Learning to Optimize on Riemannian Manifolds

Zhi Gao, *Member, IEEE*, Yuwei Wu, *Member, IEEE*, Xiaomeng Fan, Mehrtash Harandi, *Member, IEEE*, and Yunde Jia, *Member, IEEE*

Abstract—Many learning tasks are modeled as optimization problems with nonlinear constraints, such as principal component analysis and fitting a Gaussian mixture model. A popular way to solve such problems is resorting to Riemannian optimization algorithms, which yet heavily rely on both human involvement and expert knowledge about Riemannian manifolds. In this paper, we propose a Riemannian meta-optimization method to automatically learn a Riemannian optimizer. We parameterize the Riemannian optimizer by a novel recurrent network and utilize Riemannian operations to ensure that our method is faithful to the geometry of manifolds. The proposed method explores the distribution of the underlying data by minimizing the objective of updated parameters, and hence is capable of learning task-specific optimizations. We introduce a Riemannian implicit differentiation training scheme to achieve efficient training in terms of numerical stability and computational cost. Unlike conventional meta-optimization training schemes that need to differentiate through the whole optimization trajectory, our training scheme is only related to the final two optimization steps. In this way, our training scheme avoids the exploding gradient problem, and significantly reduces the computational load and memory footprint. We discuss experimental results across various constrained problems, including principal component analysis on Grassmann manifolds, face recognition, person re-identification, and texture image classification on Stiefel manifolds, clustering and similarity learning on symmetric positive definite manifolds, and few-shot learning on hyperbolic manifolds.

Index Terms—Riemannian optimization, meta-optimization, meta-learning, Riemannian manifolds

1 Introduction

Optimization techniques are pivotal to the success of the machine learning community [1]. A large body of efforts have been targeted at designing powerful optimization algorithms, where gradient-based algorithms are witnessed tremendous progress [2], [3]. In practice, numerous tasks are modeled as optimization problems with nonlinear constraints. For example, similarity learning [4], [5], kernel-based processing [6], learning Gaussian mixture model [7], and data summarization and representation [8], [9], [10] can be modeled with symmetric positive definite (SPD) matrix constraints. Principal component analysis (PCA) [11], independent component analysis (ICA) [12], subspace learning [13], [14], and matrix completion [15] are cast as optimization problems with orthogonality constraints. Many deep models benefit from orthogonality constraints as well [16]. A popular way to incorporate nonlinear constraints into optimization frameworks is to make use of Riemannian geometry and formulate constrained problems as optimization on *Riemannian manifolds* [17].

Nonlinear constraints make solving optimization problems challenging due to requirements for preserving the Riemannian geometry [18]. Optimization algorithms designed in euclidean spaces cannot be directly applied to such optimization problems, because euclidean optimizers do not comply with the Riemannian geometry and will destroy the nonlinear constraints. To address this issue, one needs to resort to gradient-based Riemannian optimization algorithms [19]. Their optimizers view optimization problems with constraints as unconstrained problems on Riemannian manifolds and utilize Riemannian operations to move along manifolds in the quest for solution.

Previous endeavors were directed towards designing Riemannian optimizers by hand. Typical examples include Riemannian stochastic gradient descent (RSGD) [20], Riemannian variance reduction methods [21], [22], [23], Riemannian adaptive optimization [24], [25], and Riemannian accelerated algorithms [26], [27], [28]. In these Riemannian optimization methods, the optimizer design, performance evaluation, and optimization scheme update require human involvement and expert knowledge about manifolds to achieve a satisfactory result. Besides, existing Riemannian optimizers are task-agnostic. Since underlying data distribution differs among various machine learning tasks [29], [30], a good optimizer should be tailored to the specific task,

- *Zhi Gao and Xiaomeng Fan are with the Beijing Lab of Intelligent Information Technology, School of Computer Science, Beijing Institute of Technology, Beijing 100081, China. E-mail: {gaozhi_2017, fanxiaomeng}@bit.edu.cn.*
- *Yuwei Wu is with the Beijing Lab of Intelligent Information Technology, School of Computer Science, Beijing Institute of Technology, Beijing 100081, China, and also with the Guangdong Lab of Machine Perception and Intelligent Computing, Shenzhen MSU-BIT University, Shenzhen, Guangdong Province 518172, China. E-mail: wuyuwei@bit.edu.cn.*
- *Mehrtash Harandi is with the Department of Electrical and Computer Systems Eng., Monash University, Melbourne, VIC 3800, Australia, and also with Data61-CSIRO, Clayton South, VIC 3169, Australia. E-mail: mehrtash.harandi@monash.edu.*
- *Yunde Jia is with the Guangdong Lab of Machine Perception and Intelligent Computing, Shenzhen MSU-BIT University, Shenzhen, Guangdong Province 518172, China, and also with Beijing Lab of Intelligent Information Technology, School of Computer Science, Beijing Institute of Technology, Beijing 100081, China. E-mail: jiayunde@bit.edu.cn.*

Manuscript received 9 January 2022; revised 16 August 2022; accepted 4 October 2022. Date of publication 19 October 2022; date of current version 3 April 2023.
This work was supported by the Natural Science Foundation of China (NSFC) under Grants 62172041 and 62176021.
(Corresponding author: Yuwei Wu.)
Recommended for acceptance by C. Zhang.
Digital Object Identifier no. 10.1109/TPAMI.2022.3215702

such as tuning hyperparameters. This heavily relies on expert knowledge and human involvement as well.

In this article, we propose a Riemannian meta-optimization (RMO) method to automatically learn a Riemannian optimizer, reducing human involvement in employing the Riemannian optimizer. Our work, inspired by progresses in meta-optimization [31], [32], offers a new perspective to optimization methods on Riemannian manifolds. Our underlying idea is to parameterize the Riemannian optimizer by a learnable meta-model and train the optimizer in a meta-learning framework, *to learn to optimize* via a data-driven manner. To this end, the following two challenges need to be handled.

- Making the meta-model faithful to Riemannian geometry is challenging. Recurrent neural networks are commonly used meta-models in existing meta-optimization methods that are designed for the euclidean setup [33], [34], [35]. As the Riemannian geometry is far more complicated than euclidean counterpart, directly using existing meta-models to perform optimization on Riemannian manifolds cannot preserve the geometry of Riemannian parameters and will destroy the nonlinear constraints, leading to inferior results.
- Training a Riemannian optimizer is challenging. Training an optimizer is usually cast as a bi-level optimization problem, i.e., an inner-loop and an outer-loop. The optimizer updates the target model in the inner-loop, and updates itself in the outer-loop. In the outer-loop, meta-gradients with respect to the optimizer are computed by differentiating through the optimization trajectory in the whole inner-loop. This causes heavy memory footprint to store the computational graph of the optimization trajectory, and involves a series of time-consuming Hessian matrices and derivatives of Riemannian operations. In addition, the meta-gradients are based on the product of these derivatives, which in turn leads to numerical instabilities, that is, exploding gradients.

To address the first challenge, we propose a novel recurrent network, namely generalized matrix Long Short-Term Memory (gmLSTM), as the meta-model to parameterize the Riemannian optimizer. By considering the structure of Riemannian parameters and utilizing Riemannian operations in gmLSTM, the optimizer complies with geometry of various Riemannian manifolds. We train the optimizer to minimize the objective of Riemannian parameters, through which the optimizer can explore the underlying data distribution and perform task-specific optimization in a data-driven manner. To address the second challenge, we introduce a Riemannian implicit differentiation training scheme, where the meta-gradients are dependent on the final two optimization steps, instead of the whole optimization trajectory. In this case, we sidestep computing the products of Hessian and gradients of retraction operations in meta-gradients, avoiding the exploding gradients. We demonstrate theoretically and empirically that our method only needs small memory and computational costs, regardless of the length of the optimization trajectory in the inner-loop.

In summary, our main contributions are three-fold.

1. We propose a Riemannian meta-optimization (RMO) method. To the best of our knowledge, it is the first method to automatically learn a Riemannian optimizer, thus minimizing human involvement in employing the Riemannian optimizer. RMO can be readily applied to various manifolds, such as Stiefel, Grassmann, SPD, and hyperbolic manifolds.
2. We develop a gmLSTM model to parameterize the Riemannian optimizer, which is capable of preserving matrix structures of Riemannian parameters. In this case, RMO is faithful to geometry of various manifolds and enables us to learn the optimizer in an end-to-end fashion.
3. We derive a Riemannian implicit differentiation training scheme that makes the training process efficient and stable. It avoids exploding gradient problems and significantly reduces both time and memory consumption.

In our previous work [36], we presented learning an optimizer on the SPD manifold, which is just one of various types of Riemannian manifolds. In this article, we extend the work to other Riemannian manifolds. **(1)** We extend our work from the SPD manifold to various manifolds, such as Stiefel, Grassmann, and hyperbolic manifolds, making a more generic method. To this end, we develop a novel recurrent network, gmLSTM, to comply with various Riemannian geometries, and provide exploration on four commonly used manifolds. **(2)** We theoretically analyze the training process of learning the Riemannian optimizer, from which we conclude that training the Riemannian optimizer may suffer from exploding gradients, and heavy computational load and memory footprint. To address this issue, we propose a Riemannian implicit differentiation training scheme. **(3)** We apply our method to various machine learning and computer vision tasks, including clustering, similarity learning, dimensionality reduction, face recognition, few-shot learning, person re-identification, and texture image classification tasks on SPD, Grassmann, Stiefel, and hyperbolic manifolds. The code is available at https://github.com/ZhiGaomcislab/learningriemannianoptimization.

2 Related Work

2.1 Optimization on Riemannian Manifolds

Gradient-based Riemannian optimization has attracted much attention in solving optimization problems with nonlinear constraints [19]. Luenberger [37] presented the first gradient descent method on Riemannian manifolds, where Riemannian operations are used to ensure that optimization remains faithful to manifolds. Bonnabel [20] developed the Riemannian stochastic gradient descent (RSGD) algorithm, solving the heavy computational burden of the Riemannian gradient descent. Along the way, many works generalize optimization algorithms from euclidean spaces to Riemannian spaces, to improve the convergence of RSGD.

Some works developed acceleration or momentum techniques for Riemannian optimization. Liu et al. [28] presented an acceleration algorithm for Riemannian optimization, while it involves an exact solution of a nonlinear equation at each iteration. Zhang and Sra [26] proposed a computationally tractable accelerated algorithm to solve this issue, and

more importantly, provided the convergence analysis of the accelerated algorithm. Alimisis et al. [27] presented a new accelerated algorithm, capable of achieving acceleration with a weaker form of convexity. Roy et al. [22] proposed an efficient momentum Riemannian stochastic gradient descent (RSGDM) algorithm that considers previous updates to speed up optimization and can be directly applied to Riemannian optimization problems.

Several works studied reducing variance of stochastic Riemannian gradients. Zhang et al. [21] extended the stochastic variance reduced gradient from euclidean spaces to Riemannian settings, while it has the computationally expensive parallel translation operation. Kasai et al. [38] addressed this challenging issue by proposing an efficient vector transport operation. Zhang et al. [39] proposed an unbiased Riemannian gradient estimator that makes the variance reduced stochastic gradient algorithm converge faster. In addition, Roy et al. [22], Kasai et al. [25], and Bécigneul and Ganea [24] proposed Riemannian adaptive optimization algorithms that achieve remarkable improvements in Riemannian optimization.

In addition to the above first-order optimization methods, Riemannian second-order optimization has also attracted much attention. Popular methods include the Riemannian Newton method [19], the Riemannian trust-region method [40], and the Riemannian Broyden–Fletcher–Goldfarb–Shanno (BFGS) method [41]. Recently, some efforts were made for more efficient Riemannian second-order optimization. Kasai et al. [42] presented a stochastic quasi-Newton method to reduce the expensive computational load by using the limited BFGS (L-BFGS). Kasai and Mishra [43] further developed an inexact Riemannian trust-region algorithm for faster convergence rate, where gradients and Hessian matrices are computed by a random sampling technique. Hu et al. [44] proposed a regularized Newton method that approximates the objective function with a sequence of quadratic subproblems, achieving locally superlinear convergence rate.

The above Riemannian optimizers are all designed by hand. Compared with them, we propose a Riemannian meta-optimization method to automatically learn a Riemannian optimizer, reducing requirements for human involvement and expert knowledge. In addition, we use the data-driven manner to train the optimizer, through which the optimizer can effectively explore underlying data distribution and perform task-specific optimization.

2.2 Meta-Optimization

Meta-optimization uses a meta-learning framework to learn trainable components for optimization. Trainable components can be key hyperparameters of optimization processes [45], recurrent models [33], reinforcement learning models [46], or parameters of traditional optimization algorithms [47], [48], [49].

Our method belongs to methods of training recurrent models. Existing methods take gradients as inputs and generate updated parameters in euclidean spaces. Andrychowicz et al. [33] proposed the pioneering work that leverages an RNN as an optimizer and train the optimizer to minimize the loss of a base-learner. Ravi and Larochelle [34] utilized the LSTM structure for gradient descent, achieving good performance on few-shot learning tasks. Wichrowska et al. [50] introduced hierarchical recurrent models as optimizers that enables the optimization algorithm generalize well to new tasks. Metz et al. [51] dynamically weighted two gradient estimators to compute unbiased meta-gradients, improving the performance of the optimizer. Chem et al. [52] studied several techniques for efficient training of RNN-based optimizer.

In contrast to the above studies, our work aims to learn to optimize on Riemannian manifolds and is unprecedented in previous studies. Considering the complexity of Riemannian geometry, existing meta-optimization methods fail to generalize faithfully to learn a Riemannian optimizer. For example, a simple extension of RNN-based optimizers may harm the structure of Riemannian parameters, failing to preserve nonlinear constraints. Besides, direct using of conventional training schemes such as the truncated backpropagation through time algorithm to learn the Riemannian optimizer, as will be shown, may result in the exploding gradient issue, and heavy computational load and memory footprint. In this article, we take the very first step in meta-learning for Riemannian optimization.

2.3 Implicit Differentiation

Implicit differentiation has been well applied to bi-level optimization problems. By utilizing the implicit function theorem [53], implicit differentiation is used to compute the gradient with respect to the outer-loop parameters, avoiding differentiating through the inner-loop optimization. Rajeswaran et al. [54] utilized the implicit differentiation to compute the gradient with respect to an initial model. Liao et al. [55] and Gudovskiy et al. [56] utilized implicit differentiation for hyperparameter optimization. Despite the success, implicit differentiation has not yet been applied to meta-optimization. In our method, we derive and prove the first implicit differentiation for meta-optimization by using the final two iterations in the inner-loop to compute the meta-gradients. Besides, we extend the implicit differentiation to Riemannian manifolds, through which we can efficiently solve challenging Riemannian optimization problems.

3 Mathematical Background

We use $\mathbf{I}_p$ to denote the $p \times p$ identity matrix and $\mathbf{0}_p$ to denote a matrix whose elements are all '0's.

3.1 Riemannian Manifolds

A smooth Riemannian manifold $\mathcal{M}$ is a locally euclidean space and can be understood as a generalization of the notion of a surface to higher dimensional spaces [17]. A euclidean space is a special case of Riemannian manifolds. For a point $\boldsymbol{M} \in \mathcal{M}$, the tangent space is denoted by $T_{\boldsymbol{M}}\mathcal{M}$. The tangent space is a euclidean space and contains all vectors that are tangent to $\mathcal{M}$ at $\boldsymbol{M}$. In this article, we take four popular Riemannian manifolds, namely hyperbolic, Stiefel, Grassmann, and SPD manifolds as examples to show our method.

Hyperbolic Manifolds. A hyperbolic manifold $\mathcal{H}^d$ is a smooth Riemannian manifold with a constant negative curvature [57]. There are five isometric models of hyperbolic spaces [58], and we choose the Poincaré ball model to work with. It is defined as $\mathcal{H}^d = \{\boldsymbol{x} \in \mathbb{R}^d, \|\boldsymbol{x}\| < 1\}$ with the curvature being -1.

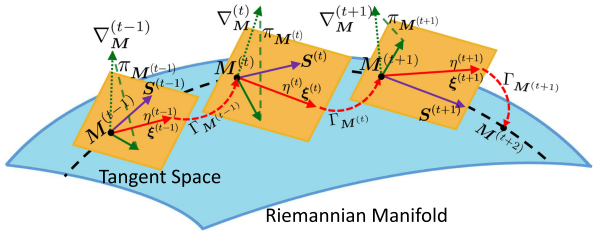


Fig. 1. The illustration of optimization via our Riemannian optimizer. The black dash line shows a geodesic. The green dotted arrows denote arbitrary euclidean gradients $\nabla_{\boldsymbol{M}}^{(t)}$. Green solid arrows, purple solid arrows, and red solid arrows denote Riemannian gradients, previous optimization state $\boldsymbol{S}^{(t)}$, and update $\eta^{(t)}\boldsymbol{\xi}^{(t)}$, respectively. Red dash arrows indicate the retraction $\Gamma_{\boldsymbol{M}^{(t)}}$, and green dash lines denote orthogonal projection $\pi_{\boldsymbol{M}^{(t)}}$. Based on the previous optimization state and the current gradient, the optimizer generates a step size and a search direction, and obtains an updated parameter by the retraction operation.

Stiefel Manifolds. A Stiefel manifold $\mathrm{St}(p,d)$ denotes a set of all $d \times p$ matrices with orthonormal columns [59],

$$\mathrm{St}(p,d) = \{\boldsymbol{M} \in \mathbb{R}^{d\times p} : \boldsymbol{M}^\top \boldsymbol{M} = \mathbf{I}_p\}, \tag{1}$$

where $p < d$ and $\top$ is a transpose operation.

Grassmann Manifolds. A Grassmann manifold $\mathcal{G}(p,d)$ represents a set of all p-dimensional subspaces in $\mathbb{R}^d$ [60]. It is formally defined as the quotient of the Stiefel manifold $\mathrm{St}(p,d)$ through the following equivalence class

$$[\boldsymbol{M}] = \{\boldsymbol{M}\boldsymbol{R} : \boldsymbol{M} \in \mathrm{St}(p,d), \boldsymbol{R} \in \mathcal{O}(p)\}. \tag{2}$$

Here, $\mathcal{O}(p)$ denotes the orthonormal group, i.e., $\boldsymbol{R} \in \mathbb{R}^{p\times p}$ with $\boldsymbol{R}^\top\boldsymbol{R} = \boldsymbol{R}\boldsymbol{R}^\top = \mathbf{I}_p$.

SPD Manifolds. A square matrix $\boldsymbol{M} \in \mathbb{R}^{d\times d}$ satisfying $\boldsymbol{M} = \boldsymbol{M}^\top$ and $\boldsymbol{v}^\top\boldsymbol{M}\boldsymbol{v} > 0, \forall \boldsymbol{v} \in \mathbb{R}^d\backslash\{\mathbf{0}\}$ characterizes an SPD matrix [61]. An SPD manifold Sym_d^+ is consequently identified as a set of all $d \times d$ SPD matrices.

3.2 Riemannian Gradient Descent

In practice, a large body of machine learning problems make use of nonlinear constraints that correspond to Riemannian manifolds. The constraints can be used to inject inductive bias, preserve physical properties of the problem, or lead to better performance, to name a few. Such constrained optimization problems can be formulated as

$$\min_{\boldsymbol{M}\in\mathcal{M}} \mathcal{L}(\boldsymbol{M}) \triangleq \frac{1}{n}\sum_{i=1}^{n} f(\boldsymbol{M}, \boldsymbol{x}_i, \boldsymbol{y}_i). \tag{3}$$

Here, $f : \mathcal{M}\times\mathcal{X}\times\mathcal{Y} \to \mathbb{R}$ is a loss function defined on a Riemannian manifold $\mathcal{M}$. Well-known gradient-based optimizers in euclidean spaces *cannot* be directly applied to the problem in Eq. (3), due to the non-euclidean geometry of $\mathcal{M}$.

One popular choice is to resort to gradient-based Riemannian optimization algorithms that exploit geometry of manifolds by making using of several Riemannian operations.

Orthogonal Projection. The orthogonal projection $\pi_{\boldsymbol{M}}(\nabla_{\boldsymbol{M}}) : \mathbb{R}^n \to T_{\boldsymbol{M}}\mathcal{M}$ transforms a euclidean gradient $\nabla_{\boldsymbol{M}}$ at point $\boldsymbol{M}$ into a Riemannian gradient. The Riemannian gradient is a vector sitting on the tangent space $T_{\boldsymbol{M}}\mathcal{M}$, and $\boldsymbol{M}$ is a point on the manifold $\mathcal{M}$.

Parallel Transport. The parallel transport operation $\Upsilon_{\boldsymbol{M}\to\boldsymbol{M}'}(\boldsymbol{P}) : T_{\boldsymbol{M}}\mathcal{M} \to T_{\boldsymbol{M}'}\mathcal{M}$ moves vectors from a tangent space $T_{\boldsymbol{M}}\mathcal{M}$ to another tangent space $T_{\boldsymbol{M}'}\mathcal{M}$ along the geodesic curve. $\boldsymbol{M}$ and $\boldsymbol{M}'$ are points on the Riemannian manifold $\mathcal{M}$, and $\boldsymbol{P}$ is a vector on the tangent space $T_{\boldsymbol{M}}\mathcal{M}$.

Retraction. $\Gamma_{\boldsymbol{M}}(\boldsymbol{P}) : T_{\boldsymbol{M}}\mathcal{M} \to \mathcal{M}$ is a smooth mapping from a tangent space $T_{\boldsymbol{M}}\mathcal{M}$ to the manifold $\mathcal{M}$ with a local rigidity condition [19], and $\boldsymbol{P}$ is a point on the tangent space $T_{\boldsymbol{M}}\mathcal{M}$. After obtaining a Riemannian gradient on a tangent space, the retraction operation is applied to find the updated Riemannian parameter on manifolds.

Based on above operations, $\boldsymbol{M}$ in Eq. (3) is updated by

$$\boldsymbol{M}^{(t+1)} = \Gamma_{\boldsymbol{M}^{(t)}}(-\eta^{(t)}\boldsymbol{\xi}^{(t)}), \tag{4}$$

where $\eta^{(t)}$ is the step size and $\boldsymbol{\xi}^{(t)}$ is the search direction on the tangent space. In Riemannian gradient descent algorithms, search directions are computed by the orthogonal projection operation $\boldsymbol{\xi}^{(t)} = \pi_{\boldsymbol{M}^{(t)}}(\nabla_{\boldsymbol{M}}^{(t)})$, and $\nabla_{\boldsymbol{M}}^{(t)}$ is a euclidean gradient in an ambient space. Some methods [21], [22], [23] take extra considerations of previous search directions $\boldsymbol{\xi}^{(t-1)}$ to speed up the convergence. For example, in a momentum-based Riemannian optimization method [22], the search direction $\boldsymbol{\xi}^{(t)}$ is computed by

$$\boldsymbol{\xi}^{(t)} = \pi_{\boldsymbol{M}^{(t)}}(\nabla_{\boldsymbol{M}}^{(t)}) + \lambda^{(t)}\Upsilon_{\boldsymbol{M}^{(t-1)}\to\boldsymbol{M}^{(t)}}(\boldsymbol{\xi}^{(t-1)}), \tag{5}$$

where $\lambda^{(t)}$ is a trade-off hyperparameter. We stress that existing Riemannian optimizers require $\eta^{(t)}$ to be carefully chosen, and schemes to compute $\boldsymbol{\xi}^{(t)}$ are hand-designed.

4 Riemannian Meta-Optimization Method

4.1 Problem Definition

euclidean meta-optimization methods learn to optimize via a neural network $g_\theta(\cdot)$ that takes the gradient $\nabla_\Theta^{(t)}$ as input and outputs an update vector. This leads to the update rule to the euclidean parameter Θ,

$$\Theta^{(t+1)} = \Theta^{(t)} - g_\theta(\nabla_\Theta^{(t)}). \tag{6}$$

In this article, we propose a Riemannian meta-optimization (RMO) method to automatically learn a Riemannian optimizer. We use recurrent neural networks to parameterize a Riemannian optimizer,

$$\boldsymbol{M}^{(t+1)} = \Gamma_{\boldsymbol{M}^{(t)}}\Big(-g_{\phi_1}(\nabla_{\boldsymbol{M}}^{(t)}, \boldsymbol{S}^{(t-1)}) \cdot g_{\phi_2}(\nabla_{\boldsymbol{M}}^{(t)}, \boldsymbol{S}^{(t-1)})\Big), \tag{7}$$

where $\boldsymbol{M}^{(t)}$ is the Riemannian parameter. $g_{\phi_1}(\cdot)$ and $g_{\phi_2}(\cdot)$ produce the step size η and search direction $\boldsymbol{\xi}$, respectively,

$$\begin{cases} \eta^{(t)} = g_{\phi_1}\left(\nabla_{\boldsymbol{M}}^{(t)}, \boldsymbol{S}^{(t-1)}\right) \\ \boldsymbol{\xi}^{(t)} = g_{\phi_2}\left(\nabla_{\boldsymbol{M}}^{(t)}, \boldsymbol{S}^{(t-1)}\right) \end{cases}. \tag{8}$$

ϕ_1 and ϕ_2 are parameters of the recurrent neural networks, and $\boldsymbol{S}^{(t-1)}$ is the optimization state at time $t-1$ generated by the optimizer. The optimization process is shown in Fig. 1.

4.2 Generalized-Matrix-LSTM-Based Optimizer

Several meta-optimization methods show that optimizers parameterized by LSTMs can achieve good performances in euclidean spaces [33], [34], [50], [62]. We note that existing LSTMs cannot be directly applied to a Riemannian optimizer. The gradients $\nabla_{\boldsymbol{M}}^{(t)}$ have matrix structures (e.g., gradients of SPD parameters are symmetric

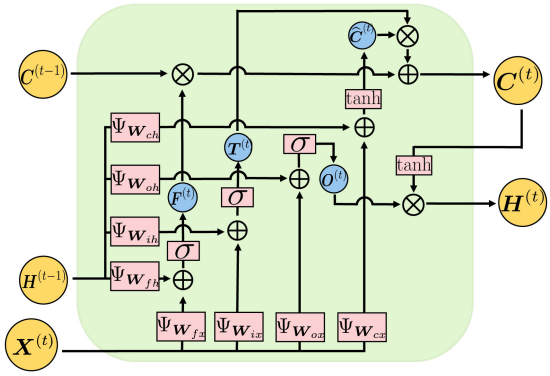


Fig. 2. The architecture of the gmLSTM. $\boldsymbol{X}^{(t)}$ is the input, $\boldsymbol{C}^{(t)}$ is the memory cell, and $\boldsymbol{H}^{(t)}$ is the output at time t. $\Psi_{\boldsymbol{W}}$ is our transformation, σ is the nonlinear activation, $\oplus$ is the matrix addition operation, and $\otimes$ denotes the Hadamard product.

matrices, and gradients of Grassmann parameters are composed of multiple column vectors). Using existing LSTMs for $\nabla_{\boldsymbol{M}}^{(t)}$ means that we will vectorize the gradients and apply vector multiplication to process the gradients. This inevitably destroys the matrix structures, such as ignoring valuable information about spatial correlations of column vectors. In our previous work [36], we proposed a matrix LSTM (mLSTM) model to preserve the symmetric property on the SPD manifold. In this work, our purpose is to learn to optimize for a big family of manifolds. As different manifolds have different matrix structures, it is challenging to use a model to preserve matrix structures for various manifolds. To address this challenge, we extend mLSTM to a generalized matrix LSTM (gmLSTM) model by utilizing a generalized transformation operation.

For convenience, we simplify the notion of gmLSTM as

$$\boldsymbol{H}^{(t)}, \boldsymbol{C}^{(t)} = gmLSTM(\boldsymbol{X}^{(t)}, \boldsymbol{S}^{(t-1)}), \tag{9}$$

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

4.2.1 Generalized Matrix LSTM (gmLSTM)

We define a simple and effective transformation $\boldsymbol{Y} = \Psi_{\boldsymbol{W}}(\boldsymbol{X})$ with parameter $\boldsymbol{W}$, to process gradients of various Riemannian parameters. The input $\boldsymbol{X} \in \mathcal{H}$ is assumed to have some structural properties, and we expect to preserve the output $\boldsymbol{Y}$ with the similar structure, or in other words, resides in the same domain, i.e., $\boldsymbol{Y} \in \mathcal{H}$. By using the transformation $\Psi_{\boldsymbol{W}}$, we can seamlessly utilize the LSTM architecture and avoid re-designing recurrent models for different manifolds. Given a manifold, we just need to consider the form of the transformation $\Psi_{\boldsymbol{W}}(\cdot)$. Taking the SPD manifold as an example, the gradient at a point is a symmetric matrix (i.e., $\mathcal{H}$ is the set of symmetric matrices). Thus, we use a bilinear projection as $\Psi_{\boldsymbol{W}}(\boldsymbol{X}) = \boldsymbol{W}^\top \boldsymbol{X} \boldsymbol{W}$, preserving the symmetric property. Forms of the transformation $\Psi_{\boldsymbol{W}}(\boldsymbol{X})$ on other manifolds, e.g., Stiefel, hyperbolic, and Grassmann manifolds, are given in Section 5.3.

Based on the transformation $\Psi_{\boldsymbol{W}}(\cdot)$ and the architecture of LSTM, we formulate gmLSTM as

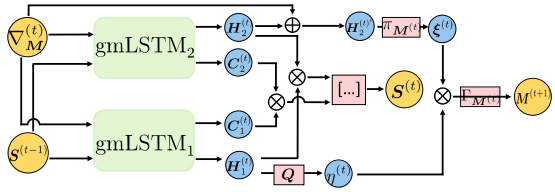


Fig. 3. The architecture of the proposed optimizer. [...] is the concatenation operation.

$$\begin{cases} \boldsymbol{F}^{(t)} = \sigma\big(\Psi_{\boldsymbol{W}_{fx}}(\boldsymbol{X}^{(t)}) + \Psi_{\boldsymbol{W}_{fh}}(\boldsymbol{H}^{(t-1)})\big) \\ \boldsymbol{T}^{(t)} = \sigma\big(\Psi_{\boldsymbol{W}_{ix}}(\boldsymbol{X}^{(t)}) + \Psi_{\boldsymbol{W}_{ih}}(\boldsymbol{H}^{(t-1)})\big) \\ \boldsymbol{O}^{(t)} = \sigma\big(\Psi_{\boldsymbol{W}_{ox}}(\boldsymbol{X}^{(t)}) + \Psi_{\boldsymbol{W}_{oh}}(\boldsymbol{H}^{(t-1)})\big) \\ \widehat{\boldsymbol{C}}^{(t)} = \tanh\big(\Psi_{\boldsymbol{W}_{cx}}(\boldsymbol{X}^{(t)}) + \Psi_{\boldsymbol{W}_{ch}}(\boldsymbol{H}^{(t-1)})\big) \\ \boldsymbol{C}^{(t)} = \boldsymbol{F}^{(t)} \otimes \boldsymbol{C}^{(t-1)} + \boldsymbol{T}^{(t)} \otimes \widehat{\boldsymbol{C}}^{(t)} \\ \boldsymbol{H}^{(t)} = \boldsymbol{O}^{(t)} \otimes \tanh(\boldsymbol{C}^{(t)}) \end{cases}, \tag{10}$$

where $\sigma(\cdot)$ is the sigmoid function, $\tanh(\cdot)$ is the hyperbolic tangent function, and $\otimes$ is the Hadamard product. $\boldsymbol{X}^{(t)}$ is the input, $\boldsymbol{C}^{(t)}$ is the memory cell, and $\boldsymbol{H}^{(t)}$ is the output. $\boldsymbol{C}^{(t)}$ and $\boldsymbol{H}^{(t)}$ have the same size with the input $\boldsymbol{X}^{(t)}$. In addition, $\boldsymbol{C}^{(t)}$ and $\boldsymbol{H}^{(t)}$ have the same matrix structure with the input $\boldsymbol{X}^{(t)}$. The gmLSTM model uses transformation operations with the parameters $\gamma = [\boldsymbol{W}_{fx}, \boldsymbol{W}_{fh}, \boldsymbol{W}_{ix}, \boldsymbol{W}_{ih}, \boldsymbol{W}_{ox}, \boldsymbol{W}_{oh}, \boldsymbol{W}_{cx}, \boldsymbol{W}_{ch}]$ to preserve the matrix structure.

4.2.2 Optimizer Architecture

We utilize two gmLSTMs to parameterize $g_{\phi_1}(\cdot)$ and $g_{\phi_2}(\cdot)$, and apply Riemannian operations to ensure that the optimizer will comply with Riemannian geometry, detailed as follows.

Calculating the Step Size. The function $g_{\phi_1}(\cdot)$ utilizes the first gmLSTM to compute the step size $\eta^{(t)}$ according to the following rule:

$$\begin{cases} \boldsymbol{H}_1^{(t)}, \boldsymbol{C}_1^{(t)} = gmLSTM_1(\nabla_{\boldsymbol{M}}^{(t)}, \boldsymbol{S}^{(t-1)}) \\ \eta^{(t)} = g_{\phi_1}(\nabla_{\boldsymbol{M}}^{(t)}, \boldsymbol{S}^{(t-1)}) = \boldsymbol{Q}^\top \boldsymbol{H}_1^{(t)} \end{cases}, \tag{11}$$

where $\boldsymbol{Q}$ is a learnable parameter.

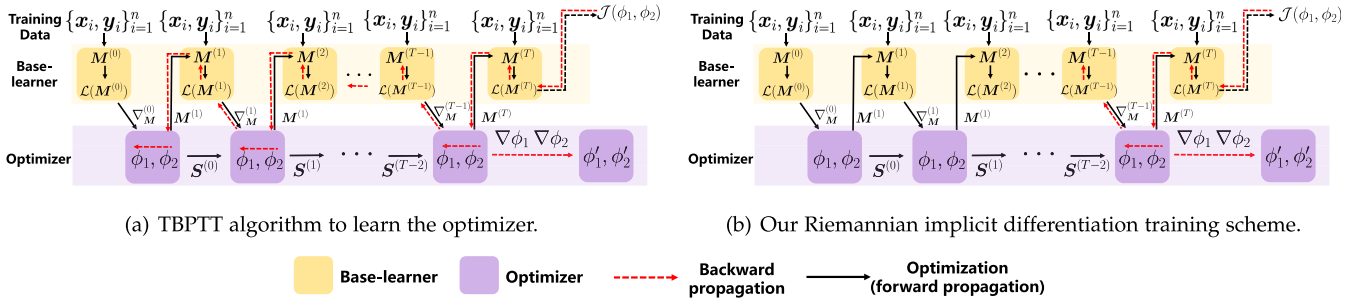
Calculating the Search Direction. The function $g_{\phi_2}(\cdot)$ utilizes the second gmLSTM to compute the search direction $\boldsymbol{\xi}^{(t)}$ according to

$$\begin{aligned} \boldsymbol{\xi}^{(t)} &= g_{\phi_2}(\nabla_{\boldsymbol{M}}^{(t)}, \boldsymbol{S}^{(t-1)}) \\ &= \pi_{\boldsymbol{M}^{(t)}}\Big(gmLSTM_2(\nabla_{\boldsymbol{M}}^{(t)}, \boldsymbol{S}^{(t-1)}) + \nabla_{\boldsymbol{M}}^{(t)}\Big), \end{aligned} \tag{12}$$

with

$$\begin{cases} \boldsymbol{H}_2^{(t)}, \boldsymbol{C}_2^{(t)} = gmLSTM_2(\nabla_{\boldsymbol{M}}^{(t)}, \boldsymbol{S}^{(t-1)}) \\ \boldsymbol{H}_2^{(t)'} = \boldsymbol{H}_2^{(t)} + \nabla_{\boldsymbol{M}}^{(t)} \\ \boldsymbol{\xi}^{(t)} = \pi_{\boldsymbol{M}^{(t)}}(\boldsymbol{H}_2^{(t)'}) \end{cases}. \tag{13}$$

Here, $gmLSTM_2$ is a gmLSTM model. We use a skip-connection to fit the residual between the search direction and the gradient for a more stable and easier training process. Hence we have $\boldsymbol{H}_2^{(t)'} = \boldsymbol{H}_2^{(t)} + \nabla_{\boldsymbol{M}}^{(t)}$. This is then followed by projecting $\boldsymbol{H}_2^{(t)'}$ onto the tangent space via the orthogonal projection $\pi_{\boldsymbol{M}^{(t)}}(\cdot)$ for the final search direction. The



(a) TBPTT algorithm to learn the optimizer.

(b) Our Riemannian implicit differentiation training scheme.

Fig. 4. Comparison between TBPTT and our Riemannian implicit differentiation training scheme. $\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^n$ is a batch of training data. $\boldsymbol{M}^{(t)}$ is the Riemannian parameter of the base-learner at time t, $\nabla_{\boldsymbol{M}}^{(t)}$ is the gradient, and $\boldsymbol{S}^{(t)}$ is the optimization state. In (a), TBPTT differentiates through the whole optimization trajectory to compute meta-gradients. In (b), the computation of meta-gradients is only related to the final two optimization steps.

optimization state $\boldsymbol{S}^{(t)}$ of the optimizer is updated by

$$\boldsymbol{S}^{(t)} = [\boldsymbol{C}_1^{(t)} \otimes \boldsymbol{C}_2^{(t)}, \boldsymbol{H}_1^{(t)} \otimes \boldsymbol{H}_2^{(t)}]. \tag{14}$$

In summary, parameters of $g_{\phi_1}(\cdot)$ and $g_{\phi_2}(\cdot)$ are $\phi_1 = [\gamma_1, \boldsymbol{Q}]$ and $\phi_2 = [\gamma_2]$, respectively, where γ_1 is the parameter of $gmLSTM_1$ and γ_2 is the parameter of $gmLSTM_2$. The architecture of the optimizer is shown in Fig. 3.

5 Training

We employ two optimization loops, i.e., an inner-loop and an outer-loop, to learn Riemannian optimizers in the meta-learning process. In the inner-loop, the Riemannian parameter $\boldsymbol{M}$ of the base-learner is updated for T steps by the optimizer, while the parameters ϕ_1 and ϕ_2 of the optimizer are learned in the outer-loop. Suppose that the initial Riemannian parameter is $\boldsymbol{M}^{(0)}$, we update the parameters ϕ_1, ϕ_2 by minimizing the following meta-objective

$$\mathcal{J}(\phi_1, \phi_2) \triangleq \mathcal{L}(\boldsymbol{M}^{(T)}), \tag{15}$$

where $\mathcal{L}$ is the loss function of the base-learner, and $\boldsymbol{M}^{(t)}$ is the Riemannian parameter at time t.

The conventional truncated backpropagation through time (TBPTT) algorithm seems to minimize Eq. (15), in line with existing meta-optimization methods in euclidean spaces [33], [34], [35]. However, training Riemannian optimizers using TBPTT will suffer from exploding gradients, and heavy computational load and memory footprint. In following sections, we provide an analysis of training Riemannian optimizers and present a Riemannian implicit differentiation training scheme to solve the aforementioned problem.

5.1 Analysis of the Traditional Training Scheme

We denote the collective parameters of the optimizer as $\phi = [\phi_1, \phi_2]$. The parameter ϕ of the optimizer is updated by $\phi \leftarrow \phi - \frac{d\mathcal{J}}{d\phi}$. The computation of the meta-gradient $\frac{d\mathcal{J}}{d\phi}$ with respect to optimizer parameters ϕ differentiates through the whole optimization trajectory in an inner-loop from $t = 0$ to $t = T$, as shown in Fig. 4a. Here, we provide a simple derivation of the meta-gradient. We ignore the optimization state and simplify the optimizer as $\boldsymbol{P}^{(t)} = b_\phi(\nabla \boldsymbol{M}^{(t)})$, where $\boldsymbol{P}^{(t)} = -\eta^{(t)}\boldsymbol{\xi}^{(t)}$ is the product of the step size $\eta^{(t)}$ and the search direction $\boldsymbol{\xi}^{(t)}$, and b_ϕ is the simplified optimizer containing g_{ϕ_1} and g_{ϕ_2}.

Recall the retraction operation $\Gamma_{\boldsymbol{M}^{(t)}}$, we can obtain a new Riemannian parameter by $\boldsymbol{M}^{(t+1)} = \Gamma_{\boldsymbol{M}^{(t)}}(-\boldsymbol{P}^{(t)})$. As the meta-objective is $\mathcal{J} \triangleq \mathcal{L}(\boldsymbol{M}^{(T)})$, the meta-gradient with respect to the parameter ϕ is

$$\begin{aligned}\frac{d\mathcal{J}}{d\phi} &= \nabla \boldsymbol{M}^{(T)} \frac{d\boldsymbol{M}^{(T)}}{d\phi} \\ &= \nabla \boldsymbol{M}^{(T)} \left(\frac{\partial \boldsymbol{M}^{(T)}}{\partial \boldsymbol{M}^{(T-1)}} \frac{d\boldsymbol{M}^{(T-1)}}{d\phi} + \frac{\partial \boldsymbol{M}^{(T)}}{\partial \boldsymbol{P}^{(T-1)}} \frac{d\boldsymbol{P}^{(T-1)}}{d\phi} \right),\end{aligned} \tag{16}$$

where $\frac{\partial \boldsymbol{M}^{(T)}}{\partial \boldsymbol{M}^{(T-1)}}$ and $\frac{\partial \boldsymbol{M}^{(T)}}{\partial \boldsymbol{P}^{(T-1)}}$ are partial derivatives of the retraction operation. $\frac{d\boldsymbol{P}^{(T-1)}}{d\phi}$ can be further computed according to the structure of the optimizer,

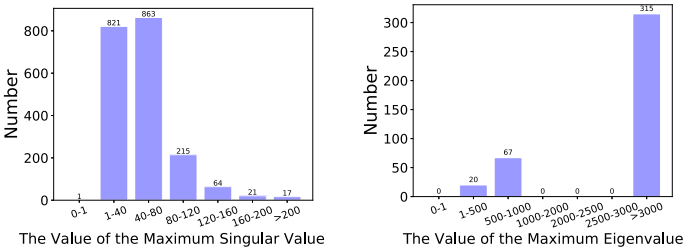
$$\begin{aligned}\frac{d\boldsymbol{P}^{(T-1)}}{d\phi} &= \frac{\partial \boldsymbol{P}^{(T-1)}}{\partial \phi} + \frac{\partial \boldsymbol{P}^{(T-1)}}{\partial \nabla \boldsymbol{M}^{(T-1)}} \frac{\partial \nabla \boldsymbol{M}^{(T-1)}}{\partial \phi} \\ &= \frac{\partial \boldsymbol{P}^{(T-1)}}{\partial \phi} + \frac{\partial \boldsymbol{P}^{(T-1)}}{\partial \nabla \boldsymbol{M}^{(T-1)}} \nabla^2 \boldsymbol{M}^{(T-1)} \frac{d\boldsymbol{M}^{(T-1)}}{d\phi},\end{aligned} \tag{17}$$

where $\frac{\partial \boldsymbol{P}^{(T-1)}}{\partial \phi}$ is the partial derivative of the optimizer with respect to ϕ, $\frac{\partial \boldsymbol{P}^{(T-1)}}{\partial \nabla \boldsymbol{M}^{(T-1)}}$ is the partial derivative of the optimizer with respect to the input gradient, and $\nabla^2 \boldsymbol{M}^{(T-1)}$ is the Hessian matrix. By substituting Eq. (17) into Eq. (16), the meta-gradient is given by

$$\begin{aligned}\frac{d\mathcal{J}}{d\phi} = \nabla \boldsymbol{M}^{(T)} \Bigg(\sum_{k=1}^{T-1} \Bigg(\frac{\partial \boldsymbol{M}^{(k+1)}}{\partial \boldsymbol{P}^{(k)}} \frac{\partial \boldsymbol{P}^{(k)}}{\partial \phi} \prod_{l=k+1}^{T-1} \\ \left(\frac{\partial \boldsymbol{M}^{(l+1)}}{\partial \boldsymbol{M}^{(l)}} + \frac{\partial \boldsymbol{M}^{(l+1)}}{\partial \boldsymbol{P}^{(l)}} \frac{\partial \boldsymbol{P}^{(l)}}{\partial \nabla \boldsymbol{M}^{(l)}} \nabla^2 \boldsymbol{M}^{(l)} \right) \Bigg) \Bigg).\end{aligned} \tag{18}$$

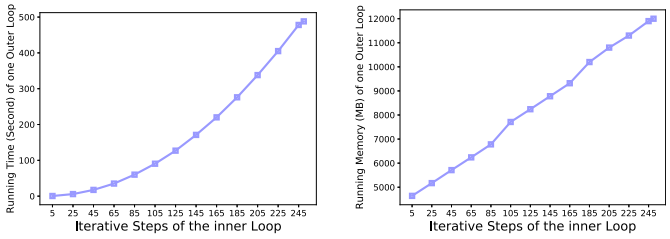
In Eq. (18), the computation of the meta-gradient differentiates through the whole optimization trajectory in the inner-loop, and involves computation of Hessian matrices $\nabla^2 \boldsymbol{M}^{(l)}$ and derivatives $\frac{\partial \boldsymbol{M}^{(l+1)}}{\partial \boldsymbol{M}^{(l)}}$ and $\frac{\partial \boldsymbol{M}^{(l+1)}}{\partial \boldsymbol{P}^{(l)}}$ of Riemannian retraction operations, over $t = 1$ to $t = T$. This makes training a Riemannian optimizer more challenging than training a euclidean optimizer. The reasons are as follows.

1) Eq. (18) is more prone to exploding gradients compared with euclidean meta-optimization. Similar to the euclidean case, the Riemannian meta-gradient involves products of Hessian matrices. This product makes meta-gradients grow exponentially with the increase of optimization steps T in the inner-loop, since the maximum eigenvalue of Hessian matrices is usually larger than one [51]. However, the Riemannian meta-gradient further requires the product of derivatives of retraction operations, over the entire



(a) Maximum singular values of gradients of retraction (b) Maximum eigenvalues of Hessian

Fig. 5. Distributions of the maximum eigenvalues or singular values of derivatives of retraction operations and Hessian matrices, measured on the PCA task in a Grassmann manifold using the MNIST dataset.



(a) Running Time (b) Running Memory

Fig. 6. Time and memory consumption of training a Riemannian optimizer using the TBPTT algorithm on the PCA task.

optimization trajectory in the inner-loop. We further study the derivatives of retraction operations to gain more insights about the Riemannian case. We empirically observe that the maximum singular value of derivatives of the retraction operation is usually larger than one, intensifying the problem of exploding gradients. Visualization for the maximum eigenvalue and singular values for a PCA problem is shown in Fig. 5. The exploding gradients heavily limit the application of Riemannian meta-optimization in practice. If there is an exploding gradient, we have to restart the training process, or manually load the previous model to continue the training process.

2) Riemannian meta-optimization suffers from heavy computational load and memory footprint. Training Riemannian optimizer needs to process and store large matrices with complex matrix functions over the whole inner-loop (e.g., time-consuming Hessian matrices and derivatives of retraction with singular value decomposition in Eq. (18)), leading to more computational load than euclidean counterparts that mainly process and store vectors. Furthermore, Eq. (18) depends on the whole optimization path in the inner-loop, which is completely stored in memory. As such, the both time and memory consumption of TBPTT is proportional to the inner-loop optimization length. We show an example of time/memory consumption for a PCA task in Fig. 6. The aforementioned factors lead to heavy memory footprint for training Riemannian optimizers. Thus, an efficient training scheme is necessary for Riemannian meta-optimization.

One way to address the training difficulties is to set a small number of optimization steps in the inner-loop, while this will lead to biased meta-gradients, damaging the quality of the optimizer and causes instabilities during training [52]. Here, we propose a Riemannian implicit differentiation training scheme that provides a feasible way to combat the inefficiency during training and mitigates the bias in meta-gradients.

5.2 Riemannian Implicit Differentiation Training Scheme

We establish a link between Riemannian optimization and the implicit function theorem [63] to derive an implicit form for computing the meta-gradient. The implicit form enables us to perform a large number of optimization steps in the inner-loop efficiently, as updating the optimizer is only related to the final two optimization steps of the inner-loop instead of the whole optimization trajectory. As a result, our training scheme avoids the exploding gradients and reduces both time and memory costs significantly (see Fig. 4b for a comparison).

5.2.1 Implicit Differentiation in Riemannian Manifolds

Suppose that we obtain an exact solution $\boldsymbol{M}^*$ by iterating in the inner-loop, i.e., $\nabla \boldsymbol{M}^* = \boldsymbol{0}$. Let us denote the solution obtained in the last update of $\boldsymbol{M}^*$ by $\boldsymbol{M}^{*'}$. Here we utilize the parameter $\boldsymbol{M}^{*'}$ to compute the meta-gradient,

$$\frac{d\mathcal{J}}{d\phi} = \nabla \boldsymbol{M}^{*'} \frac{d\boldsymbol{M}^{*'}}{d\phi}. \tag{19}$$

The derivative $\frac{d\boldsymbol{M}^{*'}}{d\phi}$ can be computed by the Riemannian implicit differentiation in Theorem 1.

Theorem 1. *If $\boldsymbol{M}^*$ is an exact solution in the inner-loop, the derivative $\frac{d\boldsymbol{M}^{*'}}{d\phi}$ can be computed implicitly by*

$$\frac{d\boldsymbol{M}^{*'}}{d\phi} = -\left(\frac{\partial \boldsymbol{M}^*}{\partial \boldsymbol{M}^{*'}} + \frac{\partial \boldsymbol{M}^*}{\partial \boldsymbol{P}^{*'}} \frac{\partial \boldsymbol{P}^{*'}}{\partial \nabla \boldsymbol{M}^{*'}} \nabla^2 \boldsymbol{M}^{*'}\right)^{-1} \cdot \frac{\partial \boldsymbol{M}^*}{\partial \boldsymbol{P}^{*'}} \cdot \frac{\partial \boldsymbol{P}^{*'}}{\partial \phi}. \tag{20}$$

Proof. Since $\boldsymbol{M}^*$ is the exact solution and $\nabla \boldsymbol{M}^* = \boldsymbol{0}$, we have

$$\frac{d\nabla \boldsymbol{M}^*}{d\phi} = \nabla^2 \boldsymbol{M}^* \frac{d\boldsymbol{M}^*}{d\phi} = \boldsymbol{0}. \tag{21}$$

The Hessian matrix $\nabla^2 \boldsymbol{M}^*$ is a symmetric positive definite matrix, $\nabla^2 \boldsymbol{M}^* \neq \boldsymbol{0}$, and thus, $\frac{d\boldsymbol{M}^*}{d\phi} = \boldsymbol{0}$.

Recall that $\boldsymbol{M}^*$ is obtained via the retraction operation, $\boldsymbol{M}^* = \Gamma_{\boldsymbol{M}^{*'}}(-\boldsymbol{P}^*)$, the derivative $\frac{d\boldsymbol{M}^*}{d\phi}$ can be computed by the chain rule,

$$\frac{d\boldsymbol{M}^*}{d\phi} = \left(\frac{\partial \boldsymbol{M}^*}{\partial \boldsymbol{M}^{*'}} + \frac{\partial \boldsymbol{M}^*}{\partial \boldsymbol{P}^{*'}} \frac{\partial \boldsymbol{P}^{*'}}{\partial \nabla \boldsymbol{M}^{*'}} \nabla^2 \boldsymbol{M}^{*'}\right) \frac{d\boldsymbol{M}^{*'}}{d\phi} + \frac{\partial \boldsymbol{M}^*}{\partial \boldsymbol{P}^{*'}} \frac{\partial \boldsymbol{P}^{*'}}{\partial \phi}. \tag{22}$$

As the derivative $\frac{d\boldsymbol{M}^*}{d\phi} = \boldsymbol{0}$, we have

$$\frac{d\boldsymbol{M}^{*'}}{d\phi} = -\left(\frac{\partial \boldsymbol{M}^*}{\partial \boldsymbol{M}^{*'}} + \frac{\partial \boldsymbol{M}^*}{\partial \boldsymbol{P}^{*'}} \frac{\partial \boldsymbol{P}^{*'}}{\partial \nabla \boldsymbol{M}^{*'}} \nabla^2 \boldsymbol{M}^{*'}\right)^{-1} \cdot \frac{\partial \boldsymbol{M}^*}{\partial \boldsymbol{P}^{*'}} \cdot \frac{\partial \boldsymbol{P}^{*'}}{\partial \phi}. \tag{23}$$

□

In Eq. (20), the derivative $\frac{d\boldsymbol{M}^{*'}}{d\phi}$ only depends on the final two optimization steps of the inner-loop, rather than the

whole trajectory in the inner-loop. In other words, no matter how long the optimization trajectory in the inner-loop is, we only need to compute the partial derivatives of retraction operations $\frac{\partial \boldsymbol{M}^*}{\partial \boldsymbol{M}^{*'}}\frac{\partial \boldsymbol{M}^*}{\partial \boldsymbol{P}^{*'}}$ in the last step, partial derivatives of the optimizer $\frac{\partial \boldsymbol{P}^{*'}}{\partial \nabla \boldsymbol{M}^{*'}}\frac{\partial \boldsymbol{P}^{*'}}{\partial \phi}$ in the penultimate step, and the Hessian matrix $\nabla^2 \boldsymbol{M}^{*'}$ in the penultimate step. We do not need to compute or store derivatives in previous steps. This circumvents computing products of gradients of Riemannian retraction operations and products of Hessian, avoiding the exploding gradient problem. Both time and memory consumption of our method is very small (complexity analysis is in Section 5.3.2 and experimental results are in Section 6.3.2).

Note that the implicit derivative $\frac{d\boldsymbol{M}^{*'}}{d\phi}$ in Theorem 1 needs a matrix inversion $(\frac{\partial \boldsymbol{M}^*}{\partial \boldsymbol{M}^{*'}} + \frac{\partial \boldsymbol{M}^*}{\partial \boldsymbol{P}^{*'}}\frac{\partial \boldsymbol{P}^{*'}}{\partial \nabla \boldsymbol{M}^{*'}}\nabla^2 \boldsymbol{M}^{*'})^{-1}$. This is nontrivial to compute as matrix inversion is intractable for large matrices. To address this problem, we utilize Neumann series [55] to iteratively approximate the matrix inversion. We initialize two matricex $\boldsymbol{v}^0$ and $\boldsymbol{u}^0$ with the identity matrix, and update $\boldsymbol{u}^{(i)}$ as

$$\begin{cases} \boldsymbol{v}^{(i+1)} = \boldsymbol{v}^{(i)} - \boldsymbol{v}^{(i)}\frac{\partial \boldsymbol{M}^*}{\partial \boldsymbol{M}^{*'}} - \boldsymbol{v}^{(i)}\frac{\partial \boldsymbol{M}^*}{\partial \boldsymbol{P}^{*'}}\frac{\partial \boldsymbol{P}^{*'}}{\partial \nabla \boldsymbol{M}^{*'}}\nabla^2 \boldsymbol{M}^{*'} \\ \boldsymbol{u}^{(i+1)} = \boldsymbol{u}^{(i)} + \boldsymbol{v}^{(i+1)} \end{cases}. \tag{24}$$

After K iterations, the derivative $\frac{d\boldsymbol{M}^{*'}}{d\phi}$ is

$$\frac{d\boldsymbol{M}^{*'}}{d\phi} = -\boldsymbol{u}^{(K)}\frac{\partial \boldsymbol{M}^*}{\partial \boldsymbol{P}^{*'}}\cdot\frac{\partial \boldsymbol{P}^{*'}}{\partial \phi}. \tag{25}$$

Eqs. (24) only need to compute the Jacobian-vector product $\boldsymbol{v}^{(i)}\frac{\partial \boldsymbol{M}^*}{\partial \boldsymbol{M}^{*'}}$ and the Hessian-vector product $\boldsymbol{v}^{(i)}\frac{\partial \boldsymbol{M}^*}{\partial \boldsymbol{P}^{*'}}\frac{\partial \boldsymbol{P}^{*'}}{\partial \nabla \boldsymbol{M}^{*'}}\nabla^2 \boldsymbol{M}^{*'}$, which are much easier to compute than the original formula with the matrix power operation [64]. Subsituting Eq. (25) into Eq. (19), the meta-gradient is

$$\frac{d\mathcal{J}}{d\phi} = \nabla \boldsymbol{M}^{*'}(-\boldsymbol{u}^{(K)})\frac{\partial \boldsymbol{M}^*}{\partial \boldsymbol{P}^{*'}}\cdot\frac{\partial \boldsymbol{P}^{*'}}{\partial \phi}. \tag{26}$$

The number of iterations in the inner-loop to obtain the exact solution is unknown. For simplicity, we set a fixed number T in the inner-loop, and use our optimizer to update $\boldsymbol{M}^{(0)}$ to an approximate solution $\boldsymbol{M}^{(T)}$.

5.2.2 Parameter Warmup Pool

Training in machine learning requires data to be independent and identically distributed. However, Riemannian parameters and optimization states obtained in an inner-loop are strongly correlated sequentially. Besides, it is difficult to optimize the Riemannian parameter from scratch to an exact solution utilizing an untrained optimizer. To handle the two issues, we build a parameter warmup pool $\mathbb{P}$ that stores good solutions as initial Riemannian parameters $\boldsymbol{M}^{(0)}$. Before training the optimizer, we utilize a hand-designed optimizer such as RSGD to obtain good solutions whose gradient norms are smaller than a small threshold, and push them into the parameter warmup pool $\mathbb{P}$. In the training stage, for each step of the outer-loop, we randomly select Riemannian parameters $\boldsymbol{M}^{(0)}$ from the parameter warmup pool $\mathbb{P}$ to train our optimizer. To avoid overfitting, we set a hyperparameter τ to update $\mathbb{P}$. After each τ steps in the outer-loop, we update the parameter warmup pool $\mathbb{P}$ by regenerating solutions as initial parameters and push them into the parameter warmup pool $\mathbb{P}$. The training process of our Riemannian meta-optimization (RMO) method is summarized in Algorithm 1.

Algorithm 1. Training of RMO

Input: The initial optimizer. The initial Riemannian parameter $\boldsymbol{M}^{(0)}$. The initial optimization state $\boldsymbol{S}^{(0)} = \boldsymbol{0}_d$. The initial experience pool $\mathbb{P} = \emptyset$. Maximum iteration T of the inner-loop, maximum iteration Υ of the outer-loop, and the hyperparameter τ to update the parameter warmup pool $\mathbb{P}$.
Output: The optimizer parameter $\phi = \{\phi_1, \phi_2\}$.

```
r = 0.
while r ≤ Υ do
  if r mod τ then
    Construct the parameter warmup pool ℙ.
  end if
  Randomly select M^(0) from ℙ, and set t = 0.
  while t ≤ T do
    Compute the loss on training data ℒ(M^(t)).
    Compute the gradient ∇M^(t).
    Update M^(t) by our optimizer via Eq. (7).
    t ← t + 1.
  end while
  Compute the loss 𝒥 of the optimizer by Eq. (15).
  Compute the implicit gradient d𝒥/dϕ by Eq. (26).
  Update ϕ of our optimizer.
  r ← r + 1.
end while
Return the parameter ϕ of our optimizer.
```

5.3 Exploration on Various Manifolds

5.3.1 Optimizers on Four Common Manifolds

RMO can be applied to various manifolds. We just need to consider forms of the transformation $\Psi_{\boldsymbol{W}}(\cdot)$, orthogonal projection $\pi_{\boldsymbol{M}}(\cdot)$, and retraction $\Gamma_{\boldsymbol{M}}(\cdot)$ for a given manifold. In this section, we develop their corresponding forms to perform optimization on four popular Riemannian manifolds, namely hyperbolic, Stiefel, Grassmann, and SPD manifolds, where details are shown in Table 1. On the Stiefel, Grassmann, and SPD manifolds, some Riemannian operations use matrix decomposition or matrix functions, such as singular value decomposition for Grassmann manifolds. An efficient way to perform these matrix decomposition/functions and compute their derivatives can be found in the work [65].

5.3.2 Complexity Analysis on the Four Manifolds

Computational Complexity of Our Optimizer. For our optimizer, its optimization procedure mainly involves matrix multiplication, eigenvalue decomposition, singular value decomposition, and element-wise operations. For a $d \times d$ matrix, the eigenvalue decomposition requires $O(d^3)$. For a $d \times p$ matrix, the element-wise operations require $O(dp)$ flops, singular value decomposition requires $O(\max(d,p)^3)$, and the matrix multiplication with a $p \times n$ matrix requires $O(dpn)$. This leads to $O(17d^2 + 51d)$, $O(3d^3 + 5pd^2 + 18p^2d + d^2 + 31pd + d)$, $O(p^3 + 20p^2d + pd^2 + 29pd)$, and $O(49d^3 + 29d^2 + 4d)$ flops for the proposed optimizer to perform optimization on hyperbolic, Stiefel, Grassmann, and SPD manifolds, respectively.

TABLE 1
Transformation $\Psi_{\boldsymbol{W}}(\cdot)$, Orthogonal Projection $\pi_{\boldsymbol{M}}(\cdot)$, and Retraction $\Gamma_{\boldsymbol{M}}(\cdot)$ on Hyperbolic, Stiefel, Grassmann, and SPD Manifolds

Manifold	Transformation $\Psi_W(X)$	Orthogonal Projection $\pi_M(\nabla_M)$	Retraction $\Gamma_M(P)$
Hyperbolic Manifold, $\boldsymbol{M} \in \mathcal{H}^d$	$\boldsymbol{WX}, \boldsymbol{W} \in \mathbb{R}^{d\times d}$	$\frac{1}{\lambda_{\boldsymbol{M}}^2}\nabla_{\boldsymbol{M}}$	$\boldsymbol{M}\oplus(\tanh(\frac{\lambda_{\boldsymbol{M}}\|\boldsymbol{P}\|}{2})\frac{\boldsymbol{P}}{\|\boldsymbol{P}\|})$
Stiefel Manifold, $\boldsymbol{M} \in St(p,d)$	$\boldsymbol{WX}, \boldsymbol{W} \in \mathbb{R}^{d\times d}$	$\nabla_{\boldsymbol{M}} - \boldsymbol{M}(\boldsymbol{M}^\top\nabla_{\boldsymbol{M}})_{Sym}$	$\mathrm{u}f(\boldsymbol{M}+\boldsymbol{P})$
Grassmann Manifold, $\boldsymbol{M} \in \mathcal{G}(p,d)$	$\boldsymbol{WX}, \boldsymbol{W} \in \mathbb{R}^{d\times d}$	$\nabla_{\boldsymbol{M}} - \boldsymbol{M}\boldsymbol{M}^\top\nabla_{\boldsymbol{M}}$	$\boldsymbol{U}_g\boldsymbol{V}_g^\top$ where $[\boldsymbol{U}_g, \boldsymbol{\Sigma}_g, \boldsymbol{V}_g] = SVD(\boldsymbol{M}+\boldsymbol{P})$
SPD Manifold, $\boldsymbol{M} \in \mathrm{Sym}_d^+$	$\boldsymbol{W}^\top\boldsymbol{XW}, \boldsymbol{W} \in \mathbb{R}^{d\times d}$	$\boldsymbol{M}(\nabla_{\boldsymbol{M}})_{Sym}\boldsymbol{M}$	$\boldsymbol{M}^{\frac{1}{2}}expm(\boldsymbol{M}^{-\frac{1}{2}}\boldsymbol{P}\boldsymbol{M}^{-\frac{1}{2}})\boldsymbol{M}^{\frac{1}{2}}$

$\lambda_{\boldsymbol{M}} = 2/(1-\|\boldsymbol{M}\|^2)$ is the conformal factor. $\boldsymbol{M}\oplus\boldsymbol{X} = \frac{(1+2\langle\boldsymbol{M},\boldsymbol{X}\rangle+\|\boldsymbol{X}\|^2)\boldsymbol{M}+(1-\|\boldsymbol{M}\|^2)\boldsymbol{X}}{1+2\langle\boldsymbol{M},\boldsymbol{X}\rangle+\|\boldsymbol{M}\|^2\|\boldsymbol{X}\|^2}$ is the addition on the hyperbolic manifold. $(\boldsymbol{A})_{Sym} = \frac{1}{2}(\boldsymbol{A}+\boldsymbol{A}^\top)$, $\mathrm{u}f(\boldsymbol{A}) = \boldsymbol{A}(\boldsymbol{A}^\top\boldsymbol{A})^{-\frac{1}{2}}$ is an operation that yields an orthogonal matrix, $[\boldsymbol{U},\boldsymbol{\Sigma},\boldsymbol{V}] = SVD(\boldsymbol{A})$ is the singular value decomposition, $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, and $expm(\boldsymbol{A})$ is the matrix exponential operation.

Computational Complexity of Meta-gradient Computation. Some works show that the time of computing gradients or Jacobian-vector products of a function is no more than 5 times it takes to compute the function itself, and the time of computing Hessian-vector products is also no more than 5 times it takes to compute the gradient [66]. Based on the two rules, the computational complexity of our implicit meta-gradient is no more than $O(90K+90)d^2 + (445K+228)d$, $O((90K+90)p^2d + (35K+25)pd^2 + (30K+15)d^3 + (198K+160)pd + (25K+5)p^2 + (5K+5)d^2 + (10K+5)d)$, $O((10K+5)p^3 + (125K+105)p^2d + (10K+5)pd^2 + (153K+145)pd + (25K+5)p^2 + (25K+5)P)$, and $O((585K+300)d^3 + (148K+145)d^2 + (110K+35)d)$ on hyperbolic, Stiefel, Grassmann, and SPD manifolds, respectively. In contrast, the computational complexity of TBPTT is $O(90T^2+45T)d^2 + (293T^2 - \frac{85}{2}T - \frac{1}{2})d$, $O((45T^2+45T)p^2d + (\frac{35}{2}T^2 + \frac{15}{2}T)pd^2 + 15T^2d^3)$, $O(5T^2p^3 + (\frac{125}{2}T^2 + \frac{75}{2}T + 5)p^2d + 5T^2pd^2)$, and $O((\frac{585}{2}T^2 - \frac{95}{2}T + 55)d^3)$ on the hyperbolic, Stiefel, Grassmann, and SPD manifolds, respectively.

The computational complexity of training our optimizer is independent of the maximum optimization steps T of the inner-loop. Thus, with the increase of optimization steps in the inner-loop, the running-time of our method to calculate the meta-gradient remains constant and very small. Though our method is linearly related to the iteration K of approximate Neumann series, K is far less than T (we set $K=5$ and $T=50$ in experiments).

Memory Cost. In the training process, our optimizer is updated by Eq. (26), where we only need to store $\boldsymbol{M}^*$, $\boldsymbol{M}^{*'}$, $\nabla\boldsymbol{M}^{*'}$, $\boldsymbol{P}^{*'}$, and the optimizer in the memory. We do not need to consider the optimization state $\boldsymbol{S}$ in the memory costs, because the memory it occupies will be immediately freed when it is sent to the optimizer for the next update iteration. Totally, the memory cost of the Riemannian implicit differentiation training scheme is $O(4dp+SP)$, where $SP = 17d^2$ is the number of parameters in the Riemannian optimizer. The memory cost is a constant, independent of the maximum optimization steps T of the inner-loop.

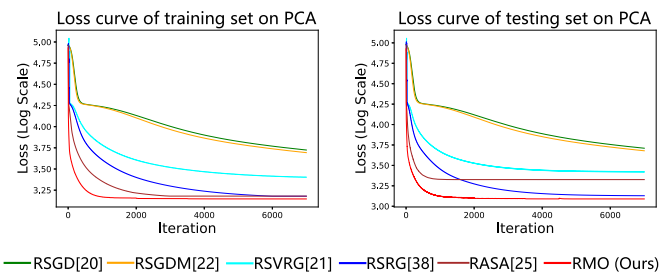


Fig. 7. Plots for the PCA task (in the log scale). Our optimizer converges very quickly. RSGD reaches a loss of 3.20 and 3.18 on the training and testing sets, respectively. RSGDM achieves 3.19 and 3.17. In contrast, RMO achieves 3.14 and 3.08.

6 Experiments

6.1 Empirical Evaluation

Following state-of-the-art Riemannian optimization works [22], [25], [38], we evaluated RMO on the principal component analysis (PCA), clustering, and face recognition, and similarity learning tasks. We compared our optimizer with state-of-the-art Riemannian optimizers: RSGD [20], RSGDM [22], RSVRG [21], RSRG [38], and RASA [25]. Following the official scheme in their papers, all hyperparameters of compared optimizers were tuned by the grid search scheme to achieve the best performance. We trained our optimizer on the training set and evaluated its performance on both the training and testing sets. We set $T=50$ in the inner-loop.

6.1.1 PCA on the Grassmann Manifold

Given a set of samples $\{\boldsymbol{x}_i \in \mathbb{R}^d\}_{i=1}^n$, the PCA task aims to learn an orthogonal matrix $\boldsymbol{M} \in \mathbb{R}^{d\times p}$, $\boldsymbol{M}^\top\boldsymbol{M} = \mathbf{I}_p$, to preserve the maximum energy of the data in a p-dimensional space, $p < d$. The orthogonal matrix $\boldsymbol{M}$ satisfies the rotation invariance property that enforces $\boldsymbol{M}$ to be on the Grassmann manifold, $\boldsymbol{M} \in \mathcal{G}(p,d)$. The task can be formulated as

$$\min_{\boldsymbol{M}\in\mathcal{G}(p,d)} \mathcal{L}(\boldsymbol{M}) = \frac{1}{n}\sum_{i=1}^{n}\left\|\boldsymbol{x}_i - \boldsymbol{M}\boldsymbol{M}^\top\boldsymbol{x}_i\right\|^2. \tag{27}$$

We used on the MNIST dataset. We represented images by $d=784$-dimensional vectors and aim to learned $p=32$-dimensional representations. Loss curves are shown in Fig. 7.

6.1.2 Clustering on the SPD Manifold

We considered the task of clustering on the SPD manifold Sym_d^+. Given a set of SPD points $\{\boldsymbol{X_i} \in \mathrm{Sym}_d^+\}$, we aimed to find SPD centroids by solving the following problem

$$\min_{\{\boldsymbol{M}_r\}_{r=1}^C\in\mathrm{Sym}_d^+} \mathcal{L}(\boldsymbol{M}) = \sum_{i\rightsquigarrow r}^{n} d(\boldsymbol{X}_i, \boldsymbol{M}_r), \tag{28}$$

where $i\rightsquigarrow r$ shows that sample $\boldsymbol{X}_i$ is assigned to $\boldsymbol{M}_r$. We adopted the affine invariance metric (AIM) [18] to

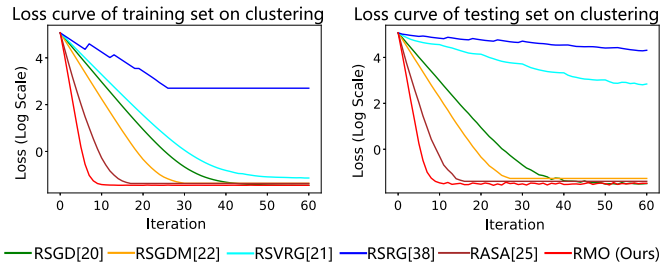


Fig. 8. Plots for the clustering task (in the log scale). On the testing set, RSVRG achieves losses of 3.54. Among compared methods, RASA achieves the lowest loss values, -0.70 on the testing set. In contrast, the proposed optimizer outperforms all the compared methods including RASA, achieving losses of values -0.88 on the testing set.

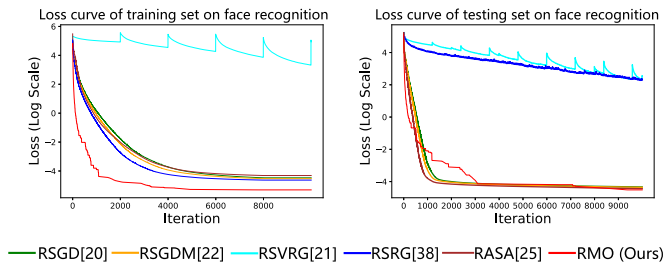


Fig. 9. Plots for the face recognition task (in the log scale). On the training set, RSRG and RSVRG converge to loss values -4.42 and -3.63, while our optimizer achieves -5.31. On the testing set, RSRG and RSVRG converge to -3.47 and -3.73, while our optimizer achieves -4.83.

compute the distance between $\boldsymbol{X}_i$ and the centroid $\boldsymbol{M}_j$. We conducted experiments on the Kylberg texture dataset [67] represented each image by a 5×5 covariance descriptor. Results are shown in Fig. 8. We also evaluated the solved centroids. We regarded the centroids as category prototypes and computed AIM between test samples and prototypes to classify test samples. Results are shown in Table 2.

6.1.3 Face Recognition on the Stiefel Manifold

The orthogonality constraint has been widely used to attain robust features, making the parameter on the Stiefel manifold. Here, we considered face recognition using a linear classifier with the orthogonality constraint. Given image features $\{\boldsymbol{x}_i \in \mathbb{R}^d\}_{i=1}^n$ and labels $\{\boldsymbol{y}_i \in \mathbb{R}^p\}_{i=1}^n$, we trained a linear model by minimizing the loss function

$$\min_{\boldsymbol{M} \in \mathrm{St}(p,d)} \mathcal{L}(\boldsymbol{M}) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{l=1}^{p} \mathbb{1}(y_{il}=1)\log \frac{\exp\big((\boldsymbol{M}^\top \boldsymbol{x}_i)_l\big)}{\sum_{l'=1}^{p}\exp\big((\boldsymbol{M}^\top \boldsymbol{x}_i)_{l'}\big)}, \tag{29}$$

where $\boldsymbol{M} \in \mathrm{St}(p,d)$ is the weight matrix of the classifier, and y_{il} is the l-th element of the label $\boldsymbol{y}_i$. $\mathbb{1}(y_{il}=1)$ denotes if $y_{il}=1$, the value of $\mathbb{1}(y_{il}=1)$ is 1, and otherwise is 0.

We used the YaleB dataset [68]. We cropped and resized the face regions to 32×32 pixels and represented each region as a 1024-dimensional feature vector. Loss curves are shown in Fig. 9. We then evaluated the linear classifier. We solved the linear classifier using the training set, and measured accuracies on the testing set. Results are shown in Table 2.

TABLE 2
Accuracies (%) on the YaleB, Kylberg, and CUB Datasets

Method	Kylberg	CUB	YaleB
w/o constraint	-	66.1	74.80
RSGD [20]	81.25	70.44	79.49
RSVRG [21]	83.19	53.74	79.67
RSRG [38]	70.31	65.53	78.05
RSGDM [22]	82.92	70.15	79.64
RASA [25]	83.40	70.90	80.21
RMO (Ours)	**86.70**	**71.01**	**95.10**

6.1.4 Similarity Learning on the SPD Manifold

Similarity learning aims to learn a Mahalanobis metric that computes the distance between $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i'}$ by

$$d_{\boldsymbol{M}}(\boldsymbol{x}_i, \boldsymbol{x}_{i'}) = \sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_{i'})^\top \boldsymbol{M}(\boldsymbol{x}_i - \boldsymbol{x}_{i'})}, \tag{30}$$

where the non-negative condition of the metric requires $\boldsymbol{M}$ to be an SPD matrix, i.e., $\boldsymbol{M} \in Sym_{++}^d$. We utilized the contrastive loss to guide learning the parameter.

We conducted experiments on the fine-grained CUB dataset [69]. We added a fully-connected layers whose output dimension is 128 following the VGG-16 network, and extracted 128-dimensional features for images. Loss curves are shown in Fig. 10. We then evaluated the performance of the solved metric matrix, and classification results are shown in Table 2.

6.1.5 Analysis

Compared to the state-of-the-art hand-designed optimizers, RSGD, RSGDM, RSRG, RSVRG, and RASA, the proposed optimizer via RMO achieves better performances for all the four tasks. The proposed optimizer achieves faster convergence speed and obtains the best optima. Furthermore, the proposed optimizer works well on both the seen training data and unseen test data, showing that the learned optimizer is able to generalize to unseen data. The reason is that the proposed optimizer is learned from data, and hence can discover an appropriate updating scheme by exploring the underlying data distribution of given tasks. Our optimizer adapts the learning rate and search direction based on both the current gradient and the previous optimization state. In contrast, existing Riemannian optimizers are hand-designed and have fixed updating schemes. That is, the learning rate is fixed during optimization steps and the search direction is obtained by

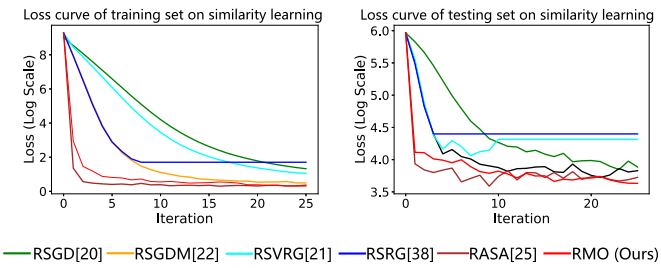


Fig. 10. Plots for the similarity learning task (in the log scale).

TABLE 3
Optimization Time (Seconds) of Riemannian Optimizers in 100 Iterations

Method	PCA	Face Recognition	Clustering	Similarity Learning
RSGD [20]	57.2	51.7	368.0	150.3
RSGDM [22]	56.9	100.2	389.8	203.9
RSRG [38]	14.9	70.3	1248.4	1226.6
RSVRG [21]	23.2	70.3	802.5	1046.8
RASA [25]	57.7	222.8	375.3	138.0
RMO-base-learner	33.9	43.2	360.0	117.4
RMO-optimizer	130.6	68.1	19.0	37.0
RMO (Ours)	164.5	111.3	379.0	154.4

a fixed scheme. Such fixed updating schemes may be not suitable for all data. In Section 6.4, we provide further analysis via visualization to better demonstrate the differences between our algorithm and hand-crafted Riemannian optimizers.

Focusing on the accuracy, the proposed method achieves the best performance in our experiments. For example, on the clustering task, our method achieves 86.70%, 5.45% higher than RSGD and 3.3% higher than RASA. On the YaleB dataset, the best accuracy of hand-designed optimizers is 80.21%, achieved by RASA. In contrast, RMO achieves 95.10%, 14.89% higher than RASA. This shows that the optima obtained by our optimizer better fits to the training data while being able to generalize to new data. Again, we conjecture that the reason is similar to the optimization performance, simply our method better explores the loss surface, and hence is able to find better optimas due to its data-driven nature.

6.2 Efficiency

We compares the wall clock time of using our optimizer against state-of-the-art Riemannian optimization algorithms, as shown in Table 3. Evaluations were conducted on an Inter (R) Core(TM) i7-7820X 3.6GHz CPU, GeForce RTX 2080Ti GPU and 32GB RAM. The wall clock time for all optimizers was measured in 100 iterations. On the clustering tasks and similarity learning tasks that is on the SPD manifold, our optimizer is faster than RSGDM, RSVRG, and RSRG. This is because the three Riemannian optimizers utilize the parallel transport to benefit from previous optimization results. On the SPD manifold, parallel transport is heavy in computation, since it requires matrix inversion and matrix power operations. In contrast, the gmLSTM in our optimizer directly works with the previous optimization state via simpler matrix multiplication and element-wise operations. We note that for the PCA and face recognition tasks that are on Grassmann and Stiefel manifolds, our optimizer requires comparable or marginally more time consumption as compared with existing Riemannian optimizers. This is because the parallel transport operations on the Stiefel and Grassmann manifolds have simpler forms. Although the optimization time is not our advantage, our optimizer has a faster convergence speed. This translate into saving time as less optimization steps are required. Besides, our optimizer achieves a better optima and avoids lots of human involvement. We observe that RSRG achieves different time consumption on different tasks, i.e., large time consumption on the clustering and similarity learning tasks and small consumption on the PCA task. The reason is that a hyperparameter (the number of iterations in the inner-loop) is set differently. RSRG is a variance reduction optimization method that uses double loops (an inner-loop and an outer-loop) in the optimization process, and the number of iterations in the inner-loop plays an dominant role in the time consumption of optimization. We set the number of iterations in the inner-loop as 5 for the similarity learning and clustering tasks, and 1000 for the PCA task, to achieve the best performance, causing the different time consumption on different tasks.

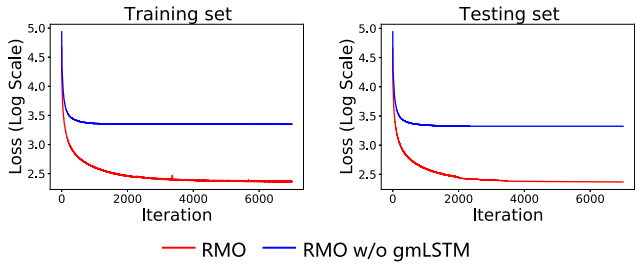


Fig. 11. Ablation study of gmLSTM on the PCA task. On the training and testing sets, RMO w/o mLSTM finally converges at 3.35 and 3.32, while RMO converges at 2.37 and 2.36 (log scale).

We also measured the time consumption of the base-learner and the optimizer. The time consumption of the base-learner was measured over the forward process, and the time consumption of the optimizer was measured over the processes of calculating gradients and updating parameters. Results are also shown in Table 3.

We can see in Table 3 that, on the PCA and face recognition tasks, more time is spent in optimizing Riemannian parameters. For the clustering and similarity learning tasks, more time is spent in the forward process of the model. The reasons are as follows. (1) The forward process is different among the four tasks. In the clustering task, we need to compute the affine invariance metric [18] on the SPD manifold in the forward process, which is heavy in computation, since it requires matrix inversion and matrix power operations on matrices. In the similarity learning task, we need to compute the distance between each pair of samples in the forward process, which is also time-consuming. In contrast, the forward processes of the PCA and face recognition tasks are not complicated. (2) The optimizer has different complexities among the four tasks. The dimension of learnable parameters in the optimizer is high in the PCA and face recognition tasks, i.e., 784×784 and 1024×1024 in the two tasks, respectively. Contrarily, the dimension is 5×5 and 128×128 in the clustering and similarity learning tasks, respectively.

6.3 Ablation Study

6.3.1 *gmLSTM*

In RMO, gmLSTM plays an important role to preserve matrix structures of gradients. In this section, we conducted ablation studies on the PCA task and the clustering task to assess gmLSTM. Concretely, we replaced gmLSTM in the optimizer with a conventional LSTM, and the LSTM will destroy the matrix structure of gradients. We use this experiment to evaluate whether we need to preserve matrix structures in Riemannian optimization. When carrying out optimization, we

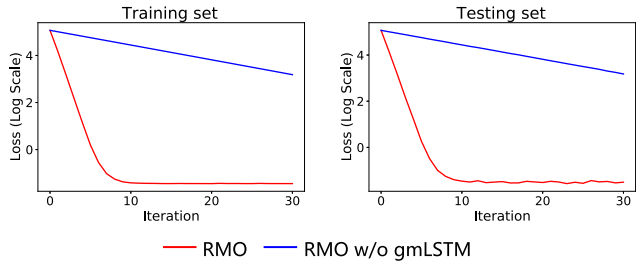


Fig. 12. Ablation study of gmLSTM on the clustering task. On the training and testing sets, RMO w/o mLSTM finally converges at 2.55 and 2.56, while RMO converges at -1.45 and -1.57 (log scale).

flattened gradient matrices $\nabla_{\boldsymbol{M}}^{(t)}$ into vectors and sent the vectors into the optimizer. Then, we reshaped outputs of conventional LSTM into matrices with the same shape as $\nabla_{\boldsymbol{M}}^{(t)}$. Results on the PCA and clustering tasks are shown in Figs. 11 and 12. Results suggest that the optimizer parameterized by conventional LSTM converges slowly and have a worse optima. In contrast, our optimizer parameterized by gmLSTM converge faster to a better optima. This confirms the importance of preserving matrix structures when learning the Riemannian optimizer.

6.3.2 Riemannian Implicit Differentiation Training Scheme

Time/memory Consumption. We measured both time and memory consumption of training the Riemannian optimizer with ('w/ RIDTS') and without ('w/o RIDTS') the Riemannian implicit differentiation training scheme (see Tables 4 and 5). For 'w/o RIDTS', we utilized the TBPTT algorithm to train the Riemannian optimizer with the same number of inner-loop and outer-loop steps as ours. Results suggest that using the Riemannian implicit differentiation training scheme saves tangible amount of time and memory as compared to using the TBPTT algorithm. The reason is that our training scheme does not need to differentiate through the optimization trajectory in the inner-loop, reducing computational cost. Furthermore, with the implicit formulation, we do not need to keep the whole optimization trajectory, which in terms reduces the memory footprint of our algorithm significantly. Add to this the fact that TBPTT suffers from the exploding gradient issue, while our training scheme does not. When the exploding gradient occurs, the TBPTT has to retrain a model or load a saved model to continue training, resulting in extra training time. Thus, Riemannian implicit differentiation training scheme makes our method more efficient.

Meta-Gradient. We visualized meta-gradients of 'w/ RIDTS' and 'w/o RIDTS' on the PCA tasks (see Fig. 13).

TABLE 4
Time (Hours) of Training an Optimizer

Method	PCA	Face Recognition	Clustering	Similarity Learning
w/o RIDTS	1.33×10^{1}	5.97	6.27	8.70×10^{-1}
w/ RIDTS	1.63×10^{-1}	4.11×10^{-2}	5.30	5.80×10^{-1}

TABLE 5
Memory (MB) of Training an Optimizer

Method	PCA	Face Recognition	Clustering	Similarity Learning
w/o RIDTS	1.19×10^{4}	5.68×10^{3}	7.89×10^{2}	3.68×10^{3}
w/ RIDTS	4.55×10^{3}	2.60×10^{3}	6.67×10^{2}	1.21×10^{3}

We chose the number of optimization steps in the inner-loop as $T = 5$, $T = 10$, $T = 15$ and $T = 20$. We find that, 'w/o RIDTS' suffers from exploding meta-gradients, and the severity of the problem escalates for larger T. In contrast, meta-gradients of 'w/ RIDTS' show stable behavior and smaller norms, regardless of the duration of optimization in the inner-loop. Even when $T = 20$, meta-gradients of 'w/ RIDTS' have smaller and more stable norms as compared to meta-gradients of 'w/o RIDTS' with $T = 5$. This confirms our analysis that products of Hessian matrices and products of derivatives of retraction in meta-gradients of 'w/o RIDTS' cause the exploding gradients. Our Riemannian implicit differentiation training scheme avoids exploding gradients by breaking the dependency of optimization trajectories in training the Riemannian optimizer.

Optimization/Accuracy Performance. Finally, we compared our Riemannian implicit differentiation training scheme with existing meta-optimization training techniques [52], [70]. For the work in [52], we pretrained an initial optimizer guided by RSGD and gradually increased the iterations for the optimizer (50 steps every 2000 epochs). We evaluated its performance with different optimization steps in the inner-loop, denoted by 'w/o RIDTS IL-T', where T means the number of inner-loop steps. Some curves are not presented (e.g., 'w/o RIDTS IL-16' and 'w/o RIDTS IL-64' on the PCA task), because the method of [52] suffered from exploding gradients. Following ideas in [70], we utilized the persistent evolution strategies to compute meta-gradients in Riemannian meta-optimization denoted by 'w/o RIDTS PES'. Loss curves are shown in Figs. 14 and 15. Accuracy comparisons are shown in Table 6.

For the optimization performance, our training scheme achieves better performance with faster convergence speed and lower loss values. Although the method [52] uses a pretrained model that is capable of alleviating the instability of training, we observed that it could still suffer from exploding gradients in the Riemannian setting. We also note that in terms of accuracy, the proposed training scheme is superior. The method [70] uses evolution strategies to compute unbiased meta-gradients. Compared with it, our training scheme is more efficient and achieves higher accuracy.

6.4 Visualization

We trained an optimizer on the PCA task and visualized generated step sizes and search directions to show how our optimizer works. The step sizes generated by our optimizer are shown in Fig. 16a. We find that our optimizer learns to

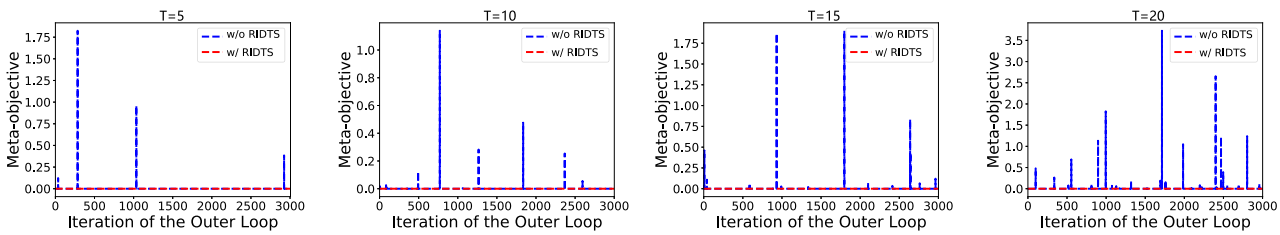


Fig. 13. Meta-gradients of RMO with and without the Riemannian implicit differentiation training scheme on the PCA task, denoted by 'w/ RIDTS' and 'w/o RIDTS', respectively. We evaluated meta-gradients with the number T of optimization steps in an inner-loop as 5, 10, 15, and 20.

automatically decrease step sizes, performing adaptive optimization. This is consistent with our common practice of tuning step sizes for a hand-designed optimizer.

We plotted cosine similarities between generated search directions and full gradients, as shown in Fig. 16b. Note that the full gradients are the optimal optimization directions as they were computed using all training data. We also plotted cosine similarities between stochastic gradients (inputs of our optimizer) and the full gradients. To further compare the generated search directions and the stochastic gradients, we projected the generated search directions, the stochastic gradients, and the full gradients to the Grassmann manifold. We then computed distances between the generated search directions and the full gradients, and distances between the stochastic gradients and the full gradients on the Grassmann manifold using the projection metric [71]. Results are shown in Fig. 16c. In Figs. 16b and 16c, compared with the stochastic gradients, the generated search directions by our method have smaller deviations from the optimal optimization directions (i.e., larger cosine similarities and smaller distances). This advantage contributes to our faster convergence speed. Thus, we conclude that, by exploring the underlying data distributions, our optimizer learns to automatically transform stochastic gradients and previous optimization states into better search directions.

We also visualized norms of search directions, as shown in Fig. 16d. The generated search directions have larger norms than stochastic gradients, especially at the beginning of training. Based on the above visualization experiments, the proposed optimizer can learn to generate adaptive step sizes and better search directions that have larger norms, implying a better optimization behaviour.

6.5 Applications

In this work, we focus on Riemannian optimization to address machine learning tasks with nonlinear constraints, e.g., dimensionality reduction [11], subspace learning [13], metric learning [72], and representation learning [73]. Naturally, our method can be applied to many practical constrained tasks. To demonstrate this point, we evaluate our method on the few-shot learning, texture image classifications, and person re-identification tasks.

6.5.1 Few-Shot Learning

The few-shot learning problems aim to recognize samples from unseen classes, given very few labeled examples. Some recent works show that representing data on a hyperbolic manifold $\boldsymbol{x} \in \mathcal{H}^d$ may lead to better performances. Among various properties of the hyperbolic manifold, its low encoding distortion might be of essential for representation learning [72]. Here, we build a hyperbolic classifier that projects euclidean features onto the hyperbolic manifold and classify them via the hyperbolic distance following [73]. In our experiments, we learn a hyperbolic optimizer to initialize the classifier for few-shot learning. If we remove the hyperbolic structure, our model reduces to the celebrated MAML method [31].

We conducted 1-shot/5-shot 5-way experiments on two popular datasets: mini-ImageNet [74] and tiered-ImageNet [75]. We used the 12-layer residual network (ResNet12 [76]) as the backbone. We compared the loss values of RMO with those of hand-designed Riemannian optimizers RSGD, RSGDM, and RASA in Fig. 17. The figure shows that our method has faster convergence speed and smaller loss values for new tasks. We compared the accuracy of our model with

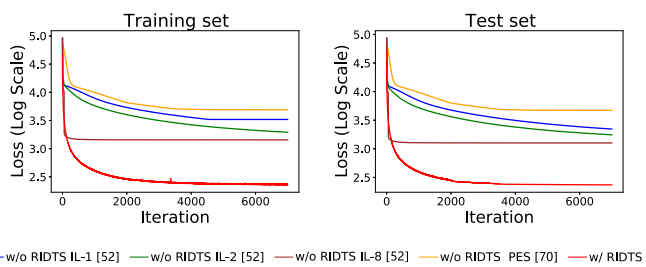


Fig. 14. Comparisons of learning the Riemannian optimizer with and without Riemannian implicit differentiation on the PCA task.

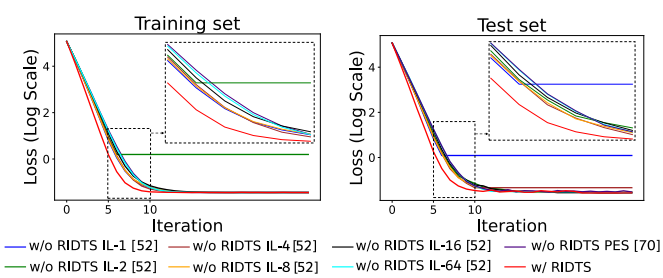
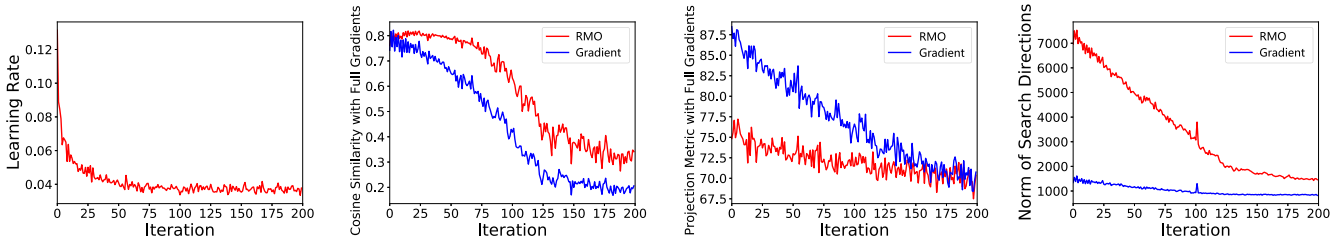


Fig. 15. Comparisons of learning the Riemannian optimizer with and without Riemannian implicit differentiation on the clustering task.

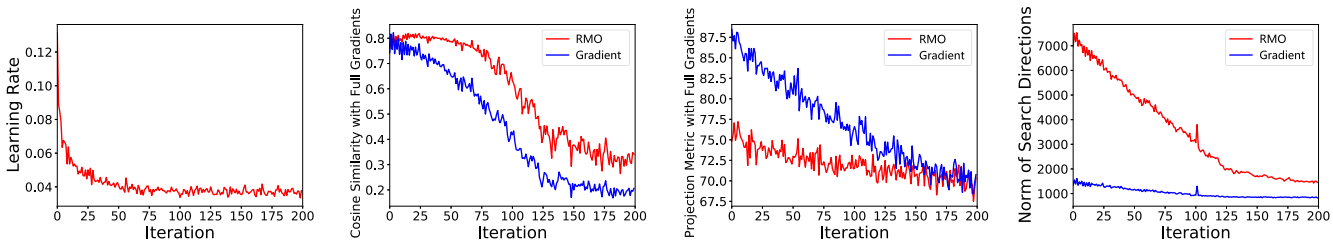
TABLE 6
Ablation on the Kylberg Dataset

Method	w/o RIDTS IL [52]	w/o RIDTS PES [70]	RMO (Ours)
Kylberg	84.15	85.00	**86.70**



(a) Step size (b) Cosine similarity of search direction (c) Projection metric of search direction (d) Norm of search direction

Fig. 16. Visualization of generated step sizes and search directions on the PCA task. (a) Generated step sizes of the proposed optimizer. (b) The red curve denotes cosine similarities between the generated search directions and full gradients, and the blue curve denotes cosine similarities between stochastic gradients and full gradients. (c) The projection metric between the generated search directions and the full gradients. (d) Norms of the generated search directions and stochastic gradients.



(a) Step size (b) Cosine similarity of search direction (c) Projection metric of search direction (d) Norm of search direction

Fig. 17. Loss plots for the few-shot learning task on the mini-ImageNet and tiered-ImageNet datasets.

state-of-the-art few-shot learning methods in Tables 7 and 8. Using a hyperbolic geometry brings clear performance boost as compared to the euclidean geometry. For example, our method has at least 10% improvement over MAML and about 5% improvement over ALFA. This is because the hyperbolic geometry can provide more power representations for data. Our optimizer learns adaptive optimization for unseen tasks, further improving the performance. For example, on the mini-ImageNet dataset, our method is 1.34% and 1.40% higher than the compared method RSGDM. Hyper ProtoNet learns a hyperbolic prototype for few-shot learning, and our method is 6.47% and 5.76% higher than it on the mini-ImageNet dataset. These experiments demonstrate the superiority of our learned optimizer.

6.5.2 *Texture Image Classification*

Texture classification, a fundamental problem in computer vision, has been a long-standing research topic with a wide variety of applications such as image retrieval. We add the orthogonality constraint to the classifier for discriminative representations of texture images. This makes parameters of the model on Stiefel manifolds. In our experiments, the DTD [77] dataset was used. We extracted 1024-dimensional second-order statistics as image representations from a VGG-16 network via a factorized bilinear pooling [78]. We trained our optimizer on the training set and then utilized the optimizer to learn a powerful classifier. Accuracy comparisons with the dimensionality of features are shown in Table 9.

TABLE 7
Accuracy (%) on the Mini-ImageNet Dataset

Method	Backbone	1-shot 5-way	5-shot 5-way
MAML [31]	ResNet12	51.03 ± 0.50	68.26 ± 0.47
L2F [79]	ResNet12	57.48 ± 0.49	74.68 ± 0.43
CAML [80]	ResNet12	59.23 ± 0.99	72.35 ± 0.71
ALFA [81]	ResNet12	60.06 ± 0.49	77.42 ± 0.42
MetaOptNet [76]	ResNet12	62.64 ± 0.61	78.63 ± 0.46
MetaFun [82]	ResNet12	62.12 ± 0.30	78.20 ± 0.16
DSN [83]	ResNet12	62.64 ± 0.66	78.83 ± 0.45
Chen et al. [84]	ResNet12	63.17 ± 0.23	79.26 ± 0.17
MeTAL [85]	ResNet12	59.64 ± 0.38	76.20 ± 0.19
LEO [35]	WRN-28-10	61.76 ± 0.08	77.59 ± 0.12
Hyper ProtoNet [73]	ResNet18	59.47 ± 0.20	76.84 ± 0.14
RSGD [20]	ResNet12	63.38 ± 0.36	80.10 ± 0.54
RSGDM [22]	ResNet12	64.60 ± 0.33	81.20 ± 0.68
RASA [25]	ResNet12	62.56 ± 0.43	78.90 ± 0.63
RMO (Ours)	ResNet12	$\mathbf{65.94 \pm 0.42}$	$\mathbf{82.60 \pm 0.39}$

TABLE 8
Accuracy (%) on the Tiered-ImageNet Dataset

Method	Backbone	1-shot 5-way	5-shot 5-way
ProtoNet [86]	ResNet12	53.51 ± 0.89	72.69 ± 0.74
MAML [31]	ResNet12	58.58 ± 0.49	71.24 ± 0.43
L2F [79]	ResNet12	63.94 ± 0.48	77.61 ± 0.41
ALFA [81]	ResNet12	64.43 ± 0.49	81.77 ± 0.39
DSN [83]	ResNet12	66.22 ± 0.75	82.79 ± 0.48
MetaOptNet [76]	ResNet12	65.99 ± 0.72	83.28 ± 0.12
MetaFun [82]	ResNet12	67.72 ± 0.14	78.20 ± 0.16
Chen et al. [84]	ResNet12	68.62 ± 0.27	83.74 ± 0.18
MeTAL [85]	ResNet12	63.89 ± 0.43	80.14 ± 0.40
LEO [35]	WRN-28-10	66.33 ± 0.05	81.44 ± 0.09
RSGD [20]	ResNet12	70.06 ± 0.66	84.44 ± 0.27
RSGDM [22]	ResNet12	69.56 ± 0.52	84.56 ± 0.41
RASA [25]	ResNet12	68.2 ± 0.47	83.4 ± 0.38
RMO (Ours)	ResNet12	$\mathbf{71.40 \pm 0.46}$	$\mathbf{85.40 \pm 0.20}$

TABLE 9
Accuracies (%) on the DTD Dataset

Method	Feature dim.	DTD
B-CNN [9]	2.6×10^5	67.5
CBP [90]	8192	67.7
LRBP [91]	100	65.8
MPN [92]	32896	68.0
FBN [93]	-	67.8
SMSO [94]	2048	69.3
FBC [78]	1024	70.7
iSQRT-COV [8]	2048	70.6
w/o constraint	1024	69.3
RMO (Ours)	1024	**71.2**

Results show that our method achieves state-of-the-art performance with smaller dimensionality. All the studied methods use the second-order statistics for classification. The difference is the use of a classifier with the orthogonality constraint in our algorithm instead of conventional softmax classifier employed by the studied works. Our method achieves better performance than all compared methods. This shows the superiority of adding the orthogonality constraint. A classifier with orthogonality constraint requires preserving orthogonality in the training process, which makes training challenging and unconventional. Our RMO method quite meets this requirement, and learns to automatically perform optimization on Stiefel manifolds. Since the classifier with the orthogonality constraint can also be applied to other classification tasks, such as face and fine-grained image recognition, our method has wide use-cases in real-world applications.

6.5.3 Person Re-Identification

Person re-identification has attracted much attention in computer vision due to its industrial prospects. The goal here is to retrieve all images with the same identity from a gallery, given an image of a specific person. The key to successful person re-identification is to learn discriminative features and a metric, such that features from the same identity have small distances and features from different identities have large distances. In our experiment, we extracted global and part features with bilinear attention [87], and concatenated them into 1024-dimensional features, where the GoogLeNet-V1 was used as the backbone. We applied a projection with the orthogonality constraint on the features, such that the resulting features become more independent and informative. We used the Market-1501 [88] and DukeMTMC-reID [89] datasets in our experiments, and report the mean average precision (mAP) and rank-1/5/10 accuracies to evaluate our model. Accuracy comparisons are shown in Table 10.

We can find that the projection with the orthogonality constraint is more discriminative than the conventional projection in euclidean spaces. In terms of mAP, our method achieves 86.1 on the market-1501 dataset, while 'w/o constraint' is just 82.7. As shown empirically, it is reasonable to apply our method to the person re-identification task. Our method not only reduces human involvement but also brings better performance. This also demonstrates that our method has wide applications, and can be readily applied to orthogonal projections to bring discriminative and informative features.

TABLE 10
Evaluation on the Market-1501 and DukeMTMC-reID Datasets

Method	Market-1501				DukeMTMC-reID			
	mAP	R-1	R-5	R-10	mAP	R-1	R-5	R-10
SVDNet [72]	62.1	82.3	92.3	95.2	56.8	76.7	86.4	89.9
DKPM [95]	75.3	90.1	96.7	97.9	63.2	80.3	89.5	91.9
HA-CNN [96]	75.7	91.2	-	-	63.8	80.5	-	-
PBR [97]	79.6	91.7	96.9	98.1	64.2	82.1	-	-
DuATM [98]	76.6	91.4	97.1	-	64.6	81.8	90.2	-
Mancs [99]	82.3	93.1	-	-	71.8	84.9	-	-
SGGNN [100]	82.8	92.3	96.1	97.4	68.2	81.1	88.4	91.2
HPM [101]	82.7	94.2	97.5	98.5	74.3	86.6	-	-
IANet [102]	83.1	94.4	-	-	73.4	87.1	-	-
AANet [103]	83.4	93.9	-	98.5	74.3	87.6	-	-
OSNet [104]	84.9	**94.8**	-	-	73.5	**88.6**	-	-
w/o constraint	82.7	92.1	97.3	98.5	72.6	84.8	93.0	94.7
RMO (Ours)	**86.1**	93.6	**98.1**	**99.3**	**76.4**	86.7	**93.9**	**95.3**

7 Conclusion

In this article, we have presented a Riemannian meta-optimization (RMO) method to learn an optimizer on Riemannian manifolds, reducing human involvement and expert knowledge in employing the Riemannian optimizer by hand. The introduced recurrent neural network, namely gmLSTM, enables us to develop the optimizer that is faithful to the geometry of the manifold. The Riemannian implicit differentiation training scheme can efficiently learning the Riemannian optimizer, avoiding the exploding gradients, and reducing both time and memory costs. We train an optimizer to minimize the objective of a base-learner in an end-to-end fashion, and thus the optimizer can explore the underlying data distribution to perform task-specific optimization. Extensive experiments on the hyperbolic, Stiefel, Grassmann, and SPD manifolds have demonstrated that our optimizer can achieve better convergence behavior than existing state-of-the-art Riemannian optimizers consistently. Developing theoretical justification of learning Riemannian optimization is our future work.

References

[1] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex Optimization*. Cambridge, U.K.: Cambridge University Press, 2004.

[2] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization methods for large-scale machine learning," *SIAM Rev.*, vol. 60, no. 2, pp. 223–311, 2018.

[3] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–23.

[4] A. Cherian, P. Stanitsas, M. Harandi, V. Morellas, and N. Papanikolopoulos, "Learning discriminative ab-divergences for positive definite matrices," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2017, pp. 4270–4279.

[5] M. Harandi, M. Salzmann, and R. Hartley, "Joint dimensionality reduction and metric learning: A geometric take," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1404–1413.

[6] A. Cherian and S. Sra, "Riemannian dictionary learning and sparse coding for positive definite matrices," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 28, no. 12, pp. 2859–2871, Dec. 2017.

[7] R. Hosseini and S. Sra, "Matrix manifold optimization for Gaussian mixtures," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 910–918.

[8] Q. Wang, J. Xie, W. Zuo, L. Zhang, and P. Li, "Deep CNNs meet global covariance pooling: Better representation and generalization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 8, pp. 2582–2597, Aug. 2021.

[9] T.-Y. Lin, A. RoyChowdhury, and S. Maji, "Bilinear convolutional neural networks for fine-grained visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 6, pp. 1309–1322, Jun. 2018.

[10] J. Zhang, L. Wang, L. Zhou, and W. Li, "Beyond covariance: Sice and kernel based visual feature representation," *Int. J. Comput. Vis.*, vol. 129, pp. 300–320, 2021.

[11] J. Yuan and A. Lamperski, "Online adaptive principal component analysis and its extensions," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7213–7221.

[12] A. Hyvärinen and E. Oja, "Independent component analysis: Algorithms and applications," *Neural Netw.*, vol. 13, no. 4/5, pp. 411–430, 2000.

[13] Z. Huang, R. Wang, S. Shan, L. Van Gool, and X. Chen, "Cross euclidean-to-riemannian metric learning with application to face recognition from video," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 12, pp. 2827–2840, Dec. 2018.

[14] R. Chakraborty, L. Yang, S. Hauberg, and B. Vemuri, "Intrinsic Grassmann averages for online linear, robust and nonlinear subspace learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 11, pp. 3904–3917, Nov. 2021.

[15] W. Dai, O. Milenkovic, and E. Kerman, "Subspace evolution and transfer (SET) for low-rank matrix completion," *IEEE Trans. Signal Process.*, vol. 59, no. 7, pp. 3120–3132, Jul. 2011.

[16] R. Chakraborty, J. Bouza, J. Manton, and B. C. Vemuri, "ManifoldNet: A deep neural network for manifold-valued data with applications," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 2, pp. 799–810, Feb. 2022.

[17] T. Lin and H. Zha, "Riemannian manifold learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 5, pp. 796–809, May 2008.

[18] X. Pennec, P. Fillard, and N. Ayache, "A riemannian framework for tensor computing," *Int. J. Comput. Vis.*, vol. 66, no. 1, pp. 41–66, 2006.

[19] N. T. Trendafilov, "P.-A. absil, R. mahony, and R. sepulchre. optimization algorithms on matrix manifolds," *Found. Comput. Math.*, vol. 10, no. 2, pp. 241–244, 2010.

[20] S. Bonnabel, "Stochastic gradient descent on Riemannian manifolds," *IEEE Trans. Autom. Control*, vol. 58, no. 9, pp. 2217–2229, Sep. 2013.

[21] H. Zhang, S. J. Reddi, and S. Sra, "Riemannian SVRG: Fast stochastic optimization on Riemannian manifolds," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 4592–4600.

[22] S. Kumar Roy, Z. Mhammedi, and M. Harandi, "Geometry aware constrained optimization techniques for deep learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4460–4469.

[23] H. Sato, H. Kasai, and B. Mishra, "Riemannian stochastic variance reduced gradient algorithm with retraction and vector transport," *SIAM J. Optim.*, vol. 29, no. 2, pp. 1444–1472, 2019.

[24] G. Bécigneul and O.-E. Ganea, "Riemannian adaptive optimization methods," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–16.

[25] H. Kasai, P. Jawanpuria, and B. Mishra, "Riemannian adaptive stochastic gradient algorithms on matrix manifolds," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 3262–3271.

[26] H. Zhang and S. Sra, "Towards riemannian accelerated gradient methods," 2018, *arXiv:1806.02812*.

[27] F. Alimisis, A. Orvieto, G. Becigneul, and A. Lucchi, "Practical accelerated optimization on Riemannian manifolds," 2020, *arXiv:2002.04144*.

[28] Y. Liu, F. Shang, J. Cheng, H. Cheng, and L. Jiao, "Accelerated first-order methods for geodesically convex optimization on Riemannian manifolds," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4868–4877.

[29] A. Gu, F. Sala, B. Gunel, and C. Ré, "Learning mixed-curvature representations in product spaces," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–21.

[30] M. Maher and S. Sakr, "SmartML: A meta learning-based framework for automated selection and hyperparameter tuning for machine learning algorithms," in *Proc. Int. Conf. Extending Database Technol.*, 2019, pp. 554–557.

[31] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1126–1135.

[32] Y. Cao, T. Chen, Z. Wang, and Y. Shen, "Learning to optimize in swarms," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 15 018–15 028.

[33] M. Andrychowicz et al., "Learning to learn by gradient descent by gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3981–3989.

[34] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–11.

[35] A. A. Rusu et al., "Meta-learning with latent embedding optimization," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–17.

[36] Z. Gao, Y. Wu, Y. Jia, and M. Harandi, "Learning to optimize on SPD manifolds," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 7697–7706.

[37] D. G. Luenberger, "The gradient projection method along geodesics," *Manage. Sci.*, vol. 18, no. 11, pp. 620–631, 1972.

[38] H. Kasai, H. Sato, and B. Mishra, "Riemannian stochastic recursive gradient algorithm," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 2521–2529.

[39] J. Zhang, H. Zhang, and S. Sra, "R-spider: A fast Riemannian stochastic optimization algorithm with curvature independent rate," 2018, *arXiv:1811.04194*.

[40] P.-A. Absil, C. G. Baker, and K. A. Gallivan, "Trust-region methods on Riemannian manifolds," *Found. Comput. Math.*, vol. 7, pp. 303–330, 2007.

[41] W. Huang, K. A. Gallivan, and P.-A. Absil, "A Broyden class of quasi-newton methods for Riemannian optimization," *SIAM J. Optim.*, vol. 25, pp. 1660–1685, 2015.

[42] H. Kasai, H. Sato, and B. Mishra, "Riemannian stochastic quasi-newton algorithm with variance reduction and its convergence analysis," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2018, pp. 269–278.

[43] H. Kasai and B. Mishra, "Inexact trust-region algorithms on Riemannian manifolds," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 4254–4265.

[44] J. Hu, A. Milzarek, Z. Wen, and Y.-X. Yuan, "Adaptive quadratically regularized newton method for Riemannian optimization," *SIAM J. Matrix Anal. Appl.*, vol. 39, pp. 1181–1207, 2018.

[45] Z. Li, F. Zhou, F. Chen, and H. Li, "Meta-SGD: Learning to learn quickly for few-shot learning," 2017, *arXiv:1707.09835*.

[46] I. Bello, B. Zoph, V. Vasudevan, and Q. V. Le, "Neural optimizer search with reinforcement learning," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 459–468.

[47] X. Xie, J. Wu, G. Liu, Z. Zhong, and Z. Lin, "Differentiable linearized ADMM," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 6902–6911.

[48] X. Chen, J. Liu, Z. Wang, and W. Yin, "Theoretical linear convergence of unfolded ISTA and its practical weights and thresholds," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, pp. 9061–9071.

[49] R. Liu, S. Cheng, Y. He, X. Fan, Z. Lin, and Z. Luo, "On the convergence of learning-based iterative methods for nonconvex inverse problems," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 12, pp. 3027–3039, Dec. 2020.

[50] O. Wichrowska et al., "Learned optimizers that scale and generalize," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3751–3760.

[51] L. Metz, N. Maheswaranathan, J. Nixon, D. Freeman, and J. Sohl-Dickstein, "Understanding and correcting pathologies in the training of learned optimizers," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 4556–4565.

[52] T. Chen et al., "Training stronger baselines for learning to optimize," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 7332–7343.

[53] Y. Bengio, "Gradient-based optimization of hyperparameters," *Neural Comput.*, vol. 12, no. 8, pp. 1889–1900, 2000.

[54] A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine, "Meta-learning with implicit gradients," in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 113–124.

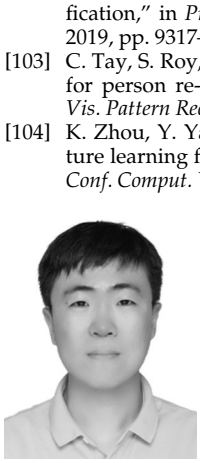
[55] R. Liao et al., "Reviving and improving recurrent back-propagation," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 3082–3091.

[56] D. Gudovskiy, L. Rigazio, S. Ishizaka, K. Kozuka, and S. Tsukizawa, "Autodo: Robust autoaugment for biased data with label noise via scalable probabilistic implicit differentiation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 16 601–16 610.

[57] J. M. Lee, *Riemannian Manifolds: An Introduction to Curvature*. Berlin, Germany: Springer, 2006, vol. 176.

[58] J. W. Cannon et al., "Hyperbolic geometry," *Flavors Geometry*, vol. 31, no. 59–115, 1997, Art. no. 2.

[59] A. Edelman, T. A. Arias, and S. T. Smith, "The geometry of algorithms with orthogonality constraints," *SIAM J. Matrix Anal. Appl.*, vol. 20, no. 2, pp. 303–353, 1998.
[60] P.-A. Absil, R. Mahony, and R. Sepulchre, "Riemannian geometry of Grassmann manifolds with a view on algorithmic computation," *Acta Applicandae Mathematica*, vol. 80, no. 2, pp. 199–220, 2004.
[61] M. Harandi, M. Salzmann, and R. Hartley, "Dimensionality reduction on SPD manifolds: The emergence of geometry-aware methods," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 1, pp. 48–62, Jan. 2018.
[62] K. Li and J. Malik, "Learning to optimize," in *Proc. Int. Conf. Learn. Representations*, 2016, pp. 1–13.
[63] J. Lorraine, P. Vicol, and D. Duvenaud, "Optimizing millions of hyperparameters by implicit differentiation," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2020, pp. 1540–1552.
[64] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: A survey," *J. Mach. Learn. Res.*, vol. 18, pp. 153:1–153:43, 2017.
[65] C. Ionescu, O. Vantzos, and C. Sminchisescu, "Matrix backpropagation for deep networks with structured layers," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2015, pp. 2965–2973.
[66] A. Griewank, "Some bounds on the complexity of gradients, jacobians, and hessians," in *Complexity in Numerical Optimization*. Singapore: World Scientific, 1993, pp. 128–162.
[67] G. Kylberg, "The kylberg texture dataset v. 1.0," Centre Image Anal., Swedish Univ., Agricultural Sci. Uppsala Univ., Uppsala, Sweden, Tech. Rep. (Blue series) 35, Sep. 2011. [Online]. Available: //www.cb.uu.se/~gustaf/texture/
[68] K.-C. Lee, J. Ho, and D. J. Kriegman, "Acquiring linear subspaces for face recognition under variable lighting," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 27, no. 5, pp. 684–698, May 2005.
[69] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The caltech-UCSD birds-200–2011 dataset," California Institute of Technology, Tech. Rep. CNS-TR-2011-001, 2011.
[70] P. Vicol, L. Metz, and J. Sohl-Dickstein, "Unbiased gradient estimation in unrolled computation graphs with persistent evolution strategies," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 10 553–10 563.
[71] J. Ham and D. Lee, "Grassmann discriminant analysis: A unifying view on subspace-based learning," in *Proc. Int. Conf. Mach. Learn.*, 2008, pp. 376–383.
[72] Y. Sun, L. Zheng, W. Deng, and S. Wang, "SVDNet for pedestrian retrieval," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2017, pp. 3820–3828.
[73] V. Khrulkov, L. Mirvakhabova, E. Ustinova, I. Oseledets, and V. Lempitsky, "Hyperbolic image embeddings," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 6417–6427.
[74] O. Vinyals et al., "Matching networks for one shot learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3630–3638.
[75] M. Ren et al., "Meta-learning for semi-supervised few-shot classification," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–15.
[76] K. Lee, S. Maji, A. Ravichandran, and S. Soatto, "Meta-learning with differentiable convex optimization," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 10 649–10 657.
[77] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi, "Describing textures in the wild," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2014, pp. 3606–3613.
[78] Z. Gao, Y. Wu, X. Zhang, J. Dai, Y. Jia, and M. Harandi, "Revisiting bilinear pooling: A coding perspective," in *Proc. AAAI Conf. Artif. Intell.*, 2020, pp. 3954–3961.
[79] S. Baik, S. Hong, and K. M. Lee, "Learning to forget for meta-learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 2376–2384.
[80] X. Jiang, M. Havaei, F. Varno, G. Chartrand, N. Chapados, and S. Matwin, "Learning to learn with conditional class dependencies," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–11.
[81] S. Baik, M. Choi, J. Choi, H. won Kim, and K. M. Lee, "Meta-learning with adaptive hyperparameters," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, pp. 20 755–20 765.
[82] J. Xu, J. Ton, H. Kim, A. R. Kosiorek, and Y. W. Teh, "MetaFun: Meta-learning with iterative functional updates," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 10 617–10 627.
[83] C. Simon, P. Koniusz, R. Nock, and M. Harandi, "Adaptive subspaces for few-shot learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 4136–4145.
[84] Y. Chen, Z. Liu, H. Xu, T. Darrell, and X. Wang, "Meta-baseline: Exploring simple meta-learning for few-shot learning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 9062–9071.
[85] S. Baik, J. Choi, H. Kim, D. Cho, J. Min, and K. M. Lee, "Meta-learning with task-adaptive loss function for few-shot learning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 9465–9474.
[86] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2017, pp. 4077–4087.
[87] P. Fang, J. Zhou, S. K. Roy, P. Ji, L. Petersson, and M. T. Harandi, "Attention in attention networks for person retrieval," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 9, pp. 4626–4641, Sep. 2022.
[88] L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang, and Q. Tian, "Scalable person re-identification: A benchmark," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2015, pp. 1116–1124.
[89] E. Ristani, F. Solera, R. S. Zou, R. Cucchiara, and C. Tomasi, "Performance measures and a data set for multi-target, multi-camera tracking," in *Proc. Eur. Conf. Comput. Vis. Workshops*, 2016, pp. 17–35.
[90] Y. Gao, O. Beijbom, N. Zhang, and T. Darrell, "Compact bilinear pooling," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 317–326.
[91] S. Kong and C. C. Fowlkes, "Low-rank bilinear pooling for fine-grained classification," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 7025–7034.
[92] P. Li, J. Xie, Q. Wang, and W. Zuo, "Is second-order information helpful for large-scale visual recognition?," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2017, pp. 2089–2097.
[93] Y. Li, N. Wang, J. Liu, and X. Hou, "Factorized bilinear models for image recognition," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2017, pp. 2098–2106.
[94] K. Yu and M. Salzmann, "Statistically-motivated second-order pooling," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 621–637.
[95] Y. Shen, T. Xiao, H. Li, S. Yi, and X. Wang, "End-to-end deep Kronecker-product matching for person re-identification," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 6886–6895.
[96] W. Li, X. Zhu, and S. Gong, "Harmonious attention network for person re-identification," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2285–2294.
[97] Y. Suh, J. Wang, S. Tang, T. Mei, and K. M. Lee, "Part-aligned bilinear representations for person re-identification," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 418–437.
[98] J. Si et al., "Dual attention matching network for context-aware feature sequence based person re-identification," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 5363–5372.
[99] C. Wang, Q. Zhang, C. Huang, W. Liu, and X. Wang, "Mancs: A multi-task attentional network with curriculum sampling for person re-identification," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 384–400.
[100] Y. Shen, H. Li, S. Yi, D. Chen, and X. Wang, "Person re-identification with deep similarity-guided graph neural network," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 508–526.
[101] Y. Fu et al., "Horizontal pyramid matching for person re-identification," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 8295–8302.
[102] R. Hou, B. Ma, H. Chang, X. Gu, S. Shan, and X. Chen, "Interaction-and-aggregation network for person re-identification," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 9317–9326.
[103] C. Tay, S. Roy, and K. Yap, "AANet: Attribute attention network for person re-identifications," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 7134–7143.
[104] K. Zhou, Y. Yang, A. Cavallaro, and T. Xiang, "Omni-scale feature learning for person re-identification," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 3701–3711.



Zhi Gao (Member, IEEE) received the BS degree from the School of Computer Science, Beijing Institute of Technology, China, in 2017. He is currently working toward the PhD degree in the School of Computer Science, Beijing Institute of Technology. His research interests include pattern recognition, machine learning, computer vision, and Riemannian geometry.

Yuwei Wu (Member, IEEE) received the PhD degree in computer science from the Beijing Institute of Technology (BIT), Beijing, China, in 2014. He is now an associate professor with the School of Computer Science, BIT. From August 2014 to August 2016, he was a postdoctoral research fellow with the School of Electrical & Electronic Engineering (EEE), Nanyang Technological University (NTU), Singapore. He has strong research interests in computer vision and machine learning.

Xiaomeng Fan received the BS degree in information and computing science from the Hebei University of Technology (HEBUT), Hebei, China, in 2020. He is currently working toward PhD degree with the School of Computer Science, Beijing Institute of Technology (BIT), Beijing, China. His research interests include Riemannian optimization, machine learning, and computer vision .

Mehrtash Harandi (Member, IEEE) is a associate professor with the Department of Electrical and Computer Systems Engineering, Monash University. He is also a contributing research scientist in the Machine Learning Research Group (MLRG) with Data61/CSIRO and an associated investigator with the Australian Center for Robotic Vision (ACRV). His current research interests include theoretical and computational methods in machine learning, computer vision, signal processing, and Riemannian geometry.

Yunde Jia (Member, IEEE) received the BS, MS, and PhD degrees from the Beijing Institute of Technology (BIT) in 1983, 1986, and 2000, respectively. He was a visiting scientist with the Robotics Institute, Carnegie Mellon University (CMU), from 1995 to 1997. He is currently a professor of Computer Science with BIT, and chair professor of Computer Science with Shenzhen MSU-BIT University. His interests include computer vision, computational perception and cognition, and intelligent systems.