Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

Ottar Passano Hellan

Riemannian Optimization for Deep Learning

Master's thesis in MTFYMA - Industrial Mathematics
Supervisor: Brynjulf Owren
July 2022

NTNU
Norwegian University of
Science and Technology

Ottar Passano Hellan

Riemannian Optimization for Deep Learning

Master's thesis in MTFYMA - Industrial Mathematics
Supervisor: Brynjulf Owren
July 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Mathematical Sciences

NTNU
Norwegian University of
Science and Technology

RIEMANNIAN OPTIMIZATION FOR DEEP LEARNING

OTTAR PASSANO HELLAN

Abstract. Central concepts and structures of Riemannian optimization are presented and discussed to give a self-contained treatment of the Riemannian gradient descent method (RGD). Properties of RGD are discussed and compared with those of euclidean gradient descent, which RGD is a generalization of.

Possible applications of Riemannian optimization and RGD in the field of deep learning are discussed along with considerations one must make in implementations of such methods. Proof-of-concept computational experiments are made using RGD for the fixed-rank matrix manifold and the orthogonal group on CIFAR-10 image classification and a long time-dependence recurrent neural network problem.

Contents

1. Introduction 1
2. Background 2
2.1. Deep Learning 2
2.2. Riemannian Optimization 3
3. Manifolds 4
3.1. Smooth Manifolds 5
3.2. The Tangent Bundle 7
3.3. Riemannian Manifolds 9
4. Riemannian Deep Learning 12
5. Riemannian Gradient Descent 13
6. Manifolds Used 16
6.1. Stiefel Manifold 16
6.2. Low-rank Matrix Manifold 19
7. Numerical Tests 21
7.1. Copying Memory Problem 21
7.2. CIFAR-10 Image Classification 24
8. Conclusion 25
References 27

1. Introduction

Riemannian optimization is a growing field of study with many applications in scientific computation and data analysis, for instance in matrix approximations and independent component analysis[1, Ch. 2.2].

Deep learning algorithms are these days the subject of much research and are showing impressive and progressively better results on a wide range of applications, for instance in image recognition, natural language processing, and biomedical research[2]. These developments have been made possible by improvements in computer hardware, parallelization, improved understanding of the principles of deep

learning, and development of application-specific network architectures[2]. With ever more research on the structures that underlie deep learning problems[3], one might wonder if Riemannian manifold structures can be beneficially exploited for solving hard problems.

In this work, we describe central definitions and results on Riemannian manifolds essential to Riemannian optimization, and try to motivate why they are so necessary. Thereafter, we treat the Riemannian gradient descent method, comparing it with the euclidean gradient descent method. Then, we collect results and operations over two specific manifolds and perform some simple computational experiments using them, to illustrate the use of and some possibilities of Riemannian optimization for deep learning.

2. Background

2.1. Deep Learning.

Deep learning is a collection of methods where a model solution to a problem is stated by the composition of many instances of simple parameterized models. Deep refers to the many *layers* of simpler models that are used and *learning* refers to the process of adapting the parameters of the model to obtain an optimal solution. Deep learning has found excellent results in a wide variety of fields, including computer vision and natural language processing, and their power is — amongst other reasons — thought to stem from the different layers of the network learning different levels of abstraction in the data[2].

Deep learning *models*, also called *networks*, are often stated in the form

$$\mathcal{N}_\theta(x) = y^N, \tag{2.1a}$$

$$y^{n+1} = \mathcal{F}^n_{\theta_n}(y^n) \text{ for } n = 0, ..., N-1, \quad y^0 = x, \tag{2.1b}$$

where each $\mathcal{F}^n_{\theta_n}$ is a parameterized function of a predetermined form. The functions used can vary between the N layers in the model and the canonical example is the fully connected layer

$$y^{n+1} = \sigma\left(W^n y^n + b^n\right),$$

where σ is some predetermined, often sigmoid-like, non-linear *activation function* applied elementwise and the tensors (W^n, b^n) contain the parameters of the layer. Another ubiquitous layer model is the two-dimensional convolutional layer, in constant use in computer vision applications, which performs a local, discrete convolution of an image-like tensor. State of the art models are today extremely complex with a high number of variables and layers. For instance, in 2015 He et al.[4] obtained state of the art results for the time on the ImageNet classification dataset using a network of 152 layers with several million parameters.

A deep learning problem is typically stated as an optimization problem

$$\min_\theta \mathcal{J}(\theta) = \min_\theta \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \mathcal{C}(\mathcal{N}_\theta(x_i), y_i), \tag{2.2}$$

where the *cost function* $\mathcal{J}$ is defined by summing the loss function $\mathcal{C}(z, y)$ over the training dataset

$$\mathcal{D} = \{(x_i, y_i) : i = 1, ..., |\mathcal{D}|\}. \tag{2.3}$$

The cost function used determines how you measure the accuracy of a given prediction $\mathcal{N}_\theta(x)$ and is chosen according to the type of problem and the properties of the

network you wish to encourage. Problems of this form, where you have a dataset consisting of both inputs x_i and targets y_i, are called *supervised learning*.

Networks are typically *trained*, meaning the process of finding solutions to (2.2), by the backpropagation algorithm, where the gradient of the cost function $\mathcal{J}$ is found by propagating reverse-mode automatic differentiation through the layers of the model (2.1) after a prediction has been made and the parameters are thereafter updated with a gradient descent step, or some variation. This process is computationally heavy for a number of reasons. For one, the models are deep and composed of a huge amount of parameters and the datasets are typically very large to obtain optimal results. In addition, reverse-mode automatic differentiation requires the intermediate results $y^k, k = 1, ..., N$, to be stored for computing the reverse pass, increasing memory requirements.

The subject of deep learning is an active field of research, with much activity and many results in the type of layers $\mathcal{F}_\theta$ used, how to best use the datasets $\mathcal{D}$ by *data augmentation*, and how to solve the optimization problem (2.2), to name a few areas of focus.

A more in depth introduction to the subject of deep learning is given in [5].

2.2. **Riemannian Optimization.** *Riemannian optimization* is the study of optimization problems posed on smooth manifolds. Smooth and Riemannian manifolds are defined concretely in later sections, but a serviceable working intuition for smooth manifolds is that of (high-dimensional) surfaces living in $\mathbb{R}^n$ with entirely smooth curves. Riemannian manifolds $(\mathcal{M}, g)$ are smooth manifolds $\mathcal{M}$ with an additional structure g that defines angles and distances on the manifold.

An optimization problem is stated as

$$\min_{x \in \Omega} f(x), \tag{2.4}$$

where $f : \Omega \to \mathbb{R}$ is a *cost function* to be minimized and Ω is the *feasible set*. The goal is to identify a *feasible point* $x^* \in \Omega$ giving the lowest possible value of the cost function $f(x^*)$. The feasible set Ω is often expressed as a topological space X along with a number of *constraints* a feasible point x needs to fulfill, typically expressed by *constraint functions* as $u(x) = 0$. Problems where a function are to be maximized instead of minimized are stated as (2.4) with the negative of the original function f. Optimization problems seldom have analytical solutions, so they are in general solved by numerical algorithms, of which there are a great many, each suited to different types of optimization problems. There is a massive amount of problems that can be stated as optimization problems, so the field of optimization has vast applications.

A comprehensive introduction to the well studied field of optimization over real variables using numerical algorithms is given in the book of Nocedal and Wright[6]. A central concept is the distinction between global and local solutions. A feasible point $x^* \in \Omega$ is a *global solution* if

$$f(x^*) \leq f(x), \, \forall x \in \Omega,$$

and a *local solution* if there exists a neighborhood U of x such that

$$f(x^*) \leq f(x), \, \forall x \in U.$$

Obviously, we would always like to find a global solution, but this is often not possible and in most cases a local solution is the most we can hope for. An exception to this is the class of convex optimization problems, where every local solution is a global

solution. Also important is the concept of optimality conditions. For instance, for $f : \mathbb{R}^n \to \mathbb{R}$ a *necessary first-order optimality condition* is that the gradient $\nabla f(x^*)$ is the zero-vector for all solutions of (2.4). A *sufficient second-order optimality condition* is that all feasible points with positive-definite Hessian matrix $H_{ij} = \frac{\partial^2}{\partial x^i \partial x^j}$ are local minima. Standard algorithms for solving real-valued optimization problems are the well known gradient descent method and Newton's method. Both of these algorithms as well as sufficient and necessary optimality conditions have analogs in the manifold optimization setting, see e.g. [1].

To state a Riemannian optimization problem we specify a Riemannian manifold $(\mathcal{M}, g)$ to take the place of our feasible set Ω and a cost function f to minimize over the manifold. With the differentiability structures given by the smooth manifold $\mathcal{M}$, central optimization concepts like directional derivatives and descent directions are well defined. The added structure of a Riemannian metric allows local approximations using gradients and Hessians, with which we can use local information of the cost function f to build algorithms such as Riemannian gradient descent, trust region, Newton's, and conjugate gradient methods, see e.g.[1], [7].

Riemannian optimization is a fairly young field that has seen increased attention in the last decades. An early development came in 1972 with Luenberger's work on the projected gradient method, where he restated the method as an approximation to a gradient flow over the manifold determined by the problem constraints[8]. This restatement is a continuous analog of the method we today call Riemannian gradient descent and was intended for theoretical use, not computations. The 1994 book of Udrişte gives an early treatment of many methods of Riemannian optimization, like the Riemannian gradient descent method and a Riemannian Newton's method[9]. The 1998 paper of Edelman et al. presents some Riemannian methods on the much used Stiefel manifold and discusses the geometry of the manifold and efficient operations on it in detail[10].

Much work has been done in the last two decades for Riemannian optimization problems defined over matrix manifolds, that is, manifolds contained nicely in a matrix space $\mathbb{R}^{m\times n}$, as these already have a rich theory from both abstract algebra and numerical linear algebra, as well as the many applications from research and industry. The numerical tests in this work are all performed on matrix manifold optimization problems. The subject of optimization over matrix manifolds is covered in great detail in the book [1].

Today, there are several software frameworks in development and available for use offering interfaces to define and solve Riemannian optimization problems featuring a variety of different manifolds, such as Manopt, PyManopt, which features integrated automatic differentiation support for deep learning, and Manopt.jl[11][12][13]. In these frameworks, one chooses a manifold along with a set of manifold operations over it and a solver, all prebuilt or custom, and then one lets the program take care of the rest. No such software has been used in this work, all manifolds and algorithms have been implemented by hand in the deep learning framework PyTorch[14].

3. Manifolds

We now explore the central object of Riemannian optimization, namely Riemannian manifolds. We start with defining smooth manifolds, which is a structure Riemannian manifolds build upon, along with properties and results that are important for defining optimization methods. Thereafter, we define the tangent bundle of smooth

manifolds and try to motivate the fairly abstract definition of it. Finally, we present the Riemannian manifolds as smooth manifolds along with a Riemannian metric, and explain how it leads to important objects such as geodesics and the Riemannian gradient.

The content of this section is mainly a summary of material from the books of Absil et al. and Lee, with adaptations to focus on the parts necessary for optimization and with an attempt at a conceptually simpler presentation[1][15][16].

3.1. **Smooth Manifolds.** A smooth manifold is defined by a set $\mathcal{M}$ along with a collection of mappings, called charts, satisfying some requirements allowing differentiability to be well defined. The sets that allow such a structure are those that locally resemble euclidean space in some sense. For a set to resemble euclidean space locally, we would at least want there to be a one-to-one correspondence between points on a subset of the manifold and points in a subset of euclidean space which respects the nice euclidean topology. Mappings defining such correspondences gives rise to topological manifolds. If we in addition require the mappings to respect differentiability by being smooth, we obtain smooth manifolds.

A d-dimensional *chart* of a set $\mathcal{M}$ is a continuous bijection φ from a subset U of $\mathcal{M}$ to an open subset $\varphi(U)$ of $\mathbb{R}^d$, whose inverse is also continuous. We write (U, φ) for the chart, or simply φ when the domain is not of interest. For $x \in U$, we say that the *coordinates* of x in the chart (U, φ) are

$$\varphi(x) = (\varphi^1(x), ..., \varphi^d(x))^T$$

We call the set U the *coordinate domain* of the chart.

For two charts φ and ψ with overlapping coordinate domains U and V, the *change of coordinates* between the two charts is given by

$$\psi \circ \varphi^{-1} : \varphi(U \cap V) \to \psi(U \cap V).$$

If the change of coordinates and its inverse are smooth as euclidean functions and both $\varphi(U \cap V)$ and $\psi(U \cap V)$ are open in $\mathbb{R}^d$, we say that the charts (U, φ) and (V, ψ) *overlap smoothly*. In this work, a function is *smooth*, *differentiable*, or C^∞, if it has continuous derivatives of all degrees.

An *atlas* of $\mathcal{M}$ into $\mathbb{R}^d$ is a set of d-dimensional charts $(U_\alpha, \varphi_\alpha)$ such that the charts cover $\mathcal{M}$, meaning

$$\bigcup_\alpha U_\alpha = \mathcal{M},$$

and all charts $(U_\alpha, \varphi_\alpha), (U_\beta, \varphi_\beta)$ with $U_\alpha \cap U_\beta \neq \emptyset$ overlap smoothly. An atlas is *maximal* if it contains all *equivalent atlases*. Two atlases $\mathcal{A}_1$ and $\mathcal{A}_2$ of a set $\mathcal{M}$ are equivalent if their charts overlap smoothly. A maximal atlas of $\mathcal{M}$ is also called a *differentiable structure* on $\mathcal{M}$. For a given atlas $\mathcal{A}$ over $\mathcal{M}$, a maximal atlas can be defined by adding any chart over $\mathcal{M}$ which overlaps smoothly with the charts of $\mathcal{A}$ and two atlases are equivalent if and only if they give rise to the same maximal atlas in this manner[1, p. 19].

A differentiable structure $\mathcal{A}^+$ on $\mathcal{M}$ generates a topology over $\mathcal{M}$ in the following way[1, Ch. 3.1.2]. A subset V of $\mathcal{M}$ is open if and only if for any chart (U, φ) in $\mathcal{A}^+$, the set $\varphi(U \cap V)$ is open in the standard topology on $\mathbb{R}^d$. This defines convergence and continuity on the manifold in a natural way.

To prevent smooth manifolds being pathological in some senses, for instance not having unique limit points, two additional conditions on the topology generated by

the differentiable structure of $\mathcal{M}$ are required. These are that the topology generated is both Hausdorff and second-countable. All manifolds considered in this work satisfy these conditions and no further attention will be given to the subject, however these topological properties are defined in the appendix of [1] as well as standard textbooks on topology.

We can now define smooth manifolds as follows:

Definition 3.1 (Smooth manifolds)**.** A *d-dimensional smooth manifold* is a tuple $(\mathcal{M}, \mathcal{A}^+)$ where $\mathcal{M}$ is a set and $\mathcal{A}^+$ is a maximal atlas of $\mathcal{M}$ into $\mathbb{R}^d$, such that the topology induced by $\mathcal{A}^+$ is Hausdorff and second-countable. If an atlas $\mathcal{A}$ of the set $\mathcal{M}$ has maximal atlas $\mathcal{A}^+$, then $\mathcal{A}$ is called an *atlas of the manifold* $(\mathcal{M}, \mathcal{A}^+)$. Charts in the maximal atlas $\mathcal{A}^+$ are called *charts of the manifold* $(\mathcal{M}, \mathcal{A}^+)$.

With a differentiable structure on a manifold, we can define differentiability of functions and directional derivatives on the manifold. In this work we are interested in optimizing real-valued functions $f : \mathcal{M} \to \mathbb{R}$, so we present a simpler definition of real-valued differentiable functions, instead of differentiable functions between arbitrary smooth manifolds.

Definition 3.2 (Differentiable real-valued functions)**.** A function $f : \mathcal{M} \to \mathbb{R}$ defined on a smooth manifold $\mathcal{M}$ is smooth at $x \in \mathcal{M}$ if the composition

$$f \circ \varphi^{-1} : \varphi(U) \to \mathbb{R}$$

is smooth over $\varphi(U) \subset \mathbb{R}^d$ in the standard euclidean sense for any chart (U, φ) with $x \in U$. A function is smooth over $\mathcal{M}$ if it is smooth at every $x \in \mathcal{M}$.

Remark 3.3. Since for any charts (U, φ) and (V, ψ) in the differentiable structure of a smooth manifold with overlapping coordinate domains it holds that $\varphi \circ \psi^{-1}$ is smooth over $\psi(U \cap V)$, if f is differentiable at x through φ, then

$$f \circ \psi^{-1} = f \circ \varphi^{-1} \circ \varphi \circ \psi^{-1} = (f \circ \varphi^{-1}) \circ (\varphi \circ \psi^{-1})$$

is the composition of smooth maps $\varphi \circ \psi^{-1} : \psi(U \cap V) \to \varphi(U \cap V)$ and $f \circ \varphi^{-1} : \varphi(U \cap V) \to \mathbb{R}$ and is therefore smooth. Thus, if f is differentiable at x for one chart, f is differentiable at x for any other chart. This is an important property of smooth manifolds, that differentiability of a function is the same for all charts in the atlas.

A particular form of smooth manifolds is often both practical and conceptually simple, namely embedded submanifolds of $\mathbb{R}^n$[1, Ch. 3.3]. Let $(\mathcal{M}, \mathcal{A}^+)$ and $(\mathcal{N}, \mathcal{B}^+)$ be two manifolds such that $\mathcal{N}$ is a subset of $\mathcal{M}$. If the topology over $\mathcal{N}$ generated by $\mathcal{B}^+$ coincides with the subspace topology induced from the topology over $\mathcal{M}$ given by $\mathcal{A}^+$, then $(\mathcal{N}, \mathcal{B}^+)$ is an *embedded submanifold* of $(\mathcal{M}, \mathcal{A}^+)$. The topology over a set $Y \subseteq X$ is the subspace topology induced from the topological space X, if a set $V \subset Y$ is open in Y if and only if there exists a set $U \subset X$ open in X such that $V = U \cap Y$. If a set $\mathcal{N} \subset \mathcal{M}$ allows a differentiable structure making it an embedded submanifold, then this differential structure is unique. If a function $f : \mathcal{M} \to \mathbb{R}$ is smooth according to 3.2, it is also smooth on any embedded submanifolds of $\mathcal{M}$.

By the Whitney embedding theorem, every smooth n-dimensional manifold can be smoothly embedded into $\mathbb{R}^{2n}$[15, Thm. 6.19]. That is, the manifold can be realized as an embedded submanifold of $\mathbb{R}^{2n}$. Therefore, every smooth manifold we view can be worked with in real coordinates, which is often easier to implement computationally.

3.2. **The Tangent Bundle.** The *tangent space* $T_x\mathcal{M}$ of a manifold $\mathcal{M}$ is in a sense the local euclidean approximation of $\mathcal{M}$ around x. Tangent spaces are vector spaces and can therefore be seen as local first-order approximations of the manifold. Another interpretation is that tangent spaces are the set of directions it is possible to move along. The definition of tangent spaces is non-obvious, however, and is inspired by directional derivatives.

To do optimization on manifolds, an important concept to develop is directional derivatives. In euclidean optimization, directional derivatives define whether or not a search direction is a descent direction, one which reduces the objective function, and this is a concept we would like to have in Riemannian optimization. However, it is not immediately clear what these directions are that define directional derivatives on manifolds. We would like that each smooth curve on $\mathcal{M}$ through x defines such a direction and that the direction gives rise to a directional derivative.

Directional derivatives for C^1 functions over $\mathbb{R}^d$ can be seen as a certain class of linear functionals. For

$$Df(x)[v] = \lim_{t\to 0} \frac{1}{t}\left[f(x+tv) - f(x)\right], \tag{3.1}$$

we can consider the operator

$$v_x : C^1(\mathbb{R}^d) \to \mathbb{R}, \quad f \mapsto Df(x)[v],$$

which is linear over $\mathbb{R}$ in the argument f and obeys the Leibniz rule

$$v_x(fg) = v_x(f)g + fv_x(g). \tag{3.2}$$

We use mappings satisfying these properties to define tangent vectors on a manifold.

In defining directional derivatives on a manifold, since vector addition is not defined over a general manifold $\mathcal{M}$, we cannot use the definition (3.1). However, if we switch out $x + tv$ for a parameterized smooth curve $\gamma : [-1, 1] \to \mathcal{M}$ with $\gamma(0) = x$, we obtain the function

$$f \circ \gamma : [-1, 1] \to \mathbb{R},$$

for which the definition of the standard derivative

$$\left.\frac{d}{dt}\right|_{t=0} f(\gamma(t)) = \lim_{t\to 0} \frac{1}{t}[f(\gamma(t)) - f(x)] \tag{3.3}$$

poses no problem. By smooth curve, we mean here that the coordinate representation $\varphi \circ \gamma$ is smooth in the euclidean sense for any chart covering γ, implying the curve is smooth through all charts in the same way as in remark 3.3. Each smooth curve γ can be parameterized in coordinates by

$$\gamma = \varphi^{-1} \circ g = \varphi^{-1} \circ \varphi \circ \gamma$$

for any chart (U, φ) covering x, at least for some segment (a, b) of the curve including $t = 0$. Then g is a smooth curve in $\varphi(U)$, meaning

$$f \circ \gamma = f \circ \varphi^{-1} \circ g$$

is smooth $(a, b) \to \mathbb{R}$ and the directional derivative (3.3) exists. By the same reasoning as in remark 3.3, this is independent of the chart used and an intrinsic quantity.

A *derivation* at x on a manifold $\mathcal{M}$ is a mapping from the set of differentiable real-valued functions at x to $\mathbb{R}$ that is linear over $\mathbb{R}$ and obeys the Leibniz rule (3.2).

Derivations at x become a vector space with the operations

$$\begin{aligned}(u_x + v_x)(f) &= u_x(f) + v_x(f),\\ (\alpha u_x)(f) &= \alpha u_x(f),\ \forall \alpha \in \mathbb{R}.\end{aligned} \tag{3.4}$$

For a smooth curve $\gamma : [-1, 1] \to \mathcal{M}, \gamma(0) = x$, let $\dot{\gamma}(0)$ denote the mapping

$$\dot{\gamma}(0)f = \left.\frac{d}{dt}\right|_{t=0} f(\gamma(t)) \tag{3.5}$$

for real-valued f differentiable at x. Now, $\dot{\gamma}(0)$ defines a derivation at x. Every such smooth curve γ defines a derivation at x and it turns out that for every derivation at x there is an equivalence class of curves over $\mathcal{M}$ through x giving rise to such a derivation. Two curves, γ_1 and γ_2, through x on $\mathcal{M}$ are here equivalent if the euclidean derivatives coincide, that is

$$\left.\frac{d}{dt}\right|_{t=0} \varphi(\gamma_1(t)) = \left.\frac{d}{dt}\right|_{t=0} \varphi(\gamma_2(t)),$$

for any chart (U, φ) covering x. By considering the functions

$$(f \circ \varphi^{-1}) \circ (\varphi \circ \gamma_i) = f \circ \gamma_i,\ i = 1, 2,$$

with the definition (3.3) of directional derivatives along curves on manifolds and the euclidean chain rule, it is clear that these curves result in the same directional derivatives.

We define the tangent vectors of $\mathcal{M}$ at x as the derivations at x and identify them with the directions that coincide in the equivalence classes. We write $T_x\mathcal{M}$ for the tangent space at x with the vector space structure (3.4) and $v_x \in T_x\mathcal{M}$ for individual tangent vectors, or simply v if x is obvious or not important. Each tangent vector $\dot{\gamma}(0)$ defines a directional derivative at x by

$$Df(x) : T_x\mathcal{M} \to \mathbb{R}, \quad v \mapsto Df(x)[v] = \left.\frac{d}{dt}\right|_{t=0} f(\gamma(t)).$$

By the linearity (3.4) of derivations, it is clear that these mappings are linear, as is the case for euclidean directional derivatives.

By taking the disjoint union of all tangent spaces over $\mathcal{M}$, we obtain the tangent bundle

$$T\mathcal{M} = \bigsqcup_{x \in \mathcal{M}} T_x\mathcal{M},$$

which is also a smooth manifold. Here, the insistence of disjoint union gives the useful identification of any tangent vector to the tangent space of a uniquely determined x. This reinforces the fact that the tangent spaces are distinct vector spaces and tangent vectors of different points in $\mathcal{M}$ can not be directly compared or added.

We can define a basis for $T_x\mathcal{M}$ in the following way[1, p. 35]. For a chart (U, φ) covering x, let

$$\gamma_i(t) = \varphi^{-1}(\varphi(x) + te_i),$$

with e_i being the i-th coordinate vector of $\mathbb{R}^d$. Then $(\dot{\gamma}_1(0), ..., \dot{\gamma}_d(0))$ forms a basis of $T_x\mathcal{M}$, where $\dot{\gamma}_i(0)$ is defined as in (3.5). This also shows that the dimension of the tangent space of a d-dimensional manifold is d.

For embedded submanifolds of $\mathbb{R}^n$, there is a simple construction of tangent spaces using curves in $\mathbb{R}^n$ that stay on the manifold $\mathcal{M}$[1, Ch. 3.5.7]. By viewing each

tangent vector as embedded in a subspace of $\mathbb{R}^n$, we can define the tangent space of $\mathcal{M}$ at x simply as

$$T_x\mathcal{M} = \left\{ \left.\frac{d}{dt}\right|_{t=0} \gamma(t) : \gamma \in \Gamma_x \right\},$$

where Γ_x is the set of smooth curves passing through x at $t = 0$. For matrix manifolds, this construction is typically simplest and allows an obvious way of storing tangent vectors digitally for numerical computations. However, these vectors are then stored with much higher dimensionality than the underlying manifold, sacrificing efficiency for practicality.

3.3. **Riemannian Manifolds.** A Riemannian manifold is a smooth manifold along with an inner product over the tangent bundle, defining angles and distances, called a Riemannian metric. All smooth manifolds admit a Riemannian metric, but not uniquely[16, Prop. 2.4].

A *Riemannian metric* is a smoothly varying inner product over the tangent bundle. For each $x \in \mathcal{M}$,

$$g_x : T_x\mathcal{M} \times T_x\mathcal{M} \to \mathbb{R}$$

is an inner product over $T_x\mathcal{M}$ and the restriction of the Riemannian metric g onto $T_x\mathcal{M}$. We write this inner product as either $g_x(v_1, v_2)$ or $\langle v_1, v_2 \rangle_x$, for $v_1, v_2 \in T_x\mathcal{M}$. Given a basis $(\xi_1, ..., \xi_d)$ of $T_x\mathcal{M}$, we represent each tangent vector as $v = (v^1, ..., v^d)$ and the Riemannian metric g_x as a symmetric, positive-definite matrix $G = g_{ij}$. Then we can write $g_x(u, v)$ in terms of our basis as

$$g_x(u, v) = \sum_{i,j} g_{ij} u^i v^j = g_{ij} u^i v^j,$$

with the last expression written using the Einstein summation-convention[17]. We say that g is smoothly varying in the sense that each component g_{ij} of the matrix G is a differentiable real-valued function over x, as defined in definition 3.2, for the entirety of the manifold $\mathcal{M}$.

The Riemannian metric g induces a norm over $T_x\mathcal{M}$ by

$$\|v\|_x = \sqrt{g_x(v, v)}.$$

Along with this norm and the inner product over the tangent bundle, the Riemannian metric defines the *length of a curve* $\gamma : [0, 1] \to \mathcal{M}$ by

$$L(\gamma) = \int_0^1 \sqrt{g_{\gamma(t)}(\dot{\gamma}(t), \dot{\gamma}(t))} \mathrm{d}t,$$

which in turn defines the *Riemannian distance* between two connected points $x, y \in \mathcal{M}$ by

$$\mathrm{dist}(x, y) = \inf \{L(\gamma) : \gamma(0) = x,\ \gamma(1) = y\}.$$

For connected Riemannian manifolds, dist fulfills the requirements of a metric and thus defines a metric space[1, p. 46]. Note that metrics and Riemannian metrics are different types of objects and only similarly named. Metrics measure distances between points of a set and Riemannian metrics define lengths of and angles between tangent vectors of a manifold.

If $\mathcal{N}$ is an embedded submanifold of a Riemannian manifold $(\mathcal{M}, g^{\mathcal{M}})$, then $\mathcal{N}$ inherits a Riemannian metric from $(\mathcal{M}, g^{\mathcal{M}})$ by the restriction of $g^{\mathcal{M}}$ to the tangent

bundle of $\mathcal{N}$, as a subset of the tangent bundle of $\mathcal{M}$. That is, $g^{\mathcal{N}}_x : T_x\mathcal{N} \times T_x\mathcal{N} \to \mathbb{R}$ is an inner product given by

$$g^{\mathcal{N}}_x(v_1, v_2) = g^{\mathcal{M}}_x(v_1, v_2),\ \forall v_1, v_2 \in T_x\mathcal{N}.$$

Such a Riemannian manifold $(\mathcal{N}, g^{\mathcal{N}})$ is called a *Riemannian submanifold* of $(\mathcal{M}, g^{\mathcal{M}})$.

On Riemannian manifolds, there is a family of curves that are in a sense the most natural curves in the direction of a given tangent vector. These are called *geodesic curves*, or *geodesics* for short. In euclidean space the most natural curves to follow are the straight line, curves through $x \in \mathbb{R}^n$ of the form

$$\gamma(t) = x + tv$$

for $v \in T_x\mathbb{R}^n \simeq \mathbb{R}^n$. We cannot immediately define these curves over a general manifold $\mathcal{M}$, but we can try to generalize some properties of straight lines, namely that straight lines are the shortest curves between two points and that straight lines have zero acceleration.

In generalizing the property of zero acceleration, we might want to say that the tangent $\dot{\gamma}(t)$ is constant for a geodesic curve γ, but since $T_{\gamma(t)}\mathcal{M}$ are not the same vector space for different values of t, they are not immediately comparable. To continue with this notion we need a way to compare vectors of different tangent spaces and this is done with a structure called a *connection*, specifically the *Levi-Civita-connection*. The theory and formulation of connections is too complicated to include in this work, so we simply state that for a given coordinate frame $(x^1, ..., x^d)$, the Riemannian metric g defines a tensor Γ^k_{ij} called the *Christoffel symbols*, which allows the statement of the *geodesic equation*

$$\frac{d^2}{dt^2}x^k(t) + \Gamma^k_{ij}(x(t))\frac{d}{dt}x^i(t)\frac{d}{dt}x^j(t) = 0, \tag{3.6}$$

the solution curves of which are geodesic curves. The book by Lee gives a thorough treatment of connections and geodesics, including the derivation of the geodesic equation[16].

For Riemannian submanifolds of $\mathbb{R}^n$, we can define the tangential acceleration of a curve $\gamma : [a, b] \to \mathcal{M}$ to be the orthogonal projection of the acceleration at each time t to the local tangent space $T_{\gamma(t)}\mathcal{M}$. Geodesic curves over $\mathcal{M}$ are then those curves that have zero tangential acceleration. That is, geodesic curves are solutions of the second order differential equation

$$\pi_{T_{\gamma(t)}\mathcal{M}}\left(\frac{d^2}{dt^2}\gamma(t)\right) = 0, \tag{3.7}$$

where $\pi_{T_x\mathcal{M}}$ is the orthogonal projection onto $T_x\mathcal{M}$.

Through classical theorems of ordinary differential equations (ODE) like the Picard-Lindelöf theorem, both initial value problems of (3.6) and (3.7) have unique solutions at least locally, see e.g. [18].

We can also generalize the concept of straight lines being the shortest curves between two points. It can be shown that all solutions of the calculus of variations problem

$$\min_{\gamma:[0,1]\to\mathcal{M}} L(\gamma) \text{ such that } \gamma(0) = x, \gamma(1) = y,\ \gamma \text{ is smooth}, \tag{3.8}$$

are geodesic curves satisfying (3.6) and that all geodesic curves are local solutions of (3.8) [16, Ch. 6]. A curve is a local solution of (3.8) if $L(\gamma)$ increases if γ is changed to another smooth curve arbitrarily alike, to put it roughly.

The geodesic curves define a mapping over Riemannian manifolds called the *exponential mapping*. This mapping takes tangent vectors $v_x \in T\mathcal{M}$ and returns a point $y \in \mathcal{M}$ such that $y = x(1)$ for a geodesic curve with the initial conditions

$$x(0) = x, \quad \frac{d}{dt}x(t) = v. \tag{3.9}$$

By ODE-theory there is a neighborhood $U \subset T_x\mathcal{M}$ of the zero tangent $0_x \in T_x\mathcal{M}$ where such a solution exists for time $t = 1$.

Definition 3.4. The *exponential mapping* at x is given by

$$\exp_x : U_x \to \mathcal{M}, \quad \exp_x(v_x) = x(1), \tag{3.10}$$

where x is a solution of (3.6) with initial conditions (3.9) and $U_x \subset T_x\mathcal{M}$ is a neighborhood of the zero tangent 0_x where such a solution exists for all $v_x \in U_x$. The exponential mapping over $T\mathcal{M}$ is taken to be

$$\exp : \bigsqcup_{x \in \mathcal{M}} U_x \to \mathcal{M}, \quad \exp(v_x) = \exp_x(v_x). \tag{3.11}$$

It can be shown that the exponential mapping is smooth, meaning that tangent vectors close to each other are mapped to points close to each other[1, Ch. 5.4]. Many Riemannian metrics can be defined over each smooth manifold and the choice of metric determines the solution of the geodesic equation (3.6). Thus, the choice of Riemannian metric determines which curves over the manifold are the natural extensions of straight lines.

Another important concept allowed by the existence of a Riemannian metric is the Riemannian gradient. In euclidean space, the gradient of a real-valued function f is usually defined in terms of partial derivatives as the vector

$$\nabla f(x) = \left(\frac{\partial f}{\partial x^1}(x), ..., \frac{\partial f}{\partial x^n}(x)\right)^T,$$

a definition which does not make sense for a Riemannian manifold. However, an equivalent definition is that the gradient of a real-valued function f at $x \in \mathbb{R}^n$ is the vector $\nabla f(x)$ for which directional derivatives are given by the inner product

$$Df(x)[v] = \nabla f(x)^T v.$$

This is to say that $\nabla f(x)$ is the vector in $T_x\mathbb{R}^n \simeq \mathbb{R}^n$ to which the corresponding linear functional $v \mapsto \nabla f(x)^T v$ is the operation of taking directional derivatives. This is simple to generalize to Riemannian manifolds, since with the Riemannian metric over $T_x\mathcal{M}$ we have an inner product that allows us to define the directional derivative functional with a tangent vector in $T_x\mathcal{M}$. The tangent space is finite-dimensional and therefore a Hilbert space, so by the Riesz representation theorem for Hilbert spaces, see e.g. [19], there is a unique element of the $T_x\mathcal{M}$ representing any bounded, linear functional over $T_x\mathcal{M}$. Since linear functions on finite-dimensional vector spaces are continuous, we can thus define the Riemannian gradient as follows:

Definition 3.5. Let $(\mathcal{M}, g)$ be a Riemannian manifold and $f : \mathcal{M} \to \mathbb{R}$ be differentiable around x according to definition 3.2. Then the Riemannian gradient of f at x is the unique element $\nabla f(x)$ of $T_x\mathcal{M}$ such that

$$Df(x)[v] = g_x(\nabla f(x), v), \ \forall v \in T_x\mathcal{M}. \tag{3.12}$$

It is simple to see that the definition of the Riemannian gradient coincides with the euclidean gradient when the Riemannian manifold in question is $\mathbb{R}^n$ with the standard euclidean metric. In addition, we can see by the Cauchy-Schwarz inequality, considering unit-norm tangent vectors, that

$$|Df(x)[v]| = |g_x(\nabla f(x), v)| \leq \|\nabla f(x)\|_x = \left|g_x(\nabla f(x), \widehat{\nabla f(x)})\right| = \left|Df(x)[\widehat{\nabla f(x)}]\right|,$$

where $\widehat{\nabla f(x)} = \nabla f(x)/\|\nabla f(x)\|_x$, so the Riemannian gradient gives the direction of steepest descent and the rate of steepest descent, just like the euclidean gradient does in $\mathbb{R}^n$.

For Riemannian submanifolds of $\mathbb{R}^n$, there is a simple way to compute Riemannian gradients using the euclidean gradient. If $\hat{f} : \mathbb{R}^n \to \mathbb{R}$ is a smooth extension of $f : \mathcal{M} \to \mathbb{R}$, and by [15, Lemma 5.34] such a smooth extension exists, then the Riemannian gradient $\nabla f(x)$ is given by

$$\nabla f(x) = \pi_{T_x\mathcal{M}} \nabla^E \hat{f}(x), \tag{3.13}$$

where ∇^E denotes the euclidean gradient and $\pi_{T_x\mathcal{M}}$ is the orthogonal projection of $\mathbb{R}^n$ into $T_x\mathcal{M}$ with regard to the euclidean metric $\langle\cdot,\cdot\rangle^E$. This holds since $\nabla f(x) = \pi_{T_x\mathcal{M}}(\nabla^E \hat{f}(x)) + r$ with $r \perp T_x\mathcal{M}$, so

$$\begin{aligned} Df(x)[v] = D\hat{f}(x)[v] &= \langle \nabla^E \hat{f}(x), v\rangle^E = \langle \pi_{T_x\mathcal{M}}(\nabla^E \hat{f}(x)) + r, v\rangle^E \\ &= \langle \pi_{T_x\mathcal{M}}(\nabla^E \hat{f}(x)), v\rangle^E + \langle r, v\rangle^E = \langle \pi_{T_x\mathcal{M}}(\nabla^E \hat{f}(x)), v\rangle^E \\ &= g_x(\pi_{T_x\mathcal{M}} \nabla^E \hat{f}(x), v), \end{aligned}$$

making $\pi_{T_x\mathcal{M}} \nabla^E \hat{f}(x)$ the Riemannian gradient of f at x. In our numerical experiments, the manifolds and tangent vectors are stored embedded in $\mathbb{R}^{m\times n}$ with cost functions defined for any $X \in \mathbb{R}^{m\times n}$, so we use this method of computing Riemannian gradients.

4. Riemannian Deep Learning

The use of Riemannian optimization in deep learning is a fairly new field and not yet well explored. With the high computational cost of modern deep learning algorithms combined with the generally higher scaling complexity of Riemannian optimization algorithms, this is maybe not so strange, as such methods might struggle to compete with other promising research directions. On the other hand, with the scale of modern deep learning networks, using the geometry of problems in a smart way might have great potential. If this is the case, Riemannian optimization is a natural tool to explore.

While there has been much work in developing algorithms that respect the geometry and symmetries of a problem, not much has been done with manifold structures in mind. In 2016, Arjovsky et al. proposed addressing the vanishing/exploding gradients problem by constraining a recurrent neural network to have a unitary hidden-to-hidden matrix, achieving great results on several difficult tasks compared to the much used long short-term memory model[20]. The approach was here to use euclidean optimization over a specific factorization of the matrix, not using manifold-based algorithms over the required parameter manifold. Massart et al. present a way to optimize this same model using a coordinate-descent variation of Riemannian gradient descent method, a method intrinsic to the geometry of the search space[21]. With an

aim to develop more effective momentum-based methods for unitary-constrained matrices, Li et al. develop efficient Cayley retraction-based stochastic gradient descent and Adam methods[22].

Many models in deep learning can be stated as having parameters constrained to Riemannian manifolds, for instance fully connected linear layers of fixed, low rank, or two-dimensional convolutional layers, which are parameterized over a space of Toeplitz-like matrices[23]. Such models are seldom trained with Riemannian optimization techniques however. The application of Riemannian optimization to such models is an interesting topic to investigate. It is clear that the computational overheads of Riemannian algorithms do not always justify the added benefit of their use, but if a problem is well suited to a given manifold, and the manifold has nice operations that are not prohibitively expensive, then it should be possible to benefit from applying Riemannian methods to the problem.

It is also worth noting, that Riemannian optimization algorithms are defined in terms of operations on Riemannian manifolds, and these can not be immediately implemented in computations[1, p. 22]. We only have hardware to work with real numbers and integers and only trivial, euclidean manifolds can be accurately represented by these. Thus, all operations we build our algorithms on will have to be carefully designed to lessen the impact of such concerns. For instance, a point on an embedded submanifold of $\mathbb{R}^n$ can in general not be represented as lying exactly on the manifold, due to the limits of machine precision in floats. By representing our manifolds in ways that minimize such problems, for instance by appropriate matrix factorizations or quotient manifolds, we can obtain more efficient algorithms where we have to compensate less for the errors we cause.

5. Riemannian Gradient Descent

Riemannian gradient descent (RGD) is a first-order method used to solve the optimization problem

$$\min_{x \in \mathcal{M}} f(x), \tag{5.1}$$

where $\mathcal{M}$ is a Riemannian manifold and $f : \mathcal{M} \to \mathbb{R}$ is differentiable over $\mathcal{M}$. RGD is a generalization of gradient descent from the euclidean to the Riemannian optimization setting. Gradient descent is the simplest iterative gradient-based optimization algorithm and has in recent years found success in optimizing deep neural network parameters through the backpropagation algorithm[2].

In gradient descent, we optimize a variable x in the euclidean space $\mathbb{R}^d$ by successively translating it some distance η_k in the direction of steepest descent by vector addition. The direction of steepest descent is given by the negative of the standard euclidean gradient,

$$\nabla^E f(x) = \left(\tfrac{\partial f}{\partial x^1}(x), \quad \ldots \quad \tfrac{\partial f}{\partial x^d}(x)\right)^T,$$

resulting in the sequence

$$x_{k+1} = x_k - \eta_k \nabla^E f(x_k),$$

for some starting point $x_0 \in \mathbb{R}^d$, converging to a stationary point of the problem. More details on gradient descent and other numerical optimization algorithms in $\mathbb{R}^d$ can be found in [6].

This algorithm depends on the geometry of $\mathbb{R}^d$ in several ways, the most important of which is the ability to move along straight lines using the vector addition operation. In the Riemannian manifold setting, this is not possible, so we have to find some other

way to update our search variable x. The natural generalization of straight lines on Riemannian manifolds is geodesic curves, which we can parameterize with the tangent bundle $T\mathcal{M}$ using the exponential mapping. However, the computation of geodesic curves is non-trivial, such that in most cases only approximations of geodesic curves are feasible. In general, computing a geodesic curve involves solving either a second order ODE or a calculus of variations problem over the constraint manifold.

In addition, we need an alternative definition of the direction of steepest descent. For a general manifold, the definition of the euclidean gradient does not make sense, since we do not have a global basis $\{x_1, ..., x_d\}$ and the partial derivatives are defined using euclidean vector subtraction, which is not defined over $\mathcal{M}$. For embedded submanifolds $\mathcal{M}$ of $\mathbb{R}^d$ and objective functions f that are smoothly extensible to $\mathbb{R}^d$, we can use the euclidean gradient to find a descent direction in the ambient space $\mathcal{M}$ is contained in. However, this will generally not give a tangent vector representing an allowed direction of movement along the manifold, making the sequence $(x_k)_{k\in\mathbb{N}}$ leave the search manifold. We therefore use the Riemannian gradient of f, see definition 3.5, which defines a unique tangent vector $\nabla f(x) \in T_x\mathcal{M}$ as an allowable descent direction.

We can now define RGD as follows:

Definition 5.1 (Riemannian gradient descent)**.** Let $\mathcal{M}$ be a Riemannian manifold with metric g and $f : \mathcal{M} \to \mathbb{R}$ a function to be minimized, which is differentiable over $\mathcal{M}$. Denote by $\exp_x(v) : T_x\mathcal{M} \to \mathcal{M}$ the exponential mapping of $\mathcal{M}$ at x, as defined in 3.4. For a sequence $\{\eta_0, \eta_1, ...\} \subset \mathbb{R}$ of stepsizes and initial guess $x_0 \in \mathcal{M}$, Riemannian gradient descent is defined by the recursion

$$x_{k+1} = \exp_{x_k}(-\eta_k \nabla f(x_k)), \tag{5.2}$$

where $\nabla f(x)$ denotes the Riemannian gradient of f at x.

To alleviate the computational burden of computing the exponential mapping $\exp_{x_k}(-\eta_k \nabla f(x_k))$, we can allow ourself to use curves that only approximate geodesics. The curves we are interested in are given by retraction mappings. Though retractions are an established tool of topology, we use in this work a narrower definition which is more suitable for Riemannian optimization, in the vein of Absil, Mahony, and Sepulchre[1]. We state the definition used in [24], which fits our needs well.

Definition 5.2 (Retraction)**.** A retraction on a manifold $\mathcal{M}$ is a smooth mapping $R : T\mathcal{M} \to \mathcal{M}$ with the following properties. Let $R_x : T_x\mathcal{M} \to \mathcal{M}$ denote the restriction of R to $T_x\mathcal{M}$.

(i) $R_x(0_x) = x$, where 0_x is the zero vector in $T_x\mathcal{M}$;
(ii) The differential of R_x at 0_x is the identity map.

Retractions approximate the exponential mapping of a Riemannian manifold in the sense that

$$\text{dist}(\exp_x(v), R_x(v)) = \mathcal{O}\left(||v||^2\right),$$

where dist and $\|\cdot\|$ denotes the distance and norm induced by the Riemannian metric[25]. In this way, retractions can be viewed as first order approximations of the exponential mapping.

We can now define a generalized form of gradient descent, where the choice of retraction also defines the iterates produced.

Definition 5.3 (Riemannian gradient descent with retractions)**.** Let $\mathcal{M}$ be a smooth manifold, $R : T\mathcal{M} \to \mathcal{M}$ a retraction over $\mathcal{M}$, and $f : \mathcal{M} \to \mathbb{R}$ a function to be minimized, which is differentiable over $\mathcal{M}$. For a sequence $\{\eta_0, \eta_1, ...\}$ of stepsizes and initial guess x_0, Riemannian gradient descent with retractions is defined by the recursion

$$x_{k+1} = R_{x_k}(-\eta_k \nabla f(x_k)), \tag{5.3}$$

where $\nabla f(x)$ denotes the Riemannian gradient of f at x.

The choice of retraction is not always obvious, but for some cases the question is simpler. Retractions do not need a Riemannian metric to be defined, but the Riemannian metric defines the exponential mapping uniquely, which is itself a retraction. If the computational cost of the exponential mapping is reasonable, it should be the preferred retraction for RGD, as it is fully intrinsic to the Riemannian manifold we optimize over.

If the manifold in question, $\mathcal{M}$, is a submanifold of euclidean space, $\mathbb{R}^d$, an intuitive retraction can be defined by

$$R_x(v) = \mathcal{P}_{\mathcal{M}}(x + v),$$

where $\mathcal{P}_{\mathcal{M}}$ denotes the projection operator onto $\mathcal{M}$ minimizing euclidean distance. This retraction is locally well defined[25]. These retractions can be practical for matrix manifolds, on which much work has been done and which is covered in detail in[1]. Note that the embedding of $\mathcal{M}$ into $\mathbb{R}^d$ is not unique, and the choice of embedding will determine the behaviour of such retractions.

The convergence of the sequence $(x_k)_{k\in\mathbb{N}}$ of RGD iterates can be stated in both a local and a global sense.

Global convergence of RGD to stationary points has been shown, in the sense that the method will converge to a stationary point regardless of initialization. If f is bounded from below on $\mathcal{M}$ and for all iterates x_k the Lipschitz-like condition

$$|f(R_{x_k}(v)) - [f(x_k) + \langle v, \nabla f(x_k)\rangle]| \leq C \|v\|^2, \forall v \in T_{x_k}\mathcal{M},$$

holds, then there is a constant step size η for which the method will produce a point $x_N \in \mathcal{M}$ with which $f(x_N) \leq f(x_0)$ and $\|\nabla f(x_N)\| < \varepsilon$ in $\mathcal{O}\left(\frac{1}{\varepsilon^2}\right)$ iterations, regardless of initial guess x_0[24]. This asymptotic bound also holds for euclidean gradient descent and other step size selection strategies, and is tight[26], which means better global convergence than $\mathcal{O}\left(\frac{1}{\varepsilon^2}\right)$ for RGD is impossible without further restrictions on f, since euclidean gradient descent coincides with RGD when $\mathcal{M}$ is $\mathbb{R}^d$.

Euclidean gradient descent can at most be guaranteed to find a stationary point of f, not a minimum, for general non-linear optimization[27, p. 32]. Thus, convergence to a stationary point on the manifold $\mathcal{M}$ is the most we can be sure of, and finding a local minimum can not be guaranteed. This does not mean the method is useless, only that one must consider the possibility of the algorithm returning saddle points that are not solutions. Since saddle points are non-stable equilibrium points of the gradient descent iterative system, it has been shown that the use of stochastic gradient descent can help escape saddle points in certain situations[28].

Locally, RGD has linear convergence rate around strict local minima[1, Thm. 4.5.6]. That is, if $(x_k)_{k\in\mathbb{N}}$ are converging to a strict local minimum x^* and f is twice continuously differentiable, then there exists $r < 1$ and $K \in \mathbb{N}$ such that

$$f(x_{k+1}) - f(x^*) \leq r(f(x_k) - f(x^*))$$

for all $k > K$. This local convergence rate is the same as for euclidean gradient descent[27, p. 35], but holds there also in terms of distance from the minimizer $\|x_k - x^*\|$. An almost complete proof of the corresponding result for RGD is given in [9, p. 266].

These results hold when RGD is performed with any retraction, not just the exponential mapping. Therefore, computationally competitive methods avoiding prohibitive exponential mappings are theoretically grounded.

The use of RGD in the stochastic setting has also been studied. Under mild conditions, RGD has been shown[29] to converge almost surely in both function value and gradient norm to stationary points, that is

$$\Pr\left(\lim_{n\to\infty} f(x_n) = f^*\right) = \Pr\left(\lim_{n\to\infty} \|\nabla f(x_n)\| = 0\right) = 1.$$

This holds for RGD with the exponential mapping and with other retractions. Convergence in the stochastic case is beneficial for Riemannian deep learning for the same reasons stochastic gradient descent is used in deep learning in general, for instance the gradient noise introduced by random selection of mini-batches.

6. Manifolds Used

6.1. **Stiefel Manifold.** The Stiefel manifold is well developed and has many retractions available in the literature. We base our experiments over the Stiefel manifold on expressions presented by Edelman et al.[10]. We present results from this paper on the structure of the manifold and some retractions, along with derivations of select results to illustrate how Riemannian manifolds can be approached. We also present some more Cayley transform-based retractions.

The *Stiefel manifold* is the set of matrices

$$\mathrm{St}(m,r) = \left\{X \in \mathbb{R}^{m\times r} : X^T X = I_r\right\}. \tag{6.1}$$

This defines the set of orthonormal r-frames. This definition of the Stiefel manifold is sometimes called the *compact* Stiefel manifold, the non-compact Stiefel manifold being

$$\mathbb{R}^{m\times r}_* = \left\{X \in \mathbb{R}^{m\times r} : X \text{ has full rank } \right\}.$$

The Stiefel manifold gains a differentiable structure as an embedded submanifold of $\mathbb{R}^{m\times r}$. The most common Riemannian metrics on the Stiefel manifold are the standard euclidean inner product inherited from $\mathbb{R}^{m\times r}$, given by

$$g(v_1, v_2) = \langle v_1, v_2\rangle_F = \mathrm{tr}(v_1^T v_2),$$

and the canonical metric, which is very similar, but is changed slightly to give equal weight to all degrees of freedom on the manifold.

Several special cases of the Stiefel manifold are interesting manifolds in themselves. For instance,

$$\mathrm{St}(m,1) = \left\{x \in \mathbb{R}^m : x^T x = 1\right\} = S^{m-1},$$

the $m-1$-dimensional hypersphere. We also have

$$\mathrm{St}(m,m) = \left\{X \in \mathbb{R}^{m\times m} : X^T X = I\right\} = \mathcal{O}(m),$$

the set of orthogonal matrices of size $m \times m$. This manifold also has a group structure with standard matrix multiplication and is ubiquitous in numerical linear algebra, for instance in QR-factorization and in orthonormalization of algorithms for numerical stability.

By the simple expression for $\mathrm{St}(m,r)$ given by $X^TX = I_r$, we can find an expression for the tangent space of $\mathcal{M} = \mathrm{St}(m,r)$ in a simple way by considering the curves over $\mathcal{M}$ through x embedded in $\mathbb{R}^{m\times r}$. For $\gamma(t) = X(t) \subset \mathcal{M}$, we must have that $\frac{d}{dt}\left(X(t)^TX(t)\right) = 0$ at all times, so

$$0 = \frac{d}{dt}\Big|_{t=0} X(t)^TX(t) = \dot{X}(t)^TX(t) + X(t)^T\dot{X}(t)$$

implies

$$T_X\mathcal{M} = \left\{Y \in \mathbb{R}^{m\times r} : (X^TY)^T = -X^TY\right\}.$$

For any $X \in \mathrm{St}(m,r)$ we can choose an orthogonal completion $X_\perp \in \mathrm{St}(m, m-r)$ of X and have the decomposition

$$\mathbb{R}^{m\times r} = X\mathbb{R}^{r\times r} + X_\perp\mathbb{R}^{m-r\times r}, \tag{6.2}$$

which is an orthogonal decomposition with the Riemannian metric

$$g_e(A,B) = \langle A,B\rangle_F = \mathrm{tr}(A^TB) = \sum_{i,j} A_{ij}B_{ij}$$

over euclidean space $\mathbb{R}^{m\times r}$. We can then write any $Y \in T_X\mathcal{M}$ as $Y = XA + X_\perp B$ and find

$$X^TY = X^TXA + X^TX_\perp B = X^TXA = A,$$

so

$$T_X\mathcal{M} = X\left\{A \in \mathbb{R}^{r\times r} : A^T = -A\right\} + X_\perp\mathbb{R}^{m-r\times r}. \tag{6.3}$$

It can be shown that the orthogonal complement of $T_X\mathcal{M}$, called the *normal space* of $\mathcal{M}$ at X and written $N_X\mathcal{M}$, is given by

$$N_X\mathcal{M} = X\left\{A \in \mathbb{R}^{r\times r} : A^T = A\right\}$$

and that

$$\pi_{N_X}(Y) = X\,\mathrm{sym}(X^TY)$$

defines an orthogonal projection onto $N_X\mathcal{M}$[10, p. 5]. Here, sym denotes the symmetric part of an $n\times n$-matrix

$$A = \mathrm{sym}(A) + \mathrm{skew}(A) = \frac{1}{2}\left(A + A^T\right) + \frac{1}{2}\left(A - A^T\right). \tag{6.4}$$

Thus, an orthogonal projection onto $T_X\mathcal{M}$, with the Frobenius metric, is given by

$$\pi_{T_X}(Y) = X - X\,\mathrm{sym}(X^TY). \tag{6.5}$$

With the orthogonal decomposition (6.2), we can write the euclidean metric as

$$\begin{aligned} g_e(Y,Z) &= \mathrm{tr}\left(Y^TZ\right) = \mathrm{tr}\left((XA + X_\perp B)^T(XC + X_\perp D)\right) = \mathrm{tr}\left(A^TC + B^TD\right) \\ &= \mathrm{tr}(A^TC) + \mathrm{tr}(B^TD). \end{aligned}$$

However, the matrices A and C are skew-symmetric, so

$$\mathrm{tr}(A^TC) = 2\sum_{i<j} A_{ij}C_{ij}$$

and each degree of freedom in the first part of the decomposition (6.3) is counted twice. The canonical metric is defined as

$$g_c(Y,Z) = \frac{1}{2}\mathrm{tr}(A^TC) + \mathrm{tr}(B^TD), \text{ where } Y = XA + X_\perp B,\ Z = XC + X_\perp D,$$

to address this concern of double-weighting. It can be shown[10, Ch. 2.4.1] that the above expression is equivalent to

$$g_c(Y,Z) = \operatorname{tr}\left(Y^T(I - \frac{1}{2}XX^T)Z\right). \tag{6.6}$$

In the case of the orthogonal group $\mathcal{O}(m) = \mathrm{St}(m,m)$ the $X_\perp$ factor falls away and the two metrics are merely scaled versions of each other.

The geodesics of the Stiefel manifold can be expressed using the *matrix exponential*. Using the euclidean metric, an expression for the geodesic curves is given by

$$X(t) = \begin{pmatrix} X(0) & \dot{X}(0) \end{pmatrix} e^{t\begin{pmatrix} A & -S(0) \\ I & A \end{pmatrix}} I_{2r,r} e^{-At},$$

where $A = X(0)^T\dot{X}(0)$, $S(0) = \dot{X}(0)^T\dot{X}(0)$, and $I_{2r,r} = \begin{pmatrix} I_r \\ 0 \end{pmatrix} \in \mathbb{R}^{2r\times r}$. The matrix exponential $\mathbb{R}^{n\times n} \to \mathbb{R}^{n\times n}$ is defined by using the power series representation of e^x to allow matrix-valued arguments, by

$$e^X = \sum_{k=0}^{\infty} \frac{1}{k!} X^k. \tag{6.7}$$

Computing the matrix exponential by the infinite sum (6.7) is not possible and the question of how to best implement the mapping is complicated. In this work we use the implementation of Scipy[30], which is based on a scaled Padé-approximation[31].

In the case of the orthogonal group $\mathcal{O}(m) = \mathrm{St}(m,m)$, a simpler expression is given by

$$X(t) = X(0)e^{At},$$

where $A = X(0)^T\dot{X}(0)$, allowing us to formulate the exponential mapping as

$$\exp_X(Y) = Xe^{X^TY} = e^{YX^T}X. \tag{6.8}$$

We use this expression in our tests with the orthogonal group.

The matrix exponential is very heavy to compute accurately and therefore retractions over the Stiefel manifold have been developed, for instance ones using the *Cayley transform*

$$X \mapsto \operatorname{cay}(X) = \left(I - \frac{1}{2}X\right)^{-1}\left(I + \frac{1}{2}X\right).$$

With the Neumann series $(I-X)^{-1} = \sum_{k=0}^{\infty} X^k$ we have that

$$\begin{aligned}\left(I - \frac{1}{2}X\right)^{-1}\left(I + \frac{1}{2}X\right) &= \sum_{k=0}\left(\frac{1}{2}X\right)^k\left(I + \frac{1}{2}X\right) = \left(\frac{1}{2}X\right)^0 + 2\sum_{k=1}^{\infty}\left(\frac{1}{2}X\right)^k \\ &= I + X + \frac{1}{2}X^2 + \mathcal{O}\left(\|X\|^3\right),\end{aligned}$$

so

$$e^X - \operatorname{cay}(X) = \mathcal{O}\left(\|X\|^3\right).$$

Therefore, a retraction over $\mathrm{St}(m,r)$ can be defined by

$$R(X,\dot{X}) = \operatorname{cay}(W)X, \tag{6.9}$$

where $W \in \mathbb{R}^{m\times m}$ is a skew-symmetric matrix such that $WX = \dot{X}$. In the special case of the orthogonal group $\mathcal{O}(m) = \mathrm{St}(m,m)$, the matrix W can be determined by

$$W = \dot{X}X^T \implies WX = \dot{X}X^TX = \dot{X}. \tag{6.10}$$

This retraction can also be computed by a solving for Y the fixed-point system

$$Y = X + \frac{1}{2}W(X+Y) \tag{6.11}$$

to avoid performing costly matrix inversion[22].

6.2. **Low-rank Matrix Manifold.** The set of matrices $X \in \mathbb{R}^{m\times n}$ of rank r is a useful structure in many applications in both computing and theory. This set also admits a manifold structure as an embedded submanifold of $\mathbb{R}^{m\times n}$ of dimension $(m+n-r)r$. We present some results from the survey paper of Absil and Oseledets on retractions over this manifold, for our use in experiments in this work[32].

The set of matrices $X \in \mathbb{R}^{m\times n}$ of rank r is an embedded submanifold of $\mathbb{R}^{m\times n}$ of dimension $(m+n-r)r$ and is a Riemannian submanifold with the euclidean metric $g_X(A,B) = tr(A^TB)$ inherited from $\mathbb{R}^{m\times n}$. We denote this manifold as

$$\mathcal{M}_r = \left\{X \in \mathbb{R}^{m\times n} : \mathrm{rank}(X) = r\right\} \tag{6.12}$$

and name it the *low-rank manifold* or *fixed-rank manifold*. This manifold is non-compact as it is unbounded in $\mathbb{R}^{m\times n}$.

There are several ways to represent this manifold, based on the theory of power manifolds and quotient manifolds. A power manifold is a product

$$\mathcal{M}\times\mathcal{N} = \{(m,n) : m \in \mathcal{M}, n \in \mathcal{N}\}$$

of two manifolds $\mathcal{M},\mathcal{N}$, and the manifold structure comes in a simple way through charts of the form

$$\varphi = (\varphi_1,\varphi_2) : U_1\times U_2 \to \mathbb{R}^{d_1}\times\mathbb{R}^{d_2},\ (m,n) \mapsto \left(\varphi_1(m)^1,...,\varphi_1(m)^d,\varphi_2(n)^1,...,\varphi_2(n)^d\right).$$

Quotient manifolds are more complicated, but are based on quotient sets

$$X/\sim = \{[x] : x \in X\},\quad [x] = \{y \in X : x \sim y\},$$

for some equivalence relation $\sim$ over X, typically defined by group actions over X. These formulations are useful for instance when a cost function is invariant under some group action, such that solutions are not isolated. An example of this is the maximum-eigenvalue problem

$$\max_{x\in\mathbb{R}^d\setminus\{\mathbf{0}\}} \frac{x^TAx}{x^Tx},$$

where the cost function is invariant under scalar multiplication of non-zero real numbers. This motivates a formulation over

$$(\mathbb{R}^d \setminus \{\mathbf{0}\})/(\mathbb{R}\setminus\{\mathbf{0}\}) \simeq \mathrm{S}^{d-1},$$

where $(\mathbb{R}\setminus\{\mathbf{0}\})$ defines the equivalence relation $x \sim y \iff x = \lambda y, \lambda \in (\mathbb{R}\setminus\{\mathbf{0}\})$. The theory and use of quotient manifolds is described in detail in the book of Absil et al., with focus on matrix quotient manifolds[1].

One practical way of representing $\mathcal{M}_r$ is as the quotient manifold

$$\mathcal{M}_r \simeq (\mathrm{St}(m,r)\times\mathrm{GL}(r)\times\mathrm{St}(n,r))\,/\,(\mathcal{O}(r)\times\mathcal{O}(r)), \tag{6.13}$$

along with the Riemannian metric inherited from $\mathbb{R}^{m\times n}$[32]. This corresponds to the decomposition

$$X = USV^T,\ U \in \mathrm{St}(m,r), S \in \mathrm{GL}(r), V \in \mathrm{St}(n,r),$$

for which

$$X = (UQ_1)(Q_1^T S Q_2)(VQ_2)^T = USV^T,\ \forall Q_1, Q_2 \in \mathcal{O}(r).$$

This factorization is used in experiments with $\mathcal{M}_r$ in this work.

The tangent space $T_X\mathcal{M}_r$ can be given implicitly by the orthogonal projection onto the tangent space given by

$$\pi_{T_X\mathcal{M}} Z = ZVV^T + UU^T Z - UU^T ZVV^T, \tag{6.14}$$

for $X = USV^T$ factorized according to (6.13).We can also factorize the $T_X\mathcal{M}_r$ as

$$T_X\mathcal{M}_r = \Big\{ U\dot{S}V^T + U_pV^T + UV_p^T : \dot{S} \in \mathbb{R}^{r\times r}, U_p \in \mathbb{R}^{m\times r}, U^TU_p = 0, V_p \in \mathbb{R}^{n\times r}, V^TV_p = 0 \Big\}. \tag{6.15}$$

Many retractions over this manifold are given in [32]. For instance the projective retraction

$$R(X,\dot{X}) = \underset{Y\in\mathcal{M}_r}{\arg\min} \left\| Y - (X+\dot{X}) \right\|_F, \tag{6.16}$$

solved into a computationally feasible expression using linear algebra, or a quotient-based retraction

$$R(X,\dot{X}) = R_{\mathrm{St}}(U,\dot{U})(S+\dot{S})R_{\mathrm{St}}(V,\dot{V})^T,$$

with an appropriate retraction R_{St} that respects the $\mathcal{O}(r)$-invariance. In this work we use the projective retraction, other retractions are left for future work.

The method of computing the projective retraction (6.16) of [32] is based on having the tangent vector $\dot{X}$ by the factors $(\dot{S}, U_p, V_p)$ according to (6.15). First we use a QR-factorization to find

$$U_p = Q_uS_u, \quad V_p = Q_vS_v. \tag{6.17}$$

Thereafter, based on

$$X + \dot{X} = \begin{pmatrix} U & Q_u \end{pmatrix} \begin{pmatrix} S+\dot{S} & S_u \\ S_v^T & 0 \end{pmatrix} \begin{pmatrix} V^T \\ Q_v^T \end{pmatrix}, \tag{6.18}$$

find the unique SVD (U_s, Σ_s, V_s) of the center matrix $\begin{pmatrix} S+\dot{S} & S_u \\ S_v^T & 0 \end{pmatrix}$ with decreasing singular values. Then the projective retraction is given by

$$R(X,\dot{X}) = U_+S_+V_+^T, \tag{6.19}$$

where U_+, S_+, V_+ are given by

$$U_+ = \begin{pmatrix} U & Q_u \end{pmatrix} U_s(:, 1:r), \tag{6.20}$$

$$V_+ = \begin{pmatrix} V & Q_v \end{pmatrix} V_s(:, 1:r), \tag{6.21}$$

$$S_+ = \Sigma_s(1:r, 1:r). \tag{6.22}$$

The notation of $A(a:b, c:d)$ means the $(b-a)\times(d-c)$-matrix obtained from the entries of A with $a \le i \le b$ and $c \le j \le d$. This retraction is also a second-order retraction, meaning that

$$\operatorname{dist}\left(\exp_X(\dot{X}), R(X,\dot{X})\right) = \mathcal{O}\left(\left\|\dot{X}\right\|_X^3\right).$$

7. Numerical Tests

We run some experiments to test the Riemannian gradient descent algorithm on some manifolds. We run one experiment on a memory-problem with the orthogonal group $\mathcal{O}(m) = \mathrm{St}(m,m)$ and one experiment on image-classification over the CIFAR-10 dataset with the manifold of low-rank matrices $\mathcal{M}_r$. Due to time limitations, we were unable to properly analyze the performance of the algorithms we present, these experiments are therefore only proofs of concept, though fit for illustration of the concepts described in this work and as a sandbox to experiment in. The focus in this work has been more about exploring the framework of Riemannian optimization and its potential application to deep learning, than developing competitive methods and algorithms to efficiently solve problems. More close analysis of the methods can be left for future work.

The experiments are run in Python code using the PyTorch deep learning framework with manifold operations written by hand from the results gathered. Though many deep learning networks use single-precision floats for increased efficiency in training these days, all experiments in this work were performed with double-precision floats. This is because with our approach of having manifold quantities embedded in euclidean space, single-precision floats forced quantities to be too far from the manifold they were supposed to stay on, contributing to drift away from the manifold.

The experiments are written in `Jupyter` notebooks collected in a `zip`-archive attachment to the thesis.

7.1. Copying Memory Problem.

We solve a memory-copying problem with a recurrent neural network with orthogonally constrained hidden-to-hidden matrix. This is the same test problem as in [20] and [21]. We use the Riemannian gradient descent method, whereas Massart and Abrol use a Riemannian coordinate descent algorithm and Arjovsky et al. use euclidean methods on a factorization of the manifold.

A *recurrent neural network* (RNN) is a deep learning architecture for efficiently learning from data with a temporal relation. RNNs are often used for problems in speech and language processing[2]. We use the model as described in [21]. Inputs of the model are sorted vectors

$$(x(t), t = 0, ..., N) \subset \mathbb{R}^{n_{\text{in}}},$$

and produces an equally long list of output vectors

$$(y(t), t = 0, ..., N) \subset \mathbb{R}^{n_{\text{out}}}.$$

At each time t the model has an internal state $h(t) \in \mathbb{R}^{n_h}$ which is affected by all previous inputs $x(\tau), 0 \le \tau \le t$, but no future ones. The model is defined by

$$h(t+1) = \sigma\left(W_{\text{in}}x(t) + W_h h(t)\right), \tag{7.1a}$$

$$y(t) = W_{\text{out}}h(t+1) + b_{\text{out}}, \tag{7.1b}$$

where σ is a non-linear activation function, $W_{\text{in}} \in \mathbb{R}^{n_h \times n_{\text{in}}}$, $W_h \in \mathbb{R}^{n_h \times n_h}$, $W_{\text{out}} \in \mathbb{R}^{n_h \times n_{\text{out}}}$, and $b \in \mathbb{R}^{n_{\text{out}}}$. We let the hidden vector be initialized by

$$h(0) = 0, \tag{7.2}$$

to ensure in a simple way that the network is unbiased before information from the specific x has entered the system. The matrix W_h is called the *hidden-to-hidden* matrix and the mapping $y \mapsto W_{\text{out}} h + b_{\text{out}}$ is called the *output layer*. An RNN working on inputs of length N is said to be an RNN of *depth* N.

When the depth of an RNN grows, the repeated multiplication with the hidden matrix can cause the hidden vector h to either grow very large in magnitude or shrink to zero in magnitude, if the magnitude of the eigenvalues of the hidden matrix are far from 1. If the hidden vector grows the system becomes unstable, and if the hidden vector decays the information from the first inputs are lost when the later inputs are used. Thus, by constraining the eigenvalues of W_h to be ± 1 by making W_h be an element of the orthogonal group $\mathcal{O}(n_h)$, the model might better be able to learn patterns of long time-dependence. This approach has also been suggested for dealing with the classical problem of vanishing / exploding gradients in RNN training, for much the same reasons[20].

We attempt to solve the copying memory problem. An alphabet of eight symbols $(0, ..., 7)$, is provided along with two special symbols. The first ten entries of an input chain $(x(0), ..., x(N))$ are symbols from $(0, ..., 7)$, which the model should be trained to be able to remember. After this follows $T-1$ instances of the first special symbol 8, signifying wait and keep remembering, after which comes the second special symbol 9, telling the model to repeat the symbols the model were to remember. The target output of the RNN is a chain of $T + 10$ entries of 8 followed by the first ten input entries in order. An example input and target output might look like

$$x = (2, 6, 7, 2, 2, 6, 4, 3, 5, 4, 8, 8, ..., 8, 8, 9, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8) \tag{7.3a}$$

$$y = (8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, ..., 8, 8, 8, 2, 6, 7, 2, 2, 6, 4, 3, 5, 4) \tag{7.3b}$$

Thus, each input and target output is a chain of $T + 20$ entries of the alphabet $(0, ..., 9)$.

We create a dataset of such chains (7.3) by drawing the first ten symbols uniformly from $(0, ..., 7)$ for D independent chains and filling in the rest of the uniquely determined entries, creating a dataset of size $|\mathcal{D}| = D$. Each entry is preprocessed before running through the network by one-hot encoding them, such that $x(t) \in \mathbb{R}^{10}$ is the vector with all zeroes except for a one in the n-th component for symbol n. The outputs $y(t) \in \mathbb{R}^{10}$ of the network are taken to signify some learned confidence for each component that the correct symbol should be the corresponding symbol. We make predictions with the network by predicting the symbol whose corresponding component has the largest entry.

We constrain the hidden-to-hidden matrix W_h to be orthogonal to address stability and vanishing / exploding gradients concerns. The activation function used is the rectified linear-unit

$$\sigma(x) = \text{ReLU}(x) = \max\{0, x\}.$$

The model is trained by performing RGD on $W_h \in \mathcal{O}(n_h)$ and gradient descent on the other parameters. The Cayley retraction (6.9)-(6.10) is used. Due to finite-precision arithmetic, the solution of the retractions drifts from the manifold over time and this is addressed by regularly performing an orthogonal projection back to the manifold.

We use the cross entropy loss function

$$\log \frac{\exp(y(t)_{z(t)})}{\sum_{i=1}^{n_{\text{out}}} \exp(y(t)_i)},$$

where $y(t)$ is the t-th entry produced by the chain x and $z(t)$ is its target output, and summing over all $x(t), t = 0, ..., T + 19$. This loss function is common in problems with categorical targets.

Figure 1 shows the convergence history of one running of the experiment along with the learning rates used for both the Riemannian and euclidean gradient descent. We use a dataset of $D = 10^3$ chains of length $T + 20 = 40 + 20 = 60$, with 30% of this being reserved as a test set. We choose a hidden size of $n_h = 100$ and choose the activation function $\sigma = \text{ReLU}$, the rectified linear-unit.

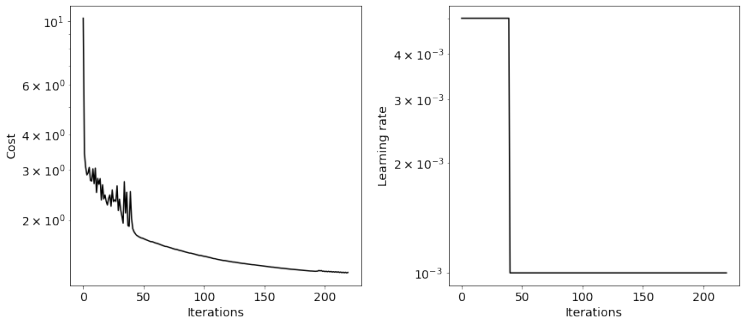


FIGURE 1. Convergence history of a running of the copying memory problem with orthogonal hidden-to-hidden matrix. On the left is plotted the cost function over the test set as a function of training epochs. On the right is plotted the learning rates that were used for each epoch in training.

According to figure 1, we can conclude that the model is able to learn for problems of this large time-dependence. The network learns to output the symbol 8 for the first $T + 10$ entries, as it should, and thereafter outputs some other symbols. The accuracy of these predicted symbols are not great however, and it seems like they are mostly representative of how often the symbols show up in the initial ten entries of x. As an example, for one input of the test set, the model gave the prediction

$$(8, 8, 8, ..., 8, 8, 8, 8, 1, 1, 1, 1, 1, 1, 1, 1)$$

for a sample with target vector

$$(8, 8, 8, ..., 8, 8, 0, 0, 4, 1, 1, 1, 1, 4, 0, 7),$$

the skipped symbols are all 8. Still, this shows that the model is able to learn something about the structure of the problem and can make some progress on the hard part of the problem.

The method shows some instability to the learning rate used. If the learning rate was too high, the retractions were prone to produce `NaN`-values. This could be due to instability in the implementation of the matrix exponential or due to the inverse of the matrix $I - \frac{1}{2}W$ not existing for all W with large magnitudes.

In early development of the experiment, the exponential retraction (6.8) with the Scipy-implementation of the matrix exponential was used, but this took much longer

to compute, showed more drift from the manifold, and gave no more effective cost function decrease per iteration than the Cayley retraction. That the iterates drifted more from the manifold might be due to the implementation used being meant for general computations, also in completely non-manifold based applications. Such a general implementation would not be optimized to prevent manifold drift specifically. A more taylored implementation could therefore be more effective in this case. Other interesting work would be to use the fixed-point based computation (6.11), but we suspect this would be too inaccurate and again cause too much drift from the manifold.

7.2. **CIFAR-10 Image Classification.** We solve an image classification problem over the CIFAR-10 dataset using a residual convolutional network with low-rank constrained fully connected final layers.

The CIFAR-10 dataset is a collection of 60000 32×32 color images, each belonging to one of ten classes[33]. This dataset is often used for testing computer vision architectures and is a considerably harder problem than the canonical MNIST digit classification-problem. In tests on the MNIST problem we have achieved results of 90% accuracy and above, but such results are available with almost any network with the scale of computations available today.

We use a *convolutional neural network* (CNN) with a convolutional block followed by a classification network of fully connected layers. The first part of the network consists of two convolutional layers, each followed by a ReLU-activation and then a max pool layer. Each of the convolutional layers uses a filter of size 5×5 and the max pool is of size 2×2. After the convolutional block, the produced filtered image is flattened and run through the classification network. This network consists of five fully connected layers, the middle three include residual connections and the first and last layers change the dimension of the propagated vector to make the hidden layers have width w. Residual connections are thought to improve learning dynamics of neural networks and were first proposed for use in convolutional networks for image classification, but now see wider use[4]. This classification network can be written as

$$\begin{aligned}
\mathcal{N}_\theta^{\text{class}}(x) &= y^5 \\
y^1 &= \sigma\left(W^0 x + b^0\right),\ W^0 \in \mathbb{R}^{\dim(x)\times w},\ b^0 \in \mathbb{R}^w \\
y^{k+1} &= y^k + \sigma\left(W^k y^k + b^k\right),\ W^k \in \mathbb{R}^{w\times w},\ b^k \in \mathbb{R}^w \ \text{ for } k = 1,2,3 \\
y^5 &= \sigma\left(W^4 y^4 + b^4\right),\ W^4 \in \mathbb{R}^{w\times n_{\text{out}}},\ b^0 \in \mathbb{R}^{n_{\text{out}}}.
\end{aligned}$$

There are ten classes of images in the dataset, so $n_{\text{out}} = 10$. We train the network to have the entry of the output vector corresponding to the known target as large as possible compared to the other entries. We use the cross entropy loss function again as it is suitable for general classification tasks.

We constrain all the weight matrices W^l to the fixed-rank matrix manifold $\mathcal{M}_r$ for a choice of r, except for the final layer which has full rank equal to 10. We train the network with Riemannian gradient descent using the projection retraction (6.17)-(6.22) for the fixed-rank constrained matrices, and gradient descent for all other parameters.

In one run of the experiment, we obtain the convergence shown in figure 2 and finish with a test-accuracy of about 46%. In this run we chose to use constrained matrices of rank $r = 40$ and network width $w = 120$. These are not great results, well-trained networks can achieve accuracies of well over 90%, see for instance [4]. Still, it shows

that the model is able to optimize for classifying images by some learned features using the manifold-constraints and Riemannian gradient descent.

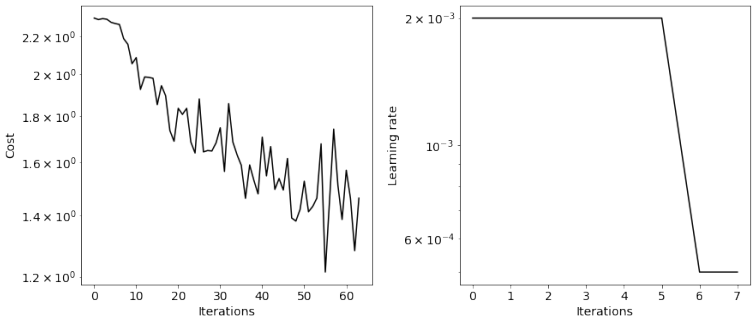


FIGURE 2. Convergence history of a running of the CIFAR-10 image-classification problem with fixed-rank constrained fully connected layers. On the left is plotted the cost function over the test set as a function of training epochs. On the right is plotted the learning rates that were used for each epoch in training.

Figure 3 shows an example sequence of four CIFAR-10 images that the model predicted to be of the classes `frog`, `frog`, `plane`, and `horse`, from left to right. The correct classes are `dog`, `cat`, `plane`, and `horse`, getting half of the predictions right, as is about what is expected with a model trained to an accuracy of 46%.

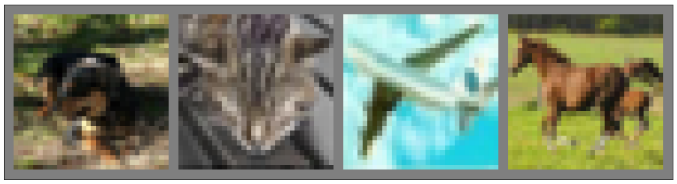


FIGURE 3. An example sequence of four CIFAR-10 images that a model trained to 46% accuracy predicted to be of the classes `frog`, `frog`, `plane`, and `horse`, from left to right. The correct classes are `dog`, `cat`, `plane`, and `horse`.

8. Conclusion

The topic of Riemannian optimization has been discussed with presentations of central definitions and results, with a focus kept on conceptual clearness and application to the specific algorithms used, as well as to concerns of practical use and implementation. The Riemannian gradient descent algorithm has been presented and

we have compared the properties of this Riemannian algorithm with the well known euclidean gradient descent method it generalizes.

In view of recent interest in Riemannian methods for deep learning, the application of Riemannian optimization and Riemannian gradient descent to deep learning problems and the benefits it might provide has been discussed, as well as some potential problems that might arise.

Simple experiments have been performed with the orthogonal group manifold applied to a recurrent neural network-problem and with the fixed-rank matrix manifold applied to an image classification problem using the CIFAR-10 dataset. The experiments show that neural network models with manifold-constrained parameters can solve non-trivial deep learning problems and that Riemannian gradient descent can be used to train these parameters.

References

[1] P.-A. Absil, R. Mahony, and R. Sepulchre. *Optimization Algorithms on Matrix Manifolds*. Princeton, NJ: Princeton University Press, 2008, pp. xvi+224. ISBN: 978-0-691-13298-3.

[2] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". en. In: *Nature* 521.7553 (May 2015), pp. 436–444. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature14539. URL: http://www.nature.com/articles/nature14539 (visited on 12/01/2021).

[3] Michael M. Bronstein et al. *Geometric Deep Learning: Grids, Groups, Graphs, Geodesics, and Gauges*. 2021. DOI: 10.48550/ARXIV.2104.13478. URL: https://arxiv.org/abs/2104.13478.

[4] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[5] Catherine F. Higham and Desmond J. Higham. "Deep Learning: An Introduction for Applied Mathematicians". en. In: *SIAM Review* 61.3 (Jan. 2019), pp. 860–891. ISSN: 0036-1445, 1095-7200. DOI: 10.1137/18M1165748. URL: https://epubs.siam.org/doi/10.1137/18M1165748 (visited on 01/03/2022).

[6] Jorge Nocedal and Stephen J. Wright. *Numerical optimization*. 2nd ed. Springer series in operations research. OCLC: ocm68629100. New York: Springer, 2006. ISBN: 9780387303031.

[7] Nicolas Boumal. *An introduction to optimization on smooth manifolds*. To appear with Cambridge University Press. Apr. 2022. URL: http://www.nicolasboumal.net/book.

[8] David G. Luenberger. "The Gradient Projection Method along Geodesics". In: *Management Science* 18.11 (1972), pp. 620–631. ISSN: 00251909, 15265501. URL: http://www.jstor.org/stable/2629156 (visited on 07/09/2022).

[9] C. Udriste. *Convex Functions and Optimization Methods on Riemannian Manifolds*. Mathematics and Its Applications. Springer Dordrecht, 1994. ISBN: 978-94-015-8390-9. DOI: 10.1007/978-94-015-8390-9.

[10] Alan Edelman, Tomás A. Arias, and S.T. Smith. "The Geometry of Algorithms with Orthogonality Constraints". In: *SIAM J. Matrix Anal. Appl.* 20 (1998), pp. 303–353.

[11] Nicolas Boumal et al. "Manopt, a Matlab Toolbox for Optimization on Manifolds". In: *Journal of Machine Learning Research* 15.42 (2014), pp. 1455–1459. URL: http://jmlr.org/papers/v15/boumal14a.html.

[12] James Townsend, Niklas Koep, and Sebastian Weichwald. "Pymanopt: A Python Toolbox for Optimization on Manifolds using Automatic Differentiation". In: *Journal of Machine Learning Research* 17.137 (2016), pp. 1–5. URL: http://jmlr.org/papers/v17/16-177.html.

[13] Ronny Bergmann. "Manopt.jl: Optimization on Manifolds in Julia". In: *Journal of Open Source Software* 7.70 (2022), p. 3866. DOI: 10.21105/joss.03866.

[14] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[15] John M. Lee. *Introduction to smooth manifolds*. 2nd ed. Graduate texts in mathematics 218. OCLC: ocn800646950. New York: Springer, 2013. ISBN: 978-1-4419-9982-5.

[16] J.M. Lee. *Introduction to Riemannian Manifolds*. Graduate Texts in Mathematics. Springer International Publishing, 2019. ISBN: 9783319917542. DOI: 10.1007/978-3-319-91755-9.

[17] Christopher Stover and Eric W. Weisstein. *Einstein Summation*. URL: https://mathworld.wolfram.com/EinsteinSummation.html.

[18] E. Hairer, S. P. Nørsett, and Gerhard Wanner. *Solving ordinary differential equations I: nonstiff problems*. 2nd rev. ed. Springer series in computational mathematics 8. OCLC: ocn620251790. Heidelberg ; London: Springer, 2009. ISBN: 9783540566700.

[19] Christopher Heil. *Metrics, Norms, Inner Products, and Operator Theory*. Applied and numerical harmonic analysis. Springer International Publishing, 2018. ISBN: 9783319653235. DOI: 10.1007/978-3-319-65322-8.

[20] Martin Arjovsky, Amar Shah, and Yoshua Bengio. "Unitary Evolution Recurrent Neural Networks". In: *Proceedings of The 33rd International Conference on Machine Learning*. Ed. by Maria Florina Balcan and Kilian Q. Weinberger. Vol. 48. Proceedings of Machine Learning Research. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 1120–1128. URL: https://proceedings.mlr.press/v48/arjovsky16.html.

[21] Estelle Massart and Vinayak Abrol. "Coordinate Descent on the Orthogonal Group for Recurrent Neural Network Training". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.7 (June 2022), pp. 7744–7751. DOI: 10.1609/aaai.v36i7.20742. URL: https://ojs.aaai.org/index.php/AAAI/article/view/20742.

[22] Jun Li, Fuxin Li, and Sinisa Todorovic. "Efficient Riemannian Optimization on the Stiefel Manifold via the Cayley Transform". In: *International Conference on Learning Representations*. 2020.

[23] Jiayun Wang et al. "Orthogonal Convolutional Neural Networks". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2020.

[24] Nicolas Boumal, P-A Absil, and Coralia Cartis. "Global rates of convergence for nonconvex optimization on manifolds". In: *IMA Journal of Numerical Analysis* 39.1 (Feb. 2018), pp. 1–33. DOI: 10.1093/imanum/drx080. URL: https://doi.org/10.1093%2Fimanum%2Fdrx080.

[25] P.-A. Absil and Jérôme Malick. "Projection-like Retractions on Matrix Manifolds". In: *SIAM Journal on Optimization* 22.1 (2012), pp. 135–158. DOI: 10.1137/100802529. URL: http://link.aip.org/link/?SJE/22/135/1.

[26] C. Cartis, N. I. M. Gould, and Ph. L. Toint. "On the Complexity of Steepest Descent, Newton's and Regularized Newton's Methods for Nonconvex Unconstrained Optimization Problems". In: *SIAM Journal on Optimization* 20.6 (2010), pp. 2833–2852. DOI: 10.1137/090774100. eprint: https://doi.org/10.1137/090774100. URL: https://doi.org/10.1137/090774100.

[27] Yurii Nesterov. *Lectures on Convex Optimization*. 2nd. Springer Publishing Company, Incorporated, 2018. ISBN: 978-3-319-91578-4.

[28] Hadi Daneshmand et al. "Escaping Saddles with Stochastic Gradients". In: *Proceedings of the 35th International Conference on Machine Learning*. Vol. 80.

Proceedings of Machine Learning Research. PMLR, Oct. 2018, pp. 1155–1164. URL: https://proceedings.mlr.press/v80/daneshmand18a.html.

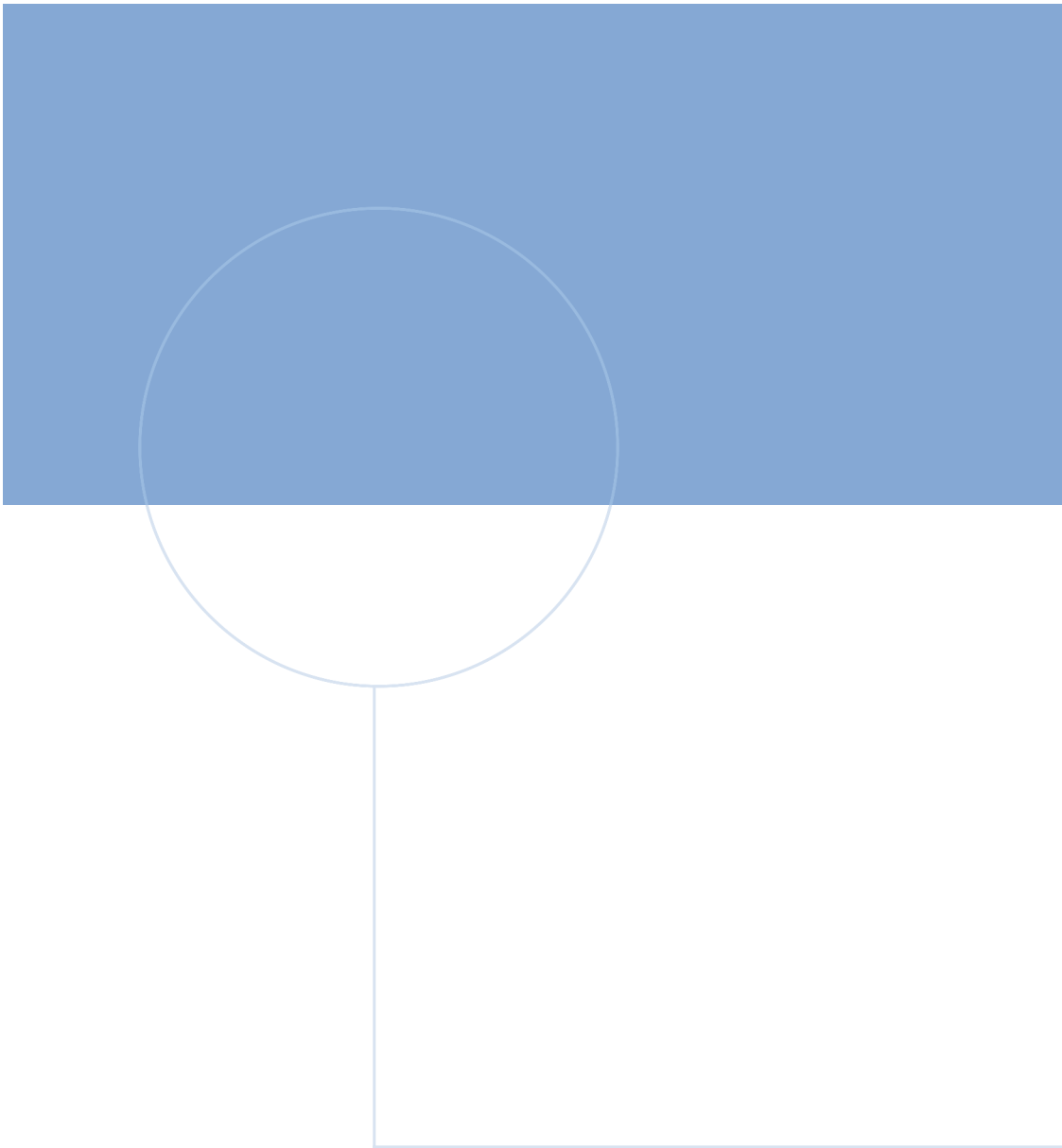
[29] Silvère Bonnabel. "Stochastic Gradient Descent on Riemannian Manifolds". In: *IEEE Transactions on Automatic Control* 58.9 (2013), pp. 2217–2229. DOI: 10.1109/TAC.2013.2254619.

[30] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". In: *Nature Methods* 17 (2020), pp. 261–272. DOI: 10.1038/s41592-019-0686-2.

[31] Awad H. Al-Mohy and Nicholas J. Higham. "A New Scaling and Squaring Algorithm for the Matrix Exponential". In: *SIAM Journal on Matrix Analysis and Applications* 31.3 (2010), pp. 970–989. DOI: 10.1137/09074721X. eprint: https://doi.org/10.1137/09074721X. URL: https://doi.org/10.1137/09074721X.

[32] P.-A. Absil and I. V. Oseledets. "Low-rank retractions: a survey and new results". In: *Computational Optimization and Applications* (2014). accepted for publication. URL: http://sites.uclouvain.be/absil/2013.04.

[33] Alex Krizhevsky. *Learning multiple layers of features from tiny images*. Tech. rep. 2009.



NTNU

Norwegian University of
Science and Technology