Deep Reinforcement Learning for Classic Control Tasks

Andrew Zhang
Department of Computer Science
Stanford University
andrewdu@stanford.edu

Abstract

Deep reinforcement learning has been shown to be very effective on well-defined environments such as standard Atari games [1]. However, environments such as OpenAI Gym's classic control tasks are less explored. This study aims to present and compare results on more abstract and robotic environments - Acrobot-v1 and CartPole-v0 from OpenAI Gym - using deep reinforcement learning implementations of Monte-Carlo vanilla policy gradients (VPG) and proximal policy optimization (PPO). Although both algorithms had varying results across over one thousand episode rollouts over multiple runs, VPG and PPO generally achieved optimal reward thresholds on both environments. However, while VPG and PPO, on averge, performed roughly equivalently on Acrobot-v1, PPO out-performed VPG on CartPole-v0 by a decent margin in both mean reward as well as learning speed every time.

1 Introduction

The goal of deep reinforcement learning is to combine standard reinforcement learning with neural networks for improved performance. For familiar environments/tasks that have a well-defined reward function such as Pong or Pac-Man (i.e. every dot collected gives one point), the problem is essentially a solved one, wherein either an off-policy or on-policy method can be used to easily converge to the optimal behaviour. However, for tasks that are seemingly more simple and abstract such as classic control tasks, regular off-policy methods do not perform as well as on-policy methods due to increased complexity of the problem and the more sparse nature of the reward. Motivated by this fact, the objective of our study is to experiment with and demonstrate results exhibited by Monte-Carlo vanilla policy gradients (VPG) and proximal policy optimization (PPO), two on-policy reinforcement learning methods, on the classic control tasks of Acrobot-v1 and CartPole-v0 from OpenAI Gym.

Since we use Gym as the backend for representing each environment, we don't need to do much reformatting on our input/output data. Specifically, we model each VPG and PPO using two neural networks, one denoting our current policy, and the other denoting the value function that estimates the expected discounted sum of rewards at any given state. Hence, our inputs come from Gym in tuples, representing variable factors within each environment, and our outputs consist of a probability action distribution for the policy network and an expected discounted return for the value network (for detailed explanations please refer to the Dataset and Features section).

The importance of the study is to show that while PPO also displays some variance in the results, it generally learns at a smoother pace than VPG does. Additionally, we also want to show that PPO and VPG both achieve decent reward thresholds across both classic control environments. On a more abstract level, we hope that this study helps push reinforcement learning beyond the shadow of supervised learning. Recent innovations in algorithms and computing power has enabled agent training to happen at a much faster rate and at a much less arbitrary and "unintelligent" manner. As a result, we hope that reinforcement learning will eventually fold into the equation for the advancement of truly artificially intelligent agents.

2 Background

Reinforcement learning methods fall into two categories: on-policy and off-policy learning. The main difference between on-policy and off-policy is that off-policy learning, such as Q-learning, utilizes a deterministic estimation policy updated based on future return. On the other hand, on-policy learning is not deterministic, thereby encouraging the agent to explore and allowing for faster convergence at the cost of introducing more variance. VPG is an on-policy method that aims to directly optimize the policy generated by a neural network. Likewise, PPO, initially released by OpenAI in 2017, is also an on-policy method that aims to some of the issues plaguing VPG and other on-policy methods. In these contexts, we define an episode to be a run from start to completion, or until an event horizon, and we denote each timestep as t. Subsequently, the policy is denoted as $\pi_\theta(a_t|s_t)$, where θ represents the parameters in the neural network and a_t and s_t represent the action and state/observation at each timestep.

3 Related work

First, we examine results presented using trust-region policy optimization (TRPO) by John Schulman et al. [2]. TRPO aims to calculate the KL-divergence between the new and old policy and use that to determine a trust-region to update the policy. While this is efficient in that it solves the issue of bad parameter updates in VPG, it is mathematically rough and difficult to implement in practice. In the paper, the authors show results of different variants of TRPO on the CartPole-v0 environment, all of which eclipse a reward threshold of 100. This is very similar to running PPO on CartPole-v0 as PPO was based off of the trust-region principle introduced in TRPO.

On the other hand, Martin Riedmiller et al. [3] evaluated three variants related to VPG on the CartPole-v0 environment. For their implementation of VPG, they utilized a similar algorithm to our implementation of Monte-Carlo vanilla policy gradient involving sampling trajectories of the episode and computing the baseline value estimate to optimize the policy network. However, instead of using an Adam optimizer, they used RMSProp as the step-size descent method. Nonetheless, they were able to see optimal results as well.

In the original paper introducing policy gradient methods by Richard Sutton et al. [4], the authors make note of using function approximation instead of the discounted return to point in the direction of the gradient. Likewise, our study makes use of this by using a neural network as a value function approximator. As explained by the authors, the benefit introduced here lies in reducing the potential variance exhibited by the random policy in VPG.

Douglas A. Aberden conducted a similar study but with a focus on the theoretical aspects behind applying policy gradient methods to partially observable Markov decision processes [5]. Although he thoroughly presented proofs for policy gradient applications, since it was published in 2003, no graphical plots or results were presented for the relevant classic control tasks since Gym was not around at that time.

Lastly, we examine work done using deep reinforcement learning in the continuous domain. According to Yan Duan et al. [6], despite the simplicity of Monte-Carlo vanilla policy gradient, it is still effective when optimizing for the "most basic and locomotion tasks" [6]. However, as we also noted in our experiment, VPG commonly suffers from premature convergence to local optima, which is the driving factor holding it back from performing as well as more advanced algorithms.

4 Dataset and Features

Due to the inherent nature of reinforcement learning, we do not have a set input dataset which is divided into training, validation, and testing subsections. Rather, the input data for our models comes from live observations from the environment that are solely dependent on the agent's current action taken (satisfying Markov property).

Acrobot-v1 is a classic control environment that contains an isolated two-link pendulum whose initial starting state is hanging directly downwards. The goal of the agent is to actuate the inter-joint such that the end of the lower link is raised above a given height and the episode ends when the agent succeeds or more than 500 timesteps have passed. The input observations of Acrobot-v1 consist of six numbers representing the sine and cosine of each rotational joint as well as the angular velocity for each link. The possible actions in this environment consist of only $+1$, 0, or -1 torque on the joint between the two links. The reward is -1 for each timestep the episode plays out.

CartPole-v0 is also a classic control environment that contains a cart with a pole in the center. From the initial state of the pole balancing upwards, the agent aims to keep the pole upright by moving the cart left or right. The episode ends when the pole is more than 15 degrees from vertical, or the cart moves more than 2.4 units from the center. Here, the observations include cart position, cart velocity, pole angle, and pole velocity at tip, where the cart position must be a value between -2.4 and 2.4 units. Similar to Acrobot-v1, the possible actions include 0 or 1, representing a push to the left or right, respectively, and the only possible reward is 1 for each timestep the agent survives.

5 Methods

We sample by taking our current policy and running it through the environment until termination, making sure to store the observations, actions, and rewards at each timestep. Using our stored information, we calculate the discounted sum of rewards given by the following function [7]:

$$R_t = \sum_{i=t}^{\infty} \gamma^{i-1} r_{t+i}$$

Monte-Carlo vanilla policy gradient (also known as REINFORCE) alternates between performing samples and optimizing the loss function of its policy network and value network. Specifically, the policy network takes in observations about the environment at the current timestep and outputs an action for that timestep. Unfortunately, since we do not know the optimal action at each state, we cannot treat this as a standard supervised learning problem. Rather, VPG optimizes the policy by computing an advantage and multiplying it by the log-likelihood of our probability distribution to ensure that actions with higher expected return receive higher probabilities and actions with lower expected return receive lower probabilities:

$$L^{VPG}(\theta) = \hat{E}[log(\pi_\theta(a_t|s_t))A_t]$$

where $\hat{E}$ represents the average over all timesteps and A_t represents the advantage. Thus, the gradient update function becomes:

$$\nabla L^{VPG}(\theta) = \hat{E}[\nabla_\theta log(\pi_\theta(a_t|s_t))A_t]$$

To calculate the advantage, we calculate the discounted sum of rewards at each timestep as well as a baseline estimate for the discounted return and subtract the two. Similar to the policy network, we use a value network to compute the baseline that consists of a simple, one-layer neural network with ten neurons that performs regression on observations and outputs the expected return from that state. By subtracting the baseline estimate from the computed discounted return of the current episode, we reduce the variance of our actual discounted return.

One major problem with VPG was the fact that due to the random nature of the policy (sampling from action distribution rather than taking the argmax), it wasn't difficult for the agent to make one terrible move and push itself into an unfavorable parameter region upon which all future updates would be dependent on and the agent may never recover. To address this, OpenAI introduced two versions of proximal policy optimization, adaptive KL penalty and clipped PPO. In this study, we chose to use the clipped PPO since, on average, it performs better than its counterpart. The goal of clipped PPO is to clip the update between $1-\epsilon$ and $1+\epsilon$ to decrease the chance that the agent makes bad updates. This bound serves as the equivalent of the trust-region in TRPO except it is much easier to implement since the only major difference between PPO and VPG lies in correctly modifying the objective function and its dependencies to the one shown below [1]:

$$L^{PPO}(\theta) = \hat{E}[min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t]$$

where $r_t(\theta)$ represents $\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$, the ratio between the new and old policies, and the clipping function sets the gradient to zero once the ratio breaches the bounds. By clipping the probability ratios, the agent is penalized for moving outside of the interval, and by taking the minimum of both, the final objective ensures a lower bound on the unclipped objective.

6 Results and Discussion

The hyperparameters for this project (both VPG and PPO) included: the number of neural network layers for both the policy and value networks, the number of neurons for each layer, the learning rate (α), the discount factor (γ), the clip ratio, and the number of episodes. While I considered the effect in the varying results using different hyperparameters, the major factor for choosing the optimal hyperparameters was to simply use values from existing architecture. The reasoning behind this choice is because reinforcement learning is unlike other types of learning in that agent learning hyperparameters transfer well, and as long as the learning rate and the discount factor are not far-fetched, the agent is guaranteed to reach a good reward threshold in the long run. Thus, we used two hidden layers and 32 neurons for the policy networks, one hidden layer and ten neurons for the value network, $\alpha = 0.01$, $\gamma = 0.98$, clip ratio $= 0.2$, and 1000 episodes to give the agent time to learn while not over-committing to failing trials.

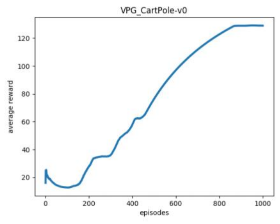

Figure 1: VPG results on CartPole-v0

As we can see from the above graph, VPG performs well on CartPole-v0, exceeding the 100 reward threshold with an overall, relatively smooth learning curve. Consequently, the small dips in the graph demonstrate the high variance of this on-policy method. In fact, while all the VPG trials on CartPole-v0 were successful in getting the agent to learn, a majority of them did not have monotonically increasing learning curves.

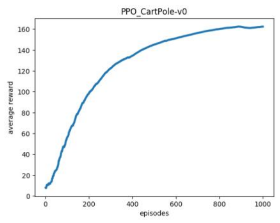

Figure 2: PPO results on CartPole-v0

In comparison, PPO's learning curve was generally much smoother than that of VPG. Thus, not only did PPO, on average, exceed VPG in average reward threshold, it also learned at a better pace. However, due to the random nature of on-policy methods, PPO can also exhibit relatively poorer results.

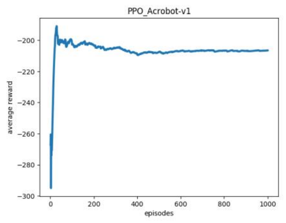

Figure 3: PPO results on Acrobot-v1

Here, in Figure 3, we show results of PPO on Acrobot-v1. Although the agent started to learn well, it ended up getting caught in a bad saddle point due to an unfavorable parameter update. Even so, PPO still demonstrated decent results across 1000 iterations, and over many trials, these imbalances become more or less outliers as PPO scales better than VPG.

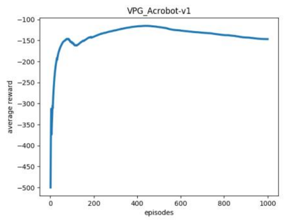

Figure 4: VPG results on Acrobot-v1

In contrast, here is a baseline comparison on Acrobot-v1 by VPG, where the agent begins sloping out at around -120. However, it should be noted that due to VPG's relatively large variance, many of our trials on both CartPole-v0 and Acrobot-v1 ended with lower reward thresholds and slower learning due to the agent being stuck in bad parameter regions. In fact, without a baseline to subtract from the discounted sum of rewards, VPG will display even more variance due to the random nature of the policy [1]. Unfortunately, since both VPG and PPO are on-policy, they suffer from sampling inefficiency as they forget data very fast in order to avoid the introduction of a bias to the gradient estimator, thus making numerous rollout samples a necessity.

7 Conclusion

In conclusion, while PPO may sometimes not out-perform VPG (comparing Figure 4 and Figure 3), on average, an agent learning by PPO exhibits better learning and reward thresholds than an agent learning by VPG on classic control environments. That being said, it seems that on classic control tasks, there is not an extremely significant difference between the results displayed by VPG and those of PPO as both tend to perform well in the long run. While it may not be apparent from the results of our study, compared to off-policy methods, policy gradients such as VPG and PPO are more advantageous in the continuous space because off-policy methods such as Q-learning require a full scan of the action space and thus are very computationally expensive.

8 Future Work

We eventually aim to refine and modify VPG and PPO implementations to work for continuous OpenAI Gym environments. Then, we can implement trust-region policy optimization (TRPO) and twin delayed deep deterministic policy gradients (TD3) and compare results to ones from VPG and PPO on both continuous and classic control tasks. Lastly, using Pickle and Python MPI packages, we would like to develop parallelized implementations for faster episode rollouts and agent training.

9 Code

All project code at (invite sent to cs230 github):

`https://github.com/andrew230/cs_230_final_project`

10 Contributions

Andrew Zhang initiated the project, wrote the pertinent code, conducted the training and testing, provided graphical figures, and typed up the project, proposal, milestone, and final report.

References

[1] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, “Proximal Policy Optimization Algorithms,” *arXiv*, 2017

[2] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, Pieter Abbeel, "Trust Region Policy Optimization," *Advances in International Conference on Machine Learning*, 2015

[3] Martin Riedmiller, Jan Peters, Stefan Schaal, "Evaluation of Policy Gradient Methods and Variants on the Cart-Pole Benchmark," *Advances in IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, 2007

[4] Richard S. Sutton, David McAllester, Satinder Singh, Yishay Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," *Advances in Neural Information Processing Systems*, 2000

[5] Douglas Alexander Aberdeen, "Policy-Gradient Algorithms for Partially Observable Markov Decision Processes," *Advances in Australian National University Open Research Reposity*, 2003

[6] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, Pieter Abbeel, "Benchmarking Deep Reinforcement Learning for Continuous Control," *Advances in International Conference on Machine Learning*, 2016

[7] OpenAI, “Spinning Up in Deep RL,” 2018, https://blog.openai.com/spinning-up-in-deep-rl/