Stochastic Flows and Geometric Optimization on the Orthogonal Group

Krzysztof Choromanski 1 2 David Cheikhi 3* 3 Jared Davis 4* 1 Valerii Likhosherstov 4* 4 Achille Nazaret 3* 3 Achraf Bahamou 2* 2 Xingyou Song 1* 1 Mrugank Akarte 2 Jack Parker-Holder 5 Jacob Bergquist 2 Yuan Gao 2 Aldo Pacchiano 6 Tamas Sarlos 7 Adrian Weller 4* 8 Vikas Sindhwani 1

Abstract

We present a new class of stochastic, geometrically-driven optimization algorithms on the orthogonal group $O(d)$ and naturally reductive homogeneous manifolds obtained from the action of the rotation group $SO(d)$. We theoretically and experimentally demonstrate that our methods can be applied in various fields of machine learning including deep, convolutional and recurrent neural networks, reinforcement learning, normalizing flows and metric learning. We show an intriguing connection between efficient stochastic optimization on the orthogonal group and graph theory (e.g. matching problem, partition functions over graphs, graph-coloring). We leverage the theory of Lie groups and provide theoretical results for the designed class of algorithms. We demonstrate broad applicability of our methods by showing strong performance on the seemingly unrelated tasks of learning world models to obtain stable policies for the most difficult Humanoid agent from OpenAI Gym and improving convolutional neural networks.

1. Introduction and Related Work

Constrained optimization is at the heart of modern machine learning (ML) (Kavis et al., 2019; Vieillard et al., 2019) as many fields of ML benefit from methods such as linear, quadratic, convex or general nonlinear constrained optimization. Other applications include reinforcement learning with safety constraints (Chow et al., 2018). Our focus in this paper is on-manifold optimization, where constraints require solutions to belong to certain matrix manifolds. We are mainly interested in three spaces: the orthogonal group $O(d) = \{M \in \mathbb{R}^{d \times d} : M^\top M = I_d\}$, its generalization, the Stiefel manifold $ST(d, k) = \{M \in \mathbb{R}^{d \times k} : M^\top M = I_k\}$ for $k \leq d$ and its subgroup $SO(d) = \{M \in \mathbb{R}^{d \times d} : M^\top M = I_d, \det(M) = 1\}$ of $d$-dimensional rotations.

Interestingly, a wide variety of constrained optimization problems can be rewritten as optimization on matrix manifolds defined by orthogonality constraints. These include:

- **metric learning**: optimization leveraging the decomposition of $k$-rank Mahalanobis matrices $M = UD_+U^\top$, where: $U \in ST(d, k)$ and $D_+$ is diagonal with positive nonzero diagonal entries (Shukla & Anand, 2015);

- **synchronization over the special Euclidean group**, where its elements $g_i \in SE(d) = \mathbb{R}^d \times SO(d)$ need to be retrieved from noisy pairwise measurements encoding the elements $g_i g_j^{-1}$ that transform $g_i$ to $g_j$; these algorithms find applications particularly in vision and robotics (SLAM) (Rosen et al., 2019);

- **PSD programs with block-diagonal constraints**: optimization on $ST(d, k)$ can be elegantly applied for finding bounded-rank solutions to positive semidefinite (PSD) programs, where the PSD matrix $Z$ can be decomposed as: $Z = R^\top R$ for a low-rank matrix $R \in \mathbb{R}^{r \times d}$ ($r \ll d$) and where $r$ can be systematically increased leading effectively to the so-called Riemannian staircase method;

- **combinatorial optimization**: combinatorial problems involving object-rearrangements such as sorting or graph isomorphism (Zavlanos & Pappas, 2008) can be cast as optimization on the orthogonal group $O(d)$ which is a relaxation of the permutation group $Perm(d)$.

Orthogonal groups are deeply rooted in the theory of manifolds. As noted in (Gallier, 2011) “…most familiar spaces are naturally reductive manifolds. Remarkably, they all arise from some suitable action of the rotation group $SO(d)$, …
A Lie group, which emerges as the master player.” These include in particular the Stiefel and Grassmann manifolds.

It is striking that the group \( O(d) \) and its “relatives” provide solutions to many challenging problems in machine learning, where unstructured (i.e. unconstrained) algorithms do exist. Important examples include the training of recurrent and convolutional neural networks, and normalizing flows. Sylvester normalizing flows are used to generate complex probabilistic distributions and encode invertible transformations between density functions using matrices taken from \( ST(d,k) \) (van den Berg et al., 2018). Orthogonal convolutional layers were recently shown to improve training models for vision data (Wang et al., 2019; Bansal et al., 2018). Finally, the learning of deep/recurrent neural network models is notoriously difficult due to the vanishing/exploding gradient problem (Bengio et al., 1993). To address this challenge, several architectures such as LSTMs and GRUs were proposed (Greff et al., 2015; Cho et al., 2014) but don’t provide guarantees that gradients’ norms would stabilize. More recently orthogonal RNNs (ORNNS) that do provide such guarantees were introduced. ORNNs impose orthogonality constraints on hidden state transition matrices (Arjovsky et al., 2016; Henaff et al., 2016; Helfrich et al., 2018). Training RNNs can now be seen as optimization on matrix manifolds with orthogonality constraints.

However, training orthogonal RNNs is not easy and reflects challenges common for general on-manifold optimization (Absil et al., 2008; Hairer, 2005; Edelman et al., 1998). Projection methods that work by mapping the unstructured gradient step back into the manifold have expensive, cubic time complexity. Standard geometric Riemannian-gradient methods relying on steps conducted in the space tangent to the manifold at the point of interest and translating updates back to the manifold via exponential or Cayley mapping (Helfrich et al., 2018) still require cubic time for those transformations. Hence, in practice orthogonal optimization becomes problematic in higher-dimensional settings.

Another problem with these transforms is that apart from computational challenges, they also incur numerical instabilities ultimately leading to solutions diverging from the manifold. We demonstrate this in Sec. 6 and provide additional evidence in the Appendix (Sec. 9.1.2, 9.3), in particular on a simple 16-dimensional combinatorial optimization task.

An alternative approach is to parameterize orthogonal matrices by unconstrained parameters and conduct standard gradient step in the parameter space. Examples include decompositions into short-length products of Householder reflections and more (Mhammedi et al., 2017; Jing et al., 2017). Such methods enable fast updates, but inherently lack representational capacity by restricting the class of representable matrices and thus are not a subject of our work.

In this paper we propose a new class of optimization algorithms on matrix manifolds defined by orthogonality constraints, in particular: the orthogonal group \( O(d) \), its subgroup of rotations \( SO(d) \) and Stiefel manifold \( ST(d,k) \).

We highlight the following contributions:

1. **Fast Optimization:** We present the first on-manifold stochastic gradient flow optimization algorithms for ML with sub-cubic time complexity per step as opposed to cubic characterizing SOTA (Sec. 3.3) that do not constrain optimization to strictly lower-dimensional subspaces. That enables us to improve training speed without compromising the representational capacity of the original Riemannian methods. We obtain additional computational gains (Sec. 3.1) by parallelizing our algorithms.

2. **Graphs vs. Orthogonal Optimization:** To obtain these, we explore intriguing connections between stochastic optimization on the orthogonal group and graph theory (in particular the matching problem, its generalizations, partition functions over graphs and graph coloring). By leveraging structure of the Lie algebra of the orthogonal group, we map points from the rotation group to weighted graphs (Sec. 3).

3. **Convergence of Stochastic Optimizers:** We provide rigorous theoretical guarantees showing that our methods converge for a wide class of functions \( F : M \rightarrow \mathbb{R} \) defined on \( O(d) \) under moderate regularity assumptions (Sec. 5).

4. **Wide Range of Applications:** We confirm our theoretical results by conducting a broad set of experiments ranging from training RL policies and RNN-based world models (Ha & Schmidhuber, 2018) (Sec. 6.1) to improving vision CNN-models (Sec. 6.2). In the former, our method is the only one that trains stable and effective policies for the high-dimensional Humanoid agent from OpenAI Gym. We carefully quantified the impact of our algorithms. The RL experiments involved running 240 training jobs, each distributed over hundreds of machines. We also demonstrated numerical instabilities of standard non-stochastic techniques (Sec. 9.1.2 and Sec. 9.3).

Our algorithm can be applied in particular in the black-box setting (Salimans et al., 2017), where function to be optimized is accessible only through potentially expensive querying process. We demonstrate it in Sec. 6.1, where we simultaneously train the RNN-model and RL-policy taking advantage of it through its latent states. Full proofs of our theoretical results are in the Appendix.

2. **The Geometry of the Orthogonal Group**

In this section we provide the reader with technical content used throughout the paper. The theory of matrix manifolds is vast so we focus on concepts needed for the exposition of our results. We refer to Lee (2012) for a more thorough introduction to the theory of smooth manifolds.
We conclude that tangent spaces for the orthogonal group are usually equipped with additional natural group structure. For a Lie group \(M \subseteq \mathbb{R}^{N \times N}\) consisting of matrices \(\{H_{i,j} : 1 \leq i, j \leq d\}\) with basis \(\text{H}_{d}\) consisting of matrices \(\{H_{i,j} : 1 \leq i, j \leq d\}\) s.t. \(H_{i,j}[i, j] = -H_{i,j}[j, i] = 1\) and \(H_{i,j}[k, l] = 0\) if \(\{k, l\} \neq \{i, j\}\).

Smooth manifolds whose tangent spaces are equipped with inner products (see: Sec. 9.4.2 for more details) are called Riemannian manifolds (Lee, 2012). Inner products provide a means to define a metric and talk about non-Euclidean distances between points on the manifold (or equivalently: lengths of geodesic lines connecting points on the manifold).

### 2.1.1. Exponential Mapping & Cayley Transform

Points \(U\Omega\) of a tangent space \(T_U(O(d))\) can be mapped back to \(O(d)\) via matrix exponentials by the mapping \(U\text{exp}(t\Omega)\) for \(t \in \mathbb{R}\). Curves \(\gamma_\Omega : \mathbb{R} \rightarrow O(d)\) defined as \(\gamma_\Omega(t) = \text{Uexp}(t\Omega)\) are in fact geodesics on \(O(d)\) tangent to \(U\Omega\) in \(U\). The analogue of the set of canonical axes are the geodesics induced by the canonical basis \(\mathcal{H}_d\). The following lemma describes these geodesics.

#### Lemma 2.4 (Givens rotations and geodesics)

*The geodesics on \(O(d)\) induced by matrices \(H_{i,j}\) are of the form \(UG_{i,j}\), where \(G_{i,j}\) is a t-angle Givens rotation in the 2-dimensional space \(\text{Span}\{e_i, e_j\}\) defined as: \(G_{i,j}[i, i] = G_{i,j}[j, j] = \cos(t), G_{i,j}[i, j] = -G_{i,j}[j, i] = \sin(t)\) and \(G_{i,j}[k, l] = I_d[k, l]\) for \(\{k, l\} \neq \{i, j\}\).*

All introduced geometric concepts are illustrated in Fig. 1.

**Figure 1.** Illustration of basic geometric manifold concepts: tangent vector space \(T_U\) at point \(U \in M\). If \(M = O(d)\) then \(T_U = \{U\Omega : \Omega \in \text{Sk}(d)\}\) and geodesics are of the form \(U\text{exp}(t\Omega)\) for \(t \in \mathbb{R}\). Geodesics tangent to points \(UH_{i,j}\) are of the form \(UG_{i,j}\).

For the general Stiefel manifold \(ST(d, k)\), exponentials can also be used to map skew-symmetric matrices to curves on \(ST(d, k)\) as follows: \(\gamma_\Omega(t) = \text{exp}(t\Omega)X\), where \(X \in ST(d, k)\) and \(\text{exp}\) is mapping \(\text{Sk}(d)\) to \(O(d)\). Analogously, curves \(\gamma_\Omega\) are tangent to \(\Omega X\) in \(X\). The exponential map is sometimes replaced by the Cayley transform defined as: \(Y(\Omega) = (I + \Omega/2)^{-1}(I - \Omega/2)\). Again, it can be shown that curves \(\gamma_\Omega\) defined as: \(\gamma_\Omega(t) = Y(t\Omega)X\) are tangent to \(\Omega X\) in \(X\) (since: \(\gamma'(0) = -\Omega X\)), even though they are no longer geodesics.

### 2.1.2. Riemannian Optimization on \(ST(d, k)\)

Consider an optimization problem of the form:

\[
\max_{X \in \mathbb{R}^{d \times k}, X^\top X = I_k} F(X)
\]

for a differentiable function: \(F : \mathbb{R}^{d \times k} \rightarrow \mathbb{R}\). 

---

**Stochastic Flows and Geometric Optimization on the Orthogonal Group**
Notice that the standard directional derivative functional $DF_X : \mathbb{R}^{d \times k} \rightarrow \mathbb{R}$ satisfies: $DF_X(Z) = \sum_{i,j} \partial_{X_{i,j}} F(Z_{i,j}) = \text{tr}(G^\top Z)$, where $G$ is a standard gradient matrix. To extend gradient-based techniques to on-manifold optimization, we need to:

1. compute the projection of $G$ into the tangent space $T_X(\mathcal{S}T(d,k))$, which we call Riemannian gradient,
2. use curves on $\mathcal{S}T(d,k)$ tangent to the Riemannian gradient to make updates on the manifold.

It turns out that to make updates on the manifold, it will suffice to use the curves $\gamma$ defined above. It remains to compute Riemannian gradients and corresponding skew-symmetric matrices $\Omega$. Using standard representation theorems (see: 9.4.3), it can be proven (under canonical inner product, see: Sec. 9.4.2) that the Riemannian gradient is of the form $\mathcal{R} = \Omega X$, where $\Omega \in \mathbb{S}(d)$ satisfies:

$$\Omega = \Omega(X, G) = GX^\top - XG^\top.$$  \hfill (2)

We are ready to conclude that for $X(t) \in \mathcal{S}T(d,k)$, a differential equation (DE):

$$\dot{X}(t) = \Omega(t)X(t),$$  \hfill (3)

where $\Omega(t) = G(t)X(t)^\top - X(t)G(t)^\top$ and $G(t)$ is a standard gradient matrix (for unconstrained optimization) in $X(t)$ for a given function $F : \mathcal{S}T(d,k) \rightarrow \mathbb{R}$ to be maximized, defines a gradient-flow on $\mathcal{S}T(d,k)$. The flow can be discretized and integrated with the use of either the exponential map or the Cayley transform $Y$ as follows:

$$X_{i+1} = \Gamma(\eta\Omega(X_i, G_i))X_i,$$  \hfill (4)

where $G_i$ is a standard gradient matrix in $X_i$, $\Gamma = Y$ or $\Gamma = \exp$ and $\eta > 0$ is a step size.

Eq. 4 leads to an optimization algorithm with cubic time per optimization step (both exponential and Cayley transforms require cubic time), prohibitive in higher-dimensional applications. We propose our solution in the next section.

Interestingly, if $\mathcal{S}T(d,k) = \mathcal{O}(d)$ (i.e. $d = k$), the order of terms in the RHS of Equation 3 can be reversed and such DE also defines a gradient flow for $\Omega(X,G) = X^\top G - G^\top X$ that can be integrated via Equation 4, with the order of terms in RHS reversed. We refer to it as a reverse form.

3. Graph-Driven On-Manifold Optimization

Our algorithms provide an efficient way of computing discretized flows given by Eq. 3, by proposing graph-based techniques for time-efficient stochastic approximations of updates from Eq. 4. We focus on $\Gamma$ being an exponential map since it has several advantages over Cayley transform (in particular, it is surjective onto the space of all rotations).

We aim to replace $\Omega(X_i, G_i)$ with an unbiased sparser low-variance stochastic structured estimate $\hat{\Omega}$. This will enable us to replace cubic time complexity of an update with sub-cubic. Importantly, an unbiased estimator of $\hat{\Omega}$ does not lead to an unbiased estimator of $X_{i+1}$. However, as seen from Taylor expansion, the bias can be controlled by the step size $\eta$ and in practice will not hurt accuracy. We verify this claim both theoretically (Sec. 5) by providing strong convergence results, and empirically (Sec. 6).

A key observation linking our setting with graph theory is that elements of the Lie algebra of $\mathcal{O}(d)$ can be interpreted as adjacency matrices of weighted tournaments. Then, specific subsamplings of subtournaments lead to efficient updates. To explain this we need the following definitions.

**Definition 3.1** (weighted tournament). For $W \in \mathbb{S}(d)$, the weighted tournament $T(W)$ is a directed graph with vertices $\{0, 1, ..., d-1\}$ and edges $E(W)$, where $(i,j) \in E(W)$ if $W[i,j] > 0$. $W$ is called an adjacency matrix of $T(W)$. The induced undirected graph is denoted $G_{T(W)}$ and its set of edges $E(G_{T(W)})$. A subtournament $S$ of $T(W)$ is obtained by deleting selected edges of $T(W)$ and zeroing corresponding entries in the adjacency matrix $W$, to form an adjacency matrix of $S$ (denoted $W[S]$).

Assume we can rewrite $\Omega \in \mathbb{R}^{d \times d}$ as:

$$\Omega = \sum_{T \in \mathcal{T}} p_T \Omega_T,$$  \hfill (5)

where: $\mathcal{T}$ is a family of subtournaments of $T(\Omega)$, $\Omega_T \in \mathbb{S}(d)$ and $\{p_T\}_{T \in \mathcal{T}}$ is a discrete probability distribution on $\mathcal{T}$. Then Eq. 5 leads to the unbiased estimator $\hat{\Omega}$ of $\Omega$, when $\hat{\Omega} = \Omega_T$ with probability $p_T$. Such a decomposition is possible only if each edge in $E(\Omega)$ appears at least in one $T$. We call this a covering family.

For a covering family $\mathcal{T}$, the matrices $\Omega_T$ can be defined as:

$$\Omega_T = \frac{1}{p_T} M_T \odot \Omega[T],$$  \hfill (6)

where $M_T[i,j] > 0$ is the inverse of the number of $T \in \mathcal{T}$ such that $(i,j) \in E(G_T)$ and $\odot$ stands for the Hadamard product. We propose to replace the update from Eq. 4 with the following stochastic variant (with probability $p_T$):

$$X_{i+1} = \Gamma(\eta\Omega_T(X_i, G_i))X_i,$$  \hfill (7)

We focus on the family of unbiased estimators $\hat{\Omega}$ given by equations 5 and 6. In order to efficiently apply the above stochastic approach for on-manifold optimization, we need to construct a family of subtournaments $\mathcal{T}$ and a corresponding probabilistic distribution $\{p_T\}_{T \in \mathcal{T}}$ such that: (1) update of $X_i$ given $\Omega_T(X_i, G_i)$ can be computed in sub-cubic time, (2) sampling $\Omega_T$ via $\mathcal{P} = \{p_T\}$ can be conducted in sub-cubic time, (3) the stochastic update is accurate enough.
3.1. Multi-connected-components graphs

To achieve (1), we consider a structured family of subtournaments \( T \). For each \( T \in \mathcal{T} \), the corresponding undirected graph \( G_T \) consists of several connected components. Then the RHS of Eq. 7 can be factorized leading to an update:

\[
X_{i+1} = \left( \prod_{k=1}^{l} \Gamma(\eta \Omega_{T_k}(X_i, G_i)) \right) X_i,
\]

(8)

where \( l \) stands for the number of connected components and \( T_k \) is a subtournament of \( T \) such that \( G_{T_k} \) is \( G_T \)'s \( k^{th} \) connected component. We can factorize because matrices \( \{\Omega_{T_k}\}_{k=1,\ldots,l} \) all commute since they correspond to different connected components. We will construct the connected components to have the same number of vertices \( s = \frac{l}{t} \) (w.l.o.g. we can assume that \( l | d \)) such that the corresponding matrices \( \Omega_{T_k} \) have at most \( s \) non-zero columns/rows.

Time complexity of right-multiplying \( \Gamma(\eta \Omega_{T_k}(X_i, G_i)) \) by a \( d \times d \) matrix is \( O(ds^2) \) thus total time complexity of the update from Eq. 8 is \( T_{\text{update}} = O(ds^2) = O(d^2 s) \). Thus if \( s = o(d) \), we have: \( T_{\text{update}} = o(d^3) \). In fact the update from Eq. 8 can be further GPU-parallelized across \( l \) threads (since different matrices \( \Gamma(\eta \Omega_{T_k}(X_i, G_i)) \) modify disjoint subsets of entries of each column of the matrix that \( \Gamma(\eta \Omega_{T_k}(X_i, G_i)) \) is right-multiplied by) leading to total time complexity \( T_{\text{parallel}} = O(ds^2) \) after GPU-parallelization.

Let \( \mathcal{T}_s \) be the combinatorial space of all subtournaments \( T \) of \( \Omega \) such that \( G_T \) has \( s \)-size connected components. Note that for \( 2 \leq s < d \), \( \mathcal{T}_s \) is of size exponential in \( d \).

3.2. Sampling subtournaments

Points (2) and (3) are related to each other thus we address them simultaneously. We observe that sampling uniformly at random \( T \sim \text{Unif}(\mathcal{T}_s) \) is trivial: we randomly permute vertices \( \{0, 1, \ldots, d-1\} \) (this can be done in linear time, see: Python Fisher-Yates shuffle), then take first \( s \) to form first connected component, next \( s \) to form next one, etc. For the uniform distribution over \( \mathcal{T}_s \) we have: \( \frac{1}{|\mathcal{T}_s|} M_{\mathcal{T}_s} = \frac{d-s}{d-1} J_d \), where \( J_d \in \mathbb{R}^{d \times d} \) is all-one matrix (see Lemma 9.3 in the Appendix), thus we have: \( \Omega_T = \frac{d-s}{d-1} G_T \). We conclude that sampling \( \Omega_T \) can be done in time \( O(d^2 s) \).

Instead of \( \mathcal{T}_s \), one can consider a much smaller family \( \mathcal{T} \), where graphs \( G_T \) for different \( T \in \mathcal{T} \) consist of disjoint sets of edges (and every edge belongs to one \( G_T \). We call such \( \mathcal{T} \) non-intersecting. It is not hard to see that in that setting \( M_T = J_d \) (since edges are not shared across different graphs \( G_T \)). Also, the size of \( \mathcal{T} \) is at most quadratic in \( d \). Thus sampling \( \Omega_T \) can be conducted in time \( O(d^2) \).

In both cases, \( \mathcal{T} = \mathcal{T}_s \) and \( \mathcal{T} \) non-intersecting, we notice that \( M_T \propto J_d \). We call such a \( \mathcal{T} \) a homogeneous family.

3.2.1. Non-uniform distributions

To reduce variance of \( \hat{\Omega} \), uniform distributions should be replaced by non-uniform ones (detailed analysis of the variance of different methods proposed in this paper is given in the Appendix (Sec. 9.6, 9.7)). Intuitively, matrices \( \Omega_T \) for which graphs \( G_T \) have large absolute edge-weights should be prioritized and given larger probabilities \( p_T \). We denote by \( \|x\|_F \) the Frobenius norm and by \( w_e \) the weight of edge \( e \).

**Lemma 3.2** (importance sampling). Given \( T \), a distribution \( p_T^{\text{opt}} \) over \( T \) producing unbiased \( \Omega \) and minimizing variance \( \text{Var}(\hat{\Omega}) = \mathbb{E}[\|\hat{\Omega} - \Omega\|^2] \) satisfies: \( p_T^{\text{opt}} \sim \sqrt{\sum_{(i,j) \in E(G_T)} (M_T \odot \Omega) [i,j]^2} \). For homogeneous families, it simplifies to: \( p_T^{\text{opt}} \sim \sqrt{\sum_{e \in E(G_T)} w_e^{\alpha}} \).

From now on, we consider homogeneous families. Sampling from optimal \( \mathcal{P} \) is straightforward if \( \mathcal{T} \) is non-intersecting and can be conducted in time \( O(d^2) \), but becomes problematic for families \( \mathcal{T} \) of exponential sizes. We now introduce a rich family of distributions for which sampling can be conducted efficiently for homogeneous \( \mathcal{T} \).

**Definition 3.3** (h-regular distributions). For even function \( h : \mathbb{R} \to \mathbb{R}_+ \) such that \( h(0) = 0 \), we say that distribution \( \mathcal{P}^h(T) \) over \( T \) is \( h \)-regular if \( p_T^{\text{opt}} \sim \sum_{e \in E(G_T)} h(w_e) \).

The core idea is to sample uniformly at random, which we can do for \( \mathcal{T}_s \), but then accept sampled \( T \) with certain easily-computable probability \( q_T^h \). If \( T \) is not accepted, the procedure is repeated. Probability \( q_T^h \) should satisfy \( q_T^h = \lambda p_T^h \), for a renormalization term \( \lambda > 0 \).

**Algorithm 1** Constructing tournament \( T \sim \mathcal{P}^h(T) \) and \( \Omega_T \)

**Hyperparameters**: \( 0 < \alpha, \beta < 1 \) (only in version II)

**Input**: \( \Omega \in S_k(d) \)

**Output**: subtournament \( T \) and corresponding \( \Omega_T \)

**Preprocessing**: Compute \( \tau = \rho ||h(\Omega)||_1 \), where \( \rho = 1 \) (version I) or \( \rho = \frac{1}{d^{\beta \alpha}} \) (version II).

**while** True **do**

1. sample \( T \sim \text{Unif}(\mathcal{T}_s) \) (see: Sec. 3.2),
2. compute \( q_T^h = \frac{2h(G_T)}{\tau} \) (as in Lemma 3.5),
3. with probability \( q_T^h \) return \( \left( T, \frac{h(\Omega_T)}{2h(G_T)} G_T \right) \).

The larger \( \lambda \), the smaller the expected number of trials needed to accept sampled \( T \). Indeed, the following is true:

**Lemma 3.4**. The expected number of trials before sampled \( T \) is accepted is: \( \mathbb{E}[X] \), where \( |X| \) stands for the size of \( X \).

On the other hand, we must have: \( \lambda \leq \frac{1}{\max_T p_T^h} \). The following result enables us to choose large enough \( \lambda > 0 \), so that the expected number of trials remains small.

**Lemma 3.5**. For an \( h \)-regular distribution on \( \mathcal{T}_s \), probability \( P_T^h \) is given as: \( P_T^h = \frac{2h(G_T)}{\|h(\Omega_T)||_1} \leq \)
Stochastic Flows and Geometric Optimization on the Orthogonal Group

\[ \frac{1}{W}, \text{ where } h(G_T) = \sum_{c \in E(G_T)} h(w_c), h(\Omega) = \|h(\Omega[i,j])\|_{i,j \in \{0, 1, \ldots, d-1\}}, \|h(\Omega)\|_1 = \sum_{i,j} h(\Omega[i,j]) \text{ and } W = (d-2)! \left( (s-2)!(s!) \right)^{\frac{1}{2}} \left( \frac{d-2}{s} \right)! \]

Lemma 3.5 enables us to choose \( \lambda = W \) and consequently:

\[ q_T^h = \frac{2h(G_T)}{W||h(\Omega)||_1}. \]

We can try to do even better, by taking:

\[ \lambda = \frac{W||h(\Omega)||_1}{\tau^\gamma} \]

for any \( \tau \) such that \( \tau^\gamma = 2\max_{T \in \mathcal{T}} h(G_T) \leq \tau \leq ||h(\Omega)||_1, \) leading to:

\[ q_T^h = \frac{2h(G_T)}{W||h(\Omega)||_1}. \]

Efficiently finding a nontrivial (i.e. smaller than \( ||h(\Omega)||_1 \)) upper bounds on \( \tau^\gamma \) is not always possible, but one can trivially be done for \( (\alpha, \beta, h) \)-balanced matrices \( \Omega \). Note that at least an \( \alpha \)-fraction of all entries \( \Omega[i,j] \) of \( \Omega \) satisfy:

\[ h(\Omega[i,j]) \geq \beta \max_{a,b} h(\Omega[a,b]). \]

It is not hard to see that for such \( \Omega \) one can take:

\[ \tau = O\left( \frac{d^2\gamma}{||h(\Omega)||_1} \right) \]

or:

\[ \tau = O\left( \frac{d^2\gamma}{||h(\Omega)||_1} \right) \]

for \( \alpha^{-1}, \beta^{-1} = O(1) \). We observed (see: Section 9.2) that in practice one can often take \( \tau \) of that order. Our general method for sampling \( T \in \mathcal{T} \) and the corresponding \( \Omega_T \) is given in Alg. 1. We have:

**Theorem 3.6.** Alg. 1 outputs \( T \in \mathcal{T} \) from distribution \( \mathcal{P}^h(\Omega_T) \) corresponding \( \Omega_T \). Its expected time complexity is:

\[ O\left( \frac{d^2\gamma}{||h(\Omega)||_1} \right) + \xi, \]

where \( \gamma \) is time complexity for computing a fixed entry of \( \Omega \) and \( \xi \) for computing \( ||h(\Omega)||_1 \).

**3.2.2. Extensions**

**Dynamic domains \( T \):** One can consider changing sampling domains \( T \) across iterations of the optimization algorithm. If non-intersecting families are used (see: Sec. 3.2), \( T \) can be chosen at each step (or periodically) to minimize the optimal variance given by Lemma 3.2. Optimizing \( T \) is almost always a nontrivial combinatorial problem. We shed light on these additional intrinsic connections between combinatorics and on-manifold optimization in Sec. 9.7.

**Partition functions:** Our sampling scheme can be modified to apply to distributions from Lemma 3.2. The difficulty lies in obtaining a renormalization factor for probabilities:

\[ h_T^2 \sim \sqrt{\sum_{c \in E(G_T)} u_c^2} \]

which is a nontrivial graph partition function. However it can often be approximated by Monte Carlo methods (Jain et al., 2017). In practice we do not need to use variance-optimal optimizers to obtain good results.

**3.3. Time Complexity of the stochastic algorithm**

To summarize, we can conduct single optimization step given a sampled \( T \) and \( \Omega_T \) in time \( O(d^2s) \) or even \( O(ds^2) \) after further GPU-parallelization. If \( \Omega \) is given then, due to Theorem 3.6 and by previous analysis, entire sampling procedure can be done in \( O(d^2) \) time leading to sub-cubic complexity of the entire optimization step. If \( \Omega \) is not given, then notice first that computing a fixed entry of \( \Omega \) takes time \( O(k) \) for \( ST(d,k) \) and \( O(d) \) for \( SO(d) \). If sampling is conducted uniformly, then it invokes getting \( \frac{k}{d} = O(ds) \) edges of the graph (i.e. entries of \( \Omega \)) and thus total complexity is still sub-cubic. Furthermore, for \( ST(d,k) \) and \( k = o(d) \) that remains true even for non-uniform sampling due to Theorem 3.6 since Alg. 1 runs in \( O(d^2k) \) as opposed to \( O(d^3) \) time.

Finally, for \( O(d) \), \( SO(d) \) and when \( d = O(k) \), by Theorem 3.6, non-uniform sampling can be conducted in time \( O(d^3) \) for \( (\alpha, \beta, h) \)-balanced \( \Omega \). The first term is sub-cubic when \( \alpha^{-1} = O(\frac{1}{s}) \) as we confirm in practical applications (see: Sec. 9.2 and our discussion above). In practice for such \( \Omega \), the value \( ||h(\Omega)||_1 \) can be accurately estimated for arbitrarily small relative error \( \epsilon > 0 \) with probability \( p = 1 - O(\exp(-\alpha\beta^2)) \) by simple Monte Carlo procedure that approximates \( h(\Omega) \) with its sub-sampled version of only \( O(\tau) \) nonzero entries (see: Sec. 9.5.5). In practice one can choose \( r = o(d^2) \) resulting in \( \xi = o(d^3) \) and sub-cubic complexity of the entire step.

**4. Optimizing with Graph Matchings**

Note that if \( T \) is chosen in such a way that every connected component of \( G_T \) consists of two vertices then \( G_T \) is simply a matching (Diestel, 2012), i.e. a graph, where every vertex is connected to at most one more vertex. In particular, if \( s = 2 \), then \( \{G_T : T \in \mathcal{T}\} \) is a collection of all perfect matchings of \( G_T(\Omega) \), i.e. matchings, where every vertex belongs to some edge. For such \( T \)'s, the update rule given by Eq. 8 has particularly elegant form, namely:

\[ X_{i+1} = \prod_{k=1}^{d} C^{\theta_k}_{i_k,j_k} X_i \]

for some \( \theta_1, \ldots, \theta_l \in [0, 2\pi] \) and \( i_1, j_1, \ldots, i_k, j_k \). In other words, the update is encoded by the product of Gives rotations (see: Section 3.1.1). This sheds new light on recently proposed algorithms using products of Givens rotations to learn neural networks for RL (Choromanski et al., 2019) or to approximate Haar measure on \( O(d) \), (see: Sec. 9.9).

Constructing a good non-intersecting family \( T \) in this setting is intrinsically related to graph matching/edge coloring optimization problems since \( T \) can be obtained by iteratively finding heavy matchings (i.e. monochromatic classes of valid edge colorings) of \( G_T \). In Sec. 9.7 we explain these connections in more detail. Even though not central for our main argument, they provide additional deeper context.

**5. Convergence Results**

We show that our stochastic optimizers have similar convergence rates as deterministic ones. The difference is quantified by the term \( \sigma \) related to the variance of the estimator of \( \Omega \) (see below). Without loss of generality we consider optimization on \( O(d) \). Analogous results hold for \( ST(d,k) \).

**Theorem 5.1.** Let \( F : \mathbb{R}^{d \times d} \rightarrow \mathbb{R} \) be such that standard gradient \( \nabla F \) is defined on \( O(d) \) and for all \( M, N \in O(d) \),

\[ \|\nabla F(M) - \nabla F(N)\|_F \leq L\|M - N\|_F, \]

where...
where $L > 0$ \(^1\) and $\| \cdot \|_F$ is a Frobenius norm. Let $\{X_i\}_{i \geq 0}$ be the sequence generated by the proposed stochastic update. $X_0 \in \mathcal{O}(d)$ is fixed and $X_{i+1} := \exp(\eta_i \hat{\Omega}_i) X_i$, where $\hat{\Omega}_i$ is drawn from distribution $\mathbb{P}(\hat{\Omega}_i)$ defined on $\mathbb{S}(d)$ s.t. $\mathbb{E}\hat{\Omega}_i = \Omega_i$ and $\Omega_i := \nabla F(X_i)X_i^\top - X_i \nabla F(X_i)^\top$. Here $\{\eta_i \geq 0\}_{i \geq 0}$ is a sequence of step sizes. Then

$$\min_{i,T} \mathbb{E}\|\nabla_\mathcal{O} F(X_i)\|^2_F \leq 2 F^* - F(X_0) + \sum_{i=0}^T \eta_i$$

where $\Sigma_T = \sigma^2 \left( (2\sqrt{d} + 1)L + \|\nabla F(X_0)\|_F \right) \sum_{i=0}^T \eta_i$, $\nabla_\mathcal{O} F$ denotes a Riemannian gradient (see: Sec. 2.1.2), $F^* = \sup_{X \in \mathcal{O}(d)} F(X)$ and $\sigma^2 > 0$ is chosen so that $\forall \Omega \in \{\nabla_\mathcal{O} F(X)X^\top | X \in \mathcal{O}(d)\} : \sigma^2 \geq \mathbb{E}\|\hat{\Omega}\|^2_F$.

6. Experiments

6.1. RL with Evolution Strategies and RNNs

Here we demonstrate the effectiveness of our approach in optimizing policies for a variety of continuous RL tasks from the OpenAI Gym (Humanoid, Walker2d, HalfCheetah) and DM Control Suite (Reacher: Hard, HopperStand and Swimmer: 15).

![Figure 2](image1)

Figure 2. Visualizations of policies learned by different algorithms for Humanoid from OpenAI Gym.

We aim at jointly learn the RNN-based world model (Ha & Schmidhuber, 2018) and a policy affine in the latent (but not original) state, using evolution strategy optimization methods, recently proven to match or outperform SOTA policy gradient algorithms (Salimans et al., 2017). We conduct stochastic optimization on $\mathcal{O}(d)$ by constraining transition-matrices of RNNs to be orthogonal (see: Sec. 1).

**Compared methods:** Our algorithm (stoch-ortRNN) applying matching-based sampling with $h : x \rightarrow |x|$ is compared with three other methods: (1) its deterministic variant where exponentials are explicitly computed (exact-ortRNN), (2) unstructured vanilla RNNs (vanRNN). (3) vanilla RNN with orthogonal initialization of the transition matrix and periodic projections back into orthogonal group (ortinit-vanRNN). For the most challenging Humanoid task we also trained purely affine policies.

Comparing with (3) enables us to measure incremental gain coming from the orthogonal optimization as opposed to just orthogonal initialization which was proven to increase performance of neural networks (Saxe et al., 2014). We observed that just orthogonal initialization does not work well (is comparable to vanilla RNN) thus in ortinit-vanRNN we also periodically every $p$ iterations project back onto orthogonal group. We tested different $p$ and observed that in practice it needs to satisfy $p \leq 20$ to provide training improvements. We present variant with $p = 20$ in Fig. 3 since it is the fastest. Detailed ablation studies with different values of $p$ are given in Table 1.

![Figure 3](image2)

Figure 3. Comparison of all RNN-based algorithms on DM Control Suite (a-c) and OpenAI Gym (d-f) tasks: Each plot shows the mean + one std dev across $n = 10$ seeds. For Swimmer : 15 we present only three curves since vanRNN did not train at all.

---

\(^1\)As noted by Shalit & Chechik (2014), because $\mathcal{O}(d)$ is compact, any $F$ with a continuous second derivative will obey (10).
Experiment with affine policies helps us to illustrate that training nontrivial hidden state dynamics is indeed crucial. Purely affine policies were recently showed to provide high rewards for OpenAI environments (Mania et al., 2018), but we demonstrate that they lead to inferior agent’s behaviors, thus their rewards are deceptive. To do that, we simulated all learned policies for the most challenging 376-dimensional Humanoid environment (see video library in Appendix).

**Setting:** For each method we run optimization for $s = 10$ random seeds (resulting in 240 experiments) and hidden state of sizes $h = 200, 400$ (similar results in both settings). For all environments distributed optimization is conducted on 800 machines.

**Results:** In Fig. 2 we show most common policies learned by all four RNN-based algorithms and trained affine policy for the most challenging Humanoid task. Our method was the only one that consistently produced good walking/running behaviors. Surprisingly, most popular policies produced by exact-ortRNN while still effective, were clearly less efficient. The behavior further deteriorated if orthogonal optimization was replaced by ortinit-vanRNN. Pure vanilla RNNs produced unnatural movements, where agent started spinning and jumping on one leg. For affine policies forward movement was almost nonexistent. We do believe we are the first to apply and show gains offered by orthogonal RNN architectures for RL.

We conjecture that the superiority of our method over exact-ortRNN is partially due to numerical instabilities of standard Riemannian optimization on $O(d)$ (see: Appendix, Sec 9.1.2). In the Appendix (Sec. 9.3) we demonstrate that it is the case even for very low-dimensional tasks ($d = 16$).

In Fig. 3 we present corresponding training curves for all analyzed RNN-based methods. For all environments stochasticRNN method did well, in particular in comparison to exact-ortRNN. In Sec. 6.3 we show that stochasticRNN is much faster than other methods using ortho-constraints.

### 6.2. Optimization on $ST(d, k)$ for Vision-Based Tasks

Here we apply our methods to orthogonal convolutional neural networks (Jia et al., 2019; Huang et al., 2018; Xie et al., 2017; Bansal et al., 2018; Wang et al., 2019).

**Setting:** For MNIST, we used a 2-layer MLP with each layer of width 100, and tanh activations specifically to provide a simple example of the vanishing gradient problem, and to benchmark our stochastic orthogonal integrator. The matching-based variants of our algorithm with uniform sampling was accurate enough. For CIFAR10, we used a PlainNet-110 (ResNet-110 (He et al., 2016) without residual layers), similar to the experimental setting in (Xie et al., 2017). For a convolutional kernel of shape $[H, W, C_{in}, C_{out}]$ (denoting respectively [height, width, in-channel, out-channel]), we impose orthogonality on the flattened 2-D matrix of shape $[H \times W \times C_{in}, C_{out}]$ as commonly used in (Jia et al., 2019; Huang et al., 2018; Xie et al., 2017; Bansal et al., 2018; Wang et al., 2019). We applied our algorithm for $s \geq 2$ and uniform sampling. Further training details are in the Appendix 9.1.1.

**Results:** In Fig. 4, we find that the stochastic optimizer (when $s = 2$) is competitive with the exact orthogonal variant and outperforms vanilla training (SGD + Momentum). In Fig. 5, we present the best test accuracy curve found using a vanilla optimizer (SGD+Momentum), comparable to the best one from (Xie et al., 2017) (66% test accuracy). As in (Xie et al., 2017), we found training PlainNet-110 with vanilla Adam challenging. We observe that orthogonal optimization improves training. Even though stochastic optimizers for $s = 2$ are too noisy for this task, taking $s > 2$ (we used $s = \lceil \frac{d}{\log(d)} \rceil$) provides optimizers competitive with exact orthogonal. Next we present computational advantages of our algorithms over other methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>ES-200</th>
<th>ES-400</th>
<th>MNIST</th>
<th>CIFAR</th>
</tr>
</thead>
<tbody>
<tr>
<td>exact-ort</td>
<td>&gt; 1000</td>
<td>&gt; 12800</td>
<td>&gt; 200</td>
<td>&gt; 40K</td>
</tr>
<tr>
<td>stochastic-ort ($s = 2$)</td>
<td>&lt; 68</td>
<td>&lt; 272</td>
<td>&lt; 8.5</td>
<td>&lt; 2K</td>
</tr>
<tr>
<td>ortinit-vanRNN ($p = 20$)</td>
<td>&gt; 84</td>
<td>&gt; 656</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ortinit-vanRNN ($p = 10$)</td>
<td>&gt; 164</td>
<td>&gt; 1296</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ortinit-vanRNN ($p = 5$)</td>
<td>&gt; 334</td>
<td>&gt; 1616</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>ortinit-vanRNN ($p = 4$)</td>
<td>&gt; 491</td>
<td>&gt; 3216</td>
<td>N/A</td>
<td>N/A</td>
</tr>
<tr>
<td>stochastic-ort ($s = r^2$)</td>
<td>N/A</td>
<td>N/A</td>
<td>&lt; 43</td>
<td>&lt; 14K</td>
</tr>
</tbody>
</table>

Table 1: Comparison of average no of FLOPS [in $10^4$] for ort-
We introduced the first stochastic gradient flows algorithms. Adrian Weller acknowledges support from the David MacKay Newton research fellowship at Darwin College, The Alan Turing Institute under EPSRC grant EP/N510129/1 and U/B/000074, and the Leverhulme Trust via CFI.

To abstract from specific implementations, we compare number of FLOPS per iteration for updates of orthogonal matrices in different methods. See results in Table 1. We see that our optimizers outperform other orthogonal optimization methods, and preserve accuracy, as discussed above.

7. Conclusion
We introduced the first stochastic gradient flows algorithms for ML to optimize on orthogonal manifolds that are characterized by sub-cubic time complexity and maintain representational capacity of standard cubic methods. We provide strong connection with graph theory and show broad spectrum of applications ranging from CNN training for vision to learning RNN-based world models for RL.

8. Acknowledgements
Adrian MacKay acknowledges support from the David MacKay Newton research fellowship at Darwin College, The Alan Turing Institute under EPSRC grant EP/N510129/1 and U/B/000074, and the Leverhulme Trust via CFI.

References


Stochastic Flows and Geometric Optimization on the Orthogonal Group


