A New Approach to Reinforcement Learning for Sequential Robotic Tasks using a Chained Options Model and Subtask-Focused Rewards

by

Somesh Daga

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2021

© Somesh Daga 2021

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Reinforcement Learning for Robotics is a trending area of research with tremendous potential for widescale industry adoption. To its detriment, large amounts of environmental interactions are typically required by robotic agents to discover good behaviours. In response, Hierarchical Reinforcement Learning methods are gaining traction and have demonstrated improved learning efficiencies through employing abstractions in the learning process. Additionally, implications on safety arising from *black-box* agents operating in physical environments, has generated interest in exploring explainable forms of learning.

In this thesis, we leverage a popular form of Hierarchical Reinforcement Learning, known as the *Options Framework*, to address learning for tasks that may be expressed as a sequential composition of subtasks. This form of task decomposition is prevalent in classical approaches to many robotic planning and control applications, and offers an avenue to segment tasks into sets of distinct and interpretable behaviors.

As our primary contribution, we propose a novel, potential-based reward formulation and decomposition, that is conducive to subtask behavior specialization and incentivizes a learning agent to solve the composite task, under the Options Framework. As a result, we offer increased visibility into the actions of the agent at the subtask level. An off-policy Maximum Entropy Deep Reinforcement Learning algorithm is developed to simultaneously discover relevant policies across subtasks and determine when to transition between subtasks in an end-to-end learning scheme. Furthermore, we propose a chained option execution model to leverage expert knowledge of the task and promote stability in the learning of subtask transitions. Finally, segmenting agent behaviors at the subtask level allows for the injection of expert knowledge into the action spaces of individual subtasks, which we exploit through the use of *default actions*.

We demonstrate the results of our work on high-dimensional, simulated 2D and 3D manipulator environments, for the tasks of pick-and-place and opening a door.

Acknowledgements

I would like to extend my thanks and appreciation to my supervisors, Prof. Soo Jeon and Prof. William Melek, for their patience and guidance through my masters program.

Also, a great shoutout to all my lab mates, from whom I've learnt a lot and whose support has been invaluable during my time at the University of Waterloo.

Special thanks to my colleague half way across the world, Jeonghoon Lee, for a series of insightful discussions into reinforcement learning and my research.

Many thanks to my family and friends for their continued love and support that kept me going in these trying times.

Lastly and most importantly, I acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC) and the University of Waterloo, without whom, none of this would have been possible.

Dedication

This thesis is dedicated to my family, whose unconditional love and support has gotten me to where I am. Also, a special dedication to a new biological reinforcement learning agent in the family, my nephew, who in his first year of existence has significantly outpaced the results of my work.

Table of Contents

List of Figures ix

List of Tables x

List of Abbreviations xii

Nomenclature xiii

1 Introduction 1

1.1 Contributions 2

1.2 Thesis Organization 3

2 Background 5

2.1 Reinforcement Learning 5

2.1.1 Markov Decision Process 5

2.1.2 Notations and Terminology 7

2.1.3 Bellman Equations 7

2.1.4 Temporal Difference Learning 9

2.1.5 Policy Gradient Theorem 10

2.1.6 On-Policy vs Off-Policy RL 11

2.1.7 Exploration vs Exploitation 11

2.2 Deep Reinforcement Learning 12

2.2.1 Proximal Policy Optimization (PPO) 12
2.2.2 Soft Actor-Critic (SAC) 13
2.3 Rewards 14
2.3.1 Potential-Based Reward Shaping 16
2.3.2 Count-Based Exploration Rewards 17
2.4 Options Framework 18
2.5 Option-Critic Architecture 20
2.5.1 Intra-Option Policy Gradient Theorem 21
2.5.2 Termination Gradient Theorem 21
2.5.3 Algorithm 22
2.5.4 Option Degeneracy 22

3 Sequential Task Learning **25**
3.1 Problem Setting 26
3.2 Options Chain Model 26
3.3 Reward Formulation 29
3.4 Default Actions 36
3.5 Entropy Regularization of Termination Conditions 36
3.6 Sequential Soft Option Actor-Critic (SeqSOAC) Algorithm 37
3.7 Relevant Works 40
3.8 Final Remarks 44

4 Experiments **45**
4.1 DeepMind Control Suite Manipulator 47
4.1.1 Subtask and Reward Formulation 48
4.1.2 Default Actions 49
4.1.3 State Representation and Abstractions 49
4.1.4 Hyperparameters 50

4.1.5 Results . . . 52
4.2 Door Opening Task . . . 58
4.2.1 Subtask and Reward Formulation . . . 59
4.2.2 Default Actions . . . 60
4.2.3 State Representation and Abstractions . . . 60
4.2.4 Hyperparameters . . . 63
4.2.5 Results . . . 64
4.3 Discussion . . . 65

5 Conclusion **70**
5.1 Summary . . . 70
5.2 Future Work . . . 71

References **72**

List of Figures

2.1 Agent-Environment Interaction in a Markov Decision Process. Image obtained from [45] 6
2.2 Options Semi-Markov Decision Process 19

3.1 Options Chain Architecture for Sequential Task Learning 27
3.2 Sequential Task Example 33

4.1 Manipulator Environment with bring_ball task from the DeepMind Control Suite [50] 47
4.2 Training Plot of Maximum Task Potentials achieved by the Options Chain, Option-Critic and SAC Agents for the 2D Manipulation Task 55
4.3 Training Plot of Rewards achieved by the Options Chain and Option-Critic Architectures for the 2D Manipulation Task 56
4.4 Evaluation of Behavior Specialization for 2D Manipulation Task 57
4.5 Visualization of Trained Agent Behavior for 2D Manipulator 58
4.6 Door environment from the robosuite [55] framework 58
4.7 Training Plot of Maximum Task Potentials achieved by the Options Chain, Option-Critic and SAC Agents for the Door Opening Task 65
4.8 Training Plot of Rewards achieved by the Options Chain and Option-Critic Architectures for the Door Opening Task 66
4.9 Evaluation of Behavior Specialization for the Door Opening Task 67
4.10 Visualization of Trained Agent Behavior for Door Opening Task 68

List of Tables

2.1 Pros and Cons of Sparse and Dense Reward Functions 16

3.1 Sequential Task Example: Scenario 1 31

3.2 Sequential Task Example: Scenario 2 31

4.1 State Space for *bring_ball* task of the *manipulator* domain from the DeepMind Control Suite [50] 50

4.2 State-Space Abstractions for Actors in 2D Manipulator Task 51

4.3 State-Space Abstractions for Critics in 2D Manipulator Task 51

4.4 State-Space Abstractions for Option Termination Conditions in 2D Manipulator Task 52

4.5 Hyperparameters for Actors in 2D Manipulator Task 52

4.6 Hyperparameters for Critics in 2D Manipulator Task 53

4.7 Hyperparameters for Termination Conditions in 2D Manipulator Task . . . 53

4.8 Hyperparameters for SAC Actor in 2D Manipulator Task 53

4.9 Hyperparameters for SAC Critic in 2D Manipulator Task 53

4.10 State Space for Door Opening Task from Robosuite [55] 61

4.11 State-Space Abstractions for Actors in Door Opening Task 61

4.12 State-Space Abstractions for Critics in Door Opening Task 62

4.13 State-Space Abstractions for Termination Conditions in Door Opening Task 62

4.14 Hyperparameters for Actors in Door Opening Task 63

4.15 Hyperparameters for Critics in Door Opening Task 63

4.16 Hyperparameters for Termination Conditions in Door Opening Task 63
4.17 Hyperparameters for SAC Actor in Door Opening Task 64
4.18 Hyperparameters for SAC Critic in Door Opening Task 64

List of Abbreviations

DRL	Deep Reinforcement Learning
HRL	Hierarchical Reinforcement Learning
MDP	Markov Decision Process
PBRS	Potential-Based Reward Shaping [35]
PPO	Proximal Policy Optimization [42]
RL	Reinforcement Learning
SAC	Soft Actor-Critic [18]
XAI	Explainable Artificial Intelligence
XRL	Explainable Reinforcement Learning

Nomenclature

α_ω	Entropy Coefficient for (Action) Policy of option ω
ξ	Scaling factor for count-based reward
γ	Discount Factor
$\mathcal{H}(\cdot)$	Entropy
$g(\cdot)$	Gating Function
$\pi_\omega(\cdot)$	(Action) Policy for option ω
Ω	A set of options
$\beta_\omega(\cdot)$	Termination Condition for option ω
$J(\cdot)$	Performance measure/Optimization Objective
$\pi_\Omega(\cdot)$	Policy over options
$\Phi(\cdot)$	Potential Function
ρ	Soft update parameter for target critic networks
$R_\omega(\cdot)$	Reward Function for subtask corresponding to option ω
$R(\cdot)$	(Task) Reward Function
κ_ω	Entropy Coefficient for Termination Condition of option ω

Chapter 1

Introduction

The potential for the application of robots to a wide range of domestic and industrial tasks, has attracted much research interest. Thus far, a great amount of success has been observed for the use of robots in structured environments, such as assembly lines for automotive manufacturing. However, progression to unstructured environments has seen relatively slow growth. Classical approaches to control and planning in robotics often face limitations in generalizing to diverse and complex task conditions, and have created a need for data-driven approaches to address these challenges.

Reinforcement Learning (RL), representing a branch of machine learning, provides a framework for solving sequential decision making tasks. Under this framework, agents interact with their environment(s) to learn and exploit good behaviours in order to maximize cumulative *rewards*. Learning from data curtails the need to pre-program complex behaviors and offers opportunities for life-long learning. In recent years, Deep Reinforcement Learning (DRL) has proved to be a promising avenue for learning complex behaviours from rich sensory inputs, through leveraging the expressive power of neural networks.

However, the promise of Reinforcement Learning is burdened by its own set of challenges. Sample Complexity (or Sample Efficiency), in particular, referring to the amount of interactions needed by an agent to discover behaviors in order to reliably carry out a task or set of tasks, is of chief concern. Robotic tasks are especially typified by high-dimensional state representations in continuous space, which encumbers efforts to distill relevant information from data. In contrast to other popular RL applications such as games, the results of simulated learning may not easily transfer to the real world, necessitating learning with hardware platforms. This imputes higher importance to the issue of sample complexity for multiple reasons. Firstly, real world learning may be magnitudes slower in training

time as compared to simulated settings which can benefit from greatly accelerated environment dynamics. Additionally, robot interaction with the environment exposes hardware to increased potential for damage and deterioration. Hierarchical Reinforcement Learning (HRL), advocated as a means of tackling the sample complexity issue, attempts to realize learning efficiencies and better generalization by employing mechanisms for abstractions in the learning process.

Many robotics applications, particularly in the domain of robotic manipulation, can be represented as sequential compositions of subtasks. Classical approaches to manipulation commonly leverage this structure to address such tasks. However, few works in Reinforcement Learning exploit abstraction at the subtask level, and fewer have attempted to learn and chain subtasks in an end-to-end learning scheme.

Explainable Artificial Intelligence (XAI) and Explainable Reinforcement Learning (XRL) [37, 14] have been gaining prominence in research, with the aim of affording greater insight and transparency into the decisions taken by learning agents. This is seen to be especially important in applications with implications on safety, in particular, with robots operating in real environments.

In this thesis, we present a framework for simultaneously learning appropriate behaviors across a set of pre-defined subtasks and when to transition between them, in order to solve robotic tasks which can be represented as a sequential composition of these subtasks. With the learning of visually meaningful and interpretable subtasks, we take a step towards fostering explainability in reinforcement learning for robotics. We leverage a popular HRL model, known as the Options Framework [47, 36], for the foundation of our work.

1.1 Contributions

Our primary contribution is the formulation of a novel, potential-based reward scheme to enable simultaneous learning of distinct subtask policies and transition policies between subtasks, under the Options Framework [47]. To the best of our knowledge, this is the first work to achieve end-to-end learning for robotic manipulation under an options-based method, without the need for human demonstrations. Our contributions can be summarized as follows:

- A novel, potential-based reward formulation and decomposition to enable cooperative and distinct behavior learning across subtasks
- A chained options execution model for improved behavior specialization

- Use of *default* actions to inject expert knowledge of subtask action spaces into the learning process and reduce the burden of exploration
- Options-based adaptation of the off-policy Soft Actor-Critic [18] (SAC) Deep RL Algorithm with demonstrated learning for high-dimensional robotics tasks

1.2 Thesis Organization

In this thesis, we draw on developments and key observations from a number of prior works targetting different aspects of the RL problem. The central works contributing to our approach have largely been adapted for our problem formulation and target applications, having previously been used in RL under different contexts.

To ensure a coherent layout of the subject matter with relevance to prior works being clearly highlighted, we organize this document as follows:

- In Chapter 2, we provide a brief review of the fundamentals of RL. We then proceed to discuss the central works surrounding our method. To this effect, we start with a review of the Proximal Policy Optimization [42] (PPO) and SAC [18] algorithms for Deep RL, two popular algorithms for on-policy and off-policy RL respectively. Next, we discuss the nature of reward functions commonly used in RL, and focus on a particular class of reward shaping known as Potential-Based Reward Shaping [35] (PBRS). Subsequently, we present an overview of a simple count-based exploration technique that allows for solving sparse reward problems. We conclude the chapter with a presentation of the Options Framework [47, 36], and the Option-Critic Architecture [4], the former representing a model for Hierarchical RL and the latter providing some key results for end-to-end learning under the Options Framework.
- In Chapter 3, we formalize our problem setting and propose an approach for constructing subtask-focused reward functions. Furthermore, we offer the use of *default actions* to reduce the burden of exploration through expert-specified abstractions of the action space for different subtasks. Additionally, we develop an off-policy learning algorithm that may be used under our framework. Lastly, an overview of related works is provided.
- In Chapter 4, we demonstrate the ability to learn under our framework through simulated experiments on 2D and 3D manipulation tasks. In particular, we showcase the ability of the agent to solve the required tasks with appropriate subtask specialization.

- In Chapter 5, we conclude this thesis with a summary of the proposed ideas- and results, and with a discussion of possibile avenues for future research.

Chapter 2

Background

In this chapter, we aim to set forth the fundamental knowledge that our work builds upon. We begin with a review of key concepts and terminology in Reinforcement Learning. Next, we provide a brief background on Deep RL, and outline the Proximal Policy Optimization (PPO) [42] and Soft Actor-Critic (SAC) [18] algorithms, representing popular choices for on-policy and off-policy Deep RL algorithms respectively. We then provide an overview of the types of reward functions commonly utilized in RL works, and focus on a particular form of reward shaping known as Potential-based Reward Shaping [35]. This is followed by a look at Count-Based Exploration Rewards [48], a simple mechanism to encourage exploration in sparse reward environments. Subsequently, we discuss the Options framework [47, 36], a Hierarchical RL extension to the standard RL framework that has observed considerable research in the last 20 years. Finally, we conclude the background material with an overview of the Option-Critic Architecture [4] and its key results.

2.1 Reinforcement Learning

2.1.1 Markov Decision Process

A Markov Decision Process (MDP) provides a framework for modelling sequential decision-making tasks, and in the context of RL, is often represented by the following elements:

- A set of states, $\mathcal{S}$
- A set of actions, $\mathcal{A}$ (optionally a function of state i.e. $\mathcal{A}(s)$, $\forall s \in \mathcal{S}$)

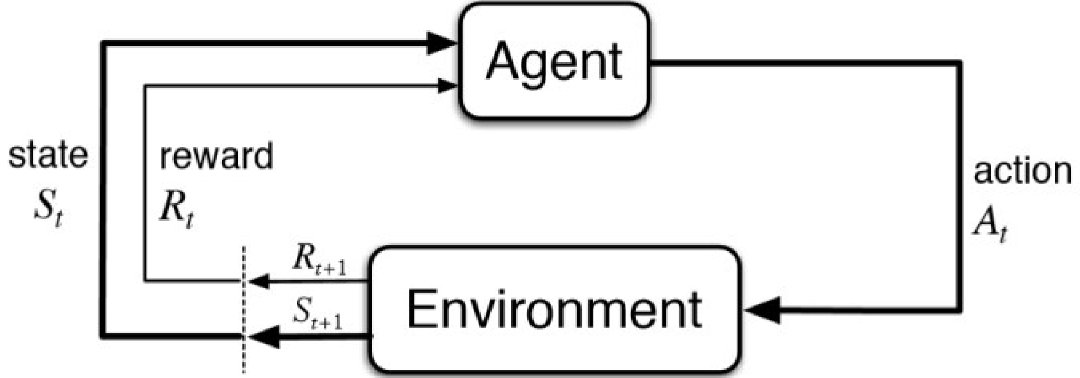


Figure 2.1: Agent-Environment Interaction in a Markov Decision Process. Image obtained from [45]

- Reward function, $r(s, a, s')$, where $r : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$
- Discount factor, $\gamma \in [0, 1)$, and,
- State Transition Probability Distribution, $p(s'|s, a)$, where $p : \mathcal{S} \times \mathcal{A} \rightarrow (\mathcal{S} \rightarrow [0, 1])$

Under the standard RL setting, the interaction between a decision-making *agent* and its *environment* under an MDP is illustrated in Figure 2.1. At each timestep, the agent receives the current state $S_t \in \mathcal{S}$ from the environment, chooses and executes an action $A_t \in \mathcal{A}$, resulting in a state transition to $S_{t+1} \in \mathcal{S}$ in accordance to the environment dynamics represented by the state transition probability distribution $p(S_{t+1}|S_t, A_t)$. The agent also receives a scalar reward $R_{t+1} \equiv r(S_t, A_t, S_{t+1})$. Note that state representations (for fully-observable environments) in $\mathcal{S}$ typically comprise of the state of the agent, the state of additional entities in the environment and any context information associated with the task.

Rewards serve as reinforcement signals, where the objective of the agent is to maximize the cumulative return (sum of rewards) obtained through interacting with the environment. The discount factor, γ, is typically employed to keep the sum of rewards finite and guarantee convergence of RL algorithms. Thus, the discounted return starting at timestep t, G_t, is given by:

$$G_t = \sum_{k=t+1}^{\infty} \gamma^{k-t-1} R_k \tag{2.1}$$

The sum to infinity represents the case for *continuing* tasks, where the agent continues to interact with the environment forever. In practice, it is common to work with *episodic* tasks, where the task terminates if the agent encounters certain *absorbing* states and/or a maximum time horizon is reached. Hence, the sum of rewards has a finite number of terms, up to the time when an episode ends.

2.1.2 Notations and Terminology

We are now ready to introduce a few key terms and concepts in Reinforcement Learning.

The objective of RL, previously defined in Section 2.1.1, can be restated as finding a state-dependent control law or *policy*, that if followed by the agent, maximizes its expected (discounted) return. A policy may be stochastic and represented by a state-conditioned probability distribution over actions, $\pi(a|s)$, $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$, or it may be a deterministic function of state, $a = \mu(s)$, $\mu : \mathcal{S} \to \mathcal{A}$. In this thesis, we are interested in the former class of policies and henceforth, we shall adopt the notation π to represent a policy.

A *Value Function*, $V_\pi(s)$, $V_\pi : \mathcal{S} \to \mathbb{R}$, seeks to estimate the expected return as a function of state, for an agent that that follows a policy π. A simple procedure for estimating the expected return for state s, is to average the returns (Equation (2.1)) starting in state s across multiple episode rollouts. Section 2.1.4 introduces some nuanced algorithms for value estimation that allow for bias-variance tradeoffs.

An *Action-Value Function*, $Q_\pi(s, a)$, $Q_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, may also be utilized to estimate the expected return for an agent in state s which takes a particular action a in the current step and subsequently acts according to π.

Lastly, an *Advantage Function*, $A_\pi(s, a)$, $A_\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, represents the gain in expected return afforded by taking the action a in the current state s and following the policy thereafter, as opposed to acting purely according to the policy. Hence, we have that $A_\pi(s, a) \equiv Q_\pi(s, a) - V_\pi(s)$.

2.1.3 Bellman Equations

The Bellman equations formalize the relationships between rewards, value and action-value functions, and policies. This forms the basis of *policy evaluation* in RL.

The *Bellman Expectation Equations* mathematically define $V_\pi(s)$ and $Q_\pi(s, a)$ as follows:

$$
\begin{aligned}
V_\pi(s) &= \mathbb{E}_\pi \left[R_{t+1} + \gamma V_\pi(S_{t+1}) \middle| S_t = s \right] \\
Q_\pi(s,a) &= \mathbb{E}_\pi \left[R_{t+1} + \gamma Q_\pi(S_{t+1}, A_{t+1}) \middle| S_t = s, A_t = a \right]
\end{aligned} \tag{2.2}
$$

These equations can be expanded to produce the following recursive formulas, with r representing the reward obtained for the state transition from state s to s' by applying action a:

$$
\begin{aligned}
V_\pi(s) &= \sum_{a \in \mathcal{A}} \pi(a|s) Q_\pi(s,a) \\
Q_\pi(s,a) &= \sum_{s' \in \mathcal{S}} p(s'|s,a) \left[r + \gamma V_\pi(s') \right] \\
&= \sum_{s' \in \mathcal{S}} p(s'|s,a) \left[r + \gamma \sum_{a' \in \mathcal{A}} \pi(a'|s') Q_\pi(s',a') \right] \\
V_\pi(s) &= \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} p(s'|s,a) \left[r + \gamma V_\pi(s') \right]
\end{aligned} \tag{2.3}
$$

Another set of equations, known as the *Bellman Optimality Equations*, seek to directly express the value and action-value functions under the optimal policy (represented by *):

$$
\begin{aligned}
V_*(s) &= \max_{a \in \mathcal{A}} Q_*(s,a) \\
Q_*(s,a) &= \sum_{s' \in \mathcal{S}} p(s'|s,a) \left[r + \gamma V_*(s') \right] \\
&= \sum_{s' \in \mathcal{S}} p(s'|s,a) \left[r + \gamma \max_{a' \in \mathcal{A}} Q_*(s',a') \right] \\
V_*(s) &= \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} p(s'|s,a) \left[r + \gamma V_*(s') \right]
\end{aligned} \tag{2.4}
$$

Note that the transition probabilities $p(s'|s,a)$ and rewards r are not (generally) explicitly known to the agent, but rather obtained as samples after each step of interaction with the environment. If both functions are known, it would be possible to solve the RL problem purely by dynamic programming without any environmental interactions. A body

of RL works dealing with *World Models* [17, 19] seek to do this by reserving environmental interactions for modelling transition dynamics and reward functions, and solving the rest of the RL problem offline. However, this is tangential to the work in this thesis. Regardless, the Bellman equations provide a means of estimating value and action-value functions in expectation, in the absence of direct knowledge of the transition probability distributions and reward functions, through Equation (2.2).

2.1.4 Temporal Difference Learning

Thus far, we have seen two methods of policy evaluation; the unbiased Monte-Carlo method of averaging returns calculated using Equation (2.1) following visits to states/state-action pairs, and the Bellman Expectation Equations (given by Equation (2.2)). The primary difference between these two schemes is that the latter truncates the sum of rewards by *bootstrapping* on current estimates of the value/action-value functions. Temporal Difference (TD) Learning allows us to vary the degree of truncation (via n-step TD methods), to achieve bias-variance tradeoffs. With a low degree of truncation, policy evaluation schemes are prone to large variance owing to stochasticity of policies and/or environment dynamics, coupled with long episode horizons. On the flip side, large degrees of truncation inject increasing bias into the policy evaluation scheme due to greater influence of bootstrapped estimates.

Temporal-Difference (TD) learning incrementally shifts value estimates towards *TD targets* formed by the bootstrapping. A n-step TD target for a value function, $V_\pi^{\text{target}}(S_t)$, is given by:

$$V_\pi^{\text{target}}(S_t) = \sum_{k=t+1}^{t+n} \gamma^{k-t-1} R_k + \gamma^n V_\pi(S_{t+n}) \tag{2.5}$$

A *TD error*, δ_t, represents the difference between the TD target and the current estimate of the value function:

$$\delta_t = V_\pi^{\text{target}}(S_t) - V_\pi(S_t) \tag{2.6}$$

With a learning rate α, the value function can be iteratively updated to reduce the TD error and converge to the true function:

$$V_\pi(s) \leftarrow V_\pi(s) + \alpha\delta_t \tag{2.7}$$

Note that action-value functions can also be estimated via TD learning in an analogous way. There exist a number of value estimation schemes utilizing n-step TD targets or a combination of different n-step TD targets, but we omit these details as they are not pertinent to the understanding of the materials in this thesis.

2.1.5 Policy Gradient Theorem

Thus far, we have seen how the performance of an agent, with a given policy, may be estimated by means of policy evaluation using Monte Carlo or Temporal Difference Learning methods. However, we also need to perform *policy improvement* steps to incrementally adjust the policy of the agent in learning to maximize returns. The iterative application of policy evaluation and policy improvement steps is referred to as *policy iteration*. The *Policy Gradient Theorem* [46] provides a method for policy improvement that directly optimizes for the expected return through stochastic gradient ascent. Parameterizing the policy π by θ and defining a performance measure $J(\theta)$, representing the expected return, as follows:

$$J(\theta) = \sum_{s_0 \in \mathcal{S}} \mu(s_0) V_\pi(s_0) \tag{2.8}$$

where $\mu(\cdot)$ represents the probability distribution over initial states, the key result of the policy gradient theorem [46, 45] is given by:

$$\nabla_\theta J(\theta) \propto \mathbb{E}_\pi \left[\nabla_\theta \ln \pi(a|s,\theta) Q_\pi(s,a) \right] \tag{2.9}$$

In policy gradient methods, this result is applied on state-action pairs generated by rolling out the policy π, starting at initial states generated according to μ. Importantly, this result is invariant to transformations of $Q_\pi(s,a)$ through addition/subtraction of *baseline* functions which are only a function of state. A common choice for this baseline function is $V_\pi(s)$, which is subtracted from $Q_\pi(s,a)$ to yield the advantage function $A_\pi(s,a)$. This reduces variance in learning with policy gradient methods.

A special case of the policy gradient theorem is realized by the REINFORCE [52] algorithm, which directly uses the observed returns G_t in place of $Q_\pi(s,a)$. However, it is common to maintain estimates of $Q_\pi(s,a)$ and/or $V_\pi(s)$ in learning with policy gradient methods. Such methods are known as *actor-critic* methods, where the *actor* and *critic* refer to the policy and value/action-value function(s) respectively.

2.1.6 On-Policy vs Off-Policy RL

In the taxonomy of RL algorithms, one can classify any algorithm as *on-policy* or *off-policy*.

On-policy methods require that any updates to the agent's current policy are derived from states and/or state-action pairs observed under that policy. Alternatively, if a different behaviour policy is utilized, deviations from the agent's current policy must be accounted for by *importance sampling*. In the former case, on-policy methods usually alternate between policy rollouts to generate samples, and training on the observed data. The samples generated from each set of rollouts are discarded after the training phase. The result of the policy gradient theorem given by Equation (2.9) is (theoretically) only valid for on-policy samples, although practical usage has yielded success with certain off-policy methods such as the Soft Actor-Critic (SAC) [18] algorithm.

Off-policy methods, on the contrary, do not have such stipulations and can (re-)use samples generated from previous versions of the agent's policy or a behavior policy that deviates from the agent's current policy, without importance sampling. A *replay buffer* is commonly used to store experience tuples, $(S_t, A_t, R_{t+1}, S_{t+1})$, for each step taken in the MDP. These experience tuples are sampled from the replay buffer to update the policy and/or value estimates. Off-policy methods typically exhibit lower sample-complexity as they can continue to reuse samples generated over the course of learning. However, theoretical guarantees of convergence are lost with off-policy methods, and training may become unstable. Off-policy learning represents one of the elements of the *Deadly Triad* [45], in addition to Bootstrapping and Function Approximation (using parameterized representations for policies, value functions etc), contributing to potential learning instability.

2.1.7 Exploration vs Exploitation

The exploration versus exploitation dilemma represents a key issue in RL. A learning agent needs to achieve a good balance between exploring the combined state-action space, and exploiting what it has previously learnt. Excessive exploration can lead to the agent taking too long to converge to a solution, whereas, too little exploration may result in locally optimal solutions or a failure to find any solutions at all. The latter applies especially to sparse reward tasks, where non-zero rewards are only encountered for a small subset of states (or state-action pairs).

There exist many mechanisms to induce exploration in RL. The stochasticity (or entropy) of a policy, in itself, provides a means of exploration. A policy with large entropy will act with more diversity, which will likely lead to visiting a more diverse set of states.

Hence, many algorithms under the umbrella of Maximum Entropy RL or Entropy Regularized RL, attempt to maximize a weighted sum of expected returns and policy entropy.

2.2 Deep Reinforcement Learning

Deep RL represents the intersection of reinforcement learning with deep learning. In Deep RL, some or all elements of RL e.g policies, value functions, state transition probability distributions etc., are represented using neural networks. Neural networks provide means of learning complex input-output relationships, and are especially useful in tasks with high-dimensional, continuous state and action spaces.

The use of parameterized representations in RL, with a finite number of parameters, is referred to as *function approximation*. As briefly mentioned in Section 2.1.6, the use of function approximation may contribute to learning instability. Over the years, there have been a number of advances aimed at mitigating the effects of the aforementioned entities constituting the Deadly Triad [45], resulting in tractable algorithms for Deep RL. Below, we review the Proximal Policy Optimization [42] and Soft Actor-Critic [18] Deep RL algorithms, two popular works for on-policy and off-policy learning respectively.

2.2.1 Proximal Policy Optimization (PPO)

PPO [42] is an on-policy, actor-critic method that maintains and updates the agent's policy $\pi(a|s)$ and value function(s) $V_\pi(s)$ over the course of learning. It directly optimizes for the expected return (and optionally the policy entropy) using a modified form of the policy gradient theorem (Equation (2.9)).

In order to promote learning stability, a number of algorithmic novelties are employed. Central to the success of the algorithm, is the use of an *approximate* trust-region formulation, to limit the degree by which a policy may change over each cycle of data collection and policy updates. This follows from a predecessor work, Trust Region Policy Optimization (TRPO) [41], which utilizes a second-order method to implement a strict trust-region setting, by imposing constraints on the Kullback-Leibler (KL) divergence between successive policy iterates. To the benefit of improved computational speed, PPO optimizes a *Clipped Surrogate Objective* [42], $L^{\text{CLIP}}(\theta)$, to achieve an approximate trust-region:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_{\pi_{\text{old}}}\left[\min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}A(s,a), \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}, 1+\epsilon, 1-\epsilon\right)A(s,a)\right)\right] \tag{2.10}$$

Note that $L^{\text{CLIP}}(\theta)$ shares a close form with $J(\theta)$ from Equation (2.9), in the absence of the `min` and `clip` operators and with the use of the advantage function in place of action-value function to reduce variance of the gradient estimates. The PPO method is presented in Algorithm 1.

Algorithm 1 Proximal Policy Optimization (PPO) ▷ Adapted from [1]

1: Input: initial policy parameters θ_0, initial value function parameters ϕ_0
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: Collect set of N trajectories $\mathcal{D}_k = \{\tau_i\}_{i=1}^N$ of length T, by running policy $\pi_k = \pi(\theta_k)$ in the environment
4: Compute rewards-to-go $\hat{R}_t$ ▷ Equation (2.1)
5: Compute advantage estimates $\hat{A}_t$ (using any method of advantage estimation) based on the current value function V_{ϕ_k}
6: Update the policy by maximizing the Clipped Surrogate Objective:

$$\theta_{k+1} = \max_{\theta} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t, \text{clip}\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1+\epsilon, 1-\epsilon\right)\hat{A}_t\right)$$

typically via stochastic gradient ascent.
7: Fit value function by regression on mean-squared error:

$$\phi_{k+1} = \arg\min_{\phi} \frac{1}{|\mathcal{D}_k|T} \sum_{\tau \in \mathcal{D}_k} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t\right)^2$$

typically via some gradient descent algorithm.
8: **end for**

2.2.2 Soft Actor-Critic (SAC)

SAC [18] is an off-policy, actor-critic method which iteratively updates the agent's policy $\pi(a|s)$ and action-value function(s) $Q(s,a)$. The foundation of the algorithm lies in Maximum Entropy Reinforcement Learning [56], where the objective of the RL problem is a combination of the expected return and policy entropy (with α as a *temperature* hyperparameter and $\mathcal{H}(\pi(\cdot|s))$ representing the entropy of the policy):

$$J(\pi) = \mathbb{E}_{\pi}\left[r(s,a) + \alpha\mathcal{H}(\pi(\cdot|s))\right] \tag{2.11}$$

In light of the Maximum Entropy RL formulation, a modified Bellman operator and policy iteration method (known as *Soft Policy Iteration*) are utilized. In essence, the value function is replaced by the *soft value function* in the policy evaluation procedure, where:

$$V_{\pi}(s) = \mathbb{E}_{a\sim\pi(\cdot|s)}\left[Q(s,a) - \alpha\log\pi(a|s)\right] \tag{2.12}$$

Hence, the Bellman Expectation Equation for action-value evaluation transforms as follows:

$$Q_{\pi}(s,a) = \mathbb{E}_{\pi}\left[R_{t+1} + \gamma\left(Q_{\pi}(S_{t+1},A_{t+1}) - \alpha\log\pi(A_t|S_t)\right)|S_t = s, A_t = a\right] \tag{2.13}$$

For the policy improvement step(s), the updated policy is derived by minimizing the KL Divergence between the exponentiated Q-function and parameterized policy:

$$\theta_{k+1} = \arg\min_{\theta} \mathrm{D}_{\mathrm{KL}}\left(\pi_{\theta}(\cdot|s) \,\middle\|\, \frac{\exp(Q_{\pi_{\theta_k}}(s,\cdot))}{Z_{\pi_{\theta_k}}(s)}\right) \tag{2.14}$$

where $Z_{\pi_{\theta_k}}(s)$ represents the partition function, $\int_{\mathcal{A}}\exp(Q_{\pi_{\theta_k}}(s,a))da$, to normalize the distribution.

Additionally, to foster stability for off-policy learning, a couple of algorithmic inventions from previous works are applied. Most notably, these include the use of *target networks* [33] for action-value estimation and, taking the minimum over two (or more) action-value networks to avoid value overestimation [15]. The complete algorithm for SAC is presented in Algorithm 2.

2.3 Rewards

We now shift our discussion to the nature of reward functions used in RL. Reward functions can generally be categorized into one of two classes; *Sparse* and *Dense* reward functions.

In sparse reward environments, the agent only receives non-zero rewards for a small subset of the state and/or state-action space. It is common to receive a non-zero reward only upon the completion of a task or on achieving particular milestones within a task.

Algorithm 2 Soft Actor-Critic (SAC) ▷ Adapted from [2]

1: Input: initial policy parameters θ, Q-function parameters ϕ_1, ϕ_2, empty replay buffer $\mathcal{D}$, Learning rates β (for Q-functions) and κ (for policy), Target update parameter ρ
2: Set target parameteres equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$
3: **repeat**
4: Observe state s and sample action $a \sim \pi_\theta(\cdot|s)$
5: Execute action a in the environment
6: Observe next state s', reward r, and done signal d to indicate whether s' is a terminal state
7: Store (s, a, r, s', d) in $\mathcal{D}$
8: If s' is terminal, reset environment state
9: **if** it's time to update **then**
10: **for** j in range(however many updates) **do**
11: Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
12: Compute targets for the Q functions:

$$y(r, s', d) = r + \gamma(1 - d)\left(\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s')\right), \quad \tilde{a}' \sim \pi_\theta(\cdot|s')$$

13: Update Q-functions by one step of gradient descent using

$$\phi_i \leftarrow \phi_i - \beta \nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d)\in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2 \quad \text{for } i\text{=1,2}$$

14: Update policy by one step of gradient ascent using

$$\theta \leftarrow \theta - \kappa \nabla_\theta \frac{1}{|B|} \sum_{s\in B} \left(\min_{i=1,2} Q_\phi(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta\left(\tilde{a}_\theta(s)|s\right)\right), \quad \tilde{a}_\theta(s) \sim \pi_\theta(\cdot|s)$$

15: Update target networks with

$$\phi_{\text{targ},i} \leftarrow \rho \phi_{\text{targ},i} + (1 - \rho)\phi_i \quad \text{for } i\text{=1,2}$$

16: **end for**
17: **end if**
18: **until** convergence

Reward Type	**Pros**	**Cons**
Sparse	• Easy to design • Low chance for reward-induced behaviour bias	• Lack of reward signals can make learning difficult
Dense	• Rich reward signals for easier learning	• Domain-level knowledge of task required for reward shaping • Potentially complex reward designs • Shaped rewards may induce behavior bias in learning

Table 2.1: Pros and Cons of Sparse and Dense Reward Functions

Dense reward functions, on the other hand, bestow the agent with non-zero rewards over a large portion of the state and/or state-action space. The act of designing these dense reward functions is referred to as *Reward Shaping*, typically done by an expert with domain-level knowledge of a task.

The pros and cons of each reward class is summarized in Table 2.1. Sample complexity can be drastically improved through the use of dense reward functions, however, reward shaping may be complex depending on the task and may inadvertently result in policy biases induced by the expert-designed reward functions.

2.3.1 Potential-Based Reward Shaping

Potential-Based Reward Shaping [35] is a special class of reward shaping that guarantees policy invariance under reward transformations. Hence, it offers a solution to transform from sparse to dense reward MDPs while ensuring the optimal policy remains unchanged, overcoming one of the typical pitfalls of reward shaping.

Formally stated, through potential-based reward shaping, we seek to transform the MDP $M = (\mathcal{S}, \mathcal{A}, r, \gamma, p)$ to $M' = (\mathcal{S}, \mathcal{A}, r', \gamma, p)$, where $r'(s, a, s') = r(s, a, s') + F(s, a, s')$, such that $\pi^*_{M'} = \pi^*_M$. $F(s, a, s')$ represents a potential-based shaping function based on a *difference in potentials*:

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s) \tag{2.15}$$

where $\Phi(s) : \mathcal{S} \to \mathbb{R}$ can be any function of state and is usually chosen based on the task. Considering the *undiscounted* variant of this shaping function, it is apparent that cyclic behaviour of the agent gives zero resultant reward i.e. for a state trajectory τ: $s_1 \to s_2 \to \ldots \to s_n \to s_1$, $\sum_\tau F(s, a, s') = \Phi(s_1) - \Phi(s_n) + \sum_{i=1}^{n-1} \Phi(s_{i+1}) - \Phi(s_i) = 0$, ensuring the agent cannot accrue positive rewards without making a net progress towards solving the task. Violation of this property, where cyclic behavior may produce positive returns, can lead to the agent learning unintended behaviours [39]. [35] offers support for the use of *undiscounted* shaping functions of the form $\Phi(s') - \Phi(s)$ even for $\gamma \neq 1$.

2.3.2 Count-Based Exploration Rewards

We turn our attention to look at a simple method to address exploration in sparse reward tasks.

A popular mechanism for inducing exploration in sparse reward MDPs is by augmenting the reward function with *exploration bonuses*, rewarding the agent for visits to novel states and/or states associated with large uncertainties with respect to particular estimated quantities. There are a significant number of exploration strategies employing this reward mechanism, however, we review a particular approach which has observed good success in solving sparse reward tasks and offers simplicity of implementation.

Count-based Exploration Rewards [48, 7, 43] bestow an agent with rewards based on state visitation counts (or optionally state-action pair visitation counts). In tabular RL settings, it is tractable to maintain visitation counts for each state. However, in high-dimensional, continuous state spaces, a set of discretized features may be used as a proxy of state, in computing the exploration bonuses. The discretized features, $\phi(s)$, can be derived from expert knowledge, learned via representation learning or obtained through locality-sensitive hashing methods [48]. Representing the number of times a set of features $\phi(s)$ has been observed by $N(\phi(s))$, the exploration bonus reward function $r^+(s, a, s')$ is given by:

$$r^+(s, a, s') = \frac{\xi}{\sqrt{N(\phi(s'))}} \tag{2.16}$$

with $\beta \in \mathbb{R}_{>0}$ representing a scaling factor for the magnitude of exploration rewards.

The reward to the agent is then given by the sum of the reward from the MDP and the exploration bonus:

$$R_{t+1} = r(S_t, A_t, S_{t+1}) + r^+(S_t, A_t, S_{t+1}) \tag{2.17}$$

In addition to the simplicity of count-based exploration rewards, another attractive property is that the exploration rewards tend to zero in the limit with infinite exploration, and the original MDP is recovered. This ensures that (sufficiently) trained policies will be optimal with respect to the original MDP.

2.4 Options Framework

Hierarchical RL represents the notion that learning efficiencies can be achieved through "decomposition of an RL problem into a hierarchy of subproblems or subtasks such that higher-level parent-tasks invoke lower-level child tasks as if they were primitive actions" [22]. This allows learning at different levels of abstraction, and is seen as a means of addressing Bellman's *Curse of Dimensionality* [8, 22].

The Options Framework [47, 36] presents a model for Hierarchical RL that attempts to realize potential learning efficiencies through the use of temporally-extended actions (or macro-actions). Under this framework, macro-actions are represented by a set of *options* $\Omega \coloneqq \{\omega\}$, where each option embodies a policy that may be executed for some period of time before switching to another option/policy. An option $\omega \coloneqq \langle \mathcal{I}_\omega, \pi_\omega, \beta_\omega \rangle$ is comprised of three elements:

- Initiation Set ($\mathcal{I}_\omega$): $\mathcal{I}_\omega \subseteq \mathcal{S}$ refers to the set of states over which the option may be invoked
- Policy (π_ω): The policy that is utilized by the agent to output actions, when an option ω is active
- Termination Condition (β_ω): A state-dependent probability distribution $\beta_\omega(s)$, $\beta_\omega : \mathcal{S} \to [0, 1]$, under which the option stochastically terminates

The interaction between the agent and the environment under the Options Framework is as follows. At time t, an action is sampled from the policy of the currently active option, $a_t \sim \pi_\omega(\cdot|s_t)$, and executed in the environment. A state transition to s_{t+1} is observed, upon which the option terminates with probability $\beta_\omega(s_{t+1})$ or continues with probability $1 - \beta_\omega(s_{t+1})$. If the option terminates, another option may be invoked in accordance to its

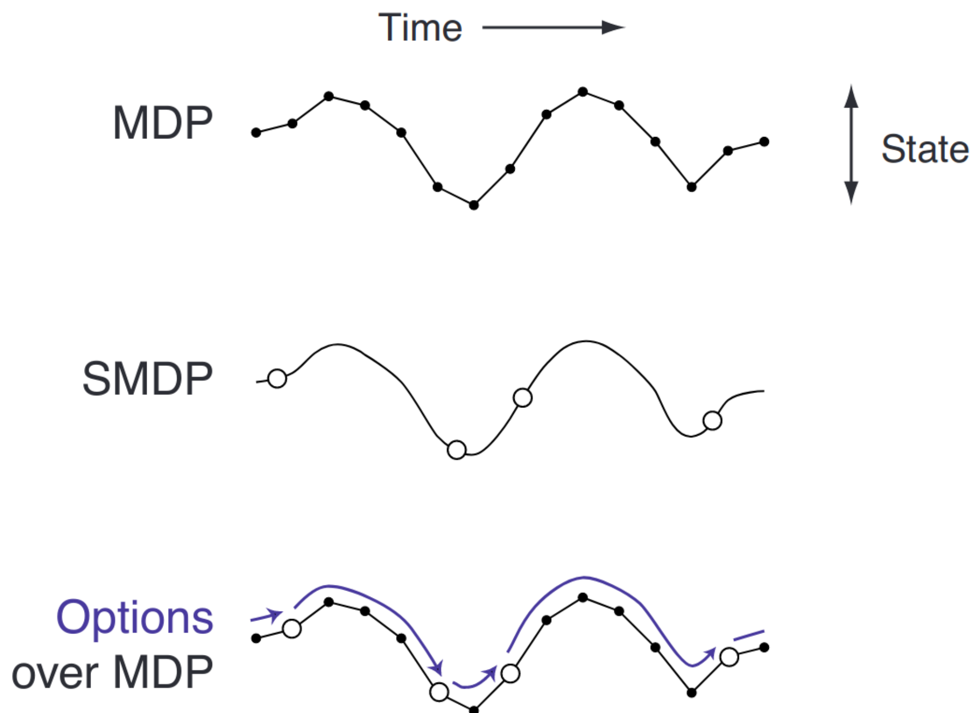


Figure 2.2: The trajectory in the MDP is comprised of small, evenly spaced, discrete-time state transitions while the SMDP contains irregularly spaced, option transitions at a coarser temporal resolution. Figure obtained from [47]

initiation set and a policy over options, $\pi_\Omega(\omega|s)$, which may be learnt. If an explicit policy over options is not learnt, a greedy or ϵ-soft policy (or any other arbitrary policy) acting on the value estimates of all options may be used to select a new option.

An MDP acted upon by a set of options, produces a Semi-Markov Decision Process (SMDP) [38, 47]. This can be graphically seen in Figure 2.2. The presence of a two level hierarchy can be observed, with the option transitions occuring at a coarse temporal resolution and state trajectories being generated at finer regular intervals in accordance with the option active at each point in time.

Learning in the SMDP necessitates the introduction of additional Bellman-like equations for value learning at the option level. Since the choice of option represents a higher-level "action", a *Value Function over Options* $V_\Omega(s)$ and *Option-Value Function* $Q_\Omega(s, \omega)$ are introduced [47]. Additionally, an *Option-Value Function Upon Arrival* [47, 4], $U_\Omega(s, \omega)$,

is used to denote the value of a state-option pair immediately prior to deciding whether to terminate or continue with the current option for the given state:

$$U_\Omega(s,\omega) = (1 - \beta_\omega(s))Q_\Omega(s,\omega) + \beta_\omega(s)V_\Omega(s) \tag{2.18}$$

This allows for the specification of the Bellman Expectation equivalents for $V_\Omega(s)$, $Q_\Omega(s,\omega)$, and optionally, the value of executing an action a under a state-option pair $Q_\Omega(s,\omega,a)$ [4]:

$$\begin{aligned} V_\Omega(s) &= \sum_{\omega\in\Omega} \pi_\Omega(\omega|s) Q_\Omega(s,\omega) \\ Q_\Omega(s,\omega) &= \sum_{a\in\mathcal{A}} \pi_\omega(a|s) \underbrace{\sum_{s'\in\mathcal{S}} p(s'|s,a)\left[r + \gamma U_\Omega(s',\omega)\right]}_{Q_\Omega(s,\omega,a)} \end{aligned} \tag{2.19}$$

Similarly, the Bellman Optimality equivalents may be defined, but we omit these details as they follow in a straightfoward manner from the above. In the following section, we will see how end-to-end learning under the Options framework may be realized using policy gradients.

2.5 Option-Critic Architecture

The *Option-Critic Architecture* [4] provides policy gradient theorems [46] for the end-to-end learning of option policies π_w (or *intra-option learning*) and option termination conditions β_ω, $\forall\omega \in \Omega$, to directly optimize for expected returns. An explicit policy over options, π_Ω, may also be learnt under this scheme [24]. This framework operates on the *call-and-return* option execution model, whereby an option ω may be picked using a policy over options π_Ω and, primitive actions are then sampled and executed from the option policy π_ω until it terminates according to β_ω which is probed at each step during the lifetime of the option, upon which the procedure is repeated. A key assumption under this architecture is that all options may be executed in any state i.e. $\mathcal{I}_\omega = \mathcal{S}$, $\forall\omega \in \Omega$.

In the following subsections, we present the policy gradient theorems employed to optimize the option policies and termination conditions, and subsequently depict an algorithm which demonstrates learning under these developments. Henceforth, we parameterize the option policies by a set of parameters θ and the termination conditions by a set of independent parameters ϑ.

2.5.1 Intra-Option Policy Gradient Theorem

Here, we describe the policy gradient theorem formulation for optimizing the intra-option policies, $\pi_{\omega,\theta}$ $\forall \omega \in \Omega$.

For the policy gradient theorem under the standard RL framework, the objective to maximize is given by Equation (2.8). In light of the developments and notations introduced by the Options Framework [47, 36], the re-stated objective to be maximized with respect to the distribution of initial states and options, $\mu(s_0, \omega_0)$, is given by:

$$J(\theta) = \sum_{\substack{s_0 \in \mathcal{S} \\ \omega_0 \in \Omega}} \mu(s_0, \omega_0) Q_\Omega(s_0, \omega_0) \tag{2.20}$$

The gradient of this objective function [4, 24] is then given in expectation by:

$$\nabla_\theta J(\theta) \propto \mathbb{E}_{\pi_\Omega, \pi_\omega, \beta_\omega} \left[\nabla_\theta \ln \pi_{\omega,\theta}(a|s) Q_\Omega(s, \omega, a) \right] \tag{2.21}$$

Variance reduction for learning with intra-option policy gradients can be achieved through subtracting $Q_\Omega(s, \omega)$ from $Q_\Omega(s, \omega, a)$ in Equation (2.21), similar to the utilization of $V_\pi(s)$ as a baseline function for policy gradients in the standard RL framework.

2.5.2 Termination Gradient Theorem

Now we look at a policy gradient theorem for learning termination conditions, $\beta_{\omega,\vartheta}$ $\forall \omega \in \Omega$.

The objective function to be optimized for the termination condition is expressed a little differently compared to its intra-option policy counterpart, and is given by:

$$J(\vartheta) = \sum_{\substack{s_1 \in \mathcal{S} \\ \omega_0 \in \Omega}} \mu(s_1, \omega_0) U_\Omega(s_1, \omega_0) \tag{2.22}$$

Both the intra-option policy and termination gradient objective functions are consistent with each other and focus on maximizing the expected reward. The reason for a modified objective function is because the termination condition never acts on the initial state s_0, and is invoked for the first time on the successor state produced from acting according ω_0 and starting in state s_0.

The policy gradient for the termination condition [4, 24] is given by:

$$
\begin{aligned}
&\nabla_\vartheta J(\vartheta) \propto \mathbb{E}_{\pi_\Omega,\pi_\omega,\beta_\omega}\left[-\nabla_\vartheta \beta_{\omega,\vartheta}(s) A_\Omega(s,\omega)\right] \\
&A_\Omega(s,\omega) \equiv Q_\Omega(s,\omega) - V_\Omega(s)
\end{aligned}
\tag{2.23}
$$

The appearance of the advantage function over options is a direct consequence of the derivation of the termination gradient theorem. It's role in the policy gradient can be interpreted as follows; if the option-value of a particular option in a given state exceeds the value function over all options for that state, then the expected reward can be improved through lowering the termination probability of the option for that state.

2.5.3 Algorithm

Consolidating both of the results from the intra-option policy gradient and termination gradient theorems, a practical algorithm from [4] is presented in Algorithm 3. Note that in this algorithm, the policy over options, π_Ω, is represented by an ϵ-soft choice over the option-value functions.

2.5.4 Option Degeneracy

Option Degeneracy [4, 20, 21] represents an area of concern in end-to-end learning with options. This refers to a collapse in learning, where the agent may learn to quickly terminate all options, in the extreme case utilizing any option for only one timestep. While short-lived options may still yield solutions to the task at hand, it goes against the main idea behind options, which is to acquire reusable sets of temporally extended skills (also known as *Option Discovery*).

Value Estimation uncertainty [20, 21] has been identified as a major contributor to the option degeneracy problem. Over the course of learning, the agent has to build up option-value estimates, $Q_\Omega(s,\omega)$, for all options. However, termination gradients (Equation (2.23)) acting on noisy/biased estimates in the computation of $A_\Omega(s,\omega)$ can result in option degeneracy and derail the learning process. To exemplify why this might happen, we consider the following scenario.

All options may share similar option-value estimates, especially in the early stages of training. This can result in an effectively random policy over options and little change in the termination probabilities for all options (since $Q_\Omega(s,\omega) \approx V_\Omega(s),\ \forall\omega \in \Omega$). As a consequence, all options may be short-lived, especially if the agent is initialized with

Algorithm 3 Option-Critic with tabular intra-option Q-learning ▷ Obtained from [4]

1: $s \leftarrow s_0$
2: Choose ω according to an ϵ-soft policy over options $\pi_\Omega(\cdot|s)$
3: **repeat**
4: Choose a according to $\pi_{\omega,\theta}(\cdot|s)$
5: Take action a in s, observe s',r

6: **1. Options Evaluation:**
7: $\delta \leftarrow r - Q_\Omega(s, \omega, a)$
8: **if** s' is non-terminal **then**
9: $\delta \leftarrow \delta + \gamma \left[(1 - \beta_{\omega,\vartheta}(s'))Q_\Omega(s', \omega) + \beta_{\omega,\vartheta}(s') \max_{\bar{\omega}} Q_\Omega(s', \bar{\omega})\right]$
10: **end if**
11: $Q_\Omega(s, \omega, a) \leftarrow Q_\Omega(s, \omega, a) + \alpha\delta$

12: **2: Options Improvement**:
13: $\theta \leftarrow \theta + \alpha_\theta \frac{\partial \pi_{\omega,\theta}(a|s)}{\partial \theta} Q_\Omega(s, \omega, a)$
14: $\vartheta \leftarrow \vartheta - \alpha_\vartheta \frac{\partial \beta_{\omega,\vartheta}(s')}{\partial \theta} (Q_\Omega(s', \omega) - V_\Omega(s'))$

15: terminate $\sim$ Bernoulli($\beta_{\omega,\vartheta}(s')$)
16: **if** terminate **then**
17: choose new ω according to ϵ-soft($\pi_\Omega(\cdot|s')$)
18: **end if**
19: $s \leftarrow s'$
20: **until** s' is terminal

an (approximately) unbiased termination probability of 0.5 for all states and options. As previously mentioned, bootstrapping using TD learning with a high degree of truncation (since we may only experience consecutive rewards for any option over a single step or a small number of steps) can make it difficult to correct for bias from the boostrapped estimates. Hence, the agent may get entangled in a web of noisy value estimates that prevents it from experiencing any option for an extended period of time and building better value estimates.

Harb et al. [20] propose the use of *deliberation costs*, a reward penalty ($\eta \in \mathbb{R}_{>0}$) incurred by the agent every time an option terminates, to encourage prolonged option use in the face of noisy value estimates. The termination gradient is accordingly changed to:

$$\nabla_{\vartheta} J(\vartheta) \propto \mathbb{E}_{\pi_{\Omega},\pi_{\omega},\beta_{\omega}} \left[-\nabla_{\vartheta} \beta_{\omega,\vartheta}(s) \left(A_{\Omega}(s,\omega) + \eta \right) \right] \tag{2.24}$$

where η may be interpreted as a margin [20]. However, learning may be highly sensitive to the hyperparameter η, which can be difficult to tune [21]. Harutyunyan et al. [21] present the *Termination Critic*, where the objective of the termination condition(s) is to *compress* or reduce the entropy of states in which options can terminate. They demonstrate that their approach produces options with more intuitive behaviours and option terminations are directed into a small number of states. However, the compromise is that the option termination behaviours no longer attempt to directly optimize for rewards, and hence, are not consistent with the objective of option policies i.e. the agent has potentially competing interests. Additionally, the approach of [21] requires estimation of additional functional entities in comparison to [4, 20], making it less appealing for learning in complex domains due to the introduction of additional sources of variance/uncertainty.

Chapter 3

Sequential Task Learning

In this chapter, we begin with a description and formalization of our problem setting, with the premise of learning for tasks which can be broken down into a sequential composition of subtasks. Subsequently, we present our *Options Chain* model and a novel reward formulation that allows for learning distinct and cooperative policies for each subtask, in order to solve the entire task. Next, we introduce the idea of *default* actions which can reduce the burden of exploration by leveraging expert knowledge of the subtasks comprising the task. Then, we introduce and emphasize the importance of entropy regularization in the learning of termination conditions. An option-based variant of the off-policy SAC [18] algorithm with policy and termination condition entropy regularization is presented, that can learn under our framework. Finally, we conclude this chapter with a quick look at some relevant works and final remarks on our methodology.

Before laying out our methodology, we re-iterate the goals of this work. We wish to develop an explainable and hierarchical reinforcement learning agent for robotic tasks.

Explainability refers to transparency into the behavior of reinforcement learning agents, which is needed for safe and predictable use of RL in real-world applications. Conventional reinforcement learning agents are often represented by monolithic neural network policies that are difficult to introspect. A major concern with contemporary deep learning models (both in RL and the broader AI space) is that generated explanations for the behaviors of these models are often not intuitive to end-users of these systems [37, 14], which we aim to rectify through this work.

The use of hierarchy, on the other hand, has been shown to improve sample efficiency in RL. Sample efficiency is a core issue in Reinforcement Learning, and is particularly important in the robotics domain, where reduced learning interactions translate to reduced

costs of operation and mitigation of potential damage to robots and their environments. While there exist many approaches to HRL, we utilize the Options Framework [47] to realize the combined goal of hierarchy and explainability.

The insight that we build upon is that many robotic tasks, especially in robotic manipulation, can be represented as an ordered sequence of subtasks. This, in turn, offers an opportunity to leverage the Options Framework, where we can specialize each option towards a particular subtask. Decomposition of robotic tasks into distinct subtasks is a common feature of classical approaches to robotics, and promotes explainability into the behavior of the agent. So the question that we attempt to answer over this chapter is "How to specialize each option towards a particular subtask (and do it well) while simultaneously learning all subtask policies and transitions between subtasks?".

3.1 Problem Setting

In this work, we aim to solve a long-horizon task by breaking it down into a chain of sequential subtasks (collectively represented by Ω), where all subtask policies and transitions between subtasks are simultaneously learnt. We represent each subtask by an option, such that an agent learns distinct policies and termination conditions specific to each subtask in the chain. In a similar vein to the Option-Critic Architecture [4], we assume that the initiation set for all options spans the entire state space i.e $\mathcal{I}_\omega = \mathcal{S},\ \forall \omega \in \Omega$.

We extend the definition of an option to include an option-specific reward function $R_\omega(s, a, s')$ and (optionally) an option-specific action space $\mathcal{A}_\omega$ such that $\pi_\omega : \mathcal{S} \times \mathcal{A}_\omega \rightarrow [0, 1]$, yielding $\omega \coloneqq \langle \mathcal{I}_\omega, \pi_\omega, \beta_\omega, R_\omega, \mathcal{A}_\omega \rangle$. The use of option-specific rewards allows for learning behaviors relevant to each subtask while the use of option-specific action spaces may allow for a reduction in the size of the action-space for subtasks, where appropriate, reducing the total amount of exploration needed in solving the entire task. For our work, we rely on an expert to specify the option-specific reward functions and action spaces. Works that attempt to learn action space manifolds [3], may potentially be employed to learn these action spaces instead.

3.2 Options Chain Model

For a sequential task, it is natural to chain subtasks, such that each subtask along the chain completes a particular portion of the task and enables successful behaviour for subtasks

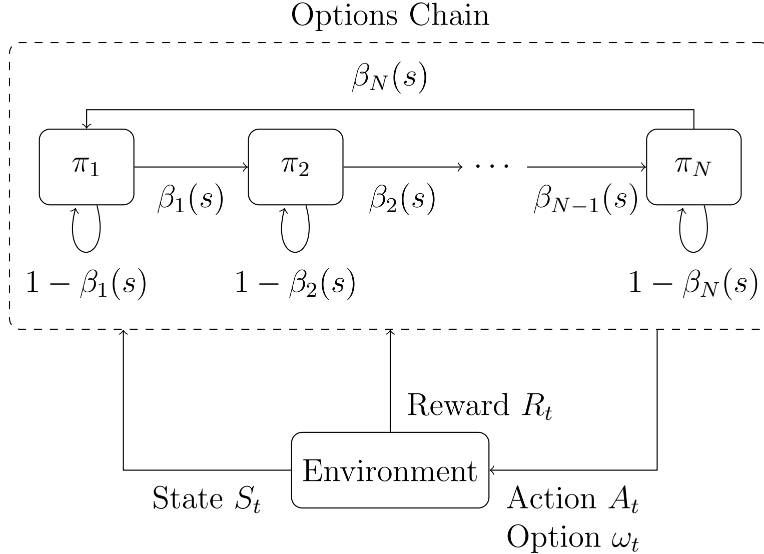


Figure 3.1: Options Chain Architecture for Sequential Task Learning

further along the chain. Using a lifting task with a robotic arm as an example, the task may be broken down into reaching, grasping and lifting subtasks, where each stage needs to be successfully completed in the respective order to finish the task. To this effect, we propose an *Options Chain* to represent the agent, as illustrated in Figure 3.1 (by the dotted box).

In the Options Chain model, each $\pi_i \; \forall i \in \{1, 2, \ldots, N\}$ represents a policy for one of N subtasks and each termination condition $\beta_i \; \forall i \in \{1, 2, \ldots, N\}$ represents a probability distribution according to which the agent stochastically transitions to the next subtask in the chain. Notably, this architecture also allows for transitions (through β_N) from the last subtask back to the first subtask. This provision is made to the benefit of both training and testing stages. During training, especially in the early phases, the agent may pre-maturely transition to successor subtasks/options along the chain. The ability to circumnavigate the chain of subtasks allows the agent to resume interactions with the appropriate subtask. Additionally, during testing time, catastrophic failures in behaviour for a subtask along the chain that leads to loss of progress for prior subtasks, can be elegantly handled by repeating a traversal of the options chain. In the context of the aforementioned lifting task, an example of a catastrophic failure might be the dropping of an object during the lifting phase, thereby eradicating any progress made during the reaching and grasping subtasks in addition to the lifting.

Corollary 1 (Options Chain Termination Gradient Theorem)**.** With the utilization of the Options Chain architecture, the termination gradient theorem (as given by Equation (2.23) for the Option-Critic Architecture) is now given by:

$$\nabla_{\vartheta} J(\vartheta) \propto \mathbb{E}_{\pi_{\Omega}, \pi_{\omega}, \beta_{\omega}} \left[-\nabla_{\vartheta} \beta_{\omega, \vartheta}(s) \left(Q_{\Omega}(s, \omega) - Q_{\Omega}(s, \omega') \right) \right] \tag{3.1}$$

where ω' corresponds to the option that immediately succeeds option ω in the chain.

Proof. The change in the termination gradient theorem comes from the replacement of $V_{\Omega}(s)$ with $Q_{\Omega}(s, \omega')$. However, both these quantities are equal for a fixed, deterministic policy over options given by $\pi_{\Omega}(\omega'|s, \omega) = 1$, that arises from the rigid, sequential structure of the Options Chain model.

The new form has a equally intuitive interpretation as the original termination gradient theorem given by Equation (2.23); if the value of an option ω in a given state s, exceeds the value of the next option ω' in the same state, then the termination probability of option ω in state s should be lowered, and vice versa.

To justify the use of the Options Chain Architecture over the Option-Critic Architecture [4], we ask the following question.

What benefits might using a chain of options provide over the ability to pick any option?

- With a pre-defined execution order for options, we no longer require learning an explicit policy over options, π_{Ω}
- Learning of good/robust policies in RL requires exposing an agent to both *good* and *bad* states [11]. Pre-mature transitions out of a subtask ensures that all successive options get to experience the *bad* states at least once as the agent attempts to traverse the Options Chain back to the appropriate subtask
- In line with the previous argument, a pre-mature transition out of a subtask requires at least N environment interactions to arrive back at the correct subtask. Hence, the agent may incur a significant loss of rewards as a result of any pre-mature subtask terminations. In turn, this can provide the agent with more robust learning signals for termination conditions

3.3 Reward Formulation

In many Options-based works [4, 24, 20], it is common to generate a reward for each agent interaction using the same reward function, regardless of the option utilized. However, this may yield trained options that lack intuitively meaningful behaviours and distinct skills. Additionally, learning may result in option degeneracy in the absence of algorithmic novelties, as previously noted in Section 2.5.4.

In order to encourage option specialization/discovery, a number of heurisitic methods have historically been employed. A popular line of thought has been to identify *bottleneck* states [31, 5] from demonstrations or during learning, through the construction of state transition graphs and application of graph partitioning and clustering methods. The bottleneck states act as subgoals to create subgoal-driven options. The subgoals, in turn, allow for the specification of *pseudo-reward* functions where entsagen are rewarded for attaining these subgoals. The construction and decomposition of state transition graphs for continuous state-spaces presents a more difficult problem than discrete state-spaces [5].

Instead, we propose the use of reward functions that can express the reward for a task using a set of potential and gating functions (defined below), which in turn may be decomposed to form option-specific reward functions to encourage behaviour specialization. We begin with the follow definitions.

Definition 1 (Potential Function)**.** A bounded, continuous and positive-valued function of state, $\Phi(s) : \mathcal{S} \to \mathbb{R}_{>0}$

Definition 2 (Gating Function)**.** A binary function of state, $g(s) : \mathcal{S} \to \{0, 1\}$

Definition 3 (Feasible Set for Gating Functions)**.** A set of states $\mathcal{S}_i$ for a gating function $g_i(s)$ s.t $g_i(s) = 1, \ \forall s \in \mathcal{S}_i$

Definition 4 (Congruent Gating Functions)**.** For any pair of gating functions (g_i, g_j) with corresponding feasible state sets $\mathcal{S}_i$ and $\mathcal{S}_j$ respectively, the gating functions are congruent iff $\mathcal{S}_i \cap \mathcal{S}_j \neq \emptyset$

In manipulation tasks (and many robotic applications), potentials/rewards for particular subtasks (e.g. reaching) may be naturally represented via distance-based functions in Euclidean space. However, it may be difficult to design continuous-valued potential/reward functions for subtasks such as grasping without inducing significant expert biases in behaviour. On the other hand, it may be reasonably easy to *verify* if an object is grasped given the state of the environment e.g. through force-closure conditions, and hence allow the condition to be expressed via a gating function.

A reward function for a task may be described using a combination of potential and gating functions corresponding to a set of subtasks. We consider a particular form of composing these functions to design a reward potential for the overall task, which we refer to as the *Composite Task Potential*.

Definition 5 (Composite Task Potential)**.** Consider an arbitrary, ordered set of potential and/or gating functions, corresponding to an ordered sequence of N subtasks, given by $\{f_1, f_2, \ldots, f_N\}$ where any $f_i,\ i \in \{1, 2, \ldots, N\}$ may be a potential or gating function. The composite task potential, $\Phi_{\text{task}}(s)$, is then given by the following program:

```
function ComputeCompositeTaskPotential(state s)
    Φ_task(s) ← f_N(s)
    for i = N − 1, . . . , 1 do
        if f_i is a potential function then
            Φ_task(s) ← f_i(s) + Φ_task(s)
        else if f_i is a gating function then
            Φ_task(s) ← f_i(s) · Φ_task(s)
        end if
    end for
    return Φ_task(s)
end function
```

Assumption 1 (Congruent Gating Function Set)**.** Any pair of gating functions used in Definition 5 are congruent.

We note that acting under Assumption 1 may restrict the class of sequential task problems that can be addressed using this reward model (in the absence of appropriate modifications to the task state-space). To illuminate this statement, we consider two variations of the task illustrated by Figure 3.2, and detailed in Table 3.1 and Table 3.2.

For the sake of understanding, without loss of generality, we can assume that the form of the potential functions is given by $\Phi_i = 1.0 - \tanh(d_i(s))$, where $d_i(s)$ corresponds to the distance between the robot and the switch (corresponding to that subtask). When the switch is in the required state, we can set $d_i(s)$ to zero, to avoid invalidating the potential associated with reaching a particular switch, as the robot moves to other locations at later times during the task execution.

For Scenario 1 (Table 3.1), we can express the composite reward potential using Definition 5 as follows:

Subtask	**Potential/Gating Function**	**Feasible Set**
Move to Switch 1	Φ_1	-
Turn on Switch 1	g_2	Switch 1 ON
Move to Switch 2	Φ_3	-
Turn on Switch 2	g_4	Switch 2 ON
Move to Flag	Φ_5	-

Table 3.1: Sequential Task Example: Scenario 1

Subtask	**Potential/Gating Function**	**Feasible Set**
Move to Switch 1	Φ_1	-
Turn on Switch 1	g_2	Switch 1 ON
Move to Switch 2	Φ_3	-
Turn on Switch 2	g_4	Switch 2 ON
Move to Switch 1	Φ_5	-
Turn off Switch 1	g_6	Switch 1 OFF
Move to Flag	Φ_7	-

Table 3.2: Sequential Task Example: Scenario 2

$$\begin{aligned}\Phi_{\text{task, Scenario 1}}(s) &= \Phi_1(s) + g_2(s) \cdot (\Phi_3(s) + g_4(s) \cdot \Phi_5(s)) \\ &= \Phi_1(s) + g_2(s)\Phi_3(s) + g_2(s)g_4(s)\Phi_5(s)\end{aligned} \tag{3.2}$$

Through Equation (3.2), we can see that in order to access any potential from turning on Switch 2 (through $g_4(s)$), we also need Switch 1 to be on (i.e. $g_2(s) = 1$). Additionally, the potentials corresponding to moving towards Switch 2 and the Flag are not available to the agent until the necessary switches have been turned on. Thus, we are able to capture the sequential nature of the task in the formulation of the task potential.

We can attempt to follow the same approach for Scenario 2 (Table 3.2) below:

$$\begin{aligned}\Phi_{\text{task, Scenario 2}}(s) &= \Phi_1(s) + g_2(s) \cdot (\Phi_3(s) + g_4(s) \cdot (\Phi_5(s) + g_6(s) \cdot \Phi_7(s))) \\ &= \Phi_1(s) + g_2(s)\Phi_3(s) + g_2(s)g_4(s)\Phi_5(s) + g_2(s)g_4(s)g_6(s)\Phi_7(s)\end{aligned} \tag{3.3}$$

However, it is immediately clear that the pair of gating functions (g_2, g_6) have mutually exclusive feasible sets since Switch 1 cannot be on and off for any given state. Hence, we have a pair of incongruent gating functions that violate Assumption 1. Since $g_2(s)g_6(s) = 0, \forall s \in \mathcal{S}$, we cannot access any potential for g_6 or Φ_7, and therefore, the agent has no incentive to turn off Switch 1 or move towards the flag. In effect, the agent needs to be able to discern the progress along a task as a function of the current state alone, which cannot be achieved in the presence of incongruent gating functions. Modifications to the state-space can resolve such issues. For example, the state-space may be augmented with a history of particular events (e.g. previous activations of switches) occuring during task execution which may in turn be directly used by the gating functions.

Having established the form of the composite task potential, we are still left with the task of deriving subtask-specific rewards for the agent. As previously noted in Section 2.3.1, the use of rewards expressed as a difference in potentials disincentivizes cyclical behaviour and otherwise induces no behaviour biases (in the undiscounted case) for the learning agent. Hence, we can express the reward obtained across the entire task from a single agent interaction, as a difference in the composite task potential over the state change induced by the agent's action i.e. $R(s, a, s') = \Phi_{\text{task}}(s') - \Phi_{\text{task}}(s)$. Using the task potential from Equation (3.2) as an example, the reward can then be expressed in its expanded form as follows:

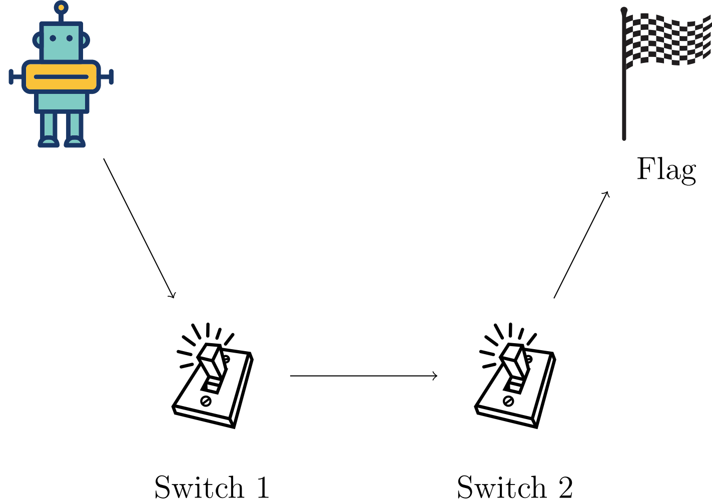


Figure 3.2: Sequential Task Example. Robot has to turn switches on/off in a defined sequence and head to the flag (**Depicted:** Task Scenario 1 from Table 3.1). Illustrations obtained from Vecteezy.com

$$
\begin{aligned}
R(s,a,s') = \quad & [\Phi_1(s') + g_2(s')\Phi_3(s') + g_2(s')g_4(s')\Phi_5(s')] \\
& - [\Phi_1(s) + g_2(s)\Phi_3(s) + g_2(s)g_4(s)\Phi_5(s)] \\
= \quad & \underbrace{[\Phi_1(s') - \Phi_1(s)]}_{R_1(s,a,s')} + \underbrace{[g_2(s')\Phi_3(s') - g_2(s)\Phi_3(s)]}_{R_{2+3}(s,a,s')} \\
& + \underbrace{[g_2(s')g_4(s')\Phi_5(s') - g_2(s)g_4(s)\Phi_5(s)]}_{R_{4+5}(s,a,s')}
\end{aligned}
\tag{3.4}
$$

As per Equation (3.4), we have decomposed the task reward into a sum of potential differences (indicated by R_1, R_{2+3} and R_{4+5} where the indices correspond to the respective subtasks) of the constituents of the composite task potential. However, thus far, we have decomposed the reward for the five stage task (based on Table 3.1) into a set of three reward functions corresponding to the first, second + third, and fourth + fifth subtasks. We can go a step further in decomposing the R_{2+3} and R_{4+5} reward functions through some algebraic manipulation:

$$
\begin{aligned}
R_{2+3}(s,a,s') &= g_2(s')\Phi_3(s') - g_2(s)\Phi_3(s) \\
&= \underbrace{[g_2(s') - g_2(s)]\,\Phi_3(s)}_{R_2(s,a,s')} + \underbrace{g_2(s')\,[\Phi_3(s') - \Phi_3(s)]}_{R_3(s,a,s')}
\end{aligned} \tag{3.5}
$$

$$
\begin{aligned}
R_{4+5}(s,a,s') &= g_2(s')g_4(s')\Phi_5(s') - g_2(s)g_4(s)\Phi_5(s) \\
&= \underbrace{[g_2(s')g_4(s') - g_2(s)g_4(s)]\,\Phi_5(s)}_{R_4(s,a,s')} + \underbrace{g_2(s')g_4(s')\,[\Phi_5(s') - \Phi_5(s)]}_{R_5(s,a,s')}
\end{aligned} \tag{3.6}
$$

Interpreting the form of $R_2(s,a,s')$, the transition from a state $s \notin \mathcal{S}_2$ to $s' \in \mathcal{S}_2$ provides the agent with a reward of $\Phi_3(s)$. Conversely, the agent will be penalized for the reverse transition. For $R_3(s,a,s')$, the agent will accrue reward based on the difference in potential of Φ_3 as long as the subsequent states remain in the feasible set of g_2. Definition 6 acting under Assumption 2 formalizes the decomposition of the reward functions into the subtask-specific reward functions. Assumption 2 is stipulated to allow for the algebraic manipulation in the decomposition of reward functions seen in Equations 3.5 and 3.6.

Assumption 2 (Consecutive Gating Functions)**.** For an ordered set of N potential and/or gating functions, $\{f_1, f_2, \ldots, f_N\}$, there exist no consecutive gating functions in the ordered set

Definition 6 (Subtask Reward Functions)**.** Under the ordered set of N potential and/or gating functions, $\{f_1, f_2, \ldots, f_N\}$, constituting the Composite Task Potential of Definition 5, the reward function for a subtask $i \in \{1, \ldots, N\}$ is given by the following program:

---

function SUBTASKREWARDFUNCTION(subtask i, state s, next state s')
  $g_{\text{combined}}(s) \leftarrow 1$
  $g_{\text{combined}}(s') \leftarrow 1$
  for j=1…i-1 **do**
    if f_j is a gating function **then**
      $g_{\text{combined}}(s) \leftarrow f_j(s) \cdot g_{\text{combined}}(s)$
      $g_{\text{combined}}(s') \leftarrow f_j(s') \cdot g_{\text{combined}}(s')$
    end if
  end for

---

if f_i is a potential function **then**
  $R_i(s,a,s') \leftarrow g_{\text{combined}}(s')(f_i(s') - f_i(s))$
else if f_i is a gating function **then**
  $g_{\text{combined}}(s) \leftarrow f_i(s) \cdot g_{\text{combined}}(s)$
  $g_{\text{combined}}(s') \leftarrow f_i(s') \cdot g_{\text{combined}}(s')$
  $k \leftarrow i+1$
  if f_k is a potential function **then**
    $R_i(s,a,s') \leftarrow 0$
    while f_k is a potential function **do**
      $R_i(s,a,s') \leftarrow R_i(s,a,s') + (g_{\text{combined}}(s') - g_{\text{combined}}(s))\, f_k(s)$
      $k \leftarrow k+1$
    end while
  else
    $R_i(s,a,s') \leftarrow g_{\text{combined}}(s') - g_{\text{combined}}(s)$
  end if
end if
return $R_i(s,a,s')$
end function

Lastly, we attempt to answer the following question, "If an agent acts using the policy of an option i (or subtask i), is it sufficient to reward the agent with the subtask reward (generated via Definition 6) of that option alone?". In short, the answer is no. This is because during the execution of an option i, the agent may attempt to maximize it's cumulative reward by inducing negative progress for other subtasks without incurring penalities. This may allow the agent to recover undone progress on the other subtasks and add to its total reward. As an example, during the subtask to press Switch 1, the agent may be incentivized to move away from the switch, so that it may collect the reaching reward associated with moving towards the switch again at a later time. To discourage such behaviour, we propose the following reward scheme:

Definition 7 (Reward Scheme)**.** For a set of subtasks $\Omega = \{1, 2, \ldots, N\}$, the reward for acting in accordance to a subtask/option policy $\omega \in \Omega$ is given by:

$$R(s,a,s',\omega) = R_\omega(s,a,s') + \sum_{\substack{\omega' \neq \omega \\ \omega' \in \Omega}} \min\left(R_{\omega'}(s,a,s'), 0\right) \tag{3.7}$$

Under Definition 7, we penalize the agent for any loss of rewards induced in other subtasks when attempting to act within a given subtask. This makes it impossible for the

agent to accumulate extra rewards through the aforementioned behaviour. Furthermore, the agent is not rewarded for inducing positive rewards in other subtasks, in order to promote behavior specialization for each option towards its assigned subtask.

3.4 Default Actions

To reduce the burden of exploration, we may also inject expert knowledge into the action spaces of any options. For example, we may require the gripper to be open during a reaching subtask and have it always close during the grasping subtask. Therefore for any option, we can specify a known function to output certain elements of the action space (thus restricting the option to a modified action space $\mathcal{A}_\omega \subseteq \mathcal{A}$), and have the option policy learn the remaining action elements. In the extreme case, all action elements for a given option may be specified through a function, obviating the need to learn an explicit option policy. We refer to the expert-defined elements of the action spaces as *default actions*.

We note that this approach to learning may prevent the agent from realizing the optimal policy. In the aforementioned example, the optimal policy may involve the gripper closing as it approaches the target object, during the reaching subtask. The use of a default action which attempts to keep the gripper open over the reaching option can restrict such behaviour. However, similar to the use of discretized control schemes [49, 34] in reinforcement learning, default actions may provide a suitable tradeoff for optimality in favour of sample complexity by limiting exploration in certain action dimensions.

3.5 Entropy Regularization of Termination Conditions

In previous works, the learning of termination conditions is generally dealt from the perspective of maximizing returns only. However, we argue that entropy regularization of the termination condition can play a significant role in learning.

To elucidate our argument, we assess the effect of biased option values in learning. Consider an option ω and its adjacent option ω' (in the Options chain) in a given state s. Assume the option value $Q(s, \omega)$ has a positive bias while the true value of $Q(s, \omega')$ is known, and where the true values of $Q(s, \omega)$ and $Q(s, \omega')$ are equal. Based on Equation (3.1), the termination probability of option ω, $\beta_\omega(s)$, should (rightly) decrease since $Q(s, \omega) > Q(s, \omega')$ based on initial option-value estimates. Over the course of learning, we can expect the option-value $Q(s, \omega)$ to converge to its true value. However, as

the option-value converges, the change in the termination probability goes to zero (since $Q(s, \omega) \rightarrow Q(s, \omega'))$. Therefore, the termination probability remains biased in favour of option ω despite the true option-values being equal, with irreversible damage occuring due to the initial bias in option-value estimates. This effect may be especially apparent in the case of sequential subtasks, as we can expect that option-values for subtasks further along the chain will take longer to converge to their true value estimates. This is because the nature of such tasks requires the agent to develop good policies and value estimates for earlier subtasks before it is able to adequately address subsequent subtasks and observe relevant state distributions.

We contend that entropy regularization of the termination conditions provides a means of addressing this shortcoming, through attempting to maximize a weighted sum of the expected return and entropy of the termination condition. Hence, we modify the objective of the original termination condition from Section 2.5.2 to include the entropy objective, with the hyperparameter α playing a role akin to the temperature hyperparameter in Equation (2.11):

$$J(\vartheta) = \mathbb{E}_{\pi_\Omega, \pi_\omega, \beta_\omega} \left[\alpha \mathcal{H}(\beta_{\omega,\vartheta}(s)) - \beta_{\omega,\vartheta}(s) A_\Omega(s, \omega) \right] \tag{3.8}$$

Note that the result from sampling a termination condition can be represented as a Bernoulli random variable i.e. sampling from the termination condition can result in one of two possible discrete events, continuing or terminating the corresponding option. Hence, the entropy of the termination condition is simply given by:

$$\mathcal{H}(\beta_{\omega,\vartheta}(s)) = -\left[\beta_{\omega,\vartheta}(s) \log \beta_{\omega,\vartheta}(s) + (1 - \beta_{\omega,\vartheta}(s)) \log(1 - \beta_{\omega,\vartheta}(s)) \right] \tag{3.9}$$

The entropy is maximized with $\beta_{\omega,\vartheta}(s) = 0.5$. Hence, in the absence of any value advantage from pursuing a given option, the new termination condition objective will shift the termination probability towards the unbiased probability of 0.5.

3.6 Sequential Soft Option Actor-Critic (SeqSOAC) Algorithm

To accomodate the developments of our work, we build on the SAC [18] algorithm to offer an off-policy method for learning under our framework. We note that there are precedents to the implementation of the SAC [18] algorithm for option-based learning [29, 30].

We propose our algorithm from the perspective of operating under the Options Chain model (Section 3.2), with associated modifications to the termination gradient equation given by Equation (3.1), and the reward formulation from Section 3.3. Additionally, we incorporate entropy regularization of the termination conditions (Section 3.5) and include the possibility of utilizing default actions (Section 3.4) in these algorithms. Another source of differentiation in our algorithm stems from avoiding the estimation of $Q(s, \omega)\ \forall \omega \in \Omega$, and only estimating $Q(s, \omega, a)$. The procedure for generating data from agent rollouts in depicted in Algorithm 4.

Algorithm 4 Collect Rollouts

1: **for** t=0 ... T-1 **do**
2: $\quad a_t \leftarrow [\pi_{\omega_t,\theta}(s_t), a_{\omega_t,\text{default}}(s_t)]$ ▷ Combine learnt and default actions
3: $\quad s_{t+1} \sim p(\cdot|s_t, a_t)$
4: $\quad r_{t+1} \leftarrow R_{\omega_t}(s_t, a_t, s_{t+1}) + \sum_{\substack{\omega' \neq \omega \\ \omega' \in \Omega}} \min\left(R_{\omega'}(s_t, a_t, s_{t+1}), 0\right)$ ▷ Section 3.3
5: $\quad$ terminate $\sim$ Bernoulli($\beta_{\omega_t,\vartheta}(s_{t+1})$)
6: $\quad$ **if** terminate **then**
7: $\quad\quad \omega_{t+1} \leftarrow$ Next option in Options Chain
8: $\quad$ **else**
9: $\quad\quad \omega_{t+1} \leftarrow \omega_t$
10: $\quad$ **end if**
11: $\quad$ **if** s_{t+1} is terminal **then**
12: $\quad\quad d_{t+1} \leftarrow 1$
13: $\quad$ **else**
14: $\quad\quad d_{t+1} \leftarrow 0$
15: $\quad$ **end if**
16: **end for**

Li et al. [29] introduce an SAC-style [18] method for option learning, called the Soft Option Actor-Critic (SOAC), and demonstrate the utility of off-policy maximum entropy deep RL in improving the sample complexity for option-based learning. Additionally, they perform entropy regularization of a high-level option selection policy, analogous to our approach of entropy regularization of the termination conditions. Lobo et al. [30] present a similar method to [29], attempting to maximize a weighted sum of the returns and, entropies of the option policies and the policy over options.

Under our maximum entropy formulation, we attempt to optimize the following objective:

$$J(\theta,\vartheta) = \mathbb{E}_{\substack{\pi_\omega,\beta_\omega \\ \forall\omega\in\Omega}}\left[\sum_{t=0}^{\infty}\gamma^t\left(R(s_t,a_t,s_{t+1},\omega_t) + \alpha_{\omega_t}\mathcal{H}(\pi_{\omega_t,\theta}(\cdot|s_t))\right) + \sum_{t=1}^{\infty}\gamma^t\kappa_{\omega_t}\mathcal{H}(\beta_{\omega_t,\vartheta}(s_t))\right] \tag{3.10}$$

where α_ω and κ_ω represent the temperature parameters for the (action) policies and termination conditions respectively.

The form of the option-value function is accordingly described by:

$$\begin{aligned} Q(s,\omega) = \sum_{a\in\mathcal{A}_\omega}\pi_\omega(a|s)\sum_{s'\in\mathcal{S}}p(s'|s,a)\{R(s,a,s',\omega) + \alpha_\omega\mathcal{H}(\pi_{\omega,\theta}(\cdot|s)) + \\ \gamma[\kappa_\omega\mathcal{H}(\beta_{\omega,\vartheta}(s')) + (1-\beta_{\omega,\vartheta}(s'))Q(s',\omega) + \beta_{\omega,\vartheta}(s')Q(s',\omega')]\} \end{aligned} \tag{3.11}$$

Hence, we express the option-value function upon arrival by:

$$U(s,\omega) = \kappa_\omega\mathcal{H}(\beta_{\omega,\vartheta}(s)) + (1-\beta_{\omega,\vartheta}(s))Q(s,\omega) + \beta_{\omega,\vartheta}(s)Q(s,\omega') \tag{3.12}$$

Similar to the relationship between the value function $V_\pi(s)$ and action-value function $Q_\pi(s,a)$ in [18], we establish the following relationship between $Q(s,\omega)$ and $Q(s,\omega,a)$:

$$Q(s,\omega) = \mathbb{E}_{a\sim\pi_\omega}\left[Q(s,\omega,a) + \alpha\mathcal{H}(\pi_\omega(a|s))\right] \tag{3.13}$$

Put differently, $Q(s,\omega,a)$ can be expressed using the following recursive form:

$$\begin{aligned} Q(s,\omega,a) &= \sum_{s'\in\mathcal{S}}p(s'|s,a)\{R(s,a,s',\omega) + \\ &\qquad \gamma[\kappa_\omega\mathcal{H}(\beta_{\omega,\vartheta}(s')) + (1-\beta_{\omega,\vartheta}(s'))Q(s',\omega) + \beta_{\omega,\vartheta}(s')Q(s',\omega')]\} \\ &= \mathbb{E}_{\substack{\pi_\omega,\beta_\omega \\ \forall\omega\in\Omega}}\begin{bmatrix} R(s,a,s',\omega) + \\ \gamma[\kappa_\omega\mathcal{H}(\beta_{\omega,\vartheta}(s')) + (1-\beta_{\omega,\vartheta}(s'))Q(s',\omega) + \beta_{\omega,\vartheta}(s')Q(s',\omega')] \end{bmatrix} \end{aligned} \tag{3.14}$$

In our algorithm, we employ a critic to estimate $Q(s,\omega,a)$, using Equation (3.14) to generate target values for $Q(s,\omega,a)$ and minimizing a mean squared error loss between the

estimated and target values. As in the Option-Critic architecture [4], option policies are optimized through maximizing $Q(s, \omega)$, which can be evaluated through Equation (3.13). Similarly, the termination conditions are optimized through maximizing $U(s, \omega)$, as expressed in Equation (3.12). Additionally, we compute critic estimates for the termination conditions from the target critic networks as opposed to the main critic networks. As argued by prior works employing hierarchical policy structures and off-policy algorithms [28], the learning of lower-level policies requires a degree of stationarity in the higher-level policies of the hierarchy, which we achieve by utilizing the smoothly changing (through exponential averaging) target critic networks in the computation of Equation (3.12). Under the options framework, this higher-level policy is represented by the termination conditions (and a policy over options, if one is utilized).

As in the SAC algorithm [18], to promote stability, two target networks are employed to derive targets for the critics. Additionally, the option policy entropy is estimated using the negative log likelihood of the actions (in the absence of a closed form expression for the policy entropy) i.e. $\mathcal{H}(\pi_\omega(a|s)) = \mathbb{E}_{a\sim\pi_\omega}[-\log \pi_\omega(a|s)]$. Our complete algorithm is presented in Algorithm 5.

3.7 Relevant Works

In addition to the preceding cited works, we highlight some other works that have attempted learning through task decomposition, and have provided inspiration for our methodology.

The idea of chaining options has support from prior works in *Skill Chaining*, where "the goal of each skill in the chain is to enable the agent to reach a state where its successor skill can be successfully executed" [25]. In these works [25, 6], a sequence of options are autonomously constructed backwards from a goal state, where the initiation and termination sets of adjacent options in the chain overlap. However, we opine that due to the back-chaining procedure employed in these works, their application has been largely restricted to navigation tasks where it is feasible to construct valid agent states leading to a goal. In manipulation tasks, due to interaction of the agent with objects in the environment, such an approach appears infeasible.

Our work shares similar ideas with the Max-Q Value Function Decomposition framework [13], which leverages expert knowledge to decompose a task into a hierarchy of subtasks and defines pseudo-reward functions for each subtask. Additionally, it leverages state/action abstractions for sub-tasks to improve learning efficiencies. However, it also

Algorithm 5 Sequential Soft Actor-Critic (SeqSOAC)

1: Input: Option set Ω. Initial parameters θ (policies), ϕ_1 and ϕ_2 (option-action-values/Q-functions), ϑ (termination conditions). Actor and Termination Condition Entropy coefficients α_ω and κ_ω respectively, and Default actions $a_{\omega,\text{default}}(\cdot)$ $\forall \omega \in \Omega$. Empty replay buffer $\mathcal{D}$. Target update parameter ρ

2: Set target parameters equal to main parameters $\phi_{\text{targ},1} \leftarrow \phi_1$, $\phi_{\text{targ},2} \leftarrow \phi_2$

3: **repeat**

4: Perform single rollout step (i.e. $T = 1$) using Algorithm 4

5: Store $(\omega_t, s_t, a_t, r_{t+1}, s_{t+1}, d_{t+1})$ in $\mathcal{D}$

6: **for** $\omega \in \Omega$ **do**

7: **if** it's time to update option ω **then**

8: **for** j in range(however many updates) **do**

9: Sample a batch of transitions generated by option ω, $B_\omega = \{(s, a, r, s', d)\}$, from the replay buffer $\mathcal{D}$

10: Compute targets for the Q functions using r, d and s' from B_ω:

$$\hat{U}_\vartheta(\omega, s') = \left(\begin{array}{c} (1 - \beta_{\omega,\vartheta}(s')) \left[\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \omega, \tilde{a}) - \alpha_\omega \log \pi_{\omega,\theta}(\tilde{a}|s') \right] + \\ \beta_{\omega,\vartheta}(s') \left[\min_{i=1,2} Q_{\phi_{\text{targ},i}}(s', \omega', \tilde{a}') - \alpha_{\omega'} \log \pi_{\omega',\theta}(\tilde{a}'|s') \right] + \\ \kappa_\omega \mathcal{H}(\beta_{\omega,\vartheta}(s')) \end{array} \right)$$

$$y(\omega, r, s', d) = r + \gamma(1 - d)\hat{U}_\vartheta(\omega, s')$$

with $\quad \tilde{a} \sim \pi_{\omega,\theta}(\cdot|s'),\ \tilde{a}' \sim \pi_{\omega',\theta}(\cdot|s')$

11: Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B_\omega|} \sum_{(s,a,r,s',d) \in B_\omega} \left(Q_{\phi_i}(s, \omega, a) - y(\omega, r, s', d) \right)^2 \quad \text{for } i=1,2$$

12: Update option policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B_\omega|} \sum_{s \in B_\omega} \left(\min_{i=1,2} Q_{\phi_i}(s, \omega, \tilde{a}_\theta(s)) - \alpha_\omega \log \pi_{\omega,\theta} \left(\tilde{a}_\theta(s) | s \right) \right), \quad \tilde{a}_\theta(s) \sim \pi_{\omega,\theta}(\cdot|s)$$

13: Update option termination condition by one step of gradient ascent using the values of $\hat{U}_\vartheta(\omega, s')$ calculated for samples in Line 10:

$$\nabla_\vartheta \frac{1}{|B_\omega|} \sum_{s' \in B_\omega} \hat{U}_\vartheta(\omega, s')$$

14: Update target networks with

$$\phi_{\text{targ},i} \leftarrow (1-\rho)\phi_{\text{targ},i} + \rho\phi_i \quad \text{for } i\text{=1,2}$$

15: **end for**
16: **end if**
17: **end for**
18: **until** convergence

requires the expert to partition the state-space for each sub-task into 2 disjoint spaces, where the sub-task either continues with probability 0 or terminates with probability 1. While such a scheme may be feasible for discrete space tasks, application to continuous space tasks may be difficult, particularly in regards to the specification of explicit goal/termination states for sub-tasks.

There have also been attempts at Hierarchical Reinforcement Learning for manipulation tasks, which we review below.

Graph-based approaches leveraging sub-task structure learnt from demonstration data [32, 27, 26] have been successfully employed for manipulation tasks in previous works, albeit only for generating manipulator trajectories without any grasping actions. Graph-based methods seek to construct options through discovery of sub-task structures via clustering methods operating on demonstration data. In these works [27, 26], however, transition regions between options identified from the demonstration data remain fixed over the training of the individual options, and effectively break the problem into a number of disjoint Markov Decision Processes (MDPs). This makes the RL agent vulnerable to out-of-distribution task contexts in regards to the demonstration data, and the use of fixed transition regions precludes the ability for RL to overcome such scenarios through further learning.

In another approach to HRL for manipulation, Stulp et al. [44] decomposed a pick-and-place task into a sequence of subtasks and subsequently trained a sequence of two Dynamic Movement Primitives (DMPs) [40] to complete the overall task through simultaneous learning of the shaping parameters for the primitives and the goals for each primitive. One primitive was utilized to move towards the object and a second primitive to transport the object to the final location, with the grasp and release operations occuring at the completion of the respective primitives. By their own admission, the object was chosen due to ease of grasping and the orientation of the end-effector used in their experiments was fixed over both primitives.

In regards to explainable HRL for manipulation, Beyret et al. [9] developed a two-level hierarchical model, with a higher level policy outputting subgoals for a lower level policy that learns to control the robot to achieve the given subgoals. The visualization of the subgoals leading to the goal offer explainability to the behavior of the RL agent. Under a similar policy structure, Gupta et al. [16] used a combination of imitation learning and RL fine-tuning to learn multi-stage tasks in a kitchen simulation environment. Both of these works represent realizations of a competing HRL approach to the Options framework, known as Feudal Reinforcement Learning [12, 51].

3.8 Final Remarks

Having delineated our methodology in the previous sections, we remark on some elements of our approach.

The reward formulation described in Section 3.3 utilizes differences in potentials to form the reward function. However, in Section 2.3.1, it was noted that complete elimination of expert bias in the specification of reward functions can only be attained with the inclusion of the discount factor γ by expressing the difference in potentials by $\gamma\Phi(s') - \Phi(s)$. We recognize that the inclusion of the discount factor may result in practical complications with learning. For example, consider that a particular action a does not result in any change of state i.e. $s = s'$, for any state $s \in \mathcal{S}$. This will result in a loss of reward equal to $(1 - \gamma)\Phi(s)$. However, since $\Phi(s)$ is higher for increasingly optimal states under a subtask, the agent will get penalized more in such states. Hence, the inclusion of the discount factor may deter the agent from making progress during learning. Additionally, we contend that the amount of bias that is introduced in the learning problem through our reward formulation will likely be minimal, as it is common to utilize discount factors close to one in most reinforcement learning problems.

There may be questions surrounding the nature of the neural networks used for learning policies, option value functions, and/or termination conditions across all options. Particularly, should parameters be shared between the different options? The practitioner may choose any approach, however, in some instances it may be beneficial to create indepedent networks for each option. This is because certain elements of the state space may only be pertinent to some options/subtasks and not others. For example, the position of the target location in a pick-and-place task is not important to the subtasks of reaching and grasping the object. Hence, using separate networks may allow for state space reductions that can improve sample complexity in learning.

Chapter 4

Experiments

To validate our methodology developed in Chapter 3, we perform simulated experiments for manipulation tasks in 2D and 3D domains. We break down these manipulation tasks into sets of sequential subtasks and design appropriate reward functions based on the formulation from Section 3.3.

We recognize that most prior options-based works operate under grid-world, arcade or locomotion environments [4, 24, 20, 21]; thereby exhibiting little applicability to real-world use cases in the robotics domain, especially for robotic manipulation. This motivates our choice of environments, along with the fact that most manipulation tasks have a clear sequential structure to allow for task decomposition. Also, of note, is the fact that the chosen environments do not constrain the posture of the end-effector for the manipulation tasks or artifically open/close the gripper at speeds inconsistent with real hardware platforms, which is commonly seen in many reinforcement learning works [10, 9].

Since the primary contribution of this work is a subtask-focused reward formulation and decomposition, we first attempt to assess the ability of the proposed reward scheme to incentivize the agents to solve the composite tasks. With the composite task potential acting as a proxy to the progress made along a task, we can use it to quantify the extent to which any agents are able to solve a given task.

Recalling our goals of a hierarchical and explainable RL agent for robotic tasks, in this chapter, we seek to evaluate the extent to which these goals are achieved. Note that hierarchical learning is automatically achieved through the nature of our formulation presented in Chapter 3, building on the HRL approach offered by the Options Framework. However, the aim of hierarchy and any concomitant mechanisms to learning that it affords (e.g. default actions, state abstractions, option execution models etc.) is to improve the

sample efficiency in addressing the given tasks. To this end, we utilize a conventional SAC agent to serve as a benchmark in evaluating the sample efficiency of our agents over the training regime, again, using the composite task potential as a measure of the progress made along the tasks. For the SAC agent, since we do not use any options (or alternatively we can interpret it as using a single option, with no termination condition, for the entire task), we express its reward function as the difference in composite task potentials seen over the state transitions i.e. $R(s, a, s') = \Phi_{\text{task}}(s') - \Phi_{\text{task}}(s)$. Additionally, to ensure a fair comparison, we utilize the same count-based reward bonuses (detailed below) as for our options-based methods, to induce appropriate exploration of the state space.

Next, with the aim of evaluating the explainability of our agent, we seek to determine if our options-based agents that specialize at the subtask level, partition the tasks appropriately and utilize the options in a discernible manner over the execution of the tasks. Moreover, with the proposal of our Options Chain model (Section 3.2), we compare option utilization behaviors between the option execution models represented by our Options Chain model and the Option-Critic Architecture [4]. Note that our reward scheme reflects the subtask specialization requirement in its design. Thereby, the returns generated by these agents also serve as a good indicator of their ability to specialize and yield explainable agents.

We briefly reiterate the similarities and differences between the two option execution models. The primary difference lies in the nature of the option invocation. Under the Option-Critic model, upon the termination of an option, another option is picked using a policy over options. In comparison, for the Options Chain model, the agent is strictly required to pick the option in the chain that is immediately adjacent to the terminated option (refer to Figure 3.1 for the visualization of the Options Chain). Speaking to the similarities between the models, we utilize the same algorithm, with minor differences arising from the nature of the option invocation, and we utilize default actions in both implementations for a fair comparison. The output of both models remains the same i.e. for N subtasks/options that are specified, the agent learns N distinct policies, N termination conditions and N critics to estimate $Q(s, \omega, a)\ \forall \omega \in \Omega$.

For the implementation of the Option-Critic baseline, we utilize the same algorithm and hyperparameters as for our Options Chain model. The policy over options, $\pi_\Omega(\cdot|s)$, is represented by a softmax distribution over option values. Accordingly, the value function, $V_\Omega(s)$, is estimated using $\sum_{\omega \in \Omega} \pi_\Omega(\omega|s) \left(Q(s, \omega, a) - \alpha_\omega \log \pi_\omega(a|s)\right)$ and appropriately utilized in Lines 10 and 13 of Algorithm 5. Additionally, we omit the use of termination condition entropy regularization, since the agent might otherwise be incentivized to keep the termination probabilities close to 0.5, as the policy over options can simply re-pick the same option upon termination.

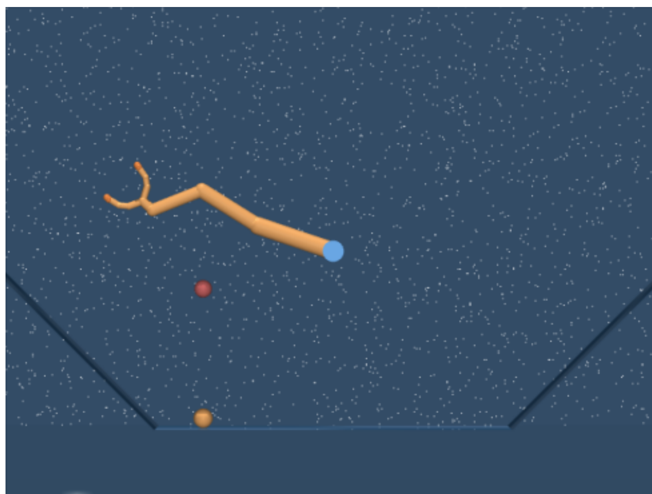


Figure 4.1: Manipulator Environment with bring_ball task from the DeepMind Control Suite [50]

For all experiments, we utilize independent feedforward neural networks for the learning of actors, critics and termination conditions, for all options.

4.1 DeepMind Control Suite Manipulator

The DeepMind Control Suite [50] provides a collection of 2D and 3D simulation environments for reinforcement learning research. In particular, we look at the *manipulator* environment with the *bring_ball* task. Under this setting, a planar, redundant manipulator (with four arm joints and a single gripper joint) operates using joint torques in a 2D environment under gravity, with the aim of picking up a ball (free to roll around) and transporting it to a specified goal position. A visualization of the environment can be seen in Figure 4.1.

Under the original task formulation, the agent is only provided a dense reward dependent on the distance between the ball and the target location. Additionally, the agent is initialized with the ball in the gripper for 10% of all task initializations, and with the ball at the target location in another 10% of the initializations. We modify this task, employing our own reward formulation and initializing the ball on the ground, away from the gripper, 100% of the time.

During training, the time limit for each episode is set to 500 timesteps. Additionally,

no terminal states are used i.e. all episodes run for 500 timesteps, regardless of whether or not the ball has reached the target location.

4.1.1 Subtask and Reward Formulation

We decompose this task into 3 distinct subtasks; reaching, grasping and placing. We define potential functions for each of the reaching and placing subtasks, and a gating function for the grasping subtask that indicates a successful/failed grasp. The form of the potential functions is given by:

$$\Phi(s) = 1.0 - \tanh\left(\operatorname{arctanh}\left(\frac{\sqrt{0.95}}{\text{scale}} d(s)\right)\right) \tag{4.1}$$

where $d(s)$ corresponds to the distance between the gripper and ball for the reaching subtask, and the distance between the ball and target position for the placing subtask. A `scale` value of 0.8 is used for both subtasks. Additionally, we use multipliers of 25 and 100 to scale the ranges of the potential functions for the reaching and placing subtasks respectively i.e. $\Phi_{\text{reach}}(s) = 25\Phi(s)$ and $\Phi_{\text{place}}(s) = 100\Phi(s)$.

For the grasping subtask, the feasible set of the corresponding gating function, $g_{\text{grasp}}(s)$, encompasses all states where the ball is sufficiently close to the center of the gripper and the gripper is adequately closed.

The resulting composite task potential is given by:

$$\Phi_{\text{task}}(s) = \Phi_{\text{reach}}(s) + g_{\text{grasp}}(s)\Phi_{\text{place}}(s) \tag{4.2}$$

and the subtask specific reward functions, generated according to Definition 6, are thus given by:

$$\begin{aligned} R_{\text{reach}}(s, a, s') &= \Phi_{\text{reach}}(s') - \Phi_{\text{reach}}(s) \\ R_{\text{grasp}}(s, a, s') &= \left(g_{\text{grasp}}(s') - g_{\text{grasp}}(s)\right) \Phi_{\text{place}}(s) \\ R_{\text{place}}(s, a, s') &= g_{\text{grasp}}(s') \left(\Phi_{\text{place}}(s') - \Phi_{\text{place}}(s)\right) \end{aligned} \tag{4.3}$$

Since the reward for the grasping condition is sparse (non-zero only for transition of state into or out of the feasible set), we additionally augment the grasping reward with

a count-based exploration reward, as given by Equation (2.16) with $\xi = 10.0$. A set of hand-engineered features with appropriate discretization are utilized for count tracking. This set of features consist of:

1. Distance between the gripper and the ball, discretized uniformly using 20 intervals, on a (base-10) logarithmic scale between 0.01m and 0.1m
2. Angle of the gripper, discretized in $10°$ intervals between $(-\pi, \pi]$
3. Distance between the gripper fingertips, uniformly discretized between 0m to 0.08m, in intervals of 0.008m

4.1.2 Default Actions

In order to reduce the burden of exploration, we utilize default actions for each of the subtasks.

For both the reaching and placing subtasks, we restrict the agent to learning joint torque actions for the four arm joints, while specifying the value of the gripper joint torque action. For the reaching option, the gripper joint torque action is set to the maximum opening torque, while the placing option applies a moderate closing torque only if the object is already grasped (and no torque otherwise). For the grasping option, all elements of the action space are specified, applying no torque on the arm joints and the maximum closing torque on the gripper joint, and thus we do not learn an explicit option policy for the grasping subtask.

Note that no default actions are utilized for the SAC benchmark, as this scheme only seems to be exploitable under an options-based worked where different neural networks are utilized for each subtask.

4.1.3 State Representation and Abstractions

In Table 4.1, we list the observations available from the environment. Note that while the gripper of the manipulator is actuated using a single joint torque, the gripper is actually comprised of four joints.

As previously mentioned, separate feedforward neural networks are employed in the learning of the actor, critic(s) and termination condition for each option. Accordingly, this allows for the specification of different subsets of the complete state space to be utilized

		Size	Description
Manipulator	arm joint positions	8	$\sin(\theta)$, $\cos(\theta)$
	gripper joint positions	8	$\sin(\theta)$, $\cos(\theta)$
	arm joint velocities	4	ω (radians/s)
	gripper joint velocities	4	ω (radians/s)
	gripper touch sensors	5	Axial force
	hand pose	4	x, z, qw, qy
Object	object position	2	x, z
	object orientation	2	qw, qy
	object velocity	3	vx, vz, ωy
Target	target position	2	x, z
	target orientation	2	qw, qy
Total		44	

Table 4.1: State Space for *bring_ball* task of the *manipulator* domain from the DeepMind Control Suite [50]

as inputs to these networks. Tables 4.2, 4.3 and 4.4 details the elements of the state space used in the actors, critics and termination conditions respectively, for each of the options/subtasks.

For the SAC benchmark, as is conventional, the agent is represented by a single actor and critic network. To make as fair of a comparison as possible with our option-based methods, the states fed into the actor and critic networks are the cumulative states used across all the actors and critics of the option-based methods respectively.

4.1.4 Hyperparameters

Again, in light of using separate neural networks for the actors, critics and termination conditions across the options, we need to define the sizes of the neural networks and other hyperparameters associated with the learning of these components. We list the hyperparameters utilized for each of the actors, critics and termination conditions in Tables 4.5, 4.6 and 4.7 respectively.

Additionally, the hyperparameters used for the actor and critic network in the SAC implementation is given by Tables 4.8 and 4.9 respectively.

	Reaching	Grasping	Placing
arm joint positions	✓	-	✓
gripper joint positions	✓	-	✓
arm joint velocities	✓	-	✓
gripper joint velocities	✗	-	✗
gripper touch sensors	✗	-	✗
hand pose	✓	-	✗
object position	✓	-	✓
object orientation	✗	-	✗
object velocity	✗	-	✗
target position	✗	-	✓
target orientation	✗	-	✗

Table 4.2: State-Space Abstractions for Actors in 2D Manipulator Task

	Reaching	Grasping	Placing
arm joint positions	✓	✗	✗
gripper joint positions	✓	✓	✓
arm joint velocities	✓	✗	✓
gripper joint velocities	✗	✗	✗
gripper touch sensors	✗	✗	✗
hand pose	✓	✓	✓
object position	✓	✓	✓
object orientation	✗	✗	✗
object velocity	✗	✗	✗
target position	✓	✓	✓
target orientation	✗	✗	✗

Table 4.3: State-Space Abstractions for Critics in 2D Manipulator Task

	Reaching	**Grasping**	**Placing**
arm joint positions	✗	✗	✗
gripper joint positions	✓	✓	✓
arm joint velocities	✓	✗	✗
gripper joint velocities	✓	✗	✗
gripper touch sensors	✗	✗	✗
hand pose	✓	✓	✓
object position	✓	✓	✓
object orientation	✗	✗	✗
object velocity	✗	✗	✗
target position	✗	✗	✗
target orientation	✗	✗	✗

Table 4.4: State-Space Abstractions for Option Termination Conditions in 2D Manipulator Task

	Reaching	**Grasping**	**Placing**
Hidden Layers	[200,100]	-	[200,100]
Activation Function	ReLU	-	ReLU
Learning Rate	5e-4	-	5e-4
Entropy Coefficient (α_ω)	1e-4	-	1e-3

Table 4.5: Hyperparameters for Actors in 2D Manipulator Task

A discount factor of $\gamma = 0.98$ is used in computing the returns for all agents.

4.1.5 Results

To validate the ability of our reward formulation to incentive the options-based agents to address the composite task, despite individual options only being offered rewards for their respective subtasks, we evaluate the maximum composite task potentials seen by these agents over the course of training. These quantities refer to the maximum composite task potential that is seen over the execution of any given episode, since the composite task potential acts as a proxy for the progress made along the task. In Figure 4.2, we see that both the Options Chain and Option-Critic architectures, operating under the same reward formulation, are appropriately motivated to maximize the composite task potential

	Reaching	**Grasping**	**Placing**
Hidden Layers	[300,200]	[300,200]	[300,200]
Activation Function	ReLU	ReLU	ReLU
Learning Rate	3e-5	3e-5	3e-5
Smoothing Constant (τ)	5e-3	5e-3	5e-3

Table 4.6: Hyperparameters for Critics in 2D Manipulator Task

	Reaching	**Grasping**	**Placing**
Hidden Layers	[200,200]	[200,200]	[200,200]
Activation Function	ReLU	ReLU	ReLU
Learning Rate	5e-7	5e-7	5e-7
Entropy Coefficient (κ_ω)	5e-1	5e-1	5e-1

Table 4.7: Hyperparameters for Termination Conditions in 2D Manipulator Task

Hidden Layers	[200,100]
Activation Function	ReLU
Learning Rate	5e-4
Entropy Coefficient (α_ω)	1e-3

Table 4.8: Hyperparameters for SAC Actor in 2D Manipulator Task

Hidden Layers	[300,200]
Activation Function	ReLU
Learning Rate	3e-5
Smoothing Constant (τ)	5e-3

Table 4.9: Hyperparameters for SAC Critic in 2D Manipulator Task

over the course of training. Note that the SAC benchmark which directly optimizes for the composite task potential through its reward function, also attains a similar level of performance towards the end of the training regime. This offers support for the claim that our options-based reward scheme is consistent with the primary goal of solving the task. It should also be noted that the upper bound for the composite task potential is 125, based on the formulation described in Section 4.1.1.

To evaluate any benefits in sample efficiency from the use of our hierarchical RL agents, we again refer to Figure 4.2. We note that the conventional SAC agent proves to be more sample efficient in the early stages of training, but is outpaced as learning proceeds. Comparing the sample efficiency of the agents in reaching their peak task potentials, the Option-Critic architecture exhibits the best performance, followed by the SAC and Options Chain agents showing a similar level of performance.

However, the goal of the (options-based) reward formulation is two-fold; the first being to solve the composite task and the second being to utilize appropriate options for different aspects of the task (i.e. explainability/behavior specialization). Our reward formulation, as described by Definition 7, penalizes errant behaviours. It does so by not allowing the agent to claim rewards if the utilization of a wrong option induces progress along the task, and penalizing the agent for undoing progress in other subtasks when operating under a given subtask/option. In Figure 4.3, we show the rewards obtained by the agent over the course of training under both option models. It is seen that the Options Chain model is able to generate higher returns compared to the Option-Critic model, towards the latter stages of training. With comparable levels of proficiency in solving the composite task as seen in Figure 4.2, we can attribute the higher returns seen by the Options Chain model to better behavior specialization.

To visualize the behavior exhibited by the agents operating under both models, we execute the final trained agents under both option models, over a test episode of 150 timesteps. We show in Figure 4.4, the nature of the option utilization and the corresponding evolution of the composite task potentials over the test episodes. We can see that the Options Chain model in Figure 4.4a exhibits cleaner option utilization, considering the sequential nature of the task. Meanwhile, referring to Figure 4.4b, the Option-Critic architecture shows larger variance in its utilization of options over the episode.

For the training plots given by Figures 4.2 and 4.3, 10 episodes are utilized for the evaluation of the mean and standard deviations.

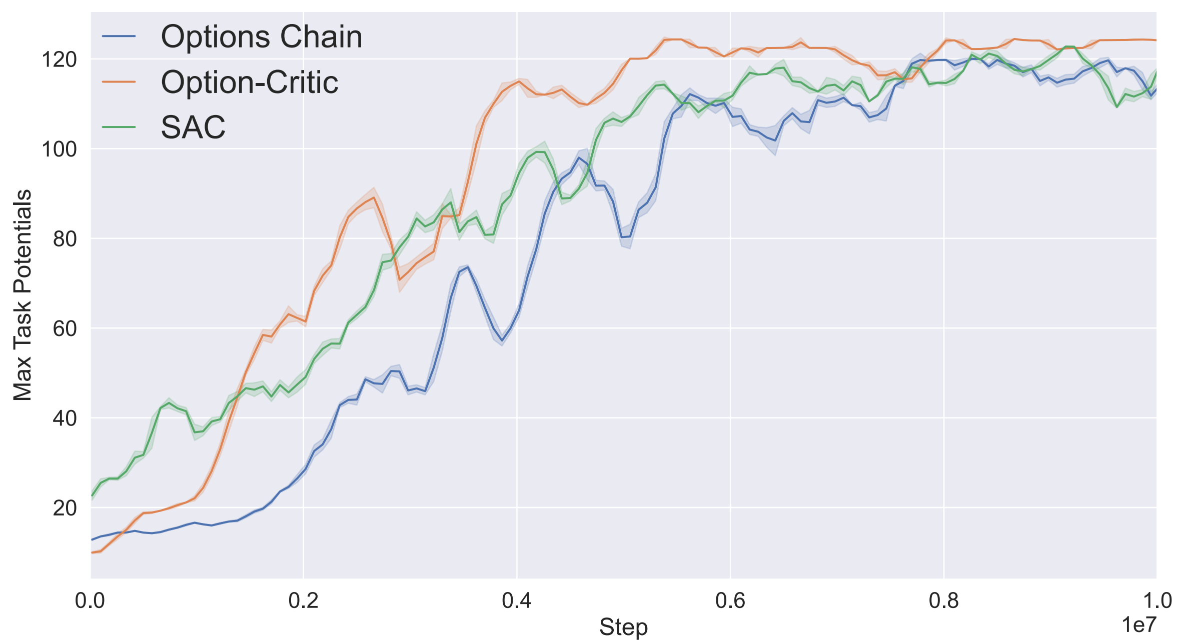


Figure 4.2: Training Plot of Maximum Task Potentials achieved by the Options Chain, Option-Critic and SAC Agents for the 2D Manipulation Task

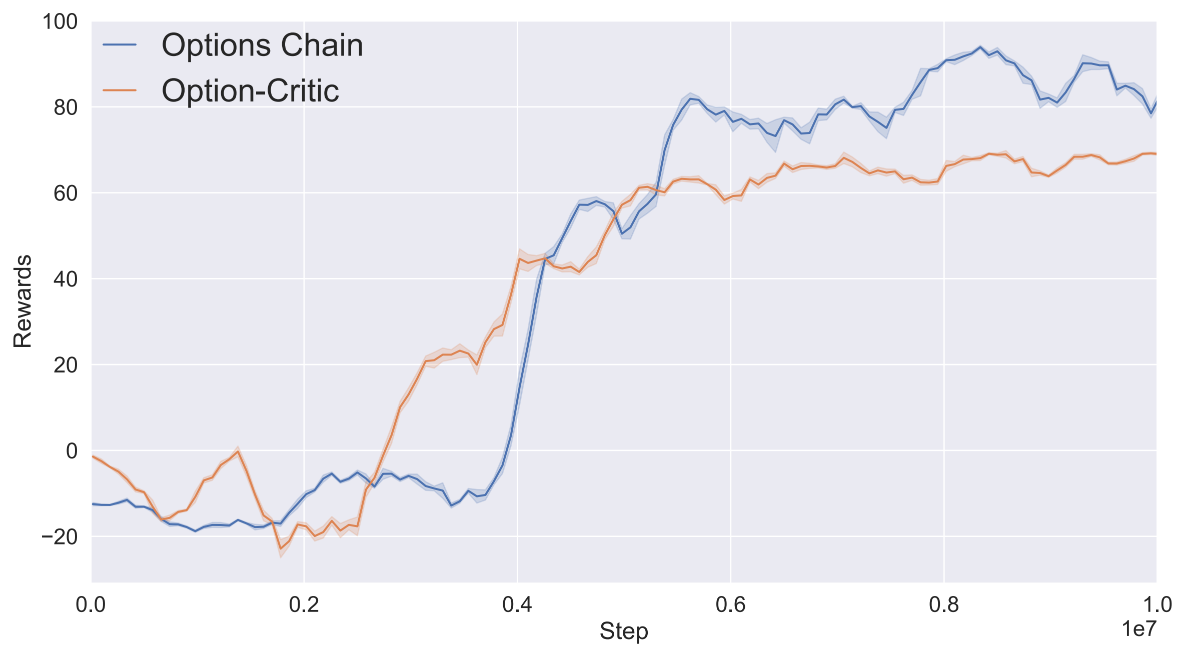
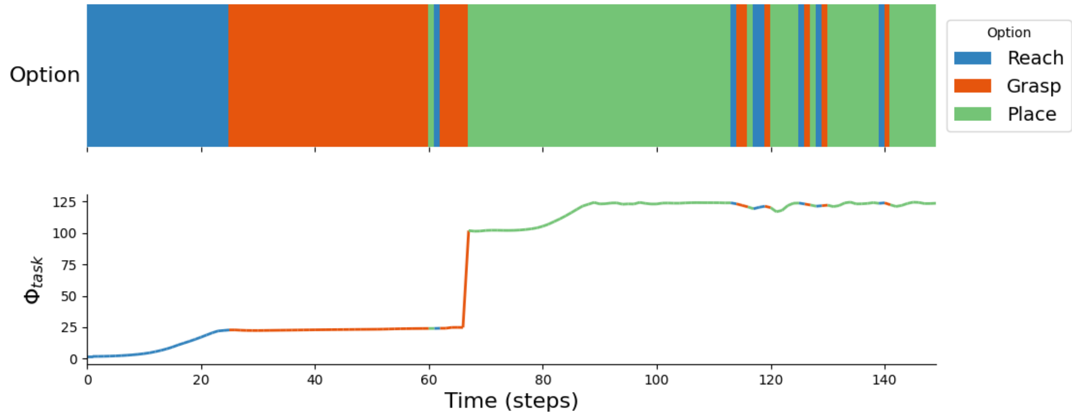
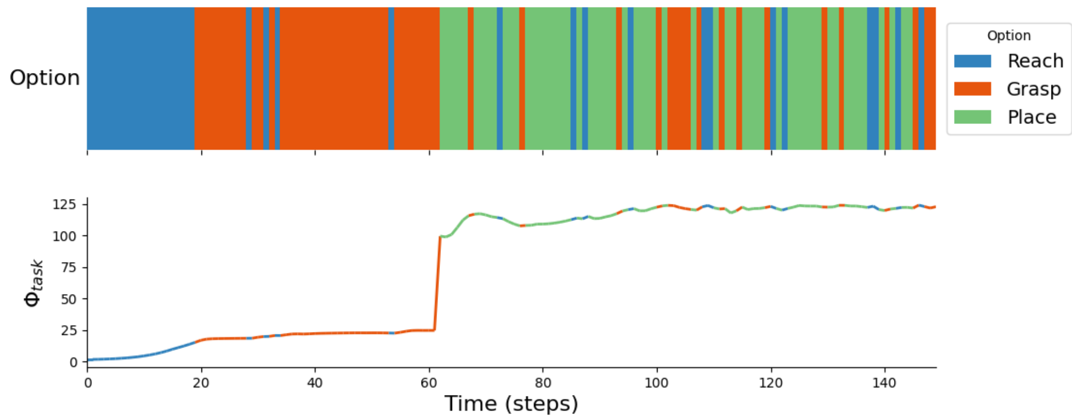


Figure 4.3: Training Plot of Rewards achieved by the Options Chain and Option-Critic Architectures for the 2D Manipulation Task



(a)



(b)

Figure 4.4: Option Usage for 2D Manipulation Task over a test episode of 150 timesteps for (a) Options Chain Architecture (b) Option-Critic Architecture

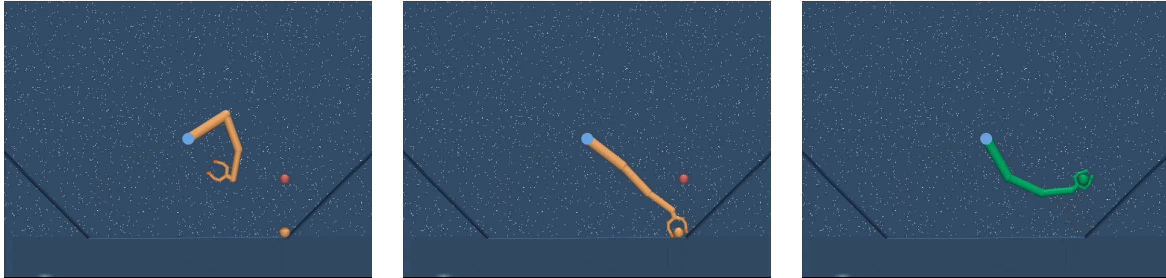


Figure 4.5: Visualization of Trained Agent Behavior for 2D Manipulator

Figure 4.6: Door environment from the robosuite [55] framework

4.2 Door Opening Task

To demonstrate the application of our work to a more complicated domain; the Kinova® Jaco™, a redundant 7-DOF manipulator, is used for the task of opening a door. The environment is obtained from the *robosuite* [55] project. A visualization of the task can be seen in Figure 4.6.

The gripper comprises of 3 fingers with 2 joints each, actuated using a single gripper torque action. In contrast to a majority of works utilizing benchmarking environments that fix the orientation of the gripper [10, 53], we allow for all 6-DOF movement (x, y, z, roll, pitch, yaw) of the end-effector. Operational Space Control (OSC) [23] is used to

output joint torques to the manipulator, taking as input the desired change in position and orientation of the end-effector. For each set of inputs specifying the desired change in the posture of the end-effector, joint torques are computed and applied in 10 evenly spaced time intervals through closed-loop control via the OSC controller.

The time limit for each episode during training is set to 500 timesteps, with no terminal states. It is to be noted that while the agent computes 500 high-level actions per episode, the OSC controller output 5000 actions, owing to a relative frequency of 10Hz for the controller with respect to the computation of the agent's actions.

4.2.1 Subtask and Reward Formulation

We decompose the task into the subtasks of reaching, grasping and pulling the door. Similar to the task potential formulation in Section 4.1.1, potential functions are defined for the reaching and pulling subtasks, and a gating condition for the grasping. Moreover, we use the same form for the potential functions, as given by Equation (4.1), with a `scale` value of 0.8.

The distance represented by `d(s)` in Equation (4.1) corresponding to the distance between the end-effector and the center of the door handle for the reaching subtask. Meanwhile, for the pulling subtask, it corresponds to the angle-to-go for the door to open to its maximum limit i.e. $d(s) = \theta_{\text{max}} - \theta$. Once again, scalar multipliers of 25 and 100 are employed to scale the ranges of the reaching and pulling subtasks respectively.

For the implementation of the gating condition for the grasping, Delaunay Tesselation is used to construct a convex hull encompassing the volume between the fingers of the end-effector. Additionally, a series of test points are generated along the center line of the handle. A successful grasp is indicated by the presence of any test points within the convex hull, and with the mean joint angles of the fingers being below a threshold of 0.55 radians. Note that the the gating conditions do not need to represent absolutely robust conditions for their associated subtasks, since the Options framework still carries the burden of determining appropriate policies and transitions between subtasks in order to maximize the task reward. However, the gating conditions should serve as a conservative guide to recognize appropriate behaviour, in order to make learning tractable.

The composite task potential and subtask reward functions share similar forms to their counterparts in Section 4.1.1, and are shown in Equation (4.4) and Equation (4.5) respectively.

$$\Phi_{\text{task}}(s) = \Phi_{\text{reach}}(s) + g_{\text{grasp}}(s)\Phi_{\text{pull}}(s) \tag{4.4}$$

$$
\begin{aligned}
R_{\text{reach}}(s,a,s') &= \Phi_{\text{reach}}(s') - \Phi_{\text{reach}}(s) \\
R_{\text{grasp}}(s,a,s') &= \left(g_{\text{grasp}}(s') - g_{\text{grasp}}(s)\right)\Phi_{\text{pull}}(s) \\
R_{\text{place}}(s,a,s') &= g_{\text{grasp}}(s')\left(\Phi_{\text{pull}}(s') - \Phi_{\text{pull}}(s)\right)
\end{aligned}
\tag{4.5}
$$

Once again, a count-based reward scheme with $\xi = 10.0$ is utilized for the purposes of improving exploration for the sparse grasping condition. The set of discretized features for count tracking is given by:

1. Distance between the end-effector and the center of the door handle, discretized in 10 evenly spaced steps on a (base-2) logarithmic scale, between 2^{-5}m and 2^{-3}m.
2. Mean joint values of the fingers of the end-effector, discretized in 10 evenly spaced steps between 0 radians and 1.51 radians.

4.2.2 Default Actions

Default actions are utilized for each of the subtasks. Recall that the action space for the environment is given by 7 continuous-values actions $\mathcal{A} = [\Delta x, \Delta y, \Delta z, \Delta ax, \Delta ay, \Delta az, \text{grip}]$, with changes in position coordinates capped between -0.05 m and 0.05 m, and changes in orientation restricted between -0.5 rad and 0.5 rad. The actions for the orientation are expressed using axis-angle representation.

For the reaching subtask, the `grip` action is fixed, applying the maximum opening torque for the gripper action. For the pulling subtask, the maximum closing torque is applied for the gripper action, for states within the feasible set of the grasping gating condition, and zero torque otherwise. Finally, for the grasping subtask, all position and orientation elements of the action space are set to zero and the maximum closing torque is specified for the gripper action.

As before, no default actions are utilized for the SAC agent.

4.2.3 State Representation and Abstractions

Observations available for the environment are listed in Table 4.10. The manipulator consists of 7 arm joints and 6 gripper joints (2 joints per finger). The elements of the state space utilized for the actors, critics and termination conditions are listed in Tables 4.11, 4.12 and 4.13 respectively.

		Size	Description
Manipulator	arm joint positions	14	$\sin(\theta)$, $\cos(\theta)$
	gripper joint positions	6	θ
	arm joint velocities	7	ω (radians/s)
	gripper joint velocities	6	ω (radians/s)
	end-effector position	3	x, y, z
	end-effector orientation	4	qx, qy, qz, qw
Door	door position	3	x, y, z
	handle position	3	x, y, z
	door angle	1	θ
Miscellaneous	door/end-effector relative position	3	x, y, z
	handle/end-effector relative position	3	x, y, z
Total		53	

Table 4.10: State Space for Door Opening Task from Robosuite [55]

	Reaching	Grasping	Pulling
arm joint positions	✓	-	✓
gripper joint positions	✗	-	✓
arm joint velocities	✓	-	✓
gripper joint velocities	✗	-	✓
end-effector position	✗	-	✗
end-effector orientation	✗	-	✗
door position	✗	-	✗
handle position	✓	-	✓
door angle	✗	-	✓
door/end-effector relative position	✗	-	✗
handle/end-effector relative position	✓	-	✓

Table 4.11: State-Space Abstractions for Actors in Door Opening Task

	Reaching	**Grasping**	**Pulling**
arm joint positions	✓	✓	✓
gripper joint positions	✓	✓	✓
arm joint velocities	✓	✓	✓
gripper joint velocities	✗	✗	✓
end-effector position	✗	✗	✗
end-effector orientation	✗	✗	✗
door position	✗	✗	✗
handle position	✓	✓	✓
door angle	✓	✓	✓
door/end-effector relative position	✗	✗	✗
handle/end-effector relative position	✓	✓	✓

Table 4.12: State-Space Abstractions for Critics in Door Opening Task

	Reaching	**Grasping**	**Pulling**
arm joint positions	✓	✗	✗
gripper joint positions	✗	✓	✓
arm joint velocities	✓	✗	✗
gripper joint velocities	✗	✓	✗
end-effector position	✗	✗	✗
end-effector orientation	✗	✗	✗
door position	✗	✗	✗
handle position	✓	✗	✗
door angle	✗	✗	✗
door/end-effector relative position	✗	✗	✗
handle/end-effector relative position	✓	✓	✓

Table 4.13: State-Space Abstractions for Termination Conditions in Door Opening Task

	Reaching	**Grasping**	**Pulling**
Hidden Layers	[200,200]	-	[200,200]
Activation Function	ReLU	-	ReLU
Learning Rate	1e-3	-	3e-4
Entropy Coefficient (α_ω)	1e-4	-	1e-3

Table 4.14: Hyperparameters for Actors in Door Opening Task

	Reaching	**Grasping**	**Pulling**
Hidden Layers	[300,200]	[300,200]	[300,200]
Activation Function	ReLU	ReLU	ReLU
Learning Rate	1e-4	1e-4	3e-5
Smoothing Constant (τ)	5e-3	5e-3	5e-3

Table 4.15: Hyperparameters for Critics in Door Opening Task

4.2.4 Hyperparameters

The hyperparameters utilized for each of the actors, critics and termination conditions are shown in Tables 4.14, 4.15 and 4.16 respectively.

The hyperparameters for the actor and critic networks of the SAC agent are shown in Tables 4.17 and 4.18 respectively.

A discount factor of $\gamma = 0.95$ is used.

	Reaching	**Grasping**	**Pulling**
Hidden Layers	[200,200]	[200,200]	[200,200]
Activation Function	ReLU	ReLU	ReLU
Learning Rate	5e-7	5e-7	5e-7
Entropy Coefficient (κ_ω)	5e-1	5e-1	5e-1

Table 4.16: Hyperparameters for Termination Conditions in Door Opening Task

Hidden Layers	[300,200]
Activation Function	ReLU
Learning Rate	3e-4
Entropy Coefficient (α_ω)	1e-3

Table 4.17: Hyperparameters for SAC Actor in Door Opening Task

Hidden Layers	[300,200]
Activation Function	ReLU
Learning Rate	3e-5
Smoothing Constant (τ)	5e-3

Table 4.18: Hyperparameters for SAC Critic in Door Opening Task

4.2.5 Results

We present results analogous to those in Section 4.1.5 for the Door Opening task. The training plots report means and standard deviations computed over 5 evaluation episodes.

The maximum composite task potentials observed over training under the Options Chain, Option-Critic and SAC agents are shown in Figure 4.7. We observe that all agents are able to achieve the same asymptotic performance on this task, once again indicating that our options-based reward formulation incentivizes the agent to solve the composite task.

With regards to sample efficiency, this time we notice that our options-based agents are able to make considerably faster progress in solving the task as compared to the non-hierarchical SAC agent. This is especially encouraging since the SAC agent directly optimizes for the composite task potential, unlike the options-based agents which also incur penalties due to behavior specialization.

Next, evaluating the explainability of the options-based agents, there is a marked difference in the returns experienced by both models, as seen in Figure 4.8. The Options Chain model appears to become more robust over training, generating consistently high returns towards the end of the training regime. However, the Option-Critic model shows large variance in its returns. While indicating an ability to solve the task (through Figure 4.7), it can be inferred that the large variance is due to errant behaviors, that either do not get rewarded or get penalized.

To exemplify this, we evaluate the final trained agents under both architectures, over

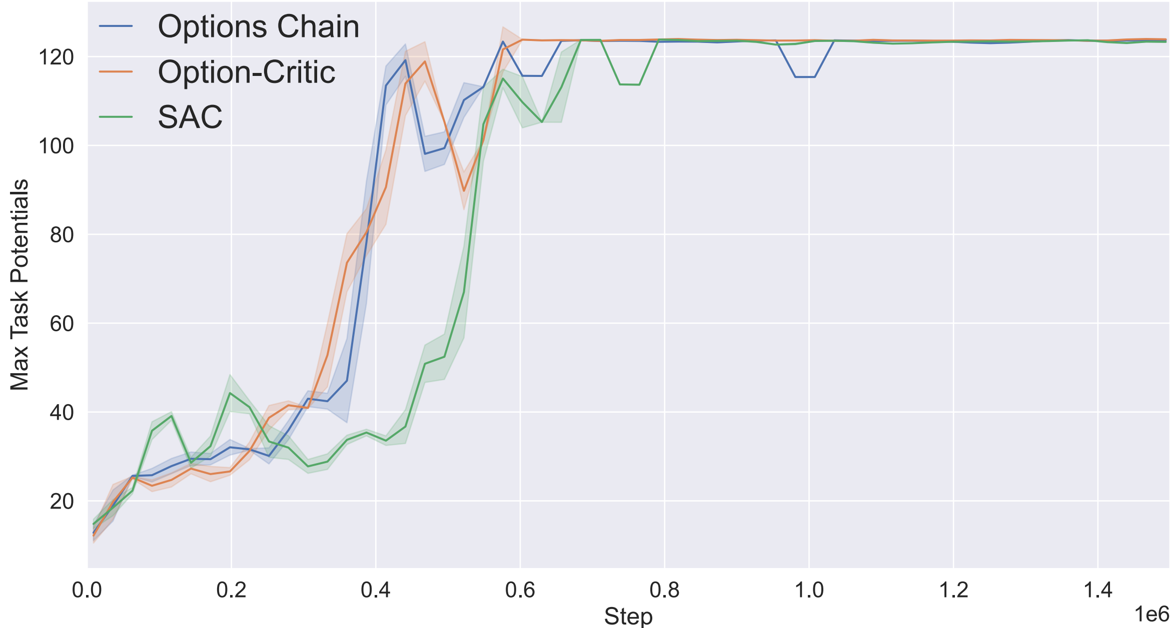


Figure 4.7: Training Plot of Maximum Task Potentials achieved by the Options Chain, Option-Critic and SAC Agents for the Door Opening Task

test episodes of 50 timesteps, with results depicted in Figure 4.9. The Options Chain model shows a perfect partitioning of the task, with options being appropriately utilized. However, the Option-Critic model displays large variability in its option usage. There is evidence of errant behaviors such as the act of grasping being realized by the pulling option (signified by the near step-like change at $t = 28$ in Figure 4.9b).

4.3 Discussion

For both tasks, we have observed that the options-based agents are appropriately incentivized to solve the composite task, achieving similar asymptotic performance as the SAC agents that directly optimize for the composite task potentials, and thus providing validation to the formulation of our options-based reward scheme. Additionally, this also demonstrates the ability of our algorithm, given by Algorithm 5, to learn under the Options Framework [47] and for both option execution models.

Speaking to improvements in sample efficiency, one of the goals of our work motivating the use of a hierarchical learning scheme, we noticed little gains in the 2D manipulation

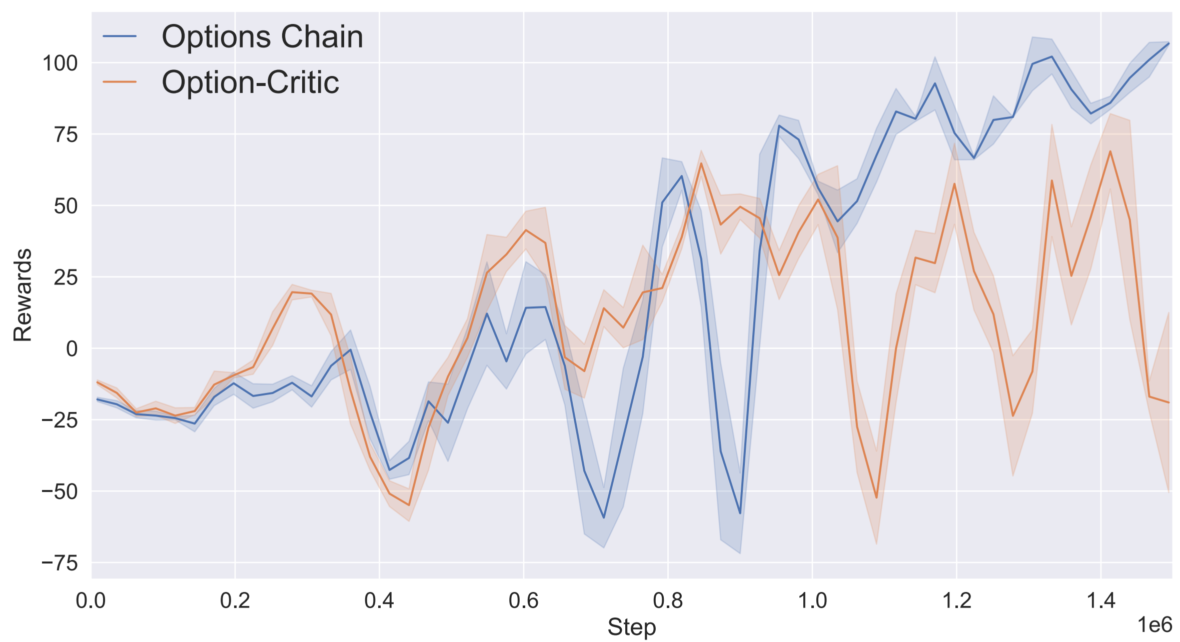
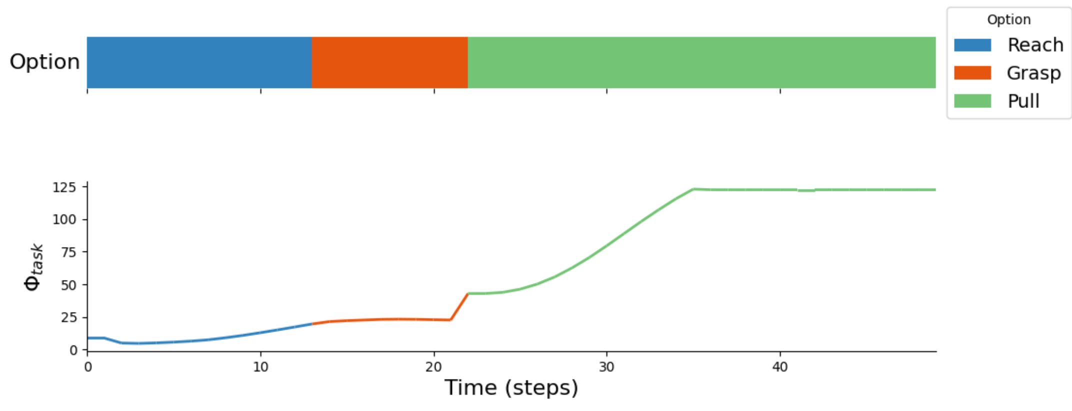
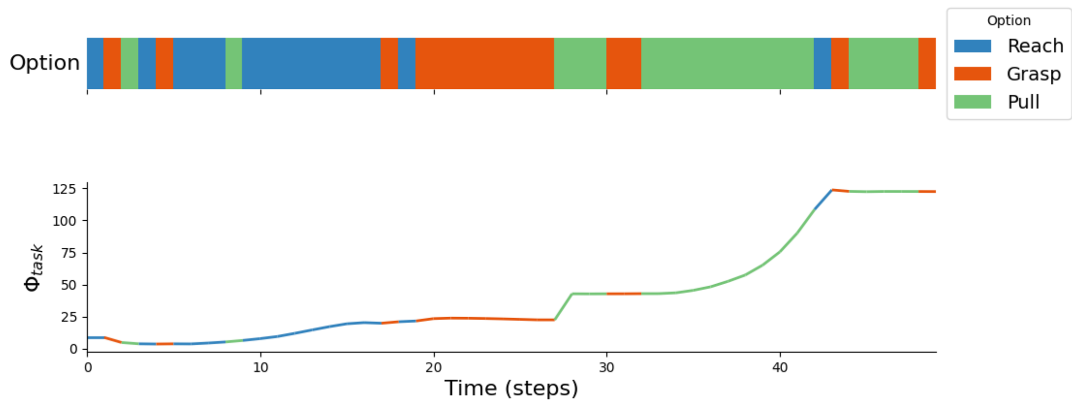


Figure 4.8: Training Plot of Rewards achieved by the Options Chain and Option-Critic Architectures for the Door Opening Task



(a)



(b)

Figure 4.9: Option Usage for Door Opening Task over a test episode of 50 timesteps for (a) Options Chain Architecture (b) Option-Critic Architecture

Figure 4.10: Visualization of Trained Agent Behavior for Door Opening Task

task and significant improvements for the door opening task. It must be noted that the benefits of hierarchy to sample-efficiency are task-dependent, as also observed in [29]. The characteristics of tasks in which hierarchy proves beneficial needs further investigation. We conjecture that since the SAC agent uses a single actor and critic network for the entire task, with all of its environmental interactions contributing to the training of these networks, it is able to implicitly leverage commonalities in learning across subtasks (such as for the reaching and placing subtasks). This is not a feature of our options-based works, since the neural networks associated with a given subtask are only trained using the environmental interactions observed under the associated option use. This suggests a potential avenue for investigation into shared learning across options for future works.

Addressing our goal of explainable RL agents next, we saw that both option execution models (denoted by the Options Chain and Option-Critic Architecture) learning under our options-based reward scheme, resulted in the development of specialized subtask behaviors which combine to carry out a given task in a largely intuitive way as depicted in Figures 4.4 and 4.9.

Speaking to the nature of the tasks, the door opening task is more vulnerable to catastrophic failures as a result of wrong options and/or actions being utilized. To illuminate this point, consider the scenario where the door handle is in the grasp of the gripper. Pulling the end-effector in the wrong direction can lead to losing the grasp. Contrasting this with the 2D manipulation task from the DeepMind Control Suite [50], with the ball contained within the gripper, a loss of grasp cannot happen for any movement of the arm joints (as long as the gripper stays closed). This contributes to the large variances in returns observed in Figure 4.8.

As mentioned in Section 3.2, the nature of the sequential option model represented by the Options Chain, requires the agent to traverse through the entire chain to re-engage with any given option after termination. It was hypothesized that this should contribute

to robustness in learning of the termination conditions, which dictates the utilization of options. Meanwhile, the Option-Critic model can jump back to using any option in a single step, allowing it more flexibility in intermittently engaging other options in the chain with reduced risk of producing catastrophic failures across the task through occasionally utilizing *wrong* options. This line of thought is corroborated by the more pronounced segregations of utilized options for both tasks under the Options Chain model, where there is a greater risk of losing progress across the task through undesired terminations of options, thereby producing a more explainable RL agent.

Chapter 5

Conclusion

We conclude this thesis with a summary of the proposed developments and results, and a consideration of possible avenues for future research.

5.1 Summary

In this thesis, we set out to leverage the use of a particular Hierarchical Reinforcement Learning approach, known as the Options Framework [47], for learning sequential robotic tasks, with the combined goal of improving the sample efficiency (as a by-product of hierarchical learning) and explainability of the resulting RL agent(s).

The Options Framework [47] has previously demonstrated potential in improving sample efficiency in learning [47, 4] and to produce distinct temporally-extended behaviors [31, 5, 20, 21]. However, previous approaches to the latter have been marked by heuristic modifications to option learning algorithms, in order to enforce distinct behavior learning. Furthermore, distinct behaviors from the perspective of the agent may not generally be visually meaningful or recognizable by humans [21].

In our work, we seek to promote interpretable behavior learning through employing task abstraction at the subtask level. Additionally, we aim to keep the objective of distinct behavior learning consistent with the primary goal of solving the composite task. To this end, we have proposed a novel, potential-based reward scheme that can be decomposed to produce option-level rewards, which in turn promote behavior specialization and incentivize the agent to solve the composite sequential task while operating under the standard

reinforcement learning objective of maximizing returns. Developing and utilizing an off-policy Maximum Entropy Deep Reinforcement Learning algorithm, inspired by the SAC [18] algorithm, we showcase the ability of the agent to accomplish both these goals.

From the perspective of tackling tasks comprising of a sequential composition of subtasks, we submit the Options Chain, a sequential option execution model that is shown to yield improved behavior specialization in comparison to the Option-Critic model. Additionally, we make use of the idea of expert-specified action spaces for each option to reduce the burden of exploration and make learning tractable.

In validating our work, we present results using simulated experiments for a pick-and-place and a door opening task, using manipulator environments with high-dimensional state and action spaces obtained from the DeepMind Control Suite [50] and the Robosuite [55] projects respectively.

5.2 Future Work

A possible avenue of future work could involve research into other forms of reward functions for learning at the subtask-level. As previously discussed in Section 3.3, our reward formulation requires adherence to a set of assumptions which can limit the nature of tasks that may be attempted under our approach.

Another line of research can involve investigation into improving the sample complexity of learning through re-using interactions experienced under a specific option to derive learning for other options. As an example, consider the reaching and placing subtasks in Section 4.1. Since both subtasks require moving the manipulator towards a given point (to the ball in the former case and towards the goal point in the latter case) using the same action space, there is a potential for shared learning in both subtasks using their combined experiences.

Additionally, application of this work to a real robotic platform and comparison of sim-to-real policy transfer [54] performance with an agent trained under this work versus a monolithic agent represented by a single "black box" policy, will likely provide some useful insights.

References

[1] Josh Achiam. Openai spinning up. https://spinningup.openai.com/en/latest/algorithms/ppo.html#pseudocode, 2018.

[2] Josh Achiam. Openai spinning up. https://spinningup.openai.com/en/latest/algorithms/sac.html#pseudocode, 2018.

[3] Arthur Allshire, Roberto Martín-Martín, Charles Lin, Shawn Manuel, Silvio Savarese, and Animesh Garg. Laser: Learning a latent action space for efficient reinforcement learning, 2021.

[4] Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture, 2016.

[5] P.L. Bacon. *On the Bottleneck Concept for Options Discovery: Theoretical Underpinnings and Extension in Continuous State Spaces*. McGill theses. McGill University Libraries, 2014.

[6] A. Bagaria and G. Konidaris. Option discovery using deep skill chaining. In *ICLR*, 2020.

[7] Marc G. Bellemare, Sriram Srinivasan, Georg Ostrovski, Tom Schaul, David Saxton, and Remi Munos. Unifying count-based exploration and intrinsic motivation, 2016.

[8] Richard Bellman. Adaptive control processes: A guided tour. (A RAND Corporation Research Study). Princeton, N. J.: Princeton University Press, XVI, 255 p. (1961)., 1961.

[9] Benjamin Beyret, Ali Shafti, and A. Aldo Faisal. Dot-to-dot: Explainable hierarchical reinforcement learning for robotic manipulation. *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov 2019.

[10] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym, 2016.

[11] Zhixin Chen and Mengxiang Lin. Self-imitation learning for robot tasks with sparse and delayed rewards, 2021.

[12] Peter Dayan and Geoffrey E Hinton. Feudal reinforcement learning. In S. Hanson, J. Cowan, and C. Giles, editors, *Advances in Neural Information Processing Systems*, volume 5. Morgan-Kaufmann, 1993.

[13] Thomas G. Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *J. Artif. Int. Res.*, 13(1):227–303, November 2000.

[14] Mengnan Du, Ninghao Liu, and Xia Hu. Techniques for interpretable machine learning, 2019.

[15] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018.

[16] Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning, 2019.

[17] David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

[18] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[19] Danijar Hafner, Timothy Lillicrap, Jimmy Ba, and Mohammad Norouzi. Dream to control: Learning behaviors by latent imagination, 2020.

[20] Jean Harb, Pierre-Luc Bacon, Martin Klissarov, and Doina Precup. When waiting is not an option : Learning options with a deliberation cost, 2017.

[21] Anna Harutyunyan, Will Dabney, Diana Borsa, Nicolas Heess, Remi Munos, and Doina Precup. The termination critic, 2019.

[22] Bernhard Hengst. *Hierarchical Reinforcement Learning*, pages 495–502. Springer US, Boston, MA, 2010.

[23] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE Journal on Robotics and Automation*, 3(1):43–53, 1987.

[24] Martin Klissarov, Pierre-Luc Bacon, Jean Harb, and Doina Precup. Learnings options end-to-end for continuous action tasks, 2017.

[25] George Konidaris and Andrew Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In Y. Bengio, D. Schuurmans, J. Lafferty, C. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, volume 22, pages 1015–1023. Curran Associates, Inc., 2009.

[26] Sanjay Krishnan, Animesh Garg, Richard Liaw, Brijen Thananjeyan, Lauren Miller, Florian T Pokorny, and Ken Goldberg. Swirl: A sequential windowed inverse reinforcement learning algorithm for robot tasks with delayed rewards. *The International Journal of Robotics Research*, 38(2-3):126–145, 2019.

[27] Sanjay Krishnan, Animesh Garg, Sachin Patil, Colin Lea, Gregory Hager, Pieter Abbeel, and Ken Goldberg. Transition state clustering: Unsupervised surgical trajectory segmentation for robot learning. *The International Journal of Robotics Research*, 36(13-14):1595–1618, 2017.

[28] Andrew Levy, George Konidaris, Robert Platt, and Kate Saenko. Learning multi-level hierarchies with hindsight, 2019.

[29] Chenghao Li, Xiaoteng Ma, Chongjie Zhang, Jun Yang, Li Xia, and Qianchuan Zhao. Soac: The soft option actor-critic architecture, 2020.

[30] Elita Lobo and Scott Jordan. Soft options critic, 2019.

[31] Amy McGovern and Andrew G. Barto. Automatic discovery of subgoals in reinforcement learning using diverse density. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, page 361–368, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[32] Matheus R. F. Mendonça, Artur Ziviani, and AndrÉ M. S. Barreto. Graph-based skill acquisition for reinforcement learning. *ACM Comput. Surv.*, 52(1), February 2019.

[33] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen

King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015.

[34] Michael Neunert, Abbas Abdolmaleki, Markus Wulfmeier, Thomas Lampe, Jost Tobias Springenberg, Roland Hafner, Francesco Romano, Jonas Buchli, Nicolas Heess, and Martin Riedmiller. Continuous-discrete reinforcement learning for hybrid control in robotics, 2020.

[35] Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, page 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[36] Doina Precup and R. Sutton. Temporal abstraction in reinforcement learning. In *ICML 2000*, 2000.

[37] Erika Puiutta and Eric MSP Veith. Explainable reinforcement learning: A survey, 2020.

[38] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., USA, 1st edition, 1994.

[39] J. Randløv and P. Alstrøm. Learning to drive a bicycle using reinforcement learning and shaping. In *ICML*, 1998.

[40] Stefan Schaal. *Dynamic Movement Primitives -A Framework for Motor Control in Humans and Humanoid Robotics*, pages 261–280. Springer Tokyo, Tokyo, 2006.

[41] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017.

[42] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[43] Alexander L. Strehl and Michael L. Littman. An analysis of model-based interval estimation for markov decision processes. *Journal of Computer and System Sciences*, 74(8):1309–1331, 2008. Learning Theory 2005.

[44] Freek Stulp and Stefan Schaal. Hierarchical reinforcement learning with movement primitives. In *2011 11th IEEE-RAS International Conference on Humanoid Robots*, pages 231–238, 2011.

[45] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, second edition, 2018.

[46] Richard S. Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, page 1057–1063, Cambridge, MA, USA, 1999. MIT Press.

[47] 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.

[48] Haoran Tang, Rein Houthooft, Davis Foote, Adam Stooke, Xi Chen, Yan Duan, John Schulman, Filip De Turck, and Pieter Abbeel. #exploration: A study of count-based exploration for deep reinforcement learning, 2017.

[49] Yunhao Tang and Shipra Agrawal. Discretizing continuous action space for on-policy optimization, 2020.

[50] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, Timothy Lillicrap, and Martin Riedmiller. Deepmind control suite, 2018.

[51] Alexander Sasha Vezhnevets, Simon Osindero, Tom Schaul, Nicolas Heess, Max Jaderberg, David Silver, and Koray Kavukcuoglu. Feudal networks for hierarchical reinforcement learning, 2017.

[52] R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

[53] Tianhe Yu, Deirdre Quillen, Zhanpeng He, Ryan Julian, Avnish Narayan, Hayden Shively, Adithya Bellathur, Karol Hausman, Chelsea Finn, and Sergey Levine. Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning, 2021.

[54] Wenshuai Zhao, J. P. Queralta, and T. Westerlund. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 737–744, 2020.

[55] Yuke Zhu, Josiah Wong, Ajay Mandlekar, and Roberto Martín-Martín. robosuite: A modular simulation framework and benchmark for robot learning. In *arXiv preprint arXiv:2009.12293*, 2020.

[56] Brian D. Ziebart. Modeling purposeful adaptive behavior with the principle of maximum causal entropy, Jul 2018.