This CVPR paper is the Open Access version, provided by the Computer Vision Foundation.
Except for this watermark, it is identical to the version available on IEEE Xplore.

Geometry Aware Constrained Optimization Techniques for Deep Learning[*]

Soumava Kumar Roy[1], Zakaria Mhammedi[1,2], and Mehrtash Harandi[1,2]

[1]Australian National University, Canberra, Australia
[2]Data61, CSIRO, Canberra, Australia
[1]{soumava.kumarroy, zak.mhammedi, mehrtash.harandi}@anu.edu.au

Abstract

In this paper, we generalize the Stochastic Gradient Descent *(SGD) and RMSProp algorithms to the setting of Riemannian optimization. SGD is a popular method for large scale optimization. In particular, it is widely used to train the weights of* Deep Neural Networks. *However, gradients computed using standard SGD can have large variance, which is detrimental for the convergence rate of the algorithm. Other methods such as RMSProp and ADAM address this issue. Nevertheless, these methods cannot be directly applied to constrained optimization problems. In this paper, we extend some popular optimization algorithm to the Riemannian (constrained) setting. We substantiate our proposed extensions with a range of relevant problems in machine learning such as incremental Principal Component Analysis, computating the Riemannian centroids of SPD matrices, and Deep Metric Learning. We achieve competitive results against the state of the art for fine-grained object recognition datasets.*

1. Introduction

The field of machine learning and computer vision is abundant with problems that can benefit from the use of constraints to obtain optimal solutions. For example, orthogonality constraints have shown to be very useful in linear dimensionality reduction algorithms such as *Principal Component Analysis* (PCA) [4]. Over the past few years, deep learning has led to tremendous improvement in many applications such as image and object recognition, speech recognition, natural language processing, and content based retrieval systems [33]. *Deep Metric Learning* (DML) [27], for example, has received a significant amount of attention lately, and it is well known that learning a metric can benefit from the use of orthogonality constraints [41]. However, most popular optimizers which use a variant of *Stochastic Gradient Descent* (SGD) cannot directly be applied to constrained optimization. We attempt to bridge this gap by extending certain popular optimization algorithms to enhance their capability in handling constraints on the parameter space. We begin our formulation by studying the popular class of empirical risk minimization problems in the Riemannian setting, where the objective can take the following form

$$\underset{\boldsymbol{\theta}\in\mathcal{M}}{\arg\min}\ \frac{1}{n}\sum_{i=1}^{n} J(\boldsymbol{y}_i, f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)), \tag{1}$$

with $(\boldsymbol{x}_i, \boldsymbol{y}_i) \in \mathbb{R}^d \times \mathbb{R}^\ell$ refers to the i^{th} data instance, $\mathcal{M}$ is a Riemannian manifold, $f_{\boldsymbol{\theta}} : \mathbb{R}^d \to \mathbb{R}^\ell$ is a "prediction" function (*e.g.* output of a neural network) parameterized by $\boldsymbol{\theta}$, and J a positive real-valued objective function to be minimized. *Gradient Descent* (GD) is a standard technique used to optimize Equation (1) by updating the parameters of the model along negative gradient directions. However, this approach can be fragile when the objective function is highly non-convex. In this case, the standard GD algorithm can get stuck in a bad local minima leading to poor performance. The SGD [10], the stochastic variant of GD, was introduced to overcome this drawback. SGD trains the model on mini-batches of the training data, resulting in faster convergence rates compared to GD.

The Computed gradients during SGD can display large variances due to the random generation of mini-batches. This can lead to undesirable oscillations of the model parameters around optimal values [29]. Ensuring that these oscillations are dampened has been the subject of several works lately [30, 17]. Two prominent algorithms which address this issue are the *Momentum SGD* (SGD-M) [29] and the *Stochastic Variance Reduced Gradient* (SVRG) [17] and they have been quite successful in achieving higher convergence rates over SGD and GD.

[*]This work was supported under the Australian Research Council's Discovery Projects funding scheme (project DP150104645)

One drawback of SGD-M is that the learning rate is manipulated globally and equally for all parameters. This results in devising various heuristics for tuning the learning rates to ensure a good overall convergence rate. Ideally the learning rates for every parameter should be adapted automatically, in some suitable way, to achieve faster convergence rates. There exist many well documented optimization algorithms; namely Adagrad [11], ADAM [18], Adadelta [45] and RMSProp [15], which maintain adaptable learning rates for different parameters.

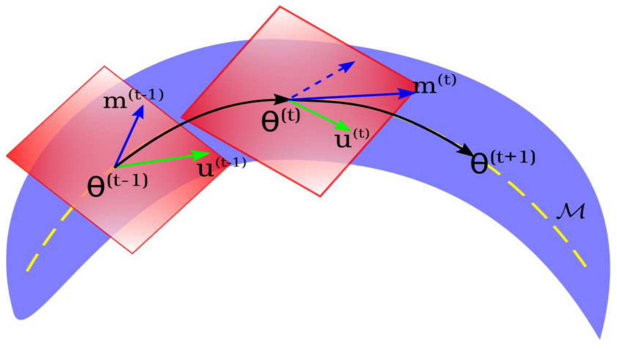
The latter methods cannot directly be applied to large-scale, nonlinear, constrained problems in machine learning. This has motivated us to extend SGD-M and RMSProp to handle constrained optimization problems. Although in the past, both SGD and SVRG have been studied extensively under the lens of Riemannian geometry [5, 47], we believe that our extensions, which we name **cSGDM** and **cRMSProp**, are novel. Inspired from the success of Riemannian optimization techniques [22, 25, 2], we employed concepts from Riemannian geometry to formulate our approach in handling constraints.

Our Contributions

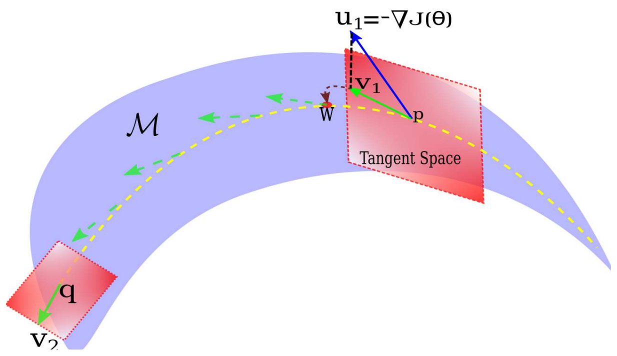
Our contributions in this work are two-fold. **1.** We propose **cSGD-M** and **cRMSProp**, representing extensions of SGD-M and RMSProp, which can be directly used in constrained problem settings. **2.**Using our novel extensions, we substantiate the impact of our proposed methods on some relevant applications in machine learning, such as incremental PCA and DML, where orthogonality constraints naturally arise. We achieve competitive results against the state of the art for fine-grained objective recognition. Our proposed approach can easily be integrated into various deep learning packages.

Related Work

The stochastic variant of the gradient descent algorithm approximates the full gradient using mini-batches of samples. This can cause undesirable oscillations around local optima [48]. One of the most popular techniques used to alleviate this issue is to anneal the learning rate during training. However, this can sometimes be detrimental to the convergence rate [31]. Thus, it is desirable to design methods which control the variance of the gradients so that higher values for the learning rate can be used during optimization. *Stochastic Average Gradient* (SAG) [34] is an example of an algorithm which attempts to do just that. As its name suggests, SAG computes an averaged gradient at each time step. However, this requires the storage of all previous gradient vector which can significantly increase the memory requirements. *Stochastic Variance Reduced Gradient* (SVRG) [17] overcomes this problem by cyclically storing



(a)



(b)

Figure 1: **(a)** Riemannian operations used in performing **cSGD-M**. Refer to the main text for a detailed explanation. The manifold $\mathcal{M}$ is shown by the blue surface, the geodesic by a dotted yellow line, the tangent spaces by light red surfaces and the gradient vectors on the tangent space by green arrows.The update vectors $\{\boldsymbol{m}^{(t)}\}$ (solid blue vectors) are used to obtain the minimizing geodesics. The solution $\theta^{(t+1)}$ at time step $t+1$ is obtained from $\theta^{(t)}$, the current gradient vector $\boldsymbol{u}^{(t)}$ and the velocity vector $\boldsymbol{m}^{(t-1)}$ mapped by parallel transport (shown by the dotted blue vector at t). This velocity vector smooths out the gradients over time and helps to improve the convergence rate of the algorithm. **cRMSProp** also follow similar operations in obtaining the minimizing geodesic, with a slight change in the representation of the vectors $\boldsymbol{m}^{(t)}$ and $\boldsymbol{u}^{(t)}$. **(b)** An Illustrative schematic of various operations used in Riemannian Optimization. $\boldsymbol{p}$ and $\boldsymbol{q}$ represent points on the manifold connected by a geodesic shown by a yellow dotted line. Vector $\boldsymbol{v}_1$ is the projection of the ambient gradient vector $\boldsymbol{u}_1$ at $\boldsymbol{p}$. To move back to the manifold from the tangent space at $\boldsymbol{p}$, we use the retraction $\mathrm{r}_{\boldsymbol{p}}(\boldsymbol{v}_1)$ operator. In a neighborhood of $\boldsymbol{p}$, the retraction (shown in brown) identifies a point on the geodesic, thus guarantees decreasing the objective function (denoted by $J(\theta)$). Here $\boldsymbol{w} = \mathrm{r}_{\boldsymbol{p}}(\boldsymbol{v}_1)$. $\boldsymbol{v}_2$ is the gradient vector obtained at $\boldsymbol{q}$ by mapping $\boldsymbol{v}_1$ through parallel transport, *i.e.*, $\boldsymbol{v}_2 = \Gamma_{\boldsymbol{p}\to\boldsymbol{q}}(\boldsymbol{v}_1)$.

only one copy of the full gradient at current parameter estimate and using it to prevent subsequent updates from deviating too far from this stored copy. Though the convergence analysis of the SVRG is very promising, convincing results in practical machine learning applications are yet to be demonstrated.

Other more popular methods used to decrease undesirable oscillations around local optima include the SGD-M which computes a smoother version of the gradient vector using an exponentially moving average with coefficient ν. This reduces the chances of radical updates in the parameter space. The update equations in this case can be expressed as

$$\boldsymbol{m}^{(t+1)} = \nu\, \boldsymbol{m}^{(t)} + \eta\, \nabla J, \tag{2}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \boldsymbol{m}^{(t+1)}. \tag{3}$$

SGD-M has the drawback that all parameters have a common learning rate, where as it is desirable to have separate learning rates for each parameter of the model based on some suitable criteria. RMSProp is an example of a gradient based optimization algorithm which does this by maintaining an exponentially moving average of the squared gradient, which is an estimate of the 2^{nd} raw moment (the unentered variance) of the gradient. The update equations for RMSProp can be expressed as

$$\boldsymbol{m}^{(t+1)} = \rho\, \boldsymbol{m}^{(t)} + (1-\rho)\, (\nabla J \odot \nabla J), \tag{4}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta \frac{\nabla J}{\sqrt{\boldsymbol{m}^{(t+1)}} + \epsilon}, \tag{5}$$

where ρ and η are hyper-parameters and $\odot$ denotes the *Hadamard* product. Note that the square root and division of vectors is performed element-wise.

Relevant to optimizing a problem under constraints, is the work of Bonnabel et al (2013), which laid down the theoretical foundation for extending SGD to the general Riemannian setting [5]. Riemannian SVRG [47](RSVRG, the Riemannian variant of SVRG) is another promising method which works in the setting of constrained optimization and has good convergence bounds. With these two pioneering work at our disposal, we propose to integrate the geometry of Riemannian manifold in SGD-M and RMSProp.

2. Mathematical Background

In this section, we briefly introduce various concepts of Riemannian geometry. This will be crucial for the understanding of our proposed extensions which enable optimization on Riemannian manifolds. Interested readers are referred to [6, 1] for a more comprehensive introduction to the concepts discussed here.

Notation

Throughout the paper, matrices are denoted by bold upper-case letters whereas column vectors are denoted by bold lower-case letters. The identity function and the identity matrix of size $n \times n$ are denoted $\mathbf{I}_n$ and id, respectively. $\mathrm{Sym}(n)$ denotes the space of real symmetric matrices of size $n \times n$. The Frobenius norm of a matrix $\boldsymbol{A} \in \mathbb{R}^{n \times d}$ denoted by $||\boldsymbol{A}||_F$ is defined as $\sqrt{\mathrm{Tr}(\boldsymbol{A}\boldsymbol{A}^T)}$, where Tr is the trace operator and $\boldsymbol{A}^T$ denotes the transpose of $\boldsymbol{A}$. $\mathcal{S}_{++}^n$ represents the manifold of *Symmetric Positive Definite* (SPD) matrices, while the *Grassmann* and *Stiefel* manifolds are denoted by $\mathcal{G}(n,p)$ and $\mathrm{St}(p,n)$, respectively. A set $\mathcal{M} \subset \mathbb{R}^n$ is a *smooth m-dimensional manifold* if at every point $\boldsymbol{p} \in \mathcal{M}$ there exists an open neighborhood of $\boldsymbol{p}$ such that its intersection with $\mathcal{M}$ is diffeomorphic to $\mathbb{R}^m$. The *tangent space* at $\boldsymbol{p} \in \mathcal{M}$, denoted by $T_{\boldsymbol{p}}\mathcal{M}$, is a real vector space containing all tangent vectors to $\mathcal{M}$ at $\boldsymbol{p}$. A vector $\boldsymbol{v}$ is tangent to $\mathcal{M}$ at $\boldsymbol{p}$, if there exists a smooth curve $\gamma : I \to \mathcal{M}$, I open interval containing 0, such that $\gamma(0) = \boldsymbol{p}$ and $\dot{\gamma}(0) = \boldsymbol{v}$. A smooth m-dimensional manifold equipped with an inner product on the tangent space at every point on the manifold is called as a *Riemannian manifold*. This inner product induces a *Riemannian metric* which then defines various geometric concepts on the manifold such as curve lengths, volumes, and gradients of functions defined on the manifold. Informally, *geodesics* on a smooth manifold $\mathcal{M}$ are smooth curves on $\mathcal{M}$ generalizing the concept of straight lines in Euclidean spaces. Locally, geodesics are length minimizing; that is, the shortest path between two points $\boldsymbol{p}, \boldsymbol{q} \in \mathcal{M}$, close enough to each other with respect the metric topology, is given by the length of the geodesic curve passing through them. A number of operations are needed to perform optimization on a Riemannian manifold. These are listed below.

Orthogonal Projection

Optimization on a manifold $\mathcal{M}$ requires the gradients of some objective function at a point $\boldsymbol{p} \in \mathcal{M}$ to be defined on the tangent space $T_{\boldsymbol{p}}\mathcal{M}$. These gradients can be used to travel appropriately along the manifold to minimize the objective function. In fact, each tangent vector defines a geodesic curve through the *exponential map*. Taking small enough steps along geodesics corresponding to directions of negative gradients ensures reduction of the objective function. However, it is often the case that gradients are computed in the ambient Euclidean space and they need, therefore, to be orthogonally projected onto the appropriate tangent space $T_{\boldsymbol{p}}\mathcal{M}$, $\boldsymbol{p} \in \mathcal{M}$. We denote $\pi_{\boldsymbol{p}} : \mathbb{R}^n \to T_{\boldsymbol{p}}\mathcal{M}$ the orthogonal projection onto the tangent space at $\boldsymbol{p}$. As an example, consider the following constrained minimization problem $\min_{\|\boldsymbol{\theta}\|=1} J(\boldsymbol{\theta})$. Figure 1(b), shows the descent direction $\boldsymbol{u}_1$ at $\boldsymbol{p}$ in the ambient space; that is, $\boldsymbol{u}_1 = -\frac{\partial J}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}=\boldsymbol{p}}$. It is obvious from the diagram that moving along the direction of $\boldsymbol{u}_1$ will violate the constraint $\|\boldsymbol{\theta}\| = 1$; the updated parameter will lie outside the desired manifold. Therefore, it is necessary to project the gradient $\boldsymbol{u}_1$ onto $T_{\boldsymbol{p}}\mathcal{M}$. Then, taking an appropriate step along the corresponding geodesic curve to reduce the objective function while keeping the parameters of the model on the manifold $\mathcal{M}$.

Retraction

As stated above, the next step after obtaining the required tangential gradient vector is to move along the corresponding geodesic curve. This geodesic curve is obtained by applying the exponential map, denoted by Exp, to scaled copies of the (negative) gradient vector. However, evaluating the exponential map can be computationally expensive. The *retraction operation* $\mathrm{r}_{\boldsymbol{p}} : T_{\boldsymbol{p}}\mathcal{M} \rightarrow \mathcal{M}$ is often used as a more efficient approximation of the exponential map (see Figure 1(b) for an illustration). Most manifolds have well defined *Inverse Exponential Map* which is a mapping $\mathrm{r}_{\boldsymbol{p}}^{-1} : \mathcal{M} \rightarrow T_{\boldsymbol{p}}\mathcal{M}$, satisfying $\mathrm{r}_{\boldsymbol{p}}^{-1}(\mathrm{Exp}_{\boldsymbol{p}}(\boldsymbol{u})) = \boldsymbol{u} \in T_{\boldsymbol{p}}\mathcal{M}$.

Parallel Transport

Most optimization techniques such as SGD-M and RMSprop use smooth estimates of first or second order moments of gradient vectors. In Euclidean spaces, these estimates are obtained merely by linearly combining previous moment vectors due to the inherent "undistorted" nature of Euclidean spaces. However, since general Riemannian manifolds can be curved, it is not possible to simply add moment vectors at different points on the manifold, as the resulting vectors may not even lie in the tangent spaces at either points. A way around this is to use *parallel transport*. The parallel transport operator $\Gamma_{\boldsymbol{p}\rightarrow\boldsymbol{q}} : T_{\boldsymbol{p}}\mathcal{M} \rightarrow T_{\boldsymbol{q}}\mathcal{M}$ takes $\boldsymbol{v}_{\boldsymbol{p}} \in T_{\boldsymbol{p}}\mathcal{M}$ and outputs $\boldsymbol{v}_{\boldsymbol{q}} \in T_{\boldsymbol{q}}\mathcal{M}$. Informally, $\boldsymbol{v}_{\boldsymbol{q}}$ is obtained by moving $\boldsymbol{v}_{\boldsymbol{p}}$ in a "parallel" fashion along the geodesic curve connecting $\boldsymbol{p}$ and $\boldsymbol{q}$, where the intermediate vectors obtained through this process have constant norm and are always lying on a tangent space. In practice, if $\Gamma_{\boldsymbol{p}\rightarrow\boldsymbol{q}}$ is not known for a given Riemannian manifold or it is computationally expensive to evaluate, one can, as an approximation, use the orthogonal projection to correct and map tangent vectors between $\boldsymbol{p}$ and $\boldsymbol{q}$ on $\mathcal{M}$ [6].

In order to be self-contained, we also provide brief definitions of the matrix manifolds used in this work as well as some key terminologies.

Definition 1 (The SPD Manifold) *It is defined as the set of $(p \times p)$ dimensional real, SPD matrices endowed with the* Affine Invariant Riemannian Metric *(AIRM) [28], as*

$$\mathcal{S}_{++}^{p} \triangleq \{\boldsymbol{M} \in \mathbb{R}^{p\times p} : \boldsymbol{v}^T\boldsymbol{M}\boldsymbol{v} > 0,\ \forall \boldsymbol{v} \in \mathbb{R}^p - \{\boldsymbol{0}_p\}\}.$$

The SPD manifold $\mathcal{S}_{++}^{p}$ represents the interior of a convex cone in the $p \times (p+1)/2$ dimensional Euclidean Space.

Definition 2 (The Grassmann Manifold) *The Grassmann manifold $\mathcal{G}(n,p)$ [16], $p \leq n$, represents the collection of subspaces spanned by the columns of $n \times p$ matrices whose columns are orthonormal. That is,*

$$\mathcal{G}(n,p) \triangleq \{Span\big(\boldsymbol{X}\big) : \boldsymbol{X} \in \mathbb{R}^{n\times p}, \boldsymbol{X}^T\boldsymbol{X} = \mathbf{I}_p\}.$$

Definition 3 (The Stiefel Manifold) *The Stiefel manifold* $\mathrm{St}(p,n)$ *[6] consists of the set of $(n \times p)$-dimensional matrices, $p \leq n$, with orthonormal columns equipped with the Frobenius inner product*

$$\mathrm{St}(p,n) \triangleq \{\boldsymbol{W} \in \mathbb{R}^{n\times p} : \boldsymbol{W}^T\boldsymbol{W} = \mathbf{I}_p\}.$$

The dimensionalities of $\mathcal{S}_{++}^{p}$, $\mathcal{G}(n,p)$ and $\mathrm{St}(p,n)$ are $p(p+1)/2$, $n(n-p)$ and $np - \frac{1}{2}p(p+1)$, respectively. Note that there is a subtle difference between the Grassmann and Stiefel manifolds; a point on the Stiefel manifold represents a basis for a subspace, whereas a point on the Grassmann manifold represents an entire subspace. Therefore, the set of orthogonal matrices defining the same subspace can be represented by a single point on a Grassmann manifold. This subtle difference between the two will be relevant in choosing appropriate constraints for different problems.

3. Our Approach

Throughout this paper, we focus on constrained problems of the form

$$\min_{\boldsymbol{\theta}\in\mathcal{M}} J(\theta). \tag{6}$$

where J is an objective function parameterized by $\boldsymbol{\theta}$. In a general manifold setting equipped with a Riemannian metric, the parameter update at time step t is given by

$$\boldsymbol{\theta}^{(t+1)} = \mathrm{r}_{\boldsymbol{\theta}^{(t)}}\Big(-\eta\pi_{\boldsymbol{\theta}^{(t)}}\big(\nabla J\big)\Big), \tag{7}$$

where ∇J denotes the gradient of J at $\boldsymbol{\theta}^{(t)}$, and π (*resp.* r) refers to the orthogonal projection (*resp.* retraction) on the manifold (*c.f.* § 2). Most of traditional optimization algorithms operate on Euclidean spaces. As a result of the flat geometry of the latter, we have $\pi_{\boldsymbol{p}}(\boldsymbol{x}) = \mathrm{id}(\boldsymbol{x})$ and $\mathrm{r}_{\boldsymbol{p}}(\boldsymbol{x}) = \boldsymbol{p} + \boldsymbol{x}$. Thus the update in Equation (7) reduces to the more familiar GD update

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta\nabla J. \tag{8}$$

We now introduce our proposed extensions to SGD-M and RMSprop in the constrained setting.

3.1. Constrained SGD-M

Our aim here is to extend the SGD-M method so that it can be directly applied to the constrained optimization problem in Equation (6). To this end, we need to derive the Riemannian equivalents of Equations (2) and (3). We first start with Equation (3). Comparing Equations (7) and (8), it can be observed that if $\boldsymbol{m}^{(t+1)} \in T_{\boldsymbol{\theta}^{(t)}}\mathcal{M}$, the Riemannian counterpart of (3) will take the following form

$$\boldsymbol{\theta}^{(t+1)} = \mathrm{r}_{\boldsymbol{\theta}^{(t)}}\big(-\boldsymbol{m}^{(t+1)}\big).$$

However, the main difficulty arises when trying to extend Equation (2). Note that it is desired to have $\boldsymbol{m}^{(t+1)} \in T_{\boldsymbol{\theta}^{(t)}}\mathcal{M}$ at time step $t+1$. Naturally, we will also have $\boldsymbol{m}^{(t)} \in T_{\boldsymbol{\theta}^{(t-1)}}\mathcal{M}$. Therefore, $\boldsymbol{m}^{(t)}$ must be transported from $T_{\boldsymbol{\theta}^{(t-1)}}\mathcal{M}$ to $T_{\boldsymbol{\theta}^{(t)}}\mathcal{M}$ to then be combined with the projected gradient vector at $\boldsymbol{\theta}^{(t)}$. This will yield the updated momentum vector $\boldsymbol{m}^{(t+1)} \in T_{\boldsymbol{\theta}^{(t)}}\mathcal{M}$. This is analogous to the rule of vector addition in Euclidean spaces. Recall that $\Gamma_{\boldsymbol{p}\to\boldsymbol{q}}(\boldsymbol{v})$ moves a vector $\boldsymbol{v} \in T_{\boldsymbol{p}}\mathcal{M}$ along a geodesic curve to obtain a new vector in $T_{\boldsymbol{q}}\mathcal{M}$ (*c.f.* § 2). Thus, we can make use of parallel transport to get the following momentum update rule for the *Constrained Stochastic Gradient Descent with Momentum* (cSGD-M).

$$\boldsymbol{m}^{(t+1)} = \gamma\Gamma_{\boldsymbol{\theta}^{(t-1)}\to\boldsymbol{\theta}^{(t)}}\Big(\boldsymbol{m}^{(t)}\Big) + \eta\pi_{\boldsymbol{\theta}^{(t)}}(\nabla J).$$

3.2. Constrained RMSProp

To derive the constrained counterpart of RMSProp, we follow a similar approach to the case of cSGD-M. Looking at Equation (5), we can express the constrained update as

$$\boldsymbol{\theta}^{(t+1)} = r_{\boldsymbol{\theta}^{(t)}}\Big(-\eta\frac{\pi_{\boldsymbol{\theta}^{(t)}}(\nabla J)}{\sqrt{\boldsymbol{m}^{(t+1)}}+\epsilon}\Big).$$

Now to translate Equation (4) to the Riemannian setting, we need to project the vector $\nabla J \odot \nabla J$ onto $T_{\boldsymbol{\theta}^{(t)}}\mathcal{M}$ and apply the appropriate parallel transport to $\boldsymbol{m}^{(t)}$ so that they can be combined. This can be summarized as follows

$$\begin{aligned}\boldsymbol{m}^{(t+1)} = \rho\Gamma_{\boldsymbol{\theta}^{(t-1)}\to\boldsymbol{\theta}^{(t)}}(\boldsymbol{m}_t) \\ + (1-\rho)\pi_{\boldsymbol{\theta}^{(t)}}(\nabla J \odot \nabla J).\end{aligned}$$

4. Experiments:

In order to draw fair comparison between the standard optimization algorithms; SGD, SGD-M, SVRG, RMSProp, and their respective Riemannian counterparts; cSGD, RSVRG, **cSGD-M** and **cRMSProp**, we consider two types of experiments. First, we look at some classical problems in machine learning; we evaluate the performance of our novel extensions on the problem setting of incremental PCA [4] as well as the task of computing the *Riemannian Centroid* of SPD matrices. These fall under the category of *Linear Dimensionality Reduction*. Second, we look at the problem of Mahalanobis metric learning [13], where we incorporate **cSGD-M** and **cRMSProp** into deep *Convolutional Neural Network* (CNN) architectures.

Algorithm 1

```
1: Input: $\nabla(J) \leftarrow$ gradient of the loss J with respect to
   the parameters $\boldsymbol{\theta}$ at time step $t$.
2: $\nabla_{\boldsymbol{\theta}(t)} \leftarrow \pi_{\boldsymbol{\theta}^{(t)}}(\nabla J)$
                         cSGD-M
3: $\boldsymbol{m}^{(t+1)} \leftarrow \gamma\,\Gamma_{\boldsymbol{\theta}^{(t-1)}\to\boldsymbol{\theta}^{(t)}}\Big(\boldsymbol{m}^{(t)}\Big) + \eta\nabla_{\boldsymbol{\theta}(t)}$
4: $\boldsymbol{\theta}^{(t+1)} \leftarrow \mathrm{r}_{\boldsymbol{\theta}^{(t)}}\big(-\boldsymbol{m}^{(t+1)}\big)$.
                         cRMSProp
3: $\nabla^2_{\boldsymbol{\theta}(t)} \leftarrow \pi_{\boldsymbol{\theta}^{(t)}}(\nabla J \odot \nabla J)$
4: $\boldsymbol{m}^{(t+1)} \leftarrow \rho\,\Gamma_{\boldsymbol{\theta}^{(t-1)}\to\boldsymbol{\theta}^{(t)}}\Big(\boldsymbol{m}^{(t)}\Big) + (1-\rho)\nabla^2_{\boldsymbol{\theta}(t)}$
5: $\boldsymbol{\theta}^{(t+1)} \leftarrow r_{\boldsymbol{\theta}^{(t)}}\Big(-\eta\frac{\nabla_{\boldsymbol{\theta}(t)}}{\sqrt{\boldsymbol{m}^{(t+1)}}+\epsilon}\Big)$
```

In all the experiments described bellow, we selected the best hyper-parameters for each algorithm by performing a standard grid search. All performance results reported here are with respect to optimal hyper-parameters. The values of the latter are provided in the supplementary material.

4.1. Linear Dimensionality Reduction

In this section, we evaluate our proposed methods on the incremental version of PCA for face recognition. Most linear dimensional reduction techniques can be formulated as optimization problems over matrix manifolds [9], that is

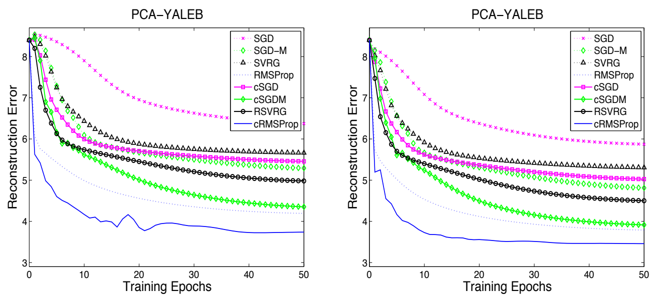
$$\min_{\boldsymbol{M}\in\mathcal{M}} \quad J(\boldsymbol{M}),$$

where $\boldsymbol{M} \in \mathcal{M} \subset \mathbb{R}^{n\times p}$, $p < n$, is usually required to satisfy $\boldsymbol{M}^T\boldsymbol{M} = \mathbf{I}_p$ [9]. As discussed before, both the Stiefel and Grassmann manifolds can represent the latter orthogonality constraint. The correct geometry that should be considered in a given problem depends on the nature of objective function J. In particular, if J exhibits a rotation invariance property, that is $J(\boldsymbol{MR}) = J(\boldsymbol{M})$ for $\boldsymbol{R} \in \mathcal{O}(p)$, then the Grassmann manifold is more suitable than the Stiefel manifold, and vice versa [9]. In incremental PCA, the goal is to minimize the reconstruction error computed through the following objective function

$$J(\boldsymbol{M}) = \left\|\boldsymbol{X} - \boldsymbol{M}\boldsymbol{M}^T\boldsymbol{X}\right\|_F^2,$$

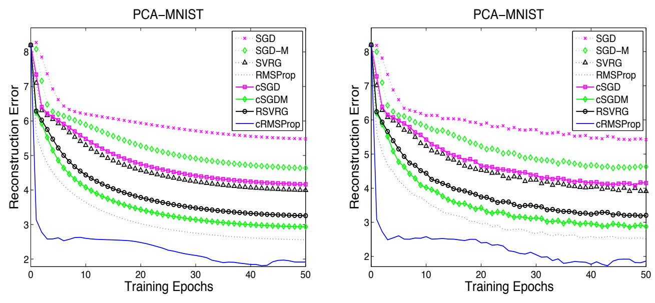
where $\boldsymbol{X} \in \mathbb{R}^{n\times N}$ is a matrix whose columns are N data points. Note that since the objective function satisfies $J(\boldsymbol{M}) = J(\boldsymbol{MR})$, for $\boldsymbol{R} \in \mathcal{O}(p)$, minimizing J can be view as constraint optimization problem over the Grassmann manifold $\mathcal{G}(n,p)$; that is

$$\min_{\boldsymbol{M}\in\mathcal{G}(n,p)} \left\|\boldsymbol{X} - \boldsymbol{M}\boldsymbol{M}^T\boldsymbol{X}\right\|_F^2. \tag{9}$$



(a) $p = 128$ (b) $p = 256$

Figure 2: Plots for objective values for all the algorithms. **(a)** and **(b)** show the plots of reconstruction error *i.e*. Equation (9) for the various algorithms for *YALEB-Test* dataset when p is set to 128 and 256 respectively for a Mini-Batch (MB) size of 256. Similar results were obtained for different values of p and MB.



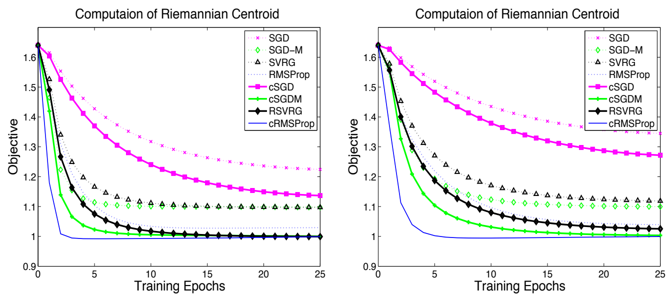
(a) $p = 128$ (b) $p = 256$

Figure 3: Plots for objective values for all the algorithms. **(a)** and **(b)** show the plots of reconstruction error *i.e*. Equation (9) for the various algorithms for *MNIST-Test* dataset when p is set to 128 and 256 respectively for a Mini-Batch (MB) size of 256. Similar results were obtained for different values of p and MB.

Dataset

- **(a)** The *YaleB* database [24] consists of images of 38 subjects captured under 64 different illumination conditions. We used a preprocessed version of the database, where each face is cropped, down-sampled to a 48×42 image, and converted to a column vector of size 2016. We used the first 52 images of every subject for training (*YALEB-Train*) and the remaining 12 images for testing (*YALEB-Test*).
- **(b)** The *MNIST* dataset [23] which contains 70000 $28\times$ 28 gray-scale images of handwritten digits. The preprocessing step involves transforming every image to 784 dimensional vector and normalizing its pixel values to lie in the range of [0, 1]. We have followed the standard train-test split strategy consisting of 60000 training images (*MNIST-Train*) and 10000 test images overall (*MNIST-Test*). We performed experiments for different values of p and mini-batch sizes for both the datasets.

Figures 2(a) and 2(b) show the reconstruction error on the *YALEB-Test* dataset for the different algorithms discussed in the previous section.Similar results for *MNIST-Test* are shown in Figures 3(a) and 3(b). It is clear from the error plots that for each optimization method, the addition of the orthogonality constraint leads to lower reconstruction errors with a faster rates of convergence. Note also that **cRMSProp** achieves both the lowest reconstruction error and the highest convergence rate compared to all other algorithms. **cSGD-M** comes in second place after the RMSprop variants.



(a) MB = 64 (b) MB = 64

Figure 4: The plots for optimizing Equation (10) to calculate the Riemannian Centroid for KYLBERG dataset is shown in **(c)** and **(d)** for the specified size of Mini-Batch (MB). Similar results were observed for different sizes of MB.

4.2. Computation of the Riemannian Centroid

Medical imaging is an example of a field where computations using SPD matrices [8] are crucial (DTI Tensor Imaging) [12]. The problem we consider here is that of finding the Riemannian centroid of a set of SPD matrices $\boldsymbol{A}_1, \boldsymbol{A}_2 \cdots \boldsymbol{A}_k$ such that $\boldsymbol{A}_i \in \mathcal{S}^n_{++}, i = 1, \ldots k$. The objective function can be expressed as the follows

$$\min_{\boldsymbol{M}} \sum_{i}^{k} \left\| \log(\boldsymbol{A}_i^{-1/2} \boldsymbol{M} \boldsymbol{A}_i^{-1/2}) \right\|_F^2 . \qquad (10)$$

In the constrained versions of our algorithms, we require the matrix $\boldsymbol{M}$ to be in the $\mathcal{S}^n_{++}$ manifold. This constraint appears natural as we are trying to compute the centroid of a set of matrices in $\mathcal{S}^n_{++}$.

Dataset

We follow similar preprocessing steps to [14] when evaluating our method on the KYLBERG dataset [20]. The latter consists of patches from 28 different texture classes of natural and artificial surfaces. There are two versions of the dataset; one *with* rotated texture patches and another *without*. Here we considered the latter dataset which has 160 unique samples per class. The original images were scaled to 128×128 pixels and Covariance Descriptors [38] were generated from 1024 4×4 non-overlapping pixel grids. At each position u, v on the coarse grid, we generate a 5 dimensional feature vector as follows

$$x_{u,v} = \left[I_{u,v}, \left|\frac{\partial I}{\partial u}\right| \left|\frac{\partial I}{\partial v}\right| \left|\frac{\partial^2 I}{\partial u^2}\right| \left|\frac{\partial^2 I}{\partial v^2}\right| \right],$$

where $I_{u,v}$ represents the intensity. We have used 80-20% split for each class to create the training and test datasets respectively. Figures 4(a)and 4(b) show the results for this experiment in terms of the objective value of Equation (10) evaluated on the test dataset. We observe a similar trend compared to the case of incremental PCA; the constrained versions of the SGD algorithms performed better than their non-constrained counterparts. Once again, the **cRMSProp** reaches the lowest objective value and the highest convergence rate, with the **cSGD-M** coming second.

4.3. Deep Metric Learning

In general, *Metric Learning* refers to algorithms which learn a suitable similarity function in the feature space which ensures that large values (resp. small values) for inter-class separability (resp. intra-class separability) are maintained [41]. Conventional classification techniques do not perform well in settings where there is either a large number of categories or very few examples in some categories [27]. This has paved the way for metric learning algorithms which directly learn the underlying geometrical structure of the relevant features in a dataset. Thus, the problem of classification reduces to that of standard clustering using the learned metric. Metric learning has been crucial in several important computer vision tasks such as content-based image retrieval, image classification, face recognition, and unsupervised learning algorithms such as clustering [3]. A large pool of algorithms exists in the literature regarding metric learning. However, for simplicity, we decided to focus our attention on learning a *Mahalanobis metric* [44]. A Mahalanobis metric is defined as

$$d_{\boldsymbol{M}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sqrt{(\boldsymbol{x}_i - \boldsymbol{x}_j)^T \boldsymbol{M} (\boldsymbol{x}_i - \boldsymbol{x}_j)}, \qquad (11)$$

where $\mathbb{R}^{n\times n} \ni \boldsymbol{M} \succeq 0$ which ensures that $d_{\boldsymbol{M}}(\cdot,\cdot)$ has the properties of a *pseudo-metric*. By learning a suitable Mahalanobis (pseudo) metric one can infer the geometrical structure of the feature space and use it to compare examples of categories not seen during training - this is known as *Zero shot Learning* [43, 46]. In this case, a model is trained on a dataset and evaluated on a second set with instances representing similar concepts and sharing the same "semantical" metric. The first step to learning a metric is to map data points to a suitable features space. *Deep Neural Networks* are good candidates for this task as they can model complex mapping while insuring that instances from the same (resp. different) categories are semantically closer (resp. further apart). For this experiment we use the Siamese architecture [7] consisting of two identically parameterized subnetworks, followed by a linear layer whose weights are captured by a matrix $\boldsymbol{L} \in \mathbb{R}^{n\times p}$. The matrix $\boldsymbol{M}$ in (11) is then expressed as $\boldsymbol{M} = \boldsymbol{L}\boldsymbol{L}^T$ which ensures that it is always positive semi-definite. Note, however, that multiplying $\boldsymbol{L}$ by an orthogonal matrix $\boldsymbol{Q}$ yields the same pseudo-metric $d_{\boldsymbol{M}}$. Such a *rotation invariance* is known to be detrimental for the convergence of learning algorithms [26, 13]. One way around this problem is to further decompose $\boldsymbol{M}$ as $\boldsymbol{M} = \boldsymbol{U}\boldsymbol{D}^{\frac{1}{2}}\boldsymbol{D}^{\frac{1}{2}}\boldsymbol{U}^T$, where $\boldsymbol{U} \in \mathrm{St}(p,n)$ and $\boldsymbol{D} \in \mathbb{R}^{p\times p}$. Note that replacing $\boldsymbol{U}$ by $\boldsymbol{U}\boldsymbol{Q}$, yields a different symmetric positive definite matrix $\boldsymbol{M}$, which circumvents rotation invariance problem. As a result of this further decomposition, the linear layer of the Siamese network is divided into two sublayers whose weights are the new matrices $\boldsymbol{U}$ and $\boldsymbol{D}^{\frac{1}{2}}$. To test our approach, we have trained the Siamese network with the triplet embedding loss [35] for fine-grained object recognition task. More specifically, from a mini-batch of N training samples, we *mine* m triplets of the form $\left\{\left(\boldsymbol{x}_i^a, \boldsymbol{x}_i^+, \boldsymbol{x}_i^-\right)\right\}_{i=1}^m$, belonging to the feature space (*i.e.* from the output of the first layer of the network) with the constraint that $\left(\boldsymbol{x}_i^a, \boldsymbol{x}_i^+\right)$ are in the same category, whereas $\left(\boldsymbol{x}_i^a, \boldsymbol{x}_i^-\right)$ are not. In our experiments, we used the semi-hard mining strategy [35] to generate the triplets as this ensures more robustness during the training phase. These triplets are then used to compute the embedding loss

$$J(\theta) = \frac{1}{|P|}\sum_{i=1}^{|P|}\left[\left\|\boldsymbol{x}_i^a - \boldsymbol{x}_i^+\right\|^2 - \left\|\boldsymbol{x}_i^a - \boldsymbol{x}_i^-\right\|^2 + \tau\right]_+.$$

where $[y]_+ = \max(0, y)$ is the hinge loss, $|P|$ is the number of positive pairs, and $\tau > 0$ is the user-specified margin.

Dataset

We followed the experimental protocol described in [21] and evaluate our method on two fine-grained datasets using the same train/test split strategy. **(1)** The *Caltech-UCSD Birds* (CUB-200-2001) [42] consists of 11,788 images of birds from 200 different varieties. The first 100 categories are for training (5,864 images), while the remaining 100 categories are for testing (5,924 images). **(2)** The *CARS196* dataset(Cars) [19] consists of 16,185 images of cars from

Table 1: NMI and Recall@K evaluation on the Birds (CUB-200-2011) [42] and CARS196 [19] dataset. Note that all the constrained optimization has been performed in the non-compact Stiefel manifold.

Dataset	CUB-200-2011					CARS196				
Method	NMI	R@1	R@2	R@4	R@8	NMI	R@1	R@2	R@4	R@8
KMeans Results										
DFL [36]	59.23	48.18	61.44	71.83	81.92	59.04	58.11	70.64	80.27	87.81
DSC [21]	56.99	47.57	59.66	71.57	81.28	56.08	57.08	69.23	79.39	87.46
SGD	51.15	42.63	53.51	61.47	72.52	51.31	52.29	59.65	70.53	78.07
cSGD	52.86	44.15	55.07	62.97	74.18	53.79	54.53	61.16	72.17	78.59
SGD-M	55.4	47.41	58.61	66.61	77.51	55.13	56.19	65.09	74.25	79.71
cSGD-M	58.74	49.53	61.81	71.95	81.78	59.03	59.28	70.56	81.02	87.79
RMSProp	58.85	48.19	61.23	72.13	81.05	60.11	61.43	72.05	81.97	89.17
cRMSProp	**61.37**	**52.40**	**64.26**	**74.97**	**83.59**	**63.58**	**69.17**	**80.31**	**85.59**	**92.13**
Spectral Clustering Results										
DSC [21]	58.12	49.78	62.56	73.55	82.78	58.04	59.37	71.25	80.62	88.32
cRMSProp	**61.37**	**52.40**	**64.26**	**74.97**	**83.59**	**63.58**	**69.17**	**80.31**	**85.59**	**92.13**

196 different models. The first 98 models (8,054 images) are allocated for training, while the remaining 98 models (8,131 images) are used for testing. We set the dimensionality of the embedding space to the number of training classes for both datasets; that is, the dimensionality of the embedding space for CUB-200-2011 and CARS196 datasets are set to 100 and 98, respectively. Moreover, the categories for training and testing are disjoint (although they still belong to the same meta-class, *i.e*. birds or cars). To avoid overfitting, we performed early stopping similar to [36, 21].

We used MatConvnet[1] [39] to implement our proposed methods. We used the Inception [37] network with batch normalization pretrained on ImageNet/ILSVRC 2012-CLS [32] dataset. We further finetuned the network on the fine-grained training datasets. All the input images have been first resized to 256 $\times$ 256 and cropped to 224 $\times$ 224. During training, we augmented the images by performing random cropping and horizontal flipping. However, the test images are cropped from the center. We followed the protocol of [36] and used a single crop for every image. We fixed the mini-batch size to 120 and made sure that there are at least two examples of the same category in each mini-batch. The metric learning experiments are performed in the ”Zero-shot” learning context where train and test instances have disjoint identities. Being fundamentally different from classification experiments, we did not cross-validate any parameters and chose parameters (*e.g*., learning rate) by observing the convergence behavior of the objective. We performed the standard *K-Means* and Spectral Clustering (SC) [40] on our test samples and used **(a)** *Normalized Mutual Information* (NMI) and **(b)** *Recall@K* (R@K) [27] to evaluate our method. The results for both datasets are reported in Table 1. We have borrowed the results reported in the baseline papers [36, 21] in our comparison without performing any finetuning of their methods on our own. Moreover, for a fair comparison we have only compared against the *end-to-end* results of of Law et al. [21] for both KMeans and SC. Similar to previous observations, **cRMSProp** had superior performance against all the baselines and **cSGD-M**. We also observe that **cSGD-M** performs almost as well as the two baseline methods used for our comparisons. This in turn suggests that one can achieve competitive results by incorporating geometrical constraints in the standard triplet loss and optimizing it with the Riemannian versions of the SGD algorithms. We also acknowledge that the best results in Law et al. [21] are obtained by finetuning only the last layer, followed by SC. Likewise, we also fine-tune the last Stiefel layer with triplet loss, followed by SC and obtain competitive results in the various evaluation metrics. (For *e.g*., Law et al. [21] achieve a NMI of 64.25% on the CARS196 dataset whereas we obtain 64.20% by just finetuning the last layer.)

[1]`https://github.com/sumo8291/cRMSProp.git`

5. Conclusion

In this paper, we introduced novel extensions of SGD-M and RMSProp which allow can directly be applied to constrained problems. Our experimental evaluations demonstrated the benefit of using our approach across several key constrained optimization problems. In particular, we obtained state-of-the art results for metric learning on two challenging fine-grained objection recognition datasets. In the future, we plan to extend our proposed method to several other optimization algorithms such as Adam [18], ADAGRAD [11] and also cover other problems with geometrical constraints not considered in this work.

References

[1] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization algorithms on matrix manifolds*. Princeton University Press, 2009. 3

[2] M. Belkin and P. Niyogi. Semi-supervised Learning on Riemannian Manifolds. *Machine learning*, 56(1-3):209–239, 2004. 2

[3] A. Bellet, A. Habrard, and M. Sebban. A Survey on Metric Learning for Feature Vectors and Structured Data. *arXiv preprint arXiv:1306.6709*, 2013. 7

[4] C. M. Bishop. *Pattern Recognition and Machine Learning*. springer, 2006. 1, 5

[5] S. Bonnabel. Stochastic Gradient Descent on Riemannian Manifolds. *IEEE Transactions on Automatic Control*, 58(9):2217–2229, 2013. 2, 3

[6] W. M. Boothby. *An Introduction to Differentiable Manifolds and Riemannian Geometry*, volume 120. Gulf Professional Publishing, 2003. 3, 4

[7] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah. Signature Verification using a" Siamese" Time Delay Neural Network. In *Advances in Neural Information Processing Systems*, pages 737–744, 1994. 7

[8] A. Cherian, P. Stanitsas, M. Harandi, V. Morellas, and N. Papanikolopoulos. Learning discriminative $\alpha\beta$-divergences for positive definite matrices. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 4280–4289. IEEE, 2017. 6

[9] J. P. Cunningham and Z. Ghahramani. Linear Dimensionality Reduction: Survey, Insights, and Generalizations. *Journal of Machine Learning Research*, 16:2859–2900, 2015. 5

[10] O. Dekel, R. Gilad-Bachrach, O. Shamir, and L. Xiao. Optimal Distributed Online Prediction using Mini-Batches. *Journal of Machine Learning Research*, 13(Jan):165–202, 2012. 1

[11] J. Duchi, E. Hazan, and Y. Singer. Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011. 2, 8

[12] P. T. Fletcher and S. Joshi. Riemannian Geometry for the Statistical Analysis of Diffusion Tensor Data. *Signal Processing*, 87(2):250–262, 2007. 6

[13] M. Harandi, M. Salzmann, and R. Hartley. Joint Dimensionality Reduction and Metric Learning: a Geometric Take. In *Proc. Int. Conference on Machine Learning (ICML)*, pages –, 2017. 5, 7

[14] M. Harandi, M. Salzmann, and F. Porikli. Bregman Divergences for Infinite Dimensional Covariance Matrices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1003–1010, 2014. 7

[15] G. Hinton, N. Srivastava, and K. Swersky. Neural Networks for Machine Learning-lecture 6a-Overview of Mini-Batch Gradient Descent. 2

[16] Z. Huang, J. Wu, and L. Van Gool. Building Deep Networks on Grassmann Manifolds. *arXiv preprint arXiv:1611.05742*, 2016. 4

[17] R. Johnson and T. Zhang. Accelerating Stochastic Gradient Descent using Predictive Variance Reduction. In *Advances in neural information processing systems*, pages 315–323, 2013. 1, 2

[18] D. Kingma and J. Ba. Adam: A Method for Stochastic Optimization. *arXiv preprint arXiv:1412.6980*, 2014. 2, 8

[19] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3D Object Representations for Fine-Grained Categorization. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 554–561, 2013. 7, 8

[20] G. Kylberg. *Kylberg Texture Dataset v. 1.0*. Centre for Image Analysis, Swedish University of Agricultural Sciences and Uppsala University, 2011. 7

[21] M. T. Law, R. Urtasun, and R. S. Zemel. Deep Spectral Clustering Learning. In *International Conference on Machine Learning*, pages 1985–1994, 2017. 7, 8

[22] G. Lebanon et al. *Riemannian Geometry and Statistical Machine Learning*. PhD thesis, 2005. 2

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 6

[24] K.-C. Lee, J. Ho, and D. J. Kriegman. Acquiring Linear Subspaces for Face Recognition under Variable Lighting. *IEEE Transactions on pattern analysis and machine intelligence*, 27(5):684–698, 2005. 6

[25] T. Lin and H. Zha. Riemannian Manifold Learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(5):796–809, 2008. 2

[26] B. Mishra, G. Meyer, S. Bonnabel, and R. Sepulchre. Fixed-rank Matrix Factorizations and Riemannian Llow-rank Optimization. *Computational Statistics*, 29(3-4):591–621, 2014. 7

[27] H. Oh Song, Y. Xiang, S. Jegelka, and S. Savarese. Deep Metric Learning via Lifted Structured Feature Embedding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4004–4012, 2016. 1, 7, 8

[28] X. Pennec, P. Fillard, and N. Ayache. A Riemannian Framework for Tensor Computing. *Int. Journal of Computer Vision*, 66(1):41–66, 2006. 4

[29] N. Qian. On the Momentum Term in Gradient Descent Learning Algorithms. *Neural networks*, 12(1):145–151, 1999. 1

[30] S. J. Reddi, A. Hefny, S. Sra, B. Poczos, and A. J. Smola. On Variance Reduction in Stochastic Gradient Descent and its Asynchronous Variants. In *Advances in Neural Information Processing Systems*, pages 2647–2655, 2015. 1

[31] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016. 2

[32] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015. 8

[33] J. Schmidhuber. Deep Learning in Neural Networks: An Overview. *Neural networks*, 61:85–117, 2015. 1

[34] M. Schmidt, N. Le Roux, and F. Bach. Minimizing Finite Sums with the Stochastic Average Gradient. *Mathematical Programming*, 162(1-2):83–112, 2017. 2

[35] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A Unified Embedding for Face Recognition and Clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015. 7

[36] H. O. Song, S. Jegelka, V. Rathod, and K. Murphy. Deep Metric Learning via Facility Location. In *Computer Vision and Pattern Recognition (CVPR)*, 2017. 8

[37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going Deeper with Convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 8

[38] O. Tuzel, F. Porikli, and P. Meer. Region Covariance: A Fast Descriptor for Detection and Classification. *Computer Vision–ECCV 2006*, pages 589–600, 2006. 7

[39] A. Vedaldi and K. Lenc. Matconvnet – Convolutional Neural Networks for MATLAB. In *Proceeding of the ACM Int. Conf. on Multimedia*, 2015. 8

[40] U. Von Luxburg. A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416, 2007. 8

[41] K. Q. Weinberger and L. K. Saul. Distance Metric Learning for Large Margin Nearest Neighbor Classification. *Journal of Machine Learning Research*, 10(Feb):207–244, 2009. 1, 7

[42] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD birds 200. 2010. 7, 8

[43] Y. Xian, B. Schiele, and Z. Akata. Zero-Shot Learning-The Good, the Bad and the Ugly. *arXiv preprint arXiv:1703.04394*, 2017. 7

[44] S. Xiang, F. Nie, and C. Zhang. Learning a Mahalanobis Distance Metric for Data Clustering and Classification. *Pattern Recognition*, 41(12):3600–3612, 2008. 7

[45] M. D. Zeiler. Adadelta: An Adaptive Learning Rate Method. *arXiv preprint arXiv:1212.5701*, 2012. 2

[46] H. Zhang and P. Koniusz. Zero-shot kernel learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 7

[47] H. Zhang, S. J. Reddi, and S. Sra. Riemannian SVRG: Fast Stochastic Optimization on Riemannian Manifolds. In *Advances in Neural Information Processing Systems*, pages 4592–4600, 2016. 2, 3

[48] P. Zhao and T. Zhang. Accelerating Minibatch Stochastic Gradient Descent using Stratified Sampling. *arXiv preprint arXiv:1405.3080*, 2014. 2