Reproducibility and Analysis of Deep Policy Gradient methods for Reinforcement Learning Tasks

Shubham Chopra(260903254)   Nishant Mishra(260903177)

Introduction

In the recent years, significant work has been done in the field of Deep Reinforcement Learning, to solve challenging problems in many diverse domains. One such example, are Policy gradient algorithms, which are ubiquitous in state-of-the-art continuous control tasks [12]. Policy gradient methods can be generally divided into two groups: off-policy gradient methods, such as Deep Deterministic Policy Gradients (DDPG) [1], Twin Delayed Deep Deterministic (TD3), Soft Actor Critic (SAC) [11] and on-policy methods, such as Trust Region Policy Optimization (TRPO) [2].
However, despite these successes on paper, reproducing deep RL results is rarely straightforward [5]. There are many sources of possible instability and variance including extrinsic factors (such as hyper-parameters, noise-functions used) or intrinsic factors (such as random seeds, environment properties).

In this project, we perform two different analysis on these policy gradient methods:
(i) Reproduction and Comparison: We implement a variant of DDPG, based on the original paper [1]. We then attempt to reproduce the results of DDPG (our implementation) and TD3 [3] and compare them with the well-established methods of REINFORCE and A2C.
(ii) Hyper-Parameter Tuning: We also, study the effect of various Hyper-Parameters(namely Network Size, Batch Sizes) on the performance of these methods.

Prerequisites

The Policy Gradient Theorem[11] states that: The derivative of the expected reward is the expectation of the product of the reward and gradient of the log of the policy π_θ. Generally, Policy Gradient methods optimize:-

$$\rho(\theta, s_0) = E_{\pi_\theta}[\sum_{t=0}^{\infty} \gamma^t r(s_t|s_0)]$$

Here s_t represents a trajectory, while $r(s_t)$ represents the reward obtained from that trajectory. Using the policy gradient theorem:-

$$\frac{\delta \rho(\theta, s_0)}{\delta \theta} = \sum_s \mu_{\pi_\theta}(s|s_0) \sum_a \frac{\delta \pi_\theta(a|s)}{\delta \theta} Q_{\pi_\theta}(s, a).$$

REINFORCE uses Monte-Carlo updates for the Policy parameter updates. Hence, the reward for the trajectory, $r(s_t)$ is replaced with $G(t)$, the return.

Actor Critic methods use neural networks for both Actor and Critic. The "Critic" estimates the value function. The "Actor" updates the policy distribution in the direction suggested by the Critic (such as with policy gradients).

i. Deep Deterministic Policy Gradients (DDPG)

DDPG uses actor-critic methods which estimate Q(s, a) and optimize a policy that maximizes the Q-function based on Monte-Carlo updates. DDPG does this using deterministic policies, and is essentially an extension of DPG[10] with the addition of Neural Networks. Here, the Q-function, describes the expected return after taking an action a_t in state s_t and following policy π being maximized. For deterministic policies, expectation depends only on the environment. Using this and the Bellman equations, we get:-

$$Q^{\mu}(s_t, a_t) = E_{r_t, s_{t+1} \sim E}[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))].$$

This update is applied to Q-Learning[11], which uses the Greedy Policy $\mu(s) = argmax_a Q(s, a)$. Given function approximators parameterized by θ^Q, the loss by being minimized is given as:-

$$L(\theta^Q) = E_{r_t \sim E}[(Q(s_t, a_t|\theta^Q) - y_t)^2]$$

Where,

$$y_t = r(s_t, a_t) + \gamma Q(s_{t+1}, \mu(s_{t+1})|\theta^Q).$$

ii. Twin Delayed Deep Deterministic Policy Gradients (TD3)

TD3 maintains a pair of critics along with a single actor, at an attempt to reduce function approximation error. For each time step, the pair of critics are updated towards the minimum target value of actions selected by the target policy:

$$y_t = r(s_t, a_t) + \gamma E_{\epsilon}[Q_{\theta'}(s', \pi_{\phi'}(s') + \epsilon)]$$

$$y_t = r(s_t, a_t) + \gamma \; min_{i=1,2} \; (Q_{\theta'}(s', \pi_{\phi'}(s') + \epsilon)).$$

Here, ϵ denotes noise, sampled as a Gaussian distribution with mean 0, and variance of σ.

$$\epsilon \sim clip(N(0, \sigma) - c, -c)$$

Experiments and Analysis

We evaluate the DDPG, TD3 and A2C algorithms on continuous control environments from the OpenAI Gym benchmark [7], using the MuJoCo physics simulator [8]. We demonstrate the results on four environments: Hopper (S $\subseteq$ R20, A $\subseteq$ R3), Inverted Pendulum (S $\subseteq$ R20, A $\subseteq$ R2), Half-Cheetah (S $\subseteq$ R20, A $\subseteq$ R6) and Pendulum-v0.

(i) Hyper-Parameter Analysis

We run these experiments for 5000 episodes, with 200 steps i.e., 1e6 total steps and average all results across 5 runs. The hyper-parameters that we investigate are: policy network architecture and batch size.
The default hyper-parameters used are: a Network architecture of (100,50,25) with ReLU activations for a Gaussian Multilayer Perception Policy [8], actor-critic learning rates of 1e3 and 1e4, a step size of 0.01, and a reward scale of 0.1.

a) Network Architecture

The Policy network architecture can provide extra information storage. Thus, it plays an important role in the maximum reward achieved by the algorithm. We investigate three multilayer perceptron (MLP) architectures commonly seen in the literature: (64, 64), (100, 50, 25), and (400, 300).
The results can be seen in Figures 1 and 2. Figures 1a and 2a show that varying Network Sizes can significantly change the performance of both algorithms, on the Half-Cheetah environment. However, on the Hopper environment, there is no clear winner. The Network Size of (400, 300) significantly outperforms both (64, 64), (100, 50, 25) for Half-Cheetah, and hence will be used for Comparison Studies.

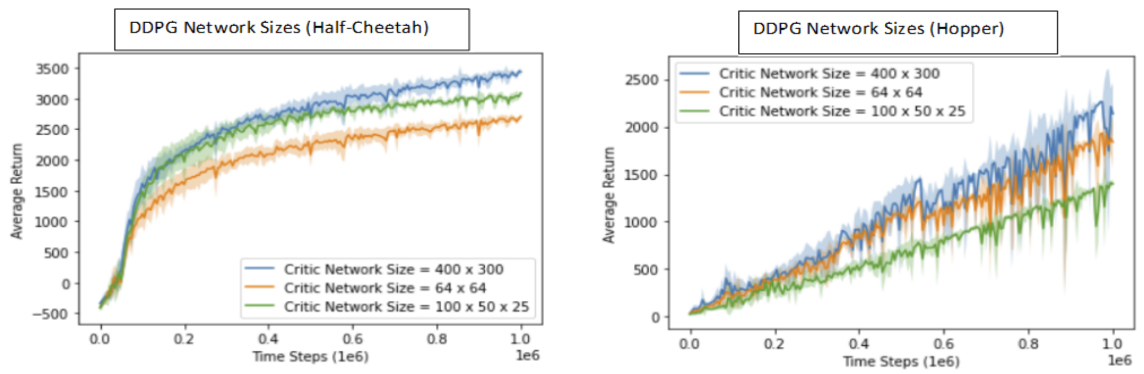

Figure 1: DDPG for different Network Sizes for Half-Cheetah and Hopper

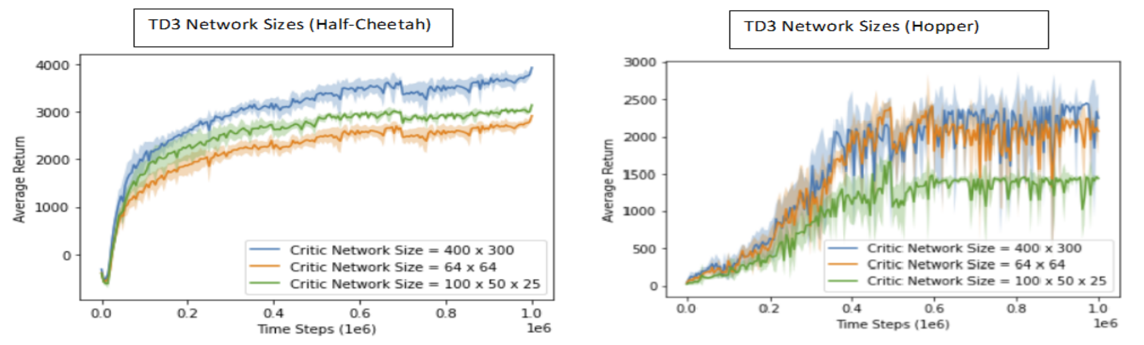

Figure 2: TD3 for different Network Sizes for Half-Cheetah and Hopper

b) Batch Sizes

To make efficient use of hardware optimizations, it is essential to learn in mini-batches, rather than online. The actor and critic updates are made by sampling a mini-batch uniformly from the replay buffer. Hence, batch sizes play an important role in the performance of these algorithms. We investigate three batch sizes: 128, 64 and 32.
The results in Figures 3 and 4 show that significant improvements can be made by increasing batch size to 128. However, there isn't much difference in 32 and 64 batch sizes for both algorithms and environments. Hence, we use a batch size of 128 in Comparison Studies.

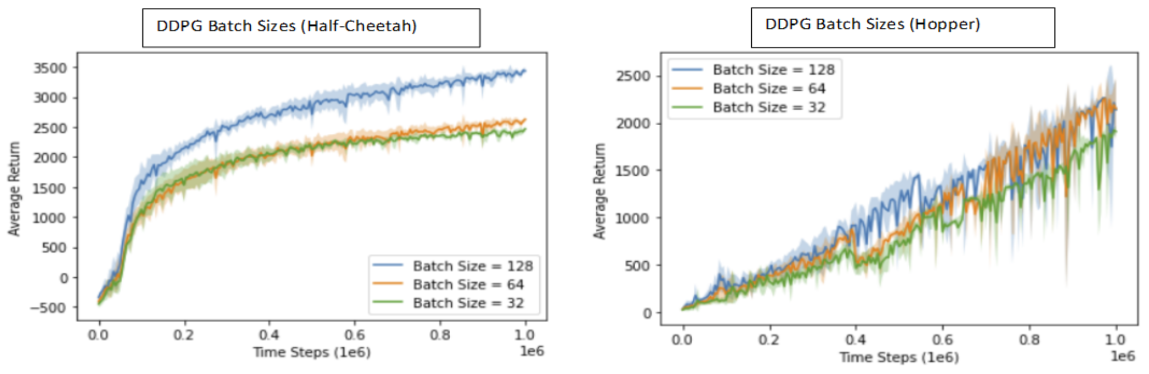

Figure 3: DDPG for different Batch Sizes for Half-Cheetah and Hopper

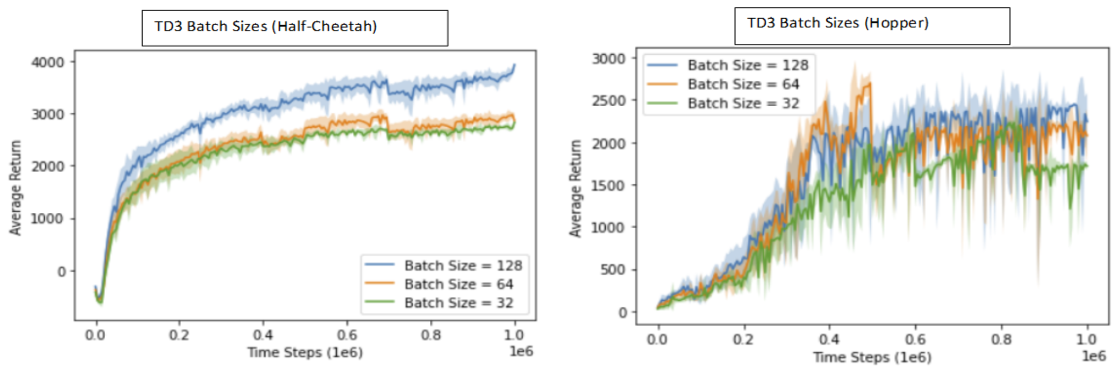

Figure 4: TD3 for different Batch Sizes for Half-Cheetah and Hopper

(ii) Comparison of Policy Gradient Algorithms

We compare the performance of DDPG, TD3 and A2C on the above mentioned OpenAI Gym environments. We use Average Rewards as the metric for testing performance, as it was used in the initial paper, and is common in the literature. As is visible from Figure 7 (Appendix), A2C severely out-performs REINFORCE in the CartPole environment, as it converges much faster.
As can be seen in Figure 5, DDPG and TD3 perform equally as well, and outperform A2C in the Pendulum environment. In the Inverted Pendulum environment, DDPG performs the best, with TD3 in second place, as there seems to be a lot of variance in the average

returns. In the environment Hopper, the TD3 significantly outperforms DDPG and A2C. DDPG seems to be getting better, however, the convergence rate of TD3 is much faster. In Half-Cheetah, the policy gradient methods DDPG and TD3 perform similarly, whereas A2C performs the worst. A2C performs the worst in all environments compared to DDPG and TD3, which is expected.

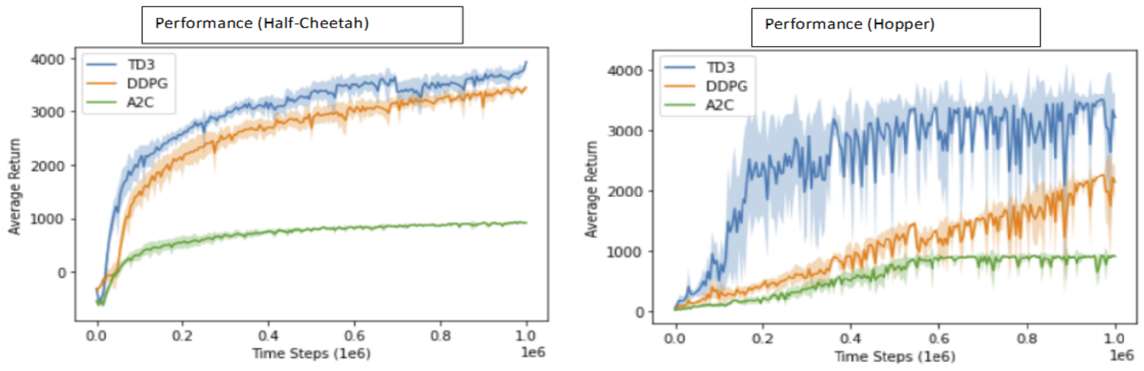

Figure 5: Comparison of DDPG, TD3 and A2C for Half-Cheetah and Hopper

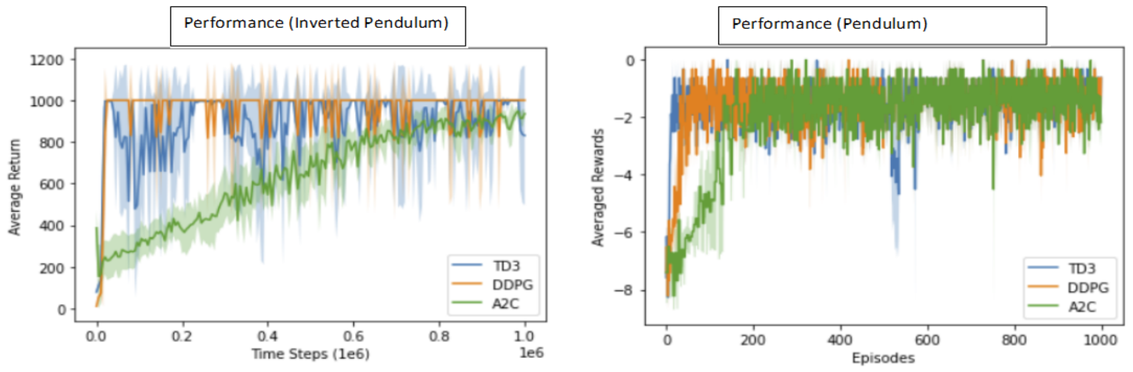

Figure 6: Comparison of DDPG, TD3 and A2C for Pendulum and Inverted Pendulum

Discussion and Conclusions

Our analysis shows that Half-Cheetah is more susceptible to performance variations from hyper-parameter tuning, while Hopper is not. This could be due to differences in stability, dynamics in the environments themselves. This result is consistent with other similar studies ([5], [9]). The network size of (400, 300) and the batch size of 128, gives the best results by far for Half-Cheetah, whereas there is no clear winner for Hopper.

From the comparison studies, it can be concluded that both DDPG and TD3 outperform A2C in all the four environments/tasks. TD3 performs overall better than DDPG, however, in most cases it is very subtle, with the exception of Hopper. DDPG is more stable than TD3 in Inverted Pendulum. Therefore, TD3 and DDPG are almost equally matched (with TD3 being better in some scenarios), and both outperform A2C in all environments.

Statement of Contribution

This project was completed with equal division of efforts and contribution of both the group members involved.

References

(1) Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa,David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. arXivpreprint arXiv:1509.02971, 2015.

(2) John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust regionpolicy optimization. In Proceedings of the 32nd International Conference on Machine Learning(ICML-15), pages 1889–1897, 2015.

(3) Scott Fujimoto, Herke van Hoof, David Meger. Addressing Function Approximation Error in Actor-Critic Methods. In Proceedings of the 35th International Conference on Machine Learning(ICML-18), pages 1889–1897, 2018.

(4) Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G.Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Pe-tersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, DaanWierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcementlearning. Nature, 518(7540):529–533, 02 2015.

(5) Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, David Meger. Deep Reinforcement Learning that Matters. In Proceedings of Association for the Advancement of Artificial Intelligence, 2018.

(6) Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deepvisuomotor policies. CoRR, abs/1504.00702, 2015.

(7) Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang,and Wojciech Zaremba. Openai gym. CoRR, abs/1606.01540, 2016.

(8) Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-basedcontrol. In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS2012, Vilamoura, Algarve, Portugal, October 7-12, 2012, pages 5026–5033, 2012.

(9) Riashat Islam, Peter Henderson, Maziar Gomrokchi, Doina Precup. Reproducibility of Benchmarked Deep ReinforcementLearning Tasks for Continuous Control.

(10) Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E Turner, and SergeyLevine. Q-prop: Sample-efficient policy gradient with an off-policy critic. arXiv preprintarXiv:1611.02247, 2016.

(11) Sutton, R. S.; Barto, A. Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 2018.

(12) Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In Proceedings of Association for the Advancement of Artificial Intelligence, 2018.

Appendix

REINFORCE algorithm is almost always outperformed by Actor-Critic methods such as A2C. This can be seen as an example on OpenAI Gym[7] environment CartPole-v0 in Figure 7. Hence, for the comparison with DDPG and TD3, we ignore REINFORCE and use A2C as a baseline.

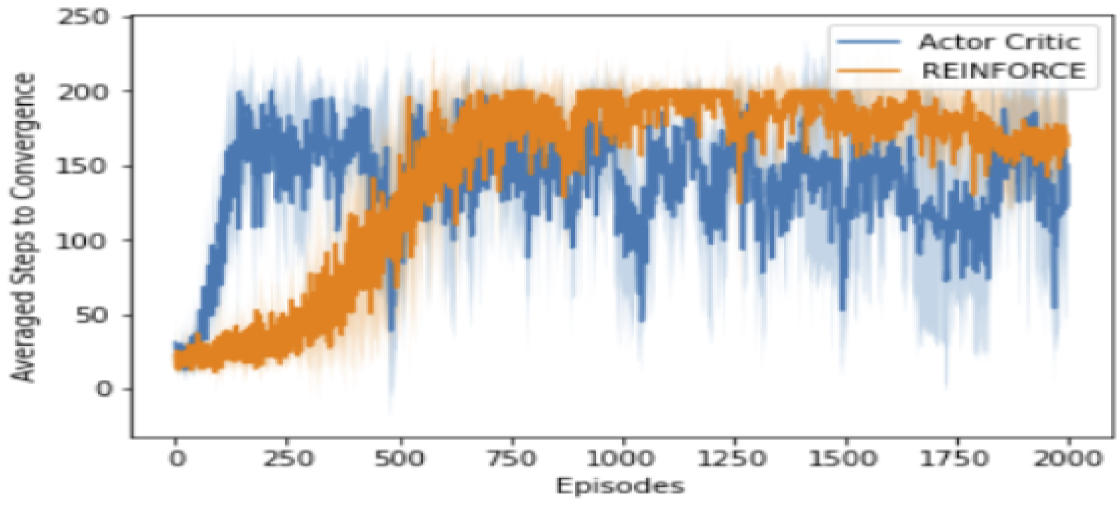

Figure 7: Average Steps to Convergence of REINFORCE vs A2C for CartPole