{\boldsymbol{C}^{(t)}} = \tanh(\boldsymbol{W}_{c1}^{\top}\boldsymbol{X}^{(t)}\boldsymbol{W}_{c1} + \boldsymbol{W}_{c2}^{\top}\boldsymbol{H}^{(t-1)}\boldsymbol{W}_{c2}) \\ \boldsymbol{C}^{(t)} = \boldsymbol{F}^{(t)} \otimes \boldsymbol{C}^{(t-1)} + \boldsymbol{I}^{(t)} \otimes \widehat{\boldsymbol{C}^{(t)}} \\ \boldsymbol{H}^{(t)} = \boldsymbol{O}^{(t)} \otimes \tanh(\boldsymbol{C}^{(t)}) \end{cases}, \tag{11}$$

where $\boldsymbol{X}^{(t)} \in sym(d)$ is the input of mLSTM, $\boldsymbol{H}^{(t)} \in sym(d)$ is the hidden state of mLSTM, and $\boldsymbol{C}^{(t)} \in sym(d)$

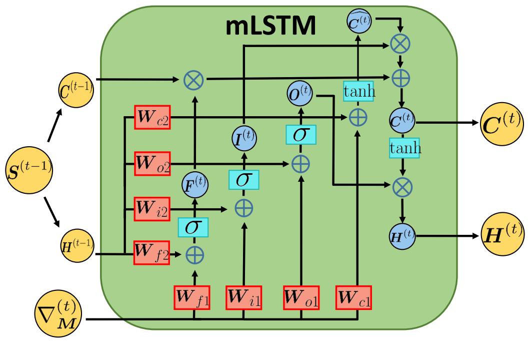

Figure 2. The architecture of the matrix LSTM.

is the memory cell. $\boldsymbol{F}^{(t)}$, $\boldsymbol{I}^{(t)}$, and $\boldsymbol{O}^{(t)}$ are the forget gate, the input gate, and the output gate, respectively. $\boldsymbol{W}_{f1}$, $\boldsymbol{W}_{f2}$, $\boldsymbol{W}_{i1}$, $\boldsymbol{W}_{i2}$, $\boldsymbol{W}_{o1}$, $\boldsymbol{W}_{o2}$, $\boldsymbol{W}_{c1}$, and $\boldsymbol{W}_{c2} \in \mathbb{R}^{d\times d}$ are parameters of mLSTM, and collectively shown by $\gamma = [\boldsymbol{W}_{f1}, \boldsymbol{W}_{f2}, \boldsymbol{W}_{i1}, \boldsymbol{W}_{i2}, \boldsymbol{W}_{o1}, \boldsymbol{W}_{o2}, \boldsymbol{W}_{c1}]$. $\sigma(\cdot)$ is the sigmoid function, $\tanh(\cdot)$ is the hyperbolic tangent function, and $\otimes$ is the Hadamard product. For the convenience, the mLSTM is simplified as

$$\boldsymbol{C}^{(t)}, \boldsymbol{H}^{(t)} = \mathrm{mLSTM}(\boldsymbol{X}^{(t)}, \boldsymbol{S}^{(t-1)}), \tag{12}$$

where $\boldsymbol{S}^{(t-1)} = [\boldsymbol{C}^{(t-1)}, \boldsymbol{H}^{(t-1)}]$ is the state of mLSTM. The architecture of the mLSTM is shown in Figure 2.

Optimizer Architecture. Our optimizer first computes a step-size $\eta^{(t)}$ and a search direction $\boldsymbol{T}^{(t)}$ in $g_{\phi}(\cdot,\cdot)$, followed by the orthogonal projection and retraction to obtain the next solution, *i.e.*, an SPD matrix. Concretely, we employ two mLSTM models, *i.e.*, mLSTM_l and mLSTM_s, to compute $\eta^{(t)}$ and $\boldsymbol{T}^{(t)}$, respectively. We empirically observed that using a skip-connection to fit the residual between the optimal search direction and the input gradient helps the overall design, which is our recommendation for the optimizer. Our optimizer updates the SPD parameter by

$$\begin{cases} \boldsymbol{M}^{(t+1)} = \Gamma_{\boldsymbol{M}^{(t)}}\Big(-\boldsymbol{P}^{(t)}\Big) \\ \boldsymbol{P^{(t)}} = g_{\phi}(\nabla^{(t)}_{\boldsymbol{M}}, \boldsymbol{S}^{(t-1)}) = \eta^{(t)}\boldsymbol{T}^{(t)} \end{cases}, \tag{13}$$

where

$$\begin{cases} \eta^{(t)} = \boldsymbol{w}_l^{\top}\boldsymbol{H}_l^{(t)}\boldsymbol{w}_l \\ \boldsymbol{C}_l^{(t)}, \boldsymbol{H}_l^{(t)} = \mathrm{mLSTM}_l(\nabla^{(t)}_{\boldsymbol{M}}, \boldsymbol{S}^{(t-1)}) \end{cases}, \tag{14}$$

$$\begin{cases} \boldsymbol{T}^{(t)} = \pi_{\boldsymbol{M}^{(t)}}(\boldsymbol{H}_s^{(t)\prime}) \\ \boldsymbol{H}_s^{(t)\prime} = \boldsymbol{W}_s^{\top}(\boldsymbol{H}_s^{(t)} + \nabla^{(t)}_{\boldsymbol{M}})\boldsymbol{W}_s \\ \boldsymbol{C}_s^{(t)}, \boldsymbol{H}_s^{(t)} = \mathrm{mLSTM}_s(\nabla^{(t)}_{\boldsymbol{M}}, \boldsymbol{S}^{(t-1)}) \end{cases}, \tag{15}$$

and $\boldsymbol{S}^{(t-1)} = \boldsymbol{S}_l^{(t-1)} \otimes \boldsymbol{S}_s^{(t-1)}$ is the previous optimization state containing previous optimization information. $\boldsymbol{S}_l^{(t-1)} = [\boldsymbol{C}_l^{(t-1)}, \boldsymbol{H}_l^{(t-1)}]$ is the state of mLSTM_l, and $\boldsymbol{S}_s^{(t-1)} = [\boldsymbol{C}_s^{(t-1)}, \boldsymbol{H}_s^{(t-1)}]$ is the state of mLSTM_s.

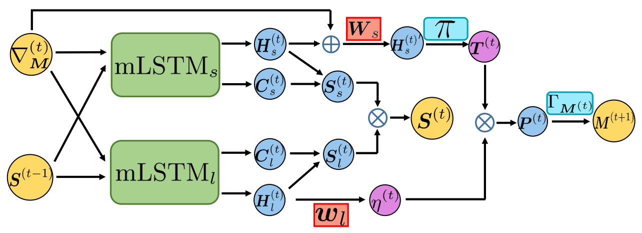

Figure 3. The architecture of our optimizer.

$\boldsymbol{W}_s \in \mathbb{R}^{d\times d}$ and $\boldsymbol{w}_l \in \mathbb{R}^d$ are parameters. The learnable parameter ϕ of the optimizer is $\phi = \{\gamma_l, \gamma_s, \boldsymbol{w}_l, \boldsymbol{W}_s\}$, where γ_l and γ_s are parameters of mLSTM_l and mLSTM_s, respectively. The architecture of the optimizer is shown in Figure 3. Our optimizer can utilize the previous optimization information and guarantee the new parameter on SPD manifolds. Compared with optimizers considering the previous update vector [25, 38, 46], our optimizer does not involve the complex parallel transport operation.

4.3. Training

In the meta-learning process, there are two loops of optimization, as shown in Figure 4. In the inner loop, the base learner is optimized by our optimizer, while ϕ is updated in the outer loop. Our method differentiates the optimization process in the inner loop and utilizes the truncated backpropagation through time algorithm to minimize the objective of the base learner. We consider T continuous steps in the inner loop once, and the meta-objective to learn ϕ is given by

$$\begin{aligned}\min_{\phi} \mathcal{J}(\phi) &= \frac{1}{m}\sum_{t}^{T}\sum_{j}^{m}\mathcal{L}(\boldsymbol{M}_j^{(t+1)}) \\ &= \frac{1}{mn}\sum_{t}^{T}\sum_{j,i}^{m,n} l\left(f\left(\Gamma_{\boldsymbol{M}_j^{(t)}}\left(g_\phi(\nabla_{\boldsymbol{M}_j}^{(t)}, \boldsymbol{S}_j^{(t-1)})\right), \boldsymbol{x}_i\right), \boldsymbol{y}_i\right),\end{aligned} \tag{16}$$

where m is the batchsize in the outer loop (*i.e.*, the meta-objective involves m individual SPD parameters to reduce undesirable training oscillations), and n is the batchsize in the inner loop. Note that forms of $f(\cdot)$ and $l(\cdot)$ differ per task. Due to page limitations, we have placed relevant details of $f(\cdot)$ and $l(\cdot)$ for the tasks considered in our experiments in the *supplementary materials*.

Backpropagation. In the optimizer, the derivatives of $\boldsymbol{P}^{(t)}$ and $\boldsymbol{M}^{(t)}$ with respect to $\boldsymbol{M}^{(t+1)}$ in the retraction operation $\Gamma_{\boldsymbol{M}^{(t)}}(-\boldsymbol{P}^{(t)})$ of Eq. (13) are non-trivial, as the retraction contains the matrix power and exponential operations: ${\boldsymbol{M}^{(t)}}^{\frac{1}{2}}$, ${\boldsymbol{M}^{(t)}}^{\frac{-1}{2}}$, and $\mathrm{expm}\big(-{\boldsymbol{M}^{(t)}}^{\frac{-1}{2}}\boldsymbol{P}^{(t)}{\boldsymbol{M}^{(t)}}^{\frac{-1}{2}}\big)$, making the backpropagation challenging. To solve this issue, we make the following observation.

Consider the eigenvalue decomposition $\boldsymbol{M}^{(t)} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^\top$. Then, the matrix power can be computed by ${\boldsymbol{M}^{(t)}}^{\frac{1}{2}} = \boldsymbol{U}\boldsymbol{\Sigma}^{\frac{1}{2}}\boldsymbol{U}^\top$ and ${\boldsymbol{M}^{(t)}}^{\frac{-1}{2}} = \boldsymbol{U}\boldsymbol{\Sigma}^{\frac{-1}{2}}\boldsymbol{U}^\top$. We denote $\boldsymbol{Q} = -{\boldsymbol{M}^{(t)}}^{\frac{-1}{2}}\boldsymbol{P}^{(t)}{\boldsymbol{M}^{(t)}}^{\frac{-1}{2}}$ and apply the eigenvalue decomposition to $\boldsymbol{Q} = \boldsymbol{U}_Q\boldsymbol{\Sigma}_Q\boldsymbol{U}_Q^\top$. The matrix exponential can be computed by $\mathrm{expm}(\boldsymbol{Q}) = \boldsymbol{U}_Q\exp(\boldsymbol{\Sigma}_Q)\boldsymbol{U}_Q^\top$. Thus, we can rewrite the retraction operation of Eq. (13) as

$$\begin{cases}\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^\top = \boldsymbol{M}^{(t)} \\ \boldsymbol{U}_Q\boldsymbol{\Sigma}_Q\boldsymbol{U}_Q^\top = \boldsymbol{Q} = -\boldsymbol{U}\boldsymbol{\Sigma}^{\frac{-1}{2}}\boldsymbol{U}^\top\boldsymbol{P}^{(t)}\boldsymbol{U}\boldsymbol{\Sigma}^{\frac{-1}{2}}\boldsymbol{U}^\top \\ \boldsymbol{M}^{(t+1)} = \big(\boldsymbol{U}\boldsymbol{\Sigma}^{\frac{1}{2}}\boldsymbol{U}^\top\big)\big(\boldsymbol{U}_Q\exp(\boldsymbol{\Sigma}_Q)\boldsymbol{U}_Q^\top\big)\big(\boldsymbol{U}\boldsymbol{\Sigma}^{\frac{1}{2}}\boldsymbol{U}^\top\big)\end{cases}. \tag{17}$$

In Eq. (17), we just need to take the derivative in eigenvalue decomposition. This derivative has been studied in [20], as shown in Proposition 1. Thus, given $\frac{\partial\mathcal{J}}{\partial\boldsymbol{M}^{(t+1)}}$, we can compute $\frac{\partial\mathcal{J}}{\partial\boldsymbol{P}^{(t)}}$ and $\frac{\partial\mathcal{J}}{\partial\boldsymbol{M}^{(t)}}$. Finally, the derivative $\frac{\partial\mathcal{J}}{\partial\boldsymbol{\phi}}$ can be computed by the chain rule.

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*

$$\frac{\partial\mathcal{J}}{\partial\boldsymbol{M}} = 2\boldsymbol{U}\Big(\boldsymbol{R}^\top \otimes \big(\boldsymbol{U}^\top\frac{\partial\mathcal{J}}{\partial\boldsymbol{U}} + \frac{\partial\mathcal{J}}{\partial\boldsymbol{U}}^\top\boldsymbol{U}\big)\Big)\boldsymbol{U}^\top + \boldsymbol{U}\big(\frac{\partial\mathcal{J}}{\partial\boldsymbol{\Sigma}}\big)_{diag}\boldsymbol{U}^\top, \tag{18}$$

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.

Training Strategy. The training of our optimizer requires data to be independent and identically distributed. However, SPD parameters and optimization states obtained in the inner loop are strongly correlated sequentially. We use the experience replay scheme [30] to address this issue.

Specifically, we divide the training of the optimizer into the observation stage and the learning stage. In the observation stage, we use the optimizer to optimize the base learner, and push the obtained SPD parameter and optimization state $(\boldsymbol{M}^{(t+1)}, \boldsymbol{S}^{(t)})$ into an experience pool Ψ. In the learning stage, for each step of the outer loop, we randomly select $\{(\boldsymbol{M}_j^{(t)}, \boldsymbol{S}_j^{(t-1)})\}_{j=1}^m$ from the experience pool Ψ constituting the training data. After T steps, we push new SPD parameters and optimization states $\{(\boldsymbol{M}_j^{(t+T)}, \boldsymbol{S}_j^{(t+T-1)})\}_{j=1}^m$ into the experience pool Ψ, compute the loss by Eq. (16), and update ϕ by backpropagation. We set a maximum optimization step τ for the base learner. When $t + T > \tau$, $\boldsymbol{M}_j^{(t+T)}$ and $\boldsymbol{S}_j^{(t+T-1)}$ are reset to $\mathbf{I}_d$ and $\mathbf{0}_d$, respectively. The training process of our optimizer is summarized in Algorithm 1.

Computational complexity. In our method, the forward and backward passes mainly involve matrix multiplication, eigenvalue decomposition, and element-wise operations (*e.g.*, Hadamard products and activation functions).

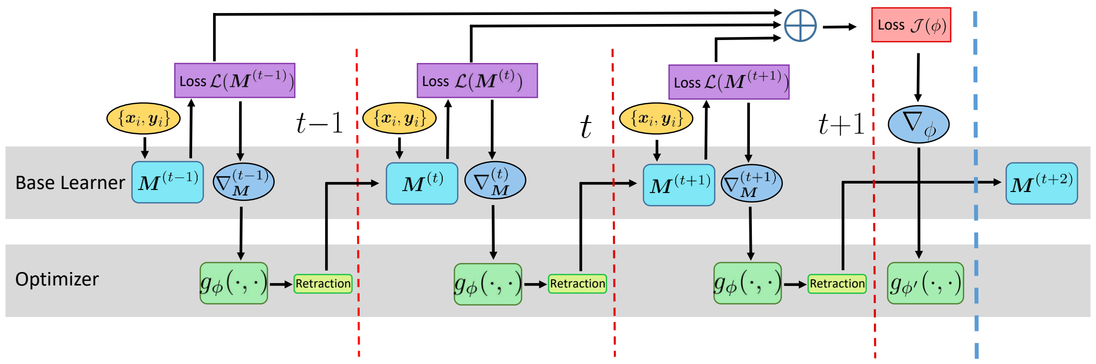

Figure 4. The computational graph of learning our optimizer.

For a $d \times d$ matrix, the computational complexities of matrix multiplication and eigenvalue decomposition are $\mathcal{O}(d^3)$. The element-wise operations require $\mathcal{O}(d^2)$ flops. This leads to $\mathcal{O}(47d^3 + 30d^2 + 3d)$ and $\mathcal{O}(126d^3 + 45d^2 + 3d)$ flops in the forward and backward passes, respectively.

Algorithm 1 Trainging Process of Our Optimizer

```
Input: The randomly initialized optimizer parameter. The initial
       SPD parameter $M^{(0)} = \mathbf{I}_d$. The initial state $S^{(0)} = \mathbf{0}_d$.
       The initial experience pool $\Psi = \emptyset$.
Output: The optimizer parameter $\phi$.
while not reach the maximum iteration of the observation stage do
    Compute the loss of the base learner by Eq. (6);
    Compute the gradient $\nabla_M^{(t)}$;
    Update the parameter $M^{(t+1)}$ by Eq. (13);
    Push $\{M^{(t+1)}, S^{(t)}\}$ into $\Psi$;
end
while not reach the maximum iteration of the learning stage do
    Randomly select $\{(M_j^{(t)}, S_j^{(t-1)})\}_{j=1}^m$ from $\Psi$;
    while not reach $T$ do
        Compute the loss of the base learner by Eq. (6);
        Compute the gradient $\nabla_M^{(t)}$;
        Update $M^{(t+1)}$ by Eq. (13);
    end
    Compute the loss of our optimizer by Eq. (16);
    Update $\phi$ of our optimizer via backpropagation;
    if $t + T > \tau$ then
        Push $\{(M_j^{(0)} = \mathbf{I}_d, S_j^{(0)} = \mathbf{0}_d\}_{j=1}^m$ into $\Psi$;
    else
        Push $\{(M_j^{(t+T)}, S_j^{(t+T-1)})\}_{j=1}^m$ into $\Psi$;
    end
end
Return the parameter $\phi$ of our optimizer.
```

5. Experiments

We learn SPD optimizers on three tasks, *i.e.*, the metric nearness, image clustering, and similarity learning tasks. Inspired by [36], our optimizer is initialized close to RSGD with a small learning rate to help initial stability of training. We learn the optimizer on the given task and data, and then evaluate its performance on seen and unseen data. Our optimizer is compared with state-of-the-art SPD optimizers: RSGD [5], RSGDM [25], RSVRG [46], and RSRG [24]. Following the protocol in [24, 25], we tune hyperparameters of all optimizers to achieve their best performances.

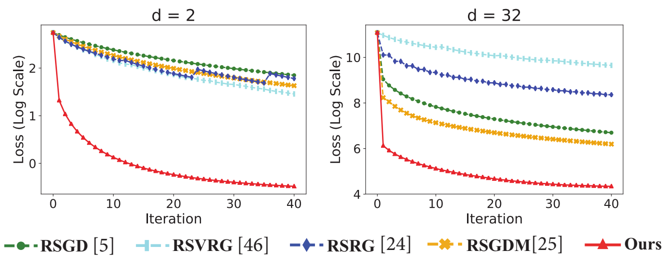

Figure 5. Plots for the metric nearness task with different sizes of the SPD parameter. The size of the SPD parameter is $d \times d$.

5.1. Metric Nearness

We conducted experiments on the metric nearness task [6]. Specifically, our objective of this task is $\mathcal{L}(M) = \frac{1}{n}\sum_i^n \|MAx_i - x_j\|_2^2$, where $x \in \mathbb{R}^d$ is a vector, $A \in \mathcal{S}_{++}^d$ is a given SPD projection matrix, and $M \in \mathcal{S}_{++}^d$ is the parameter to be solved. We expect M can project features Ax_i back to its original value in the metric space.

We randomly generated x_i and A to train and evaluate our optimizer. We set the batchsize n as 256, m as 128, the maximum optimization step τ as 200, and the learning rate as 10^{-3}. We considered $T = 5$ optimization steps in the inner loop. We evaluated optimizers with different dimensions of M: $d = 2$ and $d = 32$. We drew the log scale loss of different optimizers in Figure 5. RSVRG has different performance when $d = 2$ and $d = 32$, showing the hand-designed optimizer may perform differently once the configuration is changed. In contrast, our optimizer achieves good performance on both $d = 2$ and $d = 32$, as it can explore the underlying data distribution and be adaptive to tasks in a data-driven manner. It not only avoids lots of human involvements but also has better performance.

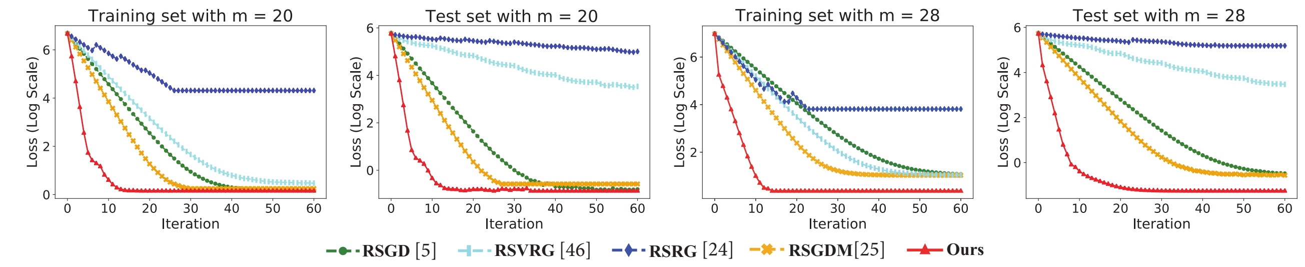

Figure 6. Plots for the clustering task. Our optimizer is trained on the training set and evaluated on both the training and test sets. m denotes the number of centers. On the test set, RSVRG finally converges at 3.54 with $m = 20$ and 3.02 with $m = 28$ (log scale).

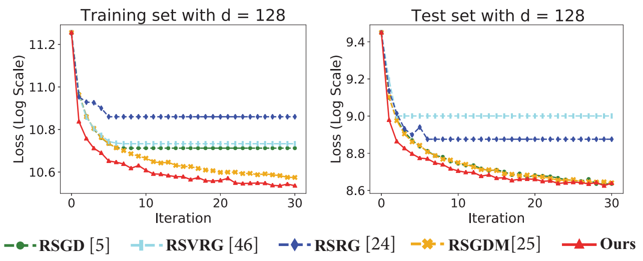

Figure 7. Plots for the similarity learning task on the MNIST dataset. The size of the SPD parameter is $d \times d$.

5.2. Clustering

Clustering is a fundamental task in engineering. Many visual representations lie on SPD manifolds, such as the covariance descriptor, the kernel matrix, and the Diffusion Tensor Image (DTI). Thus, carrying out the clustering task and learning centers with the SPD form is indispensable. We conducted clustering experiments on the Kylberg texture dataset [26] that has 20 major texture categories and 28 fine-grained categories. Following the preprocessing in [25], we resized images to 128×128 and divide the image into 1024 grids whose size was 4×4. At the position (u, v) of each grid, we extracted a 5-D feature vector as $\boldsymbol{x}_{uv} = \left[I_{uv}, |\frac{\partial I}{\partial u}|, |\frac{\partial I}{\partial v}|, |\frac{\partial I^2}{\partial u^2}|, |\frac{\partial I^2}{\partial v^2}|\right]$. Then, a 5×5 covariance descriptor was calculated to represent the image [19, 29, 40]. In our clustering task, we learned centers by minimizing the distances between each center and descriptors belonging to this cluster, where the distances were computed by the affine invariant metric (AIM) [35].

Experiments were conducted on both the 20 major categories and 28 fine-grained categories, and their batchsize m were set as 20 and 28, respectively. The batchsize n was set as 512, and τ was set as 100. We considered $T = 5$ and set the learning rate as 10^{-4}. Our optimizer was trained on the training set and evaluated on both the training and test sets. We drew the log scale loss of optimizers in Figure 6. We can find that, our optimizer can achieve good performance, no matter on $m = 20$, $m = 28$, the training set, or the test set. It converges faster and has a better optima. This shows that our optimizer can achieve good performance not only on the given data but also on the unseen data.

5.3. Similarity Learning

Similarity learning aims to learn a Mahalanobis metric, through which similar samples have small distances and dissimilar samples have large distances. The Mahalanobis metric is usually enforced to be an SPD matrix. We conducted experiments on the MNIST dataset and the CUB dataset [44] and use the contrastive loss to learn the metric. In MNIST, the size of handwritten digital images is 28×28, and we utilized principal component analysis to reduce them to 128-D feature vectors. The CUB dataset is a fine-grained image dataset that contains 200 categories. We added two fully-connected layers into the VGG-16 network, whose output dimensions are 256 and 128. Then, we fine-tuned the VGG-16 and extracted 256-D and 128-D features from the two fully-connected layers as image representations, respectively. We set the batchsize n as 160 and m as 12. The learning rate was set as 10^{-3}. We adopted the standard training and test splits on the two datasets. The optimizer was trained on the training set and evaluated on both the training and test sets. We set the maximum optimization step τ as 200 and considered $T = 5$ optimization steps. Experiments were conducted with different dimensions of SPD parameters. We drew the log scale loss of different optimizers in Figure 7 and Figure 8. It is clear that, our optimizer achieves faster convergence rate and better optima.

5.4. Ablation Study

We conducted ablation experiments on the clustering task to evaluate our mLSTM, the experience replay scheme, and the skip-connection in the optimizer. Concretely, (1) we replaced our mLSTM in the optimizer with the conventional LSTM model; (2) we removed the experience replay scheme and utilized sequential SPD parameters as training data; (3) we removed the skip-connection in the optimizer, and our model directly learns the search direction. Results are shown in Figure 9. Without the skip-connection, learning the optimizer becomes difficult, and the obtained optimizer cannot converge. The conventional LSTM model

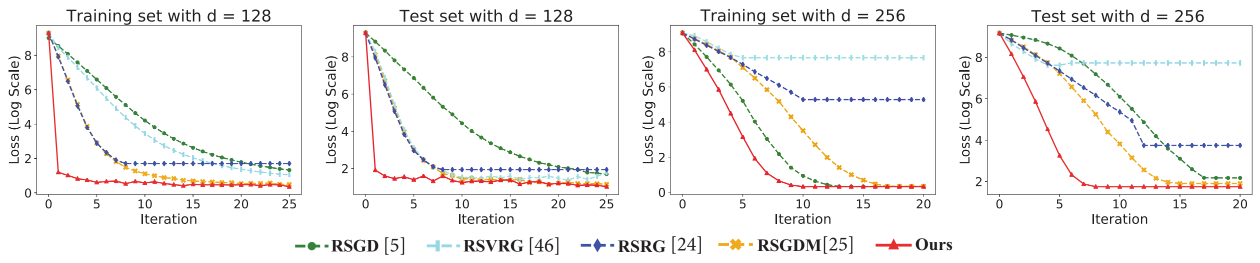

Figure 8. Plots for the similarity learning task on the CUB dataset. The size of the SPD parameter is $d \times d$.

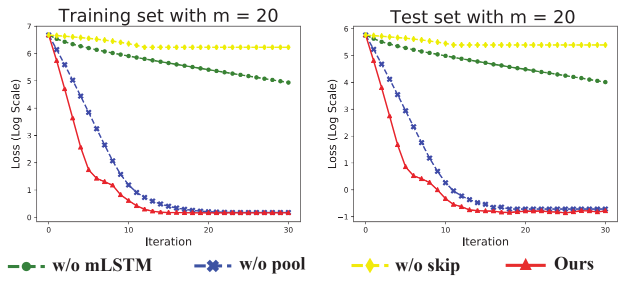

Figure 9. Ablation study on the clustering task. w/o mLSTM means replacing our mLSTM with the conventional LSTM model, w/o pool means removing the experience replay scheme and the experience pool, and w/o skip denotes removing the skip-connection in our optimizer. w/o mLSTM finally converges at 2.47 and 1.39, while ours converges at 0.16 and -0.77 (log scale).

flattens the gradient matrix, which destroys the structure of symmetric matrices. Therefore, the obtained optimizer converges slow and has a bad optima. The experience replay scheme stabilizes our meta-learning process and reduces undesirable training oscillations. The optimizer trained with the experience replay scheme has a faster convergence rate.

5.5. Wall Clock Time

We measured the time of optimization on the Kylberg, MNIST, and CUB datasets, as shown in Table.1. On each dataset, time of different optimizers was measured with the same iteration steps, using Pytorch on a computer with Intel(R) Core(TM) i7-7820X CPU 3.6GHz, GeForce GTX 1080Ti GPU, and 32GB RAM. Our optimizer is faster than RSGDM [25], RSVRG [46], and RSRG [24]. The difference between our optimizer and them is the strategy to compute the update vector $\boldsymbol{P}^{(t)}$. RSGDM, RSVRG, and RSRG utilize the parallel transport to benefit from previous optimization results, which has time-consuming matrix inversion and matrix power operations. In contrast, the mLSTM directly works with the previous optimization state, hence avoiding the expensive parallel transport.

5.6. Evaluation of SPD Parameters

We evaluated the performance of the centers and metric matrices solved by our optimizer in the clustering and similarity learning tasks, respectively. In the clustering task, we regarded the centers as category prototypes and computed AIM between test samples and prototypes to classify test samples. In the similarity learning task, we computed distances between test samples and training samples using the metric matrix and classified test samples by the 1-NN classifier. Classification results are shown in Table 2. We can find our optimizer achieves the best performance. The reason may be that our optimizer can arrive at a better optima, and the base learner can better fit the data.

Table 1. Time (seconds) of optimization on three datasets.

Method	Kylberg	MNIST	CUB
RSGD [5]	367.995	74.980	150.262
RSVRG [46]	449.516	192.443	203.880
RSRG [24]	1248.415	1014.940	1226.610
RSGDM [25]	389.076	90.770	166.578
Ours	378.996	76.757	154.370

Table 2. Performance (%) of SPD parameters.

Method	RSGD [5]	RSVRG [46]	RSRG [24]	RSGDM [25]	Ours
Kylberg	81.25	83.19	70.31	82.92	**84.37**
CUB	70.44	53.74	65.53	70.15	**70.97**

6. Conclusion

In this paper, we have presented a meta-learning method to learn an optimizer on SPD manifolds automatically. The proposed mLSTM can preserve the symmetric property of gradient matrices, and our optimizer can readily implement arithmetic operations on SPD manifolds. We trained the optimizer to minimize the objective of the base learner, and it can effectively explore the underlying data distribution and learn a good optimization trajectory. Experiment results have confirmed that our optimizer is efficient, which has a faster convergence rate and a better optima. In the future, we think the theoretical justification of our method is worth doing, which provides the guarantee of convergence of the learned SPD optimizer on the test data.

Acknowledgements. This work was supported in part by the Natural Science Foundation of China (NSFC) under Grants No. 61702037 and No. 61773062, and Beijing Municipal Natural Science Foundation under Grant No. L172027.

References

[1] P-A Absil, Robert Mahony, and Rodolphe Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009.

[2] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 3981–3989, 2016.

[3] Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V Le. Neural optimizer search with reinforcement learning. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 459–468, 2017.

[4] Samy Bengio, Yoshua Bengio, and Jocelyn Cloutier. On the search for new learning rules for anns. *Neural Processing Letters*, 2(4):26–30, 1995.

[5] Silvere Bonnabel. Stochastic gradient descent on riemannian manifolds. *IEEE Transactions on Automatic Control (T-AC)*, 58(9):2217–2229, 2013.

[6] 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.

[7] Anoop Cherian and Suvrit Sra. Riemannian Dictionary Learning and Sparse Coding for Positive Definite Matrices. *arXiv:1507.02772*, 2015.

[8] Anoop Cherian, Panagiotis Stanitsas, Mehrtash Harandi, Vassilios Morellas, and Nikolaos Papanikolopoulos. Learning discriminative ab-divergences for positive definite matrices. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[9] Melih Engin, Lei Wang, Luping Zhou, and Xinwang Liu. Deepkspd: Learning kernel-matrix-based spd representation for fine-grained image recognition. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[10] Pengfei Fang, Jieming Zhou, Soumava Kumar Roy, Lars Petersson, and Mehrtash Harandi. Bilinear attention networks for person retrieval. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[11] Maurice Fréchet. Les éléments aléatoires de nature quelconque dans un espace distancié. In *Annales de l'institut Henri Poincaré*, volume 10, pages 215–310, 1948.

[12] Zhi Gao, Yuwei Wu, Xingyuan Bu, Tan Yu, Junsong Yuan, and Yunde Jia. Learning a robust representation via a deep network on symmetric positive definite manifolds. *Pattern Recognition (PR)*, 92:1–12, 2019.

[13] Zhi Gao, Yuwei Wu, Mehrtash Harandi, and Yunde Jia. A robust distance measure for similarity-based classification on the spd manifold. *IEEE Transactions on Neural Networks and Learning Systems (T-NNLS)*, 2019.

[14] Zilin Gao, Jiangtao Xie, Qilong Wang, and Peihua Li. Global second-order pooling convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[15] Mehrtash Harandi and Mathieu Salzmann. Riemannian coding and dictionary learning: Kernels to the rescue. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3926–3935, 2015.

[16] Mehrtash T. Harandi, Mathieu Salzmann, and Richard Hartley. From manifold to manifold: Geometry-aware dimensionality reduction for spd matrices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, 2014.

[17] Sepp Hochreiter, A Steven Younger, and Peter R Conwell. Learning to learn using gradient descent. In *Proceedings of the International Conference on Artificial Neural Networks (ICANN)*, pages 87–94, 2001.

[18] Reshad Hosseini and Suvrit Sra. An alternative to em for gaussian mixture models: Batch and stochastic riemannian optimization. *Mathematical Programming*, pages 1–37, 2019.

[19] 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.

[20] Catalin Ionescu, Orestis Vantzos, and Cristian Sminchisescu. Matrix backpropagation for deep networks with structured layers. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 2965–2973, 2015.

[21] Hermann Karcher. Riemannian center of mass and mollifier smoothing. *Communications on pure and applied mathematics*, 30(5):509–541, 1977.

[22] Leonid Karlinsky, Joseph Shtok, Sivan Harary, Eli Schwartz, Amit Aides, Rogerio Feris, Raja Giryes, and Alex M. Bronstein. Repmet: Representative-based metric learning for classification and few-shot object detection. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[23] Hiroyuki Kasai, Pratik Jawanpuria, and Bamdev Mishra. Riemannian adaptive stochastic gradient algorithms on matrix manifolds. pages 3262–3271, 2019.

[24] Hiroyuki Kasai, Hiroyuki Sato, and Bamdev Mishra. Riemannian stochastic recursive gradient algorithm. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2521–2529, 2018.

[25] Soumava Kumar Roy, Zakaria Mhammedi, and Mehrtash Harandi. Geometry aware constrained optimization techniques for deep learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4460–4469, 2018.

[26] Gustaf Kylberg. *Kylberg Texture Dataset v. 1.0*. Centre for Image Analysis, Swedish University of Agricultural Sciences and , 2011.

[27] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[28] Ke Li and Jitendra Malik. Learning to optimize. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2017.

[29] 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.

[30] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[31] David G Luenberger. The gradient projection method along geodesics. *Management Science*, 18(11):620–631, 1972.

[32] Mohamed Maher and Sherif Sakr. Smartml: A meta learning-based framework for automated selection and hyperparameter tuning for machine learning algorithms. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*, 2019.

[33] Lam M Nguyen, Jie Liu, Katya Scheinberg, and Martin Takáč. Sarah: A novel method for machine learning problems using stochastic recursive gradient. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 2613–2621, 2017.

[34] Hyun Oh Song, Yu Xiang, Stefanie Jegelka, and Silvio Savarese. Deep metric learning via lifted structured feature embedding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4004–4012, 2016.

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

[36] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.

[37] Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2019.

[38] Hiroyuki Sato, Hiroyuki Kasai, and Bamdev Mishra. Riemannian stochastic variance reduced gradient algorithm with retraction and vector transport. *SIAM Journal on Optimization*, 29(2):1444–1472, 2019.

[39] Jürgen Schmidhuber. A neural network that embeds its own meta-levels. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 407–412, 1993.

[40] 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.

[41] Qilong Wang, Peihua Li, Qinghua Hu, Pengfei Zhu, and Wangmeng Zuo. Deep global generalized gaussian networks. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[42] Wen Wang, Ruiping Wang, Zhiwu Huang, Shiguang Shan, and Xilin Chen. Discriminant analysis on riemannian manifold of gaussian distributions for face recognition with image sets. *IEEE Transactions on Image Processing (T-IP)*, 27(1):151–163, 2018.

[43] Olga Wichrowska, Niru Maheswaranathan, Matthew W Hoffman, Sergio Gomez Colmenarejo, Misha Denil, Nando de Freitas, and Jascha Sohl-Dickstein. Learned optimizers that scale and generalize. In *Proceedings of the International Conference on Machine Learning (ICML)*, pages 3751–3760, 2017.

[44] Lingxi Xie, Qi Tian, Richang Hong, Shuicheng Yan, and Bo Zhang. Hierarchical part matching for fine-grained visual categorization. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pages 1641–1648, 2013.

[45] Xingyu Xie, Jianlong Wu, Zhisheng Zhong, Guangcan Liu, and Zhouchen Lin. Differentiable linearized admm. 2019.

[46] Hongyi Zhang, Sashank J Reddi, and Suvrit Sra. Riemannian svrg: Fast stochastic optimization on riemannian manifolds. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 4592–4600, 2016.