PAUL G. ALLEN SCHOOL
OF COMPUTER SCIENCE & ENGINEERING

Samuel Ainsworth, Kendall Lowrey, John Thickstun, Zaid Harchaoui, Siddhartha Srinivasa

Faster Policy Learning with Continuous-Time Gradients

SUMMARY

Who needs RL? We learn policies by differentiating directly through the physics engine. In addition, we offer significant performance improvements working in continuous time!

Research questions

Q1: Can we compute more accurate estimates of the policy gradient? Backprop. through time (BPTT) introduces numerical error by discretizing the robot trajectory under a policy. 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?

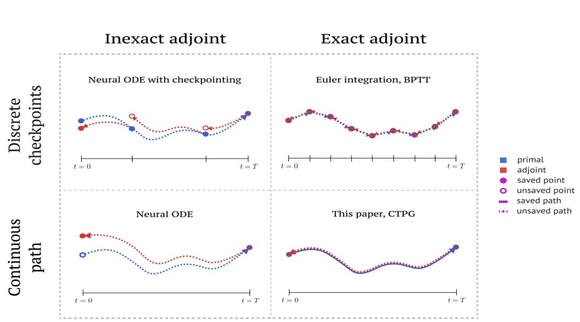
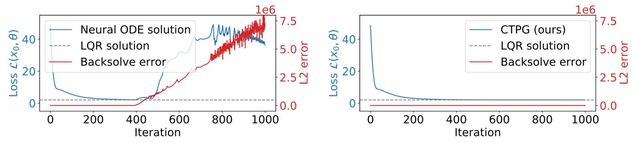


Figure 1: Comparing policy gradient estimators. Neural ODE (Sec. 3.3) approximates continuous trajectories, but accumulated error in its adjoint estimates. Neural ODE with checkpointing mitigates this error, but still accumulates error between checkpoints. Euler integration with BPTT (Sec. 3.2) stores 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.

Key idea: Rather than computing an exact gradient of a discretized system, we instead compute an approximate gradient of the continuous system.

Neural ODEs aren't sufficient

It turns out that Neural ODEs are not sufficient for control experiments. The backprop process in a Neural ODE leaves open the question: "Where'd ya come from? Where'd ya go?"



Since optimal controllers will generally navigate the system to a fixed point, Neural ODE's "backtracking" technique becomes unstable.

The CTPG algorithm

Instead, we propose maintaining a spline representation of the "forward" pass. This proves to be quite lightweight and enables us to learn rich policies. Furthermore, we show that BPTT is an instantiation of our algorithm with specific choices for the ODE solver and spline.

Algorithm 1 Continuous-Time Policy Gradients

Input: Differentiable physics simulator $f(x, u)$, cost/reward function $w(x, u)$, policy $\pi_\theta(x)$, initial state x_0, and a numerical solver Solve[initial_conditions, dynamics]

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:

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

Experiments

A differential drive "Roomba"-style robot:

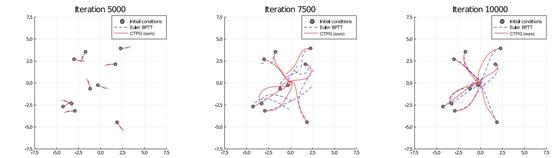


Figure 4: Stages of training policies to control a differential drive robot to stabilize towards the origin. Each curve denotes the trajectory of the robot in the (x, y) plane under the respective policies. A full video is available in the supplementary material.

Learning a policy with a physics engine implemented in DiffTaichi:

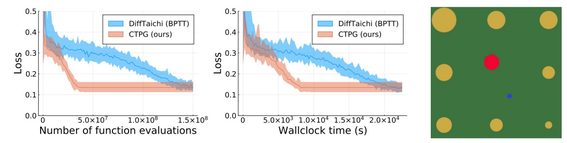


Figure 5: The DiffTaichi electric experiment involves modulating the charges of the 8 yellow electrodes to push the red charge to follow the blue dot (right). We found that CTPG performance was dominant in terms of the number of function evaluations (left) as well as wallclock time (center) across 32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

MuJoCo cartpole, using finite differences:

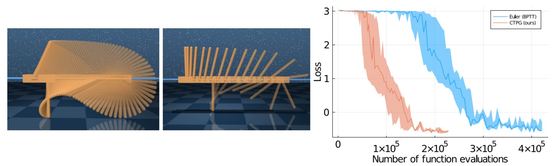


Figure 6: Visualization of all states visited by a policy trained with BPTT (left) and CTPG (middle). Fewer states are visited by CTPG during training leading to more efficient learning (right). Error bars on the training plot show the 0.25-0.75 quantile range of 5 random seeds.

Controlling a quadrotor:

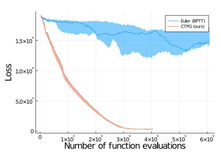


Figure 7: CTPG learns policies to stabilize the quadrotor towards the origin, while Euler integration and BPTT–with the same number of function evaluations–can not.