Learning a Gradient-free Riemannian Optimizer on Tangent Spaces

Xiaomeng Fan,[1*] Zhi Gao,[1*] Yuwei Wu,[1†] Yunde Jia,[1] Mehrtash Harandi[2]

[1] Beijing Laboratory of Intelligent Information Technology
School of Computer Science, Beijing Institute of Technology, Beijing, China
[2] Department of Electrical and Computer Systems Eng., Monash University, and Data61, Australia
{fanxiaomeng,gaozhi_2017,wuyuwei,jiayunde}@bit.edu.cn, mehrtash.harandi@monash.edu

Abstract

A principal way of addressing constrained optimization problems is to model them as problems on Riemannian manifolds. Recently, Riemannian meta-optimization provides a promising way for solving constrained optimization problems by learning optimizers on Riemannian manifolds in a data-driven fashion, making it possible to design task-specific constrained optimizers. A close look at the Riemannian meta-optimization reveals that learning optimizers on Riemannian manifolds needs to differentiate through the nonlinear Riemannian optimization, which is complex and computationally expensive. In this paper, we propose a simple yet efficient Riemannian meta-optimization method that learns to optimize on tangent spaces of manifolds. In doing so, we present a gradient-free optimizer on tangent spaces, which takes parameters of the model along with the training data as inputs, and generates the updated parameters directly. As a result, the constrained optimization is transformed from Riemannian manifolds to tangent spaces where complex Riemannian operations (*e.g.*, retraction operations) are removed from the optimizer, and learning the optimizer does not need to differentiate through the Riemannian optimization. We empirically show that our method brings efficient learning of the optimizer, while enjoying a good optimization trajectory in a data-driven manner.

Introduction

We study the problem of Riemannian optimization (also known as optimization with manifold constraints), where the goal is to minimize an objective function in the form:

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

Here, $\boldsymbol{X}$ is the parameter of the interest that should comply with the geometry of the Riemannian manifold $\mathcal{M}$, and $\boldsymbol{y}_i$ denotes the training data. Such Riemannian optimization problems are pervasive in the machine learning community with a variety of applications. For example, principal component analysis (PCA) is modeled as an optimization problem on Grassmann manifolds (Yuan and Lamperski 2019), and similarity learning is solved by searching solutions on symmetric positive definite (SPD) manifolds (Gao et al. 2020a; Karlinsky et al. 2019). Due to the nonlinear search spaces of Riemannian manifolds, solving the Riemannian optimization problems is challenging, and the optimizer should preserve the enforced manifold constraints.

Conventional optimization algorithms designed in Euclidean spaces (*e.g.*, stochastic gradient descent) cannot be directly applied to the Riemannian setting, as their updating schemes will not preserve the manifold constraints. The gradient-based Riemannian optimization algorithms make use of Riemannian operations, such as retraction and orthogonal projection, to ensure that the algorithms remain faithful to the geometry of manifolds. State-of-the-art algorithms in this regard include Riemannian stochastic gradient descent (Bonnabel 2013), Riemannian variance reduction algorithms (Kasai, Sato, and Mishra 2018; Sato, Kasai, and Mishra 2019; Zhang, Reddi, and Sra 2016; Zhang, Zhang, and Sra 2018), and Riemannian adaptive optimization algorithms (Kumar, Mhammedi, and Harandi 2018; Kasai, Jawanpuria, and Mishra 2019) to name a few.

Recently, Riemannian meta-optimization provides a promising way to solve the Riemannian optimization problems (Gao et al. 2020b). Riemannian meta-optimization leverages the meta-learning technique to learn to optimize on manifolds in a data-driven fashion. Therein, learnable Riemannian optimizers that preserve the manifold constraints are built, and the method trains the optimizer to perform gradient-based Riemannian optimization on the manifolds. This method not only reduces human involvements of designing optimizers by hand but also achieves promising performances across various tasks. Despite the success and as will be shown shortly, the process of training the optimizer on manifolds is complex and computationally expensive, as it requires to differentiate through the complex gradient-based Riemannian optimization.

Similar to many meta-learning algorithms (Finn, Abbeel, and Levine 2017; Andrychowicz et al. 2016; Baik, Hong, and Lee 2020; Li, Wang, and Yu 2020), Riemannian meta-optimization is modeled as a bi-level optimization process (*i.e.*, using an inner and an outer loop). In the inner loop, the optimizer is utilized to iteratively update Riemannian parameters of the model. In the outer loop, the optimizer is trained by minimizing the loss of the model, where we

*Xiaomeng Fan and Zhi Gao are co-first authors.
†Corresponding author:Yuwei Wu

need to unroll and differentiate through the optimization trajectory in the inner loop to compute the meta-gradient with respect to parameters of the optimizer. To preserve the nonlinear Riemannian geometry in each optimization step, the learnable Riemannian optimizer is equipped with Riemannian operations (*e.g.*, retraction operations) that usually contain nonlinear matrix functions, such as matrix power, matrix inversion, eigenvalue decomposition, and singular decomposition. Differentiating through such matrix functions inevitably leads to heavy computational loads. It is thus desirable that a scheme efficiently learning the Riemannian optimizers should be established.

To this end, we propose a simple yet efficient Riemannian meta-optimization method that learns a gradient-free optimizer on tangent spaces of manifolds. As the tangent spaces are Euclidean spaces, finding the evolution of optimization on tangent spaces can utilize optimization schemes in linear spaces, not requiring the iterative application of the retraction operations. The gradient-free optimizer takes optimization parameters (mapped to tangent spaces) and training data as inputs, and directly generates updated parameters, which frees up the algorithm from computing gradients with respect to the parameters of the model. In this way, we transform the challenging optimization problems from manifolds to linear spaces, and thus learning the optimizer does not need to differentiate through the complex Riemannian optimization. Besides, by sidestepping the retraction and gradient computation in the inner loop, our method avoids a exploding gradient issue that caused by computing products of Hessian and gradients of the retraction (as required in learning the optimizer on manifolds), leading to computational efficiencies and stabilities. We empirically show that our gradient-free optimizer can be trained efficiently and learn a good optimization trajectory in a data-driven manner. The code is available at https://github.com/XiaomengFanmcislab/Learning-a-Gradient-free-Riemannian-Optimizer-on-Tangent-Spaces

In summary, our contributions are two-fold. (1) We propose a simple yet efficient Riemannian meta-optimization method that learns the optimizer on tangent spaces. Compared with learning the optimizer on manifolds, our method is efficient and computationally cheaper. (2) We present a learnable gradient-free optimizer, which does not require to compute gradients of parameters of the model, significantly reducing computation cost.

Preliminaries

Riemannian Manifold

A manifold $\mathcal{M}$ is a locally Euclidean space and can be understood as a generalization of the notion of surface to higher dimensions (Absil, Mahony, and Sepulchre 2009). The tangent space to $\mathcal{M}$ at $\boldsymbol{X}$ is denoted by $T_{\boldsymbol{X}}\mathcal{M}$. $T_{\boldsymbol{X}}\mathcal{M}$ is a Euclidean space that contains all tangent vectors to $\mathcal{M}$ at $\boldsymbol{X}$. Commonly encountered Riemannian manifolds include the Grassmann manifold (Huang, Wu, and Van Gool 2018), the Stiefel manifold (Huang et al. 2018), and the SPD manifold (Dong et al. 2017), which has the subspace constraint, the orthogonal column constraint, and the SPD matrix constraint, respectively.

Gradient-based Riemannian Optimization

Gradient-based optimization is the workhorse of machine learning algorithms. In general, the Riemannian gradient-descent update is given by

$$\boldsymbol{X}^{(t+1)} = \Gamma_{\boldsymbol{X}^{(t)}}\Big(-\xi^{(t)}\pi_{\boldsymbol{X}^{(t)}}(\nabla_{\boldsymbol{X}}^{(t)})\Big), \tag{2}$$

where $\boldsymbol{X}^{(t)}$ is the estimated parameter at time t, $\nabla_{\boldsymbol{X}}^{(t)}$ is the Euclidean gradient with respect to $\boldsymbol{X}^{(t)}$ in the ambient space calculated at time t, and $\xi^{(t)}$ is the stepsize. $\pi_{\boldsymbol{X}^{(t)}}$ and $\Gamma_{\boldsymbol{X}^{(t)}}$ are two important operations in the Riemannian optimization, named the orthogonal projection and the retraction, respectively. The orthogonal projection $\pi_{\boldsymbol{X}}(\nabla_{\boldsymbol{X}}) : \mathbb{R}^n \rightarrow T_{\boldsymbol{X}}(\mathcal{M})$ transforms $\nabla_{\boldsymbol{X}}$ into the tangent space $T_{\boldsymbol{X}}\mathcal{M}$. The retraction operation $\Gamma_{\boldsymbol{X}}(\boldsymbol{P}) : T_{\boldsymbol{X}}\mathcal{M} \rightarrow \mathcal{M}, \boldsymbol{P} \in T_{\boldsymbol{X}}\mathcal{M}$ maps vectors from the tangent space into the manifold with a rigidty condition (Absil, Mahony, and Sepulchre 2009).

Gradient-free Optimizer

Problem Definition

Minimizing the empirical risk in Riemannian optimization problems most often takes the form:

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

where $f(\cdot) : \mathcal{M} \times \mathcal{Y} \rightarrow \mathbb{R}$ is the loss function, $\boldsymbol{X} \in \mathcal{M}$ is the parameters of interest on the Riemannian manifold $\mathcal{M}$, and $\{\boldsymbol{y}_i\}_{i=1}^n$ are the training data. To solve Eq. (3), Gao *et al*. proposed to learn a Riemannian optimizer to search solutions along the manifold (Gao et al. 2020b). Although it reduces a large number of human involvements to design the optimizer by hand, this method is prone to be complex and computationally expensive due to the need of differentiating through complex Riemannian optimization.

In this paper, we formulate the Riemannian optimization problem as searching solutions along the tangent space,

$$\min_{\boldsymbol{P}\in T_{\boldsymbol{X}}\mathcal{M}} \mathcal{L}(\boldsymbol{P}) \triangleq \frac{1}{n}\sum_{i}^{n} f(\Gamma_{\boldsymbol{X}}(\boldsymbol{P}), \boldsymbol{y}_i), \tag{4}$$

where $\boldsymbol{P}$ is the parameter on a tangent space $T_{\boldsymbol{X}}\mathcal{M}$. We propose a simple yet efficient Riemannian meta-optimization method that learns the optimizer on the tangent space to solve Eq. (4). In the following sections, we will detail our method.

Learning a Gradient-free Optimizer

In our method, we update the parameter $\boldsymbol{P}^{(t)}$ on the tangent space $T_{\boldsymbol{X}}\mathcal{M}$ iteratively as:

$$\boldsymbol{P}^{(t+1)} = \boldsymbol{P}^{(t)} - \boldsymbol{\eta}^{(t)}, \tag{5}$$

where $\boldsymbol{\eta}^{(t)} \in T_{\boldsymbol{X}}\mathcal{M}$ is an update vector on the tangent space. After T optimization steps, the parameter on the tangent space is

$$\boldsymbol{P}^{(t+T)} = \boldsymbol{P}^{(t)} - \sum_{i=0}^{T-1} \boldsymbol{\eta}^{(t+i)}, \tag{6}$$

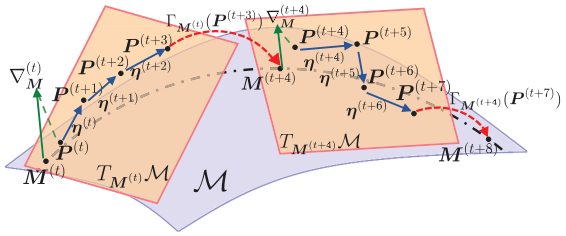


Figure 1: The illustration of optimization on tangent spaces.

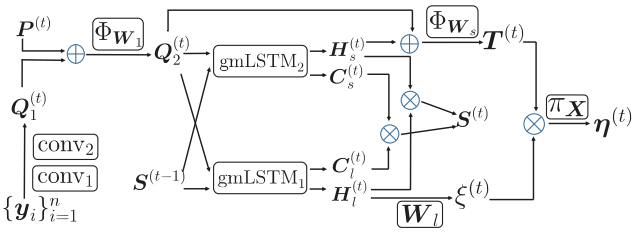


Figure 2: The architecture of our gradient-free optimizer.

and we can utilize the retraction operation to obtain the updated Riemannian parameter $\boldsymbol{X}^{(t+T+1)}$,

$$\boldsymbol{X}^{(t+T+1)} = \Gamma_{\boldsymbol{X}}(\boldsymbol{P}^{(t+T)}). \tag{7}$$

In this way, we only use one retraction operation in updating T steps. Note that, the optimization may contain multiple tangent spaces, as shown in Figure 1.

We utilize the meta-learning technique to learn the optimizer to perform such optimization on tangent spaces. Conventional meta-optimization methods usually train recurrent networks to perform optimization (Andrychowicz et al. 2016; Ravi and Larochelle 2017). This is done by taking the gradients of parameters as inputs to the recurrent networks. Unfortunately, existing solutions cannot be directly applied to our setting, as the gradients with respect to the parameter on tangent spaces is complex and requires derivatives of the retraction. As alluded to earlier, opting to include the derivatives of the retraction may lead to complex computation.

To solve this problem, we introduce a gradient-free optimizer that regards the optimization as a forward process. We utilize the neural network to learn a function $h_\theta(\cdot)$ that takes both the parameter $\boldsymbol{P}^{(t)}$ and training data $\{\boldsymbol{y}_i\}_{i=1}^n$ as inputs, and directly generates $\boldsymbol{\eta}^{(t)}$, that is,

$$\boldsymbol{\eta}^{(t)} = h_\theta\Big(\boldsymbol{P}^{(t)}, \{\boldsymbol{y}_i\}_{i=1}^n, \boldsymbol{S}^{(t-1)}\Big), \tag{8}$$

where θ is the learnable parameter of our optimizer, and $\boldsymbol{S}^{(t-1)}$ is the optimization state at time $t-1$. Our goal is to learn the parameter θ to perform good Riemannian optimization.

Generalized Matrix LSTM (gmLSTM) Meta-optimization methods show that the optimizer parameterized by LSTM are capable of learning to perform optimization (Andrychowicz et al. 2016; Ravi and Larochelle 2017). Considering inputs of our optimizer contain parameters on tangent spaces with the matrix structure, conventional LSTM cannot be directly applied to our setting, which may inevitably destroy the matrix structure. In this paper and inspired by the work of (Gao et al. 2020b), we further introduce a generalized matrix LSTM (gmLSTM) model for various Riemannian manifolds based on a transformation operation. The transformation operation $\Phi_{\boldsymbol{W}}(\cdot)$ projects inputs to targets that lies in the same domain as the inputs, and $\boldsymbol{W}$ is the learnable parameter. For example, on SPD manifolds, the input $\boldsymbol{P} \in \mathbb{R}^{d\times d}$ is a symmetric matrix, and the transformation is designed as $\Phi_{\boldsymbol{W}}(\boldsymbol{P}) = \boldsymbol{W}^\top \boldsymbol{P}\boldsymbol{W}$, where $\boldsymbol{W} \in \mathbb{R}^{d\times d}$ preserves the symmetric property. On the Stiefel and Grassmann manifolds, the input is a matrices $\boldsymbol{P} \in \mathbb{R}^{d\times p}$, and the transformation is formulated as $\Phi_{\boldsymbol{W}}(\boldsymbol{P}) = \boldsymbol{W}^\top \boldsymbol{P}$, where $\boldsymbol{W} \in \mathbb{R}^{d\times d}$. For gmLSTM, we replace the vector multiplication in conventional LSTM with the transformation operation, and the gmLSTM model is denoted as

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

where $\boldsymbol{S}^{(t-1)} = [\boldsymbol{C}^{(t-1)}, \boldsymbol{H}^{(t-1)}]$ denotes the state of gmLSTM, containing the memory cell $\boldsymbol{C}^{(t)}$ and the output $\boldsymbol{H}^{(t)}$.

Optimizer Designing We first utilize two convolutional layers to process the training data,

$$\boldsymbol{Q}_1^{(t)} = \Big(\mathrm{conv}_2\Big(\mathrm{conv}_1(\{\boldsymbol{y}_i\}_{i=1}^n)\Big)\Big), \tag{10}$$

where conv_1 and conv_2 denote two convolutional layers with the ReLU activation function. Then, we utilize a transformation operation $\Phi_{\boldsymbol{W}_1}$ to fuse $\boldsymbol{Q}_1^{(t)}$ and $\boldsymbol{P}^{(t)}$, and extract optimization information of $\boldsymbol{P}^{(t)}$ on the training data $\{\boldsymbol{y}_i\}_{i=1}^n$, that is

$$\boldsymbol{Q}_2^{(t)} = \Phi_{\boldsymbol{W}_1}(\boldsymbol{Q}_1^{(t)} + \boldsymbol{P}^{(t)}). \tag{11}$$

Based on the optimization information $\boldsymbol{Q}_2^{(t)}$ and the optimization state $\boldsymbol{S}^{(t-1)}$, we utilize two gmLSTM models to calculate a stepsize $\xi^{(t)}$ and a search direction $\boldsymbol{T}^{(t)}$ on the tangent space. The step-size $\xi^{(t)}$ is calculated by

$$\begin{cases} \boldsymbol{H}_1^{(t)}, \boldsymbol{C}_1^{(t)} = \mathrm{gmLSTM}_1\left(\boldsymbol{Q}_2^{(t)}, \boldsymbol{S}^{(t-1)}\right), \\ \xi^{(t)} = \boldsymbol{w}_l^\top \boldsymbol{H}_1^{(t)} \end{cases}, \tag{12}$$

where gmLSTM_1 is a gmLSTM model, and $\boldsymbol{w}_l$ is a learnable parameter. Suppose the tangent space is $T_{\boldsymbol{X}}\mathcal{M}$, we compute the search direction $\boldsymbol{T}^{(t)}$ according to

$$\begin{cases} \boldsymbol{H}_2^{(t)}, \boldsymbol{C}_2^{(t)} = \mathrm{gmLSTM}_2\left(\boldsymbol{Q}_2^{(t)}, \boldsymbol{S}^{(t-1)}\right), \\ \boldsymbol{T}^{(t)} = \pi_{\boldsymbol{X}}\left(\Phi_{\boldsymbol{W}_s}(\boldsymbol{H}_2^{(t)})\right) \end{cases}, \tag{13}$$

where gmLSTM_2 is the other gmLSTM model, and $\Phi_{\boldsymbol{W}_s}$ is a transformation operation. Finally, the update vector $\boldsymbol{\eta}^{(t)}$ on the tangent space is computed by

$$\boldsymbol{\eta}^{(t)} = \xi^{(t)}\boldsymbol{T}^{(t)}, \tag{14}$$

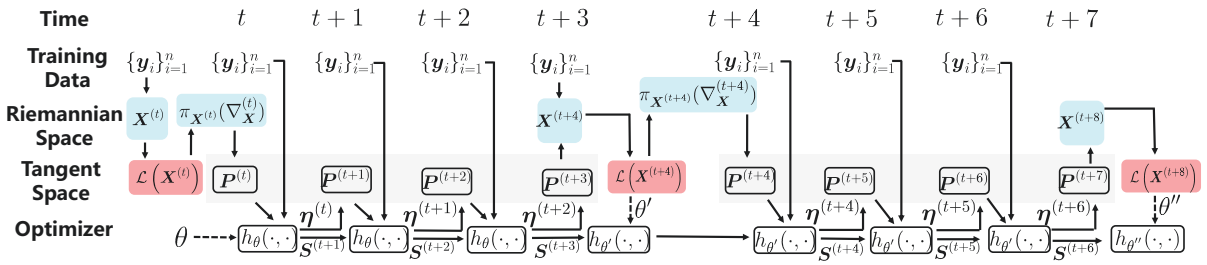


Figure 3: The computational graph of training our optimizer.

and the optimization state of the optimizer is updated by

$$\boldsymbol{S}^{(t)} = [\boldsymbol{H}_1^{(t)} \otimes \boldsymbol{H}_2^{(t)}, \boldsymbol{C}_1^{(t)} \otimes \boldsymbol{C}_2^{(t)}]. \tag{15}$$

The architecture of our optimizer is shown in Figure 2. In total, the learnable parameter θ of our optimizer contains parameters of the two convolutional layers conv_1 and conv_1, parameters $\boldsymbol{W}_1$ and $\boldsymbol{W}_s$ of two transformation operations $\Phi_{\boldsymbol{W}_1}$ and $\Phi_{\boldsymbol{W}_s}$, parameters of two gmLSTM models gmLSTM_1 and gmLSTM_2, and the parameter $\boldsymbol{w}_l$.

Optimizer Training Similar to existing meta-optimization methods (Andrychowicz et al. 2016; Ravi and Larochelle 2017), we utilize two optimization loops to train our optimizer, that is, using an inner loop and an outer loop. In the inner loop, given the tangent space $T_{\boldsymbol{X}^{(t)}}\mathcal{M}$ and the initial parameter $\boldsymbol{P}^{(t)} \in T_{\boldsymbol{X}^{(t)}}\mathcal{M}$, our optimizer updates $\boldsymbol{P}^{(t)}$ in T steps,

$$\boldsymbol{P}^{(t+T)} = \boldsymbol{P}^{(t)} - \sum_{l=0}^{T-1} h_\theta\Big(\boldsymbol{P}^{(t+l)}, \{\boldsymbol{y}_{i'}\}_{i'=1}^n, \boldsymbol{S}^{(t-1+l)}\Big). \tag{16}$$

In the outer loop, we utilize the retraction operation to project $\boldsymbol{P}^{(t+T)}$ to the manifold, $\boldsymbol{X}^{(t+T+1)} = \Gamma_{\boldsymbol{X}^{(t)}}(\boldsymbol{P}^{(t+T)})$, and train our optimizer by minimizing the following meta-objective

$$\min_\theta \mathcal{J}(\theta) \triangleq \frac{1}{m}\sum_{j=1}^m \mathcal{L}(\boldsymbol{X}_j^{(t+T+1)}) = \frac{1}{mn}\sum_{j=1}^m\sum_{i=1}^n f\Bigg(\Gamma_{\boldsymbol{X}_j^{(t)}}\Big(\boldsymbol{P}_j^{(t)} - \sum_{l=0}^{T-1} h_\theta\big(\boldsymbol{P}_j^{(t+l)}, \{\boldsymbol{y}_{i'}\}_{i'=1}^n, \boldsymbol{S}_j^{(t-1+l)}\big)\Big), \boldsymbol{y}_i\Bigg), \tag{17}$$

where m is the batchsize in the outer loop (*i.e.*, the meta-objective involves m individual Riemannian parameters of the model). Then, we compute a Riemannian gradient $\nabla_{\boldsymbol{X}}^{(t+T+1)}$ of $\mathcal{L}(\boldsymbol{X}_j^{(t+T+1)})$ to identify a new tangent space $T_{\boldsymbol{X}^{(t+T+1)}}\mathcal{M}$ and the parameter $\boldsymbol{P}^{(t+T+1)} = \pi_{\boldsymbol{X}^{(t+T+1)}}(\nabla_{\boldsymbol{X}}^{(t+T+1)})$ on the new tangent space. Training our optimizer is shown in Figure 3.

To update the optimizer, the meta-gradient with respect to θ of the meta-objective in Eq. (17) is computed as

$$\frac{d\mathcal{J}}{d\theta} = \Big\langle \frac{d\mathcal{J}}{d\boldsymbol{X}^{(t+T+1)}}, \frac{\partial \boldsymbol{X}^{(t+T+1)}}{\partial \boldsymbol{P}^{(t+T)}} \cdot \Bigg(- \frac{\partial \boldsymbol{\eta}^{(t+T-1)}}{\partial \theta} - \sum_{k=2}^{t} \Big(\prod_{l=1}^{k-1}\Big(1 - \frac{\partial \boldsymbol{\eta}^{(t+T-l)}}{\partial \boldsymbol{P}^{(t+T-l)}}\Big)\frac{\partial \boldsymbol{\eta}^{(t+T-k)}}{\partial \theta}\Big)\Bigg)\Big\rangle, \tag{18}$$

where the index j of different parameters on tangent spaces is ignored.

Comparison with Learning the Optimizer on Riemannian Manifolds. Existing Riemannian meta-optimization learns to optimize on Riemannian manifolds (Gao et al. 2020b), where its optimizer updates the Riemannian parameter by $\boldsymbol{X}^{(t+1)} = \Gamma_{\boldsymbol{X}^{(t)}}\Big(-g_\phi\Big(\nabla_{\boldsymbol{X}}^{(t)}, \boldsymbol{S}^{(t-1)}\Big)\Big)$. Compared with this optimization scheme, our optimizer does not require neither the retraction operation $\Gamma_{\boldsymbol{X}^{(t)}}$ nor gradient computation $\nabla_{\boldsymbol{X}}^{(t)}$ with respect to Riemannian parameters $\boldsymbol{X}^{(t)}$, reducing much computation cost.

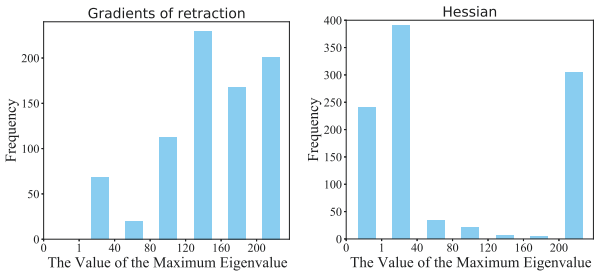
The optimizer g_ϕ of Gao *et al*. is learned by minimizing the following meta-objective

$$\begin{aligned}\min_\phi \mathcal{J}(\phi) &= \sum_{j=1}^m\sum_{t=a}^{a+T}\mathcal{L}\Big(\boldsymbol{X}_j^{(t+1)}\Big) \\ &= \sum_{j=1}^m\sum_{t=a}^{a+T}\sum_{i=1}^n f\Big(\Gamma_{\boldsymbol{X}_j^{(t)}}\Big(-g_\phi\big(\nabla_{\boldsymbol{X}_j}^{(t)}, \boldsymbol{S}_j^{(t-1)}\big)\Big), \boldsymbol{y}_i\Big).\end{aligned} \tag{19}$$

By denoting $\boldsymbol{Y}^{(t)} = -g_\phi\big(\nabla_{\boldsymbol{X}}^{(t)}, \boldsymbol{S}^{(t-1)}\big)$ and unrolling the optimization trajectory in the inner loop from $t = a$ to $t = a + T$, the meta-gradient with respect to ϕ is

$$\frac{d\mathcal{J}}{d\phi} = \sum_{t=a}^{a+T}\Big\langle \frac{d\mathcal{J}}{d\boldsymbol{X}^{(t)}}, \Bigg(\sum_{k=1}^{t}\Big(\frac{\partial \boldsymbol{X}^{(k)}}{\partial \boldsymbol{Y}^{(k-1)}}\cdot\frac{\partial \boldsymbol{Y}^{(k-1)}}{\partial \phi}\prod_{l=k+1}^{t}\Big(\frac{\partial \boldsymbol{X}^{(l)}}{\partial \boldsymbol{X}^{(l-1)}} + \frac{\partial \boldsymbol{X}^{(l)}}{\partial \boldsymbol{Y}^{(l-1)}}\cdot\frac{\partial \boldsymbol{Y}^{(l-1)}}{\partial \nabla \boldsymbol{X}^{(l-1)}}\cdot\nabla^2\boldsymbol{X}^{(l-1)}\Big)\Big)\Bigg)\Big\rangle, \tag{20}$$

where the index j of different Riemannian parameters is also ignored. Comparing Eq. (18) with Eq. (20), Gao *et al.* have to differentiate through the complex Riemannian optimization, including computing derivatives $\frac{\partial \boldsymbol{X}^{(l)}}{\partial \boldsymbol{X}^{(l-1)}}$ and $\frac{\partial \boldsymbol{X}^{(l)}}{\partial \boldsymbol{Y}^{(l-1)}}$ of retraction operations, and computing Hessian matrices $\nabla^2\boldsymbol{X}^{(l-1)}$, over all optimization steps. The retraction operations contain complex matrix functions, such as matrix



(a) (b)

Figure 4: Maximum eigenvalues of derivatives of retraction operations and Hessian matrices in a texture clustering task on SPD manifolds (Kylberg 2011). We compute the derivatives of retraction operations and Hessian matrices, and calculate the maximum eigenvalue of each matrix. We can find that, all eigenvalues in (a) and more than three quarters eigenvalues in (b) are greater than one.

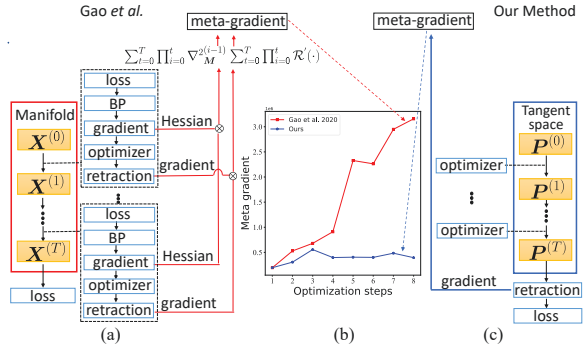


Figure 5: Comparison between Gao *et al.* (Gao et al. 2020b) and our method. (a) Gao *et al.* learn optimizers on manifolds, and its meta-gradient involves products of Hessian matrices and derivatives of retraction operations, where $\otimes$ denotes the matrix multiplication and $\mathcal{R}'$ denotes derivatives of retraction operations. (b) Gao *et al.* suffers from the exploding gradient issue, while our method does not. (c) We learn optimizers on tangent spaces, and our meta-gradient is stabler.

power, matrix inversion, eigenvalue decomposition, and singular decomposition, making calculating derivatives of retraction operations and Hessian matrices computationally expensive. In contrast, training our optimizer only requires to differentiate through the forward process of our optimizer (a simple network) and compute the derivative $\frac{\partial \boldsymbol{X}^{(t+T+1)}}{\partial \boldsymbol{P}^{(t+T)}}$ of one retraction operation in the outer loop, which is simpler and computationally cheaper.

Remark From Eq. (20), we can find the meta-gradient of Gao *et al.* has products of the Hessian $\nabla^2 \boldsymbol{X}^{(l-1)}$ and products of derivatives of the retraction operations $\frac{\partial \boldsymbol{X}^{(l)}}{\partial \boldsymbol{X}^{(l-1)}}$ and $\frac{\partial \boldsymbol{X}^{(l)}}{\partial \boldsymbol{Y}^{(l-1)}}$. It is widely known (*e.g.*, (Pascanu, Mikolov, and Bengio 2013; Metz et al. 2019)) that the meta-gradient may grow exponentially with the increase of the number T of optimization steps in the inner loop, if the maximun eigenvalues or singular values of the Hessian matrices or derivatives of the retraction operations are larger than one. Actually, as the nonlinearity of the Riemannian optimization, the maximun eigenvalues or singular values of both them are usually larger than one. We provide an example of a clustering task on SPD manifolds, as shown in Figure 4. Thus, training the optimizer in Gao *et al.* has the exploding gradient issue, as shown in Figure 5 (a) and (b). In contrast, in Eq. (18), the meta-gradient of our method does not involve products of Hessian and products of derivatives of retraction operations, since our optimizer sidesteps the retraction operations and gradients with respect to parameters of the model in the inner loop. Thus, our method refrains the exploding gradient issue, as shown in Figure 5 (b) and (c).

Related Work

Riemannian optimization problems can be effectively solved by gradient-based Riemannian optimization algorithms. Bonnabel (Bonnabel 2013) proposed the first Riemannian stochastic gradient descent algorithm. After that, many efforts have been made to design powerful Riemannian optimizers by hand. For example, several algorithms generalized accelerated or momentum techniques from Euclidean spaces to Riemannian manifolds (Liu et al. 2017; Zhang and Sra 2018; Kumar, Mhammedi, and Harandi 2018), some algorithms studied reducing the variance of stochastic Riemannian gradients (Zhang, Reddi, and Sra 2016; Sato, Kasai, and Mishra 2019; Kasai, Sato, and Mishra 2018; Zhang, Zhang, and Sra 2018), and some algorithms developed Riemannian adaptive optimization (Kumar, Mhammedi, and Harandi 2018; Kasai, Jawanpuria, and Mishra 2019; Bécigneul and Ganea 2019). Besides, two works (Lezcano-Casado and Martınez-Rubio 2019; Casado 2019) transformed optimization problems from Stiefel manifolds into Euclidean spaces. Compared with the two works, our method can be applied to various manifolds, and our optimizer is gradient-free and learned via a data-driven manner, which is computational cheap, not requiring gradients with respect to parameters.

Recently, Gao *et al*. (Gao et al. 2020b) proposed a Riemannian meta-optimization method that utilizes the meta-learning technique to learn the optimizer on SPD manifolds, reducing human involvements to design the optimizer by hand. Therein a matrix LSTM model is introduced to process Riemannian gradients. To be specific, the matrix LSTM is trained to produce search directions using the gradient information for the SPD manifold. Compared with it, our method can be applied to various manifolds and is a gradient-free optimizer. As alluded to before, our method works on tangent spaces to avoid the retraction operations and computing the gradients of the Riemannian parameters in the inner loop. In this case, our method reduces much computational cost of learning the optimizer and sidesteps the exploding gradient issue of Gao *et al*.

Experiments

In this section, we evaluate our method from three perspectives, that is, the convergence, accuracy and efficiency. We compare our optimizer with state-of-the-art Rieman-

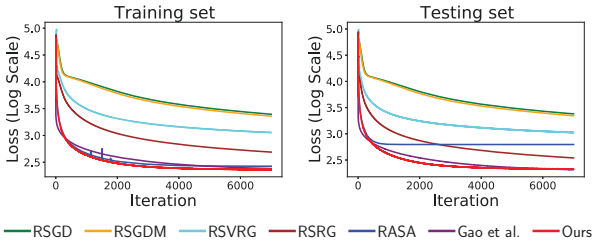


Figure 6: Plots for the PCA task (in the log scale).

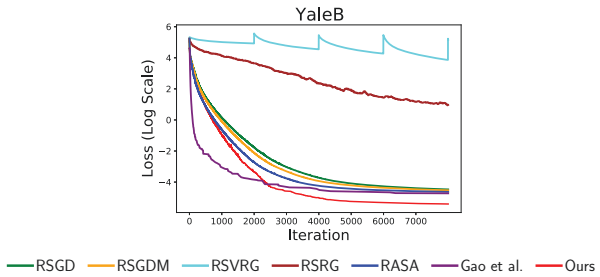


Figure 7: Plots for the face recognition task (in the log scale).

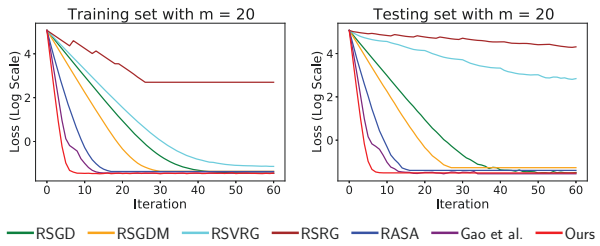


Figure 8: Plots for the clustering task (in the log scale).

nian optimizers: RSGD (Bonnabel 2013), RSGDM (Kumar, Mhammedi, and Harandi 2018), RSVRG (Zhang, Reddi, and Sra 2016), RSRG (Kasai, Sato, and Mishra 2018), RASA (Kasai, Jawanpuria, and Mishra 2019), and Gao *et al.* (Gao et al. 2020b). RSGD, RSGDM, RSVRG, RSRG, and RASA are hand-designed optimizers, and Gao *et al.* learn optimizers on manifolds. We replace the original mLSTM model in Gao *et al.* with the gmLSTM model to evaluate its performance on various manifolds. Following (Gao et al. 2020b), hyperparameters of all optimizers are tuned to achieve the best performance.

Convergence Analysis

In this section, we analyze the convergence of the learned optimizer. We conduct experiments on three tasks: PCA, face recognition, and clustering, modeled on the Grassmann manifold $\mathcal{G}(p, d)$, Stiefel manifold $\mathrm{St}(p, d)$, and SPD manifold S_{++}^{d}, respectively.

PCA on the Grassmann Manifold Principal component analysis aims to learn an orthogonal projection $\boldsymbol{X} \in \mathcal{G}(p, d)$ that minimizes the sum of squared residual errors between projected results and the original data.

We utilize the MNIST dataset to evaluate our optimizer. We resize images to 784-dimensional vectors and aim to learn 128-dimensional representations. We train the optimizer on the training set and evaluate its performance on both the training and test sets. Experimental results are shown in Figure 6. Compared with hand-designed optimizers, our optimizer has a fast convergence and lower optima, no matter on the training or test sets. Gao *et al.* also learn a Riemannian optimizer. Compared with it, our optimizer achieves comparable performance, showing that a gradient-free optimizer can also be trained to achieve good performance in a data-driven manner.

Face Recognition on the Stiefel Manifold The orthogonality constraint has been widely used to extract discriminative features, and parameters with the orthogonality constraint are forced on Stiefel manifolds. Here, we conduct the experiment on the face recognition task, where a linear classifier with the orthogonality constraint is used.

In this experiment, we use the YaleB dataset (Lee, Ho, and Kriegman 2005) to evaluate our optimizer. We train the optimizer on the training set, and its performance is shown in Figure 7. We plot the loss curve in the log scale. We can find that, the learned optimizers of Gao *et al.* and our method have faster convergences and better optimas than hand-designed optimizers, as the learned optimizers can exploit the underlying data distribution and learn a good optimization trajectory in a data-driven manner. RSRG and RSVRG finally converge to loss values of -4.42 and -3.63, Gao *et al.* converges to the loss of -4.72, while our optimizer converges to the best optima, -5.41.

Clustering on the SPD Manifold We also conduct experiments on the clustering task of SPD representations. We evaluate our optimizer on the Kylberg texture dataset (Kylberg 2011). Following the work of (Kumar, Mhammedi, and Harandi 2018), we extract a 5×5 covariance descriptor to represent each image. We train the optimizer on the training set, and its performance on both the training and test sets is shown in Figure 8. The results show our optimizer again achieves the best performance.

Accuracy Analysis

We evaluate the performance of the solved classifier in the face recognition task and centers in the clustering task. In the face recognition task, we utilize the training set to solve the classifier and then compute the accuracy on the test set. In the clustering task, we regard the centers as category prototypes and compute distances between test samples and prototypes for classification. Results are shown in Table 1. Learnable optimizers achieve better performance, surpassing hand-designed optimizers, since they can find good optimas in a data-driven manner. Our method has the highest accuracy, which achieves 90.2% and 86.1% on the two tasks, 1.2% and 0.9% higher than Gao *et al*. This shows that our optimizer can arrive at a better optima.

Efficiency Analysis

Meta-gradients We measure norms of meta-gradients of our method and Gao *et al.* (Gao et al. 2020b) in the cluster-

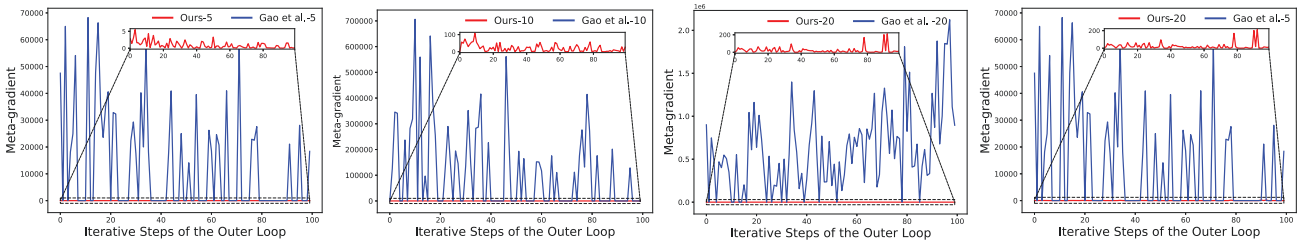


Figure 9: Meta-gradients of our method and Gao *et al.* (Gao et al. 2020b) in the clustering task.

Method	Face Recognition	Clustering
RSGD	79.4	85.1
RSGDM	79.6	85.0
RSVRG	79.6	80.3
RSRG	78.0	83.9
RASA	80.2	85.2
Gao *et al.*	89.0	85.2
Ours	**90.2**	**86.1**

Table 1: Accuracy (%) of the solved classifier and centers.

Training strategy	Mean	Variance
Ours-5	0.92	1.10
Ours-10	1.94×10^1	4.34×10^2
Ours-20	2.13×10^1	1.25×10^3
Gao *et al.*-5	1.27×10^4	3.47×10^8
Gao *et al.*-10	1.08×10^5	2.52×10^{10}
Gao *et al.*-20	6.09×10^5	2.28×10^{11}

Table 2: Mean and variance of meta-gradients in clustering.

ing task with different optimization steps T in the inner loop. Results are shown in Figure. 9, and means and variances of the meta-gradients are shown in Table 2, where 'Ours-5' denotes our method with the optimization steps in the inner loop as 5, and 'Gao *et al.*-5' means the method of (Gao et al. 2020b) with the optimization steps in the inner loop as 5, *i.e.*, $T = 5$. We can find that meta-gradients of Gao *et al.* change sharply and have large norms. With the increase of optimization steps in the inner loop, meta-gradients of Gao *et al.* grows rapidly, which confirms our analysis. They have to set a small number of optimization steps in the inner loop to alleviate this issue, but this leads to biased meta-gradients and training oscillations. In contrast, meta-gradients of our method are stabler and have much smaller norms, showing that our method can avoid the exploding gradient issue. Even when the step of the inner loop is $T = 20$, the mean of our meta-gradient is 2.13×10^1, much smaller than that 1.27×10^4 of Gao *et al.* at $T = 5$. Besides, the variance of our meta-gradient is much smaller than that of Gao *et al.*, showing that training our optimizer is stabler.

Training Time We measure the training time of our method and Gao *et al.* (Gao et al. 2020b) with the same number of optimization steps in the outer loop, which is set as 100. Results are shown in Table 3. In the Table, the method with a large number of optimization steps in the inner loop requires much training time, while it has unbiased gradients. Compared with Gao *et al.*, our method requires less training time, especially with a large number of optimization steps in the inner loop. When $T = 20$, our method requires 208.4, 272.8, and 1419.0 seconds on the three tasks, while Gao *et al.* requires 1876.6, 2508.6, and 11979.2 seconds, more time-consuming than our method. The reason is that, our optimizer is a forward process in linear spaces, not requiring Riemannian operations, and learning our optimizer does not need to differentiate through the complex Riemannian optimization. Thus, based on the analyses of training time and meta-gradients, training our optimizer is more efficient.

Training strategy	PCA	Face Recognition	Clustering
Ours-5	61.3	71.5	1365.6
Ours-10	110.8	135.9	1392.1
Ours-20	208.4	272.8	1419.0
Gao *et al.*-5	139.1	184.3	3000.5
Gao *et al.*-10	510.1	665.7	6058.8
Gao *et al.*-20	1876.6	2508.6	11979.2

Table 3: Training time (seconds) on the three tasks.

Conclusion

In this paper, we have presented a simple yet efficient method to learn optimizers for Riemannian optimization. Our optimization scheme that transforms the search space from nonlinear manifolds to linear tangent spaces can avoid complex retraction operations, being much easier to implement. Our gradient-free optimizer regards the optimization process as a forward function and does not need to compute gradients with respect to Riemannian parameters, reducing much computation cost. Besides, our method can avoid exploding gradient issue of learning the optimizer on Riemannian manifold by sidestepping the retraction operations and gradient computation in the inner loop. We empirically demonstrate that training our optimizer is more efficiently, and it has smaller and stabler meta-gradients. Meanwhile, experiments on three tasks show that our gradient-free optimizer learned in a data-driven manner can achieve state-of-the-art performance for Riemannian optimization.

Acknowledgments

This work was supported by the Natural Science Foundation of China (NSFC) under Grants No. 61773062 and No. 61702037.

References

Absil, P.-A.; Mahony, R.; and Sepulchre, R. 2009. *Optimization algorithms on matrix manifolds*. Princeton University Press.

Andrychowicz, M.; Denil, M.; Gomez, S.; Hoffman, M. W.; Pfau, D.; Schaul, T.; Shillingford, B.; and De Freitas, N. 2016. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems (NeurIPS)*, 3981–3989.

Baik, S.; Hong, S.; and Lee, K. M. 2020. Learning to Forget for Meta-Learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2376–2384.

Bécigneul, G.; and Ganea, O. 2019. Riemannian Adaptive Optimization Methods. In *International Conference on Learning Representations (ICLR)*.

Bonnabel, S. 2013. Stochastic Gradient Descent on Riemannian Manifolds. *IEEE Transactions on Automatic Control* 58(9): 2217–2229.

Casado, M. L. 2019. Trivializations for gradient-based optimization on manifolds. In *Advances in Neural Information Processing Systems (NeurIPS)*, 9154–9164.

Dong, Z.; Jia, S.; Zhang, C.; Pei, M.; and Wu, Y. 2017. Deep manifold learning of symmetric positive definite matrices with application to face recognition. In *AAAI Conference on Artificial Intelligence (AAAI)*, 4009–4015.

Finn, C.; Abbeel, P.; and Levine, S. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *International Conference on Machine Learning (ICLR)*, 1126–1135.

Gao, Z.; Wu, Y.; Harandi, M.; and Jia, Y. 2020a. A robust distance measure for similarity-based classification on the SPD manifold. *IEEE Transactions on Neural Networks and Learning Systems* 31(9): 3230–3244.

Gao, Z.; Wu, Y.; Jia, Y.; and Harandi, M. 2020b. Learning to Optimize on SPD Manifolds. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 7700–7709.

Huang, L.; Liu, X.; Lang, B.; Yu, A. W.; Wang, Y.; and Li, B. 2018. Orthogonal Weight Normalization: Solution to Optimization Over Multiple Dependent Stiefel Manifolds in Deep Neural Networks. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Huang, Z.; Wu, J.; and Van Gool, L. 2018. Building Deep Networks on Grassmann Manifolds. In *AAAI Conference on Artificial Intelligence (AAAI)*, 3279–3286.

Karlinsky, L.; Shtok, J.; Harary, S.; Schwartz, E.; Aides, A.; Feris, R.; Giryes, R.; and Bronstein, A. M. 2019. Repmet: Representative-based metric learning for classification and few-shot object detection. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 5197–5206.

Kasai, H.; Jawanpuria, P.; and Mishra, B. 2019. Riemannian adaptive stochastic gradient algorithms on matrix manifolds. In *International Conference on Machine Learning (ICML)*, 3262–3271.

Kasai, H.; Sato, H.; and Mishra, B. 2018. Riemannian Stochastic Recursive Gradient Algorithm. In *International Conference on Machine Learning (ICML)*, 2516–2524.

Kumar, Roy, S.; Mhammedi, Z.; and Harandi, M. 2018. Geometry aware constrained optimization techniques for deep learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 4460–4469.

Kylberg, G. 2011. *Kylberg Texture Dataset v. 1.0*.

Lee, K.-C.; Ho, J.; and Kriegman, D. J. 2005. Acquiring linear subspaces for face recognition under variable lighting. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 27(5): 684–698.

Lezcano-Casado, M.; and Martınez-Rubio, D. 2019. Cheap Orthogonal Constraints in Neural Networks: A Simple Parametrization of the Orthogonal and Unitary Group. In *International Conference on Machine Learning (ICML)*, 3794–3803.

Li, R.; Wang, X.; and Yu, H. 2020. MetaMT, a Meta Learning Method Leveraging Multiple Domain Data for Low Resource Machine Translation. In *AAAI Conference on Artificial Intelligence (AAAI)*, 8245–8252.

Liu, Y.; Shang, F.; Cheng, J.; Cheng, H.; and Jiao, L. 2017. Accelerated First-order Methods for Geodesically Convex Optimization on Riemannian Manifolds. In *Advances in Neural Information Processing Systems (NeurIPS)*, 4868–4877.

Metz, L.; Maheswaranathan, N.; Nixon, J.; Freeman, D.; and Sohl-Dickstein, J. 2019. Understanding and correcting pathologies in the training of learned optimizers. In *the International Conference on Machine Learning (ICML)*, 4556–4565.

Pascanu, R.; Mikolov, T.; and Bengio, Y. 2013. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning (ICML)*, volume 28, 1310–1318.

Ravi, S.; and Larochelle, H. 2017. Optimization as a model for few-shot learning. In *International Conference on Machine Learning (ICLR)*.

Sato, H.; Kasai, H.; and Mishra, B. 2019. Riemannian stochastic variance reduced gradient algorithm with retraction and vector transport. *SIAM Journal on Optimization* 29(2): 1444–1472.

Yuan, J.; and Lamperski, A. 2019. Online Adaptive Principal Component Analysis and Its extensions. In *International Conference on Machine Learning (ICML)*, 7213–7221.

Zhang, H.; Reddi, S. J.; and Sra, S. 2016. Riemannian SVRG: Fast stochastic optimization on Riemannian manifolds. In *Advances in Neural Information Processing Systems (NeurIPS)*, 4592–4600.

Zhang, H.; and Sra, S. 2018. Towards Riemannian accelerated gradient methods. *arXiv* preprint:1806.02812.

Zhang, J.; Zhang, H.; and Sra, S. 2018. R-spider: A fast riemannian stochastic optimization algorithm with curvature independent rate. *arXiv* preprint:1811.04194.