Skill Discovery in Reinforcement Learning via the Option-Critic Architecture

Akshaya Jeyaram
MSc Data Science
University of Bath

July 29, 2025

Abstract

This report presents the implementation and experimental evaluation of the Option-Critic architecture, a method for skill discovery in reinforcement learning (RL). Alongside the primary model, two baseline agents—Deep Q-Network (DQN) and Proximal Policy Optimization (PPO)—were implemented for comparison. The environment used was a Four Rooms domain, designed to test the capabilities of temporally extended actions. Detailed implementation notes, equations, code-level design decisions, experimental metrics, and visualizations are included. The results demonstrate the superior efficiency and interpretability of the Option-Critic framework, particularly in environments with hierarchical or sparse reward structures.

1 Introduction

Skill discovery in RL refers to the agent's ability to learn temporally extended actions, or "options," that allow for hierarchical decision making. The Option-Critic (OC) architecture, introduced by Bacon et al. (2017) [1], enables end-to-end learning of options, intra-option policies, and termination conditions. This report focuses on implementing a vanilla Option-Critic agent and comparing it against PPO and DQN baselines in the Four Rooms environment.

2 Option-Critic Architecture

2.1 Overview

The OC agent learns a set of k options. Each option o consists of:

- An intra-option policy $\pi(a \mid s, o)$
- A termination function $\beta(s, o) \in [0, 1]$
- An option-value function $Q(s, o)$

At each step, the agent decides whether to terminate the current option via β, and if so, samples a new option o' based on the greedy Q values. The action is then chosen using $\pi(a \mid s, o)$.

2.2 Loss Functions

The architecture optimizes:

- **Critic Loss (TD Error):**

$$L_{\text{critic}} = \frac{1}{2}\left[Q(s, o) - y\right]^2 \tag{1}$$

$$y = r + \gamma \left[(1 - \beta(s', o))Q(s', o) + \beta(s', o') \max_{o'} Q(s', o')\right] \tag{2}$$

- **Actor Loss:**

$$L_{\text{actor}} = -\log \pi(a \mid s, o)A(s, o) - \lambda H[\pi(a \mid s, o)] \tag{3}$$

$$A(s, o) = y - Q(s, o) \tag{4}$$

- **Termination Loss:**

$$L_{\text{term}} = \beta(s, o)\left(Q(s, o) - \max_{o'} Q(s, o') + \eta\right)(1 - d) \tag{5}$$

3 Detailed Code Explanation

3.1 Agent: OptionCriticMLP

This class defines the core architecture for the MLP-based OC agent:

- A two-layer MLP processes the state input into a feature vector of dimension 64.
- $Q(s, o)$ is computed using a linear layer over the feature vector.
- $\beta(s, o)$ is predicted using a sigmoid over a separate linear layer.
- $\pi(a \mid s, o)$ is parameterized by a weight tensor of shape (num_options, 64, num_actions), such that the feature vector is multiplied with the corresponding weight slice.

Action sampling uses a Categorical distribution derived from the softmax over logits:

$$\text{logits} = \text{features} \cdot W_o + b_o \tag{6}$$

$$\text{action} \sim \text{Categorical}(\text{Softmax}(\text{logits}/T)) \tag{7}$$

3.2 Termination and Option Switching

The method `should_terminate` uses the learned $\beta(s, o)$ to stochastically decide whether to switch options. If termination occurs, a new greedy option is selected using $\arg\max_o Q(s, o)$. A minimum option duration constraint is enforced to prevent unstable switching behavior.

3.3 Experience Buffer

The `ExperienceBuffer` class implements a fixed-size replay memory using `collections.deque`. Each entry stores a transition tuple:

$$(s_t, o_t, r_t, s_{t+1}, d_t)$$

Samples are drawn uniformly during training to compute critic gradients.

3.4 Gradients: Actor and Critic

`gradients.py` defines two key functions:

- `compute_critic_gradient()` — Computes TD target using termination probabilities and next state Q values, then applies mean squared loss to current Q.
- `compute_actor_gradient()` — Computes the policy gradient with entropy regularization and a termination penalty for early switching.

3.5 Training Loop

In `train_option_critic.py`, each episode executes the following:

1. Reset environment and initialize o_0
2. For each timestep:
   - Sample $a_t \sim \pi(a \mid s_t, o_t)$
   - Observe r_t, s_{t+1}, d_t
   - Store in replay buffer
   - Check termination condition; if true, sample new o_{t+1}
3. Periodically sample mini-batches and compute actor and critic gradients
4. Update target network

3.6 Logging and Evaluation

The `Logger` class records per-step and per-episode metrics (losses, entropy, epsilon, reward), exports CSV logs, and prints summaries.

Visualizations include:

- Smoothed reward curves
- Option length averages
- Entropy trends
- Termination probabilities

4 Challenges Faced During Development

- **Termination collapse:** Without clipping or regularization, β converged to 0 or 1 early.
- **Instability in TD targets:** Required use of target networks to prevent oscillation.
- **Hyperparameter sensitivity:** Entropy weight λ and termination regularizer η required annealing and grid search.
- **Training instability:** Option flapping was mitigated by enforcing a minimum option duration.

5 Baselines

5.1 DQN

A Deep Q-Network with target updates, replay buffer, and epsilon-greedy policy was implemented in `dqn_agent.py`. The loss minimized was:

$$L_{\text{DQN}} = \left[r + \gamma \max_{a'} Q_{\text{target}}(s', a') - Q(s, a) \right]^2 \tag{8}$$

5.2 PPO

The PPO baseline used a clipped surrogate objective:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t A_t, \text{clip}(r_t, 1 - \epsilon, 1 + \epsilon) A_t \right) \right] \tag{9}$$

where $r_t = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$.

6 Experimental Results

6.1 Environment

All agents were tested in the Four Rooms domain. The task was to navigate to a random goal location.

6.2 Performance Metrics

- Episode reward curves (smoothed)
- Option duration statistics
- Entropy trends
- Termination probabilities

6.3 Key Findings

- Option-Critic showed more stable and faster convergence compared to DQN.
- PPO and OC achieved similar final rewards, but OC revealed interpretable skills.
- Entropy annealing preserved exploration in early stages.
- Learned options often aligned with subgoal-like behavior.

7 Conclusion

The Option-Critic architecture effectively discovers temporally extended skills in sparse-reward environments. Compared to DQN and PPO, it provides both performance and interpretability benefits. Future work includes extending the implementation to Soft Option-Critic and applying it to more complex domains.

References

[1] Pierre-Luc Bacon, Jean Harb, and Doina Precup.
The Option-Critic Architecture. arXiv preprint arXiv:1609.05140, 2017.