Efficient Riemannian Submanifold Algorithm on Stiefel Manifolds for Deep Neural Networks

ESTR4998 Graduation Thesis I Presentation

Fong, Shi Yuk

Email: syfong1@cse.cuhk.edu.hk

Department of Computer Science and Engineering
The Chinese University of Hong Kong

December 5, 2024

Outline

Part 1. Elements of Riemannian Manifold Optimization

Riemannian Manifold

Definition (Manifold)

A manifold $\mathcal{M}$ of dimension d is a topological space that locally resembles Euclidean space. Formally, for any $W \in \mathcal{M}$, there exists an open neighbourhood $U_W \subseteq \mathcal{M}$ and a homeomorphism (topological isomorphism) $\phi : U_W \to V$ where V is an open subset of $\mathbb{R}^d$.

Definition (Riemannian Manifold)

A Riemannian manifold is a smooth manifold equipped with a smooth inner product $\langle \cdot, \cdot \rangle_W$ of tangent vectors at each point W.

Definition (Tangent Space)

The tangent space $\mathcal{T}_W \mathcal{M}$ for a manifold $\mathcal{M}$ at a point $W \in \mathcal{M}$ is defined by the collection of tangent vectors at W for all smooth curves $c : \mathbb{R} \to \mathcal{M}$ passing through W.

The Stiefel Manifold and the Orthogonal Group

Definition (Stiefel Manifold)

The Stiefel manifold is defined by

$$\mathrm{St}(d,p) = \{W \in \mathbb{R}^{d\times p} : W^\top W = I_p\}.$$

The Orthogonal Group is a special case of Stiefel Manifold, which is defined by

$$\mathcal{O}_d = \mathrm{St}(d,d) = \{W \in \mathbb{R}^{d\times p} : W^\top W = WW^\top = I_p\}.$$

Definition (Tangent Space of Stiefel Manifold)

Given a point $W \in \mathrm{St}(d,p)$, the tangent space at W is defined as

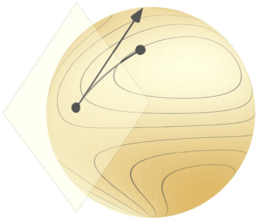
$$\mathcal{T}_W \,\mathrm{St}(d,p) = \{\xi \in \mathbb{R}^{d\times p} \mid \xi^\top W + W^\top \xi = \mathbf{0}\}.$$

Optimization on Stiefel Manifold

Consider the optimization problem:

$$\min_{W \in \mathrm{St}(d,p)} f(W), \tag{1}$$

where $f : \mathbb{R}^{d \times p} \to \mathbb{R}$ may be non-smooth non-convex.



Example: Dimensionality Reduction (PCA)

Given a data matrix $X \in \mathbb{R}^{N \times m}$ consisting of m samples of N-dimensional data, the goal is to find a linear map $T : \mathbb{R}^N \to \mathbb{R}^n$ characterized by a matrix $U \in \mathbb{R}^{N \times n}$ that maps the data into an n-dimensional subspace that best preserves the reconstruction error of the data.
The optimization problem is then

$$\min_{U \in \mathrm{St}(N,n)} ||X - UU^\top X||_F^2. \tag{2}$$

Part 2. Generic Framework for First-Order Riemannian Methods

Riemannian Gradient

For a manifold $\mathcal{M}$, given a point W and its corresponding tangent space $\mathcal{T}_W\mathcal{M}$, the Riemannian gradient is defined as

$$\widetilde{\nabla} f(W) = \mathcal{P}_{\mathcal{T}_W\mathcal{M}}(\nabla f(W)), \tag{3}$$

where $\nabla f(W)$ is the Euclidean gradient of f at p, and $\mathcal{P}_{\mathcal{S}}(\xi) := \arg\min_{\zeta \in \mathcal{S}} ||\zeta - \xi||_2$ is the projection operator onto the set $\mathcal{S}$.

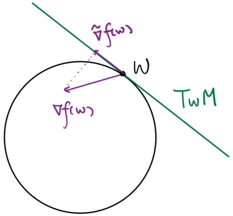


Figure: Illustration of Riemannian gradient on a simple manifold S_1, which is a circle embedded in ambient space $\mathbb{R}^2$.

Riemannian Gradient for Stiefel Manifold

For the Stiefel manifold $\mathrm{St}(d, p)$, the Riemannian gradient is given by

$$\widetilde{\nabla} f(W) = W \operatorname{skew}(W^\top \nabla f(W)), \tag{4}$$

where $\operatorname{skew}(A) = \frac{1}{2}(A - A^\top)$ is the skew-symmetric operator, and $\nabla f(W)$ is the Euclidean gradient of f at W.

Retraction Map

Each gradient descent step is then performed by iteratively updating the point W as

$$W \leftarrow \mathrm{Retr}_W(-\lambda \widetilde{\nabla} f(W)), \tag{5}$$

where $\mathrm{Retr}_W(\cdot)$ is the retraction map at W onto the manifold $\mathcal{M}$, and $\lambda \in \mathbb{R}_{++}$ is the step size. This usually can be done by a two-step process:

1. Compute the Riemannian gradient update $W \leftarrow W - \lambda \widetilde{\nabla} f(W)$.
2. Project the updated point onto the manifold by $W \leftarrow \mathcal{P}_{\mathcal{M}}(W)$.

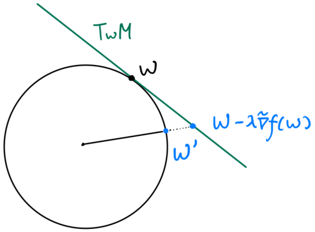


Figure: Illustration of Retraction map on manifold S_1,

Retraction Map for Stiefel Manifold

- Polar-decomposition (Polar) based retraction:

$$\mathrm{Retr}_W^{\mathsf{Polar}}(\xi) = (W + \xi)(I + \xi^\top \xi)^{-\frac{1}{2}}$$

- Exponential map (Expm) retraction:

$$\mathrm{Retr}_W^{\mathsf{Expm}}(\xi) = W \,\mathrm{Exp}(W^\top \xi)$$

where $\mathrm{Exp}(\cdot)$ is the matrix exponential function.

Part 3. Problem Statement

Orthogonal Recurrent Neural Networks

Given the input sequence $x = \{x_t\}_{t=1}^N$, where $x_t \in \mathbb{R}^{d_{\text{in}}}$, the RNN aims to predict the output sequence $\hat{y} = \{y_t\}_{t=1}^N$, where $y_t \in \mathbb{R}^{d_{\text{out}}}$ is close to some ground truth y^*. The forward pass:

$$\begin{cases} h_0 = \mathbf{0}, \\ h_t = \phi(W_{\text{in}} x_t + W h_{t-1}), \\ y_t = W_{\text{out}} h_t + b_{\text{out}}, \end{cases} \tag{6}$$

where $h_t \in \mathbb{R}^d$ with hidden size d is the hidden state at time t, $W_{\text{in}} \in \mathbb{R}^{d \times d_{\text{in}}}$, $W \in \mathcal{O}_d$, $W_{\text{out}} \in \mathbb{R}^{d_{\text{out}} \times d}$, ϕ is the nonlinearity function, and $b_{\text{out}} \in \mathbb{R}^{d_{\text{out}}}$ is the bias term.

Orthogonal Recurrent Neural Networks

We aim to train an RNN with the hidden weight matrix W lying on the orthogonal group $\mathcal{O}_d$: The optimization problem is formulated as follows:

$$\begin{aligned} \min_{\Theta} \quad & \mathcal{L}_\Theta(\hat{y}, y^*) && (7) \\ \text{s.t.} \quad & W \in \mathcal{O}_d, && (8) \end{aligned}$$

where Θ is the set of all trainable parameters, and $\mathcal{L}_\Theta$ is the loss function. In this note, the loss function is assumed to be the cross-entropy loss.

Contribution

In this project, we would like to propose a new Riemannian optimization algorithm on Stiefel manifolds for training ODNNs that

- is more scalable than the existing algorithms;
- can make use of the full gradient information;
- can be implemented in standard machine learning libraries;
- can utilize a larger learning rate.

Part 4. Existing Methods

Singular Value Bounding

Algorithm Singular Value Bounding (**SVB**) [LJW+21]

```
1: Input: Sequence of step sizes $\{\lambda^k\}$; $\epsilon \in \mathbb{R}^+$.
2: Initialize: Set $k = 0$. Set variable $X^0 \in \mathbb{R}^{m \times n}$ and $W^0 \in \mathcal{O}_d$.
3: while not converged do
4:     Update the variables $X^{k+1} \leftarrow X^k - \lambda^k \nabla_{X^k} f$, $W^{k+1} \leftarrow W^k - \lambda^k \nabla_{W^k} f$.
5:     $[U, \Sigma, V] \leftarrow \mathrm{svd}(W^{k+1})$
6:     Clamp each diagonal term $\Sigma_{ii}$ into $[1/(1+\epsilon), 1+\epsilon]$.
7:     $W^{k+1} \leftarrow U \Sigma V^\top$
8:     $k \leftarrow k + 1$
9: end while
```

Issue: Singular value decomposition becomes computationally expensive for large matrices.

Gradient Descent Methods

Algorithm Riemannian Gradient Descent (**RGD-Z**) [Bon13, CWYS24]

```
1: Input: Sequence of step sizes $\{\lambda^k\}$.
2: Initialize: Set $k = 0$. Set variable $X^0 \in \mathbb{R}^{m \times n}$ and $W^0 \in \mathcal{O}_d$.
3: while not converged do
4:     Update unconstrained variable $X^{k+1} \leftarrow X^k - \lambda^k \nabla_{X^k} f$.
5:     Update constrained variable $W^{k+1} \leftarrow \mathrm{Retr}^{\mathbf{Z}}_{W^k}(-\lambda^k \widetilde{\nabla}_{W^k} f)$.
6:     $k \leftarrow k + 1$
7: end while
```

Here, $\mathbf{Z} \in \{\text{Polar}, \text{Expm}\}$ determines the retraction map used in the algorithm. **Issue**: No matter which retraction map is used, the algorithm is still inefficient.

- Polar: eigendecomposition is computationally expensive
- Expm: matrix exponential is computationally expensive

Coordinate Descent Method: Expm

Randomly pick a pair of indices $(i, j) \in \mathcal{I} := \{(i, j) : 1 \leq i < j \leq d\}$ representing the entry in upper triangular part of W. Riemannian partial gradient:

$$\widetilde{\nabla}^{ij} f(W) = \operatorname{tr}(H_{i,j}^\top W^\top \nabla f(W)) W \operatorname{skew}(e_i e_j^\top), \tag{9}$$

Given the Euclidean gradient $\nabla f(W)$, the partial gradient $\widetilde{\nabla}^{ij} f(W)$ can be efficiently computed by the Givens matrix method in $O(d)$ time. $\widetilde{\nabla}^{ij} f(W)$ is the Riemannian partial gradient of f with respect to the columns i and j of W.

Coordinate Descent Method: Expm

Algorithm Riemannian Coordinate Descent (**RCD**) [MA22]

```
1: Input: Sequence of step sizes $\{\lambda^k\}$.
2: Initialize: Set $k = 0$. Set variable $X^0 \in \mathbb{R}^{m\times n}$ and $W^0 \in \mathcal{O}_d$.
3: while not converged do
4:     Update unconstrained variable $X^{k+1} \leftarrow X^k - \lambda^k \nabla_{X^k} f$.
5:     Select a coordinate $(i, j) \in \mathcal{I}$ of the tangent space $\mathcal{T}_{W^k}\mathcal{O}_d$.
6:     Update constrained variable $W^{k+1} = \mathrm{Retr}^{\mathsf{Expm}}_{W^k}(-\lambda^k \widetilde{\nabla}^{ij}_{W^k} f)$.
7:     $k \leftarrow k + 1$
8: end while
```

Issue: Commonly used machine learning libraries (e.g., PyTorch) can only compute **full** gradient, but only a tiny fraction of the full gradient is used in each iteration!

Block Coordinate Descent Method: Polar

Consider a partition $\mathfrak{C}$ of the index set $[d]$ into l blocks $C_1, C_2, \ldots, C_l$. Each index represents a column in W. Randomly pick a pair of indices $(i, j) \in \binom{l}{2}$. Riemannian partial gradient:

$$\widetilde{\nabla}^{ij} f(W) = W_{ij} \operatorname{skew}(W_{ij}^\top \nabla^{C_{ij}} f(W)) - (I - WW^\top) \nabla^{C_{ij}} f(W). \quad (10)$$

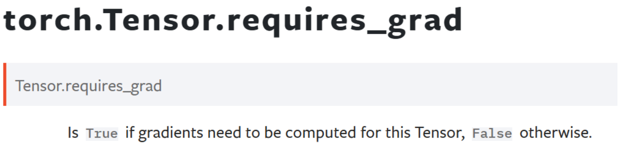
Here, $C_{ij} = C_i \cup C_j$, $\nabla^{C_{ij}} f(W), \widetilde{\nabla}^{ij} f(W)$ are the (Riemannian) partial gradient of f with respect to the block C_{ij}, and W_{ij} is the submatrix of W with columns indexed by C_{ij}.

Block Coordinate Descent Method: Polar

Algorithm Riemannian Submanifold Gradient Method (**RSGM-l**) [CWYS24]

```
1: Input: Sequence of step sizes $\{\lambda^k\}$; a partition $\mathfrak{C} := \{C_1, \dots, C_l\}$ of $[d]$ with $l \geq 2$.
2: Initialize: Set $k = 0$. Set variable $X^0 \in \mathbb{R}^{m \times n}$ and $W^0 \in \mathcal{O}_d$.
3: while not converged do
4:     Uniformly sample $\{i, j\} \sim \binom{[l]}{2}$.
5:     Update unconstrained variable $X^{k+1} \leftarrow X^k - \lambda^k \nabla_{X^k} f$.
6:     Update constrained variable $W_{ij}^{k+1} = \mathrm{Retr}_{W^k}^{\text{Polar}}(-\lambda^k \widetilde{\nabla}_{W^k}^{ij} f), W_{-ij}^{k+1} = W_{-ij}^k$.
7:     $k \leftarrow k + 1$
8: end while
```

Issue: The same as **RCD**.



torch.Tensor.requires_grad

Tensor.requires_grad

Is `True` if gradients need to be computed for this Tensor, `False` otherwise.

Figure: PyTorch only allows gradient freezing at the layer level, without finer-grained control.

Main Issues of Existing Methods

- Scalability
- Partial Gradient Availability & Backpropagation

Main Issues of Existing Methods

- **Scalability**
- Partial Gradient Availability & Backpropagation

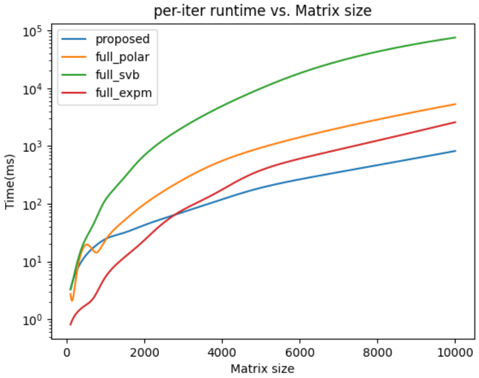


Figure: Runtime comparison for different algorithms that utilize full gradient information.

Main Issues of Existing Methods

- Scalability
- **Partial Gradient Availability & Backpropagation**

Method	Utilize Full Gradient?	Coordinate Method?	Source of Overhead	Overhead Complexity
SVB [LJW$^+$21]	✓	✗	SVD	$O(dp^2)$
RGD-Expm [Bon13]	✓	✗	Matrix exponential	$O(p^3)$
RGD-Polar [CWYS24]	✓	✗	Eigendecomposition	$O(p^3)$
RCD [MA22]	✗	✓	Givens rotation	$O(d)$
RSGM-l [CWYS24]	✗	✓	Eigendecomposition	$O(p^3/l^3)$
Proposed-l	✓	✓	Eigendecomposition	$O(p^3/l^2)$

Table: Comparison of different algorithms.

Part 5. Proposed Method

Proposed Method: Iterative Block Coordinate Descent

Algorithm Proposed Algorithm (Proposed-l)

```
1: Input: Sequence of step sizes $\{\lambda^k\}$; a partition $\mathfrak{C} := \{C_1, \dots, C_l\}$ of $[d]$ with $l \geq 2$.
2: Initialize: Set $k = 0$. Set variable $X^0 \in \mathbb{R}^{m \times n}$ and $W^0 \in \mathcal{O}_d$.
3: while not converged do
4:     Update the unconstrained variable $X^{k+1} \leftarrow X^k - \lambda^k \nabla_{X^k} f$.
5:     $\mathcal{I} \leftarrow [l]$
6:     while $\mathcal{I} \neq \varnothing$ do
7:         Uniformly sample two indices $i, j \sim \mathcal{I}$ without replacement; then, $\mathcal{I} \leftarrow \mathcal{I} \setminus \{i, j\}$.
8:         Update the constrained variable $W_{ij}^{k+1} = \mathrm{Retr}_{W^k}^{\text{Polar}}(-\lambda^k \widetilde{\nabla}_{W^k}^{ij} f)$.
9:     end while
10:    $k \leftarrow k + 1$
11: end while
```

Key difference with **RSGM**: Here, we freeze the full gradient computation and iteratively update **all** columns of W, instead of updating only a portion of the columns.

Part 6. Experiments

Memory Copying Task

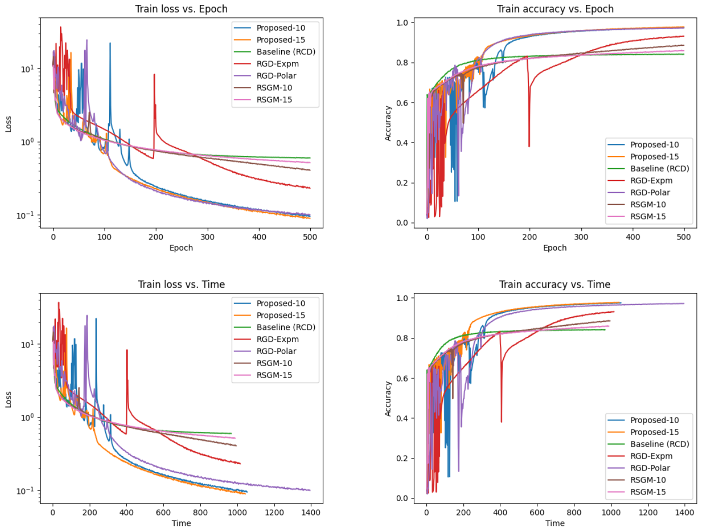


Figure: Convergence result of copying task. The best-known learning rate is used.

Sequential CIFAR-10 Classification Task

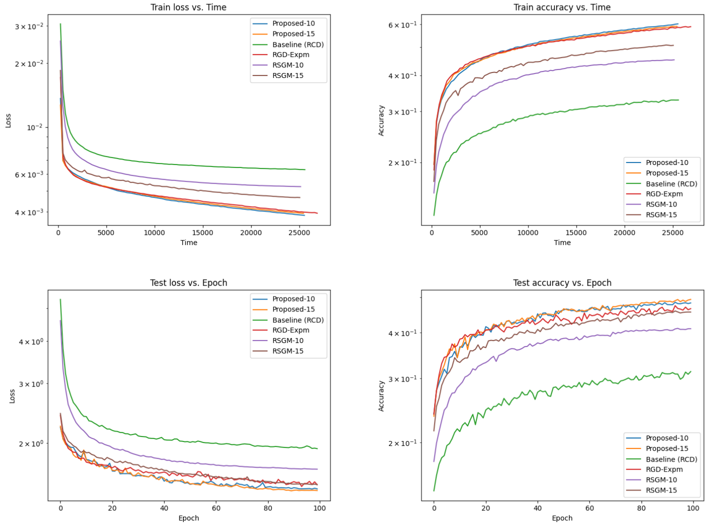


Figure: Convergence result of sequential CIFAR-10 task. The best-known learning rate is used.

Future Work

- Analyze convergence properties and computational complexity
- Develop optimal learning rate selection scheme
- Extend other architectures such as LSTM, ConvNet and ResNet

References

[Bon13] Silvère Bonnabel.
Stochastic gradient descent on riemannian manifolds.
IEEE Transactions on Automatic Control, 58(9):2217–2229, 2013.

[CWYS24] Andy Yat-Ming Cheung, Jinxin Wang, Man-Chung Yue, and Anthony Man-Cho So.
Randomized submanifold subgradient method for optimization over stiefel manifolds, 2024.

[LJW$^+$21] S. Li, K. Jia, Y. Wen, T. Liu, and D. Tao.
Orthogonal deep neural networks.
IEEE Transactions on Pattern Analysis & Machine Intelligence, 43(04):1352–1368, apr 2021.

[MA22] Estelle Massart and Vinayak Abrol.
Coordinate descent on the orthogonal group for recurrent neural network training.
Proceedings of the AAAI Conference on Artificial Intelligence, 36(7):7744–7751, Jun. 2022.