Deep Reinforcement Learning for Classic Control Tasks

Andrew Zhang (andrewdu)
CS 230 (Deep Learning), Stanford University

Abstract

- Deep reinforcement learning (RL) was been shown to be effective on well-defined environments such as standard Atari games [1].
- This study aims to present results on more abstract and robotic environments such as Acrobot-v1 and CartPole-v0 from OpenAI Gym using deep RL implementations of Monte-Carlo vanilla policy gradients (VPG) and proximal policy optimization (PPO).
- Over 1000 episode rollouts across multiple runs, VPG and PPO exhibited optimal results on both environments. However, while VPG and PPO 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.

Methods

- REINFORCE (Monte-Carlo VPG) uses the return of episode rollouts to update the policy parameters by optimizing the following objective [2]:

$$L^{PG}(\theta) = E_t\left[log\left(\pi_\theta(a_t|s_t)\right) A_t\right]$$

- $\pi_\theta(a_t|s_t)$ represents the policy and A_t denotes the advantage at each timestep, which is calculated by subtracting a baseline value function estimate (1-layer neural network) from the total discounted sum of rewards [2].
- One major issue with VPG is that a large update can push the agent into an unfavorable parameter region, upon which the agent may never recover [1]. To address this issue, PPO instead optimizes a clipped surrogate objective to ensure that the ratio between the new and old policy stays within $1 - \epsilon$ and $1 + \epsilon$:

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

- For VPG, we used a 2-layer, 32 hidden units neural network that outputted a softmax distribution with $\alpha = 0.01, \gamma = 0.98$.
- For PPO, both the new and old policies used the same architecture as VPG with a clip ratio of 0.2, except the old policy was not trainable and we updated the parameters from the new to the old policy every two iterations.

Data

- Acrobot-v1 (swing two link pendulum above base height):
  - Observations – 6-tuple containing sin() and cos() of the two rotational joint angles and the joint angular velocities
  - Actions – +1, 0 or -1 torque on the joint between the two pendulum links
- CartPole-v0 (balance a pole on a cart):
  - Observations – 4-tuple containing cart position, velocity, angle, and tip velocity
  - Actions – 0 or 1 denoting a push to the left or right

Results

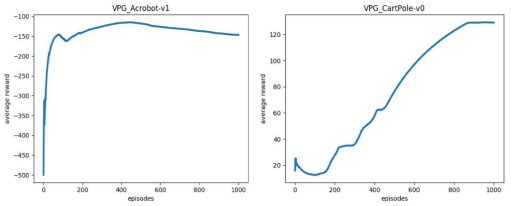

Figure 1: Vanilla policy gradients average return over episodes

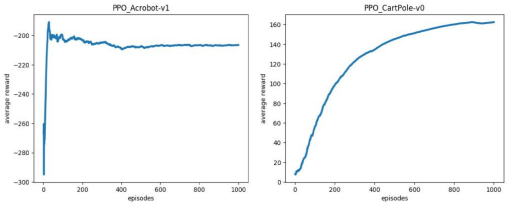

Figure 2: Proximal policy optimization average return over episodes

Conclusions

- As expected, over 1000 iterations, both VPG and PPO were able to make benchmark reward average thresholds of -200 and 100 for Acrobot-v1 and CartPole-v0, respectively,.
- 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.
- However, PPO generally learns at a smoother pace than VPG due to the clipping function providing a surrogate "trust-region" for the policy to update in.
- Without a baseline to subtract from the discounted sum of rewards over all timesteps, VPG will display more variance due to the pseudo-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.
- Policy gradient methods 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.

Future Work

- Refine and modify VPG and PPO implementations to work for continuous OpenAI Gym environments.
- Implement trust-region policy optimization (TRPO) and twin delayed deep deterministic policy gradients (TD3) and compare results to ones from VPG and PPO.
- Use Pickle and Python MPI packages to develop parallelized implementations for faster episode rollouts and agent training.

References

[1] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, Oleg Klimov, "Proximal Policy Optimization Algorithms," *arXiv*, 2017
[2] OpenAI, "Spinning Up in Deep RL," 2018, https://blog.openai.com/spinning-up-in-deep-rl/