Learning Options In Continuous State And Action Spaces

David Nitchi, Franco Del Balso
McGill University
Address
`david.nitchi@mail.mcgill.ca, franco.delbalso@mail.mcgill.ca`

Abstract

Our work extends on the Option-Critic algorithm, a learning framework designed to leverage temporal abstraction by learning temporally extended actions known as options. While the original work was general enough to allow for continuous action spaces in theory, it was only shown to work on discrete action environments. We show that to achieve results on continuous action spaces, the algorithm must be modified. Our results indicate that for fine-grained control tasks, the use of options can complicate learning while yielding no benefit. However, for tasks of a sufficiently compositional nature, options can provide improved performance, even in continuous action domains.

1 Introduction

1.1 Background

In reinforcement learning, agents take actions and carve out trajectories in time. Some tasks might require very fine grained and instantaneous control while others might require long term considerations. In more complex environments, an optimal solution might consist of a hybrid composition of both short term and long term actions. One framework that allows models to take advantage of this temporal abstraction is known as Options Learning Sutton et al. [1999].
An option ω has 3 components. First, it has an initiation set $\mathcal{I}_\omega$ which contains all the states from which ω can be executed. Next, it has an intra-option policy π_ω which defines the behavior of the model as it is executing ω. Finally, it has a termination function $\beta_\omega : S \to [0,1]$ which assigns at each state a probability for halting the execution of ω. A common simplifying assumption, which we also adopt, is that the initiation sets span the entire state space. In other words, all options can be executed from all states.
The Option-Critic framework Bacon et al. [2016] was developed to allow option learning using differentiable function approximators for the intra-option policies and termination functions. Given an option ω with intra-option policy $\pi_{\omega,\theta}$ parametrized by θ and termination function $\beta_{\omega,\phi}$ parameterized by ϕ, the relevant policy gradients take the following forms:

$$\frac{\partial L(\theta)}{\partial \theta} = \mathbb{E}_t \left[\frac{\partial log \pi_{\omega,\theta}(a_t|s_t)}{\partial \theta} (Q_U(s_t,\omega,a_t) - Q_\Omega(s_t,\omega)) \right] \tag{1}$$

where $Q_\Omega(s,\omega)$ is the value of a state-option pair and $Q_U(s,\omega,a)$ is the value of taking action a given a state-option pair. Note that $Q_\Omega(s,\omega)$ does not actually appear in the original form of this policy gradient, but is added as a baseline to reduce variance and improve learning.

$$\frac{\partial L(\phi)}{\partial \phi} = \mathbb{E}_t \left[-\frac{\partial \beta_{\omega,\phi}(s_t)}{\partial \phi} (Q_\Omega(s_t,\omega) - V_\Omega(s_t) + \eta) \right] \tag{2}$$

where $V_\Omega(s)$ is the value over all options and a small number η is added to the advantage in order to prevent shrinkage of options. It acts as a small margin to prevent the termination probability from being driven up when an option has a value that is only slightly lower than average.
With these policy gradients in hand, the next step is to define a policy over options $\pi_\Omega(\omega|s)$ to be used at the start of every episode and after every option termination. The simplest choice is to have an ϵ-greedy policy conditioned on the state-option value $Q_\Omega(s,\omega)$. Finally, one must decide on a method to estimate the values. We opt to use function approximation to learn $Q_\Omega(s,\omega)$, then estimate $Q_U(s_t,\omega_t,a_t)$ using $r_{t+1}+\gamma((1-\beta_{\omega_t,\phi}(s_{t+1}))Q_\Omega(s_{t+1},\omega_t)+\beta_{\omega_t,\phi}(s_{t+1})\max_\omega Q_\Omega(s_{t+1},w))$ as suggested by the authors of Option-Critic. Let the parameters of Q_Ω be denoted ζ. We also estimate $V_\Omega(s)$ using $\sum_\omega \pi_\Omega(\omega|s)Q_\Omega(s,\omega)$.
The vanilla Option-Critic learning algorithm consists of making one update to the values, intra-option policies and termination functions at every step in the environment. If the option terminates, π_Ω is used to select the next option, then the process is repeated.

1.2 The trajectory of our project

In our proposal, we outlined our goal of implementing this Option-Critic algorithm and using it to learn on a quadrotor simulation environment called PyFlyt Tai et al. [2023], comparing our performance against a state of the art PPO implementation from Stable Baselines 3 Raffin et al. [2021]. What we did not highlight, and what turned out to be the biggest crux of the project, is the fact that this is a continuous action environment. Furthermore, the curriculum learning aspect of the proposal had to be discarded because our algorithm was not able to learn the initial task of hovering, despite being able to learn the more difficult waypoints task which will be discussed later. The methods and algorithm presented in the Option-Critic paper turn out to not be very well suited for an environment that is continuous in both action and state space. As such, a large amount of our time was spent exploring ways to improve the learning algorithm to account for continuous actions. We leave the details of this exploration to the video portion of our project submission. We ended up with an algorithm inspired by the PPOC (Proximal Policy Option-Critic) algorithm Schulman et al. [2017]. The next section will take you through our algorithm and highlight the aspects that differ from PPOC. After that, we present the experiments we did with our algorithm on different environments. This is followed by a discussion of what we would like to improve and how we would go about it.

2 Algorithm

The algorithm we use follows a PPO style update regime, where we let the agent act in the environment while saving information about the states, actions and rewards. After a certain number of steps ,called the horizon, the agent performs an update step where for each option ω we update π_ω, $Q_\Omega(\omega)$ and β_ω based on experience obtained while using ω. These updates are repeated for multiple iterations through all of the saved replay information. The intra-option policy is updated using $\mathcal{L}^{\text{PPO}}$ loss described in equation (3), where $\rho_t(\theta)=\frac{\pi(A_t,S_t)}{\pi_{old}(A_t,S_t)}$ and A_t^π is the advantage of taking an action given the current policy, computed as $Q_U(s,\omega,a)-Q_\Omega(s,\omega)$. Our approach is similar to the algorithm described in PPOC, however, there are two key differences. The first is that we do not learn a policy over options and simply use ϵ-greedy. The second is that PPOC Schulman et al. [2017] uses GAE to estimate advantages, whereas we use a one-step boot-strapping approach to estimate advantages.

$$\mathcal{L}^{\text{PPO}}(\theta)=\mathbb{E}_t\left[\min\left(\rho_t(\theta)A_t^\pi,\ \text{clip}\left(\rho_t(\theta),1-\epsilon,1+\epsilon\right)A_t^\pi\right)\right] \tag{3}$$

Algorithm 1 Option Critic with Intra-Option PPO

```
1:  for iteration $= 1, 2, \ldots$ do
2:      $c_t \leftarrow 0$
3:      $s_t \leftarrow s_0$
4:      Choose $\omega_t$ using $\epsilon$-greedy selection over $Q_\Omega(\omega, s_t)$
5:      repeat
6:          Choose $a_t$ according to $\pi_\omega(a_t \mid s_t)$
7:          Take action $a_t$ in $s_t$, observe $s_{t+1}, r_t$
8:          $\hat{r}_t = r_t - c_t$
9:          if $\beta$ terminates in $s_{t+1}$ then
10:             Choose $\omega_{t+1}$ using $\epsilon$-greedy selection over $Q_\Omega(\omega, s_{t+1})$
11:             $c_t = \eta$
12:         else
13:             $c_t = 0$
14:         end if
15:     until $T$ timesteps
16:     Compute the advantage estimates for each timestep
17:     for $\omega = \omega_1, \omega_2, \ldots$ do
18:         $\theta_{\text{old}} \leftarrow \theta$
19:         for $K$ optimizer iterations with minibatches $M$ do
20:             $\theta \leftarrow \theta + \alpha_\theta \frac{\partial \mathcal{L}_t^{\text{PPO}}}{\partial \theta}$
21:             $\phi \leftarrow \phi - \alpha_\phi \frac{\partial \beta_\omega(s_t)}{\partial \phi} \left(Q_\Omega(s_t, \omega) - V_\Omega(s_t) + \eta\right)$
22:             $\zeta \leftarrow \zeta - \alpha_\zeta \frac{\partial (Q_U(s_t, \omega, a_t) - Q_\Omega(s_t, \omega))^2}{\partial \zeta}$
```

3 Experiments

For our experiments, we tested out our adjusted option-critic algorithm on 3 different tasks with continuous action spaces for 1, 2, and 4 options. The 1 option case represents a control, indicating whether or not there is any actual benefit to using multiple options in this environment. We ran the experiments with 0.0001 learning rate and $\epsilon = 0.15$ with a buffer horizon of 5000 steps.

We first consider a simple control task called Inverted Pendulum from MuJoco Todorov et al. [2012]. The environment involves a cart that can move linearly with a pole balancing on it. The agent can push the cart left or right. A reward of +1 is given for each time step the pole is upright, with a maximum of 1000 steps per episode. We then tested our algorithm with 1, 2 and 4 options in this environment, running 500 thousand steps. The goal of this experiment is to examine the performance of options in a simple low-level control task which is not suited to it. In this environment we expect to see that the 1 option case will be the best performing since there is not really a need for multiple options in such a simple control task.

For the second experiment, we used the Walker2d environment from MuJoco Todorov et al. [2012]. In this environment the agent controls a pair of legs with a joints at the hip, knee and ankle. The rewards in the environment are +1 per time step that the walker remains upright, a positive reward for the distance traveled in the forward direction and a negative reward for taking actions that are too large. This is a more complicated control task where options would make sense to use since the task can be broken up into subroutines (like taking a step) to accomplish the goal. Here we expect to see that using multiple options will outperform using a single option because we think multiple options will allow the agent to better break up the task into simpler sub problems that it can learn with options. Again we run this environment for 500 thousand steps.

The last environment we tested was QuadX-Waypoints from PyFlyt Tai et al. [2023]. In this environment, the agent controls a 4 rotor UAV with the goal of flying through randomly placed waypoints in the environment. For this environment, we decided to run 1.5 million steps instead of 500 thousand since the task and the environment are more complicated than the previous two. As such, we were only able to test the performance of the 1 and 2 option agents. The reward in this environment is $1/\delta$ at each time step where δ is the distance to the next waypoint. Furthermore, there is a reward of -100 for a crash and +100 for passing through a waypoint. In this environment the waypoints represent natural subgoals in the problem so we again expect multiple options to outperform the single option.

4 Results

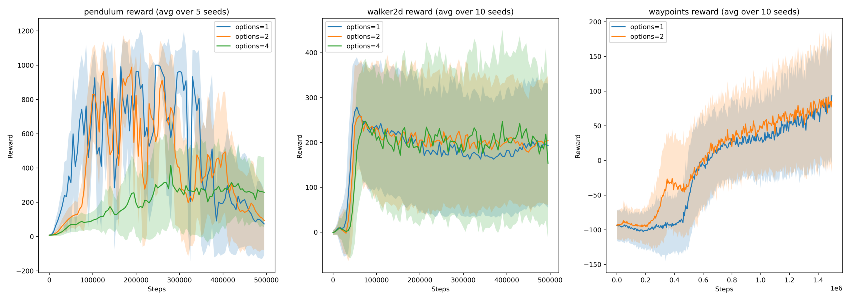

Figure 1: Reward plots over time for Option-Critic with intra-option PPO for each of the environments. Reward curves are smoothed using averaging over bins of 5000 steps.

In the first environment, inverted pendulum, we first see that 4 options is much less performant than 1 or 2 options in this task. This makes sense since 4 options is very excessive for such a simple task and the extra burden brought on by learning 4 intra-option policies and Q values leads to poor performance. The performance between 1 and 2 options in this task is quite similar, however, 1 option seems to have a slight edge on 2 options. 1 option learns faster and is able to attain the maximum possible reward in the environment on average more than 2 options. Overall, these results were expected since this task is not great for options as explained earlier. In the second environment, Walker2d, we see that all 3 variations perform around the same level, getting to about 200 reward at 50-100 thousand steps and staying near that mark for the rest of the experiment. We also notice that the speed at which they get to this level of performance is increasing in the number of options, i.e. 1 option learning the quickest and 4 options learning the slowest. We also see that 4 options is now performing at the same level as the other methods, which is likely due to the nature of the task, (controlling a pair of legs to walk forward) allowing for better usage of temporally extended actions. We were surprised that the multiple option frameworks were not able to outperform the single option in this task since we thought the usefulness of temporally extended actions would give a larger boost in performance than they did. In the last environment, Waypoints, we see the first indications of multiple options possibly outperforming the single options variant. We can see that 2 options initially learns quicker than 1 option, and furthermore around 1 million steps, it seems to be slightly outperforming 1 option. However, around 1.5 million steps they seem to perform at about the same level. Still, the fact that 2 options is learning faster than a single option is significant since the individual intra-option policies are getting much less experience than the policy from the single option case. This indicates that the algorithm is able to leverage having multiple different options to learn the task more quickly, which we think is due to the presence of natural subgoals in the environment. Subgoals are known to be very conducive to learning options which we see displayed in our results. However, overall the difference between the single and two option case was not that large, which we again found surprising due to the task being well suited for options.
We also compared PPO from stable baselines 3 Raffin et al. [2021] to our algorithms, PPO massively outperformed our algorithms in all 3 environments and the PPO rewards overtime can be viewed in the appendix.

5 Conclusion

Our algorithm allowed us to explore the potential benefits of temporally extended actions. It also helped us understand how the utility of temporal abstraction is very much at the mercy of the specific environment in question. In a fine-grained control environment, like Inverted Pendulum, options are likely to serve as nothing more than extra baggage. However, in an environment that displays a sufficient amount of compositionality, such as the waypoints task, there are benefits to learning options. Our algorithm is far from perfect, but in implementing it we learned a lot more than we would have if we simply just read about options. We believe that a hands-on approach to discovering reinforcement learning is extremely valuable, and this project has allowed us to accrue some of that value.

6 Appendix

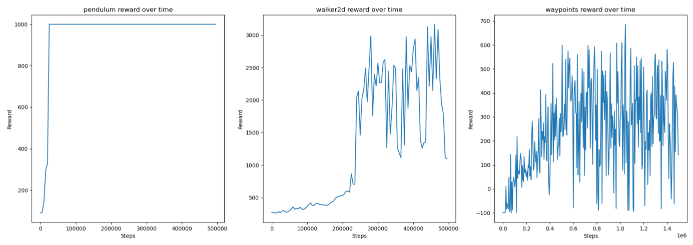

Figure 2: Reward plots for PPO over time for each of the environments.

While our original goal was to achieve comparable performance to this implementation of PPO, we quickly realized that this was way above our paygrade. We underestimated how fine tuned the stable baselines implementations are, and we would have needed much more time to understand and include every little technique that they do. We thus found it much more valuable to simply compare our multi-option results to our single option results since they are all on the same playing field.

7 Statement of contribution

Both authors worked together on realizing the algorithm, collaborating using a GitHub repository. Both authors contributed to writing the report. The Powerpoint video was narrated by Franco Del Balso.

References

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture, 2016. URL `https://arxiv.org/abs/1609.05140`.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Noah Ernestus, and Quentin Dormann. Stable-baselines3: Reliable reinforcement learning implementations. `https://github.com/DLR-RM/stable-baselines3`, 2021. Accessed: 2025-04-23.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL `https://arxiv.org/abs/1707.06347`.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(99)00052-1. URL `https://www.sciencedirect.com/science/article/pii/S0004370299000521`.

Jun Jet Tai, Jim Wong, Mauro Innocente, Nadjim Horri, James Brusey, and Swee King Phang. Pyflyt – uav simulation environments for reinforcement learning research, 2023. URL `https://arxiv.org/abs/2304.01305`.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control, 2012.