On Hierarchical Goal Based Reinforcement Learning

Nicholas Denis

Thesis submitted to the Faculty of Science in partial fulfillment of the requirements
for the degree of
Master of Science Mathematics and Statistics[1]

Department of Mathematics and Statistics
Faculty of Science
University of Ottawa

[1]The M.Sc. program is a joint program with Carleton University, administered by the Ottawa-Carleton Institute of Mathematics and Statistics

Abstract

Discrete time sequential decision processes require that an agent select an action at each time step. As humans, we plan over long time horizons and use temporal abstraction by selecting temporally extended actions such as "make lunch" or "get a masters degree", whereby each is comprised of more granular actions. This thesis concerns itself with such hierarchical temporal abstractions in the form of macro actions and options, as they apply to goal-based Markov Decision Processes. A novel algorithm for discovering hierarchical macro actions in goal-based MDPs, as well as a novel algorithm utilizing landmark options for transfer learning in multi-task goal-based reinforcement learning settings are introduced. Theoretical properties regarding the life-long regret of an agent executing the latter algorithm are also discussed.

Dedications

Marcus,

Don't ever be afraid of making mistakes.

Acknowledgment

Thanks to Epicurus for starting the conversation, to Blackwell for revealing the absurdity of finding the *greenest* greener grass, and to Auer for showing us the necessity of regret in order to live a good life.

Outline of Contributions

This thesis is centered around temporal abstraction and hierarchies. Though we live as embodied agents acting continuously in time, we make decisions at discrete time steps that invoke *chunked* behaviors that are reusable through our decision making processes (a.k.a. life). To that end, this thesis introduces two novel algorithms for hierarchical control and temporal abstraction for solving goal based Markov Decision Processes. Namely,

- Chapter 2 includes a manuscript submitted, though not accepted, to the workshop on Hierarchical Reinforcement Learning at the 2017 Neural Information Processing Systems Conference. This manuscript introduces a novel algorithm, Hierarchical Macro-Action Chaining (HMAC), for discovering and reusing *chunked* sequences of primitive actions in the form of macro-actions. Experimental results in multiple domains (tabular and function approximation using deep convolutional neural networks) demonstrate empirically that HMAC leads to a speed-up in learning applied to goal based reinforcement learning settings.

- Chapter 3 includes an accepted manuscript submitted to the 2019 Canadian Artificial Intelligence Conference, published in the CAIC proceedings, and was selected for an oral presentation. A preliminary version of this work was also accepted at the aforementioned Hierarchical Reinforcement Learning workshop at the 2017 NIPS conference, and was selected for an oral presentation. This work introduces a novel algorithm, Landmark Options Via Reflection (LOVR). LOVR is a flexible solution framework for multi-task goal based reinforcement learning settings, and Chapter 3 covers two different instantiations of the algorithm. Theoretical results on regret bounds are provided for two different settings, as well as an example that demonstrates that one of the regret bounds is tight. This work also introduces a novel type of covering of the state space in a Markov Decision Process, that of η-reachability. Experimental results in multiple domains is provided, demonstrating empirically that LOVR can drastically reduce life-long regret in multi-task reinforcement learning domains over baseline methods.

Finally, a proposition in Section 1.3 characterizing the optimal value function as

the expected first hitting time of a goal state in goal-based Markov Decision Processes is provided.

Contents

List of Figures **ix**

1 Introduction **1**
1.1 MDP . . . 2
1.1.1 Optimality in MDPs . . . 4
1.1.2 Bellman Operators . . . 5
1.2 Algorithmic Solutions to MDPs . . . 8
1.2.1 Value Iteration . . . 8
1.2.2 Q-learning . . . 9
1.2.3 Performance Measures for Algorithmic Solutions . . . 12
1.2.4 Deep Q Network and Function Approximation . . . 13
1.2.5 DQN . . . 15
1.3 Goal Based MDPs and Multi-Task RL . . . 16
1.3.1 Goal Based MDPs . . . 16
1.3.2 Multi-Task RL . . . 18
1.4 Hierarchical RL and Temporally Extended Actions . . . 18
1.4.1 Temporally Extended Actions . . . 18
1.4.2 Macro Actions . . . 19
1.4.3 Options Framework . . . 19
1.4.4 Landmark Options . . . 20

2 Hierarchical Macro Action Chaining (HMAC) **21**

3 Landmark Options Via Reflection (LOVR) **27**

4 Conclusion **40**

A **42**
A.1 HMAC: Appendix . . . 42
A.1.1 HMAC Pseudo Code . . . 42
A.1.2 Experimental Methods . . . 43
A.1.3 Hierarchical Macro Action Analysis . . . 43

A.2 LOVR: Appendix 45
A.2.1 Pseudo Code 45
A.2.2 Theoretical Results 48
A.2.3 Experimental Methods 50
A.2.4 Landmark Analysis 50

Bibliography **56**

List of Figures

1.1 Reinforcement Learning Overview [28] 2

A.1 HMAC policy at episode 50 (top left) (regret = 13), episode 200 and 250 (top middle) (regret = 13), episodes 500 (top right) (regret = 10), episodes 700,750 (bottom left) (regret = 8), episodes 900,950,1000 (bottom right) (regret =4). Green square signifies start state, red square denotes goal state. Arrows denote actions taken. Lightbulbs denote states where the policy made a decision. 44
A.2 Number of macro actions learned over time (n=10 experimental repeats). 45
A.3 LOVR Gridworld layout with (Left): 9 landmark locations; (Right): 25 landmark locations. 50
A.4 (Top) Landmark centric value functions and (bottom) policies learned from during phase I from landmarks 1,5,9. 0=Up, 1=Right, 2 = Down, 3=Left 51
A.5 (Top) Landmark centric value functions and (bottom) policies learned from during phase I from landmarks 6,12,18. 0=Up, 1=Right, 2 = Down, 3=Left 52
A.6 (Top) Landmark centric value functions and (bottom) policies learned from during phase I from landmarks 1,2,3 . 0=Up, 1=Down, 2 = Right, 3=Left 52
A.7 (Top) Landmark centric value functions and (bottom) policies learned from during phase I from landmarks 4,5,6. 0=Up, 1=Down, 2 = Right, 3=Left 53
A.8 (Top) Landmark centric value functions and (bottom) policies learned from during phase I from landmarks 7,8. 0=Up, 1=Down, 2 = Right, 3=Left 54

Chapter 1

Introduction

> *"For we know pleasure as our first and inborn good, and we have it as the beginning of our every choice and avoidance, and we turn to it in using sensation as a standard for judging every good. And since pleasure is our first and natural good, for this reason we do not choose every pleasure, but there are times when we pass over many pleasures, when greater difficulty would result from them for us, and we consider many pains to be better than pleasures, whenever a greater and long-lasting pleasure will follow for us after we have suffered the pains. So every pleasure is a good through being naturally agreeable, nevertheless not every pleasure is to be chosen; just as it is also the case that every pain is bad, but not every pain is always by its nature to be avoided."*
>
> - Epicurus, 341-270 B.C.,

This thesis is concerned with *Reinforcment Learning* (RL) as it pertains to sequential decision processes [28], specifically those that are goal oriented or *goal based*. An actor or *agent* finds itself in an *environment*, equipped with a set of *actions* that it can take. At discrete time intervals, $t = 1, 2, 3, ...$, the agent takes an action based on the current *state* of the environment, which results in the environment evolving to a new successor state, and the agent receiving a scalar *reward* representing some sense of utility. This interaction between the agent and the environment is represented in Figure 1.

This chapter will begin by introducing the mathematical framework used for solving RL problems, namely Markov Decision Processes (MDPs). Notions of optimality within MDPs is addressed, as well as some algorithmic solution frameworks. Goal based MDPs and multi-task RL is introduced, followed by the discussion of relevant notions of hierarchy and temporal abstraction.

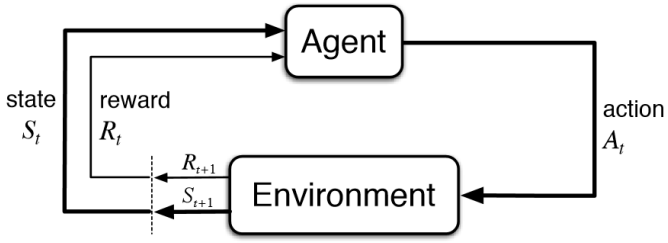

Figure 1.1: Reinforcement Learning Overview [28]

1.1 MDP

Definition 1.1.1. *(Markov Decision Process [24]) A* ***Markov Decision process*** *is a tuple,* $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, p_0, \gamma \rangle$*, where*

- $\mathcal{S}$ *is the set of states of the environment. This thesis assumes* $|\mathcal{S}| < \infty$*.*
- $\mathcal{A}$ *is the set of actions which the agent has access to. This thesis assumes* $|\mathcal{A}| < \infty$*.*
- $P := \{P^a\}_{a \in \mathcal{A}}$*, is the dynamics of the environment.* $\forall a \in \mathcal{A}$*,* P^a *is the state transition kernel associated to action* a*. Under the above assumptions,* $P^a \in [0,1]^{\mathcal{S} \times \mathcal{S}}$ *is a stochastic* $|\mathcal{S}| \times |\mathcal{S}|$ *transition matrix, where* $P^a(i,j)$ *is the probability of the environment transitioning to state* j *when action* a *is taken from state* i*,*
- $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$*, is the reward model of the environment that is a function of a transition event* (s, a, s')*, which represents the event that action* a *was taken in state* s *and resulted in the environment transitioning to the successor state* s'*. Typically* R *is assumed to be bounded (i.e.* $\exists M < \infty$*, such that* $|R(s,a,s')| \leq M$*,* $\forall (s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$*).*
- p_0 *is the distribution over initial states that the environment/agent starts in.*
- $\gamma \in [0,1]$ *is the discount-factor.*

The MDP framework receives its name from the fact (assumption) that the environment dynamics are Markovian. That is,

$$\mathbb{P}(s_{t+1}|a_t, s_t, a_{t-1}, s_{t-1}, ..., a_1, s_1) = \mathbb{P}(s_{t+1}|a_t, s_t),$$

where s_k, a_k denotes the state $s \in \mathcal{S}$ and action $a \in \mathcal{A}$ taken at time k. This property is assumed to be true $\forall s \in \mathcal{S}, a \in \mathcal{A}$ and $\forall t \in \mathbb{N}$. In order to consider the objective function of an algorithm or agent acting within an MDP, we first introduce the notion of a *policy* and of a *return*.

We say that an MDP is **communicating** if $\forall s, s' \in \mathcal{S}$, $\exists \pi, N > 0$, such that $\mathbb{P}(s_N = s | s_0 = s', \pi) > 0$. That is, an MDP is communicating if there is a non-zero probability of reaching any given state from any other given state in a finite number of time steps under *some* policy. In this thesis we assume that all MDP environments considered have the communicating property.

Definition 1.1.2. *(Policy) A **policy** can either be a deterministic function, mapping states to actions*

$$\pi : \mathcal{S} \to \mathcal{A},$$

or stochastic, mapping states to a distribution over actions

$$\pi : \mathcal{S} \to \Delta(\mathcal{A}).$$

Policies can be seen as behaviours or strategies, and encode how the algorithm will make decisions at each time step. Note that nothing in this framework is time dependent. The MDP framework assumes that the dynamics of the environment or *stationary*, in that $P(s_{t+1} = s | a_t = a, s_t = u) = P(s_{t'+1} = s | a_{t'}, s_{t'} = u)$, $\forall t \neq t'$. Moreover, the policy is assumed to also be stationary, in that for a fixed s for every time indices $t \neq t'$, $\pi(s_t) = \pi(s_{t'})$. A note on notation: for stochastic policies, $\pi(a|s)$ is used to represent the probability of taking action a under policy π at state s. Hence, $\forall s \in \mathcal{S}$, $\sum_{a \in \mathcal{A}} \pi(a|s) = 1$.

Definition 1.1.3. *(Returns) For a given $T \in \mathbb{N}$, the **finite horizon return** indexed at time $t \in \mathbb{N}$, $G_t := R_t + R_{t+1} + R_{t+2} + ... + R_{t+T}$, is the random variable representing the random sum of rewards over the next T time steps. The **infinite horizon return** indexed at time $t \in \mathbb{N}$, $G_t := R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} + \gamma^3 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k}$, is the random variable representing the random sum of discounted rewards over the infinite future for $\gamma \in [0, 1)$.*

Typically for *episodic* domains where the agent interacts with the environment for a (possibly fixed) finite number of time steps, the finite horizon return is used. Settings where the agent is meant to interact with the environment indefinitely, or $T >> 1$, then the infinite horizon setting is used. For the sake of a unified notation, we will allow $T = \infty$, and $\gamma \in [0, 1]$, then we will use the following notation of

$$G_t := \sum_{k=0}^{T} \gamma^k R_{t+k}.$$

When studying MDPs, the objective is for the algorithm or agent to take a sequence of actions which aim to maximize the expected value of G_t, where the expectation is taken with respect to both the environment dynamics, and the (possible stochastic) action selection strategy. Note that stating G_t and R_t are random variables only makes sense when considering a fixed policy (or distribution over policies). Hence, for the above definitions, we are implicitly assuming that these are random variables induced by a given policy (and of course the environment dynamics).

The final key players in the MDP and RL framework are the *value functions*. Value functions are defined with respect to a fixed policy, and encode "how good" a given state or state-action pair is with respect to maximizing G_t.

Definition 1.1.4. *(Value Functions) The* ***state value function****, with respect to a policy* π*,* $V^\pi : \mathcal{S} \to \mathbb{R}$*, is given as*

$$V^\pi(s) := \mathbb{E}_\pi\Big[G_t | s_t = s\Big].$$

The ***state-action value function****, with respect to a policy* π*,* $Q^\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$*, is given as*

$$Q^\pi(s,a) := \mathbb{E}_\pi\Big[G_t | s_t = s, a_t = a\Big].$$

It is worth noting that $Q^\pi(s,a)$ represents the value of taking action a in state s *just once*, but then forever afterwards following policy π, since it is not necessarily the case that $\pi(s) = a$.

1.1.1 Optimality in MDPs

There exist several different notions of optimality within the MDP literature, each with their own merit and consequences. It is beyond the scope of this thesis to introduce and address each, and for this reason we concern ourselves with the most common notions of optimality [24, 28]. For γ fixed, we consider a policy, π^*, **optimal** if:

$$\forall s \in \mathcal{S}, \pi^*(s) \in \underset{\pi \in \Delta(\mathcal{A})^{\mathcal{S}}}{arg\ max}\ \ V^\pi(s), \quad \textit{where}\ \ V^\pi \in \mathbb{R}^{\mathcal{S}}.$$

Hence, $\forall s \in \mathcal{S}$, $V^{\pi^*}(s) \geq V^\pi(s)$, $\forall \pi$. Moreover, we define $V^* := V^{\pi^*}$ and use the two interchangeably. Note that Q^* is defined similarly. Interestingly, under π^* the state value function and the state-action value function are related in that,

$$V^*(s) = \underset{a \in \mathcal{A}}{max}\ \ Q^*(s,a)$$

since,

$$\begin{aligned} V^*(s) &= V^{\pi^*}(s) \\ &= \mathbb{E}_{\pi^*}\Big[G_t | s_t = s\Big] \\ &= \mathbb{E}_\pi\Big[G_t | s_t = s, a_t = \pi^*(s_t)\Big] \\ &= Q^{\pi^*}(s, \pi^*(s)) \\ &= Q^*(s, \pi^*(s)) \end{aligned}$$

$$= \max_{a \in \mathcal{A}} \; Q^*(s, a)$$

A notion that describes, in some sense, a policy or value function as being "close enough" to being optimal will also be relevant for this thesis. For $\epsilon > 0$, γ fixed, a policy, π, (or value function) is **ϵ-optimal** if,

$$\max_{s \in \mathcal{S}} \; V^*(s) - V^\pi(s) < \epsilon$$

As hinted at in the definition of ϵ-optimality, since a value function is always defined with respect to a fixed policy, the two terms will often be used interchangeably, sometimes addressing the value function, or the value of a policy, for example by saying that either a value function as being (ϵ-) optimal, or a policy as being (ϵ-) optimal. We will return later to concepts akin to optimality with respect to algorithmic solutions for arriving at optimal policies, whereby we describe measures of performance for the algorithmic solutions. Before this, we will introduce some key mathematical machinery that motivates all the algorithmic solution approaches relevant to this thesis.

1.1.2 Bellman Operators

A closer inspection of V^* reveals that

$$\begin{aligned}
V^*(s) &= \max_{a \in \mathcal{A}} \; Q^{\pi^*}(s, a) \\
&= \max_{a \in \mathcal{A}} \; \mathbb{E}_{\pi^*}\Big[\sum_{k=0}^{T} \gamma^k r_{t+k} | s_t = s, a_t = a \Big] \\
&= \max_{a \in \mathcal{A}} \; \mathbb{E}_{\pi^*}\Big[r_t + \gamma \sum_{k=0}^{T} \gamma^k r_{t+k+1} | s_t = s, a_t = a \Big] \\
&= \max_{a \in \mathcal{A}} \; \sum_{s' \in \mathcal{S}} P(s'|s, a) \Big[R(s, a, s') + \gamma \mathbb{E}_{\pi^*}[G_{t+1} | s_t = s, a_t = a] \Big] \\
&= \max_{a \in \mathcal{A}} \; \sum_{s' \in \mathcal{S}} P(s'|s, a) [R(s, a, s') + \gamma V^*(s')] \quad (1.1.1)
\end{aligned}$$

Equation (1.1.1) shows the recursive nature of the optimal value function, in that, the value at the current state is realized by taking the action that maximizes the expected immediate reward plus the discounted expected value of the expected successor state. (1.1.1) plays an important role in solution approaches to MDPs, and is called the **Bellman Optimality Equation**.

Assuming $\exists M < \infty$, such that $\forall (s, a, s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$, $|R(s, a, s')| \leq M$, then clearly $\forall \pi,\ s$, and for $T = \infty, \gamma \in [0, 1)$, we see that:

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k}] | s_t = s] \\ &\leq \sum_{k=0}^{\infty} \gamma^k M \\ &= \frac{M}{1-\gamma} \quad (1.1.2) \end{aligned}$$

Determining a lower bound on V^{π} is similar, in that

$$\begin{aligned} V^{\pi}(s) &= \mathbb{E}_{\pi}[\sum_{k=0}^{\infty} \gamma^k r_{t+k}] | s_t = s] \\ &\geq \sum_{k=0}^{\infty} \gamma^k - M \\ &= \frac{-M}{1-\gamma} \quad (1.1.3) \end{aligned}$$

Implying that

$$\forall s \in \mathcal{S}, |V^{\pi}(s)| \leq \frac{M}{1-\gamma}.$$

Since we have assumed $\mathcal{S}$ is finite, we note that $(\mathbb{R}^{\mathcal{S}}, ||\ ||_{\infty})$ is in fact a Banach space (e.g. a complete normed vector space), where for $V \in \mathbb{R}^{\mathcal{S}}$, $||V||_{\infty} = \max_{s \in \mathcal{S}} |V(s)|$. Next, we introduce the important operator that is used to solve MDPs. First, a definition:

Definition 1.1.5. *(Bounded Operator) Let $(X, ||\ ||_X), (Y, ||\ ||_Y)$ be normed vector spaces, and $T : \mathcal{X} \to \mathcal{Y}$ an operator. T is a **bounded operator** if $\exists M > 0$, such that $\forall x \in \mathcal{X}$*

$$||Tx||_Y \leq M||x||_X.$$

*Moreover, T is a **contraction mapping** if $M < 1$.*

We will introduce the Bellman optimality operator, show that it is a bounded operator, and for $\gamma < 1$, that it is a contraction mapping. We will make use of the Banach fixed-point theorem to show that the fixed point of this operator is unique, and that it is in fact $V^* := V^{\pi^*}$. Hence, using this operator allows us to solve for the optimal policy for any MDP for which we have access to this operator.

Definition 1.1.6. *(Bellman Optimality Operator) Let $v \in \mathscr{V} := (\mathbb{R}^{\mathcal{S}}, ||\ ||_\infty)$. We define the* ***Bellman Optimality Operator*** *$\mathscr{T}^* : \mathscr{V} \to \mathscr{V}$ pointwise $\forall s \in \mathcal{S}$:*

$$(\mathscr{T}^* v)(s) := \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma v(s')]$$

The Bellman Optimality Operator acting on Q-value functions is defined similarly,

$$(\mathscr{T}^* q)(s,a) = \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma \max_{a' \in \mathcal{A}}\ q(s',a')].$$

Under assumptions stated in Definition 1.1.1, namely that $\mathcal{S}$, $\mathcal{A}$ are both finite and R is bounded, then $\mathscr{T}^*$ is a well defined operator on $\mathscr{V}$. Next, we state the Banach fixed point theorem without proof, then show that $\mathscr{T}^*$ is a contraction mapping for $\gamma \in [0,1)$, and hence a bounded operator.

Theorem 1.1.7. *(Banach fixed-point theorem [2])Suppose X is a Banach space and $T : X \to X$ is a contraction mapping (see Definition 1.1.5). Then,*

- *i) $\exists!\ x^* \in X$ s.t. $Tx^* = x^*$; and*
- *ii) $\forall x_0 \in X$, the sequence $\{x_n\}$ given by $x_{n+1} = Tx_n = T^{n+1}x_0$ converges to x^*.*

Theorem 1.1.8. *[24] Let $\gamma \in [0,1)$. Then $\mathscr{T}^*$ is a contraction mapping on $\mathscr{V}$.*

Proof: Let $U, V \in \mathscr{V}$, and fix $s \in \mathcal{S}$. WLOG assume $\mathscr{T}^*U(s) \leq \mathscr{T}^*V(s)$. Let $a^* \in \underset{a \in \mathcal{A}}{arg\,max}\ \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma V(s')]$. Hence, a^* is the action that realizes $\mathscr{T}^*V(s)$. Similarly, let $a' \in \underset{a \in \mathcal{A}}{arg\,max}\ \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma U(s')]$. Then,

$$\begin{aligned}
0 &\leq \mathscr{T}^*V(s) - \mathscr{T}^*U(s) \\
&= \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma V(s')] - \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} P(s'|s,a)[R(s,a,s') + \gamma U(s')] \\
&= \sum_{s' \in \mathcal{S}} P(s'|s,a^*)[R(s,a^*,s') + \gamma V(s')] - \sum_{s' \in \mathcal{S}} P(s'|s,a')[R(s,a',s') + \gamma U(s')]
\end{aligned}$$

However, since

$$\sum_{s' \in \mathcal{S}} P(s'|s,a')[R(s,a',s') + \gamma U(s')] \geq \sum_{s' \in \mathcal{S}} P(s'|s,a^*)[R(s,a^*,s') + \gamma U(s')]$$

by definition of $\mathscr{T}^*U(s)$, then we have that

$$\sum_{s' \in \mathcal{S}} P(s'|s,a^*)[R(s,a^*,s') + \gamma V(s')] - \sum_{s' \in \mathcal{S}} P(s'|s,a')[R(s,a',s') + \gamma U(s')]$$
$$\leq \sum_{s' \in \mathcal{S}} P(s'|s,a^*)[R(s,a^*,s') + \gamma V(s')] - \sum_{s' \in \mathcal{S}} P(s'|s,a^*)[R(s,a^*,s') + \gamma U(s')]$$
$$= \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a^*)[V(s') - U(s')]$$
$$\leq \gamma \sum_{s' \in \mathcal{S}} P(s'|s,a^*)||V - U||_\infty$$
$$= \gamma ||V - U||_\infty$$

Repeating the argument for the case that $\mathscr{T}^*V(s) \leq \mathscr{T}^*U(s)$ implies that

$$|\mathscr{T}^*V(s) - \mathscr{T}^*U(s)| \leq \gamma ||V - U||_\infty, \text{hence, taking sup over s yields}$$

$$||\mathscr{T}^*V(s) - \mathscr{T}^*U(s)||_\infty \leq \gamma ||V - U||_\infty$$

∎

With the two previous theorems in mind, we return to equation (1.1.1) and see that $V^* = \mathscr{T}^*V^*$. Hence, $V^{\pi^*} := V^*$ *is* the fixed point of the Bellman Optimality Operator $\mathscr{T}^*$. Moreover, from the Banach fixed-point theorem and the fact that $\mathscr{V}$ is a Banach space, then it follows that for any initial vector $V_0 \in \mathscr{V}$, as $k \to \infty$, $\mathscr{T}^k(V_0) \to V^*$

1.2 Algorithmic Solutions to MDPs

In this section we introduce two different algorithmic solutions for arriving at π^*, V^* and Q^*. The first approch, *value iteration*[24] , is a classic model-based approach that is essentially an algorithmic instantiation for solving for the Banach fixed point of the Bellman Optimality Operator. The second approach is the highly influential *Q-learning* algorithm (refer to Algorithm 2) [7], which is model-free and under mild conditions converges to Q^* with probability 1.

1.2.1 Value Iteration

Value iteration is a simple algorithm that makes full use of the previous theorems surrounding $\mathscr{T}^*$. By starting with an arbitrary initial guess to V^*, typically $V_0 \equiv 0$,

$\mathscr{T}^*$ is iteratively applied to each subsequent iterate, which ultimately converges to V^*. Upon convergence, the optimal policy, π^* can be found by finding at each state s the action a that realizes:

$$\pi^*(s) \in \underset{a \in \mathcal{A}}{arg\,max}\ \ V^*(s) = \underset{a \in \mathcal{A}}{max}\ \ Q^*(s,a)$$

Algorithm 1 Value Iteration

```
Require: M, ε
Δ ← ε
Initialize V(s) arbitrarily ∀s ∈ S
while Δ ≥ ε do
    Δ ← 0
    for all s ∈ S do
        v ← V(s)
        V(s) ← T*v
        Δ ← max(Δ, |v − V(s)|)
    end for
end while
Return V ≈ V*
```

In practice, setting ϵ small allows one to solve for an ϵ-optimal policy, $\hat{\pi}$, under a finite number of iterations. Though this is quite attractive, it comes only under strong conditions: one is required to have oracle knowledge of the environment dynamics P and reward function R, which is generally not the case. Hence, we will consider model-free methods, where the solution methods do not require a model of the environment, and solve for π^* using simply the Bellman Optimality Operator and the Q-value functions.

1.2.2 Q-learning

Q-learning was introduced in [7] and has been one of the main driving forces within reinforcement learning. Here, the state-action value function plays center stage. Since no model of the environment is present, Q^* will be approximated from empirical data collected as the agent interacts with the environment using exponentially weighted moving averages. This algorithm also utilizes the fact that there exists a Bellman Optimality Operator for the Q-value functions, which is similarly defined by replacing V^* with Q^* (see Definition 1.1.6).

The Q-learning algorithm begins with an arbitrary initial estimate $Q_0 \in \mathbb{R}^{\mathcal{S}\times\mathcal{A}}$, and at every time t

$$Q_{t+1}(s_t, a_t) \leftarrow (1-\alpha_t)Q_t(s_t, a_t) + \alpha_t\Big[r_t + \gamma \max_{a\in\mathcal{A}} \; Q(s_{t+1}, a)\Big]$$

At each time step the algorithm selects an action according to some policy that is a function of both Q_t, and some some encoding of an *exploration* procedure. The action selection strategy generally addresses the *explore vs. exploit* issue in RL [28], being that an agent has to balance acting to the best of its current knowledge, and exploring other actions so as to improve on its current knowledge. The only assumption is that the action selection strategy satisfies the following definition.

Definition 1.2.1. *(Greedy in the Limit with Infinite Exploration (GLIE)) Let $N_k(s, a)$ denote the number of times action a has been taken from state s within the first k time points, and let $N_k(s)$ denote the number of times state s has been visited within the first k time points. Let $\{\pi_k\}_{k\geq 0}$ be a sequence of policies over time $k \geq 0$. An action selection strategy is* ***GLIE*** *if with probabilitiy 1:*

- *Each action is taken infinitely often in each state that is visited infinitely often, and*
- *In the limit, the learned policy is greedy with respect to the learned Q function. More formally, $\lim_{k\to\infty} \pi_k(s) \to \pi(s) \in \underset{a\in\mathcal{A}}{arg\,max}\; Q^{\pi}(s, a)$*

One such GLIE action selection strategy is *epsilon greedy*. For epsilon greedy exploration, a sequence of exploration parameters $\{\epsilon_t\}$ are used. For example, both $\epsilon_t := \frac{1}{t}$, $\epsilon_t := \frac{c}{N_t(s)}$, or $\epsilon_t := \frac{c}{N_t(s,a)}$ for some constant $c > 0$ are suitable choices (where when the denominator of either fraction is zero, we simply set $\epsilon_t = 1$). Under either implementation, the action selection strategy of the Q-learning algorithm behaves as follows:

$$\begin{aligned}
&\pi(s) \in \underset{a\in\mathcal{A}}{arg\,max}\; Q_t^{\pi_t}(s, a), \textit{with probability } 1-\epsilon_t, \quad \textit{and}\\
&\pi(s) \sim \textit{Uniform}(\mathcal{A}), \textit{with probability } \epsilon_t.
\end{aligned}$$

We provide a short proof that ϵ-greedy exploration is GLIE.

Theorem 1.2.2. *Let $c \in (0, 1)$, and define $\epsilon_t(s) = \frac{c}{N_t(s)}$. Let $\mathcal{M}$ be a communicating MDP. Then the above described ϵ-greedy action selection strategy is GLIE.*

Proof: We begin by proving the first aspect of GLIE. In order to do so we will rely on on the Borel Cantelli lemma that states if $\{E_n\}_{n=1}^{\infty}$ are a sequence of independent

events and $\sum_{n=1}^{\infty} Pr(E_n) = \infty$, then $Pr(\limsup_{n\to\infty} E_n) = Pr(\cap_{n=1}^{\infty} \cup_{k\geq n}^{\infty} E_k) = 1$. This lemma simply states that if the sum of probabilities of a sequence of independent events diverges, then the set of events that are repeated infinitely many times must occur with probability one. Since the MDP is communicating, every state is visited infinitely often as long as each action is taken infinitely often. Hence, it is sufficient to show that $\forall s \in \mathcal{S}, a \in \mathcal{A}$, $\lim_{k\to\infty} N_k(s,a) \to \infty$.

Since the MDP is Markovian, independence of events cannot be employed. Let us consider $T_s(t)$ be the time index where state s was visited for the kth time. Let $c \in (0,1)$, and define $\epsilon_t(s) = \frac{c}{N_t(s)}$. Note, for any fixed $a \in \mathcal{A}$, $\mathbb{P}(a_t = a | s_t = s, T_s(t)) \geq \frac{\epsilon_t(s)}{|\mathcal{A}|}$, since with probability $\epsilon_t(s)$ one of the actions will be sampled uniformly at random. However,

$$\begin{aligned}\sum_{t=1}^{\infty} \mathbb{P}(a_t = a | s_t = s, T_s(t)) &\geq \sum_{t=1}^{\infty} \frac{\epsilon_t(s)}{|\mathcal{A}|} \\ &\geq \sum_{t=1}^{\infty} \frac{c}{t} = \infty\end{aligned}$$

Hence, it follows that action a is taken infinitely often from state s. Since (s,a) were arbitrary, this is true for all states and actions. Hence, in the limit, all states and actions are visited infinitely often.

To prove the second condition that epsilon greedy is GLIE, we note that since $\forall s \in \mathcal{S}$, $\lim_{k\to\infty} N_k(s) \to \infty$, then $\lim_{k\to\infty} \epsilon_k(s) \to 0$, which states that the action selection strategy is greedy in the limit, thus concluding the proof. ∎

GLIE action selection strategies such as epsilon greedy are theoretically beneficial, as they guarantee that every state-action pair is simpled infinitely often. This is important for stochastic approximation, as it ensures that any unbiased estimator, by the law of large numbers, will converge to the true distribution mean (e.g. Q-value). Since the estimates Q-values from such a sampling process gets more and more accurate as $t \to \infty$, the greedy in the limit aspect ensures that the policy followed becomes closer and closer to being optimal as $t \to \infty$.

Algorithm 2 describes the pseudo-code for the Q-learning algorithm. This algorithm will be the basis for the algorithms developed in Chapters 2 and 3, as well as the basis for the Deep Q-network (DQN) algorithm used to perform function approximation using deep neural networks.

Algorithm 2 Q-learning

Require: Learning rate and exploration schedule $\{\alpha_t\}$, $\{\epsilon_t\}$.
Initialize $Q(s, a)$ arbitrarily $\forall(s, a) \in \mathcal{S} \times \mathcal{A}$
Initialize s
for t=0,1,2,... **do**
  $\Delta \leftarrow 0$
  $t \leftarrow t + 1$
  Select a from some GLIE action selection strategy (e.g. ϵ-greedy)
  $s', r \sim Environment(a)$ ▷ Take action a and observe successor state s' and reward r
  $Q_t(s, a) \leftarrow (1 - \alpha_t)Q_{t-1} + \alpha_t[r + \gamma \max_{a \in \mathcal{A}} Q_{t-1}(s', a)]$
  $s \leftarrow s'$
end for

Theorem 1.2.3. *(Convergence of Q-learning [7]) Let $\{\alpha_t\}_{t\geq 0}$ be a sequence of learning rates such that:*

- $\sum_t \alpha_t = \infty$, *and*
- $\sum_t \alpha_t^2 < \infty$.

Then for any initialization Q_0, and for any GLIE action selection strategy, under Q-learning $\lim_{t\to\infty} Q_t \to Q^*$ with probability 1.

1.2.3 Performance Measures for Algorithmic Solutions

As mentioned briefly in section 1.1.1, not only are we concerned about arriving at an (approximately) optimal policy, we also want the approach by which we arrive at this (approximately) optimal policy to have favorable properties amenable to real life use. With this in mind, we introduce two valuable notions. The first definition, *regret*, captures the intuitive idea of the difference in rewards obtained had one taken the optimal policy, as opposed to the policy actually followed. The second definition, *sample complexity*, measures how many time steps the algorithm must interact with the environment before reaching some threshold level of performance (e.g. ϵ-optimality).

Definition 1.2.4. *(Regret [30]) Let $\mathcal{M}$ be an MDP, s be the initial state of the environment, $\mathscr{A}$ be an RL learning algorithm and π^* be an optimal policy. Then, the* ***regret*** *after T time steps, $R(\mathcal{M}, \mathscr{A}, s, T)$, is defined as,*

$$R(\mathcal{M}, \mathscr{A}, s, T) := \mathbb{E}_{\pi^*}\Big[\sum_{t=1}^{T} R_t|s\Big] - \mathbb{E}_{\mathscr{A}}\Big[\sum_{t=1}^{T} R_t|s\Big]$$

Since the regret is defined as the expected sum of rewards over T time steps, where the expectation is taken over the action selection behaviour of the optimal policy and that of the algorithm (as well as the environment dynamics), if an algorithm $\mathscr{A}$ *never* converges to π^*, then as $T \to \infty$, the regret will grow unbounded, thus making it suitable for finite-time analysis. This wil be the case, for example, if we ask our algorithm to stop learning and return any policy that is ϵ-optimal. In situations where we do not require optimality, but find ϵ-optimality to be sufficient, the notion of *sample complexity* is a more useful measure of algorithmic efficiency. For the following definition, we abuse notation and let $\mathcal{S}, \mathcal{A}$ denote both the state and action space, as well as their respective cardinalities (e.g. $\mathcal{S} \equiv |\mathcal{S}|$).

Definition 1.2.5. *(Sample Complexity and Probably Approximately Correct in MDPs [13]) Let $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, p_0, \gamma \rangle$ be an MDP, $\mathscr{A}$ an RL learning algorithm, π^* an optimal policy, $\epsilon > 0$ and $\hat{\pi}_k$ the policy returned by $\mathscr{A}$ after k time steps. Then, the* ***sample complexity*** *of $\mathscr{A}$ is the infimum T such that*

$$\forall t \geq T, \quad \max_{s \in \mathcal{S}} \; V^{\pi^*}(s) - V^{\hat{\pi}_t}(s) < \epsilon.$$

$\mathscr{A}$ is ***Probably Approximately Correct in MDPs (PAC-MDP)*** *if $\forall \epsilon > 0, \delta \in (0,1)$, $\forall \mathcal{M}$, with probability $\geq 1 - \delta$, $\mathscr{A}$ returns an ϵ-optimal policy with finite sample complexity. Moreover, $\mathscr{A}$ is considered* ***efficient*** *if $T = poly(S, A, \frac{1}{1-\gamma}, log(\frac{1}{\delta}))$.*

Sample complexity measures the amount of time required for an algorithm to return an ϵ-optimal policy.

Though there are a wide diversity of reinforcement learning solution approaches ranging from actor-critic, policy search, policy gradient and Probably Approximately Correct in MDPs (PAC-MDP), a full discussion is beyond the scope of this thesis. We highlight some important works in [13, 28, 24, 11, 9, 14, 25, 27]. We have touched on solution approaches when access to the environment model exists, as well as a model-free approach. Both of these solutions assumed $\mathcal{S}$ is finite, and hence were *tabular* in nature (e.g. a look up table). We next consider the setting where either $\mathcal{S} = \infty$, or so large that function approximation is required.

1.2.4 Deep Q Network and Function Approximation

When $\mathcal{S}$ is quite large or uncountable, using tabular (look up table) methods is insufficient. For such settings function approximation offer an attractive setting which allow for a compact representation of the value function, and depending on the function approximator used, may allow for generalization of the value function to new states yet to be encountered by the agent. Though many function approximators have been used historically, including linear methods, radial basis functions and kernel methods,

we will restrict ourselves through the use of deep neural networks. We will specifically consider the Deep Q-network (DQN) framework [33], where we represent the Q-value function using a parameterized neural network, Q^θ, with parameters θ. We assume that the state space $\mathcal{S} \equiv \mathbb{R}^{\mathcal{S}}$, and therefore $Q^\theta : \mathbb{R}^{\mathcal{S}} \to \mathbb{R}^{\mathcal{A}}$. For example, in visual domains where the DQN receives as input an image of size (h, w) pixels, where h, w represent the height and width of the image, and the agent has 4 actions to move the agent in the cardinal directions, then $\mathcal{S} \equiv \mathbb{R}^{h \times w}$, and DQN outputs four real numbers, $Q^\theta(s, .) \in \mathbb{R}^4$, one for each action. For a given policy π, the DQN is attempting to learn an approximation: $Q^\theta(s, .) \approx Q^\pi(s, .)$, $\forall s$.

Neural Networks

This section provides a brief introduction to deep neural networks. A more in-depth introduction can be found [26]. For the sake of simplicity we omit *biases* from the discussion. Let $NN : \mathbb{R}^n \to \mathbb{R}^m$ be a neural network, parameterized by parameters θ. A neural network is a directed graph composed of a sequence of *layers*, whereby each layer is composed of a set of *neurons*, each assigned an *activation function*, fully connected to the previous layer, and fully connected to the next layer. The input layer $\ell_0 \equiv \mathbb{R}^n$ has no prior layers, and is fully connected to the following layer ℓ_1. The edge weights of these connections are the parameters of the network, and are the subject of optimization and learning. ℓ_1 is fully connected to ℓ_2, and this process continues until ℓ_{N-1} is fully connected to $\ell_N \equiv \mathbb{R}^m$, which we call the *output* layer. The number of neurons or *units* in each layer ℓ_k specifies the dimensionality of the representation of the input, $x \in \mathbb{R}^n$ at layer k. For example, let $\ell_1 \equiv \mathbb{R}^p$, then the weights of the network connecting the input layer ℓ_0 to ℓ_1 is simply a weight matrix of parameters $W_{0,1} \in \mathbb{R}^{p \times n}$. The image of such a map is called a *pre-activation*. Once the pre-activation function is computed at layer ℓ_1 a (typically) non-linear activation function is applied, resulting in a non-linear map between two Euclidean spaces. Each neuron at layer ℓ_1 applies such a non-linear activation function, whose output is applied to another weight matrix $W_{1,2}$, and this process continues iteratively until the pre-activation is computed at the final output layer. Depending on the objective, the final output activation applied to the pre-activation may either be a linear or non-linear (e.g. softmax) activation function. Hence, the input of the neural network undergoes a finite sequence of linear transformations followed by non-linear operations. The weights of the directed graph form the parameterization of this non-linear function, and are *learned* by iteratively receiving *batches* of data $\{x_i, y_i\}_{i=1}^B$, computing the output of the neural network over the batch,

$$\{NN(x_i) \equiv \hat{y}_i\}_{i=1}^D,$$

followed by computing some objective loss (a function of the approximations made by the neural network and the *targets*), for example the $L2$ loss,

$$Loss = \frac{1}{B}\sum_{i=1}^{B}\big(NN(x_i) - y_i\big)^2.$$

Finally, some gradient-based optimizer is applied, (e.g. vanilla stochastic gradient descent or Adam [15]) which updates the parameters in the direction of the negative gradient by some small step size parameter α_t,

$$\theta_{t+1} \leftarrow \theta_t - \alpha_t \nabla(Loss)$$

This process continues iteratively until some convergence criteria has occurred. The method above describes a batch gradient descent protocol which, depending on the optimizer and conditions, have theoretical convergence properties [22, 15] which ultimately aim at arriving at a set of parameters θ^* that minimizes the loss objective function mentioned above, arriving at some (local) minima.

1.2.5 DQN

[33] Introduced the groundbreaking work that utilized deep neural networks to surpass human level performance in playing Atari video games, purely from images. The DQN acted as a map as described above, with $Q^\theta(s,.) \approx Q^*(s,.) \in \mathbb{R}^{\mathcal{A}}$. The authors set out to minimize a sequence of loss functions $Loss_i(\theta_i)$ that change over time, with

$$Loss_i(\theta_i) = \mathbb{E}\Big[(y_i - Q^{\theta_i}(s,a))^2\Big],$$

where $y_i = \mathbb{E}_{s'}\big[r + \gamma \max_{a' \in \mathcal{A}} Q^{\theta_{i-1}}(s',a')|s,a\big]$. Here r denotes the immediate reward. The authors note that using the same network to both provide the estimates and the target value being estimated is not ideal and easily leads to instabilities throughout the learning process. Due to this ubiquitously moving target phenomena, [33] introduces the notion of a target network parameterized by θ^{target}. The target DQN network is initialized with the same weights as the DQN used for learning. These parameters remain fixed for a constant number K training iterations, at which point the parameters of the DQN are copied to the target network, and remain fixed for another K training iterations. By freezing the weights of the target network, the DQN is able to have a stationary target (for K iterations), which empirically leads to a more stable training procedure. In doing so, the loss function now becomes

$$Loss_i(\theta_i) = \mathbb{E}\Big[(y_i - Q^{\theta^{target}}(s,a))^2\Big].$$

However, rather than representing the function approximation target as $Q^{\theta^{target}}(s,a)$, the target is represented as $r + \gamma \max_{a' \in \mathcal{A}} Q^{\theta^{target}}(s',a')$, which yields the final true loss function as

$$Loss_i(\theta_i) = \mathbb{E}\Big[\big(Q^{\theta}(s,a) - (r + \gamma \max_{a' \in \mathcal{A}} Q^{\theta^{target}}(s',a'))\big)^2\Big].$$

In order to increase sample efficiency and continuously use past experiences, the authors introduce an *experience replay buffer*, $D = \{(s_i, a_i, r_i, s_i')\}_{i=1}^{|D|}$, which stores transition events over the life time of the agent. This experience replay buffer is used as a dataset, where batches of transitions are sampled uniformly at random, and the loss function is approximated using B samples so that

$$\hat{L}_i(\theta_i) = \frac{1}{B}\sum_{i=1}^{B}\Big(Q^{\theta}(s_i,a_i) - (r_i + \gamma \max_{a' \in \mathcal{A}} Q^{\theta^{target}}(s_i',a'))\Big)^2$$

1.3 Goal Based MDPs and Multi-Task RL

1.3.1 Goal Based MDPs

Goal based MDPs are a popular MDP formulation [28, 6, 16, 25, 17, 19], whereby distinct goals are encoded into the MDP structure. A goal can either be encoded as a state $g \in \mathcal{S}$ or a subset $\mathcal{G} \subset \mathcal{S}$ of the state space. Henceforth, the goal is assumed to be a particular state in the state space. In such instances, the MDP is defined as

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, p_0, \gamma, g \rangle$$

Typically, goal based MDPs are episodic in nature, where an episode denotes a finite length of time, H, for the agent to interact with the environment. Typically an episode may terminate when the agent has taken H time steps without having reached the goal state, or has arrived at the goal state before H steps. The state space can be modified such that for goal state g, $P(g|g,a) = 1$, $\forall a \in \mathcal{A}$. This modification of the transition dynamics encodes the goal state as an *absorbing state*, which the agent cannot escape from upon arrival. Typically one of either two reward functions are used for goal based MDPs:

- $R(s,a,g) = 1$, $\forall (s,a)$, and $R(s,a,s') = 0$, $\forall s' \neq g$, and $(s,a) \in \mathcal{S} \times \mathcal{A}$, or
- $R(s,a,s') = -1$, $\forall (s,a,s') \in \mathcal{S} \times \mathcal{A} \times \mathcal{S}$.

The former encoding gives sparse rewards such that only the goal state receives a positive reward. This thesis gives this reward structure the name **goal reward structure**. The latter reward function encoding, called the **action-penalty reward**

structure [16] gives the agent a "slap on the wrist" no matter what it does. Since the agent is motivated to maximize the expected cumulative sum of rewards, this promotes the agent to solve the task as quickly as possible. Moreover, compared to the first reward function, at each step the agent is given a negative feedback which is a more dense reward signal.

Note that though these domains are episodic in nature, often in practice the goal reward structure is used with $\gamma < 1$, hence it is solved using an infinite horizon setting, not an episodic setting [6, 25].

Before continuing to Multi-Task RL, we introduce some important definitions for MDPs, as well characterize the optimal value function for goal based MDPs under the action penalty reward structure.

Definition 1.3.1. *(Expected First Hitting Time) Given $s, g \in \mathcal{S}$, and π, the **expected first hitting time** of $\tau_\pi(g; s)$, g, when starting in state s, under polict π is*

$$\tau_\pi(g; s) = \sum_{k=1}^{\infty} kP(s_k = g | s_0 = s, s_{0:k-1} \neq g, \pi),$$

where the notation $s_{0:t}$ denotes that states at time time $0, 1, ..., t$.

An important notion within the MDP and RL fields is that of the *diameter* of an MDP, which is in some sense a measure of complexity or size of the state-space. Worst case regret and performance bounds are often stated in terms of the diamater of the MDP [16, 3, 30] . It measures the longest expected first hitting time of any two states under some (stochastic) shortest path between two distinct states, or stated more succinctly the longest shortest path within $\mathcal{S}$.

Definition 1.3.2. *(Diameter) The **diameter**, D, of an MDP is*

$$\max_{s \neq s'} \min_{\pi} \tau_\pi(s; s').$$

We now provide an intuitive way to characterize V^* in goal-based RL problems where the action penalty reward structure is used. Note that though the following proposition is stated and proved with respect to π^*, in fact, the statement is true $\forall \pi$.

Proposition 1.3.3. *Let $\mathcal{M}$ be a goal-based MDP with goal state g, $\gamma = 1$, action penalty reward structure, and $V^{\pi^*} = V^*$ be the optimal value function. Then $|V^*(s)| = \tau_{\pi^*}(s; g)$.*

Proof: We see that this proposition follows immediately, as

$$V^* = \mathbb{E}\Big\{ \sum_{k=0}^{\infty} R_{t+k} \Big\}$$

$$
\begin{aligned}
&= \mathbb{E}\Big\{ \sum_{k=0}^{\infty} -\mathbb{1}_{s_{t+k} \neq g} \Big\} \\
&= -\sum_{k=1}^{\infty} k P(s_k = g | s_0 = s, \pi^*) \\
&= -\tau_{\pi^*}(g; s)
\end{aligned}
$$

∎

1.3.2 Multi-Task RL

Multi-task RL is a rich framework that considers a sequence of T tasks, as encoded by a sequence of T MDPS, $\{\mathcal{M}_i\}_{i=1}^T$. For the purpose of this thesis we assume that the agent is constrained to act in the *same* environment over its "lifetime", and hence the sequence of tasks is encoded through a sequence of goal states $\{g_i\}_{i=1}^T$. This thesis considers the action-penalty reward structure, hence the reward function, R, does not change based on a given task. Given a sequence of task MDPs,

$$\{\mathcal{M}_i\}_{i=1}^T := \{\langle \mathcal{S}, \mathcal{A}, P, R, p_0, \gamma, g_i \rangle\}_{i=1}^T,$$

the goal of the agent is to solve for a sequence of optimal policies, $\{\pi_i^*\}_{i=1}^T$. Common to such problem formulations is the notion of *transfer learning* whereby what has been learned from previous tasks can be re-used to solve a novel task faster had the agent not had access to what was learned from the previous tasks [11].

1.4 Hierarchical RL and Temporally Extended Actions

1.4.1 Temporally Extended Actions

An active area of research within RL concerns itself with hierarchical action selection and temporally extended actions (often called temporal abstraction). The notion of hierarchical control is intuitive, as we do not make decisions at each nanosecond of our lives. Rather, we *chunk* our decision making events into higher and lower level abstractions such as "make breakfast", "get dressed", "shower", where each of those higher level temporally extended actions is comprised of lower level actions such as "crack eggs", "find clean socks" and "wash face". It goes without saying that each of those previously mentioned lower level actions themselves are comprised of sequences of even lower level actions, where at the lowest level of the hierarchy are basic muscle

contractions and applications of torque. This thesis concerns itself with two methods for encoding and representing temporally extended actions: macro actions [20] and options [29, 23]. Temporally extended actions have also been formulated under different names such as *skills* [17, 18, 4, 10] and *sub-goals* [1, 21], and are generally viewed as necessary for transfer learning in the multi-task RL settings [23, 17].

1.4.2 Macro Actions

A macro action (often called *closed loop temporally extended actions*), is an ordered sequence of actions of length l,

$$m = \langle a_1, a_2, ..., a_l \rangle.$$

An agent selecting m from state s commits to taking each action in m in succession. Macro actions form a deterministic concatenation of lower level (e.g. primitive) actions. Macro actions [20] have shown empirical speed ups in solving for (approximately) optimal policies over similar approaches that are restricted to solely primitive actions, however no theoretical results explaining this phenomena have been provided to date.

1.4.3 Options Framework

The options framework [29] provides a mechanism to extend the notion of policies over *primitive* actions to those over temporally extended actions.

Definition 1.4.1. *(Options) A set of* ***options****,* $\mathcal{O}$*, is such that* $\forall o \in \mathcal{O}$*,* $o = \langle \mathcal{I}_o, \pi_o, \beta_o \rangle$*, where for a higher level policy* $\pi : \mathcal{S} \to \mathcal{O}$*,*

- $\mathcal{I}_o \subseteq \mathcal{S}$ *is the initiation set where the option,* o*, can be taken by* π*.*
- $\pi_o : \mathcal{S} \to \mathcal{A}$ *is the option policy.*
- $\beta_o : \mathcal{S} \to [0, 1]$ *is the termination function, where at state* s*, option* o *terminates with probability* $\beta_o(s)$*, and returns control to* π*.*

In this setting a higher level policy π selects an option o, which is itself a Markovian policy over actions, and upon termination returns control back to π. Options generalize the concept of primitive actions, since a set of primitive actions $\mathcal{A}$ are in fact options where $\mathcal{I}_a \equiv \mathcal{S}$, $\forall a \in \mathcal{A}$, and $\beta_a(s) \equiv 1$, $\forall a \in \mathcal{A}$, $\forall s \in \mathcal{S}$. Since primitive actions are sufficient for solving for optimal policies and value functions, the previously mentioned algorithmic solution frameworks also yield π^* or V^* when primitive actions are augmented with options. All RL algorithms are compatible with the options framework and interestingly, when higher level policies are defined solely over

options (thus primitive actions are only accessible via the option policies themselves), the aforementioned algorithmic solutions observe a computational speed-up, arriving at a solution faster than considering only primitive actions [29, 23, 8, 32, 31]. Despite this speed-up, the quality of the solution is no longer guaranteed. That is, for a policy $\pi : \mathcal{S} \to \mathcal{O}$, with associated value function over options $V^{\pi}_{\mathcal{O}}$, $\forall s \in \mathcal{S}$ we have that $V^{*}_{\mathcal{O}}(s) \leq V^{*}(s)$. The observed speed-up in algorithmic solutions is due to the temporally extended nature of the options, where values of temporally distant states can be propagated in fewer iterations [32, 31].

1.4.4 Landmark Options

The notion of landmarks in navigation is intuitive, and helps guide search and planning in real life. We introduce the notion of landmarks, however their formal definitions are presented in Chapter 3. Generally speaking, a set of landmarks $\mathbb{L} \subset \mathcal{S}$ is a set of states. Typically, we will be considering options that aim at arriving at or nearby landmark states, which may be useful towards arriving at a given goal state g.

Landmarks in RL were first introduced in the context of options [29], where a set of options were hand-coded to arrive at previously selected landmark states within a state-space. Here, the goal-based RL setting selected one of the landmark states as the goal state, and a policy over the pre-defined landmark options was learned. Approximate value iteration (AVI) was considered in the context of landmark options, where AVI utilizes the value iteration algorithm using function approximation [31]. The theoretical properties of approximate value iteration using options and landmark options revealed that approximate value iteration converged faster when a set of landmark options was sparsely distributed over the state space, and when the landmark options have a longer temporal duration[31]. By sparsely distributed, it is meant that the set of states where a landmark option terminates is small and not concentrated in any region of the state space. The authors also provide a finite time error analysis on the output of their approximate value iteration algorithms. The authors assume that the set of landmarks and landmark options is provided to the learning algorithm, moreover that the learning algorithm has access to an approximate oracle simulator of the MDP, and that the value functions are locally Lipschitz around the landmarks. That is: given a metric σ over $\mathcal{S}$, such that for all landmarks ℓ and $s \in \mathcal{S}$, if $\sigma(s, \ell) < \eta$ then $V^{*}(\ell) \geq V^{*}(s) - \kappa\sigma(s, \ell)$, for some $\kappa \geq 0$. The authors also show empirically under several test domains that their landmark options variant of approximate value iteration both converges faster than with options or primitive actions, but also the resulting policy outperformed those with regular options and primitive actions.

Chapter 2

Hierarchical Macro Action Chaining (HMAC)

Hierarchical Macro Action Chaining In Reinforcement Learning

Anonymous Author(s)
Affiliation
Address
email

Abstract

Temporally extended actions allow planning at multiple time scales and have been shown to speed up convergence to a stationary policy. Here we introduce HMAC, an algorithm that learns temporally extended actions in the form of macro actions added to the agent's action set in an online data-driven manner. New macro actions may be composed of primitive actions or other macro actions, thus yielding a form of hierarchical chaining. This algorithm applies in any episodic reinforcement learning setting with well-defined terminal or goal states. We report positive results in different episodic settings demonstrating the empirical value of HMAC.

1 Introduction

Temporal abstraction has become a promising approach for improving performance of reinforcement learning (RL) algorithms. The use of temporally extended actions allows for increased efficiency by planning over multiple time scales. Recent approaches include the options framework, macro actions and skill chaining, all of which use meta-actions composed in some way from primitive actions.

Here we propose and study an online algorithm which not only uses but iteratively *learns* new macros across successive episodes. This approach, termed Hierarchical Macro Action Chaining (HMAC), applies in any episodic domain with well-defined termination or goal states and serves as a meta-algorithm that can be run over top of most existing RL algorithms. In it, the agent maintains a set of macro actions that grows by hierarchical composition. Results using HMAC with tabular Q-learning in the grid-world and n-chain domains, as well with DQN in a Doom domain, demonstrate that HMAC can significantly speed up learning of the optimal policy for such tasks.

2 Background

Notation and setting We assume the standard model of an RL task as a Markov Decision Process (MDP) $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, \mathcal{R}, \gamma \rangle$, where $\mathcal{S}$ is the state space of the environment, $\mathcal{A}$ a set of (primitive) actions, P a transition kernel which models the transition dynamics, $\mathcal{R}(s_t, a_t, s_{t+1})$ the reward function mapping transition events to some subset of $\mathbb{R}$, and $\gamma \in [0, 1]$ the discount factor. The goal of the agent is to find a (deterministic) policy $\pi^* : \mathcal{S} \to \mathcal{A}$ from states to actions, which is optimal in the sense of maximizing expected cumulative reward over the lifetime of the agent. Specifically we consider tasks with well-defined termination or goal states, and assume an episodic setting, i.e., the agent performs the same task over and over across successive (finite) episodes, attempting to converge to an optimal policy. We restrict to model-free $tabula\ rasa$ learning (i.e. Q-learning).

Macro actions as temporally extended actions A macro action (macro), $m = \langle a_1, a_2, ..., a_l \rangle$ is an ordered sequence of actions of length l. An agent selecting m from state s_t commits to take each action in m at successive time steps until the final action a_l has been taken - regardless of the states encountered along the way. Note: length-l macros can be viewed recursively as length-2 macros formed from "actions" in the union of the set $\mathcal{A}$ and the set of length-$(l-1)$ macros.

Submitted to 31st Conference on Neural Information Processing Systems (NIPS 2017). Do not distribute.

Motivation In vision-based RL domains such as Atari, Doom and Minecraft, a method known as k-frame skipping is used extensively in order to speed up learning[1-6,12]. Here, an agent selects an action a and then takes this action k times. In fact, almost all studies implementing DQN learning under these domains utilize this approach (typically with $k = 4$). Effectively - though the terminology of macros is not used - these agents learn to act in their environments exclusively with macro-actions of length-k (of the form $\langle a, \ldots, a \rangle$) rather than primitive actions.

Hierarchical or temporally extended actions can be seen as "chunking" sequences of useful actions in order to accomplish a specific goal. Versions of macros, sometimes called "motifs", are observed in many forms in biological learning from motion, to speech or birdsong and even on the level of protein-folding. As for clues to mechanisms whereby macros might naturally arise in sentient animals, recent neuroscientific research suggests that rats play back memories encoded in the hippocampus during the waking state [13], in addition to during sleep. Here, the rat plays back successful experiences $backwards$ in order to evaluate behaviour in a reinforcement learning manner.

Algorithm Motivated by this intuition of "starting at the goal and moving backwards" HMAC is a meta-algorithm that runs a base RL algorithm performing value function estimation and that additionally maintains in buffer data describing the trajectory taken from the two most recent actions selected from π, including the γ-discounted reward over that trajectory. Upon arriving at the goal state and thereby terminating the episode, the agent creates a new macro composed of the two most recent actions selected from π. If this macro, m is not already in $\mathcal{A}$, then it is added. If the base algorithm is Q-learning (resp. deep Q-learning) the Q-function (resp. DQN) is expanded accordingly.

Related work with temporally extended actions The options framework [7-11] is an elegant and mathematically principled approach wherein options are represented as policies. A challenge for the options framework is finding a set of good options. Various works have used heuristics involving frequency based methods[15,16], identifying bottleneck states[17], or relied on manually defined options [7,8]. Another approach has been to learn goals or subgoals, which has seen success in deep learning approaches such as Minecraft [18]. In [19] a method called Skill Chaining is developed to learn a set of options where one options goal state is the initiation state of a separate option. Hence the agent learns to chain options together to traverse from initial state to goal state effectively. In [20] Strategic Attentive Writer for Learning Macro-Actions (STRAW) is a creative approach towards developing macros which utilizes principles from deep learning such as attention. Perhaps most similar to our study is the work of [21] which addresses macros for deep RL. Frequency- and repetition-based approaches were used to build macros in a data-driven manner. Implementations on the Atari domain found some positive results, however demonstrating high variability. Their approach is limited as they specify how long the macros should be (length 3 or 5), and do not grow hierarchically. Additionally, they limit the number of macro-actions learned, whereas we do not impose these limitations.

3 Experimental results

Grid world We tested HMAC running (tabular) Q-learning in the 15x15 grid world setting, under both deterministic and stochastic dynamics. HMAC+Q-learning was compared to baseline Q-learning where each implementation learns over 1000 episodes with the same initial and goal states. The greedy policy is tested every 50 episodes to evaluate learning progression. Exploration is done through ϵ-greedy ($\epsilon = 0.1$) behaviour. Both the deterministic setting and stochastic setting are tested. A full grid search of relevant hyper-parameters was performed and though we report on the top performing hyper-parameters, results were highly consistent over hyper-parameters. The reported results follow from n=100 experimental trials.

Figure 1 shows the learning curves over training and testing for both HMAC and baseline Q-learning for the grid world domain. Figure 1a clearly demonstrates that an agent utilizing HMAC learns a stationary policy faster than the baseline approach under both deterministic and stochastic settings. Testing results following the learned policy shows that that stationary policy learned using HMAC is the optimal policy. Evaluation of the learned policies over time reveals that HMAC not only promotes faster convergence to a stationary policy over the baseline Q-learning, but also that quality of the converged policies was optimal. Over 100 experimental runs, HMAC learned 29.8 macros on average, most of which were hierarchical in nature. In the deterministic setting the optimal policy learned by HMAC involved selecting a single macro from the start state that directed the agent to the goal state.

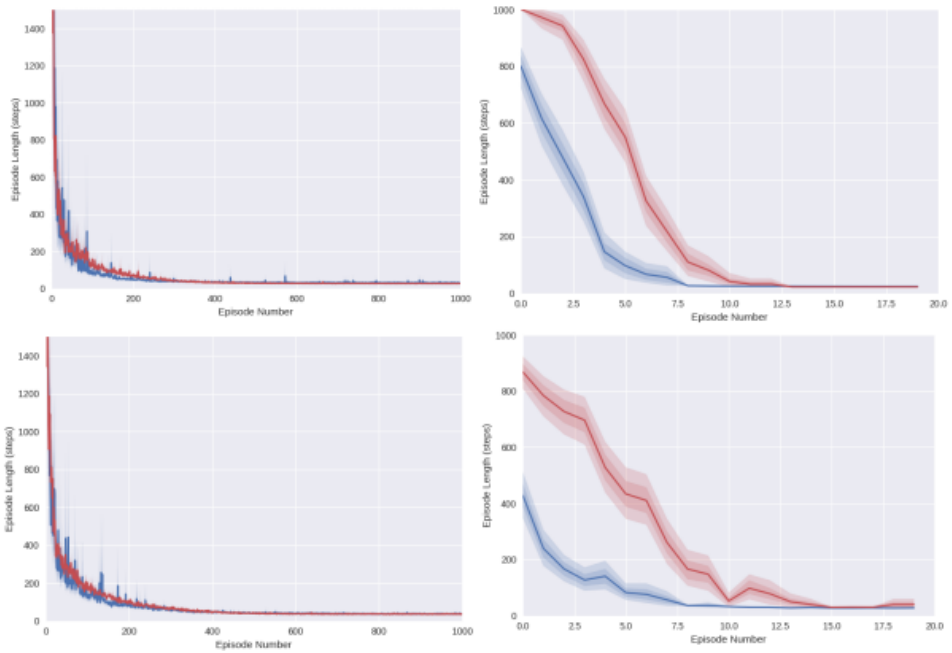

Figure 1: Learning curve of HMAC and baseline implementation for grid world domain after 100 trials. Top: Training (left) and evaluation (right)learning curves for deterministic grid world. Bottom: Training (left) and evaluation (right) learning curves for stochastic (probability of taking action successfully p=0.8) grid world. HMAC in blue and baseline in red.

N-chain Next we tested HMAC with Q-learning in a setting where deep exploration is needed: the classic stochastic n-chain (n=6), in which the agent must perform a long sequence of specific actions before it can discover the globally optimum reward, as opposed to greedily returning to the initial state. We compared HMAC+Boltzmann and HMAC+ϵ-greedy behaviour implementations to Boltzmann and ϵ-greedy baselines, respectively - all implementations with Q-learning. Briefly, $\mathcal{A} = \{left, right\}$, where $left$ returns the agent to s_0 with a $r = 1$, and $right$ takes the agent from state s_n to s_{n+1} w.p. $p = 0.9$, and returns the agent to s_0 w.p. $1 - p$. Upon returning to s_0 in this way, the agent receives $r = -1$. As the agent moves "up" the n-chain, it receives no reward, until finally arriving at the final goal state, whereupon it receives $r = 10^8$.

We ran 100 trials. Figures have been omitted for lack of space but with ϵ-greedy behaviour both HMAC and the baseline implementations learned the optimal policy in all trials, while HMAC did so faster. With Boltzmann exploration both implementations performed worse than the ϵ-greedy implementations and moreover the baseline Boltzmann implementation never learned the optimal policy, always selecting to move left and take the greedy reward. By contrast HMAC-Boltzmann succeeded in learning the optimal policy, albeit with quite large variance.

Doom Here we set out to test the HMAC approach in a more complex setting: the Deadly Corridor test scenario from VizDoom[22]. Doom is a 3D first person shooter game where at each time point the agent receives an observation in the form of a screen shot of its environment. This setting is a Partially Observable MDP (POMDP) making learning more difficult. The agent is equipped with a gun, and actions to move left, right, forward, or to shoot the gun. The agent begins at one end of a corridor that is flanked by three sets of monsters trying to kill the agent. The episode ends when the agent reaches the other end of the corridor and finds a vest, when the agent dies or when 4200 time steps have passed. The agent receives a -100 reward if it dies, otherwise it receives a reward at every time step proportional to its change in distance to the goal state vest, s_g, and an additional reward of 1000 if it reaches s_g. We modify the reward structure during training to $r = -1$ for all transitions in order to increase the difficulty of the task, but maintain the native rewards during testing.

For function approximation we used a standard implementation of the DQN[1-6,12] however we deviate here by using as input a single RGB image of size 60x45, rather than the 4 most recent images concatenated together. The network is trained using ADAM with learning rate $\alpha = 0.00001, \gamma = 0.99$. Training occurs over 100,000 frames with ϵ-greedy exploration, annealing from 1.0 to 0.1 over the first 50,000 frames, then remains at 0.1 for the last 50,000 frames. An experience replay buffer is maintained of size 10,000 with learning updates occurring after every action with batch size 32. Every 5000 frames of training is considered an epoch, and after every training epoch 50 test episodes are performed. Target network updates were performed every 2500 frames. The HMAC DQN agent is compared to a standard DQN agent as baseline. Experiments were run ten times.

Due to the nature of adding macro-actions to the repertoire of the DQN agent in the form of additional output unit in the final DQN layer, instability was a concern and we slowed the acquisition of macro-actions by modifying the approach used in the tabular setting and restricting HMAC to greedy episodes only (i.e. test episodes). Though [21] temporarily increase ϵ to 0.5 after the addition of a new macro, we do not modify the exploration behaviour which might provide bias and advantage towards preferring macro selection. Additionally, though [21] limit the number of possible actions learned to $|\mathcal{A}|$, we provide no upper bound limit on the number of macros learned. The learning curve results (Figure 3) show that the HMAC DQN agent learns to solve the task within the training time, however baseline DQN makes no learning progress. Training with native reward structure was also performed, due to the generous reward structure this task is learned within the first epoch.

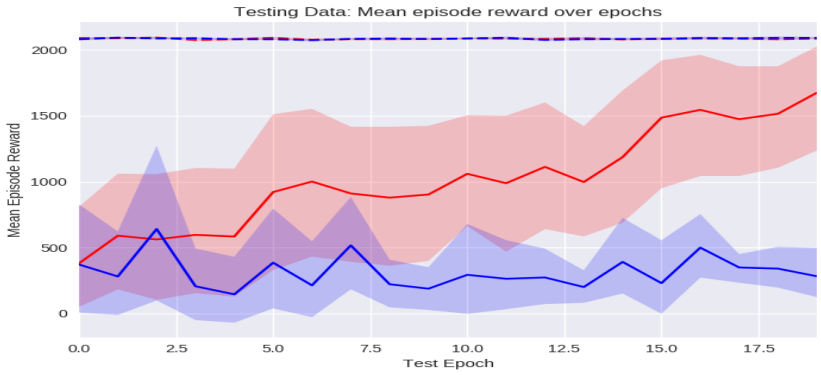

Figure 2: Mean rewards obtained during testing epochs throughout learning in Doom: Red - HMAC DQN; Blue - DQN Baseline; Dashed - Default reward structure during training; Solid lines - Action penalty reward during training

4 Discussion

This study presented a novel method to discover useful temporally extended actions in an online data-driven manner. We empirically tested HMAC performance in three RL settings: first with tabular Q-learning in the grid world domain, comparing HMAC Q-learning to a baseline Q-learning approach; second similarly but in the n-chain setting; and finally with a DQN in a test scenario of the Doom domain, comparing HMAC DQN with standard DQN learning. The results in each case demonstrated that HMAC is able to converge to a superior policy faster than the baseline approach. In each setting the agent learns to compose primitive and/or macro-actions together in a hierarchical nature. Despite the potential for instability in growing the set of actions throughout the learning process, HMAC with DQN resulted in strong performance compared to the top performing baseline DQN agent. These test domains were specifically chosen for their episodic nature, with clearly defined goal state, and HMAC's superior performance appears to stem from its ability to explore more deeply via temporally extended actions. We expect that HMAC could yield an improvement in performance in real-life robotics settings where an agent may face planning constraints or receive a negative reward with every planning step[17].

References

[1] Mnih, V., et al. Playing Atari with deep reinforcement learning. arXiv:1312.5602, 2013.

[2] Mnih, V., et al. Human-level control through deep reinforcement learning. Nature 517:529-533, 2015.

[3][6] van Hasselt, H., Guez, A., Silver, D. Deep reinforcement learning with double Q-learning. AAAI 2094-2100, 2016.

[4][7] Wang, Z., et al. Dueling network architectures for deep reinforcement learning. ICML. 47: 1995-2003, 2016.

[5][8] Schaul, T., Quan, J., Antonoglou, I., Silver, D. Prioritized experience replay. arXiv:1509.02971, 2016.

[6][9] Mnih, V., et al. Asynchronous methods for deep reinforcement learning. ICML. 47: 1928-1937, 2016.

[7][10] Sutton, S. R., Precup, D., Singh, S. Between MDPs and semi-MDPs: a framework for temporal abstraction in reinforcement learning. Artificial Intelligence 112: 181-211, 1999.

[8][11] Sutton, R.S., Precup, D., Singh, S. Intra-option learning about temporally abstract actions. ICML 556-564, 1998.

[9][12] Sorg, J., Singh, S. Linear options. Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems. 31-38, 2010.

[10][13] Mann, T.A., Mannor, S. Scaling up approximate value iteration with options: better policies with fewer iterations. ICML. 127-135, 2013.

[11][14] Mann, T.A., Mannor, S., Precup, D. Approximate value iteration with temporally extended actions. Journal of Artificial Intelligence Research. 53, 375-438, 2015.

[12][15] Lample, G., Chaplot, D.S. Playing FPS games with deep reinforcement learning. AAAI. 2140-2146, 2017.

[13][16] Foster, D.J., Wilson, M.A. Reverse replay of behavioural sequences in hippocampal place cells during the wake state. Nature 440:680-683, 2006.

[14][17] Leon, A., Denoyer, L. Options discovery with budgeted reinforcement learning. arXiv 1611.06824, 2016.

[15][19] Stolle, M., Precup, D. Learning options in reinforcement learning. International Symposium on Abstraction, Reformulation, and Approximation. 212-223, 2002.

[16][20] McGovern, A., Barto, A.G. Automatic discovery of subgoals in reinforcement learning using diverse density. Computer Science Department Faculty Publication Series, 2001.

[17][21] Menache, I., Mannor, S., Shimkin, N. Q-cut - dynamic discovery of sub-goals in reinforcement learning. European conference on machine learning. 295-306, 2002.

[18][22] Tessler, C., Givony, S., Zahavy, T. Mankowitz, D.J., Mannor, S. A deep hierarchical approach to lifelong learning in Minecraft. AAAI. 1553-1561, 2017.

[19][23] Konidaris, G., Barto, A. Skill discovery in continuous reinforcement learning domains using skill chaining. NIPS. 1015-1023, 2009.

[20][28] Vezhnevets, A.S., et al. Strategic attentive writer for learning macro-actions. arXiv:1606.0469, 2016.

[21][30] Duragkar, I.P., Rosenbaum, C., Dernbach, S., Mahadevan, S. Deep reinforcement learning with macro-actions. arXiv:1606.0461, 2016.

[22][32] Kempka, M., et al. ViZDoom: A doom-based AI research platform for visual reinforcement learning. arXiv:1605.02097, 2016.

Chapter 3

Landmark Options Via Reflection (LOVR)

Options in multi-task reinforcement learning - transfer via reflection

Nicholas Denis[1] and Maia Fraser[2 [0000−0003−4240−7178]]

[1] Dept. of Mathematics and Statistics, University of Ottawa
[2] Dept. of Mathematics and Statistics, University of Ottawa

Abstract. Temporally extended actions such as *options* are known to lead to improvements in reinforcement learning (RL). At the same time, transfer learning across different RL tasks is an increasingly active area of research. Following Baxter's formalism for transfer, the corresponding RL question considers the benefit that an RL agent can achieve on new tasks based on experience from previous tasks in a common "learning environment". We address this in the specific context of goal-based multi-task RL, where the different tasks correspond to different goal states within a common state space, and we introduce Landmark Options Via Reflection (LOVR), a flexible framework that uses options to transfer domain knowledge. As an explicit analog of principles in transfer learning, we provide theoretical and empirical results demonstrating that when a set of *landmark* states covers the state space suitably, then a LOVR agent that learns optimal value functions for these in an initial phase and deploys the associated optimal policies as options in the main phase, can achieve a drastic reduction in cumulative regret compared to baseline approaches.

Introduction

The strength of reinforcement learning (RL) in modelling and solving sequential decision problems is apparent in the broad and increasing range of problem-types considered by RL researchers. One example of these are lifelong learning settings [**?**]. They assume an RL agent is faced with a sequence of MDPs that share certain properties and are thus viewed as coming from a common *environment*. As in transfer learning [**?**], rather than beginning fresh with each new task, an efficient lifelong agent should leverage past knowledge of its experience within the environment in order to solve new tasks as quickly as possible. We consider such transfer in the form of options.

Options, or "temporally extended actions" have been found to be useful in overcoming the curse of dimensionality in many settings [**?**]. They were originally motivated as a form of inductive transfer [**?**], appearing soon after the survey [**?**] where earlier approaches to "learning to learn" in RL had been addressed. The use of such actions inherently sets up a hierarchy in time-scales, with primitive actions at the bottom and longer-running options higher up. How to structure control and learning within an RL hierarchy remain areas of active current research [**?**,**?**,**?**,**?**].

In this paper we introduce the Landmark Options Via Reflection (LOVR) framework. This is a general hierarchical framework for inductive transfer via options, that implements the shifting of control in the hierarchy through a dedicated action $a_{reflect}$ called

“reflection". For simplicity we consider here only two levels. The framework assumes a set of landmark states, $\mathbb{L}$, that covers the state space in a manner to be made precise, analogous to coverings of a learning environment studied in [**?**]. In Phase I, the agent learns accurate Q-value functions associated to arriving at $l \in \mathbb{L}$, then based on these Q-value functions defines options for use in Phase II, where the agent can access options only through $a_{reflect}$. The action $a_{reflect}$ is subject to the same learning updates (e.g. Q-value) as other primitive actions, and thus provides an RL-native way to shift control between levels in the hierarchy. Further levels could be defined analogously by invoking further phases but this is out of the scope of the present paper. We focus here on the transfer-benefits achieved through LOVR.

The versions of LOVR described here work in lifelong-learning settings consisting of a sequence of episodic task MDPs, each with its own well-defined goal state, s_g, and the rest of the MDP common across tasks. We give theoretical results for LOVR in finite state spaces and also empirical results that show LOVR drastically reduces cumulative regret for a lifelong learning agent given a sequence of MDP tasks. This holds even with the number of landmark states being a small proportion of the state space.

Related work

In the Separation of Concerns (SoC) framework [**?**,**?**] multiple agents estimate the value of a given state, and an aggregator function makes decisions by combining input of lower agents. Reflection via $a_{reflect}$ plays a similar role in our framework. Hierarchical RL approaches such as SoC or Feudal RL [**?**,**?**,**?**] allow for temporal abstraction at different levels; this is achieved at two levels by our use of landmark options.

Both MLSH [**?**] and LOVR are meta-algorithms for multi-task lifelong settings that allow various underlying RL agents. MLSH performs joint optimization over two sets of parameters, one for a higher-level agent which learns commonalities among tasks, and one for each task. Per task optimization is done in a reduced space of *sub*-policies which speeds discovery of a sequence leading to reward. Reducing learning problems is a motivation in our work as well, but the difference in our approaches is akin to learning in development vs. plasticity ([**?**] vs. [**?**]). Learning in mature nervous systems involves online training in both directions simultaneously [**?**] - similar to MLSH - while LOVR is a development-style agent, which first fine-tunes lower-level tools and then makes these available during learning by a higher-level agent.

Schaul et al [**?**] introduced the Universal Value Function Approximator (UVFA) for learning goal-based tasks in the multi-task setting. Here, the state and goal representations are concatenated and used as inputs to the network, which learns to approximate the Q-values of the state with respect to the current goal, g, parameterized by weights θ. The UVFA network, $U_\theta(s; g)$, allows for a single network to learn several tasks.

Brunskill et al [**?**] studied multi-task lifelong learning and provided PAC-MDP theoretical guarantees. Similar to LOVR, their framework involves two phases. Clustering of sub-policies provides the structure by which the higher-level agent accesses them. In contrast to LOVR, [**?**] allow for different tasks to have different transition kernels and reward functions. In their framework, during the second phase as the agent is experiencing the task MDP, the data acquired allows the agent to estimate which cluster the current MDP belongs to. In some sense, the decision to make such an estimation and

(temporarily) commit to a policy built for the estimated MDP cluster could be seen as an instantiation of $a_{reflect}$, however only in spirit as $a_{reflect}$ is an action selected by a higher level policy (thus conditionally dependent on the current state), whereas in [**?**] this decision is dependent on a sequence of data.

Konidaris et al [**?**] introduce the concept of an agent space, acting as an ego-centric view of the agent in a class of environments. Here, options learned in one MDP can be transferred to a new MDP coming from a similar class of environments (e.g. class of all grid worlds) via a mapping through the agent space. In this way, policies learned earlier can be transferred into the future as options (skills). A similar principle arises in [**?**].

Background

Multi-task and lifelong reinforcement learning We consider a multi-task RL setting where the agent is confined to a stationary environment, and throughout its lifetime is assigned a sequence of tasks. These tasks are represented as episodic MDPs $\mathcal{M}_i = \langle \mathcal{S}, \mathcal{A}, P, \mathcal{R}, s_{g_i}, \nu \rangle$, $i \in [T]$, with respective terminal states s_{g_i}. In summary: the state space $\mathcal{S}$, set of actions $\mathcal{A}$, transition probability kernel P, reward function $\mathcal{R}$ and initial state distribution ν all remain fixed across tasks and only the goal state varies. The goal state s_{g_i} thus encodes the i'th task, though when speaking of a single task this will be denoted simply s_g. In many settings it may be more applicable to consider goal sets $\mathcal{G}_i \subset \mathcal{S}$, rather than a single goal state. Our usage of terminal state follows the common formulation [**?**] for episodic tasks where one wants the agent, such as a robot, to perform a single specific function, and upon completion to become inactive and remain so until a new episode or task begins. In general one could let K_i be the number of episodes that $\mathcal{M}_i$ is run before the next task is assigned, but we will take $K_i \equiv K$ to be independent of i in the present paper. We consider the action-penalty reward structure [**?**] of -1 for all transitions, which reinforces the agent towards a policy of arriving at the current goal in as few steps as possible. The goal of the lifelong learning agent is to solve for a sequence of policies $\{\pi_i^*\}_{i=1}^T$ which minimize cumulative regret. It is clear that under this reward structure that $\pi^*(s)$ is the negative expected number of steps from state s to goal g, and since π^* is optimal, it encodes the (expected) most efficient path to the goal. For our empirical results we use two empirical measures of regret: the *empirical per-episode regret*: $\mathcal{R}^g_{episode}(\pi) := R_{\pi^*} - R_\pi$, which measures the difference in accumulated reward over that episode for the policy being considered vs. the optimal policy for the task (goal) g at hand, and the *empirical lifelong regret*: $\mathcal{R}_{life} := \sum_{t=1}^{T} \sum_{k=1}^{K} \mathcal{R}^t_{episode_k}$, which measures the cumulative empirical per-episode regret of an agent over a sequence of T tasks, each run for K episodes. Likewise, *per-episode regret* and *lifelong regret* are the expectations of the respective empirical regrets.

Options Framework The options framework is a mathematically principled approach for temporally extended actions [**?**]. An option $o \in \mathcal{O}$ is a triple, $o = \langle I_o, \pi_o, \beta_o \rangle$, where $\mathcal{I}_o \subseteq \mathcal{S}$ is the initiation set, π_o is the option policy that maps states to primitive actions, and β_o is the termination function which controls when to terminate π_o and return control back to π. The inclusion of options in an MDP results in a Semi-MDP (SMDP), where standard RL algorithms apply in the SMDP setting [**?**]. Landmark options were first

introduced in [**?**,**?**] as policies leading towards “landmark" states. We retain this aspect but employ them in a different way.

Landmark Options Via Reflection (LOVR) framework

Landmark value functions, landmark coverings and phase I For all approaches considered here, we assume that during a preliminary phase the agent solves a set of tasks corresponding to a set of *landmark states* $\mathbb{L} \subset \mathcal{S}$, with associated Q-value functions. For $l \in \mathbb{L}$, we denote by $V^l(s)$ the value of state s with respect to arriving at the goal-state l. We allow this phase to be general, where the landmark states are picked by a researcher, or built *tabula rasa* online; in either case, we only assume the following covering conditions are achieved by the algorithm: $A1)$ $\exists \eta > 0$, $\forall s \in \mathcal{S}$ $\exists l \in \mathbb{L}$ such that $|V^l(s)| \leq \eta$, which we call η-reachability, and in addition $A2)$ $\forall l \in \mathbb{L}$, V^l is ϵ-optimal. $A2$ can be achieved with a PAC-MDP algorithm such as Delayed Q-learning [**?**] or through building a model (i.e. E^3 [**?**]), however as we demonstrate experimentally, even without this assumption we achieve impressive results. The LOVR framework assumes that upon completion of the first phase $A1 - A2$ are satisfied before the second phase begins.

Note that when $\mathbb{L} = \mathcal{S}$, A1 is automatically satisfied. Moreover, in this work we restrict ourselves to environments that satisfy what we call κ*-approximate reversibility*, namely $\exists \kappa \geq 1$, such that $V^{s*}(g) \leq V^{g*}(s)\kappa$, $\forall s, g \in \mathcal{S}$, where V^{s*} denotes the optimal value function associated with arriving at goal-state s. This assumption is trivially satisfied when the state space communicates with a finite diameter, and when $\kappa = 1$ is called *reversible* [**?**]. If $\mathcal{S}$ satisfies κ-approximate reversibility, then a natural metric on $\mathcal{S}$ can be defined that makes it possible to succinctly describe “good" choices of $\mathbb{L}$, namely, if this set is chosen such that η-balls around $\mathbb{L}$ cover $\mathcal{S}$ then applying any algorithm that achieves ϵ-optimality (e.g. PAC-MDP algorithms) automatically achieves conditions $A1 - A2$. We leave discussion of these details for the extended version of the paper and here simply assume that $A1$ and $A2$ hold at the end of Phase I.

Phase II The LOVR framework uses the options framework to accomplish transfer in the goal-based multi-task RL setting by leveraging previously solved $\{Q^l\}_{l \in \mathbb{L}}$ from the first phase to define a set of landmark options $\mathcal{O}_l$ for the second phase. Options can only be accessed through a single action $a_{reflect}$, which acts as a gating mechanism to the set of landmark options. The effect of $a_{reflect}$ is then to execute a specific option $o_l \in \mathcal{O}_l$. Between initiation and termination, o_l follows the policy associated to the corresponding learned Q^l. LOVR’s generality, however, allows flexibility to encode in a domain-specific manner both $a_{reflect}$’s choice of option, as well as the definition of *initiation and termination conditions* for every $o_l \in \mathcal{O}_l$. The latter determine respectively the states from which o_l can be initiated, and the state at which to stop following o_l and return control to the higher level policy. This flexibility allows the designer to control how options are invoked in a manner that promotes positive transfer and reduces negative transfer. To demonstrate this, we introduce two instantiations of the LOVR framework, termed LOVR1 and LOVR2, each used in different settings and with different biases. We provide a high level description of how they differ, and then a technical description of each.

Both LOVR1 and LOVR2 assume the same choice of $\mathbb{L}$ and the same Phase I to produce $\{Q^l\}_{l\in\mathbb{L}}$. We utilize LOVR1, in the setting where the agent is not provided knowledge of the goal state g and, hence, must first arrive at g to identify this goal, while LOVR2 is provided the goal state g at the start of each task. Subsequently, once g is known, both instantiations proceed similarly but differ in subtleties of initiation and termination. LOVR1 follows the landmark option policy o_l attempting to arrive all the way to the landmark state l, while LOVR2, on the other hand, does not attempt to arrive at the landmark state itself, but rather, terminates the landmark option once the agent is within η steps of the landmark, in expectation. The difference in these two termination conditions reflect two different biases. If the agent follows a landmark option to a given landmark l, the goal is expected to be at most $\eta\kappa$ steps away from l under some optimal policy, however en route to the landmark l, the agent may "pass through" a state s' from which g can be directly reached in $\lambda << \eta\kappa$ steps. LOVR1 will not be able to take advantage of this shortcut. It accepts this potential inefficiency, in exchange for being in expectation at most $\eta\kappa$ steps from the goal once it stops. LOVR2 on the other hand, terminates the landmark option sooner, at which point the agent is in expectation at most $\eta + \eta\kappa$ steps from the goal, but it might find a shorter path and so escapes the potential inefficiency just described for LOVR1. The two instantiations, LOVR1 and LOVR2 - aside from having access to g or not - thus differ primarily in how they handle the tradeoff between expected closeness at option's termination vs. ability to take a shortcut. In addition initiation is slightly different in the two, forcing LOVR2 to take options when it is far from the goal. We now describe the technicalities of both instantiations. Throughout, we assume the landmark option policy, $\pi_{o_l}(s)$ takes the action that maximizes the associated $Q^l(s,a)$. In describing LOVR1, for a given landmark option $o_l \in \mathcal{O}_l$, the initiation set $\mathcal{I}_{o_l}$ is defined as

$$\mathcal{I}_{o_l} = \begin{cases} \mathcal{S} & if\ V^l(g) \geq V^{l'}(g),\ \forall l' \neq l \in \mathbb{L} \\ \emptyset & else. \end{cases} \tag{1}$$

LOVR1's instantiation of $a_{reflect}$ chooses any landmark option o_l such that l satisfies **??**. The initiation set is defined above, and the binary termination function, β_{o_l}, terminates o_l once the number of actions taken under this policy exceeds the value $|V^l(s)|$. Recall this value is an ϵ-optimal estimate of the expected number of steps from s to l under o_l. Thus, LOVR1 follows the optimal policy for the expected number of steps to reach the selected landmark. In describing LOVR2, we have,

$$\mathcal{I}_{o_l} = \begin{cases} \{s \in \mathcal{S} | \eta < |V^l(s)|\} & if\ V^l(g) \geq V^{l'}(g),\ \forall l' \neq l \in \mathbb{L} \\ \emptyset & else. \end{cases}$$

For LOVR2, we use the state dependant action set notation $\mathcal{A}(s)$ for the set of actions callable by π when in state s and we define: $\forall s \in \mathcal{S}$ s.t. $\eta < |V^l(s)|$, $\mathcal{A}(s) := \{a_{reflect}\}$, and $\forall s$ s.t. $\eta \geq |V^l(s)|$, $\mathcal{A}(s) := \mathcal{A}$, the set of primitive actions. In words, LOVR2 forces $a_{reflect}$ when the agent is at a state that is greater than η steps from the landmark state. Like in LOVR1, $a_{reflect}$ chooses any o_l such that l satisfies **??**. LOVR2 then defines the initiation set as just stated, and defines the termination function, $\beta_{o_l}(s; g) = \mathbb{1}\{V^l(s) \geq V^l(g)\}$, i.e., while following the landmark option policy, β_{o_l} terminates the option at any

state s such that $\eta \geq |V^l(s)|$. The idea behind LOVR2 is that the landmark option policy isn't attempting to arrive at the landmark state, rather to drive the agent to a pseudo-ball of states that can reach the landmark within η steps. While within that pseudo-ball, no landmark options can be taken, hence the agent must explore-exploit within that pseudo-ball. Rather than solving a large MDP defined over all of $\mathcal{S}$, the LOVR2 agent need only solve over a smaller constrained MDP defined over $\{s \in \mathcal{S} | \eta \geq |V^l(s)|\}$, which may be significantly smaller, and can also be controlled by the landmark covering parameter η.

Results

We test the two instantiations, LOVR1 and LOVR2, in two different settings. In the first, we use LOVR1 in a tabular gridworld where for each task the goal state is not provided to the agent. Hence, the agent has no access to $a_{reflect}$ nor the landmark options until the goal state has first been discovered. In the second setting, LOVR2 is tested in the function approximation setting in a visual gridworld. Here the goal state is provided to the agent. We provide some theoretical results for the first setting when $\mathbb{L} = \mathcal{S}$, and theoretical results for when the goal is provided to the agent, $\mathbb{L} \subset \mathcal{S}$, and under assumption A2.

Proposition 1. *Let $\mathcal{M}$ be an MDP as defined above with action-penalty reward structure, and P deterministic, $\mathcal{M}$ of finite diameter D and $\mathbb{L} = \mathcal{S}$. Let $D', d \in \mathbb{N}$ such that $d < D < D'$. Under A2, for $\epsilon < 1$, then given a sequence of episodic tasks in $\mathcal{M}$ parametrized by goal state, and with initialization of $Q(s,a) = -D'$, $\forall (s,a) \in \mathcal{S} \times \mathcal{A} - \{a_{reflect}\}$ and $Q(s, a_{reflect}) = -d$, $\forall s \in \mathcal{S}$, the lifelong cumulative regret of the LOVR1 agent after T tasks each lasting K episodes is $\leq T\big(2\mathcal{S}\mathcal{A}D + (K-1)\big)$.*

The proof is omitted due to space restrictions, however follows easily from [**?**]. Next we provide a result that shows the asymptotic inefficiency of a LOVR2 agent, associated to forcing the agent to traverse towards a landmark first, before traversing towards the goal state g.

Proposition 2. *Consider LOVR2 and goal-based multi-task RL setting as described above. Given assumptions A1, A2 with $\eta \geq 1$, and environment satisfying κ-approximate reversibility for $\kappa \geq 1$. Let initial and goal states $s, g \in \mathcal{S}$ be arbitrary, then under the optimal LOVR2 policy the per-episode regret, $\mathcal{R}$, is no more than $\eta(1+\kappa)$. Moreover, this proof is tight, in the sense that there exists an MDP such that under these conditions, the bounds are realized (Figure 1).*

Proof. Denote $V^{g*}_{LOVR2}(s)$ the optimal value function associated to the optimal LOVR2 policy that first takes the optimal path landmark option from s, then upon termination of the landmark option, takes the optimal policy from some termination state s' to g. Recall that all value functions are non-positive. Then,

$$
\begin{aligned}
\mathcal{R} &= V^{g*}(s) - V^{g*}_{LOVR2}(s) && \text{definition of regret} \\
&\leq V^{g*}(s) - (V^{l*}(s) + V^{g*}(l)) && \text{see above} \\
&\leq V^{g*}(s) - (V^{l*}(s) + \kappa V^{l*}(g)) && \kappa\text{-approximate reversibility} \\
&\leq V^{g*}(s) - (V^{l*}(s) - \eta\kappa) && \eta\text{-reachability; } \eta, \kappa \geq 1 \\
&\leq V^{g*}(s) - (V^{l*}(g) + V^{g*}(s) - \eta\kappa) && \text{inefficiency of first arriving at } g \\
&\leq V^{g*}(s) - (V^{g*}(s) - \eta - \eta\kappa) && \eta\text{-reachability} \\
&= \eta(1+\kappa).
\end{aligned}
$$

To see that the bounds are tight, we provide an example of an MDP where the bounds are realized (Figure 1). It can be seen that this MDP satisfies κ-approximate reversibility, and condition $A1$. When the initial state is s_0, and the goal state g, the LOVR2 optimal policy will select landmark option o_{l_1}, until the termination condition is satisfied (in this case, $s = l_1$). From l_1, the agent will follow the optimal path to g. Thus, the regret of the optimal LOVR2 agent is $-\frac{1}{p_0} - (-\eta - \frac{1}{p_0} - \kappa\eta) = \eta(1 + \kappa)$.

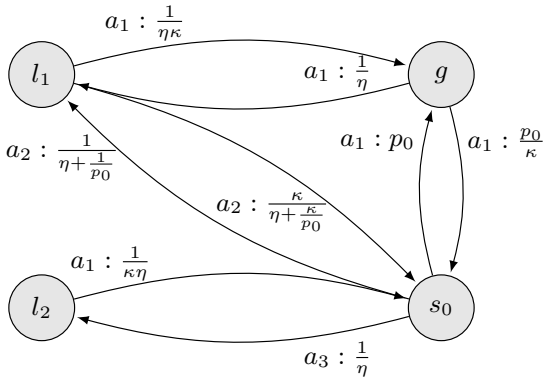

Fig. 1. Example of MDP that realizes bounds of proposition 2. Initial state s_0 with goal task g. The values listed represent the transition probabilities for the given action. For clarity we do not include self-loops in the figure, which occur for each action with probability $1 - p$, for p as listed. We allow $p_0 > 0$ to be arbitrary.

LOVR1 results We first test LOVR1 experimentally using tabular Q-learning in a 15x15 gridworld domain. We assume that the agent is not provided the current goal state g, but only learns of g upon successfully arriving at $g \in \mathcal{S}$. Before g is known, the LOVR1 agent selects actions according to $\pi : \mathcal{S} \to \mathcal{A}$, thus only has access to primitive actions. Upon arrival at the goal state, each subsequent episode the LOVR1 agent can now select actions according to $\pi : \mathcal{S} \to \mathcal{A}_+ := \mathcal{A} \cup \{a_{reflect}\}$, where $a_{reflect}$ selects the landmark option that maximizes $a_{reflect}$ as defined previously. In the first phase, we experiment with hand-selected landmarks $|\mathbb{L}| \in \{9, 25, 225\}$ (with respective pseudo-coverings of $\eta \in \{4, 2, 0\}$), where these associated Q-value functions are learned off-policy while the agent explores the environment taking actions randomly for a fixed number (100k) of steps. During the second phase of learning 100 tasks are sampled uniformly at random and each task is represented by a random goal and initial state, ensuring that $s_0 \neq g$. Each task is run for 1000 episodes where after every exploration episode (using ϵ-greedy, $\epsilon = 0.1$), an evaluation episode is carried out following the greedy policy. Greedy evaluation episodes terminate when the agent arrives at the goal or after for 5000 time steps have passed. For these experiments, as a proof of concept we aim to demonstrate the reduction in regret over a baseline Q-learning agent that does not use any form of transfer, as well as demonstrate the benefits of LOVR1, which constrains which options are available to the agent, over a version of the LOVR1 agent with access to $a_{reflect}$ but

also can select any landmark options independently of $a_{reflect}$. We denote the agent with unrestricted access to the landmark options via its defined action set, $\mathcal{A}_{\mathcal{O}_l} := \mathcal{A}_+ \cup \mathcal{O}_l$. Note that the LOVR1 agent increases its set of actions by only a single action, while the naive $\mathcal{A}_{\mathcal{O}_l}$ agent increases its set of actions by the number of landmark options plus one. We consider both the deterministic setting and the stochastic setting where the agent successfully transitions along the intended cardinal direction with probability $p = 0.85$, and along any of the other cardinal directions uniformly with mass $1 - p$. Under these dynamics, it is clear that for the deterministic setting $\kappa = 1$,while it can be shown that under the stochastic setting, for all values of η cosidered, we have $\kappa < \frac{5}{2}$. Each set of experiments are repeated for n=100 trials with a different random seed.

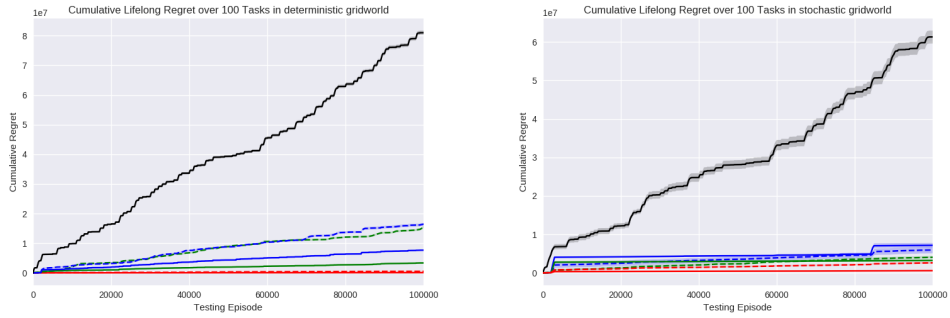

Fig. 2. Cumulative lifelong regret. Left: Deterministic, Right: Stochastic gridworld. Black- Baseline Q-learning agent, Blue-$|\mathbb{L}|$ = 9, Green-$|\mathbb{L}|$ = 25, Red-$|\mathbb{L}|$ = 225 landmarks. Dashed: $\mathcal{A}_{\mathcal{O}_l}$, solid: $\mathcal{A}_+$. Mean +/- 1 s.d. is shown.

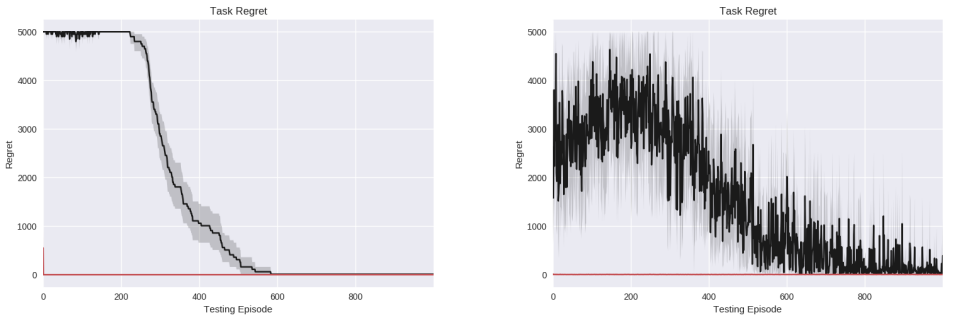

Fig. 3. Single Task Regret (Task 1). Left: Deterministic, Right: Stochastic gridworld. Black-Baseline Q-learning agent, Red-$|\mathbb{L}|$ = 225 landmarks with $\mathcal{A}_+$.Mean +/- 1 s.d. is shown.

Figure 2 demonstrates that the LOVR1 implementations drastically reduce the regret over the lifetime of the agent in both the stochastic and deterministic settings, as

compared to the baseline Q-learning agent with no form of transfer. We observe also that implementations of LOVR1 using $\mathcal{A}_+$ consistently outperform those using $\mathcal{A}_{\mathcal{O}_l}$. These empirical findings are a nice proof of concept, since using $\mathcal{A}_+$ only augments the set of actions by a single element, $a_{reflect}$, while adding the landmark options as well would significantly increase the number of actions, and hence sample complexity and regret, as many landmark options are not useful for a given task, and hence can result in negative transfer. Table 1 demonstrates that though the $\mathcal{A}_{\mathcal{O}_l}$ agent still achieves a reduction in regret over a baseline agent with no options and no ability for transfer, the LOVR1 implementation drastically outperforms it, achieving orders of magnitude mean fold reduction in cumulative lifelong regret.

Table 1. Mean fold reduction in cumulative lifelong regret over 100 tasks as compared to baseline.

$\lvert\mathbb{L}\rvert$,$\mathcal{A}$	9,$\mathcal{A}_+$	9,$\mathcal{A}_{\mathcal{O}_l}$	25,$\mathcal{A}_+$	25,$\mathcal{A}_{\mathcal{O}_l}$	225, $\mathcal{A}_+$	225, $\mathcal{A}_{\mathcal{O}_l}$
Deterministic	10.6	4.9	24.3	5.2	1148.3	159.9
Stochastic	56.9	12.0	186.9	7.2	1491.4	6.8

To demonstrate the regret experienced during a single task, the regret from the first task was plotted for both deterministic and stochastic settings with $L = 225$, using $\mathcal{A}_+$ (Fig. 3). Note these results are quite typical across all tasks. In the deterministic setting, the agent experiences greater regret in the first episode, but upon selecting $a_{reflect}$, follows a landmark option that has extremely small regret for the remaining 999 episodes. In comparison, the flat Q-learning agent with no landmarks must solve each task separately, requiring a few hundred episodes before finding the optimal policy. Similar results can be seen in the stochastic setting. Though the stochastic plot appears to receive no regret for the LOVR agent, very small (but non-zero) regret is accumulated.

Taken together, the results from LOVR1 experiments demonstrate a strong proof of concept for using landmark value functions for transfer learning on future goal-based tasks. For the following set of experiments we test the utility of landmark value functions in transfer learning using function approximation (neural networks).

LOVR2 results We considered next a set of experiments to test the utility of landmark coverings using function approximation (convolutional neural networks), to experiment with a different instantiation of LOVR, specifically LOVR2 (described previously), and the setting where the agent is provided the current goal state g.

For LOVR2 experiments we constructed an MDP environment as seen in Figure 4. The state is an RGB image of size (100,100) pixels. Actions move the avatar in the cardinal directions in a stochastic manner similar to the gridworld used in the LOVR1 experiments. A goal is encoded by the location of the avatar on the map. Here, the LOVR2 agent learns a landmark covering with $|\mathbb{L}| = 8$ in the first phase, and the second phase comprises of 25 goal taks uniformly sampled from $\mathcal{S} \setminus \mathbb{L}$. Each task is run for 1000 episodes, similar to experiments for LOVR1 (ϵ-greedy exploration episode followed by greedy policy evaluation episode). Each episode terminates when either the goal has been reached, or a maximum of 250 time steps have elapsed. The start of each episode the avatar start state is sampled uniformly at random amongst the four corner positions of the

map. Each set of experiments is repeated 5 times. We instantiate the LOVR2 agent using a Deep Q-network (DQN), and compare its performance to a baseline DQN agent with no transfer, and to a baseline transfer agent using a UVFA network trained with a similar phase I using the same landmark goal states as the LOVR2 agent. Note that UVFA encodes the goal state into the state representation, while our LOVR2 implementation does not, but accesses the goal state information via the options framework encoding.

As the main goal of this work is to assess mechanisms for positive transfer learning *given the agent has previously solved* a set of prior tasks, we used supervised learning to train the LOVR2 and UVFA agent networks on learning the optimal Q-value functions for each of the landmark value functions, for both LOVR2 and UVFA agents. We leave a more thorough exploration and study of how to efficiently construct landmark coverings and learning landmark value functions for future work. Hence, beginning the second phase the agent has access to approximately optimal Q-value functions, Q^l_θ, $\forall l \in \mathbb{L}$, parameterized by the network weights θ, and the UVFA agent has access to an approximately optimal UVFA, $U_\theta(s; g)$, $\forall g \in \mathbb{L}$.

Figure 4 (right) shows the mean cumulative lifelong regret of each agent. We see that the baseline DQN agent solves each task, eventually, however without any transfer mechanism is forced to solve each task from scratch, and hence has the highest lifelong regret. As expected, the UVFA agent outperforms the baseline DQN agent, however the LOVR2 drastically outperforms both baseline agents, receiving an almost 4-fold mean reduction in lifelong regret over the baseline DQN agent and an over 3-fold mea reduction in lifelong regret over the UVFA agent (Table 2). Taken together, the two sets of experiments demonstrate empirically that the LOVR framework can provide a principled approach towards transfer learning through landmark options, and can result in a significant reduction in lifelong regret in the goal-based multi-task RL setting.

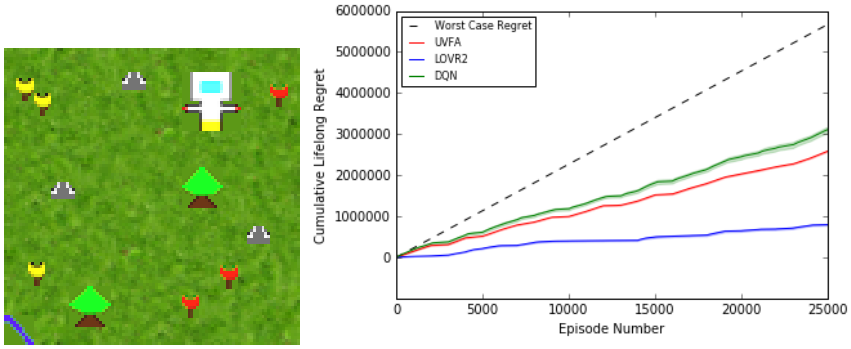

Fig. 4. LOVR2 experimental results. Left: Image of the MDP state space. Right: Cumulative lifelong regret over 25 tasks. Mean across 5 runs +/- s.d. is plotted for various implementations.

Discussion We presented the landmark options via reflection (LOVR) framework as a general approach to solving goal-based multi-task RL problems where the use of options can be shown to reduce regret over the lifetime of the agent. We show that landmark states that suitably cover $\mathcal{S}$ allow for an attractive mechanism for transfer

Table 2. Statistical summary of lifetime of agent implementations. Mean (1 s.d.) x10^5 reported. Fold reduction reported with respect to baseline DQN agent regret. S.C. - sample complexity

Agent	Lifelong S.C.	Lifelong Regret	Fold Reduction in Regret
DQN	36.88 (0.72)	31.07 (0.72)	1.00
UVFA	31.65 (0.12)	25.84 (0.12)	1.20
LOVR2	13.76 (0.22)	7.95 (0.22)	3.91
Oracle	0.58	0	∞

learning through encoding properties of $\{Q^l\}_{l\in\mathbb{L}}$ into the options framework. In doing so, access of a high level controller to a possibly large set of options is restricted in a manner that promotes positive transfer and limits negative transfer across future tasks. For the setup considered in this study, theoretical results with respect to regret bounds as well as empirical results in both tabular and function approximation settings, both demonstrate the strength of LOVR as a principled approach for transfer learning in RL.

One attractive aspect, which we leave for future work, is that by using landmark options, the LOVR agent must only solve the optimal path from l to g that is constrained to lie in an $\eta\kappa$-pseudo ball around l. Given any base learning algorithm with sample complexity and regret dependant on $|\mathcal{S}|$, this $\eta\kappa$-pseudo ball may have a considerably smaller set of states to search over, and thus solving for the optimal policy from l to g requires significantly less interactions with the environment than solving from scratch the optimal policy from s to g, using the same base learning algorithm. Hence, up until some number of episodes K, LOVR will achieve a provable reduction in regret.

As η is a hyper-parameter, this allows for a clear mechanism to trade off what an acceptable level of regret may be for tasks given in the second phase, and the amount of training time that can be allocated to the first phase. For example, given a consumer product such as a floor cleaning agent that can be given commands such as “clean under the table", the commercial success of such an agent would depend on how quickly it can learn a sequence of optimal policies. If an agent had to learn completely from scratch, such a product would fail. However, a LOVR agent trained in a virtual environment on a set of landmark tasks before leaving the factory (phase I) could allow for more efficient transfer to tasks given in the home. The framework presented here is a first step towards a general framework for transfer in goal-based RL, in which the set $\mathbb{L}$ of landmarks plays a key role and will be the subject of further study in follow-up work.

Acknowledgments The authors thank Doina Precup for helpful discussions.

Chapter 4

Conclusion

This thesis concerned itself with two algorithmic approaches to temporally extended actions in goal-based Markov Decision Processes. Chapter 2 included a manuscript that introduced a novel algorithm for learning hierarchical macro actions. Empirical results across multiple domains, including the use of function approximation with deep convolutional neural networks, suggested that this algorithm can confer more efficient learning in goal-based reinforcement learning settings. Chapter 3 included a manuscript that introduced a novel algorithm for building and using a set of landmark options in the multi-task goal-based Markov Decision Process setting. Theoretical results regarding the lifelong regret of the algorithm were provided in two different settings, as well as strong empirical results that demonstrated how LOVR confers a dramatic improvement in empirical lifelong regret as compared to baseline approaches.

With the intuition and understanding gained from these works and results, a number of questions and research directions remain open, specifically with respect to landmark options:

- Chapter 3 assumed that $\mathcal{S}$ was η-reachably covered by the set of landmarks $\mathbb{L}$, and that the environment satisfied κ-reversibility. An interesting direction would be to consider a topological covering of $\mathcal{S}$ under a useful metric, and to investigate how such a covering could improve upon the theoretical properties of the LOVR algorithmic framework.

- An increasingly active area of study in RL and artificial intelligence literature is that of *safety* [12]. One interesting facet of the LOVR algorithm is that, $\forall \ell \in \mathbb{L}, s \in \mathcal{S}, a \in \mathcal{A}$, $Q^{l^*}(s, a)$ encodes the expected number of steps from state s to landmark state ℓ, given the agent takes action a at the current state, but then forever afterwards continues following the associated policy. Moreover, by knowing $V^{l^*}(g)$ for goal state g, the expected number of steps from g to ℓ, and with κ-reversibility, a bound on the number of steps from ℓ to g can be made. The combined information regarding expected first hitting times with respect

to a set of landmark states can be used to *reason* during exploration when the agent is attempting to reach the current goal state g from state s. This can be illustrated by a simple example where, suppose $|V^{l^*}(g)| = 2$, and suppose $\kappa = 1$, then it follows that $|V^{g^*}(l)| \leq 2$, which means it takes no more than 2 steps (in expectation) from state ℓ to reach the goal state under some optimal policy for arriving at g. If at state s, $\exists\ a \in \mathcal{A}$ such that $|Q^{l^*}(s,a)| = 10$, and we were given oracle knowledge that $\tau(g;s) = 2$, then it would follow that with respect to arriving at goal g, $\pi^*(s) \neq a$, since we see that otherwise there exists optimal policies from s to g, then from g to ℓ that would take $2+2 = 4$ steps in expectation, however we know that in actuality the expected number of steps from s to ℓ of any policy that takes action a from s is at least 10 steps. Though this simple example assumes an extra piece of information that LOVR currently does not have access to, that being oracle knowledge of $\tau_{\pi^*}(g;s)$. if bounds on $\tau_{\pi^*}(g;s)$ can be made, then the algorithm could further improve regret by not sampling actions during exploration which cannot possibly be taken by the optimal policy, as described above. Such an action-pruning approach could be interpreted as conferring safety to an agent, as in many aspects of our lives when the consequence that makes an action dangerous is realized, it will tend to lead us astray from our goals and idealized paths.

- A final future direction addresses the elephant in the reinforcement learning room: where do rewards come from? Benchmark domains that RL studies consider are variants of gridworld, video games such as Atari, and other environments that utilize a game engine that is designed by humans. In reality, if a human wanted to use reinforcement learning to train an embodied agent, such as a robot, to "come here", or to " pick up my glass", how would such an agent receive its reward? Encoding such a reward function *a priori* seems doomed to fail, as it requires oracle knowledge of how to measure how good any state an embodied agent may find itself in the *real world* is, with respect to the task at hand. This is far from trivial, and the successful arival at such a goal state follows more of a *family resemblence* in the Wittgenstein sense and is clearly subjective to the issuer of the command. In these settings, it is most likely that an *active human-in-the-loop* reinforcement learning approach must be taken [5], where the agent is provided rewards directly from a human. It would be interesting to apply the LOVR algorithm to such settings, since the general nature of the first phase of LOVR may allow an agent to learn to navigate its environment in a self-supervised manner, learning to reach (landmark state) goals that it sets for itself which could be more quickly transferred to novel tasks assigned by a human.

Appendix A

Due to space restrictions of both manuscripts important technical details, pseudo code and results were ommitted from the submission. We place them in the thesis Appendix for completeness.

A.1 HMAC: Appendix

A.1.1 HMAC Pseudo Code

procedure HMAC WITH TABULARQ-LEARNING(γ, $\{\epsilon_n\}_{n\in N}$, α)
  Initiailize Q_0, $\rho \leftarrow |\mathcal{A}|$, $n \leftarrow 0$
  Initialize previousAction, $a_n \leftarrow \emptyset$, s_0
  for $n \leftarrow 0, 1, 2, ...$ **do**
    previousAction $\leftarrow a_n$
    Draw $\theta \sim$ Bernoulli(ϵ_n)
    if $\theta = 1$ **then**
      $a_n \leftarrow \underset{a\in\mathcal{A}}{\text{argmax}}\ Q_n(s_n, a)$
    else
      $a_n \leftarrow a \sim$ Uniform($\mathcal{A}$)
    end if
    $s_{n+1}, r_{n+1} \sim$envinronment(a_n)
    if episode done **then**
      $Q_{n+1}(s_n, a_n) \leftarrow (1-\alpha)Q_n(s_n, a_n) + \alpha r_{n+1}$
      if $\langle$previousAction, $a_n\rangle \notin \mathcal{A}$ **then**
        $m_\rho \leftarrow \langle$previousAction, $a_n\rangle\}$
        $\mathcal{A} \leftarrow \mathcal{A} \cup \{m_\rho$
        Initialize $Q_{m_\rho} \in \mathbb{R}^{|\mathcal{S}|}$
        Q_{n+1}.append(Q_{m_ρ})
        $\rho \leftarrow \rho + 1$
      end if
    else
      $Q_{n+1}(s_n, a_n) \leftarrow (1-\alpha)Q_n(s_n, a_n) + \alpha[r_{n+1} + \gamma \max_{a\in\mathcal{A}} Q_n(s_{n+1}, a)]$
    end if
  end for
  return Q_{n+1}
end procedure

A.1.2 Experimental Methods

Gridworld. For the gridworld experiments, a 15x15 gridworld was used with $|\mathcal{A}| = 4$ actions, one for each cardinal direction. For the stochastic gridworld, an action was successful with probability 0.85, otherwise the agent moved in one of the other three directions with equal probability. For all experiments the environment begins at state $(2, 2)$ and goal state was $(12, 12)$. The HMAC agent interacted with the environment for 1000 episodes, following $\epsilon = 0.1$ greedy exploration. Every 50 episodes the current policy is evaluated through greedy action selection. A discount factor $\gamma = 0.95$ was used. The Q values were intialized randomly from $Uniform([-19, -20])$. The learning rate was set to 0.1, and each experiment was repeated 100 times.

Doom. The VizDoom "my way home" environment was implemented in OpenAI Gym. The default reward is the following: death = -100, reach goal = 1000, else inversely proportional to distance to goal, however, this endogeous reward was hidden from the agent, which only received the action penalty reward. The state representation is a 60x45 RGB image. The DQN architecture was a standard CNN based DQN (filters, kernel size, stride): (32,8x8,4), (64, 4x4,2), flatten to fc 512 to $|\mathcal{A}| = 4$. The Adam optimizer (learning rate 0.001) was used, discount factor was set to 0.99. The target DQN network was updated every 2500 frames, and the DQN policy was tested (greedy evaluation) every 5000 frames. Batch sizes were set to 32 with an experience replay buffer of 100,000 frames. When adding a new macro action to the HMAC DQN, the same architecture was instantiated except an additional neuron in the output layer. Previous weights were copied from the previous network, and simply initialize the new weights going to the new neuron (vector in $\mathbb{R}^{512}$) randomly. Each set of experiments were repeated 10 times.

A.1.3 Hierarchical Macro Action Analysis

Gridworld. Though showing the learned macro actions for each experiment is infeasible due to the scale of the experiments, the hierarcical macro actions learned from a single experimental run (1000 episodes) were analyzed. Below we see the greedy policy of the agent during evaluation runs at various episodes. As can be seen, as training progresses (episode 0 to 1000) the learned policies are increaslingy efficient, as measured by regret. As seen in the figure, light bulbs represent states where the policy over macro actions made a decision. Hence, sequences of actions were selected between "light bulbs" signifying the selection of a hierarchical macro action.

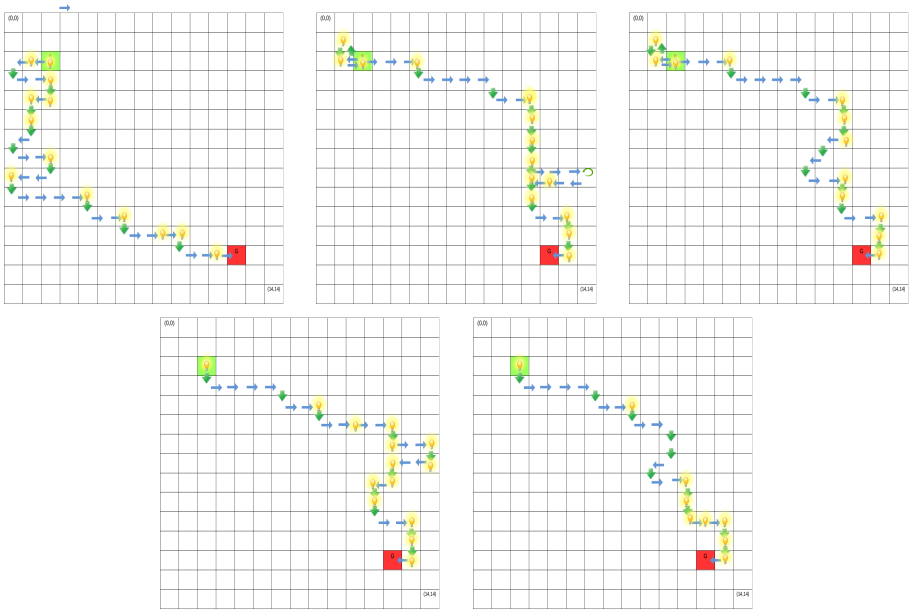

Figure A.1: HMAC policy at episode 50 (top left) (regret = 13), episode 200 and 250 (top middle) (regret = 13), episodes 500 (top right) (regret = 10), episodes 700,750 (bottom left) (regret = 8), episodes 900,950,1000 (bottom right) (regret =4). Green square signifies start state, red square denotes goal state. Arrows denote actions taken. Lightbulbs denote states where the policy made a decision.

Next, we show a single example of a learned hierarchical macro action in terms of its iterative decomposition into other hierarchical macro actions and finally to primitive actions. We demonstrate as an example the hierarchical macro action selected by the agent as seen in Figure A.1 (bottom right) for episodes 900, 950, 1000. We show the hierarchical nature of the macro action m_{13} and m_{41}:

- $m_{13} = \langle m_8, m_6 \rangle$
  - $m_8 = \langle m_6, m_4 \rangle$
    - $m_6 = \langle \text{down}, m_4 \rangle$
    - $m_4 = \langle \text{right, right} \rangle$
- $m_{41} = \langle m_{36}, m_6 \rangle$
  - $m_{36} = \langle m_6, m_{34} \rangle$
    - $m_{34} = \langle \text{down}, m_{15} \rangle$
    - $m_{15} = \langle \text{down, left} \rangle$

We see that m_{13} itself is composed of two lower level hierarchical macro actions. If we "un-roll" the macro action down to its deterministic sequence of primitive actions, it would be $m_{13} = \langle down, right, right, right, right, down, right, right \rangle$. Note that this macro action was selected from the goal state in Figure A.1, whose trajectory can be followed from the initial state. Upon completion of m_{13} the policy over hierarchical macro actions then selected m_{41}, which, "un-rolled" leads to the deterministic sequence of primitive actions: $\langle down, right, right, down, down, left, down, right, right \rangle$.

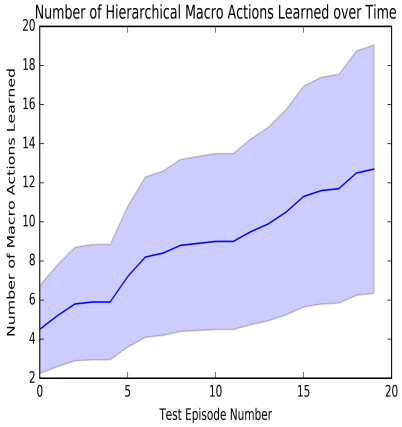

Figure A.2: Number of macro actions learned over time (n=10 experimental repeats).

Doom. In the doom environment, we first plot the number of hierarchical macro actions over time. As seen in Figure A.2, the mean number of hierarchical macro actions at the end of training is around 12. Despite having no mechanism to prevent adding macro actions the DQN acquired a modest number of macro actions.

We repeat an analysis of the learned hierarchical macro actions in the Doom domain. For each experimental repeat, after the final training episode the final policy was evaluated for 10 episodes. A total of 10 experimental repeats were performed. Some interesting results include that for two of the experimental (run 3, run 9) for each episode the higher level policy over hierarchical macro actions acted only once, in the sense that the policy selected a hierarchical macro action at the inital state, and this hierarchical macro action was sufficient to drive the agent to the goal state. Hence, the higher level policy only had to act once. Specifically, for run 3, the macro action learned was:

- $m_{11} = \langle m_{10}, m_{10} \rangle$
  - $m_{10} = \langle m_8, m_6 \rangle$
    - $m_8 = \langle m_6, m_6 \rangle$
      - $m_6 = \langle \text{forward}, \text{forward} \rangle$

Hence, the "un-rolled" hierarchical macro action $m_{11} = \langle f, f, f, f, f, f, f, f, f, f, f, f, f, f, f, f \rangle$, where $f = \text{forward}$.

A.2 LOVR: Appendix

A.2.1 Pseudo Code

We begin by providing the pseudo code for LOVR1, LOVR2 and all the accompanying functions required for the full training procedure. First, the overall LOVR1 algorithm with tabular Q-learning, second the Reflect function, third the TrainLOVR1 pseudo code, finally the Phase I pseudo code.

procedure LOVR1 WITH TABULAR Q-LEARNING($\mathbb{L}, \{Q^\ell\}_{\ell\in\mathbb{L}}$, K, g, ϵ, α)
  Initiailize Q^g
  $\mathcal{A}_s \leftarrow \mathcal{A} - \{a_{reflect}\}$, $\forall s \in \mathcal{S}$
  for $episode \leftarrow 1, 2, ..., K$ **do**
    $n \leftarrow 1$
    $s_n \sim InitialState(g)$
    while $s_n = / = g$ **do**
      $a_n \leftarrow$ EpsilonGreedy(s_n, Q^g, ϵ)
      if $a_n == a_{reflect}$ **then**
        $n^{'}, h, s_{n'} \leftarrow$Reflect$(s_n, \mathbb{L}, \{Q^\ell\}_{\ell\in\mathbb{L}}, \mathbb{L}, n, g)$
      else
        $n^{'}, h, s_{n'} \leftarrow$Env$(s_n, a_n)$
      end if
      if $s_{n'} == g$ **then**
        $Q^g(s_n, a_n) \leftarrow (1-\alpha)Q^g(s_n, a_n) - \alpha h$
      else
        $Q^g(s_n, a_n) \leftarrow (1-\alpha)Q^g(s_n, a_n + \alpha(-h + \max_{a'\in\mathcal{A}_s} Q(s_{n'}, a'))$
      end if
      $n \leftarrow n^{'}$, $s_n \leftarrow s_{n'}$
    end while
    if episode == 1 **then**
      $\mathcal{A}_s \leftarrow \mathcal{A} \cup \{a_{reflect}\}$, $\forall s \in \mathcal{S}$
    end if
  end for
  return Q^g
end procedure

procedure REFLECT($s, \mathbb{L}, \{Q^\ell\}_{\ell\in\mathbb{L}}, \mathbb{L}, n, g$)
  $\ell^* \leftarrow \underset{\ell\in\mathbb{L}}{arg\ max} V^\ell(g)$
  $\beta \leftarrow ceil(|V^{\ell^*}(s)|)$
  steps $\leftarrow 0$
  for $step \leftarrow 1, 2, ..., \beta$ **do**
    $a \leftarrow \underset{a'\in\mathcal{A}}{arg\ max} Q^{\ell^*}(s, a')$
    $s \sim Env(s, a')$
    steps += 1
    if $s == g$ **then**
      return $n+$ steps, steps, s
    end if
  end for
  return $n+$ steps, steps, s
end procedure

procedure TRAINLOVR1($\mathbb{L}, K, T, P, \alpha, \epsilon$)
  $\{Q^\ell\}_{\ell\in\mathbb{L}} \leftarrow PhaseI(\mathbb{L}, P)$ ▷ Phase I
  for $task = 1, 2, ..., k$ **do**
    $goal \leftarrow sampleGoal()$
    $Q^{goal} \leftarrow LOVR1(\mathbb{L}, \{Q^\ell\}_{\ell\in\mathbb{L}}, K, goal, \epsilon, \alpha)$
  end for
end procedure

procedure PHASEI($\mathbb{L}, P, \alpha$)
  for $\ell \in \mathbb{L}$ **do**
    $Q^\ell \leftarrow Initialize()$
    $s \leftarrow InitialState()$
  end for
  for $steps = 1, 2, ..., P$ **do**
    $a \sim RandomUniform()$
    $s' \sim Environment(s, a)$
    for $\ell \in \mathbb{L}$ **do**
      $Q^\ell(s, a) \leftarrow QLearningUpdate(s, s', a, Q^\ell)$
    end for
    $s \leftarrow s'$
  end for
end procedure

LOVR2 Pseudo Code Next, the pseudo code for the LOVR2 experiments are provided.

procedure LOVR2($\mathbb{L}, \{Q^\ell\}_{\ell \in \mathbb{L}}$, K, g, ϵ, α, η)
  Initiailize Q^g
  $\{\mathcal{A}_s\}_{s \in \mathcal{S}} \leftarrow InitializeInitiationSets(\mathbb{L}, \{Q^\ell\}_{\ell \in \mathbb{L}}, \eta, g)$
  for $episode \leftarrow 1, 2, ..., K$ **do**
    $n \leftarrow 1$
    $s_n \sim InitialState(g)$
    while $s_n = / = g$ **do**
      $a_n \leftarrow$ EpsilonGreedy(s_n, Q^g, ϵ)
      if $a_n == a_{reflect}$ **then**
        $Q^g, n', h, s_{n'} \leftarrow$Reflect($Q^g, s_n, \mathbb{L}, \{Q^\ell\}_{\ell \in \mathbb{L}}, \mathbb{L}, n, g, \alpha, \eta$)
      else
        $n^{'}, h, s_{n'} \leftarrow$Env($s_n, a_n$)
      end if
      if $s_{n'} == g$ **then**
        $Q^g(s_n, a_n) \leftarrow (1 - \alpha)Q^g(s_n, a_n) - \alpha h$
      else
        $Q^g(s_n, a_n) \leftarrow (1 - \alpha)Q^g(s_n, a_n + \alpha(-h + \underset{a' \in \mathcal{A}}{max} Q(s_{n'}, a'))$
      end if
      $n \leftarrow n'$, $s_n \leftarrow s_{n'}$
    end while
  end for
  return Q^g
end procedure

procedure REFLECT($Q^g, s, \mathbb{L}, \{Q^\ell\}_{\ell\in\mathbb{L}}, \mathbb{L}, n, g, \alpha, \eta$)
  $\ell^* \leftarrow \underset{\ell\in\mathbb{L}}{arg\ max} V^\ell(g)$
  $steps \leftarrow 0$
  while $|V^{\ell^*}(s)| > \eta$ **do**
    $a \leftarrow \underset{a'\in\mathcal{A}-\{a_{reflect}\}}{arg\ max}\ Q^{\ell^*}(s, a')$
    $s' \leftarrow Env(s, a)$
    if $s' == g$ **then**
      $Q^g(s, a) \leftarrow (1-\alpha)Q^g(s, a) - \alpha$
    else
      $Q^g(s, a) \leftarrow (1-\alpha)Q^g(s, a) + \alpha(-1 + \underset{a'\in\mathcal{A}-\{a_{reflect}\}}{arg\ max}\ Q^g(s', a'))$
    end if
    $steps+ = 1$
    $s \leftarrow s'$
  end while
  return $n+$ steps, steps, s, Q^g
end procedure

procedure INITIALIZEINITIATIONSETS($\mathbb{L}, \{Q^\ell\}_{\ell\in\mathbb{L}}, \eta, g$)
  $\ell^* \leftarrow \underset{\ell\in\mathbb{L}}{arg\ max} V^\ell(g)$
  for $s \in \mathcal{S}$ **do**
    if $|V^{\ell^*}(s)| > \eta$ **then**
      $\mathcal{A}_s \leftarrow \{a_{reflect}\}$
    else
      $\mathcal{A}_s \leftarrow \mathcal{A} - \{a_{reflect}\}$
    end if
  end for
  return $\{\mathcal{A}_s\}_{s\in\mathcal{S}}$
end procedure

A.2.2 Theoretical Results

Due to space restrictions the proof of Proposition 1 from Chapter 3 was ommitted from the manuscript. The Proposition is restated and proven here.

Proposition A.2.1. *Let $\mathcal{M}$ be an MDP as defined above with action-penalty reward structure, and P deterministic, $\mathcal{M}$ of finite diameter D and $\mathbb{L} = \mathcal{S}$. Let $D', d \in \mathbb{N}$ such that $d < D < D'$. Under A2, for $\epsilon < 1$, then given a sequence of episodic tasks in $\mathcal{M}$ parametrized by goal state, and with initialization of $Q(s, a) = -D'$, $\forall (s, a) \in \mathcal{S} \times (\mathcal{A} - \{a_{reflect}\})$ and $Q(s, a_{reflect}) = -d$, $\forall s \in \mathcal{S}$, the lifelong cumulative regret of the LOVR1 agent after T tasks each lasting K episodes is $\leq T\big(2\mathcal{S}\mathcal{A}D + (K-1)\big)$.*

Proof: [16] showed that under deterministic transition dynamics, finite state and action spaces, an *admissible* Q-value initialization with Q-learning updates of the form:

$$Q_{t+1}(s, a) \leftarrow -1 + \underset{a\in\mathcal{A}}{max}\ Q_t(s', a),$$

reaches the goal state from any initial state in at most $2\mathcal{S}\mathcal{A}D$ time steps, and hence is an upperbound on the episodic regret. With this result, then $\forall$ task each lasting for K episodes, the first episode of the task will last at most $2\mathcal{S}\mathcal{A}D$ time steps. After the first episode and knowledge of goal state g, LOVR now has access to $a_{reflect}$. The remaining $K-1$ episodes will be ϵ-optimal by hypothesis, with $\epsilon < 1$, since $a_{reflect}$ will select the landmark option associated to arriving at goal state g and

hence achieve regret less than $K-1$ over the following $K-1$ episodes, each starting from any initial state s. Note that π will automatically select $a_{reflect}$ on the second episode due to the initialization, and will continue to do so for each subsequent episode (due to the initialization and ϵ-optimality of the landmark option). Given that there are T tasks, the bounds follow.

∎

Also due to space restrictions, the proof of tightness for Proposition 2 from Chapter 3 was ommitted from the manuscript. The proof is given here (see Chapter 3 for reference to MDP used in proof).

Proof: First,

$$\begin{aligned} V^{g^*}(s_0) &= p_0(-1 + V^{g^*}(g)) + (1-p_0)(-1 + V^{g^*}(s_0)) \\ &= -p_0 - 1 + p_0 + V^{g^*}(s_0) - p_0 V^{g^*}(s_0) \\ &= V^{g^*}(s_0) - 1 - p_0 V^{g^*}(s_0) \end{aligned}$$

Subtracting $V^{g^*}(s_0)$ from both sides, then solving for $V^{g^*}(s_0)$ yields, $V^{g^*}(s_0) = -\frac{1}{p_0}$. A similar computation shows that $V^{g^*}(\ell_1) = -\eta\kappa$. Now for $V^{g^*}_{LOVR2}(s_0)$,

$$\begin{aligned} V^{g^*}_{LOVR2}(s_0) &= \Big(\frac{1}{\eta + \frac{1}{p_0}}\Big)(-1 + V^{g^*}_{LOVR2}(\ell_1)) + \Big(1 - \frac{1}{\eta + \frac{1}{p_0}}\Big)(-1 + V^{g^*}_{LOVR2}(s_0)) \\ &= \frac{-1}{\eta + \frac{1}{p_0}} + \frac{V^{g^*}_{LOVR2}(\ell_1)}{\eta + \frac{1}{p_0}} - 1 + \frac{1}{\eta + \frac{1}{p_0}} - \frac{V^{g^*}_{LOVR2}(s_0)}{\eta + \frac{1}{p_0}} + V^{g^*}_{LOVR2}(s_0) \end{aligned}$$

Hence,

$$0 = \frac{V^{g^*}_{LOVR2}(\ell_1)}{\eta + \frac{1}{p_0}} - 1 + -\frac{V^{g^*}_{LOVR2}(s_0)}{\eta + \frac{1}{p_0}}$$

Now, we note that for the state ℓ_1, $V^{g^*}(\ell_1) = V^{g^*}_{LOVR2}(\ell_1) = -\eta\kappa$. Isolating for $V^{g^*}_{LOVR2}(s_0)$, we have

$$\frac{V^{g^*}_{LOVR2}(s_0)}{\eta + \frac{1}{p_0}} = -1 - \frac{\eta\kappa}{\eta + \frac{1}{p_0}}$$

Clearly, it follows that $V^{g^*}_{LOVR2}(s_0) = -(\eta + \frac{1}{p_0}) - \eta\kappa$. Together, this shows that the regret is $\frac{-1}{p_0} + (\eta + \frac{1}{p_0}) + \eta\kappa = \eta(1+\kappa)$ ∎

A.2.3 Experimental Methods

Gridworld. For gridworld experiments a 15x15 gridworld was used. Epsilon greedy with $\epsilon = 0.1$ was used. A total of 100 tasks were sampled randomly, each lasting 1000 episodes for each experimental run. A total of 100 experimental runs (repeats) were performed. The learning rate was set at 0.1, no discounting ($\gamma = 1.0$). For Phase I, the agent randomly explored for $1,000,000$ time steps, performing off-policy Q-learning updates for each of the landmark states. $\mathbb{L}$ was picked manually (see below). The initial state for each task was sampled randomly from $\mathcal{S} - \{g\}$. The gridworld dynamics were the same as explained for the HMAC experiments.

DQN. For function approximation experiments, the state representation was a 100x100 RGB image, however, the underlying state space was a 20x20 gridworld. Since the environment constructed, value iteration was used to solve for an approximately optimal Q function, Q^{ℓ}, $\forall \ell \in \mathbb{L}$. For these experiments $|\mathbb{L}vert = 8$. For Phase I, the landmark DQNs were trained via supervised learning using Adam optimizer (learning rate = 0.001) for 5000 epochs using the L1 loss function and batch sizes of 500. The DQN architecture followed the following (filters, kernel size, stride) format: (64, 8x8,4) - (128, 6x6,3) - (256, 4x4, 1) - (512, 3x3, 1) - Flatten to linear - 512 - 4. The UVFA had a similar architecture. UVFA hyperparameters were similar, except the learning rate was optimized over gridsearch $\{1e^{-2}, ..., 1e^{-5}\}$, and used batch sizes of 128. For Phase II, a replay buffer of size 25,000 was used, batch sizes of 64. Each experiment was repeated 5 times. A total of 25 goal tasks were sampled, each lasting 1000 episodes. The target DQN was updated every 1000 time steps. Each episode had a maximum episode length of 250 time steps. The (greedy) policy was evaluated after every episode. The Adam learning rate was set to 10^{-4}.

A.2.4 Landmark Analysis

LOVR1. In this section an analysis of the set of landmarks is provided. First, a figure to demonstrate the layout of the landmarks used for the gridworld experiments for both when $|\mathbb{L}| = 9, 25$. Figure A.3 shows that the landmarks selected were evenly spaced out throughout the state space (gridworld).

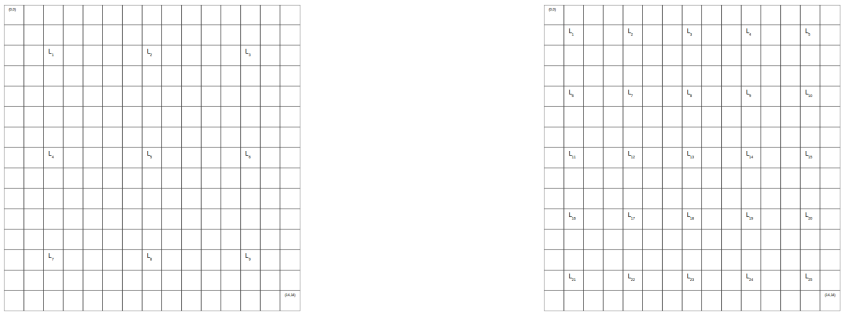

Figure A.3: LOVR Gridworld layout with (Left): 9 landmark locations; (Right): 25 landmark locations.

Next, samples of landmark value functions and landmark policies from the set of experiments with $|\mathbb{L}| = 9, 25$ are provided. As seen in Figure A.4, the landmark centric value functions for landmarks 1,5,9 are represented using a heatmap over the gridworld. A darker color signifies a higher value,

and a lighter color signifying a lower value. Figure A.4 also shows the corresponding policy for each of the states as represented by a colored map where 0 (green) corresponds to the action "up", 1 (gray) corresponds to the action "right", 2 (light green) corresponds to the action "down", and 3 (yellow) corresponds to the action "left". Figure A.5 is similar, except it samples landmarks 6, 12, 18 from the $|\mathbb{L}| = 25$ set of experiments.

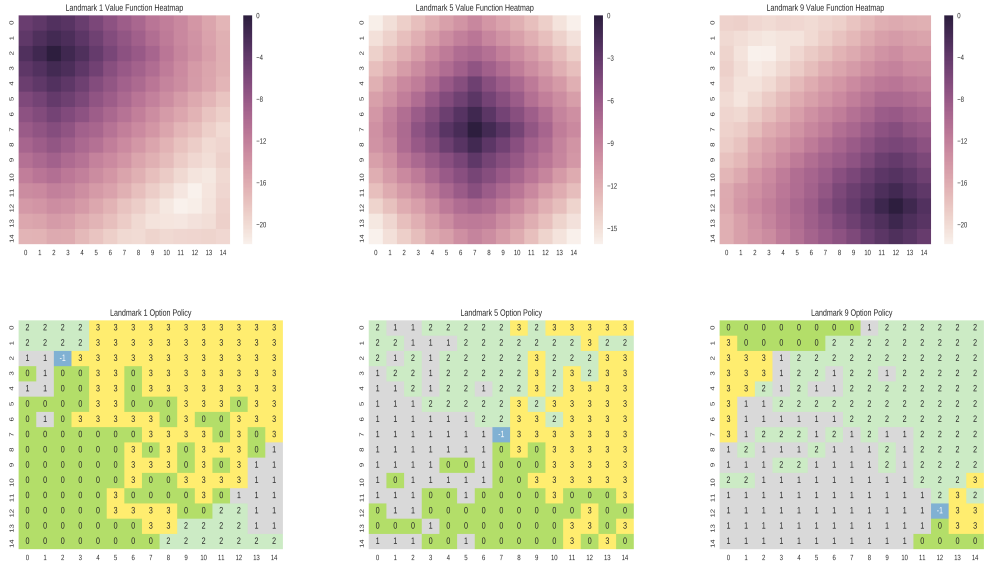

Figure A.4: (Top) Landmark centric value functions and (bottom) policies learned from during phase I from landmarks 1,5,9. 0=Up, 1=Right, 2 = Down, 3=Left

LOVR2. As there are only 8 landmarks for the LOVR2 experiments, the value function and policy for each landmark state is provided in Figure A.5, A.6 and A.7. For the value function heatmaps, a lighter/brighter color corresponds to a higher state value, while a darker corresponds to a lower value state. For the policy plot, the action "up" corresponds to the lightest color (0 on the legend), "down" corresponds to the lightest shade of blue (1 on the legend), "right" corresponds to the dark blue color (2 on the legend) and finally "left" corresponds to the darkest color (3 on the legend).

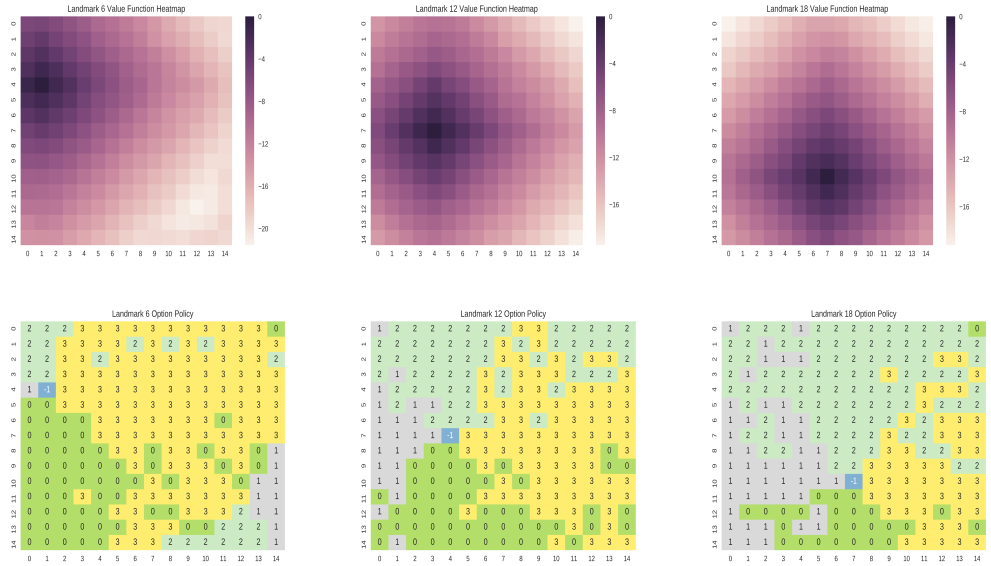

Figure A.5: (Top) Landmark centric value functions and (bottom) policies learned from during phase I from landmarks 6,12,18. 0=Up, 1=Right, 2 = Down, 3=Left

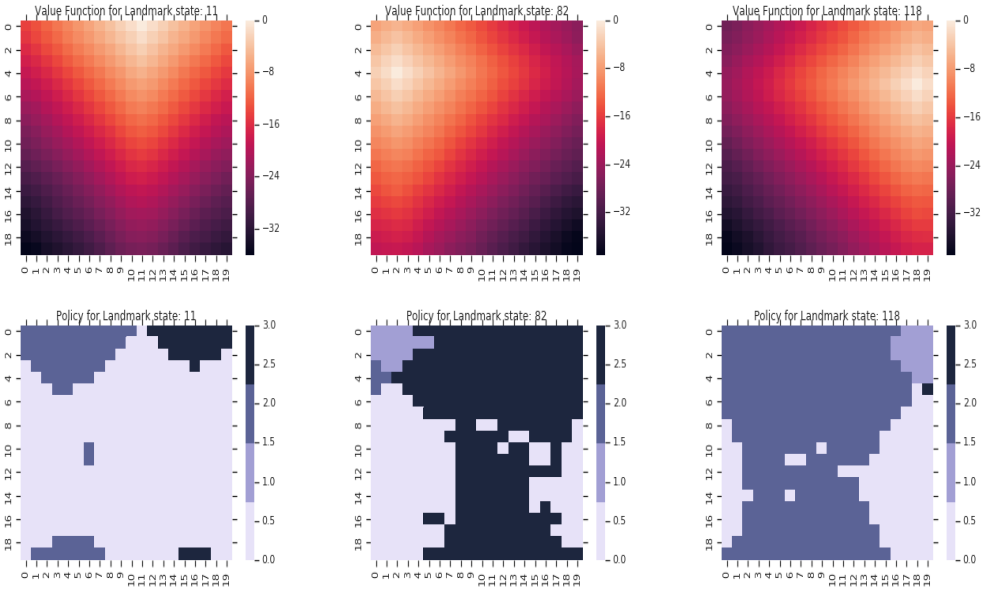

Figure A.6: (Top) Landmark centric value functions and (bottom) policies learned from during phase I from landmarks 1,2,3 . 0=Up, 1=Down, 2 = Right, 3=Left

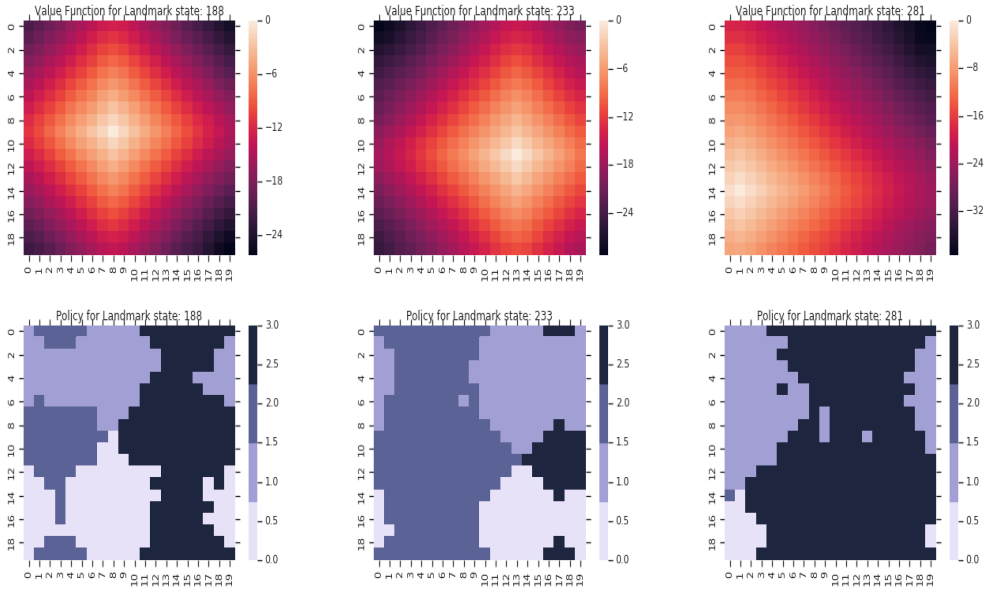

Figure A.7: (Top) Landmark centric value functions and (bottom) policies learned from during phase I from landmarks 4,5,6. 0=Up, 1=Down, 2 = Right, 3=Left

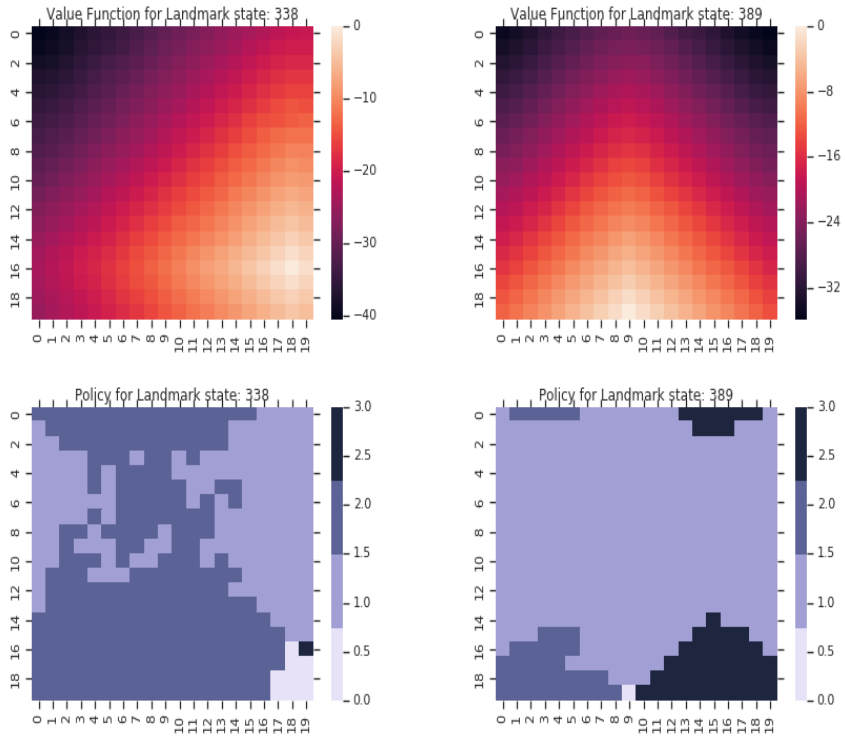

Figure A.8: (Top) Landmark centric value functions and (bottom) policies learned from during phase I from landmarks 7,8. 0=Up, 1=Down, 2 = Right, 3=Left

Bibliography

[1] A.G. Barto A. McGovern. Automatic discovery of subgoals in reinforcement learning using diversity density. 2001.

[2] S. Banach. Sur les operations dans les ensembles abstraits et leur application aux equations integrales. *Fund. Math.*, 3:133–181, 1922.

[3] Peter L. Bartlett and Ambuj Tewari. Regal: A regularization based algorithm for reinforcement learning in weakly communicating mdps. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, UAI '09, pages 35–42, Arlington, Virginia, United States, 2009. AUAI Press.

[4] A.G. Barto B.C. da Silva, G. Konidaris. Learning parameterized skills. 2012.

[5] W. Bradley Knox and P. Stone. Tamer: Training an agent manually via evaluative reinforcement. In *2008 7th IEEE International Conference on Development and Learning*, pages 292–297, 2008.

[6] S. Levine C. Finn, P. Abbeel. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

[7] P. Dayan C.J.C.H. Watkins. Q-learning. *Machine Learning*, 8:279–292, 1992.

[8] S. Singh D. Precup, R.S. Sutton. Theoretical results in reinforcement learning with temporally asbtract options. In *10th European conference on Machine Learning*, pages 382–393, 1998.

[9] P. Dayan and G.E. Hinton. Feudal reinforcement learning. In *NIPS*, pages 271–278, 1998.

[10] S. Mannor D.J. Makowitz, T.A. Mann. Adaptive skills adaptive partitions (asap). 2016.

[11] L. Li E. Brunskill. Sample complexity of multi-task reinforcement learning. In *Conference on Uncertainty in Artificial Intelligence (UAI)*, 2013.

[12] F. Fernandez J. Garcia. A comprehensive survey on safe reinforcement learning. *Journal of Machine Learning Research*, 16:1437–1480, 2015.

[13] S. Kakade. *On the Sample Complexity of Reinforcement Learning.* PhD thesis, Gatsby Computational Neurosceince Unit, University College London, 2003.

[14] M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49:209–232, 2002.

[15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[16] S. Koenig and R.G. Simmons. Complexity analysis of real-time reinforcement learning. *AAAI*, pages 99–105, 1993.

[17] G. Konidaris and A. Barto. Building portable options: skill transfer in reinforcement learning. *IJCAI*, pages 895–900, 2007.

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

[19] Y. Liu and E. Brunskill. When simple exploration is sample efficient: identifying sufficient econditions for random exploration to yield pac rl algorithms. In *European Workshop on Reinforcement Learning*, 2019.

[20] L.P. Kaelbling T. Dean C. Boutilier M. Hauskrecht, N. Meuleau. Hierarchical solution of markov decision processes using macro-actions. In *Fourteenth Annual Conference on Uncertainty in Artificial Intelligence*, pages 220–229, 1998.

[21] Ishai Menache, Shie Mannor, and Nahum Shimkin. Q-cut - dynamic discovery of sub-goals in reinforcement learning. In *Proceedings of the 13th European Conference on Machine Learning*, ECML '02, pages 295–306, London, UK, UK, 2002. Springer-Verlag.

[22] T. Zhang O. Shamir. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal average schemes. volume 28, 2013.

[23] T.J. Perkins and D. Precup. Using options for knowledge transfer in reinforcement learning. Technical report, 1999. Technical Report UM-CS-99-34.

[24] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.

[25] T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal value function approximators. In *ICML*, 2015.

[26] J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[27] A. Strehl, L. Li, E. Wiewiora, J. Langford, and M. Littman. Pac model-free reinforcement learning. In *ICML*, pages 881–888, 2006.

[28] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 2016.

[29] S.R. Sutton, D. Precup, and S. Singh. Beteween mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112:181–211, 1999.

[30] P. Auer T. Jaksch, R. Ortner. Near-optimal regret bounds for reinforcement learning. *Journal of Machine Learning Research*, 11:1563–1600, 2010.

[31] D. Precup T.A. Mann, S. Mannor. Approximate value iteration with temporally extended actions. *Journal of Artificial Intelligence Research*, 53:375–438, 2015.

[32] S. Mannor T.A. Mann. Scaling up approximate value iteration with options: better policies with fewer iterations. In *Proceedings of the 31st International Conference on Machine Learning*, 2014.

[33] D. Silver A.A. Rusu J. Veness M.G. Bellemare A. Graves M. Riedmiller A.K. Fidjeland G. Ostrovski S. Petersen C. Beattie A. Sadik I. Antonoglou H. King D. Kumaran D. Wierstra S. Legg D. Hassabis V. Mnih, K. Kavukcuoglu. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2015.