# Analysis and Comparison of Option-Critic and Actor-Critic Architectures in Multi Agent Reinforcement Learning

**Atharva Wandile, Douglas Schonholtz**
Northeastern University ,Boston , Masssachusetts , 02115
wandile.a@northeastern.edu , schonholtz.d@northeastern.edu
Code can be found at: https://github.com/atharvaw1/CS5180-project

**Abstract**

Hierarchical methods like option critic have shown to improve transfer ability and environment changes when compared to a2c methods, however, there hasn't been much work proving their superiority in the multi-agent setting. We compare option critic to n-step A2C and n-step soft actor-critic methods in single-agent and multi-agent domains. Specifically, we compare those algorithms in the four rooms environment and the simple spread MPE petting zoo environment(Terry et al. 2020). We find that soft actor-critic with a step size of 5 appears to be able to converge faster than option critic in the four rooms domain. However, in the more complicated environment of simple spread, we find soft actor-critic is outperformed by option critic while the environment isn't too complicated. Once the state space reach 3 agents, that proved to be too complicated for option critic as well and neither algorithm could sufficiently learn.

## Introduction

Reinforcement learning (RL) methods have recently shown a wide range of positive results, including beating humanity's best at Go(Silver et al. 2016), learning to play Atari games just from the raw pixels(Mnih et al. 2013), and teaching computers to control robots in simulations or in the real world. These achievements are the culmination of research on trial-and-error learning and optimal control since the 1950s.

However, despite its popularity and success RL suffers from a variety of defects that hinder learning and prevent it from being applied to more complex environments. Hierarchical Reinforcement Learning (HRL) aims at alleviating precisely this learning complexity by breaking down specific parts of learning (Flet-Berliac 2019).

The main weaknesses of RL, in comparison to the promises of HRL, can be broken down as follows.
**Sample efficiency:** training data generation is often a choke point and many existing RL methods are not data efficient. With HRL, sub-tasks and abstract actions can be used in different tasks on the same domain. Often referred to as transfer learning, this abstraction may make previously intractable problems feasible.
**Scaling up:** the application of classic RL to problems with large action and/or state space is infeasible. This is due to a combinatorial explosion of features attempting to be mapped to some finite set of actions. As the dimensionality of inputs increases the computation an algorithm must execute to predict the output explodes combinatorically. HRL aims to decompose large problems into smaller ones. This should enable more sample-efficient learning.
**Generalization:** trained agents can solve complex tasks, but if we want them to transfer their experience to new (even similar) environments, most state-of-the-art RL algorithms will fail. This is due to the same problems discussed in scaling up. Prioritization of transfer learning and abstract representations is extremely valuable.
**Abstraction:** state and temporal abstractions allow the RL algorithm to simplify the problem since resulting sub-tasks can effectively be solved by existing RL approaches.

In addition, all the basic algorithms for reinforcement learning are so-called flat methods. They treat the state space as a huge, flat search space, meaning that the paths from the starting state to the target state are very long. If we look at this with the example of performing any physical task, it would give us a sequence of actions solely composed of a series of muscular micro-contractions. Having to manually process this in order to accomplish any task would be intractable for humans and it would be far easier for us to process abstractions of the pieces of the task that we hope to accomplish. The same process can be applied to existing RL methods. In addition, the length of these paths dictates the cost of learning, as information on future rewards must be disseminated backwards along these paths. In short, the reward signal is weak and delayed due to having to propagate along many minute actions rather than along a few abstract conceptions.

Option critic methods (Bacon, Harb, and Precup 2016) attempt to address these issues by adding a layer of temporal abstraction to the MDP so that decisions can be made at a higher level and have been shown to be fairly successful in single-agent domains.

However, there has not been a lot of empirical evidence for these methods in the multi-agent reinforcement learning (MARL) setting. Since MARL is a lot more complex in terms of state, action, and reward spaces, it is not obvious that these methods will perform similarly in this domain. One simple distinction is that agents cannot act purely greedily in these environments and must learn to cooperate with other agents to reach the goal.

Hence this paper attempts to bring more clarity to this question and compares 2 methods (actor-critic and option-critic) in the single-agent and multi-agent setting.

## Background

### Actor Critic

Actor-critic algorithms are a popular class of value-based RL algorithms. In actor-critic algorithms, the agent maintains two separate neural networks: the actor-network, which estimates the action to take in each state, and the critic network, which estimates the expected return of each action.

In TD(0) actor-critic, the actor-network and the critic network are updated at each time step. The actor-network is updated using policy gradient methods, while the critic network is updated using the TD(0) temporal difference error.

$$\begin{aligned} \delta = R + \gamma V^{\pi_\theta}(s') - V^{\pi_\theta}(s) \\ w \leftarrow w + \delta \nabla V^{\pi_\theta}(S, w) \\ \theta \leftarrow \theta + \delta \nabla \ln \pi(A|S, \theta) \end{aligned} \tag{1}$$

where $\delta$ is the TD(0) error, $r$ is the reward received at time step $t$, $\gamma$ is the discount factor, $s'$ is the next state, and $\alpha$ is the learning rate.

The actor and critic networks are updated at each time step using these equations, and the process continues until the agent converges to a satisfactory policy.

All of this can end up stuck in local optima due to actor critic being on-policy. One thing that can help with this is by maximizing entropy, or the certainty that an action will have a greater value associated with it with respect to the others. This helps foster exploration of uncertain states.

### Soft Actor Critic

This method is referred to as Soft-Actor-Critic (SAC). In soft actor-critic (SAC), the actor-network is trained to maximize the expected return predicted by the critic network, as well as to maximize the entropy of the action distribution. This helps the agent to explore different actions and to find a more diverse and robust set of policies. (Haarnoja et al. 2018)

The actor-network is updated using the following equation, which combines the policy gradient update with the entropy maximization term:

$$\nabla_\theta J(\theta) \leftarrow \mathbf{E}_{s \sim \rho^{\pi_\theta}}[\nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a) - \alpha \mathcal{H}(\pi_\theta(s))] \tag{2}$$

where $\pi_\theta$ is the policy learned by the actor-network, with parameters $\theta$, $\rho^{\pi_\theta}$ is the state distribution under the policy, $Q^{\pi_\theta}$ is the action-value function approximated by the critic network, $\alpha$ is a hyper-parameter that controls the strength of the entropy term, and $\mathcal{H}(\pi_\theta(s))$ is the entropy of the action.

What this means in practice, is that for each step the loss can be updated with a term of some entropy constant multiplied by the negative sum of the average actor's action probability times the log values of the actor's action probabilities.

$$entropy \leftarrow -(\log \pi_\theta(s, a) + \pi_\theta(s, a)) \tag{3}$$

By adding this term multiplied by some small constant to every loss in every timestep we should see an agent more prone to exploration when it is less certain of future actions.

This can be represented as:

$$\theta \leftarrow \theta + \delta \nabla \ln \pi(A|S, \theta) + k * entropy \tag{4}$$

### Multi Agent RL

These frameworks can also be extended to a multi-agent reinforcement learning (MARL) technique. There are 3 main methods:

1. **Fully Decentralized** In this method, each agent has its own set of parameters and learns independently of the other agents. This allows the agents to learn in parallel and can speed up the training process, but it can also lead to suboptimal solutions if the agents do not communicate effectively with each other.
2. **Fully Centralized** In this method, all of the agents share a common set of parameters, which are updated based on the collective experiences of the agents. This allows the agents to learn from each other and improve their performance as a group.
3. **Centralized Training Decentralized Execution(CTDE)** There also is a centralized training and decentralized execution method of training where normally each agent is trained in a decentralized method and then the critic is trained in a centralized manner. This approach allows the agents to benefit from the shared knowledge learned during training, while still maintaining the flexibility to adapt to changes in the environment at runtime.

Overall, the most effective method for training multi-agent systems will depend on the specific application and the goals of the system. It is important to carefully consider the trade-offs between individual learning (Lyu et al. 2021) and collaborative learning in order to find the best approach for a given situation.

In this paper we primarily explore Fully Centralized training and execution for cooperative tasks in Option-Critic and Actor-Critic frameworks. However, we also show that fully decentralized training in option critic does not appear to converge or work as well as

## Related Work

There have been related works in the multi-agent field, however, all of them use the CTDE training method and hence cannot be directly compared to a fully centralized method.

### Multi Agent Deep Deterministic Policy Gradients

MADDPG (Lowe et al. 2017) was introduced along with CTDE for multi-agent cooperative and competitive environments and showed a significant improvement over the existing methods like DDPG, AC, DQN. It used a centralized critic and separate actor policies to execute independently.

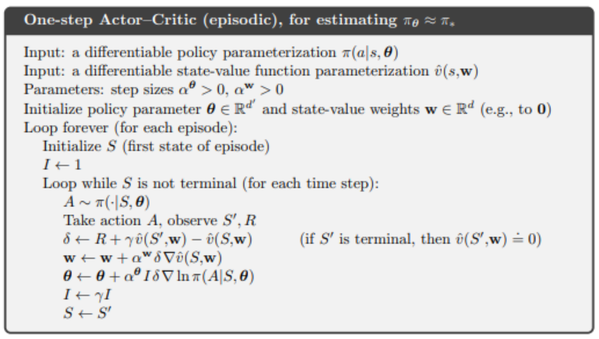
## Distributed Option Critic

Recent work has been made in using CTDE methods along with the notion of beliefs and information broadcasting to solve the decentralized POMDP problem . They propose a Distributed Option Critic (DOC) (Chakravorty et al. 2019) algorithm which uses centralized option evaluation and decentralized intra-option improvement. This outperforms all the single-agent counterparts like a2c, ppo and oc. It also outperforms a centralized version of a2c. However, due to the complexity and high variance of this method it is not always practical to use in most multi-agent scenarios.

## Project Description

We initially built a TD(0) Advantage actor-critic algorithm, as described in the below pseudocode in Figure 1. Seeing a failure of convergence, we also then built TD(N) versions of the algorithm where the TD error was found with a sum of discounted n rewards rather than just the current reward and the next state used as the baseline was n steps later instead of just 1 step later. This is described in equations 5 and 6.

## Advantage Actor Critic



```
One-step Actor–Critic (episodic), for estimating $\pi_\theta \approx \pi_*$

Input: a differentiable policy parameterization $\pi(a|s,\boldsymbol{\theta})$
Input: a differentiable state-value function parameterization $\hat{v}(s,\mathbf{w})$
Parameters: step sizes $\alpha^{\boldsymbol{\theta}} > 0$, $\alpha^{\mathbf{w}} > 0$
Initialize policy parameter $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ and state-value weights $\mathbf{w} \in \mathbb{R}^d$ (e.g., to $\mathbf{0}$)
Loop forever (for each episode):
    Initialize $S$ (first state of episode)
    $I \leftarrow 1$
    Loop while $S$ is not terminal (for each time step):
        $A \sim \pi(\cdot|S,\boldsymbol{\theta})$
        Take action $A$, observe $S', R$
        $\delta \leftarrow R + \gamma\hat{v}(S',\mathbf{w}) - \hat{v}(S,\mathbf{w})$        (if $S'$ is terminal, then $\hat{v}(S',\mathbf{w}) \doteq 0$)
        $\mathbf{w} \leftarrow \mathbf{w} + \alpha^{\mathbf{w}}\delta\nabla\hat{v}(S,\mathbf{w})$
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha^{\boldsymbol{\theta}} I\delta\nabla\ln\pi(A|S,\boldsymbol{\theta})$
        $I \leftarrow \gamma I$
        $S \leftarrow S'$
```

Figure 1: The pseudocode for TD(0) actor-critic. Pseudocode and actor-critic implimentation from (Sutton and Barto 2018)

$$\delta = G + \gamma V^{\pi_\theta}(s_{t+n}) - V^{\pi_\theta}(s_t) \tag{5}$$

Where t is the timestep of the state and n is the number of steps between updates and G is the discounted sum of rewards per for the last n time-steps.

$$G = \sum_{i=0}^{n} \gamma^i * r \tag{6}$$

We then also made a soft actor-critic implementation as described in equations 3 and 4. We then tested all of these algorithms in both the four rooms' single agent environment and in the simple spread environment varying these hyperparameters.

## Option Critic

Temporal abstraction allows representing knowledge about courses of action that take place at different time scales. In reinforcement learning, options (Sutton, Precup, and Singh 1999; Precup 2000) provide a framework for defining such courses of action and for seamlessly learning and planning with them.

The options framework formalizes the idea of temporally extended actions. A Markovian option $\omega \in \Omega$ is a triple $(I_\omega$ , $\pi_\omega$ , $\beta_\omega)$ in which $I_\omega \subseteq S$ is an initiation set, $\pi_\omega$ is an intra-option policy, and $\beta_\omega : S \rightarrow [0, 1]$ is a termination function. We also assume that $\forall s \in S, \forall\omega \in \Omega : s \in I_\omega$ (i.e., all options are available everywhere), an assumption made in the majority of option discovery algorithms.

Now using this framework we can define our objective for policy gradient methods as the discounted return $\mathbf{E}_{\Omega,\theta,\omega}[\sum_{t=0}^{\infty} \gamma^t r_{t+1}|s_0, \omega_0]$. Where $s_0$ and $\omega_0$ are the initial state and option. Option value function can be written as

$Q_\Omega(s, \omega) = \sum_a \pi_{\omega,\theta}(a|s)Q_U(s, \omega, a),$

where $Q_U$ is the value of execution an action in that state-option pair.

$Q_U(s, \omega, a) = r(s, a) + \gamma \sum_{s'} P(s'|s, a)U(\omega, s')$

Where U is the value of executing $\omega$ upon entering state s' and is called the option value function upon arrival.

$U(\omega, s') = (1 - \beta_{\omega,\upsilon}(s')Q_\Omega(s', \omega)) + \beta_{\omega,\upsilon}V_\Omega(s')$

These functions depend on the parameters $\theta$ and $\upsilon$ and can be derived to the following gradients for policy gradient methods[2].

$\theta \leftarrow \theta + \alpha_\theta \frac{\delta log \pi_{\omega,\theta}(a|s)}{\delta\theta} Q_U(s, \omega, a)$

$\upsilon \leftarrow \upsilon + \alpha_\upsilon \frac{\delta\beta_{\omega,\upsilon}(s')}{\delta\upsilon}(Q_\Omega(s', \omega) - V_\Omega(s'))$

## Options with Deliberation Cost

A further improvement to option critic was proposed by adding a deliberation cost to choose options (Harb et al. 2017). These deliberation costs can be modeled in the loss function so that the agent is incentivised to learn longer options and only switch options at difficult decision points.

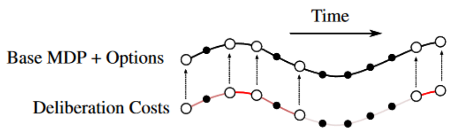


Figure 2: A deliberation cost is incurred upon switching to a new option and is subtracted from the reward of the base MDP.

Options can be elongated by using a cost function which penalizes for frequent options switches. Since $\beta_\theta(s', \omega)$ is the mean of a Bernoulli random variable over the two possible outcomes, switching or continuing (1 or 0), the cost function corresponding to the switching event is $c_\theta(s', \omega) = \gamma\beta_\theta(s', \omega)$ (where $\gamma$ was added for mathematical convenience). When expanding the value function over the transformed reward for this choice of $c_\theta(s', o)$, we get:

$$Q_\theta(s,\omega) = \sum_a \pi_\theta(a|s,\omega)(r(s,a) + \gamma \sum_{s'} P(s'|s,a) \\ [Q_\theta(s',\omega) - \beta_\theta(s',\omega)A_\theta(s,\omega) + \eta)]) \tag{7}$$

Here $\eta$ is the scalar deliberation cost. The termination gradient then becomes

$$\upsilon \leftarrow \upsilon + \alpha_\upsilon \frac{\delta \beta_{\omega,\upsilon}(s')}{\delta \upsilon}(Q_\Omega(s',\omega) - V_\Omega(s') + \eta) \tag{8}$$

**Algorithm 1:** Option-critic with tabular intra-option Q-learning

```
$s \leftarrow s_0$
Choose $\omega$ according to an $\epsilon$-soft policy over options
  $\pi_\Omega(s)$
repeat
    Choose $a$ according to $\pi_{\omega,\theta}(a \mid s)$
    Take action $a$ in $s$, observe $s'$, $r$

    1. Options evaluation:
    $\delta \leftarrow r - Q_U(s,\omega,a)$
    if $s'$ is non-terminal then
        $\delta \leftarrow \delta + \gamma(1 - \beta_{\omega,\vartheta}(s'))Q_\Omega(s',\omega) +$
          $\gamma\beta_{\omega,\vartheta}(s') \max_{\bar\omega} Q_\Omega(s',\bar\omega)$
    end
    $Q_U(s,\omega,a) \leftarrow Q_U(s,\omega,a) + \alpha\delta$

    2. Options improvement:
    $\theta \leftarrow \theta + \alpha_\theta \frac{\partial \log \pi_{\omega,\theta}(a \mid s)}{\partial\theta} Q_U(s,\omega,a)$
    $\vartheta \leftarrow \vartheta - \alpha_\vartheta \frac{\partial \beta_{\omega,\vartheta}(s')}{\partial\vartheta} (Q_\Omega(s',\omega) - V_\Omega(s'))$

    if $\beta_{\omega,\vartheta}$ terminates in $s'$ then
        choose new $\omega$ according to $\epsilon$-soft($\pi_\Omega(s')$)
    $s \leftarrow s'$
until $s'$ is terminal
```

**Centralized Options** In the fully centralized setting all the agents have separate set of parameterized options, and terminations and each of these options have primitive polices. However these are all shared in a common network with knowledge of the entire state. Hence each agent makes an informed action with knowledge of the entire state space and hopefully also the other agents inferred actions.

Hence for all agents j:

$$\theta^j \leftarrow \theta^j + \alpha_\theta \frac{\delta log \pi^t_{\omega,\theta}(a|s)}{\delta\theta} Q_U(s,\omega,a) \tag{9}$$

$$\upsilon^j \leftarrow \upsilon^j + \alpha_\upsilon \frac{\delta \beta^j_{\omega,\upsilon}(s')}{\delta\upsilon}(Q_\Omega(s',\omega) - V_\Omega(s')) \tag{10}$$

For all runs in the Four rooms (Huang et al. 2022) 4 options were chosen and for the Simple spread environment 5 options were chosen to maintain consistency across results.

## Experiments

We decided to compare our algorithms in two environments. One single agent environment and one multi-agent environment. The single agent environment is the four rooms environment.

### Four Rooms

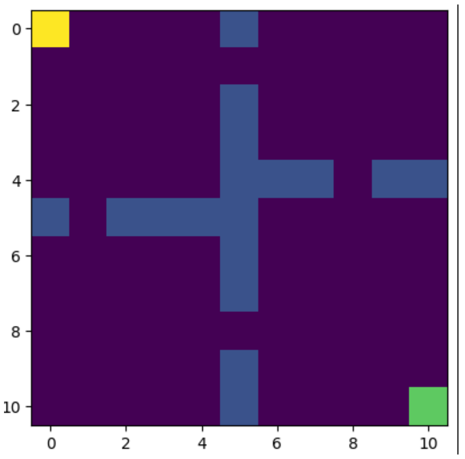


Figure 3: An image of the four rooms environment.

The agent must get from the starting point at position 0,0 to the ending position in position 10,10 without hitting any of the walls.

The multi-agent environment is the simple spread environment from MPE pettingzoo. N agents must work to be as close to N markers without touching each other as possible.

We did several experiments with actor-critic methods to see what version of actor-critic would be optimal. The first was TD(0) with and without entropy. Regardless of entropy, the agent initially found the exit randomly and updated value functions for the goal state in a way that should allow the agent to prioritize actions towards the goal state.

Unfortunately, in later episodes, the agent randomly finds its way back into one of the top two rooms. This triggers updates associated with the value of the transition squares between rooms to be extremely negative. In turn, the agent then never transitions into rooms that actually would take the agent back to the goal state. This value persists between episodes until the agent just creates a uniform distribution of mediocre weights in the two above rooms as shown in figure 4.

This prompted building an n-step version of actor-critic update. This also generates values for episode length and discounted returns that clearly show convergence.

From both of the charts in figures 5 and 6 it is clear the A2C algorithm converges to the optimal distance path of 20 steps in approximately 500 episodes.

We can see from the state value heat map that OC learns a pretty good approximation of the true state values in four rooms environment but only for the path that it explores. The bottom left room has not been sufficiently explored and hence does not have a good approximation of the state value.

The expected behavior was for the agent to pick 1 option per room and learn to navigate that room in that option. However, it looks like the agent has picked some arbitrary options at each stage of the path. It can be seen that the simple straight-line paths follow the same option however the options change frequently at decision boundaries like the

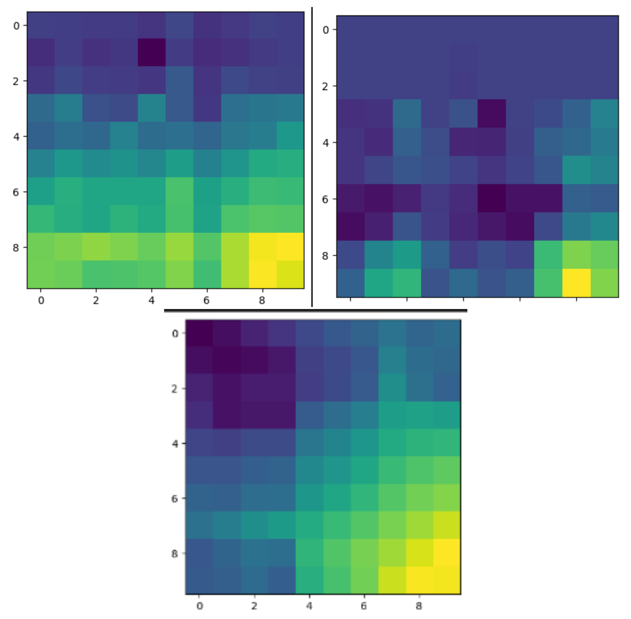


Figure 4: Four Rooms Environment TD(0) Value Network Heat map after 140 episodes, and after 900 episodes. Then the TD(5) Step value map after 950 episodes. This shows the value network failing in TD(0) and succeeding in TD(5)

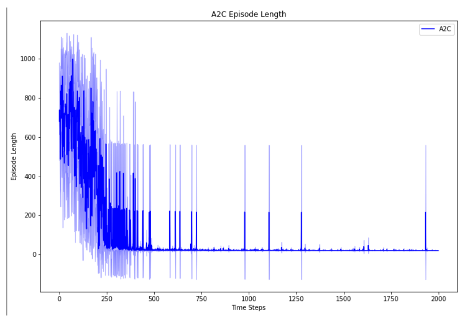


Figure 5: Four Rooms Environment Episode length per episode with 5 steps.

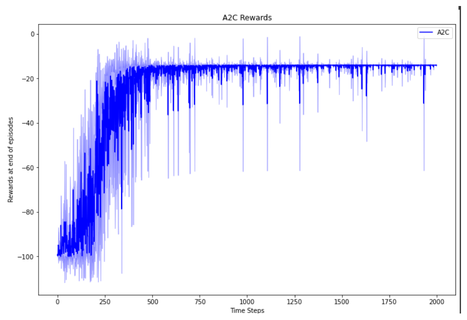


Figure 6: Four Rooms Environment discounted returns per episode with 5 steps.

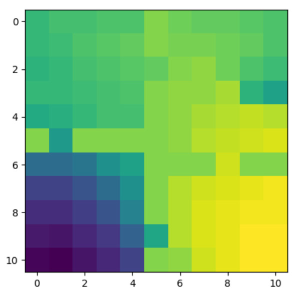


Figure 7: State Values of OC after 2000 episodes

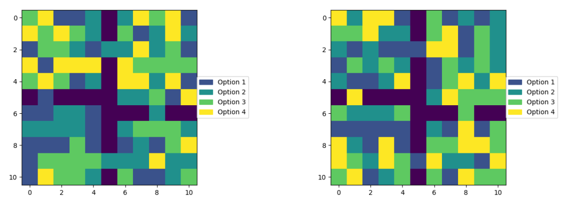


Figure 8: Options chosen at each state during rollout for 0.01 and 0.5 deliberation cost

walls of the rooms. Also the deliberation cost does not seem to change option lengths by a significant amount and only has a slight effect on longer options.

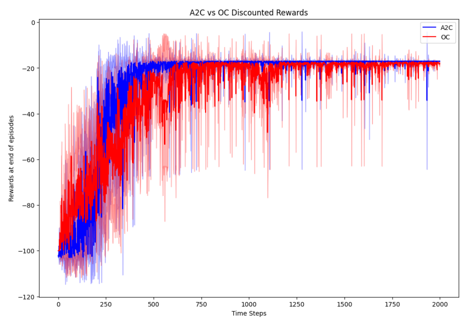


Figure 9: SAC with 5 step update compared to Option Critic Discounted Episode Rewards

When we compare our episode lengths and discounted rewards in the simple spread domain. and soft actor-critic with 5 steps we find that soft actor-critic does appear to converge faster. This makes sense as the environment is very simple and option critic has substantially more parameters to learn.

## Simple Spread

In the simple spread domain (Terry et al. 2020), we wanted to again be able to compare the best version of A2C with the best version of option critic. To do this we optimized several hyperparameters for advantage and option-critic so we could compare the best versions of both algorithms.

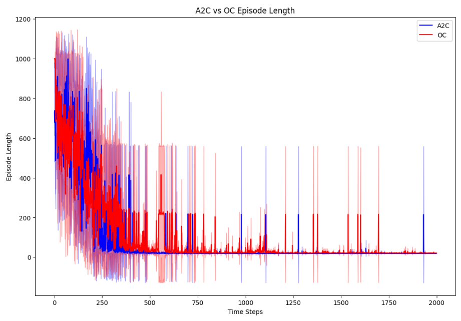


Figure 10: SAC with 5-step update compared to Option Critic episode length

In A2C we looked at n-step, soft actor-critic, and the number of agents to see what performs best and how scaling to large multi-agent environments with centralized training and execution impacts results.

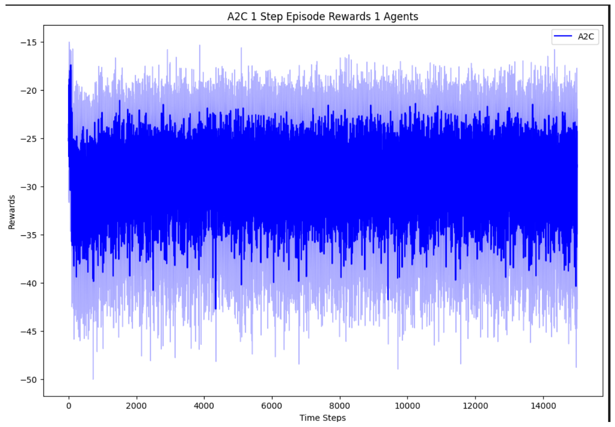


Figure 11: A2C Simple Spread with 1 step and 1 agent discounted rewards per episode

In A2C with one step and 1 agent in the simple spread domain, we found the rewards were consistently worse than the initial random policy. This is because the initial loss is extremely high as the domain is stochastic and unpredictable. However, the agents quickly learn that they can manipulate the markers to the center of the state space by moving quickly. Then the agents can move to the edge of the board and the expected value of the loss is very well known. This effectively minimizes the critic loss even if the actual reward function fails to be maximized. In theory, having an environment where losses are minimized will allow the agent to maximize reward in the long run as it will be able to make reliably better choices. We do later see this with Soft-actor-critic (SAC)

We then attempted to learn with 2 agents and various step sizes. For 2 agents, a step size of 4 appears to allow the most robust learning. It has the steepest slope and the highest reward value after 25,000 episodes. It also appears to look like it will be able to keep learning after this point where that is not immediately obvious in the other environments.

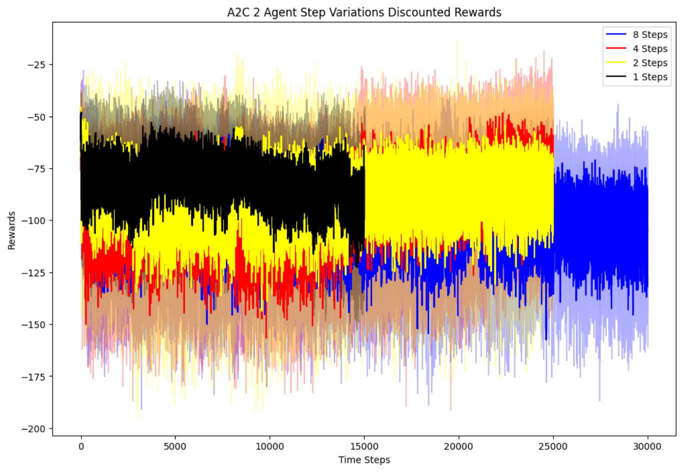


Figure 12: Comparing 1, 2, 4, and 8 steps in n step A2C in the simple spread domain. A step size of 4 appears to be optimal followed closely by a step size of 2.

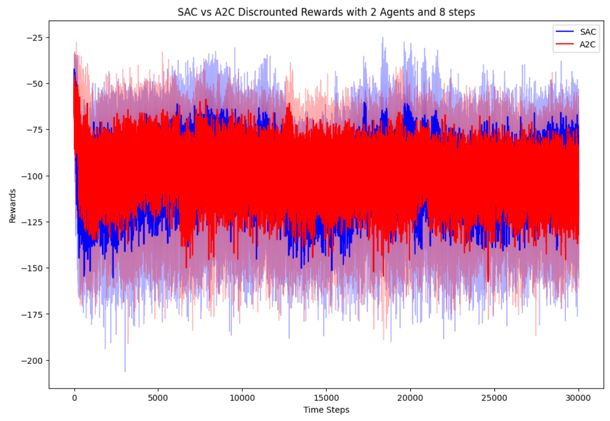


Figure 13: SAC compared to A2C Simple Spread with 8 steps and 2 agents discounted rewards per episode

We then compared the 8-step results to a soft actor-critic implementation. See Figure 13. Neither seemed to be particularly vastly better than the other. This may be because the step size of 8 was found to be two high in figure 12.

We then compared SAC and A2C again with 4 steps and 3 agents. See figure 14. We found that SAC does consistently outperform A2C when we have three agents and 4 steps.

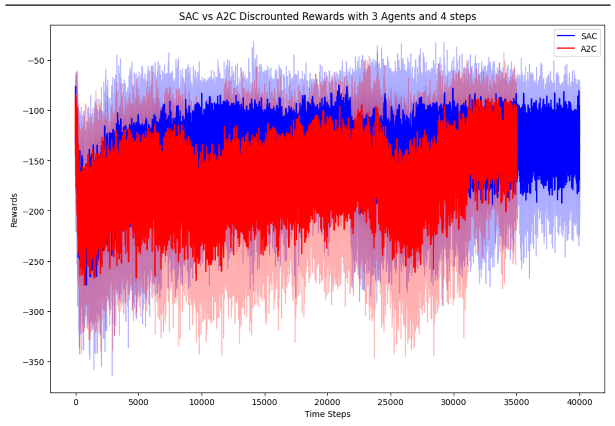


Figure 14: SAC compared to A2C with 3 agents and 4 steps. SAC consistently outperforms A2C.

As previously mentioned, few of these implementations outperform the original random state. To expand on that below is a series of figures showing what the three-agent case does for some of the advantage actor-critic implementations.

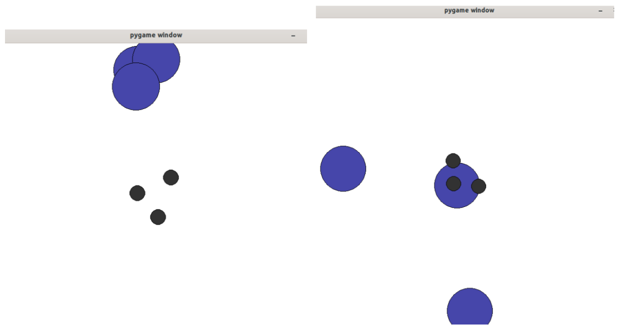


Figure 15: (left)A2C 3 Agents Corralling markers, but colliding. (right)A2C 3 Agents Corralling markers and not colliding. Agents successfully separate themselves with one agent on the marker, and two others on either the x or y plane of the marker. Earlier in the training the agents learn they can move quickly to push the markers into the center. Later, they learn they can then have one agent sit on top of the markers, the others sit in a safe space to the side.

It appears that the A2C algorithm finds a good strategy quickly. Then that strategy without entropy doesn't change for thousands of episodes, despite this strategy being found after approximately 20 episodes. Rollouts were not observed for SAC or for thousands of episodes due to the extra training time associated with visualization, but it is projected that with entropy and enough steps the agents would eventually break out of this pattern. It is presumed, that is exactly what is happening in the 4-step chart that is slowly learning over thousands of episodes.

We tested many training methods and hyperparameters for option critic. One of which was decentralized training. Because there are no shared weights or shared internal representation, we expected decentralized training to do very poorly.

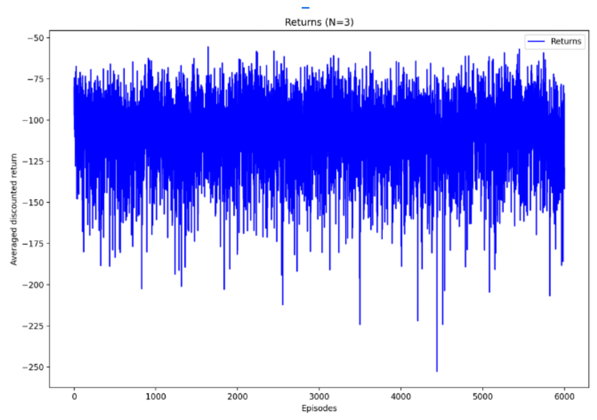


Figure 16: Fully Decentralized Returns of Option Critc

We were correct as Figure 16 suggests very poor learning and performance for the decentralized training method.

We then compared the best version of SAC with the best version of option critic for each number of agents.

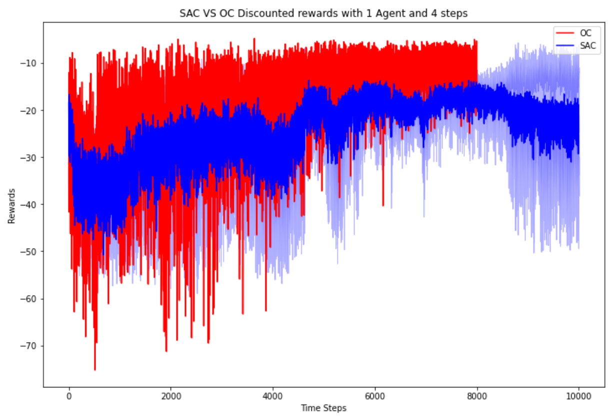


Figure 17: 4 Step SAC vs Option Critic in the 1 agent Simple Spread Environment. Option critic appears to do better, but both appear to learn effectively.

We see that in the single and double-agent environments option critic appears to be able to learn faster than actor-critic. This suggests that the hierarchies learned in option critic due appear to lend some value as long as the environment isn't so complicated that not even option critic can learn meaningfully in it.

This still leaves the question up for debate as it is possible the algorithm may still be learning at the end of the shown episodes, so we ran SAC for another 100k episodes without averaging it with any other runs. See figure 20. That did not make any substantive improvements with additional training. This suggests that these results are unlikely to change with additional training, all else being equal.

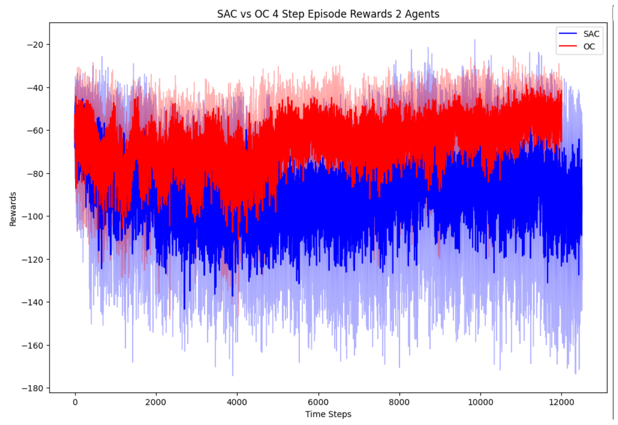


Figure 18: 4 Step SAC vs Option Critic in the 2 agent Simple Spread Environment. In the two agent environment, we can see option critic significantly outperformed SAC.

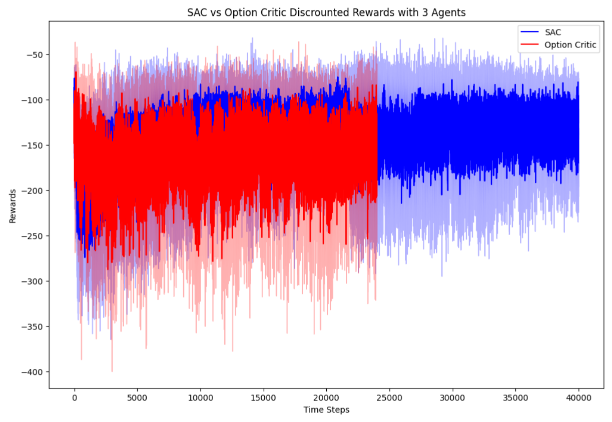


Figure 19: 4 Step SAC vs Option Critic in the 3 agent Simple Spread Environment. In the three-agent environment, it appears that neither algorithm was able to learn well enough to outperform a random strategy.

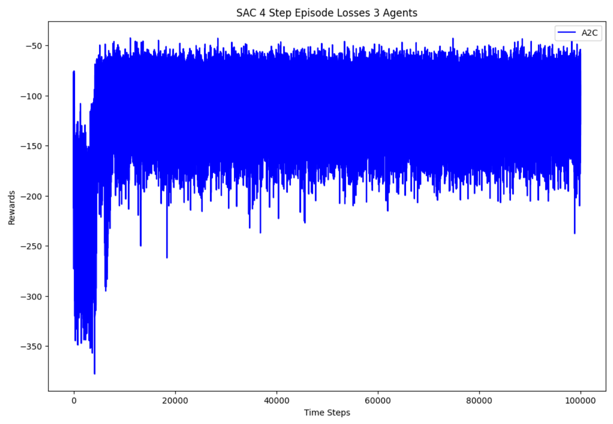


Figure 20: 4 Step SAC Trained for 100k episodes to see if it would additionally learn. It should be noted this is a single run, and not averaged with 2 or 4 other additional runs as all of our other results are.

## Conclusion

We see evidence that in simple domains like four rooms, the option-critic doesn't learn useful hierarchies and the Soft-Actor-Critic appears to be able to converge faster than it. However, in the simple spread environment we see option critic vastly outperform Soft-Actor Critic in single and two agent domains. We believe that the state space was too complicated for soft actor critic to meaningfully learn in. However, the three agent space was too complicated for option critic to learn in either. Therefore, it seems that neither algorithm could do better than random in that complicated environment. We believe that by scaling up the networks these environments could likely be learned by either algorithm eventually.

## References

Bacon, P.-L.; Harb, J.; and Precup, D. 2016. The option-critic architecture.

Chakravorty, J.; Ward, N.; Roy, J.; Chevalier-Boisvert, M.; Basu, S.; Lupu, A.; and Precup, D. 2019. Option-critic in cooperative multi-agent systems.

Flet-Berliac, Y. 2019. The promise of hierarchical reinforcement learning. *The Gradient*.

Haarnoja, T.; Zhou, A.; Abbeel, P.; and Levine, S. 2018. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR* abs/1801.01290.

Harb, J.; Bacon, P.-L.; Klissarov, M.; and Precup, D. 2017. When waiting is not an option : Learning options with a deliberation cost.

Huang, S.; Dossa, R. F. J.; Ye, C.; Braga, J.; Chakraborty, D.; Mehta, K.; and Araújo, J. G. 2022. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research* 23(274):1–18.

Lowe, R.; Wu, Y.; Tamar, A.; Harb, J.; Abbeel, P.; and Mordatch, I. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments.

Lyu, X.; Xiao, Y.; Daley, B.; and Amato, C. 2021. Contrasting centralized and decentralized critics in multi-agent reinforcement learning.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; and Riedmiller, M. 2013. Playing atari with deep reinforcement learning. cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013.

Silver, D.; Huang, A.; Maddison, C. J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; Dieleman, S.; Grewe, D.; Nham, J.; Kalchbrenner, N.; Sutskever, I.; Lillicrap, T.; Leach, M.; Kavukcuoglu, K.; Graepel, T.; and Hassabis, D. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529(7587):484–489.

Sutton, R. S., and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. The MIT Press, second edition.

Terry, J. K.; Black, B.; Grammel, N.; Jayakumar, M.; Hari, A.; Sulivan, R.; Santos, L.; Perez, R.; Horsch, C.; Dieffendahl, C.; Williams, N. L.; Lokesh, Y.; Sullivan, R.; and Ravi, P. 2020. Pettingzoo: Gym for multi-agent reinforcement learning. *arXiv preprint arXiv:2009.14471*.