Leveraging Reward Gradients For Reinforcement Learning in Differentiable Physics Simulations

Sean Gillen, Katie Byl

Abstract—In recent years, fully differentiable rigid body physics simulators have been developed, which can be used to simulate a wide range of robotic systems. In the context of reinforcement learning for control, these simulators theoretically allow algorithms to be applied directly to analytic gradients of the reward function. However, to date, these gradients have proved extremely challenging to use, and are outclassed by algorithms using no gradient information at all. In this work we present a novel algorithm, cross entropy analytic policy gradients, that is able to leverage these gradients to outperform state of art deep reinforcement learning on a certain set of challenging nonlinear control problems.

I. INTRODUCTION

Computer simulation has become an indispensable tool for researchers and engineers of robotic systems for design, control, and verification. Recent advances in model free deep reinforcement learning (DRL) have been able to leverage simulation and the continued exponential increase computer resources to solve a variety of challenging control and perception problems. Examples include dexterously manipulating objects with a 24 DOF robotic hand [24], and recent work from the Robotic Systems Lab at ETH Zurich demonstrates rough terrain quadruped robot arguably on par with Boston Dynamics [22]. In both cases, training was done in simulation, and then successfully transferred to the real world.

These simulations have largely been treated as black boxes by DRL algorithms. This is both a strength and a weakness of DRL, and this stems partially from the fact that in commonly used simulators like MuJoco [32] or Bullet [5], are not differentiable (nor are physical robots, for that matter). That is, we are unable to compute the gradient of the state at time t+1, with the action from time t. Therefore we are also unable to take the gradients of the reward function with respect to controller parameters. Thus, RL must thus rely on various approximations of the true gradients, like finite differences or vanilla APG and state of the art DRL algorithms in at least one class of challenging nonlinear control problems.

However in recent years, fully differentiable physics simulators have started to emerge [15] [14] [9], which offer analytic gradients using automatic differentiation. These simulators already have a number of interesting applications. For example, it is possible to use data from a physical system as data for a learning algorithm to make simulation to better match. Furthermore the nature of some these simulators allow them to be run on hardware accelerators, which offers some obvious speed advantages for algorithms which can make use of them. And, most relevant to this work, they also provide analytic gradients for any differentiable function of simulation state variable. This means we can use stochastic gradient descent, which is the gold standard for training neural networks, directly using the negative sum of rewards as the loss function. We will call training policies in this way an analytic policy gradient algorithm.

However, in practice, these gradients have proven extremely challenging to use, for a number of reasons. Part of the problem is that to take the gradient through any iterated dynamical system required back propagation through time (BPTT). Long chains of BPTT have long been known to cause exploding or vanishing gradients, which naturally causes difficulty in learning [1]. A recent paper by Metz et al. [21] also offer some exposition on the challenges of using the analytic gradients offered by these new rigid body simulators. They highlight that in addition to problems inherent to BPTT, the naturally chaotic dynamics of many rigid body systems exasperate the problems with diverging gradients significantly.

Another difficulty are severe local maxima. Local maxima are a common problem in all of deep learning, but it is apparent that using APG for reinforcement learning with rigid body systems is especially prone to falling into these extrema. In [21] they also show the reward landscape of the Ant system in Brax. The Ant is a standard benchmark problem in the RL community, it consists of a quadruped robot who’s objective is to move forward as fast as possible. They show that reward landscape and reward gradient for this system has extremely high variance, and fraught with local extrema. One may expect that this is due primarily to the fact that the Ant system is relatively high dimensional and subject to contact and frictional forces with the ground. However as we show in figure [1] an Acrobat system, which is a simple two DOF system that is not subject to any contact forces, suffers from many of the same problems.

Given these difficulties, it remains an open question what role, if any, analytic policy gradients have to play in reinforcement learning for robotic control. In this paper we present a novel algorithm, Cross Entropy Analytic Policy Gradients (CE-APG), which we have found is able to outperform both vanilla APG and state of the art DRL algorithms in at least one class of challenging nonlinear control problems. Specifically, we demonstrate that under-actuated planar kinematic chains, like the Acrobat or inverted cart pole pendulum
Deterministic policy gradient (TD3) [12] [29] [10]. Proximal Policy Optimization (PPO), and Twin Delayed Deep Policy卷积是现代模型自由强化学习算法的典范。

这些方法通常比模型自由RL更有效率，但是这些方法通常会带来一些缺点。例如，算法我们目前应该能适配到高维任务。然而，我们的方法没有模型自由RL的偏差。例如，我们算法的求解是有限的。

B. Policy Search in Differentiable Simulations

许多的论文中介绍不同可微模拟器还包括一个基本的例子，使用了非线性梯度的直接优化。则可能的选项包括进化策略（ES） [28]，或增强随机搜索（ARS） [19]，也可以被考虑为模型自由强化学习算法。

我们的算法可以被看作是模型自由强化学习算法，同时具有很多他们的优点和缺点。例如，我们算法的求解是有限的。

In [26]，他们介绍了一个模拟器，并建议在算法中称为“策略增强”和他们增加一个模型自由RL算法，用非线性梯度控制一个完全可控的双摆。在[23]，作者在不同的政策优化中使用了不同的模拟器。他们没有直接使用非线性策略梯度，而是发展了一个间接的第二级优化。他们也证明他们的系统在不完全未被控制的系统中，效果都是有限的。我们算法在对角线稳定平面上表现得非常艰难，所以我们通常会考虑到传统的控制设计。

DRL是通常分为模型基和模型自由控制。基模型本的控制算法学习了一个模型的系统在下面控制，用算法做计划，经典动态规划是一个例子，如PILOCO [7]。这些方法在理论上非常有效，模型自由RL，但是这些算法通常有一个硬的尺度到更高维度的问题，而且可能导致在运行时间的昂贵重规划。

模型自由RL在另一方直接学习一个策略，以最大化一个奖励函数。这意味着解决一个更困难的优化问题，但是这是已经被用过的方法，能够达到我们讨论的在这个论文的结果。模型自由算法包括Soft Actor Critic (SAC), Proximal Policy Optimization (PPO), and Twin Delayed Deep Deterministic policy gradient (TD3) [12] [29] [10].

尽管他们不严格使用神经网络，非线性自由的方法包括进化策略（ES） [28]，或增强随机搜索（ARS） [19]，也可以被看作是模型自由强化学习算法。

我们的算法可以被看作是模型自由强化学习算法，同时具有很多他们的优点和缺点。例如，算法我们目前应该能适配到高维任务。然而，我们的方法没有模型自由RL的偏差。我们用一种不同的政策优化方法，将sim2real转移可被淘汰到适用于物理系统。我们注意到，成功的转移的政策从模拟直接到硬件已经被证明多次。

B. Policy Search in Differentiable Simulations

许多的论文中介绍不同可微模拟器还包括一个基本的例子，使用了非线性梯度的直接优化。则可能的选项包括进化策略（ES） [28]，或增强随机搜索（ARS） [19]，也可以被考虑为模型自由强化学习算法。

我们的算法可以被看作是模型自由强化学习算法，同时具有很多他们的优点和缺点。例如，算法我们目前应该能适配到高维任务。然而，我们的方法没有模型自由RL的偏差。例如，我们算法的求解是有限的。

In [26]，他们介绍了一个模拟器，并建议在算法中称为“策略增强”和他们增加一个模型自由RL算法，用非线性梯度控制一个完全可控的双摆。在[23]，作者在不同的政策优化中使用了不同的模拟器。他们没有直接使用非线性策略梯度，而是发展了一个间接的第二级优化。他们也证明他们的系统在不完全未被控制的系统中，效果都是有限的。我们算法在对角线稳定平面上表现得非常艰难，所以我们通常会考虑到传统的控制设计。

D. Back Propagation Through Time

在为了传导政策通过一个迭代系统，我们必须使用一个称为反向传播通过时间的算法。如我们已经提到，这种导致在爆炸或
vanishing gradients, especially in long chains of computation. The problem essentially is that the reward at time t depends not only on the state and action at time t-1, but on the state and action from t-2, t-3, back to the initial state, even if system itself is Markovian. For even modest length roll-outs this causes instability in the gradients that make learning with them challenging.

There are many other contexts that this problem arises, including in natural language processing (NLP). One of the tools used to combat this in the NLP community are specialized recurrent neural networks, which are specifically designed to stop gradients that pass through the network from diverging. In our case we use gated recurrent unit GRU [8] as our control policy, we outline this architecture with more detail in the methods section.

III. PROBLEM FORMULATION

A. Reinforcement Learning

In reinforcement learning, the goal is to train an agent, acting in an environment, to maximize a scalar reward function. The environment is a discrete time dynamical system described in an environment, to maximize a scalar reward function.

\[ R(t) \rightarrow R(\theta) \]

The controller is a function parameterized by a vector \( \theta \in \mathbb{R}^d \) that maps states to actions \( g: \mathbb{R}^n \times \mathbb{R}^d \rightarrow \mathbb{R}^m \) such that:

\[ a_t = g(s_t, \theta) \]

The goal is to maximize a scalar reward function \( r: \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R}^n \rightarrow \mathbb{R} \). We consider the finite time undiscounted reward case. The objective function then is:

\[ R(\theta) = \sum_{t=0}^{T} r(s_t, a_t, s_{t+1}) \]

B. Kinematic Chains and Simulation

We consider three systems, a classic cartpole pendulum, a double cartpole pendulum, and an Acrobot [31]. These are all under-actuated, kinematic, chains, and are often used as benchmark problems for both reinforcement learning and nonlinear control. Their dynamics in general can be described with the following:

\[ M(q)\ddot{q} + C(q, \dot{q})\dot{q} + T_g(q)/g = Bu \]

with \( M \) being the inertia matrix, \( C \) being the Coriolis matrix, \( T \) being a matrix capturing gravitational affects. For these systems \( u \) is the torque outputted at each motor, and \( q \) is the vector of state variables. In each of these systems, there is a single unstable equilibrium point. Our goal is to swing the system from an initial condition to it’s unstable point and the maintain balance there. More details on formulating these environments into an MDP can be found in the appendix.

It was demonstrated in [11] that most DRL struggles with the full version of the Acrobot in particular. It is worth elaborating on that point. Many benchmarking suites (for example, OpenAI’s gym [4]) have underactuated systems like the acrobot or cartpole pendulum, however the tasks are typically either to swing the system up, or to balance it, asking one controller to do both becomes a much more challenging problem.

C. The Brax Simulator

The above systems are simulated using Brax [9]. Brax is a differentiable physics engine that can simulate systems made up of rigid bodies, joint constraints, and actuators. The simulations primary advantage is that it can run massively parallel simulations very quickly on accelerator hardware, i.e. TPUs and GPUs. By virtue of being written entirely in Jax [3], we can also take arbitrary gradients through the simulator using autodiff.

IV. METHODS

A. Analytic policy gradients

Stochastic gradient descent and its variants (in our case adam [17]) are the gold standard for training deep neural networks. We seek to train our policy using the gradient of the sum of the reward function for a given episode.

\[ \theta^+ = \theta + \alpha \nabla_{\theta} R(\theta) \]

To be more specific, we perform N policy rollouts using the current parameters, take the gradient of the mean of the sum of rewards for each these roll outs, use that gradient to update the current policy, and repeat until convergence. Thus our update step is:

\[ \theta^+ = \theta + \alpha \nabla_{\theta} \frac{1}{N} \sum_{i=0}^{N} \sum_{t=0}^{T} r(s_t, a_t, s_{t+1}) \]

As we’ve discussed, because we are using a differentiable simulation, the analytic gradient with respect to \( \theta \) is available to us.

B. The Cross Entropy Method

The Cross Entropy Method (CEM [27]) is a well established algorithm for importance sampling and optimization. CEM maintains a probability distribution over its decision variables, in this case the decision variables are the parameters for our policy. The most common formulation is to use a normal Distribution, thus we must maintain a vector of means \( \mu_j \), and a covariance matrix \( \sigma_j \). \( \mathcal{N}(\mu_j, \sigma_j) \). At each step we sample candidate policies from this distribution, and use the following update rules:

\[ \mu_j^+ = \frac{1}{K_e} \sum_{i=0}^{K_e} \mu_j \]

\[ \sigma_j^+ = \frac{1}{K_e} \sum_{i=0}^{K_e} \sigma_j \]
$\sigma^2_\pi = \frac{1}{K_e} \sum_{i=0}^{K_e} (\mu_\pi - \mu_i) (\mu_\pi - \mu_i)^T$ (8)

However, the covariance matrix grows quadratically with the number of policy parameters, and neural networks can have thousands of parameters even for small systems. Thus, we make the following simplification to the variance:

$\sigma^2_\pi = \frac{1}{K_e} \sum_{i=0}^{K_e} (\mu_\pi - \mu_i)^2$ (9)

This implicitly ensures that our covariance "matrix" only has entries on the diagonal, and can thus be stored as a vector. This simplification is also made by [23].

C. Cross Entropy Analytic Policy Gradients

**Algorithm 1** Cross Entropy Analytic Policy Gradients

**Require:** Policy $\pi$ with trainable parameters $\theta$

**Require:** Hyper-parameters $- \sigma_0$, $K_a$, $K_e$

Sample $\theta_e = [\theta_1, ..., \theta_n]$ from $\mathcal{N}(\theta, \sigma_0^2)$

**for** $\theta_i$ in $\theta_e$ **do**

Run APG with initial policy weights $\theta_i$.

Collect sum of rewards $R_i$ and final policy $\theta_i^*$.

**end for**

Sort $\theta^*$ values in descending order according to reward

$\theta^* = \frac{1}{K_e} \sum_{i=0}^{K_e} \theta_i^*$

$\sigma^* = \sqrt{\frac{1}{K_e} \sum_{i=0}^{K_e} (\theta - \theta_i^*)^2}$

We combine these two algorithms as follows. Start with initial policy weight $\theta$, and an initial parameter variance $\sigma_0$. We then generate $K_a$ candidate policies by sampling from $\mathcal{N}(\theta, \sigma_0)$. Using these policies as initial conditions, we run $K_a$ analytic policy gradient algorithms in parallel, which gives us new weight vector $[\theta_0, \theta_1...\theta_{K_a}]$, and the final rewards for these new policy weights, $R_1, R_2...R_{K_a}$. We then sort the policy weights in descending order based on their associated final return. Finally we select the top $K_e$ and use equations [7] and [9] to update our parameter and variance vector. This is repeated until some stopping criteria, for this paper we simply train for fixed number of steps.

D. Controller Architecture

As we have already mentioned, we employ a Gated Recurrent Unit (GRU) network as our control policy to help combat the exploding / vanishing gradient problem. For our CE-APG experiments, we used a GRU with two fully connected layers on the output, with ReLU hidden activations and a Tanh non-linearity on the final layer. Network sizes for each experiment can be found in the appendix. We found that by using the GRU we are able to train with episodes lengths of at least 500 steps.

In addition to this, we use deterministic policies, rather than the stochastic ones usually associated with deep reinforcement learning. Typically in DRL, the policy actually parameters a probability distribution over possible actions. At every time step, one generates a new distribution based on the current state, and then samples from that distribution to select an action. While our algorithm is compatible with stochastic policies of this nature, we instead compute the action directly. We believe this is especially advantageous for systems with unstable and highly sensitive dynamics (like the acrobot, cartpole etc).

V. RESULTS

For each environment, we ran trials with 8 random seeds. The seeds affect all the sources of randomness during training, of which there are several. The initial value of the policy parameters, the noise added to the policy at the beginning of each iteration, and the initial condition of the simulator at the beginning of each episode. Figure 2. For each of these trials we used the GRU controller architecture discussed above, details on layer sizes etc. can be found in the appendix. We report the resulting rewards obtained at the end of in table I. In addition to our own algorithm, we also run comparisons from PPO, SAC, and Brax’s implementation of APG (which has some note-able difference from our own, discussed in the appendix).

| Environment | Pendulum | Acrobat | Double Pendulum |
|-------------|----------|---------|-----------------|
| CE-APG (ours) | -355 ± 10 | 3053 ± 458 | 4295 ± 75 |
| PPO         | -464 ± 223 | 405 ± 927 | 4249 ± 549 |
| SAC         | -2274 ± 1000 | 359 ± 159 | -3.9e5 ± 4.9e6 |
| Brax Apg    | -3485 ± 819 | 1949 ± 814 | -1982 ± 4056 |

**TABLE I:** Results of the training on our test environments, we report the mean and standard deviation of rewards obtained from training each algorithm with 8 random seeds.

We can see that across all three environments, our method outperforms the other benchmark algorithms. On the double inverted pendulum we get the same final reward as PPO, exceed it slightly on the cartpole, and get significantly higher reward on the acrobot. In fact for the cart-pole in particular our algorithm significantly outperforms the others. It is worth putting this in context, as it is difficult to understand what a reward of 2000 vs. 3000 really means.

To do this we perform roll-outs with the best performing seeds for both CE-APG and the best performing benchmark, which was Brax’s APG implementation. We perform 10 rollouts with these top performers. For completeness we report the mean and standard deviation of the resulting rewards, CE-APG: 3507 ± 5, Brax’s APG: 2778 ± 89. However it is much more revealing to visualize one of these roll outs, which we do in figure 5.
a policy that does stabilize our system. Unsurprisingly given the rewards from table I, none of the other systems manage to balance the acrobat either. Of course such stabilization is exactly what we as humans had in mind when defining the environment.

In both cases we still used the GRU controller architecture as before. In the case of CEM we used 2000 iterations for every environment, which we found to be well past the point of convergence. For APG we used 200*100 iterations, which results in the same number of environment interactions to CE-APG. The results are shown in table II.

Table II: Results of our ablation studies. We report the mean and standard deviation of rewards obtained from training each algorithm with 8 random seeds

| Environment   | Pendulum | Acrobot | Double Pendulum |
|---------------|----------|---------|-----------------|
| CE-APG (ours) | -355 ± 10 | 3053 ± 458 | 4295 ± 75 |
| PAPG          | -1034 ± 208 | 2456 ± 647 | 2335 ± 431 |
| CEM           | -3019 ± 559 | 626 ± 160 | -2.7e5 ± 7.6e5 |

As we can see, the performance of either method individually is generally poor, though we do see that PAPG does about as well as the best agent from the APG results reported in table I, however when we perform rollouts the resulting policies exhibit the same behavior shown in figure 3, that is they continuously spin, spending as much time as they can near the goal state, but never settling there.

VI. ABLATION STUDIES

In addition to the comparisons done above, we also conducted several ablation studies, this isolates the effect of our implementation of APG from the results presented above. We conducted experiments using only the gradient free CEM method, with no analytic policy gradients being used. And also compared to our implementation of APG, which we are calling PAPG for parallel APG. This essentially means that we run APG N times in parallel, and picking the best result to return.

As we can see, the performance of either method individually is generally poor, though we do see that PAPG does about as well as the best agent from the APG results reported in table I, however when we perform rollouts the resulting policies exhibit the same behavior shown in figure 3, that is they continuously spin, spending as much time as they can near the goal state, but never settling there.

VII. DISCUSSION AND FUTURE WORK

We have shown that analytic policy gradients can be leveraged effectively for at least one class of system, nonlinear underacted systems with unstable target states. This is a limited scope of problems, but it is interesting because other modern DRL algorithms struggle with such systems. However, we are as of yet unable to get our algorithm to perform well in contact rich environments, likely because the contacts introduce huge variance into the gradients. As of writing, there is active work in the community to make Brax friendlier to analytic gradient based algorithms, in particular adding soft contacts, which may very well help a lot.

In addition, the sample complexity of our algorithm is comparable to other on policy DRL algorithm, and is behind what we might expect from off policy algorithms. However there are many algorithmic improvements could be made, for example importance sampling has been found to improve the sample efficiency of CEM by up to a factor of 10. We are
currently unable to effectively implement this due technical limitations in the way we implement parallelization.

VIII. Conclusion

In conclusion, we presented Cross Entropy Analytic Policy Gradients, an algorithm that can exploit analytic gradients. We covered some relevant background, introduced our method, and placed it in the broader context. We then presented our algorithm and the environments we used for testing. We presented results of our algorithm compared to state of the art baselines, and performed ablation studies. We then contextualized the rewards obtained on Acrobat in particular. This demonstrated that our algorithm was able to successfully stabilize the system, whereas the baseline algorithms where not. We think this algorithm shows that analytic policy gradients can be leveraged productively in at least context.

References

[1] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994. doi: 10.1109/72.279181.

[2] Homanga Bharadhwaj, Kevin Xie, and Florian Shkurti. Model-predictive control via cross-entropy and gradient-based optimization, 2020.

[3] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax

[4] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[5] Erwin Coumans and Yunfei Bai. Pybullet, a python module for physics simulation for games, robotics and machine learning. http://pybullet.org 2016–2020.

[6] Jonas Degrave, Michiel Hermans, Joni Dambre, and Francis wyffels. A differentiable physics engine for deep learning in robotics. *Frontiers in Neurorobotics*, 13, 2019. ISSN 1662-5218. doi: 10.3389/fnbot.2019.00006. URL https://www.frontiersin.org/article/10.3389/fnbot.2019.00006

[7] Marc Deisenroth and Carl E Rasmussen. Pilco: A model-based and data-efficient approach to policy search. In *Proceedings of the 28th International Conference on machine learning (ICML-11)*, pages 465–472. Citeseer, 2011.

[8] Rahul Dey and Fathi M. Salem. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1597–1600, 2017. doi: 10.1109/MWSCAS.2017.8053243.

[9] C. Daniel Freeman, Erik Frey, Anton Raichuk, Sertan Girgin, Igor Mordatch, and Olivier Bachem. Brax - a differentiable physics engine for large scale rigid body simulation, 2021. URL http://github.com/google/brax.

[10] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018.

[11] S. Gillen, Marco Molnar, and K. Byl. Combining deep reinforcement learning and local control for the acrobot swing-up and balance task. 2020 59th IEEE Conference on Decision and Control (CDC), pages 4129–4134, 2020.

[12] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[13] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin Riedmiller, and David Silver. Emergence of locomotion Behaviours in Rich Environments. *arXiv:1707.02286 [cs]*, July 2017. URL http://arxiv.org/abs/1707.02286 arXiv: 1707.02286.

[14] Eric Heiden, David Millard, Erwin Coumans, Yizhou Sheng, and Gaurav S Sukhatme. NeuralSim: Augmenting differentiable simulators with neural networks. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2021. URL https://github.com/google-research/tiny-differentiable-simulator

[15] Yuanning Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédou Durand. Diftaiichi: Differentiable programming for physical simulation, 2020.

[16] Kevin Huang, Sahin Lale, Ugo Rosolia, Yuanyuan Shi, and Anima Anandkumar. Cem-gd: Cross-entropy method with gradient descent planner for model-based reinforcement learning, 2021.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[18] Joonho Lee, Jemin Hwangbo, and Marco Hutter. Robust Recovery Controller for a Quadrupedal Robot using Deep Reinforcement Learning. *arXiv:1901.07517 [cs]*, January 2019. URL http://arxiv.org/abs/1901.07517 arXiv: 1901.07517.

[19] Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search of static linear policies is competitive for reinforcement learning. *Advances in Neural Information Processing Systems*, 2018-December(NeurIPS): 1800–1809, 2018. ISSN 10495258.

[20] Steven L. McCarty, Laura M. Burke, and Melissa McGuire. Parallel Monotonic Basin Hopping for Low Thrust Trajectory Optimization. doi: 10.2514/6. 2018-1452. URL https://arc.aiaa.org/doi/abs/10.2514/6. 2018-1452

[21] Luke Metz, C Daniel Freeman, Samuel S Schoenholz, and Tal Kachman. Gradients are not all you need. *arXiv preprint arXiv:2111.05803*, 2021.

[22] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz
IX. APPENDIX

A. Implementation Details

There are some notable differences between our implementation of APG (which we call PAGP for parallel apg) and the implementation of APG provided Brax. First, we use different controller architectures, our method uses a deterministic GRU, and theirs uses a stochastic multi layer perceptron. Furthermore the parallelization characteristics are quite different, the Brax implementation of APG was designed for use with a TPU, and thus performs hundreds or thousands of rollouts in parallel for every update step. We note that we found the performance of CE-APG to be significantly faster on CPU compared to GPU/TPU.

B. Network Architecture

Each policy consists of GRU cell with two fully connected layers on the output. The fully connection network has ReLU hidden activations and a tanh non-linearity on the final layer.

C. Hyper Paramaters

In all cases we selected the best hyper parameters we could find using a manual search, using a coarse parameter sweep as a starting point.

| CE-APG Hyperparameter                  |       |
|---------------------------------------|-------|
| APG Epochs                            | 100   |
| Total Epochs                          | 200   |
| Total Env Interactions                | 1e7   |
| initial std                           | 0.05  |
| learning rate                         | 1e-3  → 1e-6 |
| batch size (N)                        | 4     |
| $K_a$                                 | 24    |
| $K_c$                                 | 8     |

| PPO Hyperparameter                   |       |
|--------------------------------------|-------|
| Total Timesteps                      | 8e7   |
| Minibatch Size                       | 32    |
| Batch Size                           | 256   |
| Unroll Length                        | 50    |
| N Update Epochs                      | 8     |
| Discounting                          | 0.99  |
| Learning Rate                        | 3e-4  |
| Entropy Cost                         | 1e-3  |
| N Envs                               | 512   |
| SAC Hyperparameter       | Value |
|--------------------------|-------|
| Total Timesteps          | 2e6   |
| Discounting              | .95   |
| Learning Rate            | 1e-3  |
| N Envs                   | 64    |
| Batch Size               | 128   |

| Brax-APG Hyperparameter  | Value |
|--------------------------|-------|
| Total Env interactions   | 1e7   |
| N Environments           | 24    |
| Learning rate            | 5e-4  |

X. Environment Details

A. Acrobat

The Acrobat is kept relatively simple, the only state variables are the joint angles and velocities, the reward is just the negative squared distance between the goal state and the current state:

$$r_a = -\phi_1^2 - \phi_2^2$$

With $\phi_1 = \phi_2 = 0$ being the upright balanced position with both links straight up.

B. Cartpole Pendulum

The cart-pole pendulum is also kept simple, the states are simply the joint angles and velocities, and the reward function is simply the negative square of the pendulum angle, with $\phi_1 = 0$ corresponding the the upright position.

C. Inverted Double Cartpole Pendulum

The double cartpole pendulum is modified from Brax’s existing benchmark environments, the only difference is that the initial condition is rotated 180 degrees from the upright. These environments use some reward shaping, adding an an alive bonus as well as some feature extraction. rather than directly feeding in joint angles, both the reward and state variables are fed in as the x,y coordinate for the end of each link. The reward function for the environment is:

$$r_{dp} = r_{alive} - r_{distance} - r_{velocity}$$  \hspace{1cm} (10)

Where

$$r_{alive} = 10$$ \hspace{1cm} (11)

$$r_{distance} = 0.05x^2 + (y - y_{des})^2$$ \hspace{1cm} (12)

$$r_{velocity} = \dot{\phi}_1 + \dot{\phi}_2$$ \hspace{1cm} (13)

and $y_{des}$ co-responds to the height of the second link when in the upright position. Put another way, the reward function is an alive bonus minus the euclidean distance between the end of the second link and the goal state.