# Riemannian Implicit Differentiation via a Fixed-Point Equation for Riemannian Bilevel Optimization

Xiaomeng Fan, Yuwei Wu, *Member, IEEE*, Zhi Gao, Zhipeng Lu, Feng Li, Mehrtash Harandi, *Member, IEEE*, and Yunde Jia, *Member, IEEE*

***Abstract*—Various Riemannian optimization tasks, such as Riemannian metaoptimization (RMO) and Riemannian metalearning, can be formulated as Riemannian bilevel optimization problems (i.e., the inner-level and outer-level optimization). Implicit differentiation has shown effectiveness in solving RMO, which decouples the computation of outer gradients from the inner-level process, avoiding huge computational burdens. However, extending implicit differentiation to other Riemannian bilevel optimization tasks is nontrivial because it requires much expert involvement for case-by-case derivations. In this article, we propose a Riemannian implicit differentiation method that provides a unified expression for outer gradients, leading to flexible application to other tasks with less expert involvement. Specifically, we formulate the inner-level optimization as a root-finding process of a fixed-point equation, through which the inner-level optimization among different tasks is formulated in a unified way. By differentiating the fixed-point equation, we derive a unified expression for outer gradients, circumventing the case-by-case derivations for different tasks. Then, we present convergence analysis and approximation error analysis, which guarantee the effectiveness of our method in various Riemannian optimization tasks. We further conduct experiments on multiple Riemannian optimization tasks, and the experimental results confirm the effectiveness.**

***Index Terms*—Hyperparameter optimization, implicit differentiation, metalearning, metaoptimization, Riemannian bilevel optimization.**

Received 13 November 2024; revised 29 June 2025; accepted 12 October 2025. This work was supported in part by the Natural Science Foundation of China (NSFC) under Grant 62172041, Grant 62176021, and Grant 62406009; in part by Shenzhen Science and Technology Program under Grant JCYJ20241202130548062; and in part by the Natural Science Foundation of Shenzhen under Grant JCYJ20230807142703006. *(Corresponding authors: Zhi Gao; Zhipeng Lu.)*

Xiaomeng Fan, Yuwei Wu, Zhi Gao, and Yunde Jia are with Beijing Laboratory of Intelligent Information Technology, School of Computer Science, Beijing Institute of Technology (BIT), Beijing 100081, China, and also with the Guangdong Laboratory of Machine Perception and Intelligent Computing, Shenzhen MSU-BIT University, Shenzhen 518172, China (e-mail: fanxiaomeng@bit.edu.cn; wuyuwei@bit.edu.cn; gaozhibit@bit.edu.cn; jiayunde@smbu.edu.cn).

Zhipeng Lu is with Guangdong Laboratory of Machine Perception and Intelligent Computing, Shenzhen MSU-BIT University, Shenzhen 518172, China (e-mail: zhipeng.lu@smbu.edu.cn).

Feng Li resides in Beijing 100081, China.

Mehrtash Harandi are with the Department of Electrical and Computer Systems Engineering, Monash University, Clayton, VIC 3800, Australia, and also with Data61-CSIRO, Clayton South, VIC 3169, Australia (e-mail: mehrtash.harandi@monash.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/TNNLS.2025.3624316, provided by the authors.

Digital Object Identifier 10.1109/TNNLS.2025.3624316

## I. Introduction

RIEMANNIAN bilevel optimization is a hierarchical mathematical paradigm where the outer-level optimization problem is enclosed in an inner-level optimization problem with manifold constraints [1]. Such a problem can be formulated as

$$\begin{aligned} &\min_{\boldsymbol{\theta}} \mathcal{F}(\boldsymbol{\theta}) := \mathcal{F}(\boldsymbol{X}^*(\boldsymbol{\theta}), \boldsymbol{\theta}) \\ &\text{s.t.} \quad \boldsymbol{X}^*(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{X}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{\theta}) \quad \text{and } \boldsymbol{X} \in \mathcal{M} \end{aligned} \tag{1}$$

where $\boldsymbol{X}$ are the inner-level parameters on the Riemannian manifolds $\mathcal{M}$, $\boldsymbol{\theta}$ are the outer-level parameters, $\mathcal{F}(\cdot)$ is the outer-level objective, and $\mathcal{L}(\cdot)$ is the inner-level objective. The goal of (1) is to minimize the outer-level objective function $\mathcal{F}(\boldsymbol{X}^*(\boldsymbol{\theta}), \boldsymbol{\theta})$, where $\boldsymbol{X}^*(\boldsymbol{\theta})$ is obtained by solving the inner-level optimization problem. Multiple Riemannian optimization tasks can be naturally formulated as bilevel optimization problems, such as Riemannian metaoptimization (RMO), Riemannian metalearning (RML), and Riemannian hyperparameter optimization (RHO) [2], [3], [4].

The key challenge in solving Riemannian bilevel optimization lies in the heavy computational burden, since computing the outer gradients requires differentiating through the whole procedure of the inner-level optimization. Recently, Fan et al. [5] proposed the efficient RMO (E-RMO) method that utilizes implicit differentiation, significantly reducing the computational burdens in Riemannian bilevel optimization. Briefly, there are three steps in the derivation of implicit differentiation. The first step is to find the stationary point $\boldsymbol{X}^*$ in the inner-level optimization. Then, $(d\boldsymbol{X}^*)/(d\boldsymbol{\theta})$ is derived by differentiating the optimal condition, where the gradients of the stationary point are zero. Finally, the outer gradients are computed by the chain rule.

However, applying this derivation of implicit differentiation to other Riemannian bilevel optimization tasks (such as RML and RHO) is nontrivial, as it requires much expert involvement.

1) It requires complex case-by-case derivations of the outer gradients for specific Riemannian optimization tasks. Due to the variability in inner-level optimization processes across different tasks, separate analyses are necessary for differentiating the optimal conditions to obtain the corresponding $(d\boldsymbol{X}^*)/(d\boldsymbol{\theta})$.
2) It requires convergence analysis to guarantee the effectiveness on specific tasks. Without the convergence

analysis, the general applicability of Riemannian bilevel optimization methods across diverse tasks cannot be guaranteed. Given the complexity of Riemannian bilevel optimization, applying the derivations by trial-and-error is expensive and labor-intensive.

In this article, we propose a Riemannian implicit differentiation method that provides a unified expression of outer gradients to address the above two challenges. Specifically, we formulate the inner-level optimization process as a root-finding process of a unified fixed-point equation on Riemannian manifolds. This, as we will show, boils down to defining an update function with the outer-level parameters, which takes Riemannian parameters as input and outputs the update vector in the inner-level optimization. The proposed fixed-point equation ensures that inner-level optimization across different tasks is formulated in a unified way. Then, by differentiating the fixed-point equation, we derive a unified expression for the outer gradient computation. As a result, the proposed method can be directly and easily applied to various Riemannian optimization tasks, without any requirements for case-by-case derivations.

The effectiveness of our method is demonstrated both theoretically and experimentally. Theoretically, we present the convergence analysis and approximation error analysis for the Riemannian implicit differentiation method. We provide an upper bound for the approximation error in the proposed method. We also prove that our algorithm can converge by bounding the gradients. These analyses theoretically guarantee the effectiveness of the proposed method across various Riemannian optimization tasks. To the best of our knowledge, this is the first convergence analysis for a Riemannian bilevel optimization method, providing a critical theoretical foundation for such methods. Empirically, evaluations on three Riemannian optimization tasks: RMO, RHO, and RML show that our method can achieve competitive (and even better) performance compared with existing bilevel optimization methods.

In summary, our contributions are threefold.

1) We propose a Riemannian implicit differentiation method for Riemannian bilevel optimization that provides a unified expression of outer gradients, avoiding the expert involvement for case-by-case derivations.
2) We model the inner-level optimization process as a root-finding process of a fixed-point equation, through which the inner-level optimization across different tasks is formulated in a unified way.
3) We provide the approximation error analysis and convergence analysis, which guarantees that our method can be effectively applied to various Riemannian optimization tasks.

## II. Related Work

### A. Riemannian Optimization

Gradient-based Riemannian optimization has emerged as a powerful paradigm for solving optimization problems under manifold constraints. The seminal work [6] introduces the Riemannian gradient descent (RGD) method, which employs Riemannian operations to comply with the constraints. Bonnabel [7] extended RGD to stochastic settings, leading to a surge of research in Riemannian optimization. Subsequent works either focus on manually crafting advanced Riemannian optimizers to enhance the convergence of Riemannian stochastic gradient descent (RSGD) or focus on automatically learning optimizers through data-driven approaches.

Riemannian accelerated gradient methods [8], [9] introduce acceleration techniques to Riemannian manifolds, leading to enhanced convergence performance. Riemannian stochastic gradient descent with momentum method [momentum Riemannian stochastic gradient descent (RSGDM)] [10] extends the idea of momentum from Euclidean spaces to Riemannian manifolds, which uses previous update information to speed up optimization. Riemannian variance-reduced gradient algorithms [11], [12], [13] focus on reducing the variance of stochastic gradients to improve the convergence of Riemannian optimization. Riemannian adaptive optimization algorithms [10], [14], [15] adaptively adjust learning rates and search directions based on the geometry of the Riemannian parameters, achieving convincing convergence performance. Riemannian conjugate gradient descent (RCGD) method [16] develops the conjugate gradient method to Riemannian manifolds, using parallel translation and vector transport operations. Decentralized RCGD method [17] introduces the first distributed RCGD algorithm. Besides, several works [18], [19], [20] focus on designing Riemannian min–max optimization methods to solve the Riemannian min–max optimization problem.

RMO method (RMM), first proposed by Gao et al. [3], aimed to automatically learn Riemannian optimizers in a data-driven way. Concretely, Gao et al. [3] introduces a matrix LSTM as the optimizer that takes gradients as input and generates step-sizes and search directions for symmetric positive definite (SPD) manifolds. However, training such optimizers brings the exploding gradient problem. To address this issue, they propose a gradient-free optimizer on tangent spaces for Riemannian optimization [21], which removes gradient computation with respect to Riemannian parameters in the inner-level optimization. Existing RMMs impose significant computational overhead due to the need for storage and differentiation through the entire inner-level optimization process. To solve this issue, they further introduce an E-RMO method that utilizes implicit differentiation, where the computation of the outer gradients is independent of the inner-level optimization process, thereby reducing the computational costs. While effective, implicit differentiation has not been applied to other Riemannian optimization tasks, and applying Riemannian implicit differentiation to other Riemannian tasks requires expert involvement for case-by-case derivation. In contrast, we propose a Riemannian implicit differentiation method with convergence guarantees, which removes the expensive and labor-intensive case-by-case derivation.

### B. Bilevel Optimization

The origin of bilevel optimization can be traced back to the Stackelberg competition [22]. Recently, various applications in the domain of machine learning and computer vision are formulated as bilevel optimization problems, including hyperparameter optimization [23], [24], metalearning [25], [26], neural architecture search [27], [28], generative adversarial learning [29], and deep reinforcement learning [30]. Bilevel

optimization methods can be broadly classified into three categories. Methods of the first category opt for investigating the convergence analysis of bilevel optimization methods. The first convergence analysis, presented by Ghadimi and Wang [31], investigates the convergence performance of the bilevel optimization method with the convex assumption. Subsequent studies extend this analysis to bilevel optimization methods to both deterministic and stochastic settings [32], [33]. The second category of methods concentrates on developing more robust bilevel optimization algorithms, addressing challenging cases such as the absence of inner-level singleton assumptions [34] and nonconvexity in the optimization problem [35].

Methods of the third category aim to overcome the notorious challenge in the bilevel optimization problem: short optimization iterations in the inner-level optimization process result in the bias issue, while long optimization iterations inevitably cause an intractable computational burden [36], [37]. Regarding the bias issue, the work [38] introduces the persistent evolution strategies method to obtain unbiased gradient estimates and proves the unbiased estimate of true gradients for quadratic losses. To address the computational burden, implicit differentiation provides an analytic expression for outer gradients, significantly reducing the time and memory complexities [39], [40]. Subsequent research has extended implicit differentiation to specific tasks such as automatic data generation [41] and auxiliary learning [42], as well as to more restrictive problems like linearly constrained [43] and lasso-type bilevel optimization problems [44]. Moreover, the Hessian-free method [45] is proposed to reduce training time by avoiding explicit Hessian computations, and later extended to handle nonconvex inner-level problems [46]. Besides, Liu et al. further propose a Moreau envelope-based Hessian-free method and provide the nonasymptotic convergence analysis [47]. In contrast, we focus on reducing the computational burden in Riemannian bilevel optimization, where parameters are endowed with Riemannian constraints. Besides, the proposed Riemannian implicit differentiation method can be directly applied to various Riemannian optimization tasks without the need for case-by-case expert-driven derivations.

## III. Background

### A. Riemannian Manifolds

A smooth Riemannian manifold $\mathcal{M}$ is a topological space that is locally Euclidean [6]. The tangent space to $\mathcal{M}$ at $\boldsymbol{X}$ denoted by $T_{\boldsymbol{X}}\mathcal{M}$ is the set of all tangent vectors to $\mathcal{M}$ at $\boldsymbol{X}$. In this article, we consider four popular manifolds, namely hyperbolic, Stiefel, Grassmann, and SPD manifolds.

*1) Hyperbolic Manifolds:* A hyperbolic manifold $\mathcal{H}^d$ is a smooth manifold with a constant negative curvature [48]. In this article, we choose the Poincaré ball model to work with, which is commonly used in the fields of image processing [49], [50]. It is defined as $\mathcal{H}^d = \{\boldsymbol{X} \in \mathbb{R}^d, \|\boldsymbol{X}\| < 1\}$ with the curvature being $-1$.

*2) Stiefel Manifolds:* A Stiefel manifold $\mathrm{St}(p,d)$ denotes the set of all $d \times p$ matrices with orthonormal columns [51]

$$\mathrm{St}(p,d) = \left\{\boldsymbol{X} \in \mathbb{R}^{d\times p} : \boldsymbol{X}^\top \boldsymbol{X} = \mathbf{I}_p\right\} \tag{2}$$

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

*3) Grassmann Manifolds:* A Grassmann manifold $\mathcal{G}(p,d)$ is formally defined as the quotient of the Stiefel manifold $\mathrm{St}(p,d)$ through the following equivalence class [52]:

$$[\boldsymbol{M}] = \{\boldsymbol{M}\boldsymbol{R} : \boldsymbol{M} \in \mathrm{St}(p,d), \boldsymbol{R} \in \mathcal{O}(p)\} \tag{3}$$

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

*4) SPD Manifolds:* A square matrix $\boldsymbol{X} \in \mathbb{R}^{d\times d}$ satisfying $\boldsymbol{X} = \boldsymbol{X}^\top$ and $\boldsymbol{v}^\top\boldsymbol{X}\boldsymbol{v} > 0\ \forall \boldsymbol{v} \in \mathbb{R}^d\backslash\{\mathbf{0}\}$ characterizes an SPD matrix [53], [54]. An SPD manifold $\mathrm{Sym}_d^+$ is consequently identified as a set of all $d \times d$ SPD matrices

$$\mathrm{Sym}_d^+ = \left\{\boldsymbol{X} \in \mathbb{R}^{d\times d} : \boldsymbol{X} = \boldsymbol{X}^\top, \boldsymbol{v}^\top\boldsymbol{X}\boldsymbol{v} > 0 \quad \forall \boldsymbol{v} \in \mathbb{R}^d \backslash \{\mathbf{0}\}\right\}. \tag{4}$$

### B. Riemannian Optimization

In general, the RGD update [7] is formulated as

$$\boldsymbol{X}^{(t+1)} = \Gamma_{\boldsymbol{X}^{(t)}}\left(-\xi \cdot \pi_{\boldsymbol{X}^{(t)}}\left(\frac{\partial \mathcal{L}\left(\boldsymbol{X}^{(t)}\right)}{\partial \boldsymbol{X}^{(t)}}\right)\right) \tag{5}$$

where $\boldsymbol{X}^{(t)}$ denotes the Riemannian parameters at time $t$, $\partial$ denotes the partial derivative, $(\partial\mathcal{L}(\boldsymbol{X}^{(t)}))/(\partial\boldsymbol{X}^{(t)})$ is the gradients with respect to $\boldsymbol{X}^{(t)}$ on Euclidean spaces, and $\xi$ is the step-size. Due to the nonlinear search space of Riemannian manifolds, the retraction operation $\Gamma_{\boldsymbol{X}^{(t)}}(\cdot)$ and orthogonal operation $\pi_{\boldsymbol{X}^{(t)}}(\cdot)$ are utilized to preserve the manifold constraint. Here, we give the definitions of two operations and their specific expressions on four manifolds.

*1) Orthogonal Projection:* The orthogonal projection $\pi_{\boldsymbol{X}^{(t)}}(\cdot) : \mathbb{R} \to T_{\boldsymbol{X}}\mathcal{M}$ takes Euclidean gradients as input, and outputs an update vector for the Riemannian parameters that are located on the tangent space $T_{\boldsymbol{X}}\mathcal{M}$.

*2) Retraction Operation:* The retraction operation $\Gamma_{\boldsymbol{X}}(\boldsymbol{P}) : T_{\boldsymbol{X}}\mathcal{M} \to \mathcal{M}, \boldsymbol{P} \in T_{\boldsymbol{X}}\mathcal{M}$ maps the tangent vector onto the manifold with a local rigidity condition [55].

### C. Riemannian Bilevel Optimization

Recall that Riemannian bilevel optimization is defined as

$$\begin{aligned} &\min_{\boldsymbol{\theta}} \mathcal{F}(\boldsymbol{\theta}) := \mathcal{F}(\boldsymbol{X}^*(\boldsymbol{\theta}), \boldsymbol{\theta}) \\ &\text{s.t.} \quad \boldsymbol{X}^*(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{X}} \mathcal{L}(\boldsymbol{X}, \boldsymbol{\theta}) \quad \text{and } \boldsymbol{X} \in \mathcal{M}. \end{aligned} \tag{6}$$

Here, we introduce three commonly used Riemannian tasks that are modeled as Riemannian bilevel optimization.

*1) Riemannian Hyperparameter Optimization:* The goal of RHO is to find optimal hyperparameters $\boldsymbol{\theta}$ that minimize the loss at the end of training on the validation set. In RHO, $\boldsymbol{\theta}$ are the hyperparameters (e.g., learning rate) and $\boldsymbol{X}$ are the weights of a network. In the RHO task, the inner-level update process is formulated as

$$\boldsymbol{X}^{(t+1)} = \Gamma_{\boldsymbol{X}^{(t)}}\left(-\xi \times \pi_{\boldsymbol{X}^{(t)}}\left(\frac{\partial \mathcal{L}\left(\boldsymbol{X}^{(t)}(\boldsymbol{\theta})\right)}{\partial \boldsymbol{X}^{(t)}(\boldsymbol{\theta})}\right)\right). \tag{7}$$

The outer-level parameters on the Euclidean spaces are optimized by the gradient descent algorithm

$$\boldsymbol{\theta}^{(t+1)} \leftarrow \boldsymbol{\theta}^{(t)} - e \times \frac{d\mathcal{F}}{d\boldsymbol{\theta}^{(t)}} \tag{8}$$

where $e$ denotes step size of the outer-level parameters $\boldsymbol{\theta}$.

*2) Riemannian Metaoptimization:* The goal of RMO [3] is to train a Riemannian optimizer $g_{\boldsymbol{\theta}}(\cdot)$ that is parameterized by a neural network, where $\boldsymbol{\theta}$ are parameters of the network, and $\boldsymbol{X}$ are Riemannian parameters to be optimized. In the RMO task, the inner-level optimization aims to utilize the learnable optimizer $g_{\boldsymbol{\theta}}(\cdot)$ to update the Riemannian parameters $\boldsymbol{X}$ according to

$$\boldsymbol{X}^{(t+1)} = \Gamma_{\boldsymbol{X}^{(t)}}\left(g_{\boldsymbol{\theta}}\left(\frac{\partial \mathcal{L}\left(\boldsymbol{X}^{(t)}(\boldsymbol{\theta})\right)}{\partial \boldsymbol{X}^{(t)}(\boldsymbol{\theta})}\right)\right). \tag{9}$$

RMO optimizes $\boldsymbol{\theta}$ in the outer-level optimization process via (8).

*3) Riemannian Metalearning:* The goal of the RML task is to train an initialization $\boldsymbol{\theta}$ that aggregates knowledge of various subtasks and can be trained to adapt to new subtasks. In the RML task, the inner-level optimization process aims to set $\boldsymbol{\theta}$ as initialization, and optimizes the inner-level parameters $\boldsymbol{X}$ as

$$\begin{cases} \boldsymbol{X}^{(1)} = \Gamma_{\boldsymbol{\theta}}\left(\pi_{\boldsymbol{\theta}}\left(-\xi \times \nabla \mathcal{L}(\boldsymbol{\theta})\right)\right), & \text{if} \quad t = 0 \\ \boldsymbol{X}^{(t+1)} = \Gamma_{\boldsymbol{X}^{(t)}}\left(\pi_{\boldsymbol{X}^{(t)}}\left(-\xi \times \dfrac{\partial \mathcal{L}\left(\boldsymbol{X}^{(t)}(\boldsymbol{\theta})\right)}{\partial \boldsymbol{X}^{(t)}(\boldsymbol{\theta})}\right)\right), & \text{if} \quad t > 0. \end{cases} \tag{10}$$

Similarly, RML optimizes $\boldsymbol{\theta}$ in the outer-level optimization process via (8).

### D. Analysis of Riemannian Bilevel Optimization Problem

The outer gradients need to unroll the whole inner-level optimization process, which brings a heavy computational burden since it depends on the whole inner-level optimization trajectory. Here, we take the RMO problem as an example to analyze. By the chain rule, the outer gradients are given by

$$\frac{d\mathcal{F}}{d\boldsymbol{\theta}} = \frac{d\mathcal{F}}{d\boldsymbol{X}^{(T)}} \sum_{k=1}^{T} \left( \frac{\partial \boldsymbol{X}^{(k)}}{\partial \boldsymbol{Y}^{(k-1)}} \cdot \frac{\partial \boldsymbol{Y}^{(k-1)}}{\partial \boldsymbol{\theta}} \prod_{l=k+1}^{T} \left( \frac{\partial \boldsymbol{X}^{(l)}}{\partial \boldsymbol{X}^{(l-1)}} + \frac{\partial \boldsymbol{X}^{(l)}}{\partial \boldsymbol{Y}^{(l-1)}} \cdot \frac{\partial \boldsymbol{Y}^{(l-1)}}{\partial \nabla \mathcal{L}\left(\boldsymbol{X}^{(l-1)}\right)} \cdot \frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{X}^2} \right) \right) \tag{11}$$

where $\boldsymbol{Y} = g_{\boldsymbol{\theta}}((\partial \mathcal{L})/(\partial \boldsymbol{X}))$.

Apparently, (11) includes products of Hessian matrix $(\partial^2 \mathcal{L})/(\partial \boldsymbol{X}^2)$ and partial derivatives $(\partial \boldsymbol{X}^{(l)})/(\partial \boldsymbol{X}^{(l-1)})$ and $(\partial \boldsymbol{X}^{(l)})/(\partial \boldsymbol{Y}^{(l-1)})$ of the retraction operation, over all inner-level optimization steps. This imposes heavy computation loads. Furthermore, (11) depends on the whole inner-level optimization path explicitly, which needs to be completely stored in memory. The time and space complexities of (11) are proportional to the inner-level optimization length (which will be proved in Section IV-D).

## IV. Method

In this section, we first review the E-RMO method [5], which explores the idea of implicit differentiation to solve the computational burden in RMO tasks. To reduce the human involvement in E-RMO, we propose the Riemannian implicit differentiation method that generalizes the idea to various Riemannian optimization tasks, avoiding case-by-case analyses and derivations.

### A. Implicit Differentiation in E-RMO

Implicit differentiation in the RMO task is introduced by decoupling the outer gradients from the inner-level optimization process [5]. Suppose that the exact solution $\boldsymbol{X}^*$ in the inner level, i.e., $(\partial \mathcal{L})/(\partial \boldsymbol{X}^*) = 0$. On the RMO task, based on its inner-level optimization process in (9), the result of differentiating $(\partial \mathcal{L})/(\partial \boldsymbol{X}^*) = 0$ is

$$\frac{\partial}{\partial \boldsymbol{\theta}}\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{X}^*}\right) = \frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{X}^{*2}} \frac{d\boldsymbol{X}^*}{d\boldsymbol{\theta}} = 0. \tag{12}$$

By utilizing the chain rule and the results in (12), one can derive the outer gradients, as shown in Proposition 1.

*Proposition 1:* On the RMO task, the outer gradients are given by

$$\frac{d\mathcal{F}}{d\boldsymbol{\theta}} = -\frac{\partial \mathcal{F}}{\partial \boldsymbol{X}'}\left(\frac{\partial \boldsymbol{X}^*}{\partial \boldsymbol{X}'} + \frac{\partial \boldsymbol{X}^*}{\partial \boldsymbol{Y}'}\frac{\partial \boldsymbol{Y}'}{\partial \frac{\partial \mathcal{L}}{\partial \boldsymbol{X}'}}\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{X}^{*2}}\right)^{-1} \cdot \frac{\partial \boldsymbol{X}^*}{\partial \boldsymbol{Y}'} \cdot \frac{d\boldsymbol{Y}'}{d\boldsymbol{\theta}} \tag{13}$$

where $\boldsymbol{X}'$ is the Riemannian parameters that are the results of previous iterative steps of $\boldsymbol{X}^*$, and $\boldsymbol{Y}' = g_{\boldsymbol{\theta}}((\partial \mathcal{L})/(\partial \boldsymbol{X}'))$.

The proof of Proposition 1 is provided in the supplementary materials.

Notably, the derived outer gradients of the RMO task in (13) are independent of the inner-level optimization process, thereby reducing the computational burden.

However, applying this derivation to other Riemannian tasks is nontrivial, as it causes much expert involvement for complex case-by-case derivations of the outer gradients on specific Riemannian tasks. We take the RHO as an example. On the RHO task, due to the change of the inner-level process in (7), the result of differentiating $(\partial \mathcal{L})/(\partial \boldsymbol{X}^*) = 0$ becomes

$$\frac{\partial}{\partial \boldsymbol{\theta}}\left(\frac{\partial \mathcal{L}}{\partial \boldsymbol{X}^*}\right) = \frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{X}^* \partial \boldsymbol{\theta}} + \frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{X}^{*2}}\frac{d\boldsymbol{X}^*}{d\boldsymbol{\theta}} = 0. \tag{14}$$

By arranging the results in (14) and utilizing the chain rule, the outer gradients can be derived, as shown in Proposition 2.

*Proposition 2:* On the RHO task, the outer gradients are given by

$$\frac{d\mathcal{F}}{d\boldsymbol{\theta}} = \frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}} - \frac{\partial \mathcal{F}}{\partial \boldsymbol{X}^*}\left(\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{X}^{*2}}\right)^{-1}\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{X}^* \partial \boldsymbol{\theta}}. \tag{15}$$

*1) Our Finding:* By comparing (7) and (9), as to different Riemannian optimization tasks, the inner-level optimization processes are various. Obviously, different inner-level optimization processes lead to different differentiation processes of $(d\mathcal{L})/(d\boldsymbol{X}^*) = 0$, as shown in (14) and (12). This causes outer gradients for various Riemannian optimization tasks to be different and case-specific. In other words, to apply the idea of implicit differentiation in the E-RMO method to other Riemannian optimization tasks, one needs case-by-case derivations.

### B. Riemannian Implicit Differentiation for Various Riemannian Tasks

To avoid the case-by-case derivations, we propose a Riemannian implicit differentiation method. We uniformly model the inner-level problem as a root-finding process of the fixed-point equation on the Riemannian manifolds. Specifically, we define an update function $f_{\boldsymbol{\theta}}(\cdot)$ with the outer-level parameters $\boldsymbol{\theta}$, which take Riemannian parameters $\boldsymbol{X}$ as input and generates the updated vector of $\boldsymbol{X}$ in the inner-level optimization. In doing so, we first compute the loss on the training set and obtain the gradients of Riemannian parameters $(\partial\mathcal{L})/(\partial\boldsymbol{X})$. Then, we compute the update vector $f_{\boldsymbol{\theta}}(\boldsymbol{X})$ according to $(\partial\mathcal{L})/(\partial\boldsymbol{X})$. For example, on the RMO task, the update function $f(\cdot)$ is actually realized as the learnable Riemannian optimizer $g_{\boldsymbol{\theta}}(\cdot)$

$$f_{\boldsymbol{\theta}}(\boldsymbol{X}) = g_{\boldsymbol{\theta}}\left(\frac{\partial\mathcal{L}}{\partial\boldsymbol{X}}\right). \tag{16}$$

As to other Riemannian optimization tasks, the update function $f(\cdot)$ is realized by orthogonal projection to project the gradients $(\partial\mathcal{L})/(\partial\boldsymbol{X})$ onto the tangent space

$$f_{\boldsymbol{\theta}}(\boldsymbol{X}) = -\xi \times \pi_{\boldsymbol{X}}\left(\frac{\partial\mathcal{L}(\boldsymbol{X}(\boldsymbol{\theta}))}{\partial\boldsymbol{X}(\boldsymbol{\theta})}\right). \tag{17}$$

In this way, the inner-level problem of Riemannian bilevel optimization can be modeled as

$$\boldsymbol{X}^{(t+1)} = \Gamma_{\boldsymbol{X}^{(t)}}\left(f_{\boldsymbol{\theta}}\left(\boldsymbol{X}^{(t)}\right)\right). \tag{18}$$

Suppose $\boldsymbol{X}^*$ is the stationary point of inner-level optimization, we have $(\partial\mathcal{L})/(\partial\boldsymbol{X}^*) = \boldsymbol{0}$. By utilizing $f_{\boldsymbol{\theta}}(\boldsymbol{0}) = \boldsymbol{0}$ and $\Gamma_{\boldsymbol{X}}(\boldsymbol{0}) = \boldsymbol{X}$, we model the inner-level optimization process as a root-finding process of a fixed-point equation on Riemannian manifolds

$$\boldsymbol{X}^* = \Gamma_{\boldsymbol{X}^*}\left(f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)\right). \tag{19}$$

This ensures the inner-level optimization processes of various Riemannian tasks are modeled in a unified way.

The stationary point of inner-level optimization is the solution of (19). Thus, by solving (19) on the Riemannian manifold, we can obtain the stationary point of the objective function $\mathcal{L}(\cdot)$. Based on the stationary point, we can compute the gradients in the outer-level process. From the chain rule, the outer gradients are computed as

$$\frac{d\mathcal{F}}{d\boldsymbol{\theta}} = \frac{\partial\mathcal{F}}{\partial\boldsymbol{\theta}} + \frac{\partial\mathcal{F}}{\partial\boldsymbol{X}^*}\frac{d\boldsymbol{X}^*}{d\boldsymbol{\theta}} \tag{20}$$

where the direct gradients $(\partial\mathcal{F})/(\partial\boldsymbol{\theta})$ and $(\partial\mathcal{F})/(\partial\boldsymbol{X}^*)$ can be computed easily. Now we focus on $(d\boldsymbol{X}^*)/(d\boldsymbol{\theta})$. By differentiating the fixed-point equation in (19), we have that

$$\frac{d\boldsymbol{X}^*}{d\boldsymbol{\theta}} = \frac{d\Gamma_{\boldsymbol{X}^*}\left(f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)\right)}{d\boldsymbol{\theta}}. \tag{21}$$

By the chain rule, $(d\Gamma_{\boldsymbol{X}^*}(f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)))/(d\boldsymbol{\theta})$ can be expanded as

$$\begin{aligned}\frac{d\Gamma_{\boldsymbol{X}^*}\left(f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)\right)}{d\boldsymbol{\theta}} &= \frac{\partial\Gamma_{\boldsymbol{X}^*}\left(f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)\right)}{\partial\boldsymbol{X}^*}\frac{d\boldsymbol{X}^*}{d\boldsymbol{\theta}} \\ &+ \frac{\partial\Gamma_{\boldsymbol{X}^*}\left(f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)\right)}{\partial f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)}\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)}{\partial\boldsymbol{\theta}} \\ &+ \frac{\partial\Gamma_{\boldsymbol{X}^*}\left(f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)\right)}{\partial f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)}\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)}{\partial\boldsymbol{X}^*}\frac{d\boldsymbol{X}^*}{d\boldsymbol{\theta}}.\end{aligned} \tag{22}$$

We substitute (22) into (21) and have

$$\begin{aligned}\frac{d\boldsymbol{X}^*}{d\boldsymbol{\theta}} &= \frac{\partial\Gamma_{\boldsymbol{X}^*}\left(f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)\right)}{\partial\boldsymbol{X}^*}\frac{d\boldsymbol{X}^*}{d\boldsymbol{\theta}} + \frac{\partial\Gamma_{\boldsymbol{X}^*}\left(f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)\right)}{\partial f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)}\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)}{\partial\boldsymbol{\theta}} \\ &+ \frac{\partial\Gamma_{\boldsymbol{X}^*}\left(f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)\right)}{\partial f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)}\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)}{\partial\boldsymbol{X}^*}\frac{d\boldsymbol{X}^*}{d\boldsymbol{\theta}}.\end{aligned} \tag{23}$$

By arranging the term, (23) can be computed as

$$(\boldsymbol{I}-\boldsymbol{J})\frac{d\boldsymbol{X}^*}{d\boldsymbol{\theta}} = \frac{\partial\Gamma_{\boldsymbol{X}^*}\left(f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)\right)}{\partial f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)}\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)}{\partial\boldsymbol{\theta}} \tag{24}$$

where

$$\boldsymbol{J} \triangleq \frac{\partial\Gamma_{\boldsymbol{X}^*}\left(f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)\right)}{\partial\boldsymbol{X}^*} + \frac{\partial\Gamma_{\boldsymbol{X}^*}\left(f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)\right)}{\partial f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)}\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)}{\partial\boldsymbol{X}^*}. \tag{25}$$

Then we have that

$$\frac{d\boldsymbol{X}^*}{d\boldsymbol{\theta}} = (\boldsymbol{I}-\boldsymbol{J})^{-1}\frac{\partial\Gamma_{\boldsymbol{X}^*}\left(f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)\right)}{\partial f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)}\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)}{\partial\boldsymbol{\theta}}. \tag{26}$$

By substituting (26) into (20), the outer gradients can be computed implicitly as

$$\frac{d\mathcal{F}}{d\boldsymbol{\theta}} = \frac{\partial\mathcal{F}}{\partial\boldsymbol{\theta}} + \frac{\partial\mathcal{F}}{\partial\boldsymbol{X}^*}(\boldsymbol{I}-\boldsymbol{J})^{-1}\frac{\partial\Gamma_{\boldsymbol{X}^*}\left(f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)\right)}{\partial f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)}\frac{\partial f_{\boldsymbol{\theta}}(\boldsymbol{X}^*)}{\partial\boldsymbol{\theta}}. \tag{27}$$

To apply the derived outer gradients to specific Riemannian tasks, one only needs to define the update function $f_{\boldsymbol{\theta}}(\boldsymbol{X}^{(t)})$ of the inner level and substitute it into (27) to obtain concrete outer gradients. For example, on the RMO task, by utilizing the update function in (16), the outer gradients of RMO are obtained. Moreover, by substituting $(d\mathcal{F})/(d\boldsymbol{\theta}) = 0$ and (16) into the outer gradients in (27), the gradients become the same as the outer gradients derived in the E-RMO method [see (13)]. This indicates that E-RMO [5] is a special case of the proposed Riemannian implicit differentiation method. On the RHO task, by utilizing the update function in (17), the outer gradient of RMO is obtained.

Thus, in our method, outer gradients for different tasks do not need case-by-case analyses and derivations. The proposed Riemannian implicit differentiation method can be applied directly and easily for various Riemannian tasks, while the implicit differentiation in E-RMO [5] can only be applied for RMO.

### C. Implementation

To compute the outer gradients in (27), one needs the stationary point $\boldsymbol{X}^*$ for the inner-level optimization process, while it is not trivial to obtain in solving (19). We consider an approximate solution $\hat{\boldsymbol{X}}$ to the inner-level optimization. Instead of directly optimizing the specific loss function $\mathcal{L}(\cdot)$ in the inner-level optimization, we use loss function $\mathcal{L}'$ to replace $\mathcal{L}$, where the loss function $\mathcal{L}'$ is defined as

$$\mathcal{L}' = \left\|\boldsymbol{X} - \Gamma_{\boldsymbol{X}}\left(f_{\boldsymbol{\theta}}(\boldsymbol{X})\right)\right\|^2. \tag{28}$$

Then, we optimize $\mathcal{L}'$ to obtain the approximation solution $\hat{\boldsymbol{X}}$.

Computing the outer gradients in (27) requires the inversion of the matrix $\boldsymbol{J}$ that involves the Jacobian matrix. Due to the high dimensionality of the Jacobian, it is intractable to compute the matrix inversion for $\boldsymbol{J}$. To alleviate this issue,

based on Neumann series [8], [56], we compute the matrix inversion, and we review (27) as

$$\frac{d\mathcal{F}}{d\boldsymbol{\theta}} = \frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}} + \frac{\partial \mathcal{F}}{\partial \hat{\boldsymbol{X}}} \sum_{k=0}^{\infty} (\boldsymbol{J})^k \frac{\partial \Gamma_{\hat{\boldsymbol{X}}} \left(f_\theta \left(\hat{\boldsymbol{X}}\right)\right)}{\partial f_\theta \left(\hat{\boldsymbol{X}}\right)} \frac{\partial f_\theta \left(\hat{\boldsymbol{X}}\right)}{\partial \boldsymbol{\theta}} \tag{29}$$

where $\boldsymbol{J}$ is rewritten as

$$\boldsymbol{J} = \frac{\partial \Gamma_{\hat{\boldsymbol{X}}} \left(f_\theta \left(\hat{\boldsymbol{X}}\right)\right)}{\partial \hat{\boldsymbol{X}}} + \frac{\partial \Gamma_{\hat{\boldsymbol{X}}} \left(f_\theta \left(\hat{\boldsymbol{X}}\right)\right)}{\partial f_\theta \left(\hat{\boldsymbol{X}}\right)} \frac{\partial f_\theta \left(\hat{\boldsymbol{X}}\right)}{\partial \hat{\boldsymbol{X}}}. \tag{30}$$

We approximate (29) by the first $K$ terms in the infinite sum, i.e.,

$$\frac{d\mathcal{F}}{d\boldsymbol{\theta}} = \frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}} + \frac{\partial \mathcal{F}}{\partial \hat{\boldsymbol{X}}} \sum_{k=0}^{K} (\boldsymbol{J})^k \frac{\partial \Gamma_{\hat{\boldsymbol{X}}} \left(f_\theta \left(\hat{\boldsymbol{X}}\right)\right)}{\partial f_\theta \left(\hat{\boldsymbol{X}}\right)} \frac{\partial f_\theta \left(\hat{\boldsymbol{X}}\right)}{\partial \boldsymbol{\theta}} \tag{31}$$

which can be expanded as

$$\begin{aligned} &\frac{\partial \mathcal{F}}{\partial \hat{\boldsymbol{X}}} \sum_{k=0}^{K} (\boldsymbol{J})^k \frac{\partial \Gamma_{\hat{\boldsymbol{X}}} \left(f_\theta \left(\hat{\boldsymbol{X}}\right)\right)}{\partial f_\theta \left(\hat{\boldsymbol{X}}\right)} \frac{\partial f_\theta \left(\hat{\boldsymbol{X}}\right)}{\partial \boldsymbol{\theta}} \\ &= \left[\frac{\partial \mathcal{F}}{\partial \hat{\boldsymbol{X}}} (\boldsymbol{J})^0 + \frac{\partial \mathcal{F}}{\partial \hat{\boldsymbol{X}}} (\boldsymbol{J})^1 + \cdots + \frac{\partial \mathcal{F}}{\partial \hat{\boldsymbol{X}}} (\boldsymbol{J})^k\right] \\ &\quad \times \frac{\partial \Gamma_{\hat{\boldsymbol{X}}} \left(f_\theta \left(\hat{\boldsymbol{X}}\right)\right)}{\partial f_\theta \left(\hat{\boldsymbol{X}}\right)} \frac{\partial f_\theta \left(\hat{\boldsymbol{X}}\right)}{\partial \boldsymbol{\theta}}. \end{aligned} \tag{32}$$

We utilize an iterative manner to compute $[(\partial \mathcal{F})/(\partial \hat{\boldsymbol{X}})(\boldsymbol{J})^0 + (\partial \mathcal{F})/(\partial \hat{\boldsymbol{X}})(\boldsymbol{J})^1 + \cdots + (\partial \mathcal{F})/(\partial \hat{\boldsymbol{X}})(\boldsymbol{J})^k]$. we first initialize two intermediate variables $\boldsymbol{v}^0$ and $\boldsymbol{p}^0$ both as $(d\mathcal{F})/(d\hat{\boldsymbol{X}})$ and iteratively update them as

$$\begin{aligned} \boldsymbol{v}^0 &= \frac{d\mathcal{F}}{d\hat{\boldsymbol{X}}} (\boldsymbol{J})^0, \quad \boldsymbol{v}^1 = \frac{d\mathcal{F}}{d\hat{\boldsymbol{X}}} (\boldsymbol{J})^1, \ldots, \boldsymbol{v}^K = \frac{d\mathcal{F}}{d\hat{\boldsymbol{X}}} (\boldsymbol{J})^K \\ \boldsymbol{p}^0 &= \frac{d\mathcal{F}}{d\hat{\boldsymbol{X}}} (\boldsymbol{J})^0, \quad \boldsymbol{p}^1 = \frac{d\mathcal{F}}{d\hat{\boldsymbol{X}}} (\boldsymbol{J})^0 + \frac{d\mathcal{F}}{d\hat{\boldsymbol{X}}} (\boldsymbol{J})^1, \ldots, \boldsymbol{p}^K \\ &= \frac{\partial \mathcal{F}}{\partial \hat{\boldsymbol{X}}} \sum_{k=0}^{K} (\boldsymbol{J})^K. \end{aligned} \tag{33}$$

In this way, we have

$$\left[\frac{\partial \mathcal{F}}{\partial \hat{\boldsymbol{X}}} (\boldsymbol{J})^0 + \frac{\partial \mathcal{F}}{\partial \hat{\boldsymbol{X}}} (\boldsymbol{J})^1 + \cdots + \frac{\partial \mathcal{F}}{\partial \hat{\boldsymbol{X}}} (\boldsymbol{J})^k\right] = \sum_{k=0}^{K} \boldsymbol{v}^i = \boldsymbol{p}^K. \tag{34}$$

Therefore, for iteration from $i = 0$ to $i = K-1$, the intermediate variables are updated as

$$\boldsymbol{v}^{i+1} = \boldsymbol{v}^i \boldsymbol{J}, \qquad \boldsymbol{p}^{i+1} = \boldsymbol{p}^i + \boldsymbol{v}^{i+1}. \tag{35}$$

After $K$ iterations, the approximated outer gradients in (31) are computed as

$$\frac{d\mathcal{F}}{d\boldsymbol{\theta}} = \frac{\partial \mathcal{F}}{\partial \boldsymbol{\theta}} + \boldsymbol{p}^K \frac{\partial \Gamma_{\hat{\boldsymbol{X}}} \left(f_\theta \left(\hat{\boldsymbol{X}}\right)\right)}{\partial f_\theta \left(\hat{\boldsymbol{X}}\right)} \frac{\partial f_\theta \left(\hat{\boldsymbol{X}}\right)}{\partial \boldsymbol{\theta}}. \tag{36}$$

By using the specific expression of $\boldsymbol{J}$ in (30), the specific format of intermediate vector $\boldsymbol{v}$ is

$$\boldsymbol{v} \leftarrow \boldsymbol{v} \frac{\partial \Gamma_{\hat{\boldsymbol{X}}} \left(f_\theta \left(\hat{\boldsymbol{X}}\right)\right)}{\partial \hat{\boldsymbol{X}}} + \boldsymbol{v} \frac{\partial \Gamma_{\hat{\boldsymbol{X}}} \left(f_\theta \left(\hat{\boldsymbol{X}}\right)\right)}{\partial f_\theta \left(\hat{\boldsymbol{X}}\right)} \frac{\partial f_\theta \left(\hat{\boldsymbol{X}}\right)}{\partial \hat{\boldsymbol{X}}}. \tag{37}$$

Because the intermediate variables $\boldsymbol{v}$ and $\boldsymbol{p}$ are vectors in all aforementioned iterations, computing (35) and (36) only needs Jacobian-vector product [57], [58], [59], eliminating the need of time-consuming and memory-consuming Jacobian in the original formula in (27). The whole method is summarized in Algorithm 1.

**Algorithm 1** Riemannian Implicit Differentiation

**Input:** Initial Riemannian parameters $\boldsymbol{X}^{(0)}$, initial outer-level parameters $\boldsymbol{\theta}^{(0)}$, maximum iteration $\Upsilon$ of the outer-level, maximum iteration $K$ of Neumann series.

```
τ = 0.
while τ ≤ Υ do
    Approximate the solution X̂.
    Compute the outer-level loss F(X̂).
    Compute the gradients ∂F(X̂)/∂X̂
    v^0 = p^0 = ∂F(X̂)/∂X̂
    i = 0
    while i < K do
        Update intermediate vector v^i and p^i via (35).
    end while
    Compute the implicit gradients by (36).
    Update θ by gradient descent algorithm.
end while
Return the outer-level parameters θ.
```

### D. Complexity

*1) Time Complexity:* Some works show that the complexity of computing gradients or Jacobian-vector products of a differentiable function is no more than five times the complexity of computing the function itself, and the complexity of computing Hessian-vector products is no more than five times the complexity of computing gradients, see [58], [60] for details.

By utilizing the above two principles, we compare the time complexity of computing the outer gradients of our method with the unrolling method. We denote the the unrolling outer gradients as $\nabla \mathcal{F}(\boldsymbol{\theta})^u$ and denote the proposed outer gradients as $\nabla \mathcal{F}(\boldsymbol{\theta})^a$. The time complexity of $\nabla \mathcal{F}(\boldsymbol{\theta})^u$ satisfies that

$$\begin{aligned} \mathcal{O}\left(\nabla \mathcal{F}\left(\boldsymbol{\theta}\right)^u\right) \leq\ & 5T^2 \mathcal{O}\left(\Gamma_{\boldsymbol{X}}\left(\cdot\right)\right) + \left(\frac{5}{2}T^2 + \frac{5}{2}T\right) \mathcal{O}\left(f_\theta\left(\cdot\right)\right) \\ & + \left(\frac{25}{2}T^2 - \frac{25}{2}T\right) \mathcal{O}\left(\mathcal{L}\left(\cdot\right)\right) + 10\mathcal{O}\left(\mathcal{F}\left(\cdot\right)\right). \end{aligned} \tag{38}$$

While the time complexity of $\nabla \mathcal{F}(\boldsymbol{\theta})^a$ satisfies that

$$\begin{aligned} \mathcal{O}\left(\nabla \mathcal{F}\left(\boldsymbol{\theta}\right)^a\right) \leq\ & K\left(10\mathcal{O}\left(\Gamma\left(\cdot\right)\right) + 5\mathcal{O}\left(f_\theta\left(\cdot\right)\right)\right) \\ & + 10\mathcal{O}\left(\mathcal{F}\left(\cdot\right)\right) + 5\mathcal{O}\left(\Gamma\left(\cdot\right)\right) + 5\mathcal{O}\left(f_\theta\left(\cdot\right)\right). \end{aligned} \tag{39}$$

Here, $\mathcal{O}(\Gamma(\cdot))$, $\mathcal{O}(\mathcal{F}(\cdot))$, and $\mathcal{O}(f_\theta(\cdot))$ represent the time complexity of the retraction operation, the loss function, and the update function, respectively. As to the various Riemannian manifolds and Riemannian tasks, $\mathcal{O}(\Gamma(\cdot))$, $\mathcal{O}(\mathcal{F}(\cdot))$, and $\mathcal{O}(f_\theta(\cdot))$ are various. We provide specific complexity analysis in the supplementary materials.

Apparently, the computational complexity of our method is independent of the maximum optimization steps $T$ of the inner-level optimization. With the increase of $T$, the time for computing the outer gradients of the unrolling method increases significantly, while the running time of our method to compute the outer gradients is constant. Though our method is related to the iteration steps $K$ in the Neumann series, $K$ is far less than $T$. Compared to the unrolling method, the proposed method reduces computational complexity significantly.

Since the implicit differentiation method in E-RMO [5] is only designed for RMO, we compare the complexities of our method with those of E-RMO in the RMO task. E-RMO approximates the outer gradients as

$$\frac{d\mathcal{F}}{d\boldsymbol{\theta}} = \frac{\partial \mathcal{F}}{\partial \hat{\boldsymbol{X}}} \sum_{k=0}^{K} (\boldsymbol{Z})^k \frac{\partial \Gamma_{\hat{\boldsymbol{X}}} \left( g_{\boldsymbol{\theta}} \left( \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{X}}} \right) \right)}{\partial g_{\boldsymbol{\theta}} \left( \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{X}}} \right)} \frac{\partial g_{\boldsymbol{\theta}} \left( \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{X}}} \right)}{\partial \boldsymbol{\theta}} \tag{40}$$

where $\boldsymbol{Z}$ is

$$\boldsymbol{Z} = \frac{\partial \Gamma_{\hat{\boldsymbol{X}}} \left( g_{\boldsymbol{\theta}} \left( \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{X}}} \right) \right)}{\partial \hat{\boldsymbol{X}}} + \frac{\partial \Gamma_{\hat{\boldsymbol{X}}} \left( g_{\boldsymbol{\theta}} \left( \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{X}}} \right) \right)}{\partial g_{\boldsymbol{\theta}} \left( \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{X}}} \right)} \frac{\partial g_{\boldsymbol{\theta}} \left( \frac{\partial \mathcal{L}}{\partial \hat{\boldsymbol{X}}} \right)}{\partial \hat{\boldsymbol{X}}}. \tag{41}$$

The time complexity satisfies that

$$\begin{aligned} \mathcal{O}\left(\frac{d\mathcal{F}}{d\boldsymbol{\theta}}\right) \leq K \left(10\mathcal{O}\left(\Gamma\left(\cdot\right)\right) + 5\mathcal{O}\left(g_{\boldsymbol{\theta}}\left(\cdot\right)\right)\right) \\ + 5\mathcal{O}\left(\mathcal{F}\left(\cdot\right)\right) + 5\mathcal{O}\left(\Gamma\left(\cdot\right)\right) + 5\mathcal{O}\left(g_{\boldsymbol{\theta}}\left(\cdot\right)\right). \end{aligned} \tag{42}$$

As to our method, by substituting $(\partial \mathcal{F})/(\partial \boldsymbol{\theta}) = \mathbf{0}$ and the update function in (16) into the outer gradients in (31), (31) becomes the same as (40). Because $\mathcal{O}((\partial \mathcal{F})/(\partial \hat{\boldsymbol{X}})) = 0$ and $\mathcal{O}(f_{\boldsymbol{\theta}}(\cdot)) = \mathcal{O}(g_{\boldsymbol{\theta}}(\cdot))$, the complexity of our method in the RMO task becomes the same as that of E-RMO method in (42).

*2) Memory Cost:* To compute the outer gradients, unrolling methods need to save the entire inner-level optimization process, including the inner-level parameters, their gradients, the update vectors, the updated parameters, and the outer-level parameters. As a result, these methods incur a memory cost of $\mathcal{O}(4\mathrm{Td}^2 + H)$, where $H$ is the size of the outer-level parameters. This memory cost scales with the number of inner-level optimization steps $T$. Because the outer gradients only depend on the final solution of the inner-level optimization, we do not need to save the computation graph of the inner-level optimization. In our method, we only need to save $\hat{\boldsymbol{X}}$, $(\partial \mathcal{L})/(\partial \hat{\boldsymbol{X}})$, $f_{\boldsymbol{\theta}}(\hat{\boldsymbol{X}})$, $\Gamma_{\hat{\boldsymbol{X}}}(f_{\boldsymbol{\theta}}(\hat{\boldsymbol{X}}))$, and $\boldsymbol{\theta}$ in memory. The memory cost of our method is $O(4d^2 + H)$. In E-RMO [5], $\hat{\boldsymbol{X}}$, $(\partial \mathcal{L})/(\partial \hat{\boldsymbol{X}})$, $g_{\boldsymbol{\theta}}((\partial \mathcal{L})/(\partial \hat{\boldsymbol{X}}))$, $\Gamma_{\hat{\boldsymbol{X}}}(g_{\boldsymbol{\theta}}((\partial \mathcal{L})/(\partial \hat{\boldsymbol{X}})))$, and $\boldsymbol{\theta}$ are need to be saved in the memory, and thus the memory cost is $\mathcal{O}(4d^2 + H)$, which is consistent with that of our method.

## V. Theoretical Analysis

We provide the approximation error analysis and convergence analysis for our method, which prove that our method can be effectively applied to various Riemannian tasks.

### A. Assumptions

Let $\mathcal{F}(\boldsymbol{X}, \boldsymbol{\theta})$ is $\mathcal{S}$ Lipschitz continuous and $\mathcal{I}$ smooth with respect to $\boldsymbol{X}$, and is $\beta_3$ Lipschitz continuous with respect to $\boldsymbol{\theta}$. We assume that $\Gamma_{\boldsymbol{X}}(\boldsymbol{Y})$ is $\beta_1$ Lipschitz continuous with respect to $\boldsymbol{Y}$ and $\rho_1$ Lipschitz continuous with respect to $\boldsymbol{X}$. Let $\mathcal{L}(\boldsymbol{X}, \boldsymbol{\theta})$ be strongly convex with respect to $\boldsymbol{X}$.

*Discussion*: These assumptions are commonly used in machine learning, especially in the bilevel optimization community, e.g., in the work of [33] and [61].

Moreover, we assume that $f_{\boldsymbol{\theta}}(\boldsymbol{X})$ is $\alpha_1$ Lipschitz continuous with respect to $\boldsymbol{X}$ and $\alpha_2$ Lipschitz continuous with respect to $\boldsymbol{\theta}$; and assume $\Gamma_{\boldsymbol{X}} \circ f_{\boldsymbol{\theta}}(\cdot)$ is $\beta_2$ Lipschitz continuous, and $\beta_2 < 1$. As to the derivative, we assume that $(\partial f_{\boldsymbol{\theta}}(\boldsymbol{X}))/(\partial \boldsymbol{\theta})$ is $\gamma_1$ Lipschitz continuous with respect to $\boldsymbol{X}$ and $\gamma_2$ Lipschitz continuous with respect to $\boldsymbol{\theta}$. $(\partial f_{\boldsymbol{\theta}}(\boldsymbol{X}))/(\partial \boldsymbol{X})$ is $\gamma_3$ Lipschitz continuous with respect to $\boldsymbol{\theta}$, and $\gamma_4$ Lipschitz continuous with respect to $\boldsymbol{X}$. $(\partial \Gamma_{\boldsymbol{X}}(\boldsymbol{Y}))/(\partial \boldsymbol{Y})$ is $\lambda_1$ Lipschitz continuous with respect to $\boldsymbol{X}$ and $\rho_2$ Lipschitz continuous with respect to $\boldsymbol{Y}$.

*Discussion*: These assumptions are also commonly used in bilevel optimization. For example, as to (17), we only need the Lipschitz continous assmuption of $(\partial \mathcal{L})/(\partial \boldsymbol{X}), (\partial^2 \mathcal{L})/(\partial \boldsymbol{X}^2)$, and $(\partial(\partial \mathcal{L})/(\partial \boldsymbol{X}))/(\partial \boldsymbol{\theta})$ to satisfy these assumptions. These assumptions are also used in the proof of convergence analysis in the work of [33] and [62].

### B. Approximation Error Analysis

In (29), we utilize $\hat{\boldsymbol{X}}$ to approximate the stationary point $\boldsymbol{X}^*$. This approximation error is upper-bounded as shown below. For convenience, we denote the approximated outer gradients in (29) as $\widetilde{\nabla} \mathcal{F}(\boldsymbol{\theta})$ and denote the outer gradients in (27) as $\nabla \mathcal{F}(\boldsymbol{\theta})$.

*Theorem 1:* Supposing that Riemannian parameters $\hat{\boldsymbol{X}}$ satisfy

$$\left\| \boldsymbol{X}^* - \hat{\boldsymbol{X}} \right\| \leq \delta \tag{43}$$

the approximated outer gradients $\widetilde{\nabla} \mathcal{F}(\boldsymbol{\theta})$ satisfy

$$\left\| \nabla \mathcal{F}\left(\boldsymbol{\theta}\right) - \widetilde{\nabla} \mathcal{F}\left(\boldsymbol{\theta}\right) \right\| \leq \mathcal{C}\delta \tag{44}$$

where $\mathcal{C}$ is a constant, and is computed by

$$\begin{aligned} \mathcal{C} = & \frac{\mathcal{S}\left(\beta_1\gamma_1 + \lambda_1\alpha_2 + \rho_2\alpha_1\alpha_2\right) + \alpha_2\beta_1\mathcal{I}}{1-\beta_2} \\ & + \frac{\mathcal{S}\alpha_2\beta_1\left(\rho_2\alpha_1 + \lambda_1 + \beta_1\gamma_4 + \rho_2\alpha_1^2 + \lambda_1\alpha_1\right)}{(1-\beta_2)^2}. \end{aligned} \tag{45}$$

We utilize the first $K$ terms of the Neumann series to approximate $\widetilde{\nabla} \mathcal{F}(\boldsymbol{\theta})$ in (36). This approximation error is upper-bounded as shown below. For convenience, we denote the outer gradients in (36) as $\hat{\nabla} \mathcal{F}(\boldsymbol{\theta})$.

*Theorem 2:* Supposing that the assumption holds, the approximated outer gradients $\hat{\nabla} \mathcal{F}(\boldsymbol{\theta})$ satisfy

$$\left\| \widetilde{\nabla} \mathcal{F}\left(\boldsymbol{\theta}\right) - \hat{\nabla} \mathcal{F}\left(\boldsymbol{\theta}\right) \right\| \leq \mathcal{A} \times \mathcal{B}^{K+1} \tag{46}$$

where $\mathcal{A}$ and $\mathcal{B}$ are constants and are computed by

$$\mathcal{A} = \frac{\mathcal{S}\alpha_2\beta_1}{1-\beta_2} \quad \text{and} \quad \mathcal{B} = \beta_2. \tag{47}$$

By using the triangle inequality, the whole approximation error of our method can be summarized as

$$\left\| \nabla \mathcal{F}\left(\boldsymbol{\theta}\right) - \hat{\nabla} \mathcal{F}\left(\boldsymbol{\theta}\right) \right\| \leq \mathcal{A} \times \mathcal{B}^{K+1} + \mathcal{C}\delta. \tag{48}$$

The proofs of Theorems 1 and 2 are provided in supplementary materials.

### C. Convergence Analysis

To prove the convergence performance of our method, we provide an auxiliary lemma as shown below.

*Lemma 1:* For any outer-level parameters $\boldsymbol{\theta}$, $\boldsymbol{\theta}'$

$$\left\| \nabla \mathcal{F}\left(\boldsymbol{\theta}\right) - \nabla \mathcal{F}\left(\boldsymbol{\theta}'\right) \right\| \leq L_{\Phi} \left\| \boldsymbol{\theta} - \boldsymbol{\theta}' \right\| \tag{49}$$

where $L_\Phi$ is a constant and is computed by

$$L_\Phi = \mathcal{S}\left[\frac{\beta_1\gamma_2+\rho_2\alpha_2^2}{1-\beta_2}+\frac{\beta_1\alpha_2\left(\beta_1\gamma_3+\rho_2\alpha_1\alpha_2+\rho_2\alpha_2\right)}{(1-\beta_2)^2}\right]+\beta_3. \tag{50}$$

Motivated by the work of [33], we prove the convergence of our method by bounding the gradients of the outer-level parameters, which is provided in Theorem 3.

*Theorem 3:* Choose the step-size of the outer-level parameters $1/(8L_\Phi) \le e \le 1/(4L_\Phi)$, and set the outer-level optimization iteration number as $\Upsilon$. The outputs of our algorithm satisfy

$$\frac{1}{\Upsilon}\sum_{\tau=1}^{\Upsilon}\left\|\nabla\mathcal{F}\left(\boldsymbol{\theta}^\tau\right)\right\|^2 \le \frac{64}{3}L_\Phi\frac{\mathcal{F}\left(\boldsymbol{\theta}^1\right)-\mathcal{F}\left(\boldsymbol{\theta}^{\Upsilon+1}\right)}{\Upsilon} + 6\mathcal{A}\times\mathcal{B}^{2K+2}+6\mathcal{C}^2\delta^2. \tag{51}$$

The proofs of Lemma 1 and Theorem 3 are provided in supplementary materials.

### D. Discussions

The theoretical analyses in our method are capable of being applicable to all Riemannian manifolds. The theoretical analyses are independent of the specific shape or topology of the underlying manifolds and rely only on the fundamental definition of Riemannian manifolds, that is, the tangent spaces of the Riemannian manifolds are equipped with an inner product $[d, e]$. As long as the manifold under consideration satisfies this definition, the theoretical results hold. Specifically, the conducted theoretical analysis requires the gradient computation and Lipschitz continuity assumptions of Riemannian operations, both of which depend on the inner product defined on the tangent spaces at each point on the manifold—the fundamental definition of the Riemannian manifolds. Although different Riemannian manifolds may have different inner products, this only leads to variations in Lipschitz constants $\beta_1, \rho_1, \rho_2, \lambda_1, \beta_2, \alpha_1$, and $\gamma_4$, but does not affect the validity of the assumptions. Thus, the variations do not affect the structure or validity of our theoretical results in Theorems 1–3. These variations only lead to different values for constants $\mathcal{A}, \mathcal{B}, \mathcal{C}$, and $L_\Phi$ on different manifolds. Manifold-specific convergence and approximation guarantees can be derived by computing the Lipschitz continuity constants of Riemannian operations for the chosen manifold and substituting them into the general expressions for $\mathcal{A}, \mathcal{B}, \mathcal{C}$, and $L_\Phi$.

Our theoretical results in Theorems 1–3 demonstrate that both the approximation error and convergence performance are strongly dependent on the precision of the inner-level solution that is captured by $\delta$, and $\delta \ge \|\boldsymbol{X}^* - \hat{\boldsymbol{X}}\|$. A smaller $\delta$ leads to lower approximation error and improved convergence guarantees. Our method can achieve a smaller $\delta$ by additionally incorporating the second-order information in inner updates. Specifically, existing methods minimize the loss function $\mathcal{L}(\cdot)$ in the inner-level optimization problem to update $\boldsymbol{X}$ by utilizing the classic first-order RGD method in (5). We can observe that inner updates only contain first-order gradient information $(\partial\mathcal{L}(\boldsymbol{X}^{(t)}(\boldsymbol{\theta})))/(\partial\boldsymbol{X}^{(t)}(\boldsymbol{\theta}))$. Instead of directly minimizing $\mathcal{L}(\cdot)$, we establish the inner-level optimization as a root-finding process of a fixed-point equation in (19), which is further formulated as

$$\boldsymbol{X}^*(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{X}}\mathcal{L}'(\boldsymbol{X}, \boldsymbol{\theta}) \tag{52}$$

where

$$\mathcal{L}'(\boldsymbol{X}, \boldsymbol{\theta}) = \left\|\boldsymbol{X} - \Gamma_X\left(f_\theta(\boldsymbol{X})\right)\right\|^2. \tag{53}$$

To update $\boldsymbol{X}$, the gradients of $\boldsymbol{X}$ are formulated as

$$\frac{\partial\mathcal{L}'}{\partial\boldsymbol{X}} = \left(\boldsymbol{X}-\Gamma_X\left(f_\theta(\boldsymbol{X})\right)\right)^\top \times\left(\boldsymbol{I}-\frac{\partial\Gamma_X\left(f_\theta(\boldsymbol{X})\right)}{\partial\boldsymbol{X}}-\frac{\partial\Gamma_X\left(f_\theta(\boldsymbol{X})\right)}{\partial f_\theta(\boldsymbol{X})}\frac{\partial f_\theta(\boldsymbol{X})}{\partial\boldsymbol{X}}\frac{\partial^2\mathcal{L}}{\partial\boldsymbol{X}^2}\right). \tag{54}$$

Compared with existing methods, our method improves the precision of $\boldsymbol{X}$ by introducing second-order information $(\partial^2\mathcal{L})/(\partial\boldsymbol{X}^2)$ into the updates in the inner-level optimization, thereby reducing $\delta$. In this way, our method achieves smaller approximation errors and better theoretical guarantees.

## VI. Experiments

### A. Settings

We evaluate the performance of our method across three tasks: RMO, RHO, and Riemannian metalearning.

*1) Riemannian Metaoptimization:* In the RMO task, we compare our method with two unrolling-based RMMs: the original RMM [3] and the gradient-free RMM (GF-RMM) [21]. Moreover, we compare our optimizer with hand-designed Riemannian optimizers: RSGD [7], RSGDM [10], and Riemannian adaptive stochastic gradient algorithms (RASA) [15]. We conduct two experiments on RMO as shown below.

*a) PCA on the Grassmann manifold:* Principal component analysis (PCA) aims to learn an orthogonal matrix $\boldsymbol{X}$ that projects the original data to a lower-dimensional space, while minimizing the squared residual errors between projected results and the original data. Given a set of samples $\{\boldsymbol{y} \in \mathbb{R}^d\}_{i=1}^n$, the PCA task can be formulated as

$$\min_{\boldsymbol{X}\in\mathcal{G}(p,d)}\mathcal{L}(\boldsymbol{X};\boldsymbol{y}) = \frac{1}{n}\sum_{i=1}^{n}\left\|\boldsymbol{y}_i - \boldsymbol{X}\boldsymbol{X}^\top\boldsymbol{y}_i\right\|^2. \tag{55}$$

The orthogonal matrix $\boldsymbol{X}$ should satisfy the rotation invariance property that enforces $\boldsymbol{X}$ to be on the Grassmann manifold. We utilize the MNIST dataset to evaluate our optimizer.

*b) Clustering on the SPD manifold:* We also conduct clustering experiments on the SPD manifold, using the Kylberg texture dataset [63], where each image is represented by a $5\times5$ covariance descriptor. Given a set of SPD points $\{\boldsymbol{y}_i \in \mathrm{Sym}_d^+\}$, the clustering task aims to find SPD centroids by solving the following problem

$$\min_{\{\boldsymbol{X}_r\}_{r=1}^C\in\mathrm{Sym}_d^+}\mathcal{L}(\boldsymbol{X}) = \sum_{i\rightsquigarrow r}^{n} d\left(\boldsymbol{y}_i, \boldsymbol{X}_r\right) \tag{56}$$

where $i \rightsquigarrow r$ shows that sample $\boldsymbol{y}_i$ is assigned to $\boldsymbol{X}_r$. We adopted the affine invariance metric (AIM) [64] to compute the distance between $\boldsymbol{y}_i$ and the centroid $\boldsymbol{X}_j$.

To optimize the Riemannian bilevel optimization objective, we utilize the learnable optimizer $g_{\boldsymbol{\theta}}(\cdot)$ to update the parameters on the Grassmann manifold and SPD manifold in the inner level for $T$ iterations, specifically. Then, the gradients regarding the parameters $\boldsymbol{\theta}$ of the learnable optimizer are computed by (36). Finally, we utilize the gradient descent algorithm to update $\boldsymbol{\theta}$.

*2) Riemannian Hyperparameter Optimization:* For RHO, we evaluate our method on the classification task using long-tailed datasets. We observe that in the last fully connected layer of the model, the amount of data in each class affects the norm of the corresponding column in the layer. The class that has a large amount of data usually has a large norm. In addition, the small amount of tail class leads to overfitting. We apply the orthogonal constraint to the fully connected layer to keep each column having the same norm and alleviate the overfitting issue.

We adopt the two-component weighting (TCW) method [69] as the baseline, which proposes to weigh each training example by a fixed component $\boldsymbol{w}$ and a trainable component $\boldsymbol{\theta}$. The objective of RHO in our experiments is modeled as

$$\begin{aligned} &\min_{\boldsymbol{\theta}} \mathcal{F}(\boldsymbol{\theta}) := l_V(\boldsymbol{X}^*(\boldsymbol{\theta})) \\ &\text{s.t.} \quad \boldsymbol{X}^*(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{X}} (\boldsymbol{\theta} + w)\, l_T(\boldsymbol{X}) \end{aligned} \tag{57}$$

where $l_T$ is the loss function on the training set and $l_V$ is the loss function on the validation set. Here, we train the neural network in the inner-level optimization and train $\boldsymbol{\theta}$ in the outer-level optimization. Following the experiment setup in the TCW method, we conduct experiments on two datasets, CIFAR-LT-10 and CIFAR-LT-100, which are created by removing some training examples from CIFAR-10 [71] and CIFAR-100 [71] according to different power law distributions. We compare our method with the hyperparameter optimization methods that unroll the inner-level optimization to compute the outer gradients on Euclidean spaces.

*3) Riemannian Metalearning:* The goal of RML is to learn a suitable initialization $\boldsymbol{\theta}$ of a Riemannian model (e.g., a neural network with weights complying with the geometry of a manifold), enabling rapid adaptation with few training examples. $\boldsymbol{\theta}$ and $\boldsymbol{X}$ are networks in RML. We evaluate our method on a few-shot learning task in the RML setting. Research proves that modeling data on the tehyperbolic manifolds leads to better performance [4]. Inspired by it, we project the Euclidean features onto the hyperbolic manifold and build a hyperbolic classifier via hyperbolic distances. Following the setting of model-agnostic meta-learning (MAML) [72], we learn to initialize the hyperbolic classifier. We conduct one-shot/five-shot five-way experiments on two datasets: mini-ImageNet [87] and tiered-ImageNet [67]. We compare our approach with metalearning methods that compute outer gradients by unrolling on the Euclidean spaces [72], [73], [74], [75], [76], [77], [78], [79], [80], [81], [82] or hyperbolic space [4], [83], [84], [85].

## B. Convergence Analysis

We conduct convergence analyses on the RMO task. Concretely, we train the Riemannian optimizer on the training set and evaluate the convergence on both the training and test datasets. Results are shown in Figs. 1 and 2. Among the compared methods, ours achieves the best performance in terms of the convergence speed and the obtained optima. Moreover, ours converges to the smaller values (in log scale), compared to the other methods. Specifically, in the PCA task, ours converges to 2.36, while RMM and GF-RMM converge to 2.38 and 2.37, respectively. In the clustering task, ours converges to $-1.58$, while RMM and GF-RMM converge to $-1.51$ and $-1.51$, respectively. These results show the effectiveness and superiority of our method.

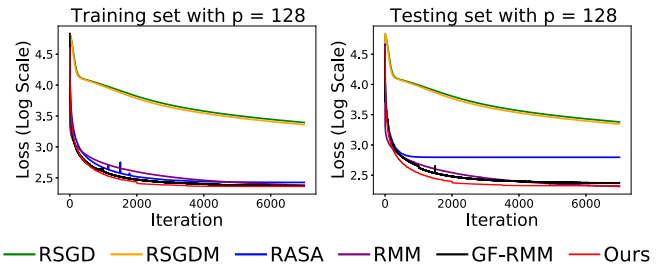



Fig. 1. Plots for the PCA task (in the log scale).

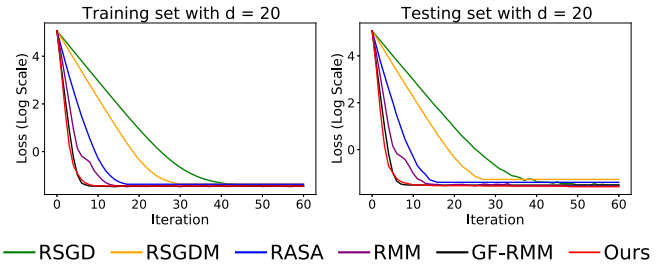



Fig. 2. Plots for the clustering task (in the log scale).

## C. Accuracy Analyses

We conduct accuracy analyses on the RHO and RML tasks.

*1) Riemannian Hyperparameter Optimization:* The top-1 errors on test data are presented in Table I, showing our method consistently outperforms others. For example, on the CIFAR-LT-10 dataset, our method is 8.94%, 6.10%, 4.70%, and 2.0% higher than the baseline method TCW, as to imbalance ratios equal to $200, 100, 50$, and $20$, respectively. And our method is 3.24%, 4.58%, 3.82%, and 3.13% higher than the baseline method TCW on the CIFAR-LT-100 dataset, as to imbalance ratio equals $200, 100, 50$, and $20$, respectively. The experimental results again demonstrate the superiority of our method.

*2) Riemannian Metalearning:* Results are shown in Tables II and III. The performance of our method surpasses all compared methods, demonstrating the superiority of our method.

## D. Efficiency Analyses

Given that the idea of implicit differentiation in the E-RMO method [5] is designed to alleviate computational burden, we conduct efficiency analyses on RMO and RHO tasks in terms of both training time and memory usage.

TABLE I
TOP-1 ERROR (%) ON CIFAR-LT-10/CIFAR-LT-100

| | CIFAR-LT-10 | | | | CIFAR-LT-100 | | | |
|---|---|---|---|---|---|---|---|---|
| **Imbalance Ratio** | **200** | **100** | **50** | **20** | **200** | **100** | **50** | **20** |
| Cross-entropy training | 34.32 | 29.63 | 25.19 | 17.77 | 65.16 | 61.68 | 56.15 | 48.86 |
| Class-balanced cross-entropy loss [65] | 31.11 | 27.63 | 21.95 | 15.64 | 64.30 | 61.44 | 55.45 | 42.88 |
| Class-balanced fine-tuning [66] | 33.76 | 28.66 | 22.56 | 16.78 | 61.34 | 58.5 | 53.78 | 47.70 |
| L2RW [67] | 33.75 | 27.77 | 23.55 | 18.65 | 67.00 | 61.10 | 56.83 | 49.25 |
| Meta-weight net [68] | 32.8 | 26.43 | 20.9 | 15.55 | 63.38 | 58.39 | 54.34 | 46.96 |
| Two-component weighting [69] | 29.34 | 23.59 | 19.49 | 13.54 | 60.69 | 56.65 | 51.47 | 44.38 |
| Divide and Retain [70] | - | - | - | - | 59.47 | 55.21 | 50.68 | - |
| Ours | **20.4** | **17.49** | **14.74** | **11.54** | **57.45** | **52.07** | **47.64** | **41.25** |

TABLE II
ACCURACY (%) ON THE MINI-IMAGENET DATASET

| **Method** | **Backbone** | **1-shot 5-way** | **5-shot 5-way** |
|---|---|---|---|
| MAML [72] | ResNet12 | $51.03 \pm 0.50$ | $68.26 \pm 0.47$ |
| L2F [73] | ResNet12 | $57.48 \pm 0.49$ | $74.68 \pm 0.43$ |
| CAML [74] | ResNet12 | $59.23 \pm 0.99$ | $72.35 \pm 0.71$ |
| ALFA [75] | ResNet12 | $60.06 \pm 0.49$ | $77.42 \pm 0.42$ |
| MetaOptNet [76] | ResNet12 | $62.64 \pm 0.61$ | $78.63 \pm 0.46$ |
| MetaFun [77] | ResNet12 | $62.12 \pm 0.30$ | $78.20 \pm 0.16$ |
| DSN [78] | ResNet12 | $62.64 \pm 0.66$ | $78.83 \pm 0.45$ |
| Chen *et al.* [79] | ResNet12 | $63.17 \pm 0.23$ | $79.26 \pm 0.17$ |
| MeTAL [80] | ResNet12 | $59.64 \pm 0.38$ | $76.20 \pm 0.19$ |
| LEO [81] | WRN-28-10 | $61.76 \pm 0.08$ | $77.59 \pm 0.12$ |
| Con-MetaReg [82] | ResNet12 | $53.68 \pm 0.50$ | $66.88 \pm 0.42$ |
| Hyper ProtoNet [4] | ResNet18 | $59.47 \pm 0.20$ | $76.84 \pm 0.14$ |
| Hyperbolic kernel [83] | ResNet18 | $61.04 \pm 0.21$ | $77.33 \pm 0.15$ |
| CurAML [84] | ResNet12 | $63.13 \pm 0.41$ | $81.04 \pm 0.39$ |
| Poincaré radial kernel [85] | ResNet18 | $62.15 \pm 0.20$ | $77.81 \pm 0.15$ |
| Ours | ResNet12 | $\mathbf{64.5 \pm 0.23}$ | $\mathbf{82.1 \pm 0.15}$ |

TABLE III
ACCURACY (%) ON THE TIERED-IMAGENET DATASET

| **Method** | **Backbone** | **1-shot 5-way** | **5-shot 5-way** |
|---|---|---|---|
| ProtoNet [86] | ResNet12 | $53.51 \pm 0.89$ | $72.69 \pm 0.74$ |
| MAML [72] | ResNet12 | $58.58 \pm 0.49$ | $71.24 \pm 0.43$ |
| L2F [73] | ResNet12 | $63.94 \pm 0.48$ | $77.61 \pm 0.41$ |
| ALFA [75] | ResNet12 | $64.43 \pm 0.49$ | $81.77 \pm 0.39$ |
| DSN [78] | ResNet12 | $66.22 \pm 0.75$ | $82.79 \pm 0.48$ |
| MetaOptNet [76] | ResNet12 | $65.99 \pm 0.72$ | $83.28 \pm 0.12$ |
| MetaFun [77] | ResNet12 | $67.72 \pm 0.14$ | $78.20 \pm 0.16$ |
| Chen *et al.* [79] | ResNet12 | $68.62 \pm 0.27$ | $83.74 \pm 0.18$ |
| MeTAL [80] | ResNet12 | $63.89 \pm 0.43$ | $80.14 \pm 0.40$ |
| LEO [81] | WRN-28-10 | $66.33 \pm 0.05$ | $81.44 \pm 0.09$ |
| Con-MetaReg [82] | ResNet12 | $54.41 \pm 0.53$ | $68.23 \pm 0.47$ |
| Hyper ProtoNet [4] | ResNet18 | $54.44 \pm 0.23$ | $71.96 \pm 0.20$ |
| Hyperbolic kernel [83] | ResNet18 | $57.78 \pm 0.23$ | $76.48 \pm 0.18$ |
| CurAML [84] | ResNet12 | $68.46 \pm 0.56$ | $83.84 \pm 0.40$ |
| Poincaré radial kernel [85] | ResNet18 | $65.33 \pm 0.21$ | $77.48 \pm 0.20$ |
| Ours | ResNet12 | $\mathbf{71.56 \pm 0.46}$ | $\mathbf{85.75 \pm 0.20}$ |

*1) Training Time:* We compared our training time for each outer-level step with that of the unrolling methods for different numbers of optimization steps in the inner-level process. Results on the RMO task are shown in Table IV, and results on RHO tasks are shown in Table V. For unrolling methods, training time is highly related to the number of inner-level optimization steps, leading to high time costs. In contrast, the training time of our method is a small constant value, independent of the number of inner-level optimization steps. As a result, our method reduces much time consumption. For example, when $T = 165$, the unrolling method requires a factor of 45 times our method on the PCA task. This allows us to use more inner-level steps, improving optimization performance and stability.

*2) Training Memory:* We evaluate the memory consumption of our method, and results are shown in Tables VI and V. Similar to the results of training time, our method requires less memory, regardless of the number of inner-level optimization steps. For example, when $T = 165$, the compared methods require a factor of 1.93 times of our method on the PCA task.

### *E. Comparison With Implicit Differentiation in E-RMO*

E-RMO method [5] introduces implicit differentiation to the RMO task, which is the only implicit differentiation method on Riemannian manifolds. In this section, we compare our method with E-RMO. For implementing ID-RMO on other tasks (RHO and RML), we first derive the outer gradients for other tasks by case-by-case analysis. Then, we utilize the inner-level loss function $\mathcal{L}(\cdot)$ and RSGD [7] algorithm to approximate $\boldsymbol{X}^*$. Finally, we compute the implicit gradients for other tasks and utilize the gradient descent algorithm to update $\boldsymbol{\theta}$.

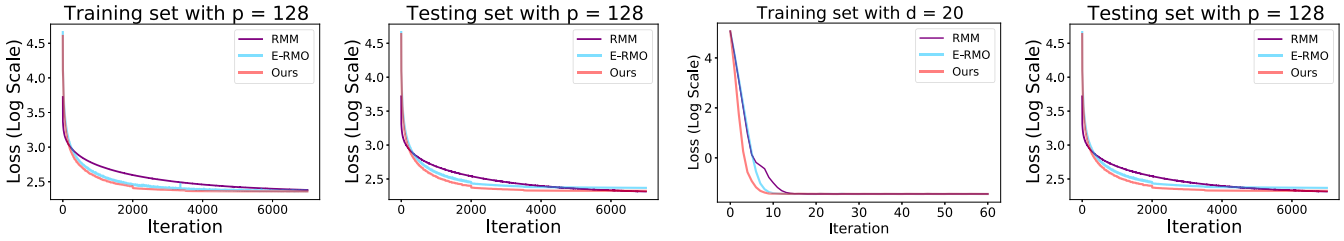
*1) Convergence Analyses:* We conduct convergence analyses on the *RMO* and *RHO* tasks. In terms of RMO, we first train the learnable Riemannian optimizer on the training set. Then, we utilize the trained learnable optimizer to optimize the Riemannian parameters on both the training set and the test set. Results are shown in Fig. 3. As to RHO, experiments are conducted by training models on the training set, while computing the loss on the test dataset. Results are shown in Fig. 4. Both tasks exhibit similar trends: the proposed method outperforms all compared methods, achieving superior performance. This is due to its ability to obtain higher-precision inner-level solutions by utilizing the second-order information in $\mathcal{L}'$.

We further conduct experiments to validate that the introduced loss function $\mathcal{L}'(\cdot)$ in our method improves the precision of the converged inner-level parameters. Specifically, we employ the inner-level objective defined in (53) and adopt the inner-level optimization procedure from E-RMO to update the inner-level parameters, respectively. We evaluate our approach on two tasks, i.e., the RMO task (PCA on Grassmann manifolds) and the RHO task. The results, shown in Fig. 5,

TABLE IV

TRAINING TIME (SECONDS) COMPARISONS ON THE PCA TASK. "IMP." REPRESENTS THE PERFORMANCE IMPROVEMENT RATIO OF OURS OVER UNROLLING

| Inner-Level Steps | 5 | 25 | 45 | 65 | 85 | 105 | 125 | 145 | 165 |
|---|---|---|---|---|---|---|---|---|---|
| Unrolling | 0.24 | 3.24 | 9.87 | 19.8 | 33.2 | 50.1 | 70.5 | 94.6 | 122 |
| Ours | 0.16 | 0.48 | 0.80 | 1.11 | 1.42 | 1.77 | 2.05 | 2.41 | 2.68 |
| Imp. | 50% | 58% | 113% | 168% | 224% | 273% | 334% | 383% | 445% |




Analyses on the PCA task (in the log scale). Analyses on the clustering task (in the log scale).

Fig. 3. Convergence analyses on the RMO task.

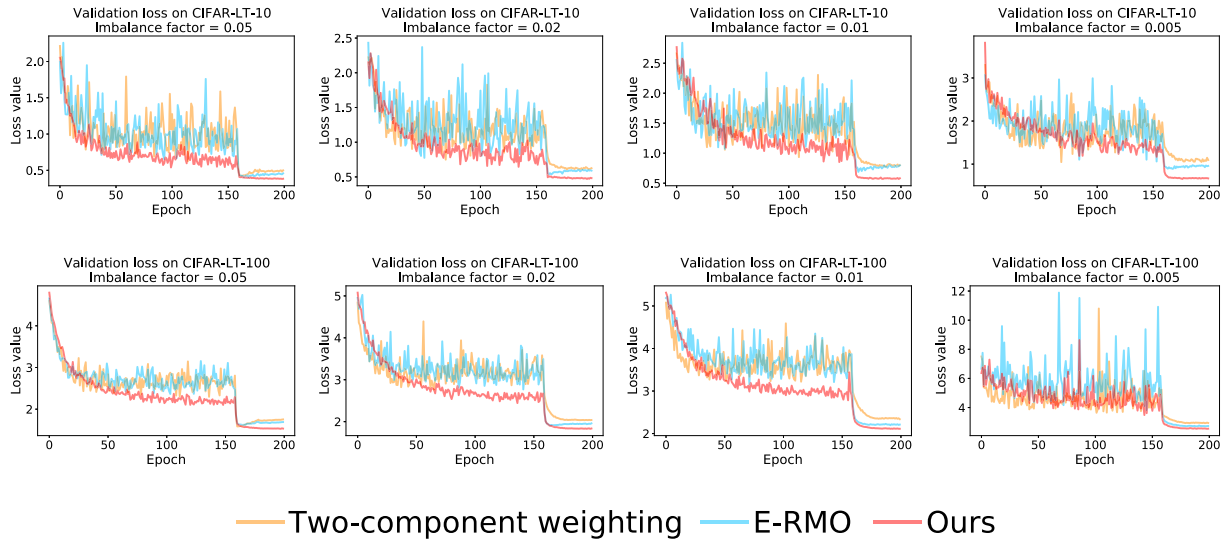



Fig. 4. Convergence analyses on the RHO task.

TABLE V

TRAINING MEMORY (MB) AND TIME (SECONDS) COMPARISONS ON THE RHO TASK. "IMP." REPRESENTS THE PERFORMANCE IMPROVEMENT RATIO OF OURS OVER UNROLLING

| Inner-Level steps | Training time | | | Training memory | | |
|---|---|---|---|---|---|---|
| | Unrolling | Ours | Imp. | Unrolling | Ours | Imp. |
| 3 | 749 | 250 | 200% | $4.78 \times 10^3$ | $2.56 \times 10^3$ | 88% |
| 4 | 782 | 291 | 169% | $6 \times 10^3$ | $2.56 \times 10^3$ | 134% |
| 5 | 849 | 324 | 162% | $7.27 \times 10^3$ | $2.56 \times 10^3$ | 184% |

demonstrate that our method achieves faster convergence and smaller converged loss values. On the RMO task, our method converges to a loss of 2.45 on both the training and testing sets, while E-RMO converges to 2.64. On the RHO task, our method converges to 0.10 on the training set and 3.01 on the validation set, whereas E-RMO converges to 0.39 and 3.30, respectively. These results indicate that our method consistently achieves lower final losses. Furthermore, we compute the norm of the gradients of the converged inner-level parameters. On the RMO task, the gradient norm is 0.0065 for our method, compared to 0.0233 for E-RMO. On the RHO task, our method yields a norm of 0.0008, while E-RMO yields 0.0012. These smaller gradient norms and lower final losses indicate that our method achieves higher precision in the obtained inner-level parameters.

*2) Accuracy Analyses:* Accuracy analyses are conducted on the RHO task and the RML task. Results of RHO and RML task are shown in Tables VII and VIII, respectively. Results show that the Riemannian implicit differentiation method achieves the best performance, demonstrating the superiority of the proposed method.

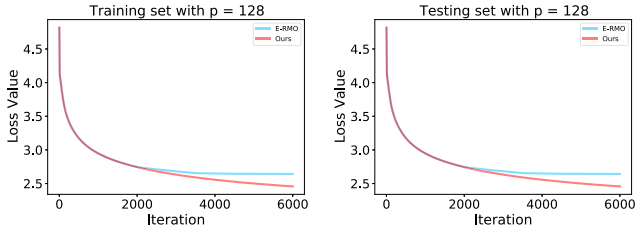
The reasons why our method outperforms MAML are twofold.

1) MAML is a bilevel optimization method in Euclidean spaces, whereas our method formulates this problem as a bilevel optimization in Riemannian manifolds. Concretely, we leverage hyperbolic geometry to represent image features, which is more effective in capturing the inherent hierarchical structures present in many

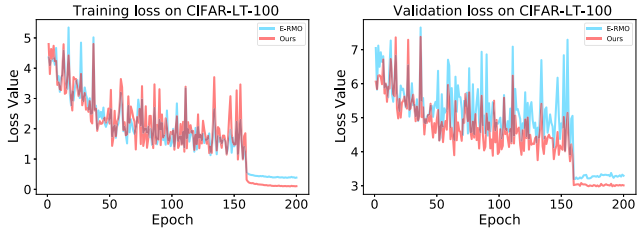
TABLE VI

TRAINING MEMORY (MB) COMPARISONS ON THE PCA TASK. "IMP." REPRESENTS THE PERFORMANCE IMPROVEMENT RATIO OF OURS OVER UNROLLING

| Inner-Level Steps | 5 | 25 | 45 | 65 | 85 | 105 | 125 | 145 | 165 |
|---|---|---|---|---|---|---|---|---|---|
| Unrolling | $5.87 \times 10^3$ | $6.42 \times 10^3$ | $7.05 \times 10^3$ | $7.68 \times 10^3$ | $8.31 \times 10^3$ | $9.34 \times 10^3$ | $9.98 \times 10^3$ | $10.62 \times 10^4$ | $11.03 \times 10^3$ |
| Ours | $5.68 \times 10^3$ | $5.68 \times 10^3$ | $5.68 \times 10^3$ | $5.68 \times 10^3$ | $5.68 \times 10^3$ | $5.68 \times 10^3$ | $5.68 \times 10^3$ | $5.68 \times 10^3$ | $5.68 \times 10^3$ |
| Imp. | 3% | 13% | 24% | 35% | 46% | 64% | 76% | 87% | 94% |




Analyses on the Riemannian meta-optimization task (PCA on Grassmann manifolds).




Analyses on the Riemannian hyperparameter optimization task.

Fig. 5. Comparison of different inner-level optimization processes.

TABLE VII

ACCURACY ANALYSES ON THE RHO TASK

| | CIFAR-LT-10 | | | | CIFAR-LT-100 | | | |
|---|---|---|---|---|---|---|---|---|
| **Imbalance Ratio** | **200** | **100** | **50** | **20** | **200** | **100** | **50** | **20** |
| Two-component weighting [69] | 29.34 | 23.59 | 19.49 | 13.54 | 60.69 | 56.65 | 51.47 | 44.38 |
| E-RMO | 26.2 | 22.04 | 18.32 | 14.73 | 59.04 | 55.42 | 50.56 | 43.95 |
| Ours | 20.40 | 17.49 | 14.74 | 11.54 | 57.45 | 52.07 | 47.64 | 41.25 |

TABLE VIII

ACCURACY ANALYSES ON THE RML TASK

| | mini-ImageNet | | tiered-ImageNet | |
|---|---|---|---|---|
| | **1-shot 5-way** | **5-shot 5-way** | **1-shot 5-way** | **5-shot 5-way** |
| MAML [72] | $51.03 \pm 0.50$ | $68.26 \pm 0.47$ | $58.58 \pm 0.49$ | $71.24 \pm 0.43$ |
| E-RMO | $63.38 \pm 0.36$ | $80.1 \pm 0.54$ | $70.06 \pm 0.66$ | $84.44 \pm 0.27$ |
| Ours | $64.5 \pm 0.23$ | $82.1 \pm 0.15$ | $71.56 \pm 0.46$ | $85.75 \pm 0.20$ |

TABLE IX

ANALYSIS OF NEUMANN SERIES ON CIFAR-LT-10/CIFAR-LT-100

| | CIFAR10-LT | | | | CIFAR100-LT | | | |
|---|---|---|---|---|---|---|---|---|
| Imbalance Ratio | 200 | 100 | 50 | 20 | 200 | 100 | 50 | 20 |
| Neumann series $K = 2$ | 20.40 | 17.49 | 14.74 | 11.54 | 57.45 | 52.07 | 47.64 | 41.25 |
| Neumann series $K = 4$ | 20.60 | 17.70 | 14.31 | 11.91 | 57.70 | 52.91 | 47.42 | 41.58 |
| Neumann series $K = 6$ | 20.75 | 18.05 | 14.53 | 11.85 | 57.75 | 52.58 | 47.76 | 41.14 |
| Neumann series $K = 8$ | 20.09 | 16.94 | 14.48 | 11.86 | 57.48 | 52.31 | 48.28 | 41.38 |
| Neumann series $K = 10$ | 20.87 | 17.3 | 14.96 | 12.13 | 56.95 | 52.81 | 47.75 | 41.32 |

real-world datasets, leading to more expressive feature representations and improved downstream task performance [4].

2) MAML is an exact unrolling method and, as such, suffers from the well-known dilemma between computational cost and the bias issue. Specifically, a short inner-level optimization length leads to unstable optimization and suboptimal solutions, while a long inner-level optimization length incurs significant computational loads. In contrast, our method decouples the costs of outer gradient computation from the inner-level optimization process. This allows us to use a larger number of inner-level iteration steps $T$ to improve optimization stability and quality without increasing time or memory complexity, ultimately leading to better performance.

### F. Analysis of the Neumann Series K

We utilize the Neumann series to approximate the gradients of the outer-level parameters $\boldsymbol{\theta}$. As shown in Theorem 2, the approximation error is related to $K$, where more Neumann series can lead to less approximation error and more training time. We utilize the Riemannian hyperparameter-optimization, i.e., long-tailed classification task to analyze the Neumann series $K$. Results are shown in Table IX, which exhibit that our method is not agnostic about the Neumann series $K$. We consider the reason is that the approximation error is small enough when $K = 2$, and the decline of the approximation error is limited by increasing $K$. This also demonstrates that the approximation error in our method is controllable.

## VII. Conclusion

In this article, we have extended the implicit differentiation method to various Riemannian bilevel optimization tasks. We have presented a Riemannian implicit differentiation method for Riemannian bilevel optimization by modeling the inner-level optimization as a root-finding process of the fixed-point equation. The proposed method can remove any requirements for case-by-case derivations and can be applied to a variety of different Riemannian optimization tasks directly. Moreover, we have presented convergence analysis and approximation error analysis for our method, which demonstrates our method can be applied to various Riemannian optimization tasks effectively. Experiments of three Riemannian optimization tasks on different manifolds demonstrate the superiority of our method.

Neural ordinary differential equations (ODE)-type methods have shown effective performance in certain scenarios. Applying such methods to bilevel optimization problems could be a promising direction. Specifically, the dynamics of various inner-level optimization processes across different settings can be formulated using neural ODE-type frameworks, and the corresponding outer gradients can then be computed via the implicit function theorem. However, due to the nonlinear constraints inherent in Riemannian optimization, these methods are not directly applicable to Riemannian bilevel problems. In particular, Riemannian inner-level optimization involves pointwise, time-independent operations such as retraction and orthogonal projection, which do not define continuous-time flows as ODEs do. As a result, Riemannian inner-level optimization cannot be interpreted as Euler discretizations of continuous dynamical systems, making it infeasible to obtain inner-level solutions using neural ODE-type methods. Therefore, current neural ODE-type methods cannot be directly applied to Riemannian bilevel optimization. In the future, we are interested in addressing the aforementioned challenges and exploring continuous-time Riemannian dynamics that may allow the application of neural ODE-type methods to Riemannian bilevel optimization.

A possible failure mode of our method arises when the inner-level optimization problem admits multiple solutions. In such cases, the inner-level parameters may converge to a suboptimal solution, which in turn leads the outer-level optimization to also converge to suboptimal parameters. This cascading effect can ultimately result in degraded overall performance. In the future, we will propose an effective method for Riemannian bilevel optimization where inner-level objectives have multiple solutions.

## References


[1] R. Liu, J. Gao, J. Zhang, D. Meng, and Z. Lin, "Investigating bi-level optimization for learning and vision from a unified perspective: A survey and beyond," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 12, pp. 10045–10067, Dec. 2022.

[2] Z. Gao, Y. Wu, Y. Jia, and M. Harandi, "Hyperbolic feature augmentation via distribution estimation and infinite sampling on manifolds," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 35, 2022, pp. 34421–34435.

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

[4] V. Khrulkov, L. Mirvakhabova, E. Ustinova, I. Oseledets, and V. Lempitsky, "Hyperbolic image embeddings," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 6418–6428.

[5] X. Fan, Y. Wu, Z. Gao, Y. Jia, and M. Harandi, "Efficient Riemannian meta-optimization by implicit differentiation," in *Proc. AAAI Conf. Artif. Intell. (AAAI)*, vol. 36, 2022, pp. 3733–3740.

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

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

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

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

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

[11] H. Zhang, S. J. Reddi, and S. Sra, "Riemannian SVRG: Fast stochastic optimization on Riemannian manifolds," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 29, 2016, pp. 4599–4607.

[12] H. Kasai, H. Sato, and B. Mishra, "Riemannian stochastic recursive gradient algorithm," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2018, pp. 2516–2524.

[13] 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, Jan. 2019.

[14] G. Bécigneul and O.-E. Ganea, "Riemannian adaptive optimization methods," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–10.

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

[16] H. Sato, "Riemannian conjugate gradient methods: General framework and specific algorithms with convergence analyses," *SIAM J. Optim.*, vol. 32, no. 4, pp. 2690–2717, Dec. 2022.

[17] J. Chen et al., "Decentralized Riemannian conjugate gradient method on the Stiefel manifold," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2023, pp. 1–12.

[18] F. Huang and S. Gao, "Gradient descent ascent for minimax problems on Riemannian manifolds," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 7, pp. 8466–8476, Jul. 2023.

[19] P. Zhang, J. Zhang, and S. Sra, "Sion's minimax theorem in geodesic metric spaces and a Riemannian extragradient algorithm," *SIAM J. Optim.*, vol. 33, no. 4, pp. 2885–2908, Dec. 2023.

[20] M. I. Jordan, T. Lin, and E.-V. Vlatakis-Gkaragkounis, "First-order algorithms for min–max optimization in geodesic metric spaces," in *Proc. Adv. Neural Inf. Process. Syst.*, pp. 6557–6574, May 2022.

[21] X. Fan, Z. Gao, Y. Wu, Y. Jia, and M. Harandi, "Learning a gradient-free Riemannian optimizer on tangent spaces," in *Proc. AAAI Conf. Artif. Intell. (AAAI)*, May 2021, vol. 35, no. 8, pp. 7377–7384. [Online]. Available: https://ojs.aaai.org/index.php/AAAI/article/view/16905

[22] H. V. Stackelberg, *The Theory of the Market Economy*. WilliamHodge, 1952. [Online]. Available: https://books.google.co.jp/books?id=o3ceAAAAIAAJ

[23] M. Mackay, P. Vicol, J. Lorraine, D. Duvenaud, and R. Grosse, "Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019, pp. 1–14.

[24] T. Okuno, A. Takeda, A. Kawana, and M. Watanabe, "On $\ell_p$-hyperparameter learning via bilevel nonsmooth optimization," *J. Mach. Learn. Res.*, vol. 22, no. 245, pp. 1–47, 2021.

[25] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil, "Bilevel programming for hyperparameter optimization and meta-learning," in *Proc. 35th Int. Conf. Mach. Learn.*, vol. 80, Jul. 2018, pp. 1568–1577. [Online]. Available: https://proceedings.mlr.press/v80/franceschi18a.html

[26] T. Hospedales, A. Antoniou, P. Micaelli, and A. Storkey, "Meta-learning in neural networks: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 9, pp. 5149–5169, Sep. 2022.

[27] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–11.

[28] D. Lian et al., "Towards fast adaptation of neural architectures with meta learning," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2019, pp. 1–13.

[29] Y. Tian, L. Shen, L. Shen, G. Su, Z. Li, and W. Liu, "AlphaGAN: Fully differentiable architecture search for generative adversarial networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 10, pp. 6752–6766, Oct. 2022.

[30] H. Zhang et al., "Bi-level actor-critic for multi-agent coordination," in *Proc. AAAI Conf. Artif. Intell.*, vol. 34, May 2020, pp. 7325–7332.

[31] S. Ghadimi and M. Wang, "Approximation methods for bilevel programming," 2018, *arXiv:1802.02246*.

[32] R. Grazzi, L. Franceschi, M. Pontil, and S. Salzo, "On the iteration complexity of hypergradient computation," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 3748–3758.

[33] K. Ji, J. Yang, and Y. Liang, "Bilevel optimization: Convergence analysis and enhanced design," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2020, pp. 4882–4892.

[34] R. Liu, P. Mu, X. Yuan, S. Zeng, and J. Zhang, "A generic first-order algorithmic framework for bi-level programming beyond lower-level singleton," in *Proc. Int. Conf. Mach. Learn. (ICML)*, vol. 1, 2020, pp. 6305–6315.

[35] R. Liu, Y. Liu, W. Yao, S. Zeng, and J. Zhang, "Averaged method of multipliers for bi-level optimization without lower-level strong convexity," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2023, pp. 21839–21866.

[36] Y. Wu, M. Ren, R. Liao, and R. Grosse, "Understanding short-horizon bias in stochastic meta-optimization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–36.

[37] T. Chen et al., "Training stronger baselines for learning to optimize," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 33, 2020, pp. 7332–7343.

[38] 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. (ICML)*, Jun. 2021, pp. 10553–10563.

[39] A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine, "Meta-learning with implicit gradients," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 32, 2019, pp. 113–124.

[40] J. Lorraine, P. Vicol, and D. Duvenaud, "Optimizing millions of hyperparameters by implicit differentiation," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2019, pp. 1540–1552.

[41] 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.*, Jun. 2021, pp. 16601–16610.

[42] A. Navon, I. Achituve, H. Maron, G. Chechik, and E. Fetaya, "Auxiliary learning by implicit differentiation," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, Jul. 2020, pp. 1–12.

[43] P. Khanduri et al., "Linearly constrained bilevel optimization: A smoothed implicit gradient approach," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2023, pp. 16291–16325.

[44] Q. Bertrand, Q. Klopfenstein, M. Blondel, S. Vaiter, A. Gramfort, and J. Salmon, "Implicit differentiation of lasso-type models for hyperparameter optimization," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2020, pp. 810–821.

[45] B. Liu, M. Ye, S. Wright, P. Stone, and Q. Liu, "Bome! Bilevel optimization made easy: A simple first-order approach," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 35, 2022, pp. 17248–17262.

[46] S. Lu, "SLM: A smoothed first-order Lagrangian method for structured constrained nonconvex optimization," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 36, 2024, pp. 80414–80454.

[47] R. Liu, Z. Liu, W. Yao, S. Zeng, and J. Zhang, "Moreau envelope for nonconvex bi-level optimization: A single-loop and hessian-free solution strategy," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2024, pp. 31566–31596.

[48] J. M. Lee, *Riemannian Manifolds: An Introduction to Curvature*, vol. 176. Cham, Switzerland: Springer, 2006.

[49] C. Hu, K.-Y. Zhang, T. Yao, S. Ding, and L. Ma, "Rethinking generalizable face anti-spoofing via hierarchical prototype-guided distribution refinement in hyperbolic space," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2024, pp. 1032–1041.

[50] Y. Wang, Y. Li, and S. Wang, "G3-LQ: Marrying hyperbolic alignment with explicit semantic-geometric modeling for 3D visual grounding," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2024, pp. 13917–13926.

[51] 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, Jan. 1998.

[52] T. Long, Y. Sun, J. Gao, Y. Hu, and B. Yin, "Domain adaptation as optimal transport on Grassmann manifolds," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 34, no. 10, pp. 7196–7209, Oct. 2023.

[53] R. Wang, X.-J. Wu, and J. Kittler, "SymNet: A simple symmetric positive definite manifold deep learning method for image set classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 5, pp. 2208–2222, May 2022.

[54] R. Wang, X.-J. Wu, Z. Chen, C. Hu, and J. Kittler, "SPD manifold deep metric learning for image set classification," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 7, pp. 8924–8938, Jul. 2024.

[55] P.-A. Absil, R. Mahony, and R. Sepulchre, *Optimization Algorithms on Matrix Manifolds*. Princeton, NJ, USA: Princeton Univ. Press, 2009.

[56] A. Shaban, C.-A. Cheng, N. Hatch, and B. Boots, "Truncated back-propagation for bilevel optimization," in *Proc. 22nd Int. Conf. Artif. Intell. Statist.*, 2018, pp. 1723–1732.

[57] B. Christianson, "Automatic hessians by reverse accumulation," *IMA J. Numer. Anal.*, vol. 12, no. 2, pp. 135–150, 1992.

[58] A. Griewank and A. Walther, *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Philadelphia, PA, USA: SIAM, 2008.

[59] A. G. Baydin, B. A. Pearlmutter, A. Radul, and J. M. Siskind, "Automatic differentiation in machine learning: A survey," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 5595–5637, 2017.

[60] A. Griewank, "Some bounds on the complexity of gradients, jacobians, and hessians," in *Complexity in Numerical Optimization*. Singapore: World Scientific, 1993, pp. 128–162.

[61] J. Li, J. Yu, B. Liu, Z. Wang, and Y. Nie, "Achieving hierarchy-free approximation for bilevel programs with equilibrium constraints," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2023, pp. 20312–20335.

[62] D. Sow, K. Ji, and Y. Liang, "On the convergence theory for hessian-free bilevel algorithms," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2021, pp. 4136–4149.

[63] G. Kylberg, "Kylberg texture dataset v. 1.0," Swedish Univ. Agricult. Sci., Uppsala Univ., Uppsala, Sweden, Tech. Rep. (Blue series) 35, Sep. 2011. [Online]. Available: http://www.cb.uu.se/~gustaf/texture/

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

[65] Y. Cui, M. Jia, T.-Y. Lin, Y. Song, and S. Belongie, "Class-balanced loss based on effective number of samples," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2019, pp. 9268–9277.

[66] Y. Cui, Y. Song, C. Sun, A. Howard, and S. Belongie, "Large scale fine-grained categorization and domain-specific transfer learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, Jun. 2018, pp. 4109–4118.

[67] M. Ren, W. Zeng, B. Yang, and R. Urtasun, "Learning to reweight examples for robust deep learning," in *Proc. Int. Conf. Mach. Learn.*, Mar. 2018, pp. 4334–4343.

[68] J. Shu et al., "Meta-weight-Net: Learning an explicit mapping for sample weighting," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, Jun. 2019, pp. 1919–1930.

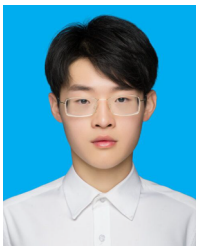
[69] M. A. Jamal, M. Brown, M.-H. Yang, L. Wang, and B. Gong, "Rethinking class-balanced methods for long-tailed visual recognition from a domain adaptation perspective," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 7610–7619.

[70] H. Zhang, L. Zhu, X. Wang, and Y. Yang, "Divide and retain: A dual-phase modeling for long-tailed visual recognition," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 35, no. 10, pp. 13538–13549, Oct. 2024.

[71] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Univ. Toronto, Toronto, ON, Canada, Tech. Rep., 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[72] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. 34th Int. Conf. Mach. Learn.*, vol. 70, Aug. 2017, pp. 1126–1135.

[73] S. Baik, S. Hong, and K. M. Lee, "Learning to forget for meta-learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 2376–2384.
[74] 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. Represent. (ICLR)*, 2018, pp. 1–10.
[75] S. Baik, M. Choi, J. Choi, H. Kim, and K. M. Lee, "Meta-learning with adaptive hyperparameters," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2020, pp. 20755–20765.
[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. (CVPR)*, Jun. 2019, pp. 10657–10665.
[77] J. Xu, J.-F. Ton, H. Kim, A. R. Kosiorek, and Y. W. Teh, "MetaFun: Meta-learning with iterative functional updates," in *Proc. Int. Conf. Mach. Learn. (ICML)*, vol. 1, 2020, pp. 10617–10627.
[78] C. Simon, P. Koniusz, R. Nock, and M. Harandi, "Adaptive subspaces for few-shot learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Jun. 2020, pp. 4135–4144.
[79] 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. (ICCV)*, Oct. 2021, pp. 9062–9071.
[80] 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. (ICCV)*, Oct. 2021, pp. 9465–9474.
[81] A. A. Rusu et al., "Meta-learning with latent embedding optimization," in *Proc. Int. Conf. Learn. Represent. (ICLR)*, 2018, pp. 1–13.
[82] P. Tian, W. Li, and Y. Gao, "Consistent meta-regularization for better meta-knowledge in few-shot learning," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 33, no. 12, pp. 7277–7288, Dec. 2022.
[83] P. Fang, M. Harandi, and L. Petersson, "Kernel methods in hyperbolic spaces," in *Proc. IEEE/CVF Int. Conf. Comput. Vis. (ICCV)*, Oct. 2021, pp. 10665–10674.
[84] Z. Gao, Y. Wu, M. Harandi, and Y. Jia, "Curvature-adaptive meta-learning for fast adaptation to manifold data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 45, no. 2, pp. 1545–1562, Feb. 2023.
[85] P. Fang, M. Harandi, Z. Lan, and L. Petersson, "Poincaré kernels for hyperbolic representations," *Int. J. Comput. Vis.*, vol. 131, no. 11, pp. 2770–2792, 2023.
[86] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 30, 2017, pp. 4077–4087.
[87] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *Proc. Adv. Neural Inf. Process. Syst. (NeurIPS)*, vol. 29, 2016, pp. 3637–3645.



**Xiaomeng Fan** received the B.S. degree in information and computing science from Hebei University of Technology (HEBUT), Hebei, China, in 2020. He is currently pursuing the Ph.D. degree with Beijing Laboratory of Intelligent Information Technology, BIT, Beijing, China.

His research interests include pattern recognition, machine learning, and computer vision on Riemannian manifolds.

**Yuwei Wu** (Member, IEEE) received the Ph.D. degree in computer science from Beijing Institute of Technology (BIT), Beijing, China, in 2014.

From August 2014 to August 2016, he was a Post-Doctoral Research Fellow at the Rapid-Rich Object Search (ROSE) Lab, School of Electrical and Electronic Engineering (EEE), Nanyang Technological University (NTU), Singapore. He is now a Tenured Associate Professor at the School of Computer Science, BIT. His research interests include computer vision and machine learning.

**Zhi Gao** received the B.S. and Ph.D. degrees in computer science from Beijing Institute of Technology (BIT), Beijing, China, in 2017 and 2023, respectively.

He was a Post-Doctoral Research Fellow at the School of Intelligence Science and Technology, Peking University, Beijing, from 2023 to 2025. He is an Associate Professor (tenure-track) at the School of Computer Science and Technology, BIT. His current research interests include computer vision, machine learning, multimodal learning, and Riemannian geometry.

**Zhipeng Lu** received the Ph.D. degree in mathematics from Indiana University Bloomington, IN, USA, in 2018.

From September 2018 to August 2021, he was a Post-Doctoral Research Fellow at the Mathematical Institute, University of Göttingen, Göttingen, Germany. He is now a Senior Lecturer at the Engineering Department, Shenzhen MSU-BIT University, Shenzhen, China. His research interests include theoretical computation, algebraic combinatorics, and mathematical foundations of machine learning.

**Feng Li** received the M.S. degree from the School of Software, Tsinghua University, Beijing, China, in 2017.

He is currently a Researcher with the Curvature Space Representation Learning Technology Project, Beijing Institute of Technology, Beijing. His research interests include representation learning, user behavior modeling, and large language models.

**Mehrtash Harandi** (Member, IEEE) is a Senior Lecturer with the Department of Electrical and Computer Systems Engineering, Monash University, Clayton, VIC, Australia. He is also a contributing research scientist with the Machine Learning Research Group (MLRG), Data61-CSIRO, Eveleigh, NSW, Australia, and an Associate Investigator at Australian Center for Robotic Vision (ACRV), Brisbane, QLD, Australia. 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 B.S., M.S., and Ph.D. 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), Pittsburgh, PA, USA, from 1995 to 1997. He is currently a Professor at the Department of Engineering, Shenzhen MSU-BIT University (SMBU), Shenzhen, China, where he serves as the Director of Guangdong Provincial Key Laboratory of Machine Perception and Intelligent Computing. His interests include computer vision, vision-based human-computer interaction (HCI) and human-robot interaction (HRI), and intelligent robotics.