Faster Policy Learning with Continuous-Time Gradients

Samuel Ainsworth   SKAINSWO@CS.WASHINGTON.EDU
Kendall Lowrey   KLOWREY@CS.WASHINGTON.EDU
John Thickstun   THICKSTN@CS.WASHINGTON.EDU
Zaid Harchaoui   ZAID@CS.WASHINGTON.EDU
Siddhartha Srinivasa   SIDDH@CS.WASHINGTON.EDU
Paul G. Allen School of Computer Science and Engineering, University of Washington

Abstract

We study the estimation of policy gradients for continuous-time systems with known dynamics. By reframing policy learning in continuous-time, we show that it is possible to construct a more efficient and accurate gradient estimator. The standard back-propagation through time estimator (BPTT) computes exact gradients for a crude discretization of the continuous-time system. In contrast, we approximate continuous-time gradients in the original system. With the explicit goal of estimating continuous-time gradients, we are able to discretize adaptively and construct a more efficient policy gradient estimator which we call the Continuous-Time Policy Gradient (CTPG). We show that replacing BPTT policy gradients with more efficient CTPG estimates results in faster and more robust learning in a variety of control tasks and simulators.

Keywords: differentiable physics, neural ODEs, reinforcement learning, optimal control, policy optimization

1. Introduction

Many robotic control problems can be reduced to finding a desirable trajectory through a physical system governed by piece-wise smooth dynamics, e.g. control of ground vehicles, robot arms, quadrotors, and so forth. When the dynamics of a system are known we can efficiently optimize a trajectory via first-order methods, constructed using explicit gradients of the dynamics. We can use these gradients to solve open-loop trajectory optimization problems (Carius et al., 2019; Orsolino et al., 2018; Carius et al., 2018) or to train a closed-loop feedback controller, i.e. a policy (Hirose et al., 2019; Faulkner et al., 2018; Kim et al., 2018). In this paper we focus on the latter problem of policy learning, although our insights can be applied equally well to trajectory optimization.

Computing exact policy gradients for most continuous-time dynamical systems proves intractable. Policy gradient algorithms rely on discrete numerical methods to estimate a robot's trajectory under a policy, and the sensitivity of this trajectory to the robot's actions dictated by the policy. The standard numerical estimate of policy gradients is back-propagation through time (BPTT); this algorithm abandons study of the continuous problem and immediately discretizes time, computing exact policy gradients with respect to discretized dynamics. The discretization step-size controls the accuracy of BPTT, resulting in a tradeoff between the accuracy of the gradient estimates and the computational cost of computing an estimate.

In this paper we defer discretization and begin by characterizing the continuous-time policy gradient. Rather than computing an exact gradient of a discretized system, we instead compute an approximate gradient of the continuous system. When we discretize to compute this approximation, we do so with the explicit goal of constructing an efficient estimate of the continuous-time policy

gradient. This approach takes inspiration from the numerical ODE literature, as well as the neural ODE (Chen et al., 2018). We are motivated by the following questions:

Q1: Can we compute more accurate estimates of the policy gradient? BPTT introduces numerical error by discretizing the robot trajectory under a policy. This error injects variance into the learning process, which could slow down learning. By computing a more accurate estimate of the policy gradient, can we optimize a policy in fewer iterations?

Q2: Can we compute policy gradient estimates more efficiently? Suppose we want a target level of accuracy for our policy gradient estimates. We can achieve the target accuracy by controlling the discretization step-size for BPTT. By using better numerical algorithms, can we achieve the same accuracy with a smaller computational budget?

We give affirmative answers to both **Q1** and **Q2**. We propose a new policy gradient estimator (Section 3) which we call the Continuous-Time Policy Gradient (CTPG). This estimator strictly improves upon the BPTT estimator, in the sense that the tradeoff curve between gradient accuracy and computational budget for CTPG Pareto-dominates the tradeoff curve for BPTT. In the spirit of algorithms like SGD, we ask: is it possible to sacrifice accuracy of the gradient estimates in order to optimize more efficiently? We find that a certain amount of numerical accuracy is required of our policy gradient estimator in order for optimization to succeed: unlike SGD, poor numerical estimates of policy gradients are not unbiased. However, in a variety of experimental settings (Section 4) we demonstrate that replacing expensive BPTT estimates with relatively inexpensive CTPG estimates of comparable accuracy results in faster overall learning.

2. Learning with Policy Gradients

Our goal is to learn how to act within a deterministic physical system while minimizing a cost functional. Given a model of the dynamics $f : \mathbb{R}^d \times \mathbb{R}^k \to \mathbb{R}^d$ and time-varying control inputs (actions) $u : [0, \infty) \to \mathbb{R}^k$, the state of the system $x : [0, \infty) \to \mathbb{R}^d$ is governed by a first-order ordinary differential equation

$$\frac{dx(t)}{dt} = f(x(t), u(t)). \tag{1}$$

Guided by a local cost (reward function) $w : \mathbb{R}^d \times \mathbb{R}^k \to \mathbb{R}$ of taking action u in state x, and a terminal cost $J : \mathbb{R}^d \to \mathbb{R}$, we seek controls u that minimize the global cost over a trajectory of length T:

$$\begin{aligned} \underset{u(t)}{\text{minimize}} \quad & \int_0^T w(x(t), u(t)) \, dt + J(x(T)) \\ \text{subject to} \quad & \frac{dx(t)}{dt} = f(x(t), u(t)). \end{aligned} \tag{2}$$

We are interested in learning feedback controllers in the form of policies $u(t) = \pi_\theta(x(t))$ with parameters θ. For any fixed initial conditions $x(0) = x_0$, the optimization Eq. (2) is a trajectory optimization problem with parameters θ. For any fixed policy $\pi_\theta(x(t))$, computation of the trajectory $x(\cdot)$ is an initial value problem that is completely determined by the starting state x_0. The global loss of following the policy π_θ along a trajectory $x(\cdot)$ is given by

$$\mathcal{L}(x(0), \theta) \triangleq \int_0^T w(x(t), \pi_\theta(x(t))) \, dt + J(x(T)). \tag{3}$$
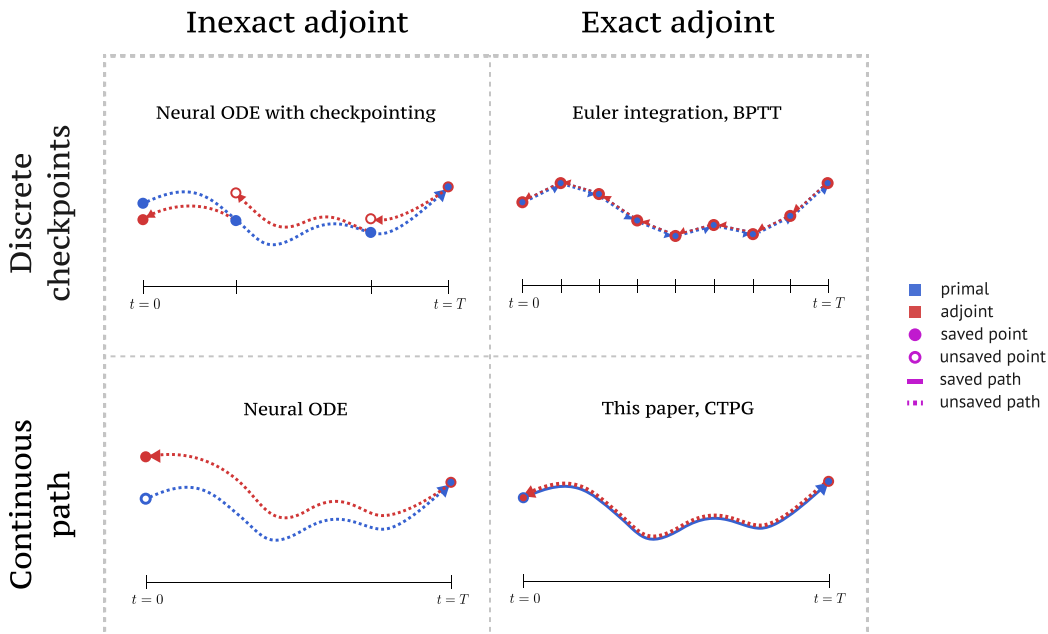
Figure 1: Comparing policy gradient estimators. Neural ODE (Sec. 3.3) approximates continuous trajectories, but accumulated error in its adjoint estimates. Neural ODE with checkpointing mitigates this error, but still accumulates error between checkpoints. Euler integration with BPTT (Sec. 3.2) stores a discretized trajectory, which prevents error accumulation, at the cost of a naive discretization. CTPG (Sec. 3.1) discretizes adaptively, and stores a spline approximation of the trajectory to avoid error accumulation in the adjoint.

We seek to learn a policy that generalizes over a distribution of initial states $x_0 \sim \rho_0$, motivating us to minimize the expected loss $\mathcal{L}(\theta) \triangleq \mathbb{E}_{x_0 \sim \rho_0} \mathcal{L}(x_0, \theta)$.

Adjoint sensitivity analysis (Pontryagin et al., 1964) characterizes $\frac{\partial \mathcal{L}(x_0, \theta)}{\partial \theta}$ in a form that is amenable to computational approximation. Consider the adjoint process $\alpha(t) \triangleq \frac{\partial \mathcal{L}(x_0, \theta)}{\partial x(t)}$; the evolution of this adjoint process for a trajectory x can be described by the dynamics

$$\frac{d\alpha(t)}{dt} = -\alpha(t)^\top \frac{\partial f}{\partial x(t)} - \frac{dw(x(t), \pi_\theta(x(t)))}{dx(t)}. \tag{4}$$

These dynamics can be compared to Chen et al. (2018), with the notable distinction that the loss in Eq. (3) is path-dependent. This causes the instantaneous loss $w(x(t), u(t))$ to accumulate continuously in the adjoint process $\alpha(t)$. The adjoint process itself is constrained by the final value $\alpha(T) = \frac{\partial J}{\partial x(T)}$. Sensitivity of the loss to the parameters θ is given by

$$\frac{\partial \mathcal{L}(x_0, \theta)}{\partial \theta} = \int_0^T \alpha(t)^\top \frac{\partial f}{\partial u} \frac{\partial u}{\partial \theta} + \frac{\partial w}{\partial u} \frac{\partial \pi_\theta(x(t))}{\partial \theta} \, dt. \tag{5}$$

The integral in Eq. (5) is amenable to estimation using various numerical techniques, each of which is susceptible to different forms of numerical error; a stylized illustration of the behaviour of various gradient estimators is presented in Figure 1.

3. Numerical Policy Gradient Estimators

Policy gradients can be calculated via the integral described in Eq. (5). This integral in turn depends upon the trajectory $x(t)$ and the adjoint process $\alpha(t)$. To estimate a policy gradient, we require solutions to the following three sub-problems:

1. The trajectory $x(t)$, governed by the dynamics in Eq. (1).

2. The adjoint process $\alpha(t)$, governed by the dynamics in Eq. (4).

3. The policy gradient accrued along $x(t)$ and $\alpha(t)$, described by the integral in Eq. (5).

The accumulated θ gradients, Eq. (5), can be described as the value at $t = 0$ of the process $g(t)$, with final value $g(T) = 0$, defined by the dynamics $\frac{dg(t)}{dt} = -\alpha(t)^\top \frac{\partial f}{\partial u} \frac{\partial u}{\partial \theta} - \frac{\partial w}{\partial u} \frac{\partial \pi_\theta(x(t))}{\partial \theta}$. Collectively, the trajectory process $x(t)$, the corresponding adjoint process $\alpha(t)$, and the accumulated loss $g(t)$ comprise the following two-point boundary value problem (BVP):

Initial Value	Dynamics		Final Value	

$$x(0) = x_0, \qquad \frac{dx(t)}{dt} = f(x(t), u(t)), \tag{6}$$

$$\frac{d\alpha(t)}{dt} = -\alpha(t)^\top \frac{\partial f}{\partial x} - \frac{dw}{dx(t)}, \qquad \alpha(T) = \frac{dJ}{dx(T)}, \tag{7}$$

$$\frac{dg(t)}{dt} = -\alpha(t)^\top \frac{\partial f}{\partial u} \frac{\partial \pi_\theta}{\partial \theta} - \frac{\partial w}{\partial u} \frac{\partial \pi_\theta}{\partial \theta}, \qquad g(T) = 0. \tag{8}$$

To understand how various subsystems of the policy gradient estimator interact, we highlight quantities computed by the physics simulator in red and quantities computed by automatic (or symbolic) differentiation in blue.

The most common approach to estimating $g(0) = \frac{\partial \mathcal{L}(x_0, \theta)}{\partial \theta}$ in the controls community is back-propagation through time (BPTT). The idea is to discretize the dynamics $x(t)$ with a constant time step h, after which sensitivities to the policy parameters θ are computed by the standard backprop-agation algorithm. In Section 3.2 we describe how this algorithm can be interpreted as an approximate solution to the continuous boundary-value problem, using Euler integration to estimate the dynamics $x(t)$, $\alpha(t)$, and $g(t)$. The numerical ODE community widely eschews Euler integration in favor of more sophisticated solvers. A natural question arises: can we improve upon naive Euler integration to construct better gradient estimates?

In Section 3.1 we introduce a new, general class of gradient estimators for this boundary value problem that we call the Continuous-Time Policy Gradient. In Section 3.2 we show how BPTT can be viewed as an instantiation of Continuous-Time Policy Gradients, using Euler integration as a numerical solver. The CTPG generalization allows us to replace Euler integration with a more sophisticated solver; in our experiments we opted for Runge-Kutta (RK4) and adaptive Adams-Moulton methods but these solvers can be seamlessly replaced by whatever solver is most suitable for a particular problem. In Section 3.3 we discuss the Neural ODE estimator and its inadequacy for use with stabilizing control problems. And in Appendix C we discuss alternate methods for solving the BVP that may be of interest in specialized settings.

3.1. Continuous-Time Policy Gradients

We propose to solve the boundary value problem described by Eqs. (6), (7), and (8) using a forward-backward meta-algorithm, which is analogous to (but distinct from) the forward and backward passes of an automatic differentiation system (and more generally, the forward-backward approach to dynamic programming). First, we use a numerical solver to estimate the solution to the initial value problem given by Eq. (6) (the forward pass). Crucially we store the estimated trajectory for later use, incurring a storage cost off $O(s)$ where s is the number of steps visited by the numerical solver. Second, we use a numerical solver to estimate the solution to initial value problem given by Eq. (7) (the backward pass) using the cached trajectory computed in the forward pass. As we compute the backward pass we accumulate an estimate of the integral in Eq. (8), the total sensitivity of the loss to θ along the estimated trajectory. Details are presented in Algorithm 1.

Figure 2 illustrates performance of Continuous-Time Policy Gradients (using the RK4 solver) and several other algorithms. The dynamics are given by the linear-quadratic regulator (LQR) and the policy is fixed to be the optimal LQR policy. By adjusting the error tolerance of the RK4 solver, we construct a trade-off curve between the accuracy of the CTPG estimates and the computational cost of computing an estimate. The CTPG estimator Pareto-dominates the other estimators. In this sense, we argue that the other estimators are inadmissible: e.g. for any BPTT estimator, there is a CTPG estimator that is either (1) more accurate at the same level of performance or (2) more efficient at the same level of accuracy.

Figure 2: Approaching the Pareto frontier: Euler integration and other methods are Pareto dominated by CTPG. Lower and to the left is better.

The effectiveness of CTPG is attributable to the efficiency of the RK4 solver, which can estimate the trajectory and adjoint paths while visiting many fewer states than a comparably accurate Euler solver (BPTT). This also means that CTPG is more memory efficient than BPTT: both algorithms require storage that grows linearly in the number of states visited in the forward pass. The (possibly surprising) under-performance of the Neural ODE estimator is explained in Section 3.3.

3.2. Backpropagation Through Time

Backpropagation through time (BPTT) is simply the application of reverse-mode automatic differentiation (AD) to computations involving a "time" axis. Like CTPG, AD proceeds with a forward and backward pass. In the forward pass, the trajectory $x(t)$ is approximated using discrete, fixed-length linearized steps (see Figure 1); this is equivalent to using the Euler numerical integration method to solve the initial value problem Eq. (6). States computed in the forward pass are stored, incurring an $O(1/h)$ memory cost, and avoiding the need to recompute these values in the backward pass. In the backward pass of AD computes adjoints of the forward computation graph, using the
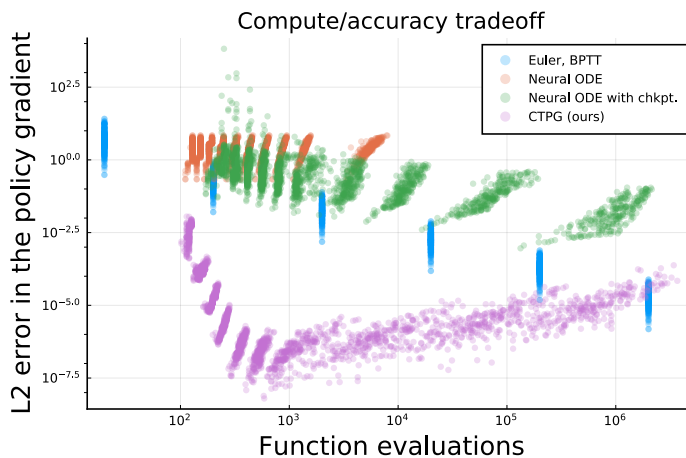

Algorithm 1 Continuous-Time Policy Gradients

Given: Differentiable physics simulator $f(x, u)$, cost/reward function $w(x, u)$, and a numerical ODE solver Solve[initial_conditions, dynamics]

Input: Policy $\pi_\theta(x)$, initial state x_0

Result: An approximation of $\frac{\partial \mathcal{L}(x_0, \theta)}{\partial \theta}$

Forward pass: (compute and store an approximation of the trajectory $x : [0, T] \to \mathbb{R}^d$)
$$\tilde{x}(\cdot) \leftarrow \text{Solve}\left[x(0) = x_0, \frac{dx(t)}{dt} = f(x, \pi_\theta(x))\right].$$

Backward pass: (compute an approximation of the Pontryagin adjoints $\alpha : [T, 0] \to \mathbb{R}^d$)
$$\tilde{\alpha}(\cdot) \leftarrow \text{Solve}\left[\alpha(T) = \frac{dJ}{d\tilde{x}(T)}, \frac{d\alpha(t)}{dt} = -\alpha(t)^\top \frac{\partial f}{\partial \tilde{x}(t)} - \frac{dw}{d\tilde{x}(t)}\right].$$

Return:
$$\int_0^T \tilde{\alpha}(t)^\top \frac{\partial f}{\partial u} \frac{\partial u}{\partial \theta} + \frac{\partial w}{\partial u} \frac{\partial \pi_\theta}{\partial \theta} \, dt \quad \text{as an approximation of } \frac{\partial \mathcal{L}(x_0, \theta)}{\partial \theta}.$$

---

values $x(t)$ stored during the forward pass. This is equivalent to using Euler integration to solve the final value problems described by Eqs. (7) and (8).

Backpropagation is an exact algorithm, in the sense that it computes exact gradients of the loss for the discretized dynamics. But these dynamics are only an approximation of the continuous trajectory $x(\cdot)$, and therefore the gradients only approximate the desired value $g(0)$. Computing an accurate estimate of the continuous trajectory using Euler integration requires an excessively small step size h, and a correspondingly large memory allocation. Furthermore, the resulting computation graph is very deep, which could make learning difficult (Quaglino et al., 2020).

3.3. The Neural ODE Algorithm

To efficiently compute the backward pass, we rely on a stored spline of states s visited from the forward pass, leading to an algorithm with $O(s)$ space complexity. For many physical systems, the dynamics $f(x, u)$ are "invertible"; this invites us to consider algorithms with constant space complexity, recomputing the state variables $x(t)$ in the backward pass following the reverse dynamics $-f(x, u)$ (Chen et al., 2018). Constant-memory algorithms are intriguing, but we observe that their results are numerically unstable for stabilizing control problems. We visualize this instability for an LQR system in Figure 3.

In principle, a trajectory $x(t)$ is uniquely defined by either its initial value $x(0)$ thanks to the Picard-Lindelöf theorem. But for control problems that seek to stabilize the system towards some fixed goal-point, it is numerically unstable to reconstruct the trajectory from its final state; moreover, this instability becomes more pronounced as the policy learns to more effectively achieve its goal, and computing the inverse trajectory becomes increasingly chaotic.

To better formalize this notion, we appeal to linear stability theory. For a system $\frac{dx}{dt} = \phi(x)$, linear stability analysis suggests inspecting the eigenspectrum of $\frac{d\phi}{dx}$: if $\frac{d\phi}{dx}$ has all eigenvalues with negative real part the system is said to be linearly stable, and if there exists an eigenvalue with positive real part it is said to be linearly unstable. Stabilizing control problems by their nature demand solutions such that the system has negative eigenvalues $-\lambda \ll 0$, as small as possible
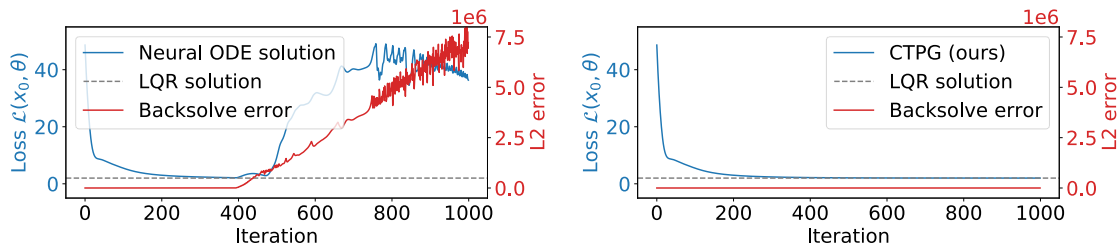
Figure 3: Left: The Neural ODE estimator used to learn an LQR policy. The trajectory loss $\mathcal{L}(x_0, \theta)$ is shown in blue, and the discrepancy $||x(0) - \tilde{x}(0)||^2$ is shown in red where $\tilde{x}(t)$ is the "recovered" path taken by the adjoint process. The discrepancy between the forward and reverse paths is negligible until the control learns to stabilize the system, at which point it becomes impossible to recover $x(0)$ from $x(T)$. Right: The same problem, but with CTPG. Unlike the Neural ODE, CTPG records a lightweight spline estimate of $x(t)$.

subject to control costs. This becomes catastrophic in the reverse-direction when we are faced with a linearly unstable system with eigenvalues $\lambda \gg 0$! Furthermore, as we discuss in Appendix B, the neural ODE backpropagation dynamics are in fact linearly unstable everywhere.

4. Experiments

To better understand CTPG's behavior, we compare CTPG with BPTT for a variety of control problems. We restrict our experiments to settings in which first-order policy gradients optimization can effectively discover the optimal policy; extension of these local search algorithms to global policy optimization is beyond the scope of this work. We begin with a relatively straightforward control problem in Sec. 4.1 for which we wrote a simple differentiable simulator. We then evaluate CTPG for a task defined in an existing differentiable physics engine from Hu et al. (2020) in Sec. 4.2. In Sec. 4.3 we evaluate CTPG for a task defined in the MuJoCo physics simulator, using finite difference approximations to the gradients of the physical system. We conclude in Sec. 4.4 with a challenging quadrotor control problem running in a purpose-built differentiable flight simulator. Code to reproduce our experiments is available at `https://github.com/samuela/ctpg`.

4.1. Differential Drive Robot

The differential drive is a "Roomba-like" robot with two wheels that are individually actuated to rotate or advance the robot. We use the dynamics defined in LaValle (2006), but control the torque applied to each wheel rather than directly controlling their angular velocities. We initialized the robot to random positions and rotations throughout the plane and learn a policy that drives it to the origin. See Figure 4 for a visualization of results. Both CTPG and BPTT are able to learn working policies, but CTPG is able to do so more efficiently. Videos of the training process are available in the supplementary material.

4.2. DiffTaichi Electric Field Control

To test CTPG's ability to work with third-party differentiable physics simulators, we integrated and compared against the DiffTaichi (meta) physics engine (Hu et al., 2020, 2019). DiffTaichi (now part of Taichi) is a differentiable, domain-specific language for writing physics simulators. We
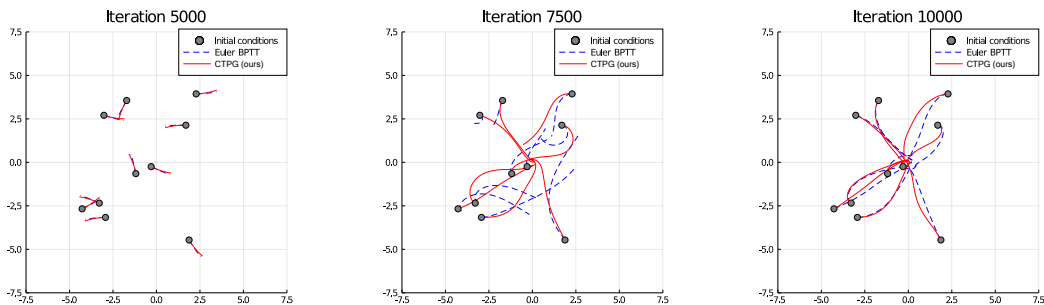
Figure 4: Stages of training policies to control a differential drive robot to stabilize towards the origin. Each curve denotes the trajectory of the robot in the (x, y) plane under the respective policies. A full video is available in the supplementary material.

tested against their electric field control experiment in which eight fixed electrodes placed in a 2-D square modulate their charge driving a red ball with a static charge around the square. The red ball experiences electrostatic forces given by Coulomb's law, $\mathbf{F} = k_e \frac{q_1 q_2}{|\mathbf{r}|^2} \hat{\mathbf{r}}$ (Zangwill, 2013).

We evaluate performance both in terms of wallclock time and in terms of the number of oracle function calls made to the simulator, namely the combined evaluations of f, $\partial f / \partial x$, and $\partial f / \partial u$. Calls to the simulator constitute the vast majority of the run time, so we find this to be the most accurate hardware-independent measure of performance. We present the results in Fig. 5. We find that CTPG reliably outperforms BPTT both in terms of wallclock time and number of function evaluations, even with DiffTaichi's exact BPTT configuration.
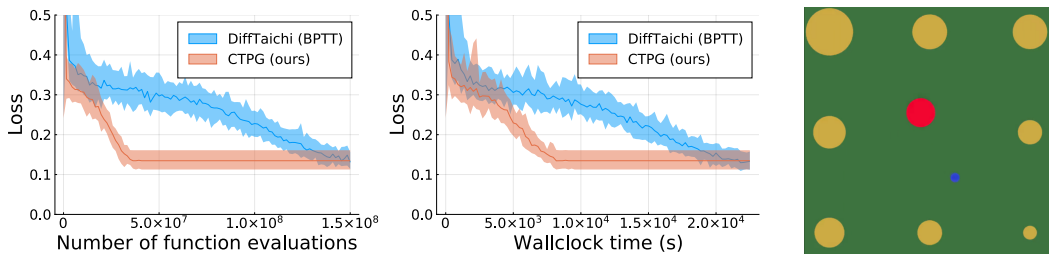

Figure 5: The DiffTaichi electric experiment involves modulating the charges of the 8 yellow electrodes to push the red charge to follow the blue dot (right). We found that CTPG performance was dominant in terms of the number of function evaluations (left) as well as wallclock time (center) across 32 random trials.

4.3. MuJoCo Cartpole

MuJoCo (Todorov et al., 2012) is a widely used physics simulator in both the robotics and reinforcement learning communities. In contrast to DiffTaichi (Section 4.2) MuJoCo models are not automatically differentiable, but rather they are designed to be finite differenced. We demonstrate that BPTT and CTPG can be applied to such black-box physics engines. These finite difference
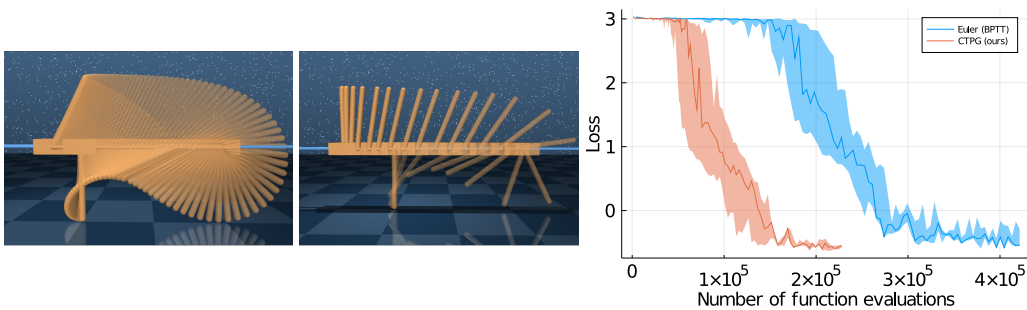
Figure 6: Visualization of all states visited by a policy trained with BPTT (left) and CTPG (middle). Fewer states are visited by CTPG during training leading to more efficient learning (right). Error bars on the training plot show the 0.25-0.75 quantile range of 5 random seeds.

derivatives are used in Eqs. 7 and 8 alongside the automatically differentiated policy derivatives. We test on the Cartpole swing-up task specified by the dm_control suite (Tunyasuvunakool et al., 2020), a classic non-linear control problem. This model was re-implemented in the Lyceum framework (**?**) with the Euler integration timestep used for BPTT kept the same as the model specification (0.01 seconds). Total dynamics evaluations included those used for finite differencing.

Results are presented in Fig. 6. We find that CTPG needs significantly fewer dynamics evaluations than BPTT per learning instance. While both techniques eventually reach a similar level of performance for this task, CTPG demonstrates substantial efficiency gains.

4.4. Quadrotor Control

Quadrotors are notoriously difficult to control, especially given the strict real-time compute requirements of a flying aircraft, and tradeoffs between compute power, weight, and power draw. As such, they make for a strong use case of the policy learning tools that CTPG brings to bear. We test CTPG's ability to learn a small, efficient neural network controller that flies a quadrotor.

We implemented a differentiable quadrotor simulator based on the dynamics from Sabatino (2015), and train policies to stabilize a quadrotor

Figure 7: CTPG learns policies to stabilize the quadrotor towards the origin, while Euler integration and BPTT–with the same number of function evaluations–can not.

at the origin from a wide distribution of initial positions, orientations, and velocities. Our experimental setup is similar to that of Tang et al. (2018). Results are presented in Fig. 7. As a flying robot, the quadrotor dynamics are inherently unstable. As a result, we found that Euler integration is often not able to integrate trajectories to sufficient accuracy for policy learning using a similar sample budget to CTPG.
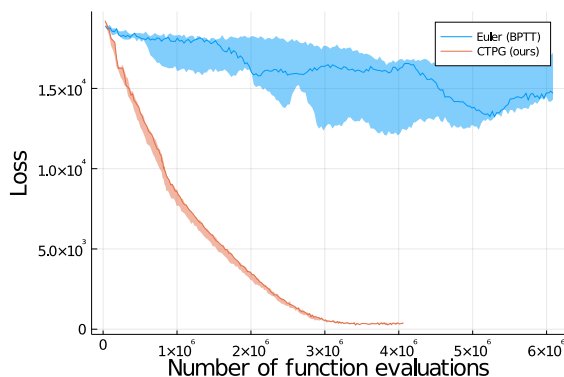
5. Related Work

Neural ODEs. Chen et al. (2018) introduced neural ordinary differential equations, based on the adjoint sensitivity analysis method of Pontryagin et al. (1964) and using backwards dynamics to reconstruct $x(t)$ as described in Eq. 6. Recent work by Du et al. (2020) and **?** studied using Neural ODEs for system identification as opposed to policy learning. Gholaminejad et al. (2019) explored a gradient checkpointing scheme for Neural ODEs.

Differentiable physics. Differentiable physics engines have recently attracted attention as a richer alternative to conventional "black-box" simulators. de Avila Belbute-Peres et al. (2018) introduced a differentiable rigid-body physics simulator based on analytic derivatives through physics defined via a linear complementarity problem. Qiao et al. (2020) expanded on the types of physical interactions that could be efficiently simulated and differentiated. Li et al. (2020, 2019) and Battaglia et al. (2016) proposed developing differentiable physics engines by learning them from data. Schenck and Fox (2017, 2018) built a differentiable simulator for liquids with applications to robotics. Hu et al. (2019, 2020) created a differentiable domain-specific language for the creation of physics simulators, enabling backpropagation through a variety of different physical phenomena.

Trajectory optimization and control theory. Pontryagin et al. (1964) introduced necessary conditions for optimality of a continuous trajectory, drawing from the calculus of variations. An alternative perspective, the Hamilton-Jacobi-Bellman equation, provides necessary and sufficient conditions for optimality of policies (Speyer and Jacobson, 2010). Prior work describes the analytic adjoints for backpropagation through time (Tedrake; Robinson and Failside, 1987; **?**; Mozer, 1989). Unlike the majority of work in trajectory optimization and control, we use Pontryagin's maximum principle to learn global feedback policies instead of fitting single trajectories. Concurrently Jin et al. (2020) explored differentiating through Pontryagin's maximum principle. However, they still considered standard time discretizations, eg. BPTT. Other works have investigated similar forward-backward schemes for solving the adjoint equations (McAsey et al., 2012).

Numerical methods for ODEs. The study of numerical methods for solving ODEs has a long and rich history, dating back to Euler's method (Hairer et al., 1993). In this paper we leveraged higher-order explicit solvers with adaptive step sizes, especially Runge-Kutta (Wang et al., 2021; Tsitouras, 2011) and Adams-Moulton methods (Shampine and Watts, 1979; LeVeque, 2007). For a thorough treatment of ODE solvers we refer the reader to Hairer et al. (1993).

6. Discussion

We introduced Continuous-Time Policy Gradients (CTPG), a new class of policy gradient estimator for continuous-time systems. By leveraging advanced integrators developed by the numerical ODE community, CTPG enjoys superior performance to the standard back-propagation through time (BPTT) estimator. Although we studied policy gradients in this work, it's worth noting that CTPG is general enough to operate as a "layer" within any larger differentiable model. This work could be extended to other related scenarios: systems with non-deterministic dynamics (using stochastic ODE solvers) infinite-time planning problems (using steady state ODE solvers) and sensor/actuation latency (using delay differential equations solvers). Adjoints for these alternatives have already been implemented in Rackauckas and Nie (2017). Another compelling direction for future work would be the incorporation of value function learning via extension to the Hamilton-Jacobi-Bellman equation. We are interested in the following questions: (1) When and why do policy gradient optimizations fail? (2) How can we properly incorporate policy gradient estimates into a global policy search?

Acknowledgments

References

Marcin Andrychowicz, Anton Raichuk, Piotr Stanczyk, Manu Orsini, Sertan Girgin, Raphaël Marinier, Léonard Hussenot, Matthieu Geist, Olivier Pietquin, Marcin Michalski, Sylvain Gelly, and Olivier Bachem. What matters in on-policy reinforcement learning? A large-scale empirical study. *CoRR*, abs/2006.05990, 2020.

Peter W. Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, and Koray Kavukcuoglu. Interaction networks for learning about objects, relations and physics. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 4502–4510, 2016.

Jan Carius, René Ranftl, Vladlen Koltun, and Marco Hutter. Trajectory optimization with implicit hard contacts. *IEEE Robotics Autom. Lett.*, 3(4):3316–3323, 2018.

Jan Carius, René Ranftl, Vladlen Koltun, and Marco Hutter. Trajectory optimization for legged robots with slipping motions. *IEEE Robotics Autom. Lett.*, 4(3):3013–3020, 2019.

Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David Duvenaud. Neural ordinary differential equations. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 6572–6583, 2018.

Filipe de Avila Belbute-Peres, Kevin A. Smith, Kelsey R. Allen, Josh Tenenbaum, and J. Zico Kolter. End-to-end differentiable physics for learning and control. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, pages 7178–7189, 2018.

Jianzhun Du, Joseph Futoma, and Finale Doshi-Velez. Model-based reinforcement learning for semi-markov decision processes with neural odes. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Taylor Kessler Faulkner, Elaine Schaertl Short, and Andrea Lockerd Thomaz. Policy shaping with supervisory attention driven exploration. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, pages 842–847. IEEE, 2018. doi: 10.1109/IROS.2018.8594312.

Amir Gholaminejad, Kurt Keutzer, and George Biros. ANODE: unconditionally accurate memory-efficient gradients for neural odes. In Sarit Kraus, editor, *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI 2019, Macao, China, August 10-16, 2019*, pages 730–736. ijcai.org, 2019. doi: 10.24963/ijcai.2019/103.

Ernst Hairer, Syvert P. Nørsett, and Gerhard Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. 1993. doi: 10.1007/978-3-540-78862-1.

Noriaki Hirose, Fei Xia, Roberto Martín-Martín, Amir Sadeghian, and Silvio Savarese. Deep visual mpc-policy learning for navigation. *IEEE Robotics Autom. Lett.*, 4(4):3184–3191, 2019. doi: 10.1109/LRA.2019.2925731.

Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Trans. Graph.*, 38(6):201:1–201:16, 2019. doi: 10.1145/3355089.3356506.

Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Wanxin Jin, Zhaoran Wang, Zhuoran Yang, and Shaoshuai Mou. Pontryagin differentiable programming: An end-to-end learning and control framework. 2020.

Taewan Kim, Chungkeun Lee, Hoseong Seo, Seungwon Choi, Wonchul Kim, and H. Jin Kim. Vision-based target tracking for a skid-steer vehicle using guided policy search with field-of-view constraint. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, pages 2418–2425. IEEE, 2018. doi: 10.1109/IROS.2018.8593843.

Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

Randall J. LeVeque. *Finite difference methods for ordinary and partial differential equations - steady-state and time-dependent problems*. SIAM, 2007. ISBN 978-0-89871-629-0. doi: 10.1137/1.9780898717839.

Yunzhu Li, Jiajun Wu, Jun-Yan Zhu, Joshua B. Tenenbaum, Antonio Torralba, and Russ Tedrake. Propagation networks for model-based control under partial observation. In *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*, pages 1205–1211. IEEE, 2019. doi: 10.1109/ICRA.2019.8793509.

Yunzhu Li, Hao He, Jiajun Wu, Dina Katabi, and Antonio Torralba. Learning compositional koopman operators for model-based control. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Michael McAsey, Libin Mou, and Weimin Han. Convergence of the forward-backward sweep method in optimal control. *Comput. Optim. Appl.*, 53(1):207–226, 2012. doi: 10.1007/s10589-011-9454-7.

Michael C. Mozer. A focused backpropagation algorithm for temporal pattern recognition. *Complex Syst.*, 3(4), 1989.

Romeo Orsolino, Michele Focchi, Carlos Mastalli, Hongkai Dai, Darwin G. Caldwell, and Claudio Semini. Application of wrench-based feasibility analysis to the online trajectory optimization of legged robots. *IEEE Robotics Autom. Lett.*, 3(4):3363–3370, 2018.

L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko. *The mathematical theory of optimal processes*. The Macmillan Co., New York, 1964. Translated by D. E. Brown, A Pergamon Press Book.

Yi-Ling Qiao, Junbang Liang, Vladlen Koltun, and Ming C. Lin. Scalable differentiable physics for learning and control. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 7847–7856. PMLR, 2020.

Alessio Quaglino, Marco Gallieri, Jonathan Masci, and Jan Koutník. SNODE: spectral discretization of neural odes for system identification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Christopher Rackauckas and Qing Nie. Differentialequations.jl–a performant and feature-rich ecosystem for solving differential equations in julia. *Journal of Open Research Software*, 5(1), 2017.

Anthony J. Robinson and F. Failside. Static and dynamic error propagation networks with application to speech coding. In Dana Z. Anderson, editor, *Neural Information Processing Systems, Denver, Colorado, USA, 1987*, pages 632–641. American Institue of Physics, 1987.

F. Sabatino. Quadrotor control: modeling, nonlinear control design, and simulation. 2015.

Connor Schenck and Dieter Fox. Reasoning about liquids via closed-loop simulation. In Nancy M. Amato, Siddhartha S. Srinivasa, Nora Ayanian, and Scott Kuindersma, editors, *Robotics: Science and Systems XIII, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA, July 12-16, 2017*, 2017.

Connor Schenck and Dieter Fox. Spnets: Differentiable fluid dynamics for deep neural networks. In *2nd Annual Conference on Robot Learning, CoRL 2018, Zürich, Switzerland, 29-31 October 2018, Proceedings*, volume 87 of *Proceedings of Machine Learning Research*, pages 317–335. PMLR, 2018.

L.F. Shampine and H.A. Watts. *DEPAC: Design of a User Oriented Package of ODE Solvers*. Sandia report. Sandia National Laboratories. UC 32, 1979.

Jason L. Speyer and David H. Jacobson. *Primer on Optimal Control Theory*. SIAM, 2010. ISBN 978-0-89871-694-8. doi: 10.1137/1.9780898718560.

Gao Tang, Weidong Sun, and Kris Hauser. Learning trajectories for real- time optimal control of quadrotors. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2018, Madrid, Spain, October 1-5, 2018*, pages 3620–3625. IEEE, 2018. doi: 10.1109/ IROS.2018.8593536.

Russ Tedrake. *Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation*. URL http://underactuated.mit.edu/.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2012, Vilamoura, Algarve, Portugal, October 7-12, 2012*, pages 5026–5033. IEEE, 2012. doi: 10.1109/IROS.2012.6386109.

Charalampos Tsitouras. Runge-kutta pairs of order 5(4) satisfying only the first column simplifying assumption. *Comput. Math. Appl.*, 62(2):770–775, 2011.

Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy P. Lillicrap, Nicolas Heess, and Yuval Tassa. dm_control: Software and tasks for continuous control. *Softw. Impacts*, 6:100022, 2020. doi: 10.1016/j.simpa.2020.100022.

Peng Wang, Yanzhao Cao, Xiaoying Han, and Peter Kloeden. Mean-square convergence of numerical methods for random ordinary differential equations. *Numer. Algorithms*, 87(1):299–333, 2021. doi: 10.1007/s11075-020-00967-w.

Andrew Zangwill. *Modern electrodynamics*. Cambridge University Press, Cambridge, 2013.