Curvature-Adaptive Meta-Learning for Fast Adaptation to Manifold Data

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

Abstract—Meta-learning methods are shown to be effective in quickly adapting a model to novel tasks. Most existing meta-learning methods represent data and carry out fast adaptation in euclidean space. In fact, data of real-world applications usually resides in complex and various Riemannian manifolds. In this paper, we propose a curvature-adaptive meta-learning method that achieves fast adaptation to manifold data by producing suitable curvature. Specifically, we represent data in the product manifold of multiple constant curvature spaces and build a product manifold neural network as the base-learner. In this way, our method is capable of encoding complex manifold data into discriminative and generic representations. Then, we introduce curvature generation and curvature updating schemes, through which suitable product manifolds for various forms of data manifolds are constructed via few optimization steps. The curvature generation scheme identifies task-specific curvature initialization, leading to a shorter optimization trajectory. The curvature updating scheme automatically produces appropriate learning rate and search direction for curvature, making a faster and more adaptive optimization paradigm compared to hand-designed optimization schemes. We evaluate our method on a broad set of problems including few-shot classification, few-shot regression, and reinforcement learning tasks. Experimental results show that our method achieves substantial improvements as compared to meta-learning methods ignoring the geometry of the underlying space.

Index Terms—Meta-learning, manifold data, constant curvature space, product manifold, curvature

1 Introduction

An important problem in machine learning and computer vision is to quickly adapt a model to novel tasks with unseen and limited data [1], [2], [3], [4]. Meta-learning methods have shown impressive performance in achieving such fast adaptation [5], [6], [7]. The central idea is to train a meta-learner on some seen tasks and generalize the acquired knowledge to adapt a base-learner to novel tasks. Most existing meta-learning methods represent data and carry out fast adaptation in euclidean space. This is mainly because euclidean space provides an appealing vectorial structure, allowing us to implement various basic operations (e.g., addition, inner product, and measuring distances) with ease. However, data in most machine learning tasks does not intrinsically comply with the euclidean geometry. For example, face images can be better modeled by a Riemannian manifold with spherical structures [8]. As a result, several studies opt to make use of Riemannian geometry for modeling and inference [9], [10], [11]. Focusing on meta-learning, modeling manifold data using euclidean space will harm the structure of data, leading to inferior generalization to novel tasks [12], [13]. Therefore, it is promising to study meta-learning in Riemannian manifold.

Recent work in constant curvature space [14], [15], [16] offers a feasible perspective to meta-learning in Riemannian manifold. Constant curvature space has a smooth Riemannian geometry and has been shown to be superior to euclidean spaces in some machine learning problems (e.g., image retrieval [12]). Curvature is a fundamental concept in constant curvature space, which characterizes the geometry of the space, representing the deviation of the space from the flat euclidean space [17]. If the curvature is equal to zero, a constant curvature space is equivalent to a euclidean space. Basic operations in constant curvature space are not overly complicated, in turn bringing convenience to algorithmic designs. Moreover, combining multiple constant curvature spaces into a Cartesian product leads to a product manifold [18], capable of modeling complex data with high flexibility [19]. This motivates us to make use of the product manifold of constant curvature spaces to design a meta-learning method for fast adaptation to manifold data.

To employ the product manifold for meta-learning, we need to address two challenges. (1) Making the base-learner faithful to the geometry of the product manifold is not straightforward and requires advanced modeling. Essentially, employing existing models (e.g., CNNs) inevitably destroys the structure of the product manifold. (2) A meta-learning method requires handling diverse manifold structures of unseen data. For example, medical images about brain activity exhibit the cyclical structure [20], and fine-grained images are modeled better by hierarchical structures [21]. Given limited data in meta-learning, it is

- *Zhi Gao, Yuwei Wu, and Yunde Jia are with the Beijing Laboratory of Intelligent Information Technology, School of Computer Science, Beijing Institute of Technology (BIT), Beijing 100081, China. E-mail: {gaozhi_2017, wuyuwei, jiayunde}@bit.edu.cn.*
- *Mehrtash Harandi is with the Department of Electrical and Computer Systems Eng., Monash University, Melbourne, VIC 3800, Australia, and also with Data61-CSIRO, Australia. E-mail: mehrtash.harandi@monash.edu.*

Manuscript received 25 Aug. 2021; revised 12 Feb. 2022; accepted 26 Mar. 2022. Date of publication 5 Apr. 2022; date of current version 6 Jan. 2023.
This work was supported in part by the Natural Science Foundation of China (NSFC) under Grants 62172041 and 62176021.
(Corresponding author: Yuwei Wu.)
Recommended for acceptance by C. G. M. Snoek.
Digital Object Identifier no. 10.1109/TPAMI.2022.3164894

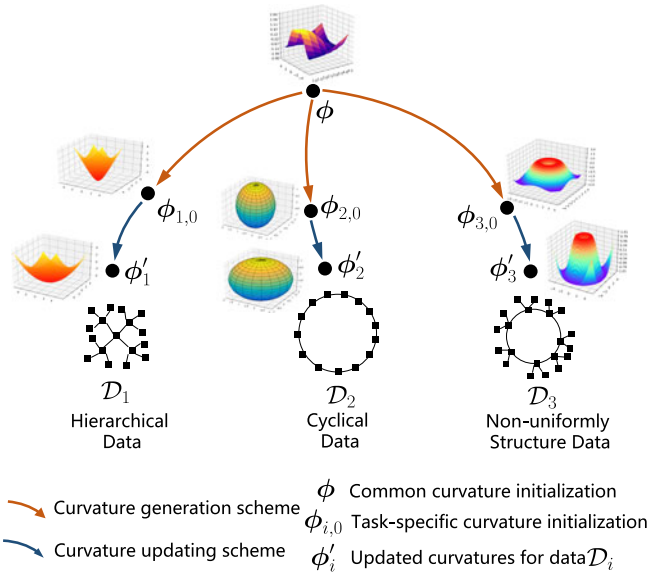


Fig. 1. A conceptual visualization of the proposed method. In real-world, data usually has various forms of manifold structures, such as hierarchical data and cyclical data. We propose a curvature-adaptive meta-learning method that encodes data using a product manifold of constant curvature spaces and performs fast adaptation to manifold data by updating curvatures. Our method contains a curvature generation scheme and a curvature updating scheme, which assigns task-specific curvature initialization and carries out adaptive optimization on curvatures, respectively. Our method can construct suitable geometry for various forms of manifold data via few optimization steps.

challenging to quickly adapt the geometry of the product manifold to match diverse manifold structures.

In this paper, we propose a curvature-adaptive meta-learning method that addresses the above two challenges, capable of fast adaptation to manifold data. To address the first challenge, we introduce a product manifold neural network as the base-learner. In our design, we make use of the tangent bundle of the product manifold to realize differentiable layers of the neural network. Since tangent spaces are local approximation to the manifold (to the first order), the representation ability of the network may get negatively affected if tangent points are not flexible and far from features in hand [22]. Hence, we learn tangent points throughout the network in the hope of better capturing intrinsic properties of data. In this way, we generalize conventional neural network architectures (e.g., fully-connected layer and convolutional block) to the product manifold of constant curvature spaces via learnable tangent spaces.

To address the second challenge, we introduce a curvature generation scheme and a curvature updating scheme to update curvatures in the product manifold neural network (see Fig. 1 for a conceptual diagram). Specifically, we learn a common curvature initialization and curvature adjustment by the curvature generation scheme. Given a novel task, the curvature adjustment modulates the common curvature initialization to a task-specific curvature initialization, shortening the optimization trajectory. The curvature updating scheme learns to produce adaptive learning rate and search direction for each curvature, leading to faster and more adaptive optimization as compared to a fixed updating scheme. In this case, suitable product manifolds are constructed for novel tasks via few optimization steps on curvatures. We also learn to initialize parameters (i.e., weights) of the product manifold neural network in the meta-learning framework and update them for given tasks. Note that, if the curvature is set to zero and the learning rate is kept fixed in the optimization, the product manifold neural network realizes a classic neural network in euclidean space, and our method is reduced to the celebrated MAML algorithm [1]. We conduct experiments on the few-shot classification, few-shot regression, and reinforcement learning tasks. Experimental results show the efficiency and effectiveness of our method, demonstrating the benefits of employing Riemannian structures as compared to meta-learning methods that are oblivious to the geometry of the underlying space.

In summary, our contributions are three-fold.

1. We propose a curvature-adaptive meta-learning method that carries out fast adaptation for novel tasks by updating curvatures in the product manifold of constant curvature spaces. Our method is capable of performing fast adaptation to various forms of manifold data, making the method more generic.
2. We introduce a product manifold neural network as the base-learner, which generalizes conventional neural network architectures to the product manifold of constant curvature spaces. Our network learns curvatures and tangent spaces in each layer, hence comes with great flexibility to capture the intrinsic structure of data.
3. We generate task-specific curvature initialization, and adaptive learning rates and search directions for curvatures, through which suitable product manifolds are constructed via few optimization steps on curvatures.

2 Related Work

2.1 Meta-Learning

Existing meta-learning methods can be broadly divided into three categories: metric-based, model-based, and optimization-based methods. Early metric-based methods [23], [24] learn a common embedding space for classification. Recent advances suggest that employing a common embedding space might not be discriminative enough and adapt the embedding space to specific tasks. For example, Ye *et al.* [25] and Kang *et al.* [26] learned a task-specific transformation, Li *et al.* [27] employed a task-specific loss function, Liu *et al.* [6] generated task-specific class prototypes, Bateni *et al.* [28] and Qiao [29] proposed task-specific distance measures, and Simon *et al.* [30] learned task-specific subspaces.

Model-based methods wrap up adaptation as a forward propagation of a black-box model based on task representations. Santoro *et al.* [7] utilized loss gradients as task representations to predict parameters of a neural network. Xu *et al.* [31] introduced an encoder-decoder architecture to generate infinite-dimensional task representations. Mishra *et al.* [32] used temporal convolutions and attention mechanisms to improve the memory capacity of model-base meta-learning. Zhen *et al.* [33] utilized task representations to produce task-specific kernel functions, showing state-of-the-art performance. Edwards and Storkey [34] learned to estimate the statistics of data as task representations, avoiding the need of external memory modules.

Our method belongs to optimization-based methods that achieve fast adaptation through an optimization process. Existing optimization-based methods mainly focus on data

in euclidean space. The pioneering work of Finn *et al.* [1], model-agnostic meta-learning (MAML), learns a good base-learner initialization. To reduce the computational complexity of MAML, Rajeswaran *et al.* [5] derived implicit differentiation to learn the initialization, Jamal *et al.* [35] utilized a teacher-student strategy to avoid the process of updating the base-learner, being immune to the risk of vanishing or exploding gradients, and Zou and Lu [36] proposed a Bayesian meta-learning method to avoid backpropagation. To overcome the overfitting issue, Baik *et al.* proposed task-and-layer-specific attenuation [37] and task-specific loss functions [38], enabling the method to be applied to different problem domains, and Zintgraf *et al.* [2] only updated part of initial parameters. Several methods show that the optimization process can also be cast as learning problem. Andrychowicz *et al.* [39] trained neural networks for optimization. Ravi and Larochelle [40] utilized the LSTM structure for gradient descent. New developments include the work of Li *et al.* [41] and Baik *et al.* [42] that learns both parameter initialization and the optimizer.

In contrast to the previous art in optimization-based methods, we focus on manifold data that is more generic in practice. Recently, Khrulkov *et al.* [12] and Qi *et al.* [13] proposed metric-based meta-learning methods for manifold data by adapting a non-euclidean distance measure to novel tasks. The two works focus on a specific type of data, and use a fixed geometry to represent data. Besides, they mainly work for classification applications, due to the dependency on the distance measure. In contrast, our method can perform fast adaptation to various forms of manifold data by using the product manifold and updating curvatures, making the method more generic. By adapting a neural network to manifold data instead of only a distance measure, our method can be applied to a wider set of applications (e.g., regression) based on the network design.

2.2 Learning in Constant Curvature Space

Constant curvature space has become an alternative to euclidean space in many machine learning problems (e.g., image retrieval [12] and language processing [43]), because constant curvature space can provide powerful representations for multiple types of data. Existing methods exploring constant curvature space can be divided into two categories: fixed-curvature methods and adaptive-curvature methods. Early work utilizes constant curvature space with fixed curvature for the problem in hand. Some methods set negative curvature in constant curvature space, that is, the hyperbolic space, which is shown to be successful in capturing the hierarchical structure in data. For example, Liu *et al.* [14], Long *et al.* [16], and Yan *et al.* [21] learned visual embeddings in hyperbolic space. Ganea *et al.* [17] and Shimizu *et al.* [44] designed hyperbolic neural networks, extracting features of hierarchical data with success. Liu *et al.* [45], Zhang *et al.* [46], and Dai *et al.* [47] developed hyperbolic graph networks. There also exists methods that study dimensionality reduction [48] and clustering [49] in hyperbolic space. On the other end of the spectrum, several methods opt for positive curvature to model the spherical structure in data (e.g., medical images about brain activity). Some notable studies are the work of Defferrard *et al.* [20] on the spherical neural network, the work of Grattarola *et al.* [15] on change detection for the spherical data, and the work of Wilson *et al.* [50] that learns embeddings on surfaces with positive curvature. The aforementioned studies regard curvature as a hyperparameter of the model and require human involvement for proper tuning.

To address this issue, adaptive-curvature methods have emerged. The underlying idea is to learn curvature using a penalty term from the data. Bachmann *et al.* [51], Chami *et al.* [52], and Zhang *et al.* [53] learned the curvature of a constant curvature graph network, Skopek *et al.* [18] and Park *et al.* [54] updated curvatures for auto-encoders in constant curvature space, and Gu *et al.* [19] searched the optimal curvature for embedding purposes. Compared with these methods, our method utilizes the meta-learning technique and is capable of quickly identifying the suitable curvature for a given task with limited annotated data. By assigning task-specific curvature initialization and applying adaptive updating scheme on curvatures, our method can construct suitable constant curvature spaces for various forms of manifold data through few updating steps. Besides, compared with existing neural networks in constant curvature space (e.g., [17], [45]), which use a common tangent space at the origin for all features, we apply learnable tangent spaces in each layer, reducing approximation error of the tangent spaces to the manifold.

3 Mathematical Background

Throughout this paper, we denote vectors by bold lower-case letters, e.g., $\boldsymbol{u}$, matrices by bold upper-case letters, e.g., $\boldsymbol{W}$, and tensors by blackboard upper-case letters, e.g., $\mathbb{X}$.

3.1 Riemannian Manifold

A Riemannian manifold $\mathcal{M}$ is a topological space that locally resembles euclidean space and can be understood as a generalization of the notion of surface to higher-dimensional space. Similar to euclidean space, the shortest path between two points on Riemannian manifold is a curve and is called a geodesic.[1] For a point $\boldsymbol{u} \in \mathcal{M}$, its tangent space $T_{\boldsymbol{u}}\mathcal{M}$ is a euclidean space that approximates $\mathcal{M}$ linearly at $\boldsymbol{u}$. It contains all vectors tangent to $\mathcal{M}$ at $\boldsymbol{u}$. A Riemannian manifold is endowed with a metric $\varrho : T_{\boldsymbol{u}}\mathcal{M} \times T_{\boldsymbol{u}}\mathcal{M} \rightarrow \mathbb{R}$ that induces an inner product structure $\langle \cdot, \cdot \rangle_{\boldsymbol{u}}$ on the tangent space $T_{\boldsymbol{u}}\mathcal{M}$, i.e., for two points $\boldsymbol{q}, \boldsymbol{s} \in T_{\boldsymbol{u}}\mathcal{M}$, $\langle \boldsymbol{q}, \boldsymbol{s} \rangle_{\boldsymbol{u}} = \varrho(\boldsymbol{q}, \boldsymbol{s})$. The norm $\| \cdot \|$ on the tangent space is also defined by the metric: $\|\boldsymbol{q}\| = \sqrt{\varrho(\boldsymbol{q}, \boldsymbol{q})}$, $\boldsymbol{q} \in T_{\boldsymbol{u}}\mathcal{M}$.

3.2 Constant Curvature Space

A d-dimensional constant curvature space $\mathcal{M}_K^d$ is a smooth Riemannian manifold with a curvature K that represents the deviation of $\mathcal{M}_K^d$ from a flat space [55]. For $\boldsymbol{u} \in \mathcal{M}_K^d$, its tangent space is denoted by $T_{\boldsymbol{u}}\mathcal{M}_K^d$. We will use the gyrovector space [56] to work with constant curvature space. In particular, the following operations are essential to our developments.

Inner Product. For $\boldsymbol{q}, \boldsymbol{s} \in T_{\boldsymbol{u}}\mathcal{M}_K^d$, their inner product is

$$\langle \boldsymbol{q}, \boldsymbol{s} \rangle_{\boldsymbol{u}} = (\lambda_{\boldsymbol{u}}^K)^2 \langle \boldsymbol{q}, \boldsymbol{s} \rangle_2, \tag{1}$$

1. To be rigorous, geodesics are paths with zero acceleration and not necessarily unique.

where $\lambda_{\boldsymbol{u}}^K = 2/(1+K\|\boldsymbol{u}^2\|)$ is the conformal factor, and $\langle\cdot,\cdot\rangle_2$ is the euclidean inner product.

Addition. For $\boldsymbol{x},\boldsymbol{y}\in\mathcal{M}_K^d$, their addition is

$$\boldsymbol{x}\oplus_K\boldsymbol{y}=\frac{(1-2K\langle\boldsymbol{x},\boldsymbol{y}\rangle_2-K\|\boldsymbol{y}\|^2)\boldsymbol{x}+(1+K\|\boldsymbol{x}\|^2)\boldsymbol{y}}{1-2K\langle\boldsymbol{x},\boldsymbol{y}\rangle_2+K^2\|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2}. \tag{2}$$

Distance Measure. For $\boldsymbol{x},\boldsymbol{y}\in\mathcal{M}_K^d$, their distance is

$$d(\boldsymbol{x},\boldsymbol{y})=\frac{2}{\sqrt{|K|}}\tan_K^{-1}(\sqrt{|K|}\cdot\|-\boldsymbol{x}\oplus_K\boldsymbol{y}\|). \tag{3}$$

Exponential Map. The exponential map $\exp_{\boldsymbol{u}}^K(\boldsymbol{q})$ is used to map a vector $\boldsymbol{q}$ from the tangent space $T_{\boldsymbol{u}}\mathcal{M}_K^d$ to the manifold $\mathcal{M}_K^d$,

$$\exp_{\boldsymbol{u}}^K(\boldsymbol{q})=\boldsymbol{u}\oplus_K\left(\tan_K(\sqrt{|K|}\frac{\lambda_{\boldsymbol{u}}^K\|\boldsymbol{q}\|}{2})\frac{\boldsymbol{q}}{\sqrt{|K|}\cdot\|\boldsymbol{q}\|}\right). \tag{4}$$

Logarithmic Map. The logarithmic map $\log_{\boldsymbol{u}}^K(\boldsymbol{x})$ is used to map a vector $\boldsymbol{x}$ from the the manifold $\mathcal{M}_K^d$ to the tangent space $T_{\boldsymbol{u}}\mathcal{M}_K^d$,

$$\log_{\boldsymbol{u}}^K(\boldsymbol{x})=\frac{2}{\sqrt{|K|}\lambda_{\boldsymbol{u}}^K}\tan_K^{-1}(\sqrt{|K|}\cdot\|-\boldsymbol{u}\oplus_K\boldsymbol{x}\|)\frac{-\boldsymbol{u}\oplus_K\boldsymbol{x}}{\|-\boldsymbol{u}\oplus_K\boldsymbol{x}\|}. \tag{5}$$

The function $\tan_K(\cdot)$ is determined by the sign of the curvature, that is

$$\tan_K(\cdot)=\begin{cases}\tan(\cdot), & \text{if } K\geq 0\\ \tanh(\cdot), & \text{if } K<0\end{cases}, \tag{6}$$

and its inverse function $\tan_K^{-1}(\cdot)$ is

$$\tan_K^{-1}(\cdot)=\begin{cases}\arctan(\cdot), & \text{if } K\geq 0\\ \operatorname{arctanh}(\cdot), & \text{if } K<0\end{cases}. \tag{7}$$

Orthogonal Projection. The orthogonal projection $\mathrm{proj}_{\boldsymbol{u}}^K$ transforms an ambient euclidean vector z into a tangent space $T_{\boldsymbol{u}}\mathcal{M}_K^d$,

$$\mathrm{proj}_{\boldsymbol{u}}^K(\boldsymbol{z})=\begin{cases}\boldsymbol{z}-\langle\boldsymbol{u},\boldsymbol{z}\rangle_2\boldsymbol{u}, & \text{if } K\geq 0\\ \left(1/(\lambda_{\boldsymbol{u}}^K)^2\right)\boldsymbol{z}, & \text{if } K<0\end{cases}. \tag{8}$$

3.3 Product Manifold

A product manifold is a Cartesian product of multiple submanifolds. Considering m submanifolds $\mathcal{M}_1,\ldots,\mathcal{M}_m$, a product manifold of the m submanifolds is defined by $\mathcal{M}_p := \times_{j=1}^m\mathcal{M}_j$, where $\times_{j=1}^m$ is the Cartesian product [57]. Any $\boldsymbol{x}\in\mathcal{M}_p$ is represented by vector concatenation, $\boldsymbol{x}=(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_m)$, where $\boldsymbol{x}_j\in\mathcal{M}_j$. Similarly, the tangent space $T_{\boldsymbol{u}}\mathcal{M}_p$ of $\mathcal{M}_p$ is defined by the Cartesian product of tangent spaces of submanifolds, $T_{\boldsymbol{u}}\mathcal{M}_p := \times_{j=1}^m T_{\boldsymbol{u}_j}\mathcal{M}_j$, with $\boldsymbol{u}=(\boldsymbol{u}_1,\ldots,\boldsymbol{u}_m)\in\mathcal{M}_p$ being the tangent point. A tangent vector $\boldsymbol{q}\in T_{\boldsymbol{u}}\mathcal{M}_p$ is $\boldsymbol{q}=(\boldsymbol{q}_1,\ldots,\boldsymbol{q}_m)$, where $\boldsymbol{q}_j\in T_{\boldsymbol{u}_j}\mathcal{M}_j$. Inner product and squared distance in product manifold are given by the sum of inner products and squared distances in submanifolds. For two vectors $\boldsymbol{x},\boldsymbol{y}\in\mathcal{M}_p$, their squared distance is $d_{\mathcal{M}_p}^2(\boldsymbol{x},\boldsymbol{y})=\sum_{j=1}^m d_{\mathcal{M}_j}^2(\boldsymbol{x}_j,\boldsymbol{y}_j)$, where $d_{\mathcal{M}_j}^2(\cdot)$ is the squared distance in $\mathcal{M}_j$. For two vectors $\boldsymbol{q},\boldsymbol{s}\in T_{\boldsymbol{u}}\mathcal{M}_p$, their inner product is $\langle\boldsymbol{q},\boldsymbol{s}\rangle_{\boldsymbol{u}}^p=\sum_{j=1}^m\langle\boldsymbol{q}_j,\boldsymbol{s}_j\rangle_{\boldsymbol{u}_j}$, where $\langle\cdot\rangle_{\boldsymbol{u}_j}$ is the inner product in $T_{\boldsymbol{u}_j}\mathcal{M}_j$.

4 Product Manifold Neural Network

This paper proposes a curvature-adaptive meta-learning method that is capable of adapting a base-learner to various forms of manifold data. As such, the base-learner should have the ability to encode manifold structure. Such capability is typically ignored by most neural networks as they are designed in euclidean space. To solve this issue, we introduce a product manifold neural network as the base-learner (detailed in this section), where data is represented by product manifold of constant curvature spaces. We adapt the neural network to manifold data by updating curvatures (detailed in the next section).

It is known that layers of a neural network encode different information. For example, in a vision task, bottom layers tend to capture low-level features such as edges, while upper layers usually capture high-level semantic features [58]. Inspired by this, each layer in our neural network represents data in an individual product manifold with layer-specific curvatures. In the lth layer, data is represented by the product manifold $\mathcal{M}_p^l$ of m constant curvature spaces. For ease of reading, we omit the index l in the following sections, unless a clear distinction is necessary. In one layer of our neural network, the product manifold is defined by

$$\mathcal{M}_p := \times_{j=1}^m\mathcal{M}_{K_j}^{d_j}, \tag{9}$$

where K_j is the curvature of the jth constant curvature space, and d_j is the dimension of the jth constant curvature space. The dimension of $\mathcal{M}_p$ is the sum of dimensions of submanifolds, $\sum_{j=1}^m d_j$. As $\mathcal{M}_p$ has non-euclidean structures, conventional network architectures (e.g., fully-connected layers and convolutional blocks) cannot be directly applied. Considering that tangent space is a euclidean space, here we utilize exponential and logarithmic maps to generalize basic architectures from euclidean space to the product manifold. The exponential and logarithmic maps on the product manifold are defined as follows.

Definition 1. *Given a vector $\boldsymbol{x}=(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_m)\in\mathcal{M}_p$, $\mathcal{M}_p$ is a product manifold of m constant curvature spaces, $\mathcal{M}_p := \times_{j=1}^m\mathcal{M}_{K_j}^{d_j}$, $\boldsymbol{x}_j$ is in the jth constant curvature space. The logarithmic map $\mathrm{Log}_{\boldsymbol{u}}(\boldsymbol{x}):\mathcal{M}_p\to T_{\boldsymbol{u}}\mathcal{M}_p$ is used to map $\boldsymbol{x}$ from $\mathcal{M}_p$ to the tangent space $T_{\boldsymbol{u}}\mathcal{M}_p$,*

$$\mathrm{Log}_{\boldsymbol{u}}(\boldsymbol{x})=\left(\log_{\boldsymbol{u}_1}^{K_1}(\boldsymbol{x}_1),\ldots,\log_{\boldsymbol{u}_m}^{K_m}(\boldsymbol{x}_m)\right), \tag{10}$$

where $\boldsymbol{u}=(\boldsymbol{u}_1,\ldots,\boldsymbol{u}_m)\in\mathcal{M}_p$ is the tangent point.

Definition 2. *Given a vector $\boldsymbol{q}=(\boldsymbol{q}_1,\boldsymbol{q}_2,\ldots,\boldsymbol{q}_m)\in T_{\boldsymbol{u}}\mathcal{M}_p$, $T_{\boldsymbol{u}}\mathcal{M}_p$ is the tangent space of a product manifold $\mathcal{M}_p$ of m constant curvature spaces, $\boldsymbol{q}_j$ is in the tangent space of the jth constant curvature space, and $\boldsymbol{u}=(\boldsymbol{u}_1,\ldots,\boldsymbol{u}_m)\in\mathcal{M}_p$ is the tangent point. The exponential map $\mathrm{Exp}_{\boldsymbol{u}}(\boldsymbol{q}):T_{\boldsymbol{u}}\mathcal{M}_p\to\mathcal{M}_p$ is used to map $\boldsymbol{q}$ from $T_{\boldsymbol{u}}\mathcal{M}_p$ to $\mathcal{M}_p$,*

$$\mathrm{Exp}_{\boldsymbol{u}}(\boldsymbol{q})=\left(\exp_{\boldsymbol{u}_1}^{K_1}(\boldsymbol{q}_1),\ldots,\exp_{\boldsymbol{u}_m}^{K_m}(\boldsymbol{q}_m)\right). \tag{11}$$

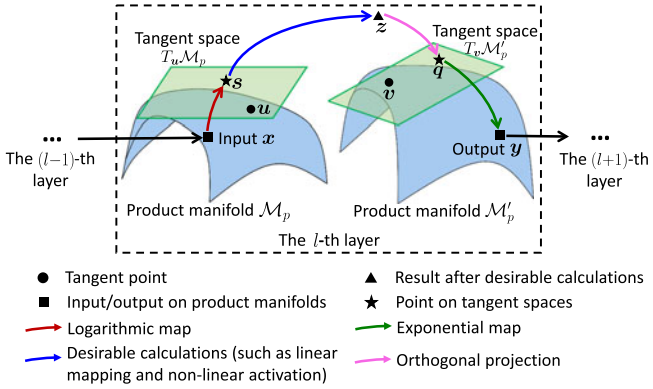


Fig. 2. We take the fully-connected layer as an example to show the data flow in the product manifold neural network. In the lth layer, the input is $\boldsymbol{x}$, and we first map $\boldsymbol{x}$ into the tangent space $T_{\boldsymbol{u}}\mathcal{M}_p$. We then carry out desired calculations, such as a linear mapping and non-linear activation as done in fully-connected layers. Since we cannot directly map the result $\boldsymbol{z}$ into the product manifold $\mathcal{M}'_p$ of the $(l+1)$th layer, we preserve results in a tangent space $T_{\boldsymbol{v}}\mathcal{M}'_p$ of $\mathcal{M}'_p$ and finally map the result to $\mathcal{M}'_p$ via the exponential map. By analogy, this is true for each layer in the product manifold neural network.

In one layer, we first map input features from the product manifold $\mathcal{M}_p$ to a tangent space $T_{\boldsymbol{u}}\mathcal{M}_p$ using the logarithmic map. Next, we perform desired calculations (e.g., linear mappings and nonlinear activation) on $T_{\boldsymbol{u}}\mathcal{M}_p$. Then, we need to convey the result to the next layer, whose data is in a product manifold $\mathcal{M}'_p$, having different dimensions and curvatures with $\mathcal{M}_p$. Since there does not exist a mapping between $T_{\boldsymbol{u}}\mathcal{M}_p$ and $\mathcal{M}'_p$, we resort to a new tangent space $T_{\boldsymbol{v}}\mathcal{M}'_p$ on $\mathcal{M}'_p$, where $\boldsymbol{v} \in \mathcal{M}'_p$ is the tangent point. We utilize the orthogonal projection operation to enforce the result on the tangent space $T_{\boldsymbol{v}}\mathcal{M}'_p$, and finally map the result to $\mathcal{M}'_p$ using the exponential map. The orthogonal projection on the product manifold is defined as follows.

Definition 3. *Given a vector $\boldsymbol{z} = (z_1, z_2, \ldots, z_m)$, the orthogonal projection operation $\mathrm{Proj}_{\boldsymbol{u}}$ on a product manifold $\mathcal{M}_p$ of m constant curvature spaces is used to enforce $\boldsymbol{z}$ on the tangent space $T_{\boldsymbol{u}}\mathcal{M}_p$,*

$$\mathrm{Proj}_{\boldsymbol{u}}(\boldsymbol{z}) = \left(\mathrm{proj}^{K_1}_{\boldsymbol{u}_1}(z_1), \ldots, \mathrm{proj}^{K_m}_{\boldsymbol{u}_m}(z_m)\right). \tag{12}$$

In summary, we use two tangent spaces $T_{\boldsymbol{u}}\mathcal{M}_p$ and $T_{\boldsymbol{v}}\mathcal{M}'_p$ in one layer of the product manifold neural network. $T_{\boldsymbol{u}}\mathcal{M}_p$ is on the product manifold $\mathcal{M}_p$, and $\boldsymbol{u} \in \mathcal{M}_p$ is the tangent point. $T_{\boldsymbol{u}}\mathcal{M}_p$ is used to perform desirable calculations (e.g., the linear mapping and nonlinear activation in fully-connected layers). $T_{\boldsymbol{v}}\mathcal{M}'_p$ is on the product manifold $\mathcal{M}'_p$, and $\boldsymbol{v} \in \mathcal{M}'_p$ is the tangent point. $T_{\boldsymbol{v}}\mathcal{M}'_p$ is used to convey the result to the next layer, where data is represented in $\mathcal{M}'_p$. Suppose that there are L layers in our backbone, we totally have $2L$ tangent spaces. The illustration of our network is shown in Fig. 2.

Since tangent spaces are only local approximation to the manifold, it will cause large approximation errors if tangent points are not flexible and far from features [22]. Thus, using a common tangent space for all features may result in non-discriminative features. In our method, we make tangent points $\boldsymbol{u}$ and $\boldsymbol{v}$ adaptive and will be learned during training. As a result, suitable tangent spaces are constructed for each layer, which in turn reduce the approximation error and enable the network to better fit to manifold data.

We denote a product manifold neural network by $\mathcal{P}_{\boldsymbol{\theta},\boldsymbol{\phi}}$, and suppose that there are L layers in the backbone to extract features and the $(L+1)$th layer is a classifier. $\boldsymbol{\theta}$ denotes parameters of the product manifold neural network, including all tangent points $\{\boldsymbol{u}^l\}_{l=1}^{L+1}$, $\{\boldsymbol{v}^l\}_{l=1}^{L}$, weights $\{\boldsymbol{W}^l\}_{l=1}^{L}$, biases $\{\boldsymbol{b}^l\}_{l=1}^{L}$, and the classifier weight $\{\boldsymbol{w}^i\}$. $\boldsymbol{\phi}$ is the set of all curvatures. Each layer has m constant curvature spaces, and thus we have $m(L+1)$ curvatures in the neural network, collectively denoted by $\boldsymbol{\phi} = \{K_1, \ldots, K_{mL+m}\}$. In the next part, we will discuss the forms of the backbone and classifier used in the product manifold.

4.1 Backbone of the Product Manifold Neural Network

A neural network is used to extract features for downstream tasks and is usually composed of fully-connected layers and convolutional blocks, as shown in Fig. 3

4.1.1 *Fully-Connected Layer*

Suppose the input of a fully-connected layer is $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m) \in \mathcal{M}_p$.

*Map $\boldsymbol{x}$ to a Tangent Space.*We map $\boldsymbol{x}$ to a tangent space $T_{\boldsymbol{u}}\mathcal{M}_p$ by

$$\boldsymbol{s} = \mathrm{Log}_{\boldsymbol{u}}(\boldsymbol{x}), \tag{13}$$

where $\boldsymbol{s} \in T_{\boldsymbol{u}}\mathcal{M}_p$, and $\boldsymbol{u} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m) \in \mathcal{M}_p$ is the tangent point for inputs.

Calculations in the Tangent Space. We then carry out the linear mapping and nonlinear activation operations in each constant curvature space, obtaining the result $\boldsymbol{z}$,

$$\begin{aligned}\boldsymbol{z} &= (\boldsymbol{z}_1, \ldots, \boldsymbol{z}_m)\\ &= (\sigma(\boldsymbol{W}_1\boldsymbol{s} + \boldsymbol{b}_1), \ldots, \sigma(\boldsymbol{W}_m\boldsymbol{s} + \boldsymbol{b}_m)).\end{aligned} \tag{14}$$

Here, $\boldsymbol{W}_j$ and $\boldsymbol{b}_j$ are the weight and bias for the jth constant curvature space, respectively, and σ is the nonlinear activation operation.

We need to convey the result $\boldsymbol{z}$ to the next layer, whose data is in a product manifold $\mathcal{M}'_p$. $\mathcal{M}'_p$ has different dimensions and curvatures from $\mathcal{M}_p$. Since $T_{\boldsymbol{u}}\mathcal{M}_p$ is not necessarily a tangent space for $\mathcal{M}'_p$, we cannot directly convey $\boldsymbol{z}$ to $\mathcal{M}'_p$ to the next layer. To solve this issue, we resort to a new tangent space $T_{\boldsymbol{v}}\mathcal{M}'_p$ on $\mathcal{M}'_p$, where $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_m) \in \mathcal{M}'_p$ is the tangent point for outputs. We utilize the orthogonal projection on product manifold, which enforces the result on the tangent space $T_{\boldsymbol{v}}\mathcal{M}'_p$,

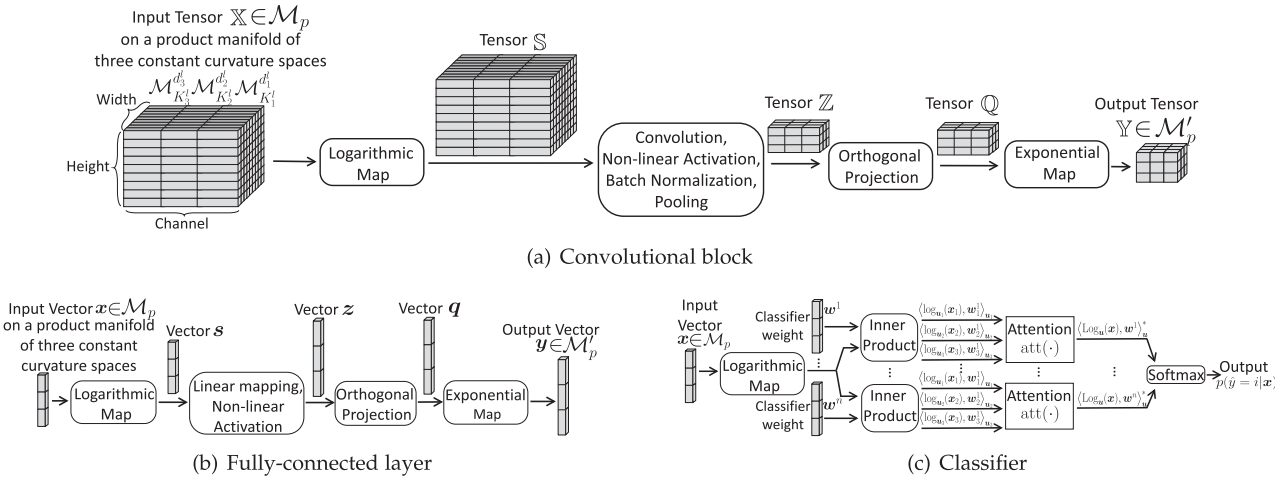
$$\boldsymbol{q} = \mathrm{Proj}_{\boldsymbol{v}}(\boldsymbol{z}), \tag{15}$$

where $\boldsymbol{q} \in T_{\boldsymbol{v}}\mathcal{M}'_p$.

Map $\boldsymbol{q}$ to a Product Manifold. Finally, we map $\boldsymbol{q}$ to the product manifold $\mathcal{M}'_p$ and obtain the output $\boldsymbol{y}$,

$$\boldsymbol{y} = \mathrm{Exp}_{\boldsymbol{v}}(\boldsymbol{q}). \tag{16}$$

$\boldsymbol{y} \in \mathcal{M}'_p$ is also the input for the next layer. The aforementioned design not only preserves the Riemannian structure of data but also provides a simple way to extract discriminative



(a) Convolutional block

(b) Fully-connected layer

(c) Classifier

Fig. 3. Illustration of a convolutional block, a fully-connected layer, and a classifier in the product manifold neural network. In this illustration, we represent data in a product manifold of three constant curvature spaces, and features are denoted by concatenating representations on the three constant curvature spaces.

features (see Algorithm 1 for a pseudo-code of a fully-connected layer on the product manifold).

Algorithm 1. Pseudo-Code of Computations Performed in a Fully-Connected Layer of the Product Manifold Neural Network

Input: Input feature $\boldsymbol{x}\in\mathcal{M}_p$, weight $W=[\boldsymbol{W}_1,\ldots,\boldsymbol{W}_m]$, and bias $\boldsymbol{b}=[\boldsymbol{b}_1,\ldots,\boldsymbol{b}_m]$.
Output: Output feature $\boldsymbol{y}\in\mathcal{M}'_p$ for the next layer.
1: Map feature $\boldsymbol{x}$ to the tangent space $T_{\boldsymbol{u}}\mathcal{M}_p$ via Eq. (13).
2: Carry out linear mapping and nonlinear activation operations in the tangent space via Eq. (14), and preserve the result $\boldsymbol{q}$ in the tangent space $T_{\boldsymbol{v}}\mathcal{M}'_p$ via Eq. (15).
3: Map $\boldsymbol{q}$ to the product manifold $\mathcal{M}'_p$ for the next layer via Eq. (16).

4.1.2 Convolutional Block

We take a convolutional block that sequentially contains a convolution operation, a nonlinear activation operation, a batch normalization operation, and a pooling operation as an example to show how to build convolutional blocks in the product manifold. The input is a 3-D tensor $\mathbb{X}\in\mathbb{R}^{c\times h\times w}$ where c is the number of channels, and h and w are the height and width of feature maps, respectively.

Map Features in $\mathbb{X}$ into a Tangent Space. We map features in $\mathbb{X}$ to a tensor $\mathbb{S}\in\mathbb{R}^{c\times h\times w}$ in the tangent space $T_{\boldsymbol{u}}\mathcal{M}_p$, which has the same size with $\mathbb{X}$. We reshape the 3-D tensor $\mathbb{X}$ into a matrix $\boldsymbol{X}\in\mathbb{R}^{c\times hw}$. Here, we regard each column $\boldsymbol{x}^i\in\mathbb{R}^c$ in $\boldsymbol{X}$ as a feature on the product manifold $\boldsymbol{x}^i\in\mathcal{M}_p$. Thus, there are hw features in total, and the dimension of each one is c. We utilize the logarithmic map $\mathrm{Log}_{\boldsymbol{u}}(\cdot)$ to map each feature $\boldsymbol{x}^i$ into the tangent space $T_{\boldsymbol{u}}\mathcal{M}_p$, $\boldsymbol{s}^i=\mathrm{Log}_{\boldsymbol{u}}(\boldsymbol{x}^i)$, where $\boldsymbol{s}^i\in T_{\boldsymbol{u}}\mathcal{M}_p$, and $\boldsymbol{u}=(\boldsymbol{u}_1,\ldots,\boldsymbol{u}_m)\in\mathcal{M}_p$ is the tangent point for inputs. We collect all $\boldsymbol{s}^i$ into a matrix $\boldsymbol{S}\in\mathbb{R}^{c\times hw}$ ($\boldsymbol{s}^i$ is the ith column of $\boldsymbol{S}$). $\boldsymbol{S}$ is finally reshaped back into a 3-D tensor $\mathbb{S}\in\mathbb{R}^{c\times h\times w}$. In summary, the process to obtain the tensor $\mathbb{S}$ is denoted as

$$\mathbb{S}=\mathrm{Log}_{\boldsymbol{u}}(\mathbb{X}). \tag{17}$$

Calculations in the Tangent Space. For the tensor $\mathbb{S}$ in the tangent space, we sequentially carry out a convolution operation, a nonlinear activation operation, a batch normalization operation, and a pooling operation, given by

$$\begin{aligned}\mathbb{Z}&=\Big(\mathbb{Z}_1,\ldots,\mathbb{Z}_m\Big)\\&=\Big(\mathrm{Pool}(\mathrm{BN}(\sigma(\boldsymbol{W}_1\otimes\mathbb{S}+\boldsymbol{b}_m))),\ldots\mathrm{Pool}(\mathrm{BN}(\sigma(\boldsymbol{W}_m\otimes\mathbb{S}+\boldsymbol{b}_m)))\Big),\end{aligned} \tag{18}$$

where $\otimes$, Pool, and BN are convolution, pooling, and batch normalization operations, respectively. $\boldsymbol{W}_j$ and $\boldsymbol{b}_j$ are the weight and bias in the convolution operation for the jth constant curvature space, respectively. $\mathbb{Z}$ is concatenated by m 3-D tensors $(\mathbb{Z}_1,\ldots,\mathbb{Z}_m)$ in the m constant curvature space along the channel dimension.

Similar to fully-connected layers in the product manifold, we also utilize the orthogonal projection operation to enforce features in $\mathbb{Z}$ in a tangent space $T_{\boldsymbol{v}}\mathcal{M}'_p$ for the next layer,

$$\mathbb{Q}=\mathrm{Proj}_{\boldsymbol{v}}(\mathbb{Z}), \tag{19}$$

where $\boldsymbol{v}=(\boldsymbol{v}_1,\ldots,\boldsymbol{v}_m)\in\mathcal{M}'_p$ is the tangent point for outputs. We collect weights and biases for the m constant curvature spaces as $\boldsymbol{W}=[\boldsymbol{W}_1,\ldots,\boldsymbol{W}_m]$ and $\boldsymbol{b}=[\boldsymbol{b}_1,\ldots,\boldsymbol{b}_m]$.

Map Features in $\mathbb{Q}$ into a Product Manifold. Finally, we use reshaping and exponential operations to map $\mathbb{Q}$ into the manifold $\mathcal{M}'_p$ and obtain the output $\mathbb{Y}$,

$$\mathbb{Y}=\mathrm{Exp}_{\boldsymbol{v}}(\mathbb{Q}). \tag{20}$$

$\mathbb{Y}$ is also the input for the next layer. The proposed convolutional block is summarised in Algorithm 2.

4.2 Classifier of the Product Manifold Neural Network

We argue that constant curvature spaces capture significantly different features and play different roles in downstream tasks. Inspired by the fact that attention mechanism has achieved success in many machine learning problems, we propose an attention classifier in the product manifold, as shown in Fig. 3. Here, we define an attention inner product between two vectors $\boldsymbol{q}=(\boldsymbol{q}_1,\ldots,\boldsymbol{q}_m)\in T_{\boldsymbol{u}}\mathcal{M}_p$ and $\boldsymbol{s}=$

$(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_m) \in T_{\boldsymbol{u}}\mathcal{M}_p$. We assign different weights to different constant curvature spaces, and weight-sum their inner products. Concretely, the attention inner product $\langle \cdot, \cdot \rangle^*_{\boldsymbol{u}}$ is computed by

$$\begin{cases} \boldsymbol{a} = \mathrm{att}\Big(\Big[\langle \boldsymbol{q}_1, \boldsymbol{s}_1 \rangle_{\boldsymbol{u}_1}, \ldots, \langle \boldsymbol{q}_m, \boldsymbol{s}_m \rangle_{\boldsymbol{u}_m}\Big]\Big) \\ \langle \boldsymbol{q}, \boldsymbol{s} \rangle^*_{\boldsymbol{u}} = \sum_{j=1}^m a_j \langle \boldsymbol{q}_j, \boldsymbol{s}_j \rangle_{\boldsymbol{u}_j} = \sum_{j=1}^m a_j (\lambda_{\boldsymbol{u}_j}^{K_j})^2 \langle \boldsymbol{q}_j, \boldsymbol{s}_j \rangle_2 \end{cases}, \tag{21}$$

where $\boldsymbol{u} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m)$ is the tangent point. $\boldsymbol{a} = [a_1, \ldots, a_m]$ contains weights for all constant curvature spaces, and a_j is the jth element of $\boldsymbol{a}$. We use an attention network $\mathrm{att}(\cdot)$ to assign different weights.

Algorithm 2. Pseudo-Code of Computations Performed in a Convolutional Block of the Product Manifold Neural Network

Input: Input tensor $\mathbb{X}$, where features $\boldsymbol{x}^i$ in $\mathbb{X}$ are in a product manifold, $\boldsymbol{x}^i \in \mathcal{M}_p$. Weights $\boldsymbol{W} = [\boldsymbol{W}_1, \ldots, \boldsymbol{W}_m]$ and bias $\boldsymbol{b} = [\boldsymbol{b}_1, \ldots, \boldsymbol{b}_m]$.

Output: Output tensor $\mathbb{Y}$ for the next layer, where features in $\mathbb{Y}$ are in a product manifold $\mathcal{M}'_p$.

1: Map features $\boldsymbol{x}^i$ in $\mathbb{X}$ to a tangent space $T_{\boldsymbol{u}}\mathcal{M}_p$ via Eq. (17).
2: Carry out operations in the tangent space via Eq. (18) and enforce the result $\mathbb{Q}$ in the tangent space $T_v\mathcal{M}'_p$ via Eq. (19).
3: Map $\mathbb{Q}$ into the product manifold $\mathcal{M}'_p$ of the next layer via Eq. (20).

In the product manifold neural network, there are several convolutional blocks or fully-connected layers to extract features, and the last layer is used for classification. Suppose the input of the classifier is $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m) \in \mathcal{M}_p$, the tangent point of the classifier as $\boldsymbol{u} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m)$, and the weight for the ith class in the classifier as $\boldsymbol{w}^i$. The probability that a feature $\boldsymbol{x}$ belongs to the ith class is computed by the softmax function,

$$p(\hat{y} = i | \boldsymbol{x}) = \frac{\exp\Big(\big\langle \mathrm{Log}_{\boldsymbol{u}}(\boldsymbol{x}), \boldsymbol{w}^i \big\rangle^*_{\boldsymbol{u}}\Big)}{\sum_n \exp\Big(\big\langle \mathrm{Log}_{\boldsymbol{u}}(\boldsymbol{x}), \boldsymbol{w}^n \big\rangle^*_{\boldsymbol{u}}\Big)}, \tag{22}$$

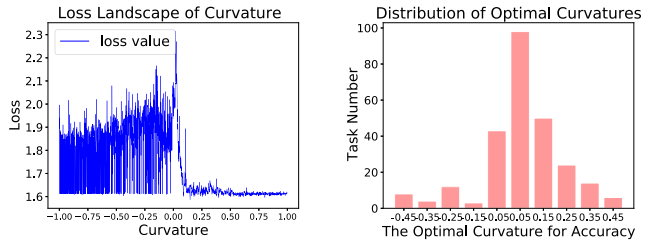
where $\hat{y}$ is the prediction.

5 Curvature Adaptation

In this section, we first show the role of curvatures in the product manifold neural network, and then we introduce a curvature generation scheme and a curvature updating scheme. Through these two schemes, the product manifold neural network is capable of quickly adapting to various forms of manifold structures by producing suitable curvatures given very limited data.

5.1 Curvatures in Neural Networks

To understand the role of curvature, we consider a few-shot classification task and plot the loss landscape with respect to the curvature in Fig. 4. This analysis reveals two key points. First and from Fig. 4a, we empirically observe the chaotic nature of the loss landscape. Fig. 4b suggests that the optimal curvature for different tasks vary, where we plot the histogram of the curvature across 150 tasks. This is of course not very unexpected as natural data usually complies with complex and various manifold structures. With cautious, we can claim that this is a confirmation of our hypothesis, traditional meta-learning methods that only learn a common curvature initialization and adopt a simple gradient descent scheme cannot accurately model the structure of the data and benefit from it for adaptation. A common curvature initialization may cause a long optimization trajectory for some tasks, and a simple gradient descent scheme may result in local optima in the trajectory due to the sharp landscape. To overcome these challenges, in our curvature-adaptive meta-learning method, we learn to assign task-specific curvature initialization, and carry out adaptive curvature updating based on the underlying data distributions, bringing a faster convergence and better optima.



(a) Loss landscape (b) Optimal curvature distribution

Fig. 4. Motivation of the curvature generation and curvature updating schemes. Experiments are conducted on the few-shot classification dataset mini-ImageNet [59]. In Fig. 4a, we plot the loss curve of a task with respect to a curvature. In Fig. 4b, we fix all the parameters of the product manifold neural network, measure the optimal curvatures across 150 tasks, and plot their distribution. The loss landscape is chaotic, and optimal curvatures for different tasks vary. Thus, quickly finding of the optimal curvature for a novel task is challenging and necessary.

5.2 Formulation

In the meta-learning setting, a collection of tasks $\{\mathcal{T}_i\}$ are available, sampled from a task distribution $\mathcal{T}_i \sim \mathcal{D}(\mathcal{T})$, where each task $\mathcal{T}_i$ has a support set $\mathcal{D}_i^s$ and a query set $\mathcal{D}_i^q$ containing the training and test data, respectively.

We propose a curvature generation scheme $\mathcal{G}_{\boldsymbol{\psi}_1}$ and a curvature updating scheme $\mathcal{A}_{\boldsymbol{\psi}_2}$ to quickly adapt curvatures to novel tasks. The curvature generation scheme $\mathcal{G}_{\boldsymbol{\psi}_1}$ assigns task-specific curvature initialization to a novel task. To be specific, we learn a common curvature initialization $\boldsymbol{\phi}$ and parameter $\boldsymbol{\psi}_1$ to modulate $\boldsymbol{\phi}$ for a novel task $\mathcal{T}_i$. The task-specific curvature initialization $\boldsymbol{\phi}_{i,0}$ is produced as $\boldsymbol{\phi}_{i,0} = \mathcal{G}_{\boldsymbol{\psi}_1}(\boldsymbol{\phi}, \mathcal{D}_i^s)$. The curvature updating scheme $\mathcal{A}_{\boldsymbol{\psi}_2}$ (parameterized by $\boldsymbol{\psi}_2$) produces adaptive learning rates and search directions for the initial curvature. That is, the curvature is updated by $\boldsymbol{\phi}'_i = \mathcal{A}_{\boldsymbol{\psi}_2}(\boldsymbol{\phi}_{i,0}, \mathcal{D}_i^s)$ for task $\mathcal{T}_i$. Similar to many optimization-based meta-learning methods [1], [5], we also learn parameter initialization $\boldsymbol{\theta}$ of the product manifold neural network and update it by $\boldsymbol{\theta}'_i \leftarrow \boldsymbol{\theta} - \beta \nabla_{\boldsymbol{\theta}} \mathcal{L}$ for task $\mathcal{T}_i$, where β is the learning rate and $\nabla_{\boldsymbol{\theta}}\mathcal{L}$ is the gradient. The adaptation process to a novel task is shown in Fig. 5.

Our goal is to learn the meta-learner, i.e., the curvature generation scheme $\mathcal{G}_{\boldsymbol{\psi}_1}$, the curvature updating scheme $\mathcal{A}_{\boldsymbol{\psi}_2}$, a common curvature initialization $\boldsymbol{\phi}$, and a parameter initialization $\boldsymbol{\theta}$, such that a product manifold neural network can quickly adapt to manifold data through few optimization

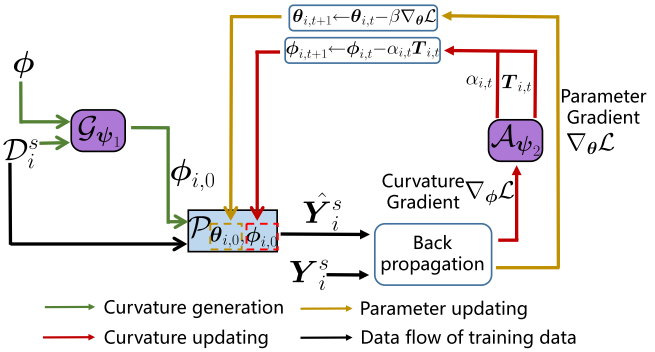


Fig. 5. Illustration of the adaptation to a novel task. The green line represents the curvature generating scheme, the red line denotes the curvature updating scheme, the yellow line denotes the process of parameter updating, and the black line denotes the flow of training data. $\hat{Y}_i^s$ is the prediction, and Y_i^s is the ground truth. $\mathcal{L}$ is the loss function on the support data $\mathcal{D}_i^s$. $\alpha_{i,t}$ and $\boldsymbol{T}_{i,t}$ are the produced learning rate and search direction for each curvature, and β is the learning rate for updating the parameters.

steps. Concretely, the optimal values of the four components are obtained by

$$\boldsymbol{\theta}^*, \boldsymbol{\phi}^*, \boldsymbol{\psi}_1^*, \boldsymbol{\psi}_2^* = \arg\min_{\boldsymbol{\theta},\boldsymbol{\phi},\boldsymbol{\psi}_1,\boldsymbol{\psi}_2} \mathcal{J}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}_1, \boldsymbol{\psi}_2). \tag{23}$$

$\mathcal{J}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}_1, \boldsymbol{\psi}_2)$ is our meta-objective function, formulated by

$$\begin{aligned}\mathcal{J}(\boldsymbol{\theta}, \boldsymbol{\phi}, \boldsymbol{\psi}_1, \boldsymbol{\psi}_2) &= \mathbb{E}_{\mathcal{T}_i \sim \mathcal{D}(\mathcal{T})}\left[\mathcal{L}\left(\boldsymbol{\theta}_i', \boldsymbol{\phi}_i', \mathcal{D}_i^q\right)\right] \\ &= \mathbb{E}_{\mathcal{T}_i \sim \mathcal{D}(\mathcal{T})}\left[\mathcal{L}\left(\boldsymbol{\theta} - \beta\nabla_{\boldsymbol{\theta}}\mathcal{L}, \mathcal{A}_{\boldsymbol{\psi}_2}\left(\mathcal{G}_{\boldsymbol{\psi}_1}(\boldsymbol{\phi}, \mathcal{D}_i^s), \mathcal{D}_i^s\right), \mathcal{D}_i^q\right)\right].\end{aligned} \tag{24}$$

$\mathcal{L}$ is a loss function to evaluate the model (e.g., a cross entropy loss). In Eq. (24), we compute the loss of the adapted product manifold neural network $\mathcal{P}_{\boldsymbol{\theta}_i', \boldsymbol{\phi}_i'}$ on the query data $\mathcal{D}_i^q$ of task $\mathcal{T}_i$. In next sections, we will detail the curvature generation scheme $\mathcal{G}_{\boldsymbol{\psi}_1}$ and the curvature updating scheme $\mathcal{A}_{\boldsymbol{\psi}_2}$.

5.3 Curvature Generation Scheme

Given a novel task $\mathcal{T}_i$, the curvature generation scheme $\mathcal{G}_{\boldsymbol{\psi}_1}$ modulates the common curvature initialization based on the second-order information of data $\mathcal{D}_i^s$, because the second-order information has an ability to capture expressive correlations between features [60], [61]. Specifically, we extract features of $\mathcal{D}_i^s$ from a product manifold neural network $\mathcal{P}_{\boldsymbol{\theta},\boldsymbol{\phi}}$ with the common curvature initialization $\boldsymbol{\theta}$ and parameter initialization $\boldsymbol{\phi}$. We represent these extracted features as $\{\boldsymbol{x}_{i,n} \in \mathcal{M}_p\}$, where n is the index of features and the dimension of $\boldsymbol{x}_{i,n}$ is d. The second-order information of $\{\boldsymbol{x}_{i,n}\}$ is denoted as $\boldsymbol{f}_i \in \mathbb{R}^{\zeta}$ (ζ is a hyperparameter), and it is computed via the factorized bilinear model [62], [63]

$$\begin{aligned} f_{i,j} &= \sum_n \text{SumPooling}\left(\boldsymbol{U}_j^\top \text{Log}_{\boldsymbol{u}}(\boldsymbol{x}_{i,n}) \circ \boldsymbol{V}_j^\top \text{Log}_{\boldsymbol{u}}(\boldsymbol{x}_{i,n})\right), \\ \boldsymbol{f}_i &= [f_{i,1}, f_{i,2}, \ldots, f_{i,\zeta}], \end{aligned} \tag{25}$$

where $f_{i,j}$ is the jth element in $\boldsymbol{f}_i$, SumPooling is the sum pooling operation, $\boldsymbol{U}_j \in \mathbb{R}^{d\times r}$, $\boldsymbol{V}_j \in \mathbb{R}^{d\times r}$, r is the other hyperparameter of the curvature generation scheme denoting the rank of $\boldsymbol{U}_j$ and $\boldsymbol{V}_j$, and $\circ$ denotes the Hadamard product. The parameters are collectively denoted by $\boldsymbol{U} = [\boldsymbol{U}_1, \ldots, \boldsymbol{U}_\zeta] \in \mathbb{R}^{d\times r\zeta}$ and $\boldsymbol{V} = [\boldsymbol{V}_1, \ldots, \boldsymbol{V}_\zeta] \in \mathbb{R}^{d\times r\zeta}$.

We use a multi-layer perceptron network MLP to generate curvature adjustment $\widehat{\boldsymbol{\phi}}_i$ based on $\boldsymbol{f}_i$. We empirically observed that arbitrary curvatures may cause numerical instabilities in the training process. Also, some studies suggest bounding the curvature (e.g., in the range of $[-1, 1]$) helps training and adaptation [12], [16]. Thus, we employ a sigmoid function on the multi-layer perceptron network MLP, and formulate the curvature adjustment $\widehat{\boldsymbol{\phi}}_i$ as

$$\widehat{\boldsymbol{\phi}}_i = (\text{Sigmoid}(\text{MLP}(\boldsymbol{f}_i)) - 0.5) \times 2, \tag{26}$$

where $\widehat{\boldsymbol{\phi}}_i \in \mathbb{R}^{mL+m}$ (recall that m is the number of constant curvature spaces in each layer, and there are L layers to extract features and one classifier). The parameter of MLP is $\boldsymbol{W}_f$. Based on the common curvature initialization $\boldsymbol{\phi}$ and the adjustment $\widehat{\boldsymbol{\phi}}_i$, the task-specific curvature initialization is computed by

$$\boldsymbol{\phi}_{i,0} = \boldsymbol{\phi} + \widehat{\boldsymbol{\phi}}_i. \tag{27}$$

Parameters of the curvature generation scheme are $\boldsymbol{\psi}_1 = \{\boldsymbol{U}, \boldsymbol{V}, \boldsymbol{W}_f\}$.

5.4 Curvature Updating Scheme

Our curvature updating scheme $\mathcal{A}_{\boldsymbol{\psi}_2}$ adapts the initial curvature $\boldsymbol{\phi}_{i,0}$ to task $\mathcal{T}_i$ via gradient descent. Noting that different constant curvature spaces may have different optimization trajectories, a single learning rate for adaptation could result in suboptimality. In addition, simply using gradients as search directions may lead to undesirable bias when training data is limited. To address this issue, we apply adaptive learning rates and search directions to each constant curvature space. Concretely, given a novel task $\mathcal{T}_i$, curvatures of a product manifold neural network are updated according to

$$K_{j,i,t+1} = K_{j,i,t} - \alpha_{j,i,t}\boldsymbol{T}_{j,i,t}, \tag{28}$$

where $K_{j,i,t+1}$ is the curvature of the jth constant curvature space after the tth optimization step for task $\mathcal{T}_i$. Recall that there are $L+1$ layers in the product manifold neural network, and each layer has m constant curvature spaces. Thus, we totally have $mL+m$ curvatures in the product manifold neural network, which we collectively denote by

$$\boldsymbol{\phi}_{i,t} = \{K_{1,i,t}, \ldots, K_{mL+m,i,t}\}. \tag{29}$$

In Eq. (28), $\alpha_{j,i,t}$ and $\boldsymbol{T}_{j,i,t}$ denote the learning rate and the search direction for the curvature $K_{j,i,t}$ at step t, respectively. Instead of designing the updating scheme by hand, we utilize neural networks $g_{\boldsymbol{\psi}_2}$ to automatically generate adaptive learning rates and search directions,

$$(\alpha_{j,i,t}, \boldsymbol{T}_{j,i,t}) = g_{\boldsymbol{\psi}_2}\left(K_{j,i,t}, \nabla_{K_{j,i,t}}\mathcal{L}\right), \tag{30}$$

where $\nabla_{K_{j,i,t}}\mathcal{L}$ is gradient with respect to the curvature $K_{j,i,t}$, $\boldsymbol{\psi}_2$ is the parameter of the used neural network $g_{\boldsymbol{\psi}_2}$.

The parameter $\boldsymbol{\theta}$ of the product manifold neural network is also updated via gradient descent. The update for task $\mathcal{T}_i$ is written as $\boldsymbol{\theta}_{i,t+1} = \boldsymbol{\theta}_{i,t} - \beta\nabla_{\boldsymbol{\theta}_{i,t}}\mathcal{L}$, where β is the learning rate for parameters and $\nabla_{\boldsymbol{\theta}_{i,t}}\mathcal{L}$ is the gradient with respect to the parameter $\boldsymbol{\theta}_{i,t}$. Suppose there are τ steps in the updating

scheme, the final adapted parameter and curvature for task $\mathcal{T}_i$ are denoted by $\boldsymbol{\theta}_i' = \boldsymbol{\theta}_{i,\tau}$ and $\boldsymbol{\phi}_i' = \boldsymbol{\phi}_{i,\tau}$.

After adapting the product manifold neural network to novel tasks, we update $\boldsymbol{\theta}$, $\boldsymbol{\phi}$, $\boldsymbol{\psi}_1$, and $\boldsymbol{\psi}_2$ by minimizing the loss of $\mathcal{P}_{\boldsymbol{\theta}_i', \boldsymbol{\phi}_i'}$ on the query set $\mathcal{D}_i^q$,

$$\begin{cases} \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \sum_{\mathcal{T}_i} \mathcal{L}\Big(\boldsymbol{\theta} - \beta \nabla_{\boldsymbol{\theta}} \mathcal{L}, \mathcal{A}_{\boldsymbol{\psi}_2}\big(\mathcal{G}_{\boldsymbol{\psi}_1}(\boldsymbol{\phi}, \mathcal{D}_i^s), \mathcal{D}_i^s\big), \mathcal{D}_i^q\Big) \\ \boldsymbol{\phi} \leftarrow \boldsymbol{\phi} - \eta \nabla_{\boldsymbol{\phi}} \sum_{\mathcal{T}_i} \mathcal{L}\Big(\boldsymbol{\theta} - \beta \nabla_{\boldsymbol{\theta}} \mathcal{L}, \mathcal{A}_{\boldsymbol{\psi}_2}\big(\mathcal{G}_{\boldsymbol{\psi}_1}(\boldsymbol{\phi}, \mathcal{D}_i^s), \mathcal{D}_i^s\big), \mathcal{D}_i^q\Big) \\ \boldsymbol{\psi}_1 \leftarrow \boldsymbol{\psi}_1 - \eta \nabla_{\boldsymbol{\psi}_1} \sum_{\mathcal{T}_i} \mathcal{L}\Big(\boldsymbol{\theta} - \beta \nabla_{\boldsymbol{\theta}} \mathcal{L}, \mathcal{A}_{\boldsymbol{\psi}_2}\big(\mathcal{G}_{\boldsymbol{\psi}_1}(\boldsymbol{\phi}, \mathcal{D}_i^s), \mathcal{D}_i^s\big), \mathcal{D}_i^q\Big) \\ \boldsymbol{\psi}_2 \leftarrow \boldsymbol{\psi}_2 - \eta \nabla_{\boldsymbol{\psi}_2} \sum_{\mathcal{T}_i} \mathcal{L}\Big(\boldsymbol{\theta} - \beta \nabla_{\boldsymbol{\theta}} \mathcal{L}, \mathcal{A}_{\boldsymbol{\psi}_2}\big(\mathcal{G}_{\boldsymbol{\psi}_1}(\boldsymbol{\phi}, \mathcal{D}_i^s), \mathcal{D}_i^s\big), \mathcal{D}_i^q\Big) \end{cases}. \tag{31}$$

η is the learning rate of meta-learning, and the meta-training process of our method is shown in Algorithm 3.

Algorithm 3. Training Procedure for Learning the Meta-Learner of the Proposed Method

Input: Task distribution $p(\mathcal{T})$.
Output: Updated $\boldsymbol{\theta}$, $\boldsymbol{\phi}$, $\boldsymbol{\psi}_1$, and $\boldsymbol{\psi}_2$.

while Not converged **do**
  Sample tasks $\{\mathcal{T}_i\} \sim p(\mathcal{T})$.
  for each task $\mathcal{T}_i \in \{\mathcal{T}_i\}$ **do**
    Construct support and query data $\mathcal{D}_i^s$ and $\mathcal{D}_i^q$ of $\mathcal{T}_i$.
    Extract features of $\mathcal{D}_i^s$ using $\mathcal{P}_{\boldsymbol{\theta},\boldsymbol{\phi}}$, and compute second-order information of the features via Eq. (25).
    Generate curvature initialization $\boldsymbol{\phi}_{i,0}$ for task $\mathcal{T}_i$ via Eqs. (26) and (27).
    Let $t = 0$ and $\boldsymbol{\theta}_{i,0} = \boldsymbol{\theta}$.
    while $t \leq \tau$ **do**
      Compute loss $\mathcal{L}(\boldsymbol{\theta}_{i,t}, \boldsymbol{\phi}_{i,t}, \mathcal{D}_i^s)$ on the support set $\mathcal{D}_i^s$, and compute gradients $\nabla_{K_{j,i,t}} \mathcal{L}$ and $\nabla_{\boldsymbol{\theta}_{i,t}} \mathcal{L}$.
      Produce learning rates $\alpha_{j,i,t}$ and search direction $\boldsymbol{T}_{j,i,t}$ for each curvature via Eq. (30), and perform gradient descent for each curvature via Eq. (28).
      Perform gradient descent for parameters $\boldsymbol{\theta}_{i,t+1} = \boldsymbol{\theta}_{i,t} - \beta \nabla_{\boldsymbol{\theta}_{i,t}} \mathcal{L}$.
    end while
    Let $\boldsymbol{\theta}_i' = \boldsymbol{\theta}_{i,\tau}$ and $\boldsymbol{\phi}_i' = \boldsymbol{\phi}_{i,\tau}$.
    Compute meta-objective $\mathcal{L}(\boldsymbol{\theta}_i', \boldsymbol{\phi}_i', \mathcal{D}_i^q)$ on the query set $\mathcal{D}_i^q$.
  end for
  Update $\boldsymbol{\theta}$, $\boldsymbol{\phi}$, $\boldsymbol{\psi}_1$, and $\boldsymbol{\psi}_2$ by minimizing $\sum_{\mathcal{T}_i} \mathcal{L}(\boldsymbol{\theta}_i', \boldsymbol{\phi}_i', \mathcal{D}_i^q)$ according to Eq. (31).
end while

5.5 Complexity Analysis

Compared with euclidean meta-learning, our extra computational overhead lies mainly in two aspects. (1) The forward and backward propagation caused by extra operations in the product manifold neural network, including the orthogonal projection operation, the exponential map, and the logarithmic map. (2) The curvature generation and curvature updating schemes to update our curvatures.

For a product manifold with m submanifolds of dimension d, complexity of the forward propagation of the orthogonal projection, the exponential map, and the logarithmic map is $O(4dm)$, $O(22dm)$, and $O(19dm)$, respectively. Complexity of the backward propagation of the orthogonal projection, the exponential map, and the logarithmic map is $O(20dm)$, $O(110dm)$, and $O(95dm)$, respectively. For the curvature generation scheme, we use a factorized bilinear model [62], [63] to produce task-specific curvature initialization. The complexity of the curvature generation scheme is $O(dmr\zeta + 2\zeta^2 + 2r\zeta + (\zeta+3)mL + 6\zeta)$, where r and ζ are hyperparameters of the curvature generation scheme, and L is the number of layers in the product manifold neural network. For the curvature updating scheme, we use three fully-connected and two ReLU layers to compute learning rates and use a two-layer LSTM to compute search directions for curvatures. The complexity of this module is $O(10(L+1)^2m^2 + 4Lm + 32TLm)$.

6 Experiments

We evaluated our method on few-shot classification, few-shot regression, and reinforcement learning tasks.

6.1 Few-Shot Classification

6.1.1 Setting

We used the mini-ImageNet [59], tiered-ImageNet [64], CUB [65], and CIFAR-FS [66] datasets for few-shot classification. The mini-ImageNet dataset has 100 classes and each category has 600 images. Following [1], we use 64, 16, and 20 classes for training, validation, and testing, respectively. The tiered-ImageNet dataset has 608 classes and 779165 images totally, where 351, 97, and 160 classes are used for training, validation, and testing, respectively. Images of the mini-ImageNet and tiered-ImageNet datasets are resized into 84×84 pixels. The CUB dataset is a fine-grained image dataset that contains 200 bird classes with 11788 images in total. Following the protocol of [25], we utilize 100, 50, and 50 classes for training, validation, and testing, respectively. We resize bird regions into 84×84. CIFAR-FS is a dataset derived from CIFAR-100 [67]. It contains 100 classes, where 64, 16, and 20 classes are used for training, validation, and testing, respectively. Each class has 600 images with size of 32×32. On the four datasets, we conducted experiments on 1-shot 5-way and 5-shot 5-way tasks.

We evaluated our method with a 4-layer convolutional network (ConvNet) [24] and a 12-layer residual network (ResNet12) [68] that has four convolutional blocks. Both backbones are extended from euclidean space to the product manifold. In doing so, we keep the architectural design and dimensions in each layer same as the original designs in euclidean space. For example, the convolutional block of the used ResNet12 has the following structure, Conv $\rightarrow$ BN $\rightarrow$ Conv $\rightarrow$ BN $\rightarrow$ Conv $\rightarrow$ BN $\rightarrow$ ReLU $\rightarrow$ Pooling, the same as that in euclidean space.

Our method was trained with $\tau = 5$ gradient steps, and evaluated with $\tau = 10$ gradient steps. The multi-layer perceptron network for curvature generation has two layers. The curvature updating network uses three fully-connected layers to produce learning rates and a two-layer LSTM to produce search directions. We pre-trained the backbone using the cross-entropy loss on training data. We carried out meta-training to learn our method over 20000 tasks. The validation set was only used to select a model after the meta-training stage, and the performance was reported as

TABLE 1
Accuracy (%) Comparisons With the State-of-the-Art Few-Shot Classification Methods on the Mini-ImageNet Dataset

Backbone	Method	Category	1-shot 5-way	5-shot 5-way
ConvNet	METAVRF [33]	Model	54.2 ± 0.8	67.8 ± 0.7
	HyperProto [12]	Metric	54.43 ± 0.20	72.67 ± 0.15
	Meta-LSTM [40]	Optim	43.56 ± 0.84	60.60 ± 0.71
	MAML [1]	Optim	48.70 ± 1.75	63.11 ± 0.91
	FOMAML [1]	Optim	48.07 ± 1.75	63.15 ± 0.91
	iMAML [5]	Optim	49.30 ± 1.88	63.47 ± 0.90
	Reptile [71]	Optim	49.97 ± 0.32	65.99 ± 0.58
	GEM-BML+ [36]	Optim	50.03 ± 1.63	-
	Lazy-Reptile [35]	Optim	51.50 ± 1.00	67.22 ± 0.97
	TMAML [72]	Optim	51.77 ± 1.86	65.6 ± 0.93
	CAVIA [2]	Optim	51.82 ± 0.65	65.85 ± 0.55
	MAML++ [69]	Optim	52.15 ± 0.26	68.32 ± 0.44
	L2F [37]	Optim	52.10 ± 0.50	69.38 ± 0.46
	MeTAL [38]	Optim	52.63 ± 0.37	70.52 ± 0.29
	ALFA [42]	Optim	52.76 ± 0.52	71.44 ± 0.45
	Ours	Optim	$\mathbf{54.66 \pm 0.55}$	$\mathbf{72.90 \pm 0.50}$
Others	HyperProto [12]	Metric	59.47 ± 0.20	76.84 ± 0.14
	METAVRF [33]	Model	$\mathbf{63.80 \pm 0.05}$	77.97 ± 0.28
	LEO [70]	Optim	61.76 ± 0.08	77.59 ± 0.12
ResNet12	SNAIL [32]	Model	55.71 ± 0.99	68.88 ± 0.92
	adaResNet [73]	Model	56.88 ± 0.62	71.94 ± 0.57
	MetaFun [31]	Model	62.12 ± 0.30	77.78 ± 0.12
	TADAM [74]	Metric	58.50 ± 0.30	76.70 ± 0.30
	MetaOptNet [75]	Metric	62.64 ± 0.61	78.63 ± 0.46
	MAML [1]	Optim	51.03 ± 0.50	68.26 ± 0.47
	MetaGAN [76]	Optim	52.71 ± 0.64	68.63 ± 0.67
	L2F [37]	Optim	57.48 ± 0.49	74.68 ± 0.43
	MeTAL [38]	Optim	59.64 ± 0.38	76.20 ± 0.19
	CAML [77]	Optim	59.23 ± 0.99	72.35 ± 0.71
	ALFA [42]	Optim	60.06 ± 0.49	77.42 ± 0.42
	Ours	Optim	$\mathbf{63.13 \pm 0.41}$	$\mathbf{81.04 \pm 0.39}$

'Optim', 'Model', and 'Metric' mean the optimization-based, model-based, and metric-based meta-learning methods, respectively. 'Others' means some other backbones that are larger than ResNet12, such as ResNet18 [12] and WRN-28-10 [33], [70].

TABLE 2
Accuracy (%) Comparisons With the State-of-the-Art Few-Shot Classification Methods on the Tiered-ImageNet Dataset

Backbone	Method	Category	1-shot 5-way	5-shot 5-way
ConvNet	MAML [1]	Optim	49.06 ± 0.50	67.48 ± 0.47
	FOMAML [1]	Optim	50.12 ± 1.82	67.43 ± 1.80
	iMAML [5]	Optim	51.51 ± 1.80	69.92 ± 1.70
	Reptile [71]	Optim	51.34 ± 0.40	68.73 ± 0.40
	L2F [37]	Optim	54.40 ± 0.50	73.34 ± 0.44
	Lazy-Reptile [35]	Optim	54.41 ± 1.00	72.21 ± 0.94
	MeTAL [38]	Optim	54.34 ± 0.31	70.40 ± 0.21
	ALFA [42]	Optim	55.06 ± 0.50	73.94 ± 0.43
	Ours	Optim	$\mathbf{57.13 \pm 0.48}$	$\mathbf{75.70 \pm 0.41}$
ResNet12	MetaFun [31]	Model	67.27 ± 0.20	83.28 ± 0.12
	ProtoNet [23]	Metric	53.51 ± 0.89	72.69 ± 0.74
	RelationNet [24]	Metric	54.48 ± 0.93	71.32 ± 0.78
	MetaOptNet [75]	Metric	65.99 ± 0.72	81.56 ± 0.53
	DSN [30]	Metric	66.22 ± 0.75	82.79 ± 0.48
	MAML [1]	Optim	58.58 ± 0.49	71.24 ± 0.43
	L2F [37]	Optim	63.94 ± 0.48	77.61 ± 0.41
	MeTAL [38]	Optim	63.89 ± 0.43	80.14 ± 0.40
	ALFA [42]	Optim	64.43 ± 0.49	81.77 ± 0.39
	Ours	Optim	$\mathbf{68.46 \pm 0.56}$	$\mathbf{83.84 \pm 0.40}$

the mean accuracy over 10000 tasks on the test set with the 95% confidence interval. We updated the curvature generation scheme, curvature updating scheme, and initialization per each task during the training. Besides, following the work [69], we applied the multi-step loss optimization scheme in the meta-training stage to obtain training stability and utilized the cosine annealing scheme to adjust the learning rate of meta-training. In few-shot classification tasks, we set $m = 16$ (number of constant curvature spaces) and utilized the same dimensionality for all submanifolds of the product manifold.

6.1.2 Results

Results on the mini-ImageNet, tiered-ImageNet, CUB, and CIFAR-FS datasets are shown in Tables 1, 2, 3, and 4, respectively. Our method belongs to optimization-based meta-learning. CAVIA [2], MAML++ [69], L2F [37], ALFA [42], CAML [77], Lazy-Reptile [35], MeTAL [38], and LEO [70] are state-of-the-art optimization-based methods. Compared with them, the main difference of our method is that we are capable of fast adaptation to various forms of manifold data, showing improvements on these methods. For example, in the 1-shot task of using ConvNet on the mini-ImageNet dataset (in Table 1), the accuracies of MAML++, L2F, ALFA, Lazy-Reptile and MeTAL read as 52.15%, 52.10%, 52.76%, 51.50%, and 52.63%, respectively. Our method achieves 54.66%, outperforming them by around 2%. LEO uses a deeper backbone (i.e., WRN-28-10 in Table 1) than ours (i.e., ResNet12), while the results imply our effectiveness, with improvements of 1.37% and 3.45% in the 1-shot and 5-shot tasks, respectively. This suggests that natural data usually has manifold structures, demonstrating the necessity of constructing suitable Riemannian geometry for manifold data. Similarly, experimental results on the tiered-ImageNet (in Table 2), CUB (in Table 3), and CIFAR-FS (in Table 4) datasets further suggest our superiority. On the CIFAR-FS dataset, our method outperforms compared methods by more than 8% and 5% in the 1-shot and 5-shot tasks.

Compared with metric-based and model-based methods, our method performs competitively or even exceeds many state-of-the-art methods. For example, in the 5-shot task on

TABLE 3
Accuracy (%) Comparisons With the State-of-the-Art Few-Shot Classification Methods on the CUB Dataset

Backbone	Method	Category	1-shot 5-way	5-shot 5-way
ConvNet	ProtoNet [23]	Metric	51.31 ± 0.91	70.77 ± 0.69
	MatchNet [59]	Metric	61.16 ± 0.89	72.86 ± 0.70
	RelationNet [24]	Metric	62.45 ± 0.98	76.11 ± 0.69
	Adver-Align [82]	Metric	63.30 ± 0.94	81.35 ± 0.67
	MAML [1]	Optim	55.92 ± 0.95	72.09 ± 0.76
	Ours	Optim	$\mathbf{64.74 \pm 0.96}$	$\mathbf{81.66 \pm 0.58}$
Others	RelationNet [24]	Metric	67.59 ± 1.02	82.75 ± 0.58
	ProtoNet [23]	Metric	71.88 ± 0.91	87.42 ± 0.48
	MatchingNet [59]	Metric	72.36 ± 0.90	83.64 ± 0.60
	Delta-encoder [78]	Aug	69.80 ± 0.46	82.60 ± 0.35
	AFHN [79]	Aug	70.53 ± 1.01	83.95 ± 0.63
	MAML [1]	Optim	69.96 ± 1.01	82.70 ± 0.65
	S2M2 [80]	Metric	72.92 ± 0.83	86.55 ± 0.51
	DEML [81]	Model	66.95 ± 1.06	77.11 ± 0.78
ResNet12	FEAT [25]	Metric	73.27 ± 0.22	85.77 ± 0.14
	RENet [26]	Metric	79.49 ± 0.44	91.11 ± 0.24
	LLP [83]	Metric	$\mathbf{79.77 \pm 0.44}$	$\mathbf{92.07 \pm 0.25}$
	TriNet [84]	Aug	69.61 ± 0.46	84.10 ± 0.35
	Ours	Optim	74.00 ± 1.00	86.77 ± 0.75

'Aug' means the data augmentation technique for few-shot learning. 'Others' means some other backbones that are larger than ResNet12, such as ResNet18 [1], [23], [24], [59], [78], [79], ResNet34 [80], and ResNet50 [81].

TABLE 4
Accuracy (%) Comparisons With the State-of-the-Art Few-Shot Classification Methods on the CIFAR-FS Dataset

Backbone	Method	Category	1-shot 5-way	5-shot 5-way
ConvNet	METAVRF [33]	Model	63.10 ± 0.70	76.50 ± 0.90
	MAML [1]	Optim	56.50 ± 1.90	70.50 ± 0.90
	FOMAML [1]	Optim	55.60 ± 1.88	69.52 ± 0.91
	Reptile [71]	Optim	57.50 ± 0.45	71.88 ± 0.42
	Lazy-Reptile [35]	Optim	59.36 ± 1.44	74.90 ± 1.28
	Ours	Optim	$\mathbf{65.43 \pm 0.90}$	$\mathbf{81.50 \pm 1.08}$
ResNet12	Shot-Free [68]	Metric	69.20	84.70
	TEWAM [29]	Metric	70.40	81.30
	ProtoNet [23]	Metric	72.20 ± 0.70	83.50 ± 0.50
	MetaOptNet [75]	Metric	72.60 ± 0.70	84.30 ± 0.50
	RENet [26]	Metric	74.51 ± 0.46	86.60 ± 0.32
	DSN [30]	Metric	75.60 ± 0.90	86.20 ± 0.60
	MCGN [85]	Metric	76.45 ± 0.99	88.42 ± 0.23
	RFS [86]	Metric	73.90 ± 0.80	86.90 ± 0.50
	Rizve *et al.* [87]	Metric	77.87 ± 0.85	89.74 ± 0.57
	MABAS [88]	Aug	73.51 ± 0.92	85.49 ± 0.68
	Ours	Optim	$\mathbf{86.40 \pm 0.80}$	$\mathbf{94.87 \pm 0.50}$

'Aug' means the data augmentation technique for few-shot learning.

the mini-ImageNet (in Table 1) and CIFAR-FS (in Table 4) datasets, our method improves upon the model-based method METAVRF [33] by 5.1% and 5%, using ConvNet as the backbone. When ResNet12 is used as our backbone, our performance gain is 3.07% in the 5-shot task on the mini-ImageNet dataset, against METAVRF using deeper WRN-28-20 as the backbone. Note that HyperProto [12] is a metric-based method and learns a robust distance measure in hyperbolic manifold. The results suggest that the proposed curvature-adaptive meta-learning method is better than HyperProto. On the mini-ImageNet dataset (in Table 1), the performance of ours is remarkable, even 4.2% higher in the 5-shot task, where we use the ResNet12 backbone while HyperProto uses a deeper ResNet18 backbone. The reason is that HyperProto focuses on a fixed form of manifold data, while the proposed method is capable of quickly adapting to various forms of manifold data. On the CUB dataset (in Table 3), the performance of our method is worse than the metric-based method RENet [26] and LLP [83]. RENet extracts expressive correlations of images via attention for robust embedding space and LLP utilizes local features, which are beneficial to fine-gradient images in CUB.

The backbone comparisons on the four datasets are also shown in the four tables. The ConvNet is a shallower backbone than the ResNet12, and thus using the ConvNet has worse performance than using the ResNet12. We observe that our method achieves better performance compared with state-of-the-art optimization-based methods on both the two backbones. For example, compared with the state-of-the-art optimization-based method ALFA [42] on the tiered-ImageNet dataset (in Table 2), improvements of 2.07% and 1.76% are achieved in the 1-shot and 5-shot tasks using the ConvNet backbone, and improvements of 4.03% and 2.07% are obtained in the 1-shot and 5-shot tasks using the ResNet12 backbone. The ConvNet and ResNet12 backbones we used have the same architectures with those in euclidean meta-learning methods. This gains are mainly attributed to the fact that our method is capable of constructing appropriate Riemannian geometry for natural data. Overall, our method can achieve good performance no matter which network architecture is used.

TABLE 5
MSE Over 100 Sampled Tasks on Few-Shot Regression (A Smaller Value Means a Better Performance)

Method	2 hidden layers			3 hidden layers		
	5-shot	10-shot	20-shot	5-shot	10-shot	20-shot
MAML [1]	1.24	0.75	0.49	0.84	0.56	0.33
TR [89]	1.09	0.66	-	-	-	-
ALFA [42]	0.92	0.62	0.34	0.70	0.51	0.25
L2F [37]	0.70	0.36	0.16	-	-	-
Ours	**0.28**	**0.09**	**0.04**	**0.14**	**0.06**	**0.03**

6.2 Few-Shot Regression

6.2.1 Sine Curves

A k-shot regression task is defined as training a neural network to fit an unknown wave, given k points. Following the standard protocol [1], [42], our goal is a sine wave with varying amplitude, frequency, and phase, sampled in the range of $[0.1, 5.0]$, $[0.8, 1.2]$, and $[0, \pi]$, respectively. We evaluated our method with $k = 5, 10, 20$ points and two neural network architectures: a two-hidden-layer multi-layer perceptron (MLP) with 40 hidden neurons and a three-hidden-layer MLP with 80 hidden neurons.

In this few-shot regression task, we set the number m of constant curvature spaces as 8, and constant curvature spaces had the same dimension. In the two-hidden-layer MLP, the dimension of each constant curvature space was 5, and in the three-hidden-layer MLP, the dimension of each constant curvature space was 10. Our method was trained with $\tau = 1$ gradient step, and evaluated with $\tau = 5$ gradient steps. We set 240 epochs in the meta-training stage, and each epoch had 500 few-shot learning tasks. The batch-size in meta-training was 5, that is, we updated our curvature generation scheme and curvature updating scheme using five few-shot regression tasks once. The performance of our method was measured in mean-square error (MSE).

Results are shown in Table 5. Compared with state-of-the-art meta-learning methods, MAML [1], TR [89], ALFA [42], and L2F [37], our method fits the wave more accurately, significantly improving the performance. Even given very few points, our method still has a lower error. For example, in the 5-shot tasks, MSE of our method is 0.28 and 0.14 with the two network architectures, while MAML achieves 1.24 and 0.84, and ALFA achieves 0.92 and 0.70, having much larger errors than our method. L2F achieves the best performance among compared methods, and our method still has 0.42, 0.27, and 0.08 improvements compared with it, when using the two-hidden-layer neural network. This demonstrates that the proposed method captures the underlying structure of data better by constructing suitable spaces. We think our gains here are that constant curvature space can encode the nonlinear sine curves and their properties better than a simple euclidean space.

We also show qualitative results in Fig. 6. We compare our method with MAML that uses a conventional neural network in euclidean space and learns parameter initialization,

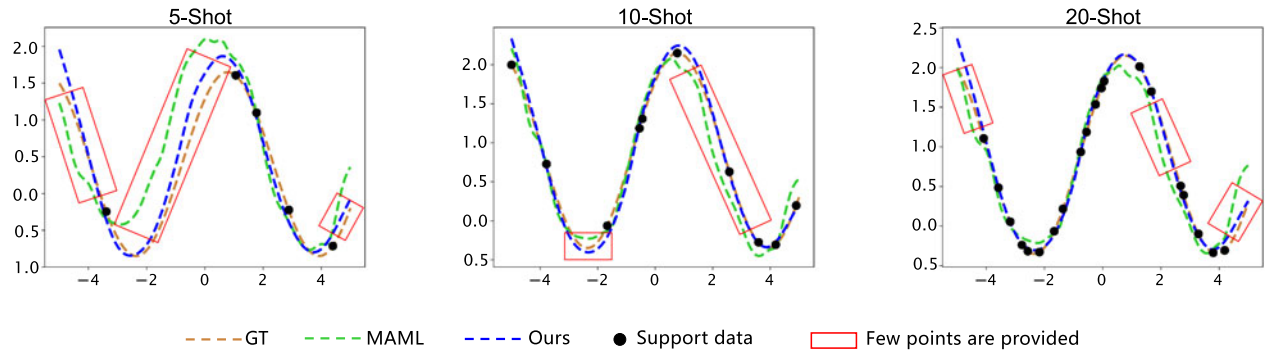


Fig. 6. Qualitative results on few-shot regression tasks, where the two-hidden-layer neural network is used. We plot the waves on the 5-shot, 10-shot, 20-shot tasks. 'GT' is the true sine wave, and the black points denote the support data for training. Our method better fits a sine wave, given only few points. In the red boxes, few points are provided. In this case, our method still fits the wave well, while MAML has a large error.

while our method uses a product manifold neural network, and learns a curvature generation scheme and a curvature updating scheme. Experimental results also show that our method better fits a wave using few points. In areas where much training points are provided, both our method and MAML fit the wave well. But in areas where few training points are provided (i.e., red boxes in the figure), MAML has a large error, while our method still fits the wave well. For example, in the 5-shot task, when the value of the horizontal axis is smaller than -4, there is no training point. In this case, our method has a small error with the ground truth, while MAML has a large deviation. In the 10-shot task, when the value of the horizontal axis is around 2, there is only one training point, and our method fits the wave better.

6.2.2 Image Completion

We evaluated our method on a more challenging few-shot regression task, i.e., the image completion task. We regard each image as an individual task. For each task, we are given an image with few pixels, and our goal is to predict the rest pixels of this image via our product manifold neural network. The given pixels in an image are the support data, and the remaining unknown pixels are the query data. We utilized a 6-layer neural network for image completion, and the size of hidden layers was 128. The input of the neural network is a coordinate in the image and the output is the predicted pixel of this coordinate. In the first five layers, we processed data in learned tangent spaces and mapped results to the manifold of the next layer. In the last layer, we projected input into the tangent space at the origin and utilized the results in the tangent space as the predicted pixel values.

We utilized the CelebA dataset [90], and trained our method using the training set of CelebA. In the meta-training stage, we sampled images from the training set to learn the initial product manifold neural network, a curvature generation scheme, and a curvature updating scheme. In the meta-test stage, we sampled images from the test set, and our results were measured over 100 images. For a new task, we first sent support pixels to the initial product manifold neural network to extract features by Eq. (25), and generated task-specific curvature initialization via our curvature generation scheme in Eqs. (26) and (27). Then, we performed optimization on the neural network by using the support data, where parameters were updated by the conventional gradient descent, and curvatures were updated by our curvature updating scheme in Eqs. (30) and (28). After several iterations of the optimization, we can obtain a task-specific product manifold neural network for the given image. We used the updated neural network to predict values of the rest pixels of this image.

We set the number m of constant curvature spaces as 4, and the dimension of each constant curvature space was 32. We set the batchsize of meta-training as 25. Our method was trained with $\tau = 5$ gradient steps, and evaluated with $\tau = 5$ gradient steps. We trained our method for 5 epochs, where each epoch had 10000 few-shot learning tasks. We evaluated our method with two settings: random pixels and ordered pixels as input data. In each image, 10 or 100 pixels were provided for training/evaluation.

Experimental results are shown in Table 6. Our method achieves better performance than compared optimization-based meta-learning methods (e.g., MAML [1] and CAVIA [2]), no matter in the random pixel setting or ordered pixel setting. Results show that using the product manifold of constant curvature spaces and updating curvatures for different tasks better fits various practical data. Here we also show some examples of our method and CAVIA, as shown in Fig. 7. Compared with CAVIA, our method achieves better image completion. For example, in the first image, our method produces better background. In the second image, our method better fits the face orientation. These experimental results show the superiority of modeling natural data with constant curvature space over methods that limit themselves with euclidean space.

TABLE 6
MSE of Pixels in Image Completion Tasks on the CelebA Dataset

Method	Random Pixels		Ordered Pixels	
	10	100	10	100
CNP [90]	0.039	0.016	0.057	0.047
MAML [1]	0.040	0.017	0.055	0.047
CAVIA [2]	0.037	**0.014**	0.053	0.047
Ours	**0.034**	**0.014**	**0.052**	**0.045**

We evaluate our method with random training pixels and ordered training pixels, and ordered pixels are chosen from the top-left corner to the bottom-right. In each image, 10 or 100 pixels are provided for training.

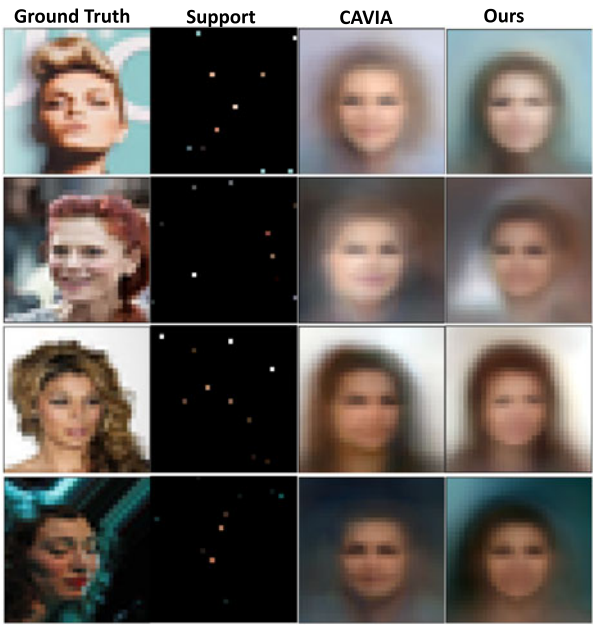
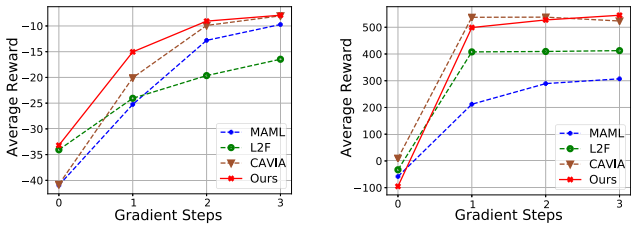


Fig. 7. Qualitative results on the CelebA dataset, where 10 random pixels are provided as the support data. Our method generates better completion results, such as better fitting background and face orientation.



(a) 2D Navigation (b) Direction

Fig. 8. Reinforcement learning results on 2D navigation and direction tasks. We plot the obtained reward and gradient steps, and the results show that our learned model can more quickly adapt to a new reinforcement learning task than compared methods and obtain greater reward.

6.3 Reinforcement Learning

6.3.1 Setting

We used a reinforcement learning (RL) environment MuJoCo [91] to evaluate our method on two types of RL tasks. First, we consider a 2D navigation task. A robot moves from a starting point to a destination point in a 2D space, and the reward is the negative squared distance between the current point and the destination point. We followed the standard protocol [1], [2], fixed the starting point, and varied the destination point for different tasks. The second task is a high-dimensional locomotion task. Therein, the goal is to find a particular direction of a cheetah robot. In this case, the reward is the magnitude of the velocity in the forward or backward direction. In our experiments, the search trajectory of a robot was collected in a rollout. For each task, we used 20 rollouts to update the model, with each rollout constituting of 200 search steps of the robot. In the meta-learning process, we updated the network through 500 batches, with the batchsize in meta-learning being 40. That is, we updated our curvature generation scheme and curvature updating scheme over 40 RL tasks. Our method was trained with $\tau = 1$ gradient step, and evaluated with $\tau = 3$ gradient steps. For the RL tasks, we used a product manifold neural network with two hidden-layers of size 100. We set the number of constant curvature spaces as $m = 10$, and the dimension of the spaces was set as 10.

6.3.2 Results

We compare our method with meta-learning methods: MAML [1], L2F [37], and CAVIA [2]. Results on the studied two RL tasks are shown in Fig. 8. Experimental results show that our method performs competitively or even exceeds various state-of-the-art algorithms in adapting a model to RL tasks. For example, in 2D navigation experiments, our method demonstrates better initialization. At beginning, the obtained reward of MAML and CAVIA is about -40, while the obtained reward of our model is larger than -33, higher than that of them. In the adaptation process, the following steps of our method further improve the performance. After the adaptation process, the reward of our method is about -7.91, still higher than compared methods. This shows that constant curvature space better encodes nonlinear data in RL tasks. Furthermore, updating curvature via our method will lead to construction of suitable space to represent data, and in turn performance improvements.

6.4 Ablation Study

In this section, we discuss ablation experiments on the few-shot classification tasks using the mini-ImageNet dataset to evaluate our method. We denote the configuration where each layer in a product manifold neural network has its individual product manifold as 'MM', and the configuration where all layers of a product manifold neural network share a common product manifold as 'SM'. We evaluated the product manifold neural network with only one constant curvature space by manually setting the curvature at -1, -0.01, 0.01, or 1. We refer to this by 'SM$= -1/-0.01/1/0.01$', respectively. We denote our curvature generation scheme as 'CGS'. If only learning a common curvature initialization is considered in the meta-training process, the result is shown with 'w/o CGS'. Similarly, we denote our curvature updating scheme as 'CUS', and using only a simple gradient descent to update curvatures is shown with 'w/o CUS'.

We also evaluated the generated learning rates and search directions in our curvature updating scheme. Removing the procedure that generates the learning rates and search directions is shown with 'CUS w/o lr' and 'CUS w/o sd', respectively. Moreover, we assessed our attention classifier. In doing so, we replaced the attention classifier with a fully-connected layer and assigned uniform weights to all constant curvature spaces. This is shown by 'Ours w/o att'. We report the results for all the aforementioned variations in Table 7.

Comparing 'SM$= -1/-0.01/1/0.01$' with 'SM' and 'MM' in Table 7 reveals that updating curvatures to suitable values leads to better performance than manually setting fixed curvatures for various tasks. This is because a fixed curvature cannot be optimal for all tasks, given the complexity of natural data. This demonstrates that real tasks have different manifold structures, requiring the space to vary its curvature. Comparing 'SM' with 'MM', using individual product manifolds in different layers brings obvious improvements, where 'MM' is 1.97% and 1.11% higher than 'SM'. Comparing 'MM +CGS' with 'MM', we find that our curvature generation

TABLE 7
Ablation Experiments on the Mini-ImageNet Dataset

Method	1-shot 5-way	5-shot 5-way
SM$=-1$	57.90	75.94
SM$=-0.01$	57.44	75.56
SM$=1$	56.06	72.90
SM$=0.01$	56.97	76.06
SM	57.13	76.14
MM	59.10	77.25
MM+CGS	61.66	80.25
MM+CUS w/o lr	61.80	78.70
MM+CUS w/o sd	61.75	78.54
MM+CUS	61.96	78.90
Ours w/o att	62.41	80.34
Ours (MM+CGS+CUS)	**63.13**	**81.04**

TABLE 8
Evaluation of m on the Mini-ImageNet Dataset

Method	1-shot 5-way	5-shot 5-way
$m=2$	58.31	76.68
$m=4$	58.75	80.54
$m=8$	59.22	80.73
$m=16$	**63.13**	**81.04**
$m=32$	60.91	77.88

scheme has 2.56% and 3% improvements. Besides, we find that in our curvature updating scheme, producing adaptive learning rates and search directions both play important roles in finding better optima. 'MM+CUS' achieves 61.96% and 78.90% on the 1-shot and 5-shot tasks, higher than the performance of 'MM+CUS w/o lr' and 'MM+CUS w/o sd'. Thus, both the curvature generation scheme and the curvature updating scheme make our method more flexible. Comparing 'Ours (MM+CGS+CUS)' with 'Ours w/o att', our attention classifier brings 0.72% and 0.70% boost, showing that it can efficiently utilize features from different spaces for classification.

6.5 Hyperparameter Analyses

In this part, we assess the number of constant curvature spaces (i.e., m) in the product manifold on the mini-ImageNet dataset. Note that while we have used different number of constant curvature spaces in various experiments, features of a layer in this experiment have the same dimension. For example, if the dimension of features in a product manifold neural network is 64, using 8 constant curvature spaces means the dimension of constant curvature spaces is 8, and using 16 constant curvature spaces means the dimension of constant curvature spaces is 4. We measured m in the range of $[2, 4, 8, 16, 32]$, and report results in Table 8. By increasing the number of constant curvature spaces, the performance of our method first increases and then decreases. We achieve the best performance when $m = 16$, 63.13% and 81.04% on the 1-shot and 5-shot tasks, respectively. Thus, we choose $m = 16$ in the classification tasks for the mini-ImageNet, tiered-ImageNet, CUB, and CIFAR-FS dataset.

6.6 Visualization

6.6.1 Feature Distribution

In this section, we show the effect of updating curvatures by visualizing feature distributions in few-shot classification tasks. From the feature distribution, we can evaluate the matching degree between updated curvatures and manifold structures of data, and further assess the quality of updated curvatures. We first meta-trained a set of common curvatures for a product manifold neural network and fixed them for all tasks, and took a task as an example to show its feature distribution before the classifier. We then show such a feature distribution of the adaptive curvatures in our method. Concretely, we sampled one task from the mini-ImageNet dataset, and used the MDS method [92] to embed features into a 2-D space. Results are shown in Fig. 9. We can find that our adaptive curvatures lead to a better feature distribution than common curvatures. For example, in the left panel for common curvatures, features of 'worm fence' and 'hotdog' classes are mixed in the space, while features of the two classes in the right figure have clear boundaries. This shows the benefit of adapting the curvatures. Fixed geometry realized by common curvatures may incur distortions to data, while our method constructs suitable space for manifold data by updating curvature through few steps, making features more discriminative.

6.6.2 Updated Curvatures

We show changes of curvature values after our curvature generation scheme (denoted by CGS) and curvature updating scheme (denoted by CUS). We stored curvatures of a product manifold neural networks into a vector, and the changes were measured by L2 distance between vectors of common curvature initialization, generated task-specific curvature initialization, and updated curvatures. We also plot the accuracies after the two schemes. Results are measured across 10000 tasks on mini-ImageNet, as shown in Fig. 10. It shows that both CGS and CUS together work to adapt curvatures. Compared with the common curvature initialization, the task-specific initialization produced via CGS is much closer to updated curvatures, bringing a shorter optimization trajectory. Then CUS further finds better curvatures.

We measured the distribution of updated curvatures across 100 tasks on mini-ImageNet, and plot their frequencies over 10 bins in the range of $[-0.5, 0.5]$. Results are shown in Fig. 11. While the histogram has several dominant bins, curvatures are spread across all bins. This confirms the necessity of curvature adaptation for features.

6.6.3 Produced Learning Rates and Search Directions

We visualize produced learning rates via our curvature updating scheme on the mini-ImageNet dataset (see Fig. 12a). We plot the gradient norms of curvatures in Fig. 12b. Both learning rates and gradient norms were measured by computing the mean over 500 tasks. Our analysis suggests that at the beginning of optimization (i.e., first step), curvatures have large gradients, and our method produces small learning rates for them. As the optimization continues, gradient norms gradually decrease. Our method produces the biggest learning rate at the second optimization step, while showing a decreasing trend afterwards. This is consistent with the recent

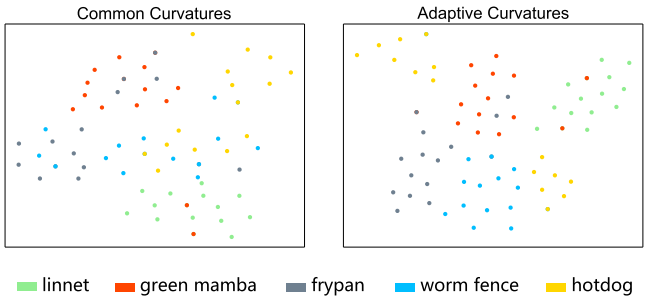


Fig. 9. Feature distributions of a few-shot classification task on the mini-ImageNet dataset. In the left panel, we plot the feature distribution of using common curvatures for all tasks. In the right panel, we plot the feature distribution of adapting the curvatures.

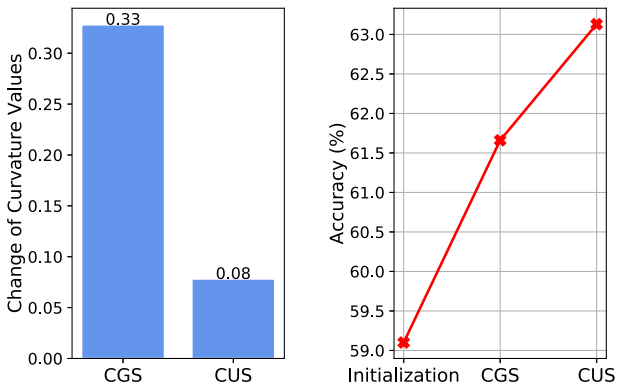


Fig. 10. In the left panel, we plot changes of curvature values after our curvature generation scheme (CGS) and curvature updating scheme (CUS). We plot accuracies after the two schemes in the right panel.

research advance [93], where small learning rates are assigned to large gradients, and large learning rates are assigned to small gradients, making optimization more robust and efficient.

We plot the cosine similarities between our generated search directions and full gradients in Fig. 13a to show the effectiveness of our generated search directions in CUS. Note that, the full gradients are the optimal optimization directions for curvatures, and are obtained using all training data in the dataset. Large cosine similarities mean the produced search directions are very aligned (i.e., small deviations) with the optimal optimization directions. Here, we also plot the cosine similarities between stochastic gradients (inputs of our CUS) and full gradients. We can conclude that our curvature updating scheme transforms input gradients into better search directions that have higher cosine similarities with the full gradients. This shows that our curvature updating scheme learns to find better search directions, leading to a faster convergence.

Besides, we show loss values using our curvature updating scheme to further show its effectiveness. We compare our curvature updating scheme with a hand-designed gradient descent scheme with a fixed learning rate and gradients being search directions, results are shown in Fig. 13b. We conducted experiments on 1-shot 5-way tasks using the mini-ImageNet datasest, and computed the average loss over 600 tasks sampled from the test set. We tuned the fixed learning rate in the hand-designed gradient descent scheme to achieve the best performance. One can draw the following conclusions. (1) The curvature updating scheme has

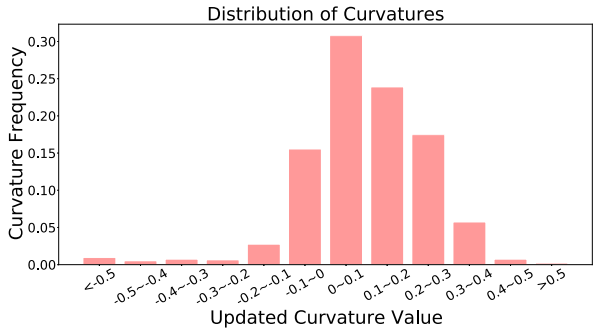
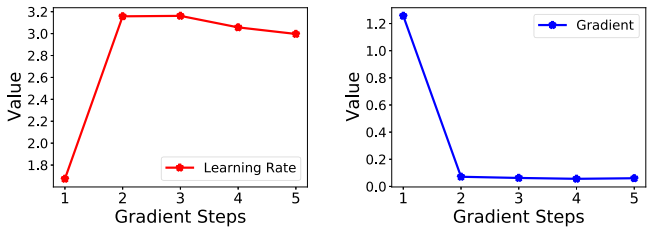
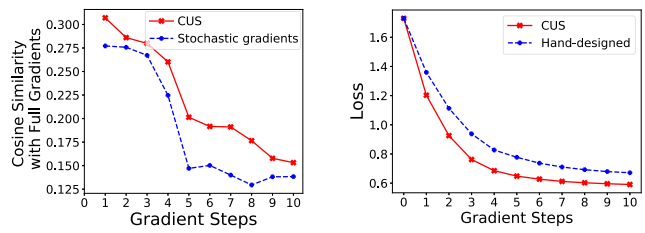


Fig. 11. Distribution of the updated curvatures on the mini-ImageNet dataset.



(a) Learning rate (b) Gradient

Fig. 12. Generated learning rates and gradient norms for the curvatures on the mini-ImageNet dataset.



(a) Search direction (b) Loss curvature

Fig. 13. In Fig. 13a, we plot cosine similarities between our generated search directions and full gradients using the mini-ImageNet dataset, (the red curve). We also plot the cosine similarities between stochastic gradients and full gradients, (the blue curve). The two curves show that our generated search directions have smaller deviations from the optimal optimization directions (the full gradients). In Fig. 13b, we measured loss values in the adaptation to unseen data using our curvature updating scheme ('CUS') and a hand-designed updating scheme that uses a fixed learning rate and uses gradients as search directions ('Hand-designed').

faster optimization convergence. (2) The curvature updating scheme obtains a better optimum. We note that the curvature updating scheme achieves the loss about 0.6, while using a hand-designed gradient descent scheme achieves a loss about 0.7. This suggests that the proposed updating scheme makes the model better fit to data. (3) A hand-designed updating scheme needs much human involvements to tune hyperparameter (e.g., learning rates). In contrast, our curvature updating scheme automatically finds good optimization trajectories, reducing human involvements.

6.7 Running Time

Here, we provide the wall-clock time of our method on the 1-shot 5-way and 5-shot 5-way tasks of few-shot classification tasks. We compare our training and testing time with the MAML method that is a standard meta-learning method in euclidean space. The training time was measured after

TABLE 9
Training and Testing Time (seconds) on the Mini-ImageNet Dataset

Method	MAML [1]	Ours w/o CGS CUS	Ours
1-shot training time	154	417	538
1-shot test time	48	144	189
5-shot training time	184	436	568
5-shot test time	56	162	198

500 episodes, and the test time was measured after 100 episodes. Experiments were conducted on the mini-ImageNet dataset, where the ConvNet backbone was used. The performance was measured using an Inter(R) Core(TM) i9-10900X 3.7GHz CPU, a GeForce RTX 3090 GPU and 128GB RAM. Results are shown in Table 9 with "Ours w/o CGS CUS" denoting the extra time cost caused by the orthogonal projection operation, the exponential map, and the logarithmic map. Also, "Ours" denotes the extra time cost caused by not only the three operations, but also the curvature generation and curvature updating schemes.

From the experiments, we find that our method brings better performance in the expense of computational overhead. Further, the three extra operations in the product manifold neural network require more time cost than the curvature generation and curvature updating schemes. Although the orthogonal projection, exponential map, and the logarithmic map in constant curvature space are element-wise operations (i.e., elements of a vector are independent in the three operations, and we do not need complex matrix function), they still incur computational cost. This motivates us to study more efficient curvature adaptation schemes in the future.

6.8 Convergence

In this section, we provide convergence analysis of our meta-learning process. We plot the meta-objective (i.e., Eq. (24)) on the mini-ImageNet 5-shot tasks in Fig. 14. The learning curve demonstrates that the curvature-adaptive meta-learning converges steady. Recall that the meta-objective is composed of loss values of updated networks on query data of given tasks. Thus, the decreasing trend of the meta-objective shows that our method discovers how to tune a model to suitable curvatures and parameters. As a result, the proposed method is able to construct suitable product manifolds for given tasks with complex and various manifold structures of data.

7 Conclusion

In this paper, we have presented a curvature-adaptive meta-learning method that can quickly adapt a model to manifold data. The proposed product manifold neural network with multiple constant curvature spaces can better model complex structures of natural data. Our curvature generation scheme and curvature updating scheme can construct suitable product manifolds for various manifold data through few optimization steps on curvatures. Assigning a task-specific curvature initialization brings a shorter optimization trajectory, and generating learning rates and search directions adaptively leads to faster convergence and better optima as compared to using a fixed optimization scheme. Extensive experiments on few-shot classification, few-shot regression, and reinforcement learning tasks show that our method outperforms existing meta-learning methods designed in euclidean space, and updating curvatures makes a model adapt to manifold data well. Since theories of Riemannian manifold lead to many strong and elegant results, we think exploring other Riemannian geometries and developing more efficient ways to adapt a model to Riemannian manifold are potential research directions for the future.

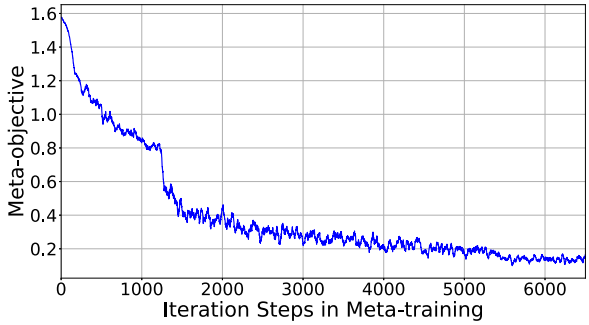


Fig. 14. Meta-objective in the 5-shot tasks on the mini-ImageNet dataset.

References

[1] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 1126–1135.
[2] L. M. Zintgraf, K. Shiarlis, V. Kurin, K. Hofmann, and S. Whiteson, "Fast context adaptation via meta-learning," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 7693–7702.
[3] R. Gong *et al.*, "Cluster, split, fuse, and update: Meta-learning for open compound domain adaptive semantic segmentation," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 8344–8354.
[4] H. Cho, Y. Cho, J. Yu, and J. Kim, "Camera distortion-aware 3D human pose estimation in video with optimization-based meta-learning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 11 169–11 178.
[5] A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine, "Meta-learning with implicit gradients," in *Proc. Int. Conf. Neural Informat. Process. Syst.*, 2019, pp. 113–124.
[6] J. Liu, L. Song, and Y. Qin, "Prototype rectification for few-shot learning," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 741–756.
[7] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2016, pp. 1842–1850.
[8] S. Li, J. Xu, X. Xu, P. Shen, S. Li, and B. Hooi, "Spherical confidence learning for face recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 15 629–15 637.
[9] W. Peng, T. Varanka, A. Mostafa, H. Shi, and G. Zhao, "Hyperbolic deep neural networks: A survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, Dec. 21, 2021, doi: 10.1109/TPAMI.2021.3136921.
[10] T. Lin and H. Zha, "Riemannian manifold learning," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 5, pp. 796–809, May 2008.
[11] R. Chakraborty, J. Bouza, J. Manton, and B. Vemuri, "ManifoldNet: A deep neural network for manifold-valued data with applications," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 2, pp. 799–810, Feb. 2022.
[12] V. Khrulkov, L. Mirvakhabova, E. Ustinova, I. Oseledets, and V. Lempitsky, "Hyperbolic image embeddings," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 6418–6428.
[13] G. Qi, H. Yu, Z. Lu, and S. Li, "Transductive few-shot classification on the oblique manifold," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 8412–8422.
[14] S. Liu, J. Chen, L. Pan, C.-W. Ngo, T.-S. Chua, and Y.-G. Jiang, "Hyperbolic visual embedding learning for zero-shot recognition," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 9273–9281.

[15] D. Grattarola, D. Zambon, L. Livi, and C. Alippi, "Change detection in graph streams by learning graph embeddings on constant-curvature manifolds," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 31, no. 6, pp. 1856–1869, Jun. 2020.
[16] T. Long, P. Mettes, H. T. Shen, and C. G. Snoek, "Searching for actions on the hyperbole," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 1141–1150.
[17] O. Ganea, G. Bécigneul, and T. Hofmann, "Hyperbolic neural networks," in *Proc. Neural Informat. Process. Syst.*, 2018, pp. 5345–5355.
[18] O. Skopek, O.-E. Ganea, and G. Bécigneul, "Mixed-curvature variational autoencoders," in *Proc. Int. Conf. Learn. Representations*, 2020, pp. 1–44.
[19] A. Gu, F. Sala, B. Gunel, and C. Ré, "Learning mixed-curvature representations in product spaces," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–21.
[20] M. Defferrard, N. Perraudin, T. Kacprzak, and R. Sgier, "DeepSphere: Towards an equivariant graph-based spherical CNN," in *Proc. Workshop Representation Learn. Graphs Manifolds*, 2019, pp. 1–18.
[21] J. Yan, L. Luo, C. Deng, and H. Huang, "Unsupervised hyperbolic metric learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 12 465–12 474.
[22] Z. Huang, R. Wang, S. Shan, L. V. Gool, and X. Chen, "Cross euclidean-to-riemannian metric learning with application to face recognition from video," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 12, pp. 2827–2840, Dec. 2018.
[23] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Proc. Neural Informat. Process. Syst.*, 2017, pp. 4077–4087.
[24] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales, "Learning to compare: Relation network for few-shot learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 1199–1208.
[25] H.-J. Ye, H. Hu, D.-C. Zhan, and F. Sha, "Few-shot learning via embedding adaptation with set-to-set functions," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 8808–8817.
[26] D. Kang, H. Kwon, J. Min, and M. Cho, "Relational embedding for few-shot classification," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 8822–8833.
[27] A. Li, W. Huang, X. Lan, J. Feng, Z. Li, and L. Wang, "Boosting few-shot learning with adaptive margin loss," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 12576–12584.
[28] P. Bateni, R. Goyal, V. Masrani, F. Wood, and L. Sigal, "Improved few-shot visual classification," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 14 493–14 502.
[29] L. Qiao, Y. Shi, J. Li, Y. Wang, T. Huang, and Y. Tian, "Transductive episodic-wise adaptive metric for few-shot learning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 3602–3611.
[30] C. Simon, P. Koniusz, R. Nock, and M. Harandi, "Adaptive subspaces for few-shot learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 4136–4145.
[31] J. Xu, J.-F. Ton, H. Kim, A. R. Kosiorek, and Y. Teh, "MetaFun: Meta-learning with iterative functional updates," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 10 617–10 627.
[32] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, "A simple neural attentive meta-learner," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–17.
[33] X. Zhen *et al.*, "Learning to learn kernels with variational random features," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 11409–11419.
[34] H. Edwards and A. Storkey, "Towards a neural statistician," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–13.
[35] M. A. Jamal, L. Wang, and B. Gong, "A lazy approach to long-horizon gradient-based meta-learning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 6577–6586.
[36] Y. Zou and X. Lu, "Gradient-em Bayesian meta-learning," in *proc. Neural Informat. Process. Syst.*, 2020, pp. 20 865–20 875.
[37] S. Baik, J. Oh, S. Hong, and K. M. Lee, "Learning to forget for meta-learning via task-and-layer-wise attenuation," *IEEE Trans. Pattern Anal. Mach. Intell.*, early access, Aug. 04, 2021, doi:10.1109/TPAMI.2021.3102098.
[38] S. Baik, J. Choi, H. Kim, D. Cho, J. Min, and K. M. Lee, "Meta-learning with task-adaptive loss function for few-shot learning," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2021, pp. 9465–9474.
[39] M. Andrychowicz *et al.*, "Learning to learn by gradient descent by gradient descent," in *Proc. Neural Informat. Process. Syst.*, 2016, pp. 3981–3989.
[40] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *Proc. Int. Conf. Learn. Representations*, 2017, pp. 1–11.
[41] Z. Li, F. Zhou, F. Chen, and H. Li, "Meta-SGD: Learning to learn quickly for few-shot learning," 2017, *arXiv:1707.09835*.
[42] S. Baik, M. Choi, J. Choi, H. won Kim, and K. M. Lee, "Meta-learning with adaptive hyperparameters," in *Proc. Neural Informat. Process. Syst.*, 2020, pp. 20 755–20 765.
[43] Y. Nagano, S. Yamaguchi, Y. Fujita, and M. Koyama, "A wrapped normal distribution on hyperbolic space for gradient-based learning," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 4693–4702.
[44] R. Shimizu, Y. Mukuta, and T. Harada, "Hyperbolic neural networks++," in *Proc. Int. Conf. Learn. Representations*, 2020, pp. 1–25.
[45] Q. Liu, M. Nickel, and D. Kiela, "Hyperbolic graph neural networks," in *Proc. Neural Informat. Process. Syst.*, 2019, pp. 8230–8241.
[46] Y. Zhang, X. Wang, X. Jiang, C. Shi, and Y. Ye, "Hyperbolic graph attention network," 2019, *arXiv:1912.03046*.
[47] J. Dai, Y. Wu, Z. Gao, and Y. Jia, "A hyperbolic-to-hyperbolic graph convolutional network," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 154–163.
[48] I. Chami, A. Gu, D. Nguyen, and C. Ré, "HoroPCA: Hyperbolic dimensionality reduction via horospherical projections," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 1419–1429.
[49] I. Chami, A. Gu, V. Chatziafratis, and C. Ré, "From trees to continuous embeddings and back: Hyperbolic hierarchical clustering," in *Proc. Neural Informat. Process. Syst.*, 2020, pp. 15 065–15 076.
[50] R. Wilson, E. Hancock, E. Pekalska, and R. Duin, "Spherical and hyperbolic embeddings of data," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 11, pp. 2255–2269, Nov. 2014.
[51] G. Bachmann, G. Bécigneul, and O.-E. Ganea, "Constant curvature graph convolutional networks," in *Proc. Int. Conf. Mach. Learn.*, 2020, pp. 486–496.
[52] I. Chami, Z. Ying, C. Ré, and J. Leskovec, "Hyperbolic graph convolutional neural networks," in *Proc. Neural Informat. Process. Syst.*, 2019, pp. 4868–4879.
[53] Y. Zhang, X. Wang, C. Shi, N. Liu, and G. Song, "Lorentzian graph convolutional networks," in *Proc. Int. World Wide Web Conf.*, 2021, pp. 1249–1261.
[54] J. Park, J. Cho, H. J. Chang, and J. Y. Choi, "Unsupervised hyperbolic representation learning via message passing auto-encoders," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 5516–5526.
[55] J. M. Lee, *Riemannian Manifolds: An Introduction to Curvature*. Berlin, Germany: Springer, 1997.
[56] A. A. Ungar, *A Gyrovector Space Approach to Hyperbolic Geometry*. San Rafael, CA, USA: Morgan and Claypool, 2009.
[57] P. Absil, R. E. Mahony, and R. Sepulchre, *Optimization Algorithms on Matrix Manifolds*. Princeton, NJ, USA: Princeton Univ. Press, 2008.
[58] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 818–833.
[59] O. Vinyals *et al.*, "Matching networks for one shot learning," in *Proc. Neural Informat. Process. Syst.*, 2016, pp. 3630–3638.
[60] T.-Y. Lin, A. RoyChowdhury, and S. Maji, "Bilinear convolutional neural networks for fine-grained visual recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 6, pp. 1309–1322, Jun. 2018.
[61] Q. Wang, J. Xie, W. Zuo, L. Zhang, and P. Li, "Deep CNNs meet global covariance pooling: Better representation and generalization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 43, no. 8, pp. 2582–2597, Aug. 2021.
[62] Z. Gao, Y. Wu, X. Zhang, J. Dai, Y. Jia, and M. Harandi, "Revisiting bilinear pooling: A coding perspective," in *Proc. Conf. Artif. Intell.*, 2020, pp. 3954–3961.
[63] Z. Yu, J. Yu, J. Fan, and D. Tao, "Multi-modal factorized bilinear pooling with co-attention learning for visual question answering," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 1839–1848.
[64] M. Ren *et al.*, "Meta-learning for semi-supervised few-shot classification," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–15.
[65] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie, "The caltech-UCSD birds-200–2011 dataset," Tech. Rep. CNS-TR-2011-001, 2011.
[66] L. Bertinetto, J. F. Henriques, P. H. S. Torr, and A. Vedaldi, "Meta-learning with differentiable closed-form solvers," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–15.
[67] A. Krizhevsky *et al.*, "Learning multiple layers of features from tiny images," Tech. Rep., CIFAR, Univ. Toronto, 2009. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.222.9220

[68] A. Ravichandran, R. Bhotika, and S. Soatto, "Few-shot learning with embedded class models and shot-free meta training," in *Proc. IEEE/CVF Int. Conf. Comput. Vis.*, 2019, pp. 331–339.
[69] A. Antoniou, H. Edwards, and A. Storkey, "How to train your MAML," in *Proc. Int. Conf. Learn. Representations*, 2018, pp. 1–11.
[70] A. A. Rusu *et al.*, "Meta-learning with latent embedding optimization," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–17.
[71] A. Nichol, J. Achiam, and J. Schulman, "On first-order meta-learning algorithms," 2018, *arXiv:1803.02999*.
[72] M. A. Jamal, G.-J. Qi, and M. Shah, "Task agnostic meta-learning for few-shot learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 11 711–11 719.
[73] T. Munkhdalai, X. Yuan, S. Mehri, and A. Trischler, "Rapid adaptation with conditionally shifted neurons," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 3661–3670.
[74] B. N. Oreshkin, P. R. López, and A. Lacoste, "Tadam: Task dependent adaptive metric for improved few-shot learning," in *Proc. Neural Informat. Process. Syst.*, 2018, pp. 721–731.
[75] K. Lee, S. Maji, A. Ravichandran, and S. Soatto, "Meta-learning with differentiable convex optimization," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 10 649–10 657.
[76] R. Zhang, T. Che, Z. Ghahramani, Y. Bengio, and Y. Song, "MetaGAN: An adversarial approach to few-shot learning," in *Proc. Neural Informat. Process. Syst.*, 2018, pp. 2371–2380.
[77] X. Jiang, M. Havaei, F. Varno, G. Chartrand, N. Chapados, and S. Matwin, "Learning to learn with conditional class dependencies," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–11.
[78] E. Schwartz *et al.*, "Delta-encoder: An effective sample synthesis method for few-shot object recognition," in *Proc. Neural Informat. Process. Syst.*, 2018, pp. 2850–2860.
[79] K. Li, Y. Zhang, K. Li, and Y. Fu, "Adversarial feature hallucination networks for few-shot learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2020, pp. 13 467–13 476.
[80] P. Mangla, M. Singh, A. Sinha, N. Kumari, V. N. Balasubramanian, and B. Krishnamurthy, "Charting the right manifold: Manifold mixup for few-shot learning," in *Proc. IEEE Winter Conf. Appl. Comput. Vis.*, 2020, pp. 2207–2216.
[81] F. Zhou, B. Wu, and Z. Li, "Deep meta-learning: Learning to learn in the concept space," 2018, *arXiv:1802.03596*.
[82] A. Afrasiyabi, J.-F. Lalonde, and C. Gagné, "Associative alignment for few-shot image classification," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 18–35.
[83] Y. Lifchitz, Y. Avrithis, and S. Picard, "Local propagation for few-shot learning," in *Proc. Int. Conf. Pattern Recognit.*, 2020, pp. 10 457–10 464.
[84] Z. Chen, Y. Fu, Y. Zhang, Y.-G. Jiang, X. Xue, and L. Sigal, "Multi-level semantic feature augmentation for one-shot learning," *IEEE Trans. Image Process.*, vol. 28, no. 9, pp. 4594–4605, Sep. 2019.
[85] S. Tang, D. Chen, L. Bai, K. Liu, Y. Ge, and W. Ouyang, "Mutual CRF-GNN for few-shot learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 2329–2339.
[86] Y. Tian, Y. Wang, D. Krishnan, J. Tenenbaum, and P. Isola, "Rethinking few-shot image classification: A good embedding is all you need?," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 266–282.
[87] M. N. Rizve, S. Khan, F. S. Khan, and M. Shah, "Exploring complementary strengths of invariant and equivariant representations for few-shot learning," in *Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit.*, 2021, pp. 10 836–10 846.
[88] J. Kim, H. Kim, and G. Kim, "Model-agnostic boundary-adversarial sampling for test-time generalization in few-shot learning," in *Proc. Eur. Conf. Comput. Vis.*, 2020, pp. 599–617.
[89] L. Collins, A. Mokhtari, and S. Shakkottai, "Task-robust model-agnostic meta-learning," in *Proc. Neural Informat. Process. Syst.*, 2020, pp. 18 860–18 871.
[90] M. Garnelo *et al.*, "Conditional neural processes," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1690–1699.
[91] E. Todorov, T. Erez, and Y. Tassa, "MuJoCo: A physics engine for model-based control," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2012, pp. 5026–5033.
[92] J. Kruskal, "Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis," *Psychometrika*, vol. 29, pp. 1–27, 1964.
[93] A. D. Gotmare, N. Keskar, C. Xiong, and R. Socher, "A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation," in *Proc. Int. Conf. Learn. Representations*, 2019, pp. 1–16.

Zhi Gao (Member, IEEE) received the BS degree in computer science from the Beijing Institute of Technology (BIT), Beijing, China, in 2017. Currently, he is working toward the PhD degree with the Beijing Laboratory of Intelligent Information Technology, BIT, Beijing, China. His research interests include pattern recognition, machine learning, computer vision, and Riemannian Geometry.

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

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

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