We present a new paradigm for Neural ODE algorithms, called ODEtoODE, where time-dependent parameters of the main flow evolve according to a matrix flow on the orthogonal group $O(d)$. This nested system of two flows, where the parameter-flow is constrained to lie on the compact manifold, provides stability and effectiveness of training and provably solves the gradient vanishing-explosion problem which is intrinsically related to training deep neural network architectures such as Neural ODEs. Consequently, it leads to better downstream models, as we show on the example of training reinforcement learning policies with evolution strategies, and in the supervised learning setting, by comparing with previous SOTA baselines. We provide strong convergence results for our proposed mechanism that are independent of the depth of the network, supporting our empirical studies. Our results show an intriguing connection between the theory of deep neural networks and the field of matrix flows on compact manifolds.

1 Introduction

Neural ODEs [13][10][27] are natural continuous extensions of deep neural network architectures, with the evolution of the intermediate activations governed by an ODE:

$$\frac{dx_t}{dt} = f(x_t, t, \theta),$$

(1)

parameterized by $\theta \in \mathbb{R}^n$ and where $f : \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}^n \rightarrow \mathbb{R}^d$ is some nonlinear mapping defining dynamics. A solution to the above system with initial condition $x_0$ is of the form:

$$x_t = x_0 + \int_{t_0}^{t} f(x_s, s, \theta)ds,$$

and can be approximated with various numerical integration techniques such as Runge-Kutta or Euler methods [48]. The latter give rise to discretizations:

$$x_{t+dt} = x_t + f(x_t, t, \theta)dt,$$
that can be interpreted as discrete flows of ResNet-like computations \[29\] and establish a strong connection between the theory of deep neural networks and differential equations. This led to successful applications of Neural ODEs in machine learning. Those include in particular efficient time-continuous normalizing flows algorithms \[11\] avoiding the computation of the determinant of the Jacobian (a computational bottleneck for discrete variants), as well as modeling latent dynamics in time-series analysis, particularly useful for handling irregularly sampled data \[44\]. Parameterized neural ODEs can be efficiently trained via adjoint sensitivity method \[13\] and are characterized by constant memory cost, independent of the depth of the system, with different parameterizations encoding different weight sharing patterns across infinitesimally close layers.

Such Neural ODE constructions enable deeper models than would not otherwise be possible with a fixed computation budget; however, it has been noted that training instabilities and the problem of vanishing/exploding gradients can arise during the learning of very deep systems \[43\], \[4\], \[23\].

To resolve these challenges for discrete recurrent neural network architectures, several improvements relying on transition transformations encoded by orthogonal/Hermitian matrices were proposed \[2\], \[33\]. Orthogonal matrices, while coupled with certain classes of nonlinear mappings, provably preserve norms of the loss gradients during backpropagation through layers, yet they also incur potentially substantial Riemannian optimization costs \[21\], \[14\], \[28\], \[11\]. Fortunately, there exist several efficient parameterizations of the subgroups of the orthogonal group \(O(d)\) that, even though in principle reduce representational capacity, in practice produce high-quality models and bypass Riemannian optimization \[36\], \[40\], \[34\]. All these advances address discrete settings and thus it is natural to ask what can be done for continuous systems, which by definition are deep.

In this paper, we answer this question by presenting a new paradigm for Neural ODE algorithms, called ODEtoODE, where time-dependent parameters of the main flow evolve according to a matrix flow on the orthogonal group \(O(d)\). Such flows can be thought of as analogous to sequences of orthogonal matrices in discrete structured orthogonal models. By linking the theory of training Neural ODEs with the rich mathematical field of matrix flows on compact manifolds, we can reformulate the problem of finding efficient Neural ODE algorithms as a task of constructing expressive parameterized flows on the orthogonal group. We show in this paper on the example of orthogonal flows corresponding to the so-called isospectral flows \[8\], \[9\], \[16\], that such systems studied by mathematicians for centuries indeed help in training Neural ODE architectures (see: ISO-ODEtoODEs in Sec. \[3.1\]). There is a voluminous mathematical literature on using isospectral flows as continuous systems that solve a variety of combinatorial problems including sorting, matching and more \[8\], \[9\], but to the best of our knowledge, we are the first who propose to learn them as stabilizers for training deep Neural ODEs.

Our proposed nested systems of flows, where the parameter-flow is constrained to lie on the compact manifold, provide stability and effectiveness of training, and provably solve the gradient vanishing/exploding problem for continuous systems. Consequently, they lead to better downstream models, as we show on a broad set of experiments (training reinforcement learning policies with evolution strategies and supervised learning). We support our empirical studies with strong convergence results, independent of the depth of the network. We are not the first to explore nested Neural ODE structure (see: \[55\]). Our novelty is in showing that such hierarchical architectures can be significantly improved by an entanglement with flows on compact manifolds.

To summarize, in this work we make the following contributions:

- We introduce new, explicit constructions of non-autonomous nested Neural ODEs (ODEtoODEs) where parameters are defined as rich functions of time evolving on compact matrix manifolds (Sec. \[5\]). We present two main architectures: gated-ODEtoODEs and ISO-ODEtoODEs (Sec. \[3.1\]).

- We establish convergence results for ODEtoODEs on a broad class of Lipschitz-continuous objective functions, in particular in the challenging reinforcement learning setting (Sec. \[4.2\]).

- We then use the above framework to outperform previous Neural ODE variants and baseline architectures on RL tasks from OpenAI Gym and the DeepMind Control Suite, and simultaneously to yield strong results on image classification tasks. To the best of our knowledge, we are the first to show that well-designed Neural ODE models with a compact number of parameters make them good candidates for training reinforcement learning policies via evolutionary strategies (ES) \[15\].

All proofs are given in the Appendix. We conclude in Sec. \[5\] and discuss broader impact in Sec. \[7\].
2 Related work

Our work lies in the intersection of several fields: the theory of Lie groups, Riemannian geometry and deep neural systems. We provide an overview of the related literature below.

Matrix flows. Differential equations (DEs) on manifolds lie at the heart of modern differential geometry [17] which is a key ingredient to understanding structured optimization for stabilizing neural network training [14]. In our work, we consider in particular matrix gradient flows on $\mathcal{O}(d)$ that are solutions to trainable matrix DEs. In order to efficiently compute these flows, we leverage the theory of compact Lie groups that are compact smooth manifolds equipped with rich algebraic structure [15] [7] [32]. For efficient inference, we apply local linearizations of $\mathcal{O}(d)$ via its Lie algebra (skew-symmetric matrices vector spaces) and exponential mappings [50] (see: Sec. 3).

Hypermnetworks. An idea to use one neural network (hypermnetwork) to provide weights for another network [26] can be elegantly adopted to the nested Neural ODE setting, where [55] proposed to construct time-dependent parameters of one Neural ODE as a result of concurrently evolving another Neural ODE. While helpful in practice, this is insufficient to fully resolve the challenge of vanishing and exploding gradients. We expand upon this idea in our gated-ODEtoODE network, by constraining the hypernetwork to produce skew-symmetric matrices that are then translated to those from the orthogonal group $\mathcal{O}(d)$ via the aforementioned exponential mapping.

Stable Neural ODEs. On the line of work for stabilizing training of Neural ODEs, [22] proposes regularization based on optimal transport, while [37] adds Gaussian noise into the ODE equation, turning it into a stochastic dynamical system. [20] lifts the ODE to a higher dimensional space to prevent trajectory intersections, while [25] [39] add additional terms into the ODE, inspired by well known physical dynamical systems such as Hamiltonian dynamics.

Orthogonal systems. For classical non-linear networks, a broad class of methods which improve stability and generalization involve orthogonality constraints on the layer weights. Such methods range from merely orthogonal initialization [46] [32] and orthogonal regularization [3] [53] to methods which completely constrain weight matrices to the orthogonal manifold throughout training using Riemannian optimization [55], by projecting the unconstrained gradient to the orthogonal group $\mathcal{O}(d)$ [47]. Such methods have been highly useful in preventing exploding/vanishing gradients caused by long term dependencies in recurrent neural networks (RNNs) [30] [31] [52] [12] note that dynamical isometry can be preserved by orthogonality, which allows training networks of very large depth.

3 Improving Neural ODEs via flows on compact matrix manifolds

Our core ODEtoODE architecture is the following nested Neural ODE system:

$$\begin{align*}
\dot{x}_t &= f(W_t x_t) \\
W_t &= W_t b_\psi(t, W_t)
\end{align*}$$

(2)

for some function $f : \mathbb{R} \rightarrow \mathbb{R}$ (applied elementwise), and a parameterized function: $b_\psi : \mathbb{R} \times \mathbb{R}^{d \times d} \rightarrow \text{Skew}(d)$, where Skew($d$) stands for the vector space of skew-symmetric (antisymmetric) matrices in $\mathbb{R}^{d \times d}$. We take $W_0 \in \mathcal{O}(d)$, where the orthogonal group $\mathcal{O}(d)$ is defined as: $\mathcal{O}(d) = \{ M \in \mathbb{R}^{d \times d} : M^T M = I_d \}$. It can be proven [55] that under these conditions $W_t \in \mathcal{O}(d)$ for every $t \geq 0$. In principle ODEtoODE can exploit arbitrary compact matrix manifold $\Sigma$ (with $b_\psi$ modified respectively so that second equation in Formula [2] represents on-manifold flow), yet in practice we find that taking $\Sigma = \mathcal{O}(d)$ is the most effective. Furthermore, for $\Sigma = \mathcal{O}(d)$, the structure of $b_\psi$ is particularly simple, it is only required to output skew-symmetric matrices. Schematic representation of the ODEtoODE architecture is given in Fig. 1. We take as $f$ a function that is non-differentiable in at most finite number of its inputs and such that $|f'(x)| = 1$ on all others (e.g. $f(x) = |x|$).

3.1 Shaping ODEtoODEs

An ODEtoODE is defined by: $x_0$, $W_0$ (initial conditions) and $b_\psi$. Vector $x_0$ encodes input data e.g. state of an RL agent (Sec. 5.1) or an image (Sec. 5.2). Initial point $W_0$ of the matrix flow can be either learned (see: Sec. 3.2) or sampled upfront uniformly at random from Haar measure on $\mathcal{O}(d)$. We present two different parameterizations of $b_\psi$, leading to two different classes of ODEtoODEs:

ISO-ODEtoODEs: Those leverage a popular family of the isospectral flows (i.e. flows preserving matrix spectrum), called double-bracket flows, and given as: $\dot{H}_t = [H_t, [H_t, N]]$, where:
Figure 1: Schematic representation of the discretized ODEtoODE architecture. On the left: nested ODE structure with parameters of the main flow being fed with the output of the matrix flow (inside dashed contour). On the right: the matrix flow evolves on the compact manifold $\Sigma$, locally linearized by vector space $T$ and with mapping $\Gamma$ encoding this linearization ($O_t = b_\psi(t, W_t)$).

$$Q \equiv H_0, N \in \text{Sym}(d) \subseteq \mathbb{R}^{d \times d}, \text{Sym}(d) \text{ stands for the class of symmetric matrices in } \mathbb{R}^{d \times d} \text{ and } [[ ]] \text{ denotes the Lie bracket: } [A, B] \equiv AB - BA. \text{ Double-bracket flows with customized parameter-matrices } Q, N \text{ can be used to solve various combinatorial problems such as sorting, matching, etc.} \text{[9]. It can be shown that } H_t \text{ is similar to } H_0 \text{ for every } t \geq 0, \text{ thus we can write: } H_t = W_t H_0 W_t^{-1}, \text{ where } W_t \in \mathcal{O}(d). \text{ The corresponding flow on } \mathcal{O}(d) \text{ has the form: } \dot{W}_t = W_t ([(W_t)^\top QW_t, N] \text{ and we take: } b_\psi^{\text{ gated}}(t, W_t) \equiv ([W_t]^\top QW_t, N], \text{ where } \psi = (Q, N) \text{ are learnable symmetric matrices. It is easy to see that } b_\psi \text{ outputs skew-symmetric matrices.}

\text{Gated-ODEtoODEs: } \text{ In this setting, inspired by } \text{[10][55], we simply take } b_\psi^{\text{ gated}} = \sum_{i=1}^{d} a_i B^i_\psi, \text{ where } d \text{ stands for the number of gates, } a = (a_1, ..., a_d) \text{ are learnable coefficients and } B^i_\psi \equiv f^{\psi}_i - (f^{\psi}_i)^\top. \text{ Here } \{f^{\psi}_i\}_{i=1, ..., a} \text{ are outputs of neural networks parameterized by } \psi \text{ s and producing unstructured matrices and } \psi = \text{concat}(a, \psi^1, ..., \psi^d). \text{ As above, } b_\psi^{\text{ gated}} \text{ outputs matrices in } \text{Skew}(d).

3.2 Learning and executing ODEtoODEs

Note that most of the parameters of ODEtoODE from Formula 2 are unstructured and can be trained via standard methods. To train an initial orthogonal matrix $W_0$, we apply tools from the theory of Lie groups and Lie algebras [35]. First, we notice that $\mathcal{O}(d)$ can be locally linearized via skew-symmetric matrices $\text{Skew}(d)$ (i.e. a tangent vector space $T_W$ to $\mathcal{O}(d)$ in $W \in \mathcal{O}(d)$ is of the form: $T_W = W \cdot \text{Skew}(d)$). We then compute Riemannian gradients which are projections of unstructured ones onto tangent spaces. For the inference on $\mathcal{O}(d)$, we apply exponential mapping $\Gamma(W) \equiv \exp(\eta b_\psi(t, W))$, where $\eta$ is the step size, leading to the discretization of the form: $W_{t+dt} = W_t \Gamma(W_t)$ (see: Fig. 1).

4 Theoretical results

4.1 ODEtoODEs solve gradient vanishing/explosion problem

First we show that ODEtoODEs do not suffer from the gradient vanishing/explosion problem, i.e. the norms of loss gradients with respect to intermediate activations $x_t$ do not grow/vanish exponentially while backpropagating through time (through infinitesimally close layers).

Lemma 4.1 (ODEtoODEs for gradient stabilization). Consider a Neural ODE on time interval $[0, T]$ and given by Formula 2. Let $\hat{L} = \hat{L}(x_T)$ be a differentiable loss function. The following holds for any $t \in [0, 1]$, where $e = 2.71828...$ is Euler constant:

$$\frac{1}{e} \| \frac{\partial \hat{L}}{\partial x_T} \|_2 \leq \| \frac{\partial \hat{L}}{\partial x_t} \|_2 \leq e \| \frac{\partial \hat{L}}{\partial x_T} \|_2. \quad (3)$$
4.2 ODEtoODEs for training reinforcement learning policies with ES

Here we show strong convergence guarantees for ODEtoODE. For the clarity of the exposition, we consider ISO-ODEtoODE, even though similar results can be obtained for gated-ODEtoODE. We focus on training RL policies with ES [45], which is mathematically more challenging to analyze than supervised setting, since it requires applying ODEtoODE several times throughout the rollout. Analogous results can be obtained for the conceptually simpler supervised setting.

Let $\text{env} : \mathbb{R}^d \times \mathbb{R}^m \rightarrow \mathbb{R}^d \times \mathbb{R}$ be an "environment" function such that it gets current state $s_k \in \mathbb{R}^d$ and action encoded as a vector $a_k \in \mathbb{R}^m$ as its input and outputs the next state $s_{k+1}$ and the next score value $l_{k+1} \in \mathbb{R}$: $(s_{k+1}, l_{k+1}) = \text{env}(s_k, a_k)$. We treat env as a "black-box" function, which means that we evaluate its value for any given input, but we don’t have access to env’s gradients. An overall score $L$ is a sum of all per-step scores $l_k$ for the whole rollout from state $s_0$ to state $s_K$:

$$L(s_0, a_0, \ldots, a_{K-1}) = \sum_{k=1}^{K} l_k, \quad \forall k \in \{1, \ldots, K\} : (s_k, l_k) = \text{env}(s_{k-1}, a_{k-1}).$$  

We assume that $s_0$ is deterministic and known. The goal is to train a policy function $g_\theta : \mathbb{R}^d \rightarrow \mathbb{R}^m$ which, for the current state vector $s \in \mathbb{R}^d$, returns an action vector $a = g_\theta(s)$ where $\theta$ are trained parameters. By Stiefel manifold $S\mathcal{T}(d_1, d_2)$ we denote a nonsquare extension of orthogonal matrices:

$$\text{if } d_1 \geq d_2 \text{ then } S\mathcal{T}(d_1, d_2) = \{ \Omega \in \mathbb{R}^{d_1 \times d_2} | \Omega \Omega^T = I \},$$

We define $g_\theta$ as an ODEtoODE with Euler discretization given below:

$$g_\theta(s) = \Omega_2 x_N, \quad x_0 = \Omega_1 s, \quad \forall i \in \{1, \ldots, N\} : x_i = x_{i-1} + \frac{1}{N} f(W, x_{i-1} + b)$$

$$W_i = G_{i-1} \exp \left( \frac{1}{N} (W_{i-1}^T Q W_{i-1} N - NW_{i-1}^T Q W_{i-1}) \right) \in \mathcal{O}(d), \quad \theta = \{ \Omega_1 \in S\mathcal{T}(n, d), \Omega_2 \in S\mathcal{T}(m, n), b \in \mathbb{R}^n, N \in \mathbb{S}(n), Q \in \mathbb{S}(n), W_0 \in \mathcal{O}(n) \} \in \mathcal{D}$$

where by $\mathcal{D}$ we denote $\theta$’s domain: $\mathcal{D} = S\mathcal{T}(n, d) \times S\mathcal{T}(m, n) \times \mathbb{R}^n \times \mathbb{S}(n) \times \mathbb{S}(n) \times \mathcal{O}(n)$. Define a final objective $F : \mathcal{D} \rightarrow \mathbb{R}$ to maximize as

$$F(\theta) = L(s_0, a_0, \ldots, a_{K-1}), \quad \forall k \in \{1, \ldots, K\} : a_{k-1} = g_\theta(s_{k-1}).$$

For convenience, instead of $(s_{out}, l) = \text{env}(s_{in}, a)$ we will write $s_{out} = \text{env}^{(1)}(s_{in}, a)$ and $l = \text{env}^{(2)}(s_{in}, a)$. In our subsequent analysis we will use a not quite limiting Lipschitz-continuity assumption on env which intuitively means that a small perturbation of env’s input leads to a small perturbation of its output. In addition to that, we assume that per-step loss $l_k = \text{env}^{(2)}(s_{k-1}, a_{k-1})$ is uniformly bounded.

Assumption 4.2. There exist $M, L_1, L_2 > 0$ such that for any $s', s'' \in \mathbb{R}^d, a', a'' \in \mathbb{R}^m$ it holds:

$$|\text{env}^{(2)}(s', a')| \leq M, \quad \|\text{env}^{(1)}(s', a') - \text{env}^{(1)}(s'', a'')\|_2 \leq L_1 \delta,$$

$$|\text{env}^{(2)}(s', a') - \text{env}^{(2)}(s'', a'')| \leq L_2 \delta, \quad \delta = ||s' - s''||_2 + ||a' - a''||_2.$$  

Assumption 4.3. $f(\cdot)$ is Lipschitz-continuous with a Lipschitz constant $1$: $\forall x', x'' \in \mathbb{R} : |f(x') - f(x'')| \leq |x' - x''|$. In addition to that, $f(0) = 0$.

Instead of optimizing $F(\theta)$ directly, we opt for optimization of a Gaussian-smoothed proxy $F_{\sigma}(\theta)$:

$$F_{\sigma}(\theta) = \mathbb{E}_{\epsilon \sim \mathcal{N}(0, I)} F(\theta + \sigma \epsilon)$$

where $\sigma > 0$. Gradient of $F_{\sigma}(\theta)$ has a form:

$$\nabla F_{\sigma}(\theta) = \frac{1}{\sigma} \mathbb{E}_{\epsilon \sim \mathcal{N}(0, I)} \nabla F(\theta + \sigma \epsilon) \epsilon,$$

which suggests an unbiased stochastic approximation such that it only requires to evaluate $F(\theta)$, i.e. no access to $F$’s gradient is needed, where $v$ is a chosen natural number:

$$\tilde{\nabla} F_{\sigma}(\theta) = \frac{1}{\sigma v} \sum_{w=1}^{v} F(\theta + \sigma \epsilon_w) \epsilon_w, \quad \epsilon_1, \ldots, \epsilon_v \sim \text{i.i.d. } \mathcal{N}(0, I),$$
By defining a corresponding metric, $D$ becomes a product of Riemannian manifolds and a Riemannian manifold itself. By using a standard Euclidean metric, we consider a Riemannian gradient of $F_\sigma(\theta)$:

$$
\nabla_{\mathbb{R}} F_\sigma(\theta) = \{ \nabla_{\Omega_1} \Omega_1^T - \Omega_1 \nabla_{\Omega_1} \nabla_{\Omega_2} \Omega_2^T - \Omega_2 \nabla_{\Omega_2} + \nabla_b, \frac{1}{2}(\nabla_N + \nabla_Q)^T, \frac{1}{2}(\nabla_Q + \nabla_Q) \}, \quad (8)
$$

$$
\nabla_{W_0} W_0^T - W_0 \nabla_{W_0}, \text{ where } \{ \nabla_{\Omega_1}, \nabla_{\Omega_2}, \nabla_b, \nabla_N, \nabla_Q, \nabla_{W_0} \} = \nabla F_\sigma(\theta). \quad (9)
$$

We use Stochastic Riemannian Gradient Descent \cite{6} to maximize $F(\theta)$. Stochastic Riemannian gradient estimate $\nabla_{\mathbb{R}} F_\sigma(\theta)$ is obtained by substituting $\nabla \nabla \nabla$ in \cite{8-9}. The following Theorem proves that SRGD is converging to a stationary point of the maximization objective $F(\theta)$ with rate $O(\tau^{-0.5+\epsilon})$ for any $\epsilon > 0$. Moreover, the constant hidden in $O(\tau^{-0.5+\epsilon})$ rate estimate doesn’t depend on the length $N$ of the rollout.

**Theorem 1.** Consider a sequence $\{\theta^{(\tau)}\} = \{\{\Omega^{(\tau)}_1, \Omega^{(\tau)}_2, b^{(\tau)}, N^{(\tau)}, Q^{(\tau)}, W_0^{(\tau)}\} \in \mathbb{D}\}_{\tau=0}^\infty$ where $\theta^{(0)}$ is deterministic and fixed and for each $\tau > 0$:

$$
\Omega^{(\tau)}_1 = \exp(\alpha_\tau \tilde{\nabla}_{\mathbb{R}, \Omega_1}^{(\tau)}), \quad \Omega^{(\tau)}_2 = \exp(\alpha_\tau \tilde{\nabla}_{\mathbb{R}, \Omega_2}^{(\tau)}), \quad b^{(\tau)} = b^{(\tau-1)} + \alpha_\tau \tilde{\nabla}_{\mathbb{R}, b}^{(\tau)},
$$

$$
N^{(\tau)} = N^{(\tau-1)} + \alpha_\tau \tilde{\nabla}_{\mathbb{R}, N}^{(\tau)}, \quad Q^{(\tau)} = Q^{(\tau-1)} + \alpha_\tau \tilde{\nabla}_{\mathbb{R}, Q}^{(\tau)}, \quad W_0^{(\tau)} = \exp(\alpha_\tau \tilde{\nabla}_{\mathbb{R}, W_0}^{(\tau)}),
$$

$$
\{ \tilde{\nabla}_{\mathbb{R}, \Omega_1}^{(\tau)}, \tilde{\nabla}_{\mathbb{R}, \Omega_2}^{(\tau)}, \tilde{\nabla}_{\mathbb{R}, b}^{(\tau)}, \tilde{\nabla}_{\mathbb{R}, N}^{(\tau)}, \tilde{\nabla}_{\mathbb{R}, Q}^{(\tau)}, \tilde{\nabla}_{\mathbb{R}, W_0}^{(\tau)} \} = \tilde{\nabla}_{\mathbb{R}} F_\sigma(\theta^{(\tau)}), \quad \alpha_\tau = \tau^{-0.5}.
$$

Then $\min_{0 \leq \tau' < \tau} \mathbb{E}[\|\nabla_{\mathbb{R}} F_{\sigma}(\theta^{(\tau')})\|_2^2 | F_{\tau, D, D_b}] \leq \mathcal{E} \cdot \tau^{-0.5+\epsilon}$, for any $\epsilon > 0$ where $\mathcal{E}$ doesn’t depend on $N$ and $D, D_b$ are constants and $F_{\tau, D, D_b}$ denotes a condition that for all $\tau' \leq \tau$ it holds $\|N^{(\tau')}\|_2, \|Q^{(\tau')}\|_2 < D, \|b^{(\tau')}\|_2 < D_b$.

5 Experiments

We run two sets of experiments comparing ODEtoODE with several other methods in the supervised setting and to train RL policies with ES. To the best of our knowledge, we are the first to propose Neural ODE architectures for RL-policies and explore how the compactification of the number of parameters they provide can be leveraged by ES methods that benefit from compact models \cite{15}.

5.1 Neural ODE policies with ODEtoODE architectures

5.1.1 Basic setup

In all Neural ODE methods we were integrating on the time interval $[0, T]$ for $T = 1$ and applied discretization with integration step size $\eta = 0.04$ (in our ODEtoODE we used that $\eta$ for both: the main flow and the orthogonal flow on $\Omega(d)$). The dimensionality of the embedding of the input state $s$ was chosen to be $h = 64$ for OpenAI Gym Humanoid (for all methods but HyperNet, where we chose $h = 16$, see: discussion below) and $h = 16$ for all other tasks.

Neural ODE policies were obtained by a linear projection of the input state into embedded space and Neural ODE flow in that space, followed by another linear projection into action-space. In addition to learning the parameters of the Neural ODEs, we also trained their initial matrices and those linear projections. Purely linear policies were proven to get good rewards on OpenAI Gym environments \cite{38}, yet they lead to inefficient policies in practice \cite{14}; thus even those environments benefit from deep nonlinear policies. We enriched our studies with additional environments from Deep Mind Control Suite. We used standard deviation stddev = 0.1 of the Gaussian noise defining ES perturbations, ES-gradient step size $\delta = 0.01$ and function $\sigma(x) = |x|$ as a nonlinear mapping. In all experiments we used $k = 200$ perturbations per iteration \cite{15}.

**Number of policy parameters:** To avoid favoring ODEtoODEs, the other architectures were scaled in such a way that they have similar (but not smaller) number of parameters. The ablation studies were conducted for them across different sizes and those providing best (in terms of the final reward) mean curves were chosen. No ablation studies were run on ODEtoODEs.
5.1.2 Tested methods

We compared the following algorithms including standard deep neural networks and Neural ODEs:

**ODEtoODE:** Matrices corresponding to linear projections were constrained to be taken from Stiefel manifold $ST(d)$ [14] (a generalization of the orthogonal group $O(d)$). Evolution strategies (ES) were applied as follows to optimize entire policy. Gradients of Gaussian smoothings of the function: $F: \mathbb{R}^m \rightarrow \mathbb{R}$ mapping vectorized policy (we denote as $m$ the number of its parameters) to
obtained reward were computed via standard Monte Carlo procedure [15]. For the part of the gradient vector corresponding to linear projection matrices and initial matrix of the flow, the corresponding Riemannian gradients were computed to make sure that their updates keep them on the respective manifolds [14]. The Riemannian gradients were then used with the exact exponential mapping from $\text{Skew}(d)$ to $\mathcal{O}(d)$. For unconstrained parameters defining orthogonal flow, standard ES-update procedure was used [45]. We applied ISO-ODEtoODE version of the method (see: Sec. 3.1).

**Deep(Res)Net:** In this case we tested unstructured deep feedforward fully connected (ResNet) neural network policy with $t = 25$ hidden layers.

**BaseODE:** Standard Neural ODE $\frac{dx(t)}{dt} = f(x_t)$, where $f$ was chosen to be a feedforward fully connected network with with two hidden layers of size $h$.

**NANODE:** This leverages recently introduced class of Non-Autonomous Neural ODEs (NANODEs) [19] that were showed to substantially outperform regular Neural ODEs in supervised training [19]. NANODEs rely on flows of the form: $\frac{dx(t)}{dt} = \sigma(W_t x_t)$. Entries of weight matrices $W_t$ are values of $d$-degree trigonometric polynomials [19] with learnable coefficients. In all experiments we used $d = 5$. We observed that was an optimal choice and larger values of $d$, even though improving model capacity, hurt training when number of perturbations was fixed (as mentioned above, in all experiments we used $k = 200$).

**HyperNet:** For that method [55], matrix $W_t$ was obtained as a result of the entangled neural ODE in time $t$ after its output de-vectorization (to be more precise: the output of size $256$ was reshaped into a matrix from $\mathbb{R}^{16 \times 16}$). We encoded the dynamics of that entangled Neural ODE by a neural network $f$ with two hidden layers of size $s = 16$ each. Note that even relatively small hypernetworks are characterized by large number of parameters which becomes a problem while training ES policies. We thus used $h = 16$ for all tasks while running HyperNet algorithm. We did not put any structural assumptions on matrices $W_t$, in particular they were not constrained to belong to $\mathcal{O}(d)$.

**5.1.3 Discussion of the results**

The results are presented in Fig. 2. Our ODEtoODE is solely the best performing algorithm on four out of eight tasks: Humanoid, HalfCheetah, Humanoid_Stand and Swimmer_15 and is one of the two best performing on the remaining four. It is clearly the most consistent algorithm across the board and the only one that learns good policies for Humanoid. Each of the remaining methods fails on at least one of the tasks, some such as: HyperNet, DeepNet and DeepResNet on more. Exponential mapping (Sec. 3.2) computations for ODEtoODEs with hidden representations $h \leq 64$ took negligible time (we used well-optimized scipy.linalg.expm function). Thus all the algorithms but HyperNet (with expensive hypernetwork computations) had similar running time.

**5.2 Supervised learning with ODEtoODE architectures**

We also show that ODEtoODE can be effectively applied in supervised learning by comparing it with multiple baselines on various image datasets.

**5.2.1 Basic setup**

All our supervised learning experiments use the Corrupted MNIST [41] (11 different variants) dataset. For all models in Table 1, we did not use dropout, applied hidden width $w = 128$, and trained for 100 epochs. For all models in Table 2, we used dropout with $r = 0.1$ rate, hidden width $w = 256$, and trained for 100 epochs. For ODEtoODE variants, we used a discretization of $\eta = 0.01$.

**5.2.2 Tested methods**

We compared our ODEtoODE approach with several strong baselines: feedforward fully connected neural networks (Dense), Neural ODEs inspired by [13] (NODE), Non-Autonomous Neural ODEs [19] (NANODE) and hypernets [55] (HyperNet). For NANODE, we vary the degree of the trigonometric polynomials. For HyperNet, we used its gated version and compared against gated-ODEtoODE (see: Sec. 3.1). In this experiment, we focus on fully-connected architecture variants of all models. As discussed in prior work, orthogonality takes on a different form in the convolutional case [49], so we reserve discussion of this framing for future work.

**5.2.3 Discussion of the results**

The results are presented in Table 1 and Table 2 below. Our ODEtoODE outperforms other model variants on 11 out of the 15 tasks in Table 1. On this particularly task, we find that ODEtoODE-1,
whereby we apply a constant perturbation to the hidden units $\theta(0)$ works best. We highlighted the best results for each task in **bolded blue**.

### Table 1: Test accuracy comparison of different methods. Postfix terms refer to hidden depth for Dense, trigonometric polynomial degree for NANOODE, and number of gates for HyperNet and ODEtoODE.

<table>
<thead>
<tr>
<th>Models</th>
<th>Dense-1</th>
<th>Dense-10</th>
<th>NODE</th>
<th>NANOODE-1</th>
<th>NANOODE-10</th>
<th>HyperNet-1</th>
<th>HyperNet-10</th>
<th>ODEtoODE-1</th>
<th>ODEtoODE-10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dotted Lines</td>
<td>92.99</td>
<td>88.22</td>
<td>91.54</td>
<td>92.42</td>
<td>92.74</td>
<td>91.88</td>
<td>91.9</td>
<td>95.42</td>
<td>95.22</td>
</tr>
<tr>
<td>Spatter</td>
<td>93.54</td>
<td>89.52</td>
<td>92.49</td>
<td>93.13</td>
<td>93.15</td>
<td>93.19</td>
<td>93.28</td>
<td>94.9</td>
<td>94.9</td>
</tr>
<tr>
<td>Stripe</td>
<td>30.55</td>
<td>20.57</td>
<td>36.76</td>
<td>16.4</td>
<td>21.37</td>
<td>19.71</td>
<td>18.69</td>
<td>44.51</td>
<td>44.37</td>
</tr>
<tr>
<td>Translate</td>
<td>25.8</td>
<td>24.82</td>
<td>27.09</td>
<td>28.97</td>
<td>29.31</td>
<td>29.42</td>
<td>29.3</td>
<td>26.82</td>
<td>26.63</td>
</tr>
<tr>
<td>Rotate</td>
<td>82.8</td>
<td>80.38</td>
<td>82.76</td>
<td>83.19</td>
<td>83.65</td>
<td>83.5</td>
<td>83.6</td>
<td>84.1</td>
<td></td>
</tr>
<tr>
<td>Scale</td>
<td>58.68</td>
<td>62.73</td>
<td>62.05</td>
<td>66.63</td>
<td>66.63</td>
<td>67.84</td>
<td>68.11</td>
<td>66.68</td>
<td>66.76</td>
</tr>
<tr>
<td>Shear</td>
<td>91.73</td>
<td>89.52</td>
<td>91.82</td>
<td>93.11</td>
<td>93.33</td>
<td>92.48</td>
<td>93.37</td>
<td>92.93</td>
<td></td>
</tr>
<tr>
<td>Motion Blur</td>
<td>78.16</td>
<td>67.25</td>
<td>75.18</td>
<td>76.53</td>
<td>79.22</td>
<td>78.82</td>
<td>78.33</td>
<td>77.58</td>
<td></td>
</tr>
<tr>
<td>Glass Blur</td>
<td>91.62</td>
<td>84.89</td>
<td>87.94</td>
<td>90.18</td>
<td>91.07</td>
<td>91.17</td>
<td>93.91</td>
<td>93.29</td>
<td></td>
</tr>
<tr>
<td>Shot Noise</td>
<td>98.16</td>
<td>91.63</td>
<td>94.24</td>
<td>94.97</td>
<td>94.73</td>
<td>94.76</td>
<td>94.83</td>
<td>96.91</td>
<td>96.71</td>
</tr>
<tr>
<td>Identity</td>
<td>97.55</td>
<td>95.73</td>
<td>97.61</td>
<td>97.65</td>
<td>97.69</td>
<td>97.72</td>
<td>97.54</td>
<td>97.94</td>
<td>97.88</td>
</tr>
</tbody>
</table>

### Table 2: Additional Sweep for the setting as in Table 1. This time all models also incorporate dropout with rate $r = 0.1$ and width $w = 256$. As above in Table 1 ODEtoODE variants achieve the highest performance.

#### 6 Conclusion
In this paper, we introduced nested Neural ODE systems, where the parameter-flow evolves on the orthogonal group $O(d)$. Constraining this matrix-flow to develop on the compact manifold provides us with an architecture that can be efficiently trained without exploding/vanishing gradients, as we showed theoretically and demonstrated empirically by presenting gains on various downstream tasks. We are the first to demonstrate that algorithms training RL policies and relying on compact models’ representations can significantly benefit from such hierarchical systems.

#### 7 Broader impact
We believe our contributions have potential broader impact that we briefly discuss below:

**Reinforcement Learning with Neural ODEs:** To the best of our knowledge, we are the first to propose to apply nested Neural ODEs in Reinforcement Learning, in particular to train Neural ODE policies. More compact architectures encoding deep neural network systems is an especially compelling feature in policy training algorithms, in particular while combined with ES methods admitting embarrassingly simple and efficient parallelization, yet suffering from high sampling complexity that increases with the number of policy parameters. Our work shows that RL training of such systems can be conducted efficiently provided that evolution of the parameters of the system is highly structured and takes place on compact matrix manifolds.

**Learnable Isospectral Flows:** We demonstrated that ISO-ODEtoODEs can be successfully applied to learn reinforcement learning policies. Those rely on the isospectral flows that in the past were demonstrated to be capable of solving combinatorial problems ranging from sorting to (graph) matching [18, 54, 59]. As emphasized before, such flows are however fixed and not trainable whereas we learn ours. That suggests that isospectral flows can be potentially learned to solve combinatorially-flavored machine learning problems or even integrated with non-combinatorial blocks in larger ML computational pipelines. The benefits of such an approach lie in the fact that we can efficiently backpropagate through these continuous systems and is related to recent research on differentiable sorting [18, 59, 24].

#### 8 Acknowledgements
Adrian Weller acknowledges support from the David MacKay Newton research fellowship at Darwin College, The Alan Turing Institute under EPSRC grant EP/N510291/1 and U/B/000074, and the Leverhulme Trust via CFI. Valerii Likhosherstov acknowledges support from the Cambridge Trust and DeepMind.
References


