UNIVERSITEIT VAN AMSTERDAM

MSc Artificial Intelligence

Master Thesis

Temporally Extended Variational Option Discovery

by

Gabriele Bani

11640758

October 26, 2020

36 European Credits

February 2019 - October 2020

Supervisor:
Dr. Herke van Hoof

Assessor:
Dr. Efstratios Gavves

Acknowledgements

This thesis has been a long and beautiful journey, where I learned a great deal about a topic that I always wanted to dive deep in and understand. Not only I had the unique chance to completely choose my path, but I also had the privilege to try to expand research solving a problem that I truly cared for.

I would like to thank my supervisor Herke, who guided me in this process, and was always supportive and at the same time always analytical of all the ideas that were passing through my mind. I really appreciated your critical eye and valuable insights, together with the patience and support you showed towards me.

A big, huge, thank you to all my friends all around the world, all of you lot. Special Thanks to Gabri, Andrii and Davide, who made these years special and fun.

Infine, voglio ringraziare i miei genitori e mio fratello, che continuano a supportarmi ogni giorno con fiducia, entusiasmo e formaggio.

Contents

1 Introduction

Reinforcement learning is a very active area of research, which has been able to solve many problems and is promising to solve more. In the recent years, Reinforcement learning (RL) has been successfully applied to a variety of tasks, often achieving superhuman performance. These task include the famous chess and go games (Silver, Huang, et al., 2016) (Schrittwieser et al., 2019), video games (Mnih, Kavukcuoglu, Silver, Rusu, et al., 2015) (Vinyals et al., 2019), and robotic manipulation tasks (S. Levine et al., 2016) (Bousmalis et al., 2018). In this thesis we are going to focus on Intrinsically Motivated approaches which, unlike standard Reinforcement Learning approaches, do not use extrinsic rewards which are given by the environment. Instead they use intrinsic rewards that come from the agent itself and allows drive the agent to obtain meaningful behaviors without external feedback. This has the goal of automatically learning new behaviors and information in an incremental way, without a specific goal, and makes it easier to build knowledge that is, for example, generalizable in different environments.

In this thesis, we want to focus on the problem of discovering skills (also called low-level policies or options) without the use of extrinsic rewards. In particular, although many approaches manage to learn skills on a variety of environments (Gregor et al., 2016) (Achiam et al., 2018) (Eysenbach et al., 2018) (Co-Reyes et al., 2018) (Sharma et al., 2019), they all exhibit the limitation of learning fixed length options. That is, options are *always* run for a fixed number of steps. We want to work on removing this limitation and allow for discovery of options with arbitrary length, based on what is most useful for a certain environment.

One limitation of current methods that work with fixed length options is that this length has to be chosen in advance, and thus requires prior knowledge about the environment. Even more: this means that the option cannot be used in similar environments but with a different size. For instance if in a gridworld we learned a policy "go to the left wall", when using the option in a gridworld with similar shape but different size, the fixed length option might stop before actually arriving to the wall. Instead, a Temporally Extended Options could correctly terminate to the left wall, and have different lengths even within the same environment. In environments with stochastic dynamics, they can even be able to adjust to noise as they only terminate once their goal has been reached, which might not happen in the fixed length case due the noise itself. Fixed length is even more limiting in goal oriented approaches, where goal selection has to take into account the max length, and cannot just learn to set arbitrary goals. This problem is usually addressed by using multiple hierarchies of policies, where the real end goal is chosen by a high level policy and low-level policies only have to deal with intermediate goals. These intermediate goals do now however necessarily correspond to semantically meaningful behaviors, which could be instead attained by using policies of flexible length.

The problem of finding a way to achieve options of varying length is very promising also because it can lead to a general way of pretraining hierarchical models without constraining on the length of low level policies. This would allow to learn more semantically meaningful options, which can be more useful for high level policies because when they terminate, we know already in which state they end up in. Let's take the example of an option "cook pasta". With a fixed length option, we would not be certain whether it would be completed within the allocated time. Instead, if the behavior would only terminate when the task is completed, we would be sure that whenever we use the option "cook pasta", we end up in a very specific state. This showcases why Temporally Extended Options could be incredibly beneficial for model based learning, where we need to learn accurate estimates of the effects of the actions performed by the agent. Usually, although it has many benefits, such as increasing sample efficiency, it is a very difficult to learn correct models. By optimizing directly for predictability of options, together with flexible options, we allow models which are more semantically meaningful, and thus possibly easier to learn.

To learn Temporally Extended Options, we use a termination function inspired by (Bacon et al., 2016). This is a state dependent function $\beta(s)$, which represents a probability from 0 to 1, and at each step allows the agent to choose whether to terminate the episode before taking the next action. We also investigate an alternative approach could be applicable for discrete action spaces, which consists in adding an additional termination action a_T to the set of actions available. This has the advantage that it can work directly with any reinforcement learning algorithm, but is not applicable for continuous action spaces.

In this thesis, we expand the capabilities of Variational Option Discovery algorithms (Gregor et al., 2016), (Achiam et al., 2018), (Eysenbach et al., 2018) and describe a framework for learning options of flexible length just by using intrinsic rewards. The main goal of this thesis is showing how Temporally Extended Options can be learned without extrinsic rewards, and demonstrating that the learned options exhibit meaningful behaviors, which can be comparable with those learned by fixed length methods but are more flexible in terms of reaching different goals at different distances and do not waste time once they reach their goal.

2 Background

2.1 Reinforcement Learning

Reinforcement Learning deals with the problem of sequential decision making, and learning how to make decisions by interacting with an environment. A Reinforcement Learning agent tries to achieve some goal by maximizing rewards received through interaction.

Reinforcement Learning agents perform actions that influence both immediate rewards and future situations, which in turn influence future rewards, requiring a trade-off between immediate and future rewards. Also, maximizing delayed rewards typically requires a balance between exploration and exploitation: while the agent wants to act according to what it believes is the optimal behavior, it might want to act in different ways to explore new parts of the environment, which might lead to higher rewards. We start the chapter by introducing the concept of Markov Decision Process, which is the way most Reinforcement Learning problems are framed.

2.1.1 Markov Decision Processes

Reinforcement Learning is typically modeled as a Markov Decision Process (MDP). Formally, a Markov decision process is a set (S, A, R, P, γ), where:

- S is a set of states
- A is a set of actions available
- $P(s'|s,a)$, the probability of transitioning from state s to state s' after executing action a. When making the time step explicit, we use the notation $P(S_{t+1}|S_t, A_t)$.
- $R(s', a, s)$ the reward obtained after the transition from s to s' after executing action a. When making the time step explicit, we use the notation $R(S_{t+1}, A_t, S_t)$,

- $0 \leq \gamma \leq 1$ is a scalar called discount factor, which handles the trade-off between immediate and future rewards

The main characteristic of MDPs is that the Markov property holds: the effects of an action taken in a state depend only on that state and not on prior history. This is the reason why both the transition probability P and the reward R at time t never depend on states or actions of times $< t$. With this property we are assuming that current state contains sufficient information to achieve optimal behavior in the MDP.

It is important to notice that time steps of an MDP are discrete, represented by integers. This can be natural for some cases such as a chess game, where we have a clear way of representing turns as integers. However, MDPs can easily be applied also in those cases where an agent has to interact with an environment in real time, even if the interval of time between one decision and the next one is not the same. As long as the agent is able to correctly observe states at multiple times, the framework can be applied. This makes the MDPs applicable to a wide range of sequential decision problems, as algorithms to solve MDPs can be usually applied to different MDPs with very little change.

The objective in an MDP is maximizing the expected cumulative reward starting from a certain initial states. In general the cumulative reward, also called return, starting at time t is typically indicated as G_t, and for a trajectory $\tau = S_0, A_0, R_1, S_1, A_1, R_2, .., S_{T-1}, A_{T-1}, R_T, S_T$, for any $0 \leq t \leq T$ it is defined as

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + R_T = \sum_{k=t+1}^{T} \gamma^{k-t-1} R_k \tag{2.1}$$

Where we use $R_{t+1} = R(S_{t+1}, A_t, S_t)$ for simplicity. Here we can see the effect of having a discount factor γ, which is typically used both for handling continuing problems and to trade off immediate and future rewards. We do not deal with continuing problems in this thesis as we always allow the agent to run for only up to a certain number T_{max} of steps. The more a reward comes from a far away future state, the more it is discounted, as . Thus, the value of the discount factor γ allows to control how much importance we want to give to immediate rewards compared to future rewards. With a discount factor of 0, we only care about choosing actions which lead to an immediate highest reward. Instead, by increasing γ, we allow the agent to care increasingly more about rewards in the future.

To act in the environment, we need a policy $\pi(a|s)$ which chooses an action in each state. More precisely, a policy π defines a probability distribution over the actions $a \in A$ in every state $s \in S$.

Before discussing how to find optimal policies with Reinforcement Learning, we want to mention that Markov Decision Processes can be solved by techniques different from Reinforcement Learning. In particular, when an exact model of the MDP is known, it is possible find an optimal solution by using classical dynamic programming, which relies on using the recursive formulation of the problem, allowing to solve subproblems and efficiently storing results in memory. When the MDP is too large however, Dynamic Programming becomes infeasible. Furthermore, when dealing with problems with continuous actions or states, standard Dynamic Programming algorithms cannot be applied directly.

Reinforcement Learning can be instead be used even in these cases, as it relies on sampling to optimize for returns. This means that in all the cases where an exact model of the MDP is not available, we can still learn optimal behaviors by interacting with the environment. For large action and state spaces instead, RL can benefit from function approximation, especially by using neural networks, which have allowed to solve very complex tasks such as Go (Silver, Huang, et al., 2016) (Silver, Schrittwieser, et al., 2017), Chess (Silver, Hubert, et al., 2017) and Atari (Mnih, Kavukcuoglu, Silver, Rusu, et al., 2015) (Mnih, Kavukcuoglu, Silver, Graves, et al., 2013) games in the last few years.

2.1.2 Value Functions

Most Reinforcement Learning algorithms use the concept of state value function $v_\pi(s)$, which is the expected return of a policy π starting from state s

$$v_\pi(s) \doteq \mathbb{E}_\pi\left[G_t|S_t = s\right] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s\right] \tag{2.2}$$

The expectation over π is an expectation over all trajectories that can be generated by acting through π in the environment. The probability of a trajectory $\tau = (S_0, A_0, R_1, S_1, A_1, R_2, .., S_{T-1}, A_{T-1}, R_T, S_T)$ is

$$p(\tau) = P(S_0)\prod_{t=0}^{T-1} P(S_{t+1}|S_t, A_t)\pi(A_t|S_t) \tag{2.3}$$

where $P(s_0)$ indicates a prior distribution over starting states. Many algorithms also use another value function called the state-action value function, which represents the expected return of a policy starting from state s using action a

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi[\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|S_t = s, A_t = a] \tag{2.4}$$

Recall the definition of return G_t in 2.5, and notice that it can be expressed in a recursive manner:

$$G_t = r_t + \gamma G_{t+1} \tag{2.5}$$

Similarly, we can write value functions in a recursive manner

$$\begin{aligned} v_\pi(s) &\doteq \mathbb{E}_\pi[G_t|S_t = s] && (2.6)\\ &= \sum_a \pi(a|s)q(s,a)] && (2.7)\\ &= \sum_a \pi(a|s)\sum_{s',r} p(s',r|s,a)[r+\gamma v_\pi(s')] && (2.8) \end{aligned}$$

This recursive formulations is an instance of Bellman Equation, and can be used to perform full updates when the model (P and R) is fully known, for example using policy improvement, policy iteration or value iteration. It can also be used to derive algorithms based on Monte Carlo sampling, where trajectories are collected by acting in the environment in order to perform updates. Finally it is also possible to use temporal difference (TD) methods, which do not need the knowledge of the model and do not need full trajectory rollouts, but use bootstrapping to learn value estimates. In all these cases, the aim is learning a correct estimate of the value function, and derive a policy implicitly, for example by an $\epsilon-$greedy strategy over q values.

One of the most successful approaches is Q-learning, which is an algorithm used to learn an optimal state-value function in the following way

$$q(S_t,A_t) = q(S_t,A_t) + \alpha[R_{t+1} + \gamma \max_a q(S_{t+1},a) - q(S_t,A_t)] \tag{2.9}$$

Maybe put a one line explanation of what an off-policy method is

Where transitions are typically collected according to an ϵ-greedy policy over Q-values. A particularly successful approach is Deep Q-Network (DQN), which use a deep neural network Q-function approximation, and have been able to achieve human level performance in several Atari (Bellemare et al., 2013) games (Mnih, Kavukcuoglu, Silver, Graves, et al., 2013) (Mnih, Kavukcuoglu, Silver, Rusu, et al., 2015) by using pixels as observations. Several improvements (such as (Van Hasselt et al., 2016), (Wang et al., 2015), (Schaul et al., 2015)) have been suggested to improve on the original DQN approach, and in (Hessel et al., 2018) these improvements have been combined to improve on the state of the art on Atari games.

2.1.3 Policy Search

Differently from value function approaches, where the policy is obtained implicitly through a learned value function, here we discuss methods to directly learn a parametrized policy that can select actions. There are many reasons for learning a policy in this way. First, in many cases it is easier to represent a policy rather than estimating its exact value function, and it can happen that small changes in the value function represent drastic change in a policy. Furthermore, value methods do not have a natural way of finding stochastic policies, and they can also learn continuous output actions. Finally, important for this work, it is much easier to include prior knowledge during the initialization of policies.

Given a parametrized policy, we want to find best parameters $\boldsymbol{\theta}$, by solving an optimization problem, where the objective is typically the expected return of the policy. There are approaches that do not use gradient over some scalar objective. Some examples are Simplex method, genetic algorithms, cross entropy method and covariance matrix adaptation. Gradient free methods often have the advantage that they are easily parallelizable, although they are typicallly not sample efficient.

In the following methods, we collect sampled trajectories by acting in the environment, and use gradient ascent to optimize our objective function

$$J(\theta) = v_{\pi_\theta}(S_0) = \mathbb{E}_{\pi_\theta}[G_0] \tag{2.10}$$

Where we have a policy with parameters $\boldsymbol{\theta}$ and for simplicity we assume episodes always start in some state s_0. In gradient based methods, we optimize the parameters of the policy iteratively by calculating the gradient of the objective function $J(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t) \tag{2.11}$$

Calculating this gradient directly is however not straightforward. The expected return, our objective function, depends both on states visited and actions chosen in a trajectory. The effects of the actions on the returns can be computed easily, so we can directly calculate the gradient of the expected return given the parameters for this. However, the effects of the policy on the state distribution cannot be computed directly. Since the state distribution is usually unknown, we need a way to calculate the gradient of the expected return with respect to the policy parameters θ without needing to know the gradient of the state distribution. This can be done through the policy gradient theorem (as stated in (Sutton, McAllester, et al., 2000), (Sutton and Barto, 2018)), which states that, for the episodic case

$$\nabla J(\theta) = \sum_s \mu(s) \sum_a \nabla \pi_{\boldsymbol{\theta}}(a|s,\theta) q_{\pi_{\boldsymbol{\theta}}}(s,a) \tag{2.12}$$

$$= \mathbb{E}\Big[\sum_a \nabla \log \pi_{\boldsymbol{\theta}}(a|s,\theta) q_{\pi_{\boldsymbol{\theta}}}(s,a)\Big] \tag{2.13}$$

Where $\mu(s)$ is the discounted state distribution $\mu(s) = \sum_{t=0}^{\infty} \gamma^t P(S_t = s|s_0, \pi)$. The fact that the state distribution appears without a gradient allows to take a sampled estimate of the gradient, by calculating the gradient given one or more trajectories generated by acting directly in the environment. In this way, we can update the policy parameters by stochastic gradient ascent. In the most simple formulation, we do not take an expectation over all actions but only sample one, and use the sampled return G as $\mathbb{E}[q(S_t, A_t)] = \mathbb{E}[G_t]$

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \mathbb{E}_{\pi_\theta}\left[\sum_a \pi_{\boldsymbol{\theta}}(a|S_t)\frac{\nabla \pi_{\boldsymbol{\theta}}(a|S_t)}{\pi_{\boldsymbol{\theta}}(a|S_t)} q_{\pi_\theta}(S_t, a)\right] && (2.14)\\
&= \mathbb{E}_{\pi_\theta}\left[\sum_a \pi_{\boldsymbol{\theta}}(a|S_t) q_{\pi_\theta}(S_t, a) \nabla \log \pi_{\boldsymbol{\theta}}(a|S_t)\right] && (2.15)\\
&= \mathbb{E}_{\pi_\theta}\left[\nabla \log \pi_{\boldsymbol{\theta}}(A_t|S_t) q_{\pi_\theta}(S_t, A_t)\right] && (2.16)\\
&= \mathbb{E}_{\pi_\theta}\left[\nabla \log \pi_{\boldsymbol{\theta}}(A_t|S_t) G_t\right] && (2.17)
\end{aligned}$$

The parameters of the policy can be then updated in the following way

$$\boldsymbol{\theta}_{t+1} \doteq \boldsymbol{\theta}_t + \alpha \log \nabla \pi(A_t|S_t) G_t \tag{2.18}$$

This kind of policy gradient algorithm is called REINFORCE (Williams, 1992), and although it is very easy to implement, the gradients have high variance due to the sampling. One way to reduce variance of updates is introducing a constant or state dependent baseline, which can be proven to not bias the gradient. REINFORCE's gradient with a baseline can be written as

$$\begin{aligned}
\nabla J(\boldsymbol{\theta}) &= \mathbb{E}_{\pi_\theta}[\nabla \log \pi_{\boldsymbol{\theta}}(A_t|S_t)(G_t - b)] && (2.19)\\
&= \mathbb{E}_{\pi_\theta}[\nabla \log \pi_{\boldsymbol{\theta}}(A_t|S_t)(G_t - v_\xi(s))] && (2.20)\\
& && (2.21)
\end{aligned}$$

Where in the second line a learned approximation v_ξ parametrized by ξ of the state-value function is used as baseline.

2.1.4 Actor-Critic methods

Policy Gradient algorithms typically have large variance and have poor sample efficiency, especially when dealing with long trajectories. Instead of relying on sampled rewards, we can introduce an approximation of the expected return with a critic $v_\xi(s)$, and use it to update the policy parameters. This is called bootstrapping, as we update the value estimate for a state using the estimates for subsequent states, introducing a bias. However, the usage of a critic is usually beneficial because it reduces variance and speeds up learning. Finally, using Actor-Critic methods allows us to work conveniently with continuing problems.

We report here a simple way to obtain a TD(0) Actor-Critic update from the update rule of REINFORCE, from Sutton, chapter 13.5 (Sutton and Barto, 2018):

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha(G_t - v_\xi(S_t)) \log \nabla \pi(A_t|S_t) \tag{2.22}$$

$$= \boldsymbol{\theta}_t + \alpha \log \nabla \pi(A_t|S_t)(R_{t+1} + \gamma v_\xi(S_{t+1}) - v_\xi(S_t)) \tag{2.23}$$

Notice that the bias is introduced in the second line, as we make the approximation $V(s_{t+1}) \approx V_\theta(s_{t+1})$. This bias is reduced the more the estimate of the value function gets close to the true expected return.

The update for the critic can be done by TD(0) semi gradient, in the following way

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \alpha^w(R_{t+1} + \gamma v_\xi(S_{t+1}) - v_\xi(S_t))\nabla_{\mathbf{w}} v_\xi(S_t) \tag{2.24}$$

Note that instead of using just one step estimates of the return, we can use arbitrary n-step approximations of it, where the $n-$step return is defined as $G_t^n = \sum_{k=0}^{N-1} \gamma^k R_{t+k+1} + \gamma^N v(S_{t+N})$

In the last years, Actor-Critic methods have become the state of the art for solving many problems, particularly when combined with neural networks. Asynchronous Advantage Actor Critic (Mnih, Badia, et al., 2016) (A3C) was the first on-policy method to successfully use neural networks without the need for an experience replay on tasks as complex as Atari games. A3C uses parallel actors to collect experience, and performs asynchronous updates on both actor and critic, which share the layers of a neural network to increase sample efficiency. Later, researchers found out that the benefit from parallel execution was often negligible, meaning that the highest benefit lies in collecting batches of experience to perform updates.

TRPO (Schulman, Sergey Levine, et al., 2015) successfully managed to solve high dimensional control problems by introducing a constraint that enforces the new policy to be not too far from the old policy. In practice this is done by including a quadratic approximation of the constraint in the loss term, which is expensive to compute. To avoid this, PPO (Schulman, Wolski, et al., 2017) uses a clipped objective to approximate TRPO updates, often outperforming its predecessor while requiring less computational effort.

As Actor-Critic methods often need to choose a suitable n for the $n-$step return to achieve best performance, (Schulman, Moritz, et al., 2015) Generalized Advantage Estimation (GAE) has been introduced to use an average over $n-$step advantages as critic, allowing to trade off bias and variance.

2.1.5 Policy Gradients with log-derivative trick

Policy gradient updates for a policy $\pi_\theta(a|s)$ can be derived using the following log-derivative trick

$$\nabla_\theta \mathbb{E}_{x\sim p_\theta}[f(x)] = \nabla_\theta \int p_\theta(x) f(x) dx = \int \frac{p_\theta(x)}{p_\theta(x)} \nabla_\theta p_\theta(x) f(x) dx \tag{2.25}$$

$$= \int p_\theta(x) \nabla_\theta \log p_\theta(x) f(x) dx \tag{2.26}$$

$$= \mathbb{E}_{x\sim p_\theta}[f(x) \nabla_\theta \log p_\theta(x)] \tag{2.27}$$

This result is used to obtain equation 4.52

2.2 Information Theory

In this thesis, we will use several concepts from Information Theory, such as Entropy and Mutual Information.

2.2.1 Entropy

Consider a probabilistic event E that occurs with probability $p(E)$. The "surprisal" value $\log \frac{1}{p(E)}$ can be used to indicate how surprised we should be when the event E occurs: events with small probability have high surprisal value, and vice versa. The entropy of a random variable X can be interpreted as a measure of the expected surprisal value, or uncertainty, or information (in *bits* when using the base 2 logarithm, or in *nats* when using the natural logarithm) contained in it.

Given a random variable X, with support χ. The entropy $H(X)$ of X is defined as

$$H(X) = \sum_{x\in\chi} p(x) \log \frac{1}{p(x)} = -\sum_{x\in\chi} p(x) \log p(x) \tag{2.28}$$

A random variable with an entropy of zero can be seen as a deterministic function, its uncertainty being zero. Instead, a random variable with high entropy is a variable whose outcomes are more unpredictable. The distribution with highest entropy for a certain support (either discrete or continuous) is the uniform distribution. Notice that when $p_X(x) = 0$, we have that $p(x) \log p(x) = 0$, due to the fact that $\lim_{x\to 0} x \log x = 0$. In this thesis we will only need to use the entropy of discrete random variables, so we will ignore the definitions for continuous random variables, although they are analogous to their discrete counterparts.

2.2.2 Conditional Entropy

We are interested in evaluating the uncertainty of a probability distribution *conditioned on some event* e. By using the definition of entropy to the conditional probability $p(X|E)$, we obtain

$$H(X|E) = \sum_{x\in\chi} p(x|E)\log\frac{1}{p(x|E)} \tag{2.29}$$

We can define the conditional entropy between two distributions X and Y as

$$H(X|Y) = \sum_{y\in\mathcal{Y}} p(y)H(X|Y=y) \tag{2.30}$$

$$= -\sum_{y\in\mathcal{Y}} p(y)\sum_{x\in\chi} p(x|y)\cdot\log p(x|y) \tag{2.31}$$

$$= -\sum_{y\in\mathcal{Y},x\in\chi} p(x,y)\log p(x|y) \tag{2.32}$$

With this definition, it is clear that we can interpret the conditional entropy $H(X|Y)$ is **not** the entropy of a probability distribution. Instead, it is an expectation: the average entropy about X when given Y

Notice that the conditional entropy $H(X|Y) = 0$ if and only if the value of X is completely determined by Y, which means X is a deterministic function of Y. On the opposite, if X and Y are independent, then $H(X|Y) = H(X)$, as knowing Y does not give any information about X. In general, we have that $H(X|Y) \leq H(X)$

2.2.3 Mutual Information

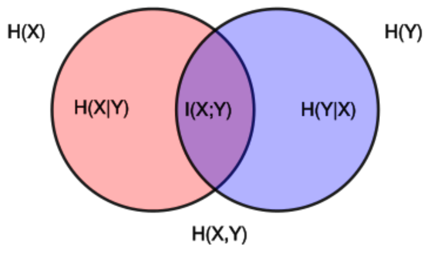


Fig. 2.1: Venn Diagram illustrating the relation between entropies, conditional entropies and mutual information of two random variables X and Y. Image taken from https://commons.wikimedia.org/

Mutual information measures the information that two random variables share. There are many equivalent ways to express mutual information, and for our discussion it is particularly useful to write it in the following equivalent ways:

$$I(X,Y) = H(X) - H(X|Y) \tag{2.33}$$

$$I(X,Y) = H(Y) - H(Y|X) \tag{2.34}$$

We can see from this definition that mutual information can be thought of a measure of how much knowing Y reduces uncertainty about X (or vice versa, as the definition is symmetric). Thus, high mutual information indicates a large reduction in uncertainty (or a high amount of shared information); low mutual information indicates a small reduction (or a small amount of shared information). Intuitively, the mutual information between two random variables is zero when the two are independent, as they do not give any information about each other. On the opposite side, if X is a deterministic function of Y, then the conditional entropy $H(X|Y) = 0$ and the mutual information is equal to the entropy of X, i.e $I(X,Y) = H(X)$, and the amount of information shared is as much as the information contained in X. The opposite holds when inverting the roles of X and Y.

Also, it will be useful later in the thesis to use the following inequalities

$$I(X,Y) \leq H(X) \tag{2.35}$$

$$I(X,Y) \leq H(Y) \tag{2.36}$$

which can be trivially obtained by the definition of mutual information due to the fact that the conditional entropy $H(X|Y)$ is non negative. Finally, it is important to note that the mutual information itself is always non negative $I(X,Y) \geq 0$, although for particular values of $x \in \chi, y \in Y$, it can take negative values.

3 Related Work

In this chapter, we provide a short overview of recent developments in Hierarchical Reinforcement Learning, explaining different approaches to learn low-level policies. We also introduce recent work Intrinsic Motivation, focusing particularly on methods used to learn low-level policies without extrinsic rewards.

3.1 Hierarchical Reinforcement Learning

The main motivation behind our work lies in Hierarchical Reinforcement Learning (HRL). As humans, we are familiar in solving problems by separating them into multiple easier sub-problems. For example, when we are cooking we learn some basic skills such as chopping, stirring, boiling, and we are naturally able to combine them with different ingredients to create a huge variety of recipes. Similarly, when we play a sport, we learn a few basic movement (and tactics), and only later we learn how to combine them most efficiently to win the game. Even more, Every once in a while we can come up with new moves or tactics, and are able to efficiently change our behavior to achieve our goal by including them.

Standard algorithms in Reinforcement Learning act in the original state space by using actions. For very difficult problems, where trajectories are composed of very long sequences of actions (thousands or tens of thousands), it becomes harder and harder to distribute credits for rewards backwards through time. Furthermore, even though some desired behavior may need tens of thousands of time steps to be achieved, it might correspond to just a few abstract actions (grab, chop, stir, etc, for our cooking example). Intuitively, introducing a way to learn and reuse abstract behaviors would be greatly benefit

Hierarchical Reinforcement Learning deals with learning a hierarchy of policies, which solve tasks at different abstraction levels. As we already mentioned, one benefit is an easier long-term credit assignment, which can lead to faster learning (Nachum, S. S. Gu, et al., 2018), (Levy, Jr., et al., 2018). Learning behaviors at different levels can also allow for better transfer: low-level behaviors can be reused explicitly to learn different high level behaviors more easily (Haarnoja, Hartikainen, et al., 2018) (Peng et al., 2019). Finally, exploration can be done in a temporally extended way, and even a more semantically meaningful way, as exploration can be done with sub-policies rather that primitive actions. Recently Nachum, H. Tang, et al. (2019) investigated the reasons that make HRL successful in environments where standard RL algorithms do not work well, and results show that exploration strategies are the main factor determining the success of current HRL methods.

3.1.1 Goal-conditioned methods

Most of the current Hierarchical Reinforcement Learning approaches use a hierarchy of policies and/or value functions composed of two levels. The high-level (or master) policy does not output standard actions, but an abstract action $g(s)$ which is used to specify the behavior that the low-level policy has to attain. It is the low level policy $\pi(a|s,g)$ which outputs standard actions based on the high-level behavior imposed from the master policy. Typically, low-level policies run for a fixed number of steps, after which the master policy takes a new high level action. Alternatively, low level policies can also have a termination condition, which causes the policy to stop when some low-level goal is achieved, regardless of the number of steps taken. This is the case of the Option-Critic architecture described in the next section.

The simplest way to communicate goals in the hierarchy is by using states as goals. By defining a distance function between pairs of states, the low-level policy can be trained to minimize it, while the high level policy can use environment rewards as signals to learn how to choose goals. This approach is used in HIRO (Nachum, S. S. Gu, et al., 2018), which solves complex tasks for simulated robots previously unsolved by HRL methods by using off-policy learning. (Levy, Konidaris, et al., 2017) takes a similar approach to HIRO but outperforms it thanks to the use of hindsight (Andrychowicz et al., 2017), allowing the learning of multiple levels of policies in parallel. Another approach that uses states as goals is h-DQN (Kulkarni et al., 2016), where a hierarchy over value functions is used instead of a hierarchy of policies.

Goals can also be expressed by abstract embeddings, which can be learned end-to-end. In Feudal Networks (Vezhnevets et al., 2017), the goal is set as the the direction of the difference vector between initial and final state. The high-level policy learns to select goals in a latent goal space, and the worker learns to achieve that direction by using intrinsic rewards. This type of approach is appealing, but is often outperformed by methods that use raw states as goal Nachum, S. S. Gu, et al. (2018) and Nachum, S. Gu, et al. (2018). Furthermore, the resulting goal representation may be under-defined, as not all possible sub-tasks may be expressible as goals in the space. Instead, when using raw states as goals no information is lost, and the low-level policy can receive meaningful (intrinsic) rewards regardless of the behavior of the high-level policy. The main drawback of this approach is that it is difficult to scale to higher dimensions (such as images). To enable the use of learned goal representations without restricting the optimality of the resulting hierarchical policy, (Nachum, S. Gu, et al., 2018) develops a theory for near-optimal goal representations, and manages to outperform existing HRL approaches in high-dimensional continuous-control tasks.

Finally, goals can be expressed as embeddings which are not directly related to the states reached by the low-level policy, and low-level policies can be learned without needing to define a distance function between states. Option approaches (Sutton, Precup, et al., 1999), (Bacon et al., 2016) are an example, and are described next.

3.1.2 The Option-Critic architecture and termination functions

We want to summarize here the Option-Critic architecture (Bacon et al., 2016), which, in the context of the options framework (Sutton, Precup, et al., 1999), is able to learn low-level policies, high-level policy (over options) and termination functions all together in an end-to-end manner. The goal of this thesis (explained in chapter 4) is specifically in learning termination functions together with internal policies (while we will ignore learning the policy over options) so that they can be used by a hierarchical model, which makes the Option Critic particularly relevant as option approaches are the only ones that can directly incorporate termination functions, and the Option-Critic Architecture is the basis for many of these approaches.

The options framework (Sutton, Precup, et al., 1999) formalizes the idea of temporally extended actions, which are called options (or skills in intrinsic motivation literature). A Markovian option $\omega \in \Omega$ is a triple $(I_\omega, \pi_\omega, \beta_\omega)$ where I_ω is the set of possible initial states of an option, $\pi_\omega = \pi(a|s, \omega)$, the low-level (intra-option) policy the policy encoded by the option, and $\beta_\omega(s)$ is a state-dependent termination function, which defines a bernoulli distribution with mean parameter $\beta_\omega(s)$ for each state. As soon as a state is reached (excluding the first one), a sample is drawn from the bernoulli distribution. When a 1 is sampled, the current option terminates and a new option has to be chosen by the high level policy.

In the Option-Critic, we have a discrete number of options $\omega \in \Omega = 1..N$. The initiation set I_ω is assumed to be the entire state space for each option, while the policies and termination functions share parameters for all options through a neural network. The high level policy $p(\omega)$ (Multi level hierarchies of options have been introduced in (Riemer et al., 2018)) is not represented explicitly but implicitly derived from an estimate of value function over options

$q_\Omega(s,\omega)$ through an ϵ-greedy strategy. The value functions for the Option-Critic are useful to understand option switching, and can be written in the following way:

$$V(s) = \sum_{\omega \in \Omega} p(\omega) q_\Omega(s,\omega) \tag{3.1}$$

$$q_\Omega(s,\omega) = \sum_a \pi_\omega(a|s) Q_U(s,\omega,a) \tag{3.2}$$

$$q_U(s,\omega,a) = r(s,a) + \gamma \sum_{s'} p(s'|s,a) U(\omega,s') \tag{3.3}$$

$$U(\omega,s') = \beta_\omega(s')V(s') + (1-\beta_\omega(s'))Q_\Omega(s',\omega) \tag{3.4}$$

The first interesting fact is the use of the value function $U(\omega,s')$. This represents the return of an option upon reaching a state, before deciding whether to terminate the option. This is opposite to $q_\Omega(s,\omega)$, which represents the return of an option while executing option ω. Thus, $U(\omega,s')$ includes the termination decision: we get a return of $v(s')$ if we choose to terminate the option, with probability $\beta_\omega(s')$, or we get a return of $q_\Omega(s',\omega)$ if we choose to continue with the current option, with probability $(1-\beta_\omega(s'))$.

Notice also that these value functions correspond to different levels of abstractions. In particular, $V(s)$ and $q_\Omega(s,\omega)$ can be seen as high level state-value function and action-value (called now option-value) function respectively. $q_U(s,\omega,a)$ is the action-value function for the low-level, while $q_\Omega(s,\omega)$ can also be seen as the state-value function for the low level.

It is interesting to analyze the form of the (policy) gradient of the termination function $\beta_\omega(s)$ (with parameters θ) obtained in (Bacon et al., 2016)

$$\frac{\partial U(\omega_0,s_1)}{\partial \theta} = \sum_{\omega,s'} \mu_\Omega(s',\omega|s_1,\omega_0) \frac{\partial \beta_\omega(s')}{\partial \theta} \left(-A_\Omega(s',\omega)\right) \tag{3.5}$$

where $\mu_\Omega(s',\omega|s_1,\omega_0)$ is the discounted visitation probability of state-option pair (s',ω) after visiting state s_1 using option w_0 (note that s_1 is used instead of s_0 as it indicates the state visited after the first transition). For details of the definition of the MDP with states and options, we refer to appendix A in the Option Critic paper (Bacon et al., 2016). The resulting update is similar to a policy gradient update, but it is shifted by one step and does not contain a log term. Also, it is interesting to note that the gradient is multiplied by the negative advantage function, which is defined as $A_\Omega(s,\omega) = Q(s,\omega) - V(s)$. This means that the higher the return of a certain option ω is with respect to the average return over all options, the less it is encouraged to terminate (which means it is encouraged to continue), and vice versa.

One of the main problems the authors report for the Option-Critic is that options tend to converge to trivial behaviors. For example, it can happen that learned options always terminate after a single step, thus making the hierarchy useless as the learned behavior is encoded entirely in the high level policy. In other cases it can happen instead that one learned options is always active and solves the task alone. To tackle this problem and try to learn meaningful low level policies while dividing the work of the high and low levels, in the recent years several approaches have been proposed. To prevent options from switching too frequently, a deliberation cost scalar can be introduced (Harb et al., 2018) as cost for switching from any option to another option, and is included in the termination gradient calculation. Results show indeed options tend to not shrink over time. In the Termination Critic (Harutyunyan et al., 2019), the termination function does not maximize returns, but instead minimizes $H(s_f|\Omega)$, the conditional entropy between final states s_f of an option. This has the effect of making options terminate where their outcome is predictable, and in practice has the effect of learning options where the termination function is active only in small parts of the state space. The method is however only applicable to discrete state MDPs. The Safe Option-Critic (Jain et al., 2018) introduces the idea of controllability of states as the variance of the temporal difference error, and uses it to constrain the updates of the low-level policy. This approach cannot be directly used for as unsupervised pretraining of options, and while it helps in cases where dangerous states are present, it does not clearly help in learning temporally well behaved options. Finally, in (Smith et al., 2018) options are learned by using probabilistic inference, and exhibit temporal coherence and separation over the state space. Furthermore, the method allows to simultaneously improve all of the options available to an agent, and thus can be employed in an off-policy manner, without observing option labels. Although the method manages to learn meaningful termination conditions, as all methods described until this point, the method requires extrinsic rewards.

Learning a hierarchy of policies is not necessarily easier than learning a single policy to solve a task (Smith et al., 2018). However, when the task is not specified in advance, it can be much easier to learn low-level policies and to learn a high-level policy only when the task is given. In the next section, we provide an overview of how it can help learning meaningful options.

3.2 Intrinsically Motivated Reinforcement Learning

We talked about the advantages of learning a hierarchy of behaviors, but have not talked precisely about the approach that can be used to learn such hierarchy. For example, given a task, would it be better to learn low-level behaviors based on what is most useful for the task itself, or do we just learn different useful behaviors we think are useful, to later combine them to solve the task? As humans, we might use either strategy in different contexts. When learning a new sport, it might be more natural to follow the first approach, where we try to achieve the goal with simple strategies at first, and gradually improve our skills. In other cases, such as when playing a new video game, we might first want to get accustomed to the basic commands and learn what kinds of interactions are possible, before actually trying to accomplish our task.

The example above refers to the fact that we can either train our hierarchical agent end-to-end based on rewards, or we can first pretrain a set of skills without external rewards, so that we can later use these skills to solve the task. In the first case, extrinsic rewards, the ones coming from the environment, are used to train all levels of the hierarchy. In the second case, we do not have a specific goal from the environment, but we ourselves set the goal of learning how to interact with the environment in different and meaningful ways, so that learned behaviors can be useful when we actually try to solve a complex task. The reward that we set for ourselves can be called **intrinsic reward**.

In general Intrinsic Motivation is composed by all those approaches where agents learn behaviors for their inherent desire rather than optimizing a reward assigned by the environment. Intrinsic Motivation is an active process unlike supervised and unsupervised learning, which can be called passive, as these do not have form of interaction with an environment. However, unlike Reinforcement Learning, it does not include an external feedback. Reinforcement Learning and Intrinsic Motivation are however not orthogonal approaches: combining them often allows solving more complex problems. Aubret et al. (2019) identify four main challenges faced by Deep Reinforcement Learning, where Intrinsic Motivation can greatly help.

The first is exploration. Current Reinforcement Learning algorithms work well in when the rewards are dense, so that the agent always gets feedback about its actions. When working with environments where rewards are sparse however there is no positive reward until some specific state is reached, which makes solving some environments (such as Montezuma's Revenge) extremely hard for current algorithms. One possible solution is manually creating an intermediate dense reward, which can however be difficult to specify for some problems, and can also lead to undesired behaviors. The three main types of approaches for exploration in current literature are are prediction error (Burda, Harrison Edwards, et al., 2018) (Burda, Harri Edwards, et al., 2018), (Pathak et al., 2017), state novelty (Savinov et al., 2018) and information gain (Houthooft et al., 2016) (Xu et al., 2020). The other challenge where intrinsic motivation can help temporal abstraction of actions. When acting in an environment for a large number of steps, it can be useful to not only rely on primitive actions defined by the environment, but to also introduce high level actions which represent meaningful and more abstract behaviors. This can make the credit assignment problem much easier, so that when having long sequences of actions, it becomes easier to identify which ones were responsible for getting high reward. This work particularly well when the long sequences of actions actually result of only a few high level actions. Intrinsic motivation can help building these abstract actions.

The other two challenges mentioned are building a curriculum, in order to have the agent learns behaviors that are increasingly complex, while being related to each other, and building a good state representation. Since these two are less relevant for this thesis, for a more detailed overview on intrinsic motivation, we refer to Aubret et al. (2019). The topic we will focus on this thesis is temporal abstraction of actions, and in particular without using any extrinsic reward.

3.2.1 Learning Skills without extrinsic rewards

Learning skills without extrinsic rewards can lead to a number of benefits. First, for some tasks, it is very difficult to specify a reward that allows efficient training of RL agents. Second, extrinsic rewards typically allow to learn to solve only one task at a time. Instead, using intrinsic rewards, we can learn to solve multiple tasks at the same time without needing to specify a reward.

Furthermore, a desirable outcome of learning is that knowledge obtained should be transferable. In Reinforcement Learning this can mean that learned behaviors should be useful for solving not only one specific task, but also several unknown similar ones. Also, it would be desirable to learn behaviors that solve the same task even when the environment changes. One can think of a similarity with Deep learning, where low level features of neural networks often provide very good initialization for tasks different from the one they were trained on, and where by just retraining high level features one can learn very efficiently with few data points. In the same way, it would be desirable to be able to learn useful policies that with just minimal effort can be adapted to different tasks or environments.

We now describe several approaches to learning low-level policies (also referred as options or skills) proposed in the past few years which do not rely on extrinsic rewards, and thus lead to more general purpose learned skills. The methods relying on mutual information will be presented in section 3.2.2, as they are directly related to the method used in this thesis.

One approach to learning options without extrinsic rewards it by using Imitation Learning. Deep Discovery of Options (DDO, (Fox et al., 2017)) uses a set of demonstration to learn options in a recursive way, allowing to discover options at an arbitrary number of levels of a hierarchy. Its extension, Deep Discovery of Continuous Options (DDCO, (Krishnan et al., 2017)), introduces a hybrid categorical-continuous distribution model to allow policies to use both discrete options and continuous actions.

Instead of requiring demonstrations, (Marios C Machado et al., 2017), uses the graph Laplacian and the concept of proto-value functions to learn representations of the state space, along with (eigen)options that traverse it. The authors show a way to use the approach with sampling, by collecting trajectories to approximate the state graph. The work has been extended in (Marlos C Machado et al., 2017) to settings with stochastic transitions and where handcrafted features are not available, discovering options while learning non-linear state representations (which makes the approach viable when using raw pixels as inputs). These methods can however learn a termination condition only by adding a special termination action to the action set, thus are not applicable to continuous actions environments.

Jinnai, Park, Abel, et al. (2019) introduces the concept of covering options, which explore the state space by minimizing the expected cover time, discovering less explored regions of the state space and learning options to reach them. As this method is limited to small discrete MDPs, the subsequent Deep Covering Options approach (Jinnai, Park, Marlos C Machado, et al., 2020) manages to work on large discrete and continuous state spaces, showing success both when used as pre-training and in an online learning setting. Notably, this approach is able to learn initiation sets I_ω and termination functions β_ω along with option policies π_ω.

Mankowitz et al. (2016) introduces ASAP, a policy gradient method for learning options together with Skill Partitions (a particular instance of initition sets I_ω, which is often ignored in skill discovery). Although the method shows promising results in simple environments, it uses linear function approximations and cannot learn termination functions.

3.2.2 Information-theoretic methods

In this section, we will describe several approaches that allow the discovery diverse skills without using extrinsic rewards, and without relying on a goal space. These methods have in common that they aim to maximize the mutual information between options and (parts of) trajectories. Using the notation of (Aubret et al., 2019) for generality, we indicate the choice of a part of a trajectory τ through the use of a function $f(\tau)$, which selects only the relevant part of a trajectory (for instance, the first and last state). The intrinsic reward for a particular option ω and a corresponding generated trajectory τ can be written as

$$r^I(\omega,\tau) = I(\omega, f(\tau)). \tag{3.6}$$

By using this kind of reward in a reinforcement learning algorithm, we are trying to maximize the mutual information between trajectories and options. Intuitively, this means that we want to maximize the amount of information that options give us about the trajectories they produce, and at the same time the amount of information that trajectories give about which options. Notice that for now we are limiting the discussion to cases where there is only one reward, obtained when a full trajectory is collected. Also, to reduce clutter, from now on we will drop the function selection f and already regard τ as the selected part of a trajectory we are interested in (for instance, a particular interesting choice is the final state s_f). To deeper understand the meaning of this objective, we can use the equalities in 2.34

$$\begin{aligned} I(\Omega,\tau) &= H(\tau) - H(\tau|\Omega) \quad (3.7) \\ &= H(\Omega) - H(\Omega|\tau) \quad (3.8) \end{aligned}$$

By looking at the first equation, we can see that it has a very intuitive interpretation. $H(\tau|\Omega)$ is the conditional entropy of trajectories given an option, which is zero when knowing the option that generated a trajectory completely determines the trajectory itself. Since we are maximizing the mutual information, we want $H(\tau|\Omega)$ to be minimized. This intuitively means that we want to be as certain as possible about the trajectories generated by option ω. Instead, the term $H(\tau)$ is the entropy of the distribution of trajectories, where the option variable is averaged out. Maximizing this term means that our objective is maximized when as many different trajectories as possible are generated, regardless of the option used. Together, mutual information is maximized by a model which can generate as many trajectories as possible, while all trajectories are as "predictable" as possible when knowing the option that generated them.

Optimizing the mutual information by maximizing the two terms of equation 3.7 requires us to know the forward model $p(\tau|\omega)$. As this requires knowledge of the environment other than the options, an approximate model can be trained to estimate $p(\tau|\omega)$. However, this is typically very difficult to do, even when we are considering a easier subset of the trajectory to estimate such as the final state s_f.

Instead, recent literature (Gregor et al., 2016; Achiam et al., 2018; Eysenbach et al., 2018; Pong et al., 2019; Co-Reyes et al., 2018; Warde-Farley et al., 2018) has mostly focused on the other side of the mutual information, in equation 3.8, as estimating the inverse model $p(\omega|\tau)$ can be done more efficiently for the discrete option case by framing a classification problem over options. By looking at the two terms of 3.8, we can also find an intuitive interpretation of maximizing mutual information. $H(\omega|\tau)$ is the conditional probability of options given trajectory, which is minimized when given any trajectory, we can completely determine which options generated it. $H(\omega)$ is the entropy of the distribution over options, which is the entropy of the high-level policy that selects options. Since the main interest in recent work has focused on learning low-level policies, the optimization problem can be made easier by choosing a uniform policy over options, and only minimizing the conditional entropy $H(\omega|\tau)$.

Notice that $I(\Omega,\tau) \leq H(\tau)$ and $H(\Omega,\tau) \leq H(\Omega)$. This means that when having a discrete number of options, the MI cannot be greater than the log of the number of options. This means that there cannot be more than N groups of trajectories that can be distinguished, thus partitioning trajectories in at most N clusters

In order to understand how to maximize mutual information in Reinforcement Learning, we need to understand how to explicitly write equations 3.7 and 3.8

$$I(\Omega,\tau) = -\sum_{\tau} p(\tau)\log p(\tau) + \sum_{\omega\in\Omega,\tau} p(\tau|\omega)p(\omega)\log p(\tau|\omega) \tag{3.9}$$

$$= -\sum_{\omega\in\Omega} p(\omega)\log p(\omega) + \sum_{\omega\in\Omega,\tau} p(\omega|\tau)p(\tau)\log p(\omega|\tau) \tag{3.10}$$

From this formulation we can see that there is a particularly convenient way to estimate mutual information through sampling. We can sample an option $\omega \sim \Omega$, then sample trajectory τ according to the policy $\pi(a|s,\omega)$, so that we are in fact obtaining a sample according to $p(\tau,\omega) = p(\tau|\omega)p(\omega)$, so that we just need to evaluate $p(\omega|\tau)$ (assuming $p(\omega)$ is uniform). By collecting enough samples, we can obtain an approximation of the mutual information without knowing the transition dynamics of the environment. Also, we can see that the contribution to the mutual information for a particular sample (ω,τ) is $\log p(\omega|\tau) - \log p(\omega)$. This can be used directly as a reward for any reinforcement learning algorithm that works with sampled trajectories in order to maximize the mutual information.

Variational approximation

Since knowing the true posterior probability $p(\omega|\tau)$ also requires full knowledge of the transition dynamics of the environment, it is typically necessary to use an approximate model (or discriminator) $q_\phi(\omega|\tau)$. It can be shown (Gregor et al., 2016) (Mohamed and Rezende, 2015) that a variational lower bound on the mutual information is obtained $I^{VB}(\Omega,\tau) \leq I(\Omega,\tau)$ when using an approximate model. The lower bound is

$$I^{VB}(\Omega,\tau) = -\sum_{\omega} p(\omega)\log p(\omega) + \sum_{\omega\in\Omega,\tau} p(\omega|\tau)p(\tau)\log q_\phi(\omega|\tau) \tag{3.11}$$

From this equation, we can see that taking the gradient with respect to ϕ can be done in a simple way: by taking a sample of an option $\omega \sim \Omega$, and a sample trajectory τ according to the policy π, we are obtaining a sample according to $p(\tau,\omega) = p(\tau|\omega)p(\omega)$, and the sample estimate of the gradient is just $\nabla_\phi \log q_\phi(\omega|\tau)$. This means that we can just take samples of options and trajectories, and maximize the log likelihood of the discriminator q_ϕ according to the samples. Notice that this is exactly the same as minimizing the cross entropy over the sampled data, so the problem can be treated, for a particular batch, as a supervised classification problem of options given trajectories: each option corresponds to a separate class, and the discriminator learns to predict which option produced trajectory τ. We can now describe a first simple algorithm for learning options by maximizing mutual information through sampling, as done in Achiam et al. (2018)

Algorithm 1: Variational Option Discovery

```
Generate initial policy $\pi_\theta$, discriminator $q_\phi$
for $k = 0, 1, 2, \dots K$ do
    Sample option-trajectory pairs $\mathcal{D} = \{(\omega^n, \tau^n)\}_{n=1,\dots,N}$, by first sampling an option
        $\omega \sim \Omega$ and then rolling out a trajectory $\tau$ in the environment,
    Calculate intrinsic rewards $r_I^n = -\log p(\omega^n) + \log q_\phi(\omega^n|\tau^n)$
    Update policy with any reinforcement learning algorithm to maximize
        the rewards $\{r_I^n\}_{n=1,\dots,N}$ using batch $\mathcal{D}$
    Update decoder by supervised learning: maximize $\mathbb{E}\left[\log q_\phi(\omega|\tau)\right]$, using batch $\mathcal{D}$
    Reset environment to the initial conditions end for
```

3.2.3 Related approaches

We now want to describe the main methods that maximize mutual information in order to learn a diverse set of predictable skills. Notice that all the methods have in common the fact that all skills learned are always of fixed length. The first set of methods described learn a discrete number of options, and maximize mutual information as described in the previous section, with each approach using a different part of trajectories τ to learn skills.

Variational Intrinsic Control (Gregor et al., 2016) is the paper that applies the idea of empowerment (Klyubin et al., 2005) to options most directly, using only the initial state s_0 and the final state (after T steps) s_T of a trajectory for the intrinsic reward, as $q_\phi(\omega|\tau) = q_\phi(\omega|s_0, s_T)$. The authors focus on the specific case of grid-worlds, where they manage to learn simple behaviors in simple grid-worlds and only use linear function approximation, although our implementation manages to work also with neural networks, noting that the size of hidden layers must be pretty high (i.e. 300+) even for very simple grid world environments. The paper also discusses how to train the high level policy $p(\omega)$ through policy gradients along with the conditional model $q_\phi(\omega|s)$ **DIAYN** (Eysenbach et al., 2018) defines $\log q_\phi(\omega|\tau) = \sum_{t=0}^{T} \log q_\phi(\omega|s_t)$, thus using every single traversed state for calculating the trajectory reward. This is particularly efficient as the reward is not sparse anymore, and can be collected at each time step, allowing the direct use of TD methods. The optimization of the intrinsic reward is in fact done by using a soft actor critic (SAC) (Haarnoja, Zhou, et al., 2018) method, which is off-policy. These two factors allow DIAYN to find meaningful policies even for high dimensional continuous control tasks. In **VALOR** (Achiam et al., 2018), the representation for a trajectory τ is chosen to be a bidirectional LSTM encoding of states visited in τ, and a variety of skills can be learned in different environments, similarly to DIAYN. Importantly, the paper successfully uses a curriculum approach to learn an increasing number of options, leading to faster and more stable convergence. Differently from these approaches, **SNN4HRL** (Florensa et al., 2017) does not learn an inverse model to estimate the mutual information directly, but builds a discretization of the state space and keep track of visitation counts $m_c(\omega)$ for each option ω, in order to estimate $p(\omega|c)$ for each discrete state bin c. Also, the approach uses stochastic neural networks (R. M. Neal, 1990; R. Neal, 1992; C. Tang and Salakhutdinov, 2013) to efficiently learn a high number of skills without impacting sample efficiency.

Continous skills

We would like now to describe methods that encode skills with continuous representations, which are typically interpreted as goal embeddings, so that the mutual information is $I(g,s) = H(g) - H(g|s)$, where g is a continuous goal (or skill) representation. Working with continuous skills poses two main technical difficulties. First, the entropy $H(g|s)$ cannot be minimized by supervised classification as done with discrete skills. Second, it is not possible to easily obtain a uniform distribution over goals $P(G)$ as this requires knowledge of the distribution of valid states. In **DISCERN** (Warde-Farley et al., 2018) the authors introduce two methods to allow training of the model $q_\phi(g|s)$. The first method estimates $q_\phi(g|s)$ by using contrastive sampling with K random decoy goals sampled from a buffer of previously visited states, while the second one uses the positive part of the cosine similarity between goal embeddings and the achieved final state. Finally, to have a uniform distribution over goals, goals are chosen from a recent experience buffer. **Skew-fit** (Pong et al., 2019) also trains goal conditioned policies by using the same variational objective for goal conditioned setting. Differently from other methods, Skew-fit does not ignore the term $H(g)$ but directly learns a generative model $P(g)$ for generating goals that are valid and reachable, gradually increasing the entropy $H(g)$. In (Co-Reyes et al., 2018) the authors propose **SeCTAR**, which encodes trajectories into a latent space and decodes them in the same way as an auto-encoder, while also learning a conditioned policy to generate the decoded trajectories. By using the decoder as a forward option model, SeCTAR can perform planning in a closed-loop way. Another method for planning with hierarchies is introduced in (Sharma et al., 2019), which uses the forward side of the mutual information 2.33 and where, like in DIAYN, the reward is calculated at each time step. The main disadvantage of the approach is that by using model predictive control (MPC) (Garcia et al., 1989) to plan and they need access to the real transition dynamics model $p(s'|s,a)$ in order to plan.

As mentioned at the beginning of this section, all the methods described have in common that although they can learn set of different skills by maximizing some form of mutual information, they cannot learn a termination function. The focus of the next chapter will be introducing a way to learn termination function that is general and can be applied to most of the approaches discussed.

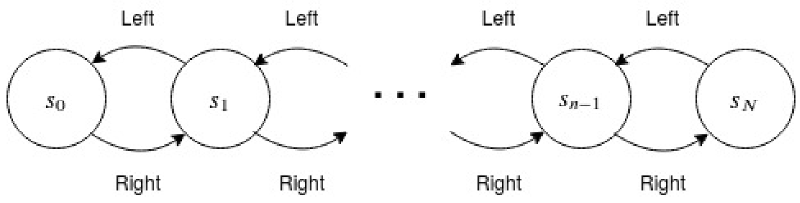
4 Temporally Extended Variational Option Discovery

In the previous chapter, we have seen how learning options (or skills, or low-level policies) without extrinsic rewards can be beneficial to learn behaviors that are more general and applicable for different tasks and environments. In this thesis, we focus on variational approaches (described in section 3.2.2), which are capable of learning options that maximize the mutual information between options and trajectories $I(\omega, \tau)$. As these approaches only allow learning options of fixed length, in this chapter we want to extend them by removing the constraint through the use of a termination condition, which allows options to terminate based on the state they reach rather than a fixed time step. This allows to remove the need of prior information and possibly achieve a more diverse set of more general options.

In this chapter, we will first consider the effects of a termination condition on an regular MDP, with just one policy (and thus one option), and later (section 4.6) discuss how to learn multiple options at the same time. As the only way options are related to each other is the intrinsic reward, we can restrict most of our initial discussion to a single option.

4.1 Limitations of fixed length options

We want to start by discussing how fixing the length of options can limit the usefulness of the learned behaviors, and how having a termination condition can lead to more useful behaviors. We first give an example of this in a simple gridworld.



Consider the case of a line environment (chain MPD), where we have states $S_1, S_2, ..S_N$, and each state is connected only to the previous state and the next state (with the exception of S_1 and S_N). Only two actions are available, *left* and *right*. *left* moves from s_n to s_{n-1}, while *right* from s_n to s_{n+1}. Let's imagine the case where $N = 100$: which values of fixed number of steps T would be the best? If we choose $T < 100$, no option will ever be able to reach states $N > T$, which is not desirable. If we choose $T \geq 100$ instead, all states can be reached. However, as options are always forced to last T steps, we have that for states s_n with $n < T$ the option has necessarily to go back and forward until time T is reached, even if the target states have been reached already. Although in this case options correctly reach their target states, the additional time spent taking unnecessary actions can be detrimental when options are used by a high-level policy in order to act in the environment. If the high-level policy has no way of terminating options before their fixed time (which is the case for most approaches that are not related to the options framework (Sutton, Precup, et al., 1999)), then the total number of steps needed to solve the task might be much higher that the optimal number of steps needed to solve it. Furthermore, we find that this behavior is not typically obtained in practice (see chapter 5). What happens instead is that options learn to use the left action with a certain probability so that on average at time step T the agent is correctly located in one of its target states. This means that the learning process lead to an option that achieves its goal, but in a sub-optimal way, due to the mismatch between the fixed length T and the actual number of steps needed to reach the goal.

In general, if we want options that are able to reach all parts of the state space, then we need T at least as big as the number of steps needed to reach the furthest state from the initial state. In this case, a termination condition is clearly beneficial. Instead, if we want to use options mainly as an initialization for a high-level policy, smaller T values can be also acceptable, as the task of actually reaching states is delegated to the high-level policy. This however limits the applicability of the options, which do not correspond necessarily to semantic behaviors; there is not necessarily a separation of tasks between high and low levels, and the advantages over a flat policy are limited to temporal dilation. If we had behaviors that are more general (such as reaching walls, walking in a particular direction), the high-level policy could work with more semantically meaningful behaviors, which would allow for easier learning and increased interpretability. Interpretability is particularly interesting and hard to achieve (Bacon et al., 2016), (Harutyunyan et al., 2019), (Smith et al., 2018), which makes the problem of finding dynamic length options even more interesting.

One possible workaround to the fixed length problem could be using different fixed length for different options. However, this still requires a lot of prior knowledge about the environment, as it is not immediate to identify how many options should have a certain length. Furthermore, it is not easy to define a fixed length for skills with continuous representation.

4.2 Using a Termination function

All the methods discussed in section 3.2.2 have one limitation in common: they limit the duration of options through a hyperparameter T, the fixed length of options. The only exception is VIC (Gregor et al., 2016), which defines a fixed termination probability $1-\gamma$ for each timestep. This implicitly defines a fixed probability distribution on the number of timesteps that options will last, and thus is not fundamentally different from choosing a fixed N, as the duration of the options cannot be learned.

The easiest way to add a termination condition for discrete actions MDPs is adding one additional action which allows to terminate the episode instead of executing any other action. Although this approach is very easy to implement, it is not applicable for environment with continuous actions. Furthermore, even when using an additional termination action, sample based Reinforcement Learning approaches usually only update the policy and value function about termination action only when it is sampled. On the contrary, by using a termination function, updates occur in parallel to actions, making the approach potentially more efficient. We will explore the differences between the termination function approach and the termination action approach in chapter 5.

In this work, we explore how to learn options of arbitrary length without relying on extrinsic rewards. To do so, we use a learnable state dependent termination function $\beta(s)$ (as in (Bacon et al., 2016)), which outputs a probability of terminating the current episode for the state s. The decision of whether to terminate the current episode happens every time a new state is visited, before taking action in it, by taking a sample from the Bernoulli distribution with mean $\beta(s)$. When we obtain a sample $b=1$, the current episode terminates in state s, while when a sample $b=0$ is obtained the episode continues. Notice that in this work, we consider the case where terminating the option is equivalent to terminating the episode, and the case where multiple options can be executed in the same episode is left for future work. To maintain the similarity with Bacon et al. (2016), we do not allow to terminate in the initial state (so one action is always taken, and episodes are at least of length one).

4.2.1 MDPs with termination functions

We now want to discuss how to model an MDP that includes a state dependent termination function $\beta(s)$. Although it would be possible to include $\beta(s)$ without any assumption about the nature of rewards, in this thesis we are interested specifically in the case where (intrinsic) rewards are collected at the end of an episode. To clarify what are the effects of introducing the termination function are, we would like to point out that all modification to the rewards

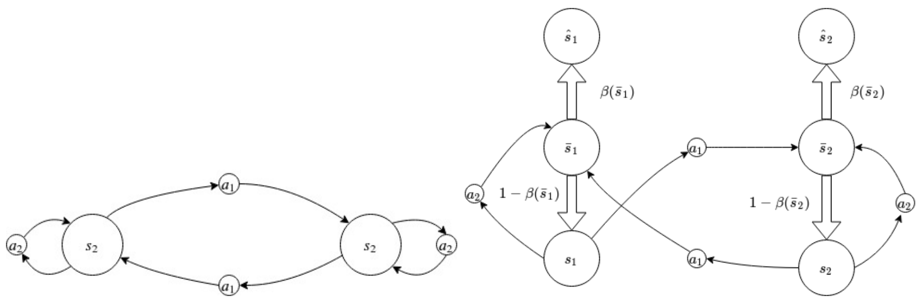


Fig. 4.1: Left: example of a simple MDP with two states and two actions. Right: modified MDP. For each state s, we have two more additional states $\bar{s}$ and $\hat{s}$, and actions take to states $\bar{s}'$ before deciding termination. The bold arrows indicate termination decisions.

of the original MDP are due to the use of the intrinsic rewards, and due to the termination function.

Another way to include episode termination through a termination function in an MDP (S, A, R, P, γ) is by adding a new sets of states, together with a separate set of termination actions. Thus, we will add termination actions $B = [0, 1]$ for each state, and two sets of states $\hat{S}$ and $\bar{S}$. $\hat{S}$ contains one state $\hat{s}$ for each original state $s \in S$, and it represents states where termination happened. Instead, $\bar{S}$ represents states where the termination still has to happen, and it also contains exactly one state $\bar{s}$ for each $s \in S$. The original states $s \in S$ retain their original meaning, as no termination happened, and an action can be taken in it. The termination decision happens through a termination function $\beta(\bar{s})$ in each state $\bar{s} \in \bar{S}$ *before* taking action in it. As in this thesis we choose to avoid termination in the starting state s_0, we can equivalently say that the termination decision happens every time a new state is encountered.

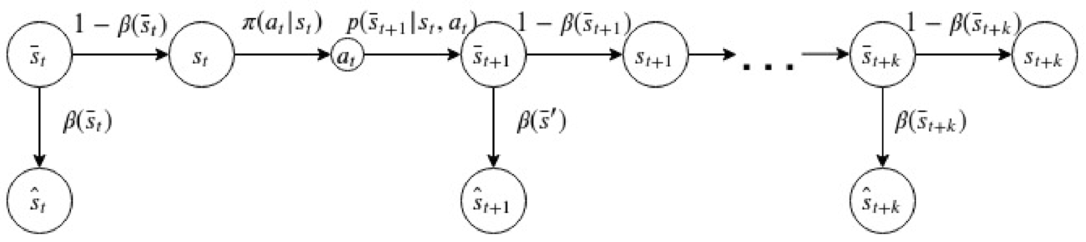
As a consequence of the introduction of the termination condition and the new states, we need to modify the transition probabilities P and the rewards R. Regarding the former, we have that transitions now do not lead to original states $s \in S$ anymore, but they lead to states **before** termination decision, so to $\bar{s} \in \bar{S}$. Thus, each original transition $P(s'|a, s)$ as becomes $P(\bar{s}'|a, s)$. Regarding rewards instead, by using the termination function together with intrinsic rewards, we have that the only non zero rewards are those received on states $\hat{s}$.

4.3 Transition process

As described in the previous section, recall that that s is a regular state, where we are sure we did not terminate and an action has to be taken. $\hat{s}$ is the state that represents that we decided we terminated in state $\bar{s}$. Lastly, $\bar{s}$ is the state before we decide whether termination will happen or not. We now want to state formally the expressions of the state transitions.

There are two main transition probabilities that we are interested on in order to perform the policy and termination function updates described in the next section. The first $P^k(\bar{s'}|\bar{s})$ (for $k \geq 0$) represents the probability of transitioning from state $\bar{s}$ to state $\bar{s}'$ in exactly k steps, so the transition from states before the termination decision. The second is $P^k(s'|\bar{s})$ (for $k \geq 0$), which represents the transition probability from state $\bar{s}$ before termination decision to state s' after termination decision. These two transition probabilities will be used for different derivations ($P^k(s'|\bar{s})$ for 7.2.1 and $P^k(\bar{s'}|\bar{s})$ for 7.2.3). To help understanding the transition process, it can also be useful to look at the following graphic representation of the transition process.

We can now compare the transition process from the option critic and termination critic, noting that they are just the same but from the point of view of s instead of $\bar{s}$.



$$\begin{tikzcd}
\bar{s}_t \arrow[r, "1-\beta(\bar{s}_t)"] \arrow[d, "\beta(\bar{s}_t)"] & s_t \arrow[r, "\pi(a_t|s_t)"] & a_t \arrow[r, "p(\bar{s}_{t+1}|s_t, a_t)"] & \bar{s}_{t+1} \arrow[r, "1-\beta(\bar{s}_{t+1})"] \arrow[d, "\beta(\bar{s}')"] & s_{t+1} \arrow[r] & \cdots \arrow[r] & \bar{s}_{t+k} \arrow[r, "1-\beta(\bar{s}_{t+k})"] \arrow[d, "\beta(\bar{s}_{t+k})"] & s_{t+k} \\
\hat{s}_t & & & \hat{s}_{t+1} & & & \hat{s}_{t+k} &
\end{tikzcd}$$

Note that in order to simplify the notation, we will use the one step transitions $p^{\pi}(\bar{s}_{t+1}|s_t) = \sum_a \pi(a|s_t)\,p(\bar{s}_{t+1}|s_t, a)$

Definition 1: Transition probabilities for pre-states $\bar{s}$

$$P^{(0)}(\bar{s}|\bar{s}_t) = I_{\bar{s}=\bar{s}_t} \tag{4.1}$$

$$P^{(1)}(\bar{s}_{t+1}|\bar{s}_t) = (1-\beta(\bar{s}_t))p^{\pi}(\bar{s}_{t+1}|s_t) \tag{4.2}$$

$$P^{(2)}(\bar{s}_{t+2}|\bar{s}_t) = \sum_{s_{t+1}}(1-\beta(\bar{s}_{t+1}))p^{\pi}(\bar{s}_{t+2}|s_{t+1})(1-\beta(\bar{s}_t))p^{\pi}(\bar{s}_{t+1}|s_t) \tag{4.3}$$

$$= \sum_{s_{t+1}}(1-\beta(\bar{s}_{t+1}))p^{\pi}(\bar{s}_{t+2}|s_{t+1})P^{(1)}(\bar{s}_{t+1}|\bar{s}_t) \tag{4.4}$$

$$= \sum_{s_{t+1}} P^{(1)}(\bar{s}_{t+2}|\bar{s}_{t+1})\,P^{(1)}(\bar{s}_{t+1}|\bar{s}_t) \tag{4.5}$$

$$P^{(3)}(\bar{s}_{t+3}|\bar{s}_t) = \sum_{s_{t+1}} P^{(2)}(\bar{s}_{t+3}|\bar{s}_{t+1})P^{(1)}(\bar{s}_{t+1}|\bar{s}_t) \tag{4.6}$$

$$\ldots \tag{4.7}$$

$$P^{(k)}(\bar{s}_{t+k}|\bar{s}_t) = \sum_{s_{t+1}} P^{(k-1)}(\bar{s}_{t+k}|\bar{s}_{t+1})P^{(1)}(\bar{s}_{t+1}|\bar{s}_t) \tag{4.8}$$

As we can see, the transition probability $P^{(k)}(\bar{s}_{t+k}|\bar{s}_t)$ can be easily expressed in a recursive way. We next write down the the transition probability $P^{(k)}(s_{t+k}|s_t)$ and note that the result is also recursive, and very similar to the one obtained before.

Definition 2: Transition probabilities of actual states s

$$P^{(0)}(s_t|s_t) = 1 \tag{4.9}$$
$$P^{(1)}(s_{t+1}|s_t) = (1-\beta(\bar{s}_{t+1}))p^{\pi}(s_{t+1}|s_t) \tag{4.10}$$
$$P^{(2)}(s_{t+2}|s_t) = (1-\beta(\bar{s}_{t+2}))\sum_{s_{t+1}} p^{\pi}(\bar{s}_{t+2}|s_{t+1})(1-\beta(\bar{s}_{t+1}))p^{\pi}(\bar{s}_{t+1}|s_t) \tag{4.11}$$
$$= \sum_{s_{t+1}}(1-\beta(\bar{s}_{t+1}))p^{\pi}(\bar{s}_{t+2}|s_{t+1})P^{(1)}(s_{t+1}|s_t) \tag{4.12}$$
$$= \sum_{s_{t+1}} P^{(1)}(s_{t+2}|s_{t+1})\,P^{(1)}(s_{t+1}|s_t) \tag{4.13}$$
$$P^{(3)}(s_{t+3}|s_t) = \sum_{s_{t+1}} P^{(2)}(s_{t+3}|s_{t+1})P^{(1)}(s_{t+1}|s_t) \tag{4.14}$$
$$\dots \tag{4.15}$$
$$P^{(k)}(s_{t+k}|s_t) = \sum_{s_{t+1}} P^{(k-1)}(s_{t+k}|s_{t+1})P^{(1)}(s_{t+1}|s_t) \tag{4.16}$$

These probabilities are the same as in Harutyunyan et al. (2019), appendix A, although we have made explicit that the transitions $P^{(k)}(\bar{s}_{t+k}|\bar{s}_t)$ are with respect to the states before deciding the termination. It is also interesting to see that both the transition probabilities have the same recursive formulation.

We now want to focus on transition probabilities to termination states $\hat{s}$. Again, there are two ways to calculate probabilities to termination states $\hat{s}$: either starting from a state before termination decision $\bar{s}$, or starting from states s after termination decision. Harutyunyan et al. (2019) uses the former, while Puterman (2014) uses a discounted version of the latter. Since the two approaches do not use an explicit notation to distinguish states before and after termination decision, we want to state here both transition probabilities with an explicit notation

Definition 3: Transition Probabilities to termination states $\hat{s}$

$$P(\hat{s}_f|s_s) = \beta(\bar{s}_f)p^{\pi}(\bar{s}_f|s_s) + \sum_{s} p^{\pi}(\bar{s}|s_s)(1-\beta(\bar{s}))P(\hat{s}_f|s) \tag{4.17}$$
$$P(\hat{s}_f|\bar{s}_s) = \beta(\bar{s}_f)\mathbb{I}_{s_f=s_s} + (1-\beta(\bar{s}_s))\sum_{s} p^{\pi}(\bar{s}|s_s)P(\hat{s}_f|\bar{s}) \tag{4.18}$$

The first equation expresses the probability of transitioning to the final state $\hat{s}_f$ starting from state s_s. This is done by noting that the probability to finish in $\hat{s}_f$ is the sum of the probability of arriving to $\bar{s}_f$ and then terminating, plus the probability of transitioning to any state $\bar{s}$ and not terminating, multiplied recursively by the probability of reaching $\hat{s}_f$, this time starting from s. Similarly, the probability of terminating in state $\hat{s}_f$ starting from $\bar{s}_s$ is the probability of terminating in $\bar{s}_s$ if we are already in $\bar{s}_f$, plus the probability of not terminating in $\bar{s}_s$, and for each possible next state $\bar{s}$ recursively reaching $\hat{s}_f$.

Notice that the two probabilities can be expressed in the terms of each other in the following way (refer to figure 4.3 for better understanding)

Definition 4: Relationship between termination state transition probabilities

$$P(\hat{s}_f|\bar{s}_s) = \beta(\bar{s}_f)\mathbb{I}_{\bar{s}_f=\bar{s}_s} + (1-\beta(\bar{s}_s))P(\hat{s}_f|s_s) \tag{4.19}$$

Which makes intuitively sense as we can express $P(\hat{s}_f|\bar{s}_s)$ the probability terminating in $\bar{s}_s$, if we are already in $\bar{s}_f$, plus the probability of not terminating multiplied by the probability of reaching $\hat{s}_f$ starting from s_s. Furthermore, it is interesting to express $P\left(\bar{s}_f|\bar{s}_s\right)$ by unrolling the recursive definition 4.18 (complete derivation in 7.2.2)

Definition 5: Termination state transition decomposition

$$P\left(\hat{s}_f|\bar{s}\right) = \beta\left(\hat{s}_f\right)\sum_{k=0}^{\infty} P^{(k)}\left(\hat{s}_f|\bar{s}\right) \tag{4.20}$$

Which is similar to Harutyunyan et al. (2019), appendix A, although here we have made the explicit distincttion between states s, pre-states $\bar{s}$, and terminal states $\hat{s}$. We have that $\sum_{k=0}^{\infty} P^{(k)}\left(\hat{s}'|\bar{s}\right) = \eta(\bar{s}')$, the count of the average number of times state $\bar{s}'$ is visited during an episode. In this way, we simply have $P\left(\hat{s}_f|\bar{s}\right) = \beta\left(\hat{s}_f\right)\eta(\bar{s}')$. This equation is useful to connect the state visitation distribution $\mu(s) = \frac{\eta(s)}{\sum_{s'}\eta(s')}$ with the state termination distribution $P(\hat{s}'|\bar{s})$, so we can derive two different versions of policy gradient algorithms: one with updates that happen according to the state visitation distribution $\mu(s)$, and the other where updates happen according to the state termination distribution $P(\hat{s}_f|\bar{s})$ (see equation 4.44. Furthermore, the equation has a really intuitive interpretation. First, notice that $\eta(s)$ includes the terms $1-\beta(s')$ implicitly through each term $P^k(\bar{s}'|s)$. Thus, the higher $\beta(s')$, the closer the visitation count gets to the termination probabilities. When $\beta(\bar{s}') = 1$, we have that the average number of times s' is visited during an episode is equal to the termination probability$P(\hat{s}_f|\bar{s})$, as whenever the state is reached the episode terminates. The more β approaches 0, the more P^o gets close to zero, and the higher $\eta(s)$ becomes.

Notice that we have used probabilities distributions which are not discounted in our discussion, for two reasons. The first one is that it is more intuitive to understand the new probability distributions without discounting. The second is that in practice, in policy gradient algorithms the weighting of γ terms are discarded. Thomas (2014) discusses how biased policy gradient updates that ignore the γ terms are in in practice more sample efficient than the unbiased version.

4.3.1 Value functions with Termination

Recall the definitions of state value function $V(s)$ and the state-action value function $Q(s,a)$ for the standard reinforcement learning problem

$$v(s) = \sum_a \pi(a|s)q(s,a) \tag{4.21}$$

$$q(s,a) = \sum_{s'} p(s'|s,a)[r(\hat{s}') + \gamma v(s')] \tag{4.22}$$

We want to highlight that for the standard RL case the reward is obtained at each step, and it depends on the next state (though it can be zero), the current state and the action that was executed. In our case, we model a slightly different problem. First of all, the reward can depend in general from the whole trajectory (although for now, we are only interested in the case where it depends on the final state). Most importantly though, we enforce that rewards are collected only when an episode terminates. We have then that when entering state $\bar{s}'$, two things can happen: we collect a reward of $r(\hat{s}'|s,a)$ and terminate the episode with probability $\beta(\bar{s}')$, or we continue the episode with probability $1-\beta(\bar{s}')$, in which case we will collect a reward of $\gamma v(s')$ in expectation. All together, we can write the reward for state s' with an additional the value function $U(\bar{s}') = \beta(\bar{s}')r(\hat{s}') + (1-\beta(\bar{s}'))\gamma v(s')$. Putting all together, we can write the value functions for our case as

Definition 6: Value functions with termination functions

$$q(s) = \sum_a \pi(s)q(s,a) \tag{4.23}$$

$$q(s,a) = \sum_{s'} p(\bar{s}'|s,a)U(\bar{s}') \tag{4.24}$$

$$U(\bar{s}') = \beta(\bar{s}')r(\hat{s}') + (1-\beta(\bar{s}'))\gamma v(s') \tag{4.25}$$

It is important to keep in mind again that only one reward can be collected per each episode, and the reward is always collected at the last state of the episode. This is fundamentally different from the case of regular sparse rewards, where most rewards are zero, but we are not guaranteed to receive a reward for each episode, and the reward might be collected also earlier in the episode.

We have now two value functions both only depending only on a state, so it is important to highlight the meaning of the two. $U(\bar{s})$ represents instead the expected return of a policy for a state, *before* deciding whether to end the episode. $v(s)$ represents the expected return of a policy before taking action in a state, as in the canonical reinforcement learning case. However, this also means that we are certainly in the state s, and we will not terminate in it, and the termination can happen only from the next state on.

We would like to mention here that we could have used an alternative way to express the same problem through value functions by using a "termination-value" function $U(\bar{s}', b)$, where b is a sample from the Bernoulli distribution defined by β. It is easy to see why in our case it makes little sense to model explicitly $U(s', b)$, for both values of b. For $b = 1$, we get the average return upon termination, which equal to the reward $r(s')$. For $b = 0$ we get the average return by continuing the episode, which is just $\gamma v(s')$. Thus, we are already implicitly using both $U(\bar{s}, b = 1)$ and $U(\bar{s}, b = 0)$.

4.3.2 Episodic return

With the introduction of the termination condition, we now have different ways to express the episodic return. First, we define $r_t^I = r(s_t | s_{t-1}, a_{t-1})$. We choose to use the superscript i to indicate an intrinsic reward since in this thesis we are working exclusively with intrinsic rewards. However, the method works also with extrinsic rewards, and the superscript i can be though to indicate the reward that we would have collected if we stopped at time t. So, although the rewards r_t^I are received at each time step, they are only collected by the policy when the termination function samples $b_t = 1$ from the Bernoulli distribution with mean $\beta(\bar{s}_t)$. Given a discount factor γ and a trajectory $\tau = (s_0, a_0, r_1^I, s_1, a_1, r_2^I, .., s_{T-1}, a_{T-1}, r_T^I, s_T)$, the only actual return collected is the last one, r_T^I, so we could express the return $G_0 = \gamma^T r_T^I$. However, this formulation is not particularly convenient because the return cannot be expressed in a recursive way anymore (see 2.5). By noting that for the trajectory τ the only b_t to be non-zero is the last one, we can write return G_t for the termination case in the following way

$$G_t = \sum_{k=0}^{\infty} \gamma^k b_{t+k} r_{t+k}^I \tag{4.26}$$

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k} \tag{4.27}$$

So for the first line, we can write the return recursively as $G_t = b_t r_t^I + \gamma G_{t+1}$. Furthermore, by defining $r_t = b_t r_t^I$, we obtain the second line and we have that $G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k}$, and that

$G_t = r_t + \gamma G_{t+1}$, so that we simply have that the return with termination is a special case of standard definition of return.

It is also interesting to write down the expectation of the return over the termination function $\beta(\bar{s})$, while still sampling actions according to $\pi(a|s)$

$$G_t^b = \mathbb{E}_\beta[G_t] = \beta_{t+1}(s_{t+1})r_{t+1} + (1-\beta_{t+1}(s_{t+1})\Big[\beta_{t+2}(s_{t+2})r_{t+2} + (1-\beta_{t+2}(s_{t+2})[...]\Big] \tag{4.28}$$

$$= \beta_{t+1}r_{t+1}^I + (1-\beta_{t+1})G_{t+1}^b \tag{4.29}$$

This expectation is particularly interesting because it uses the information of the intrinsic reward r_t^I at each time step, even when the episode did not terminate at time t. In practice, we can calculate the expected return or our model

$$\mathbb{E}_{\pi,\beta}[G_t|S_t = s] = \mathbb{E}_{\pi,\beta}\left[\mathbb{E}_\beta[G_t]|S_t = s\right] \tag{4.30}$$

Notice however that since in practice we terminate episodes and do not sample indefinitely, in order to not introduce bias we would have to consider also all possible transitions that could have happened after the last transition. This means that when using sampled termination, we have to use $G_T = v_\eta(s_T)$, and rely on function approximation in order to obtain estimates of G_T.

4.4 Policy learning

So far we have seen how to express the transition process, value functions and returns when agents can use a termination function in the way discussed. In this section, we show how a policy can be learned in this setting by using policy gradients. We want to describe now how we can update a policy π which is used together with a termination function β. In appendix 7.2.1 we prove that the policy gradient update for the policy π is almost identical to the standard policy gradient with function approximation case (Sutton, McAllester, et al., 2000). For a single initial state s_0, we have

Definition 7: Policy gradient with termination

$$\nabla J(\boldsymbol{\theta}) = \nabla v(s_0) \tag{4.31}$$

$$= \sum_s \eta_\gamma(s) \sum_a \nabla\pi(a|s)q(s,a) \tag{4.32}$$

the only difference being that now the state visitation distribution $\eta_\gamma(s) = \sum_{k=0}^{\infty} \gamma^k P^{(k)}(s|s_0)$ includes the termination function $\beta(\bar{s})$ implicitly through the terms $P^{(k)}(s'|s)$ defined in 4.3. Recall that $\eta_\gamma(s)$ represents the (discounted) visitation counts for state s, and $\mu_\gamma(s) = \sum_s \frac{\eta(s)}{\sum_{s'} \eta(s')}$ the visitation probability, where $\sum_s \eta(s)$ is a constant term which refers to the average number of states visited by the policy. Similarly to Sutton and Barto, 2018, we have

Definition 8: Policy Gradient update

$$
\begin{aligned}
\nabla J(\theta) &= \nabla v_\pi(s_0) && (4.33)\\
&= \sum_s \eta_\gamma(s) \sum_a \nabla \pi(a|s) q(s,a) && (4.34)\\
&= \sum_{s'} \eta_\gamma(s) \sum_s \frac{\eta(s)}{\sum_{s'} \eta(s')} \sum_a \nabla \pi(a|s) q(s,a) && (4.35)\\
&= \sum_{s'} \eta_\gamma(s') \sum_s \mu_\gamma(s) \sum_a \nabla \pi(a|s) q(s,a) && (4.36)\\
&\propto \sum_s \mu_\gamma(s) \sum_a \nabla \pi(a|s) q(s,a) && (4.37)\\
&= \mathbb{E}_{\pi,\beta}\left[\sum_a \nabla \pi(a|s) q(s,a)\right] && (4.38)
\end{aligned}
$$

Now like in section 2.1.3, we can use sampled returns G_t as an estimate for $q(s,a)$, and introduce $v(s)$ a baseline

$$\nabla_\theta V(s) = \mathbb{E}_{\pi,\beta}\left[\nabla \log \pi(a|s)\left[G_t - v(s)\right]\right] \quad (4.39)$$

Notice that this update takes into account all actions taken in all states, apart the last one where we terminated the episode in, as no action has actually been taken in it. Also, we can use this update rule in two ways: in the first one, we use the return G_t, so the only reward collected is at the final state. Alternatively, we can use G_t^b so at each step we use the average over the two outcomes of $U(\bar{s})$. Hopefully, the latter is more sample efficient, although care must be taken as we need an estimation of $V(s_T)$.

4.5 Learning a Termination Function

We want to show now how to learn a termination function by using policy gradients in different ways, which are going to be compared in chapter 5. The first way to learn a termination function follows what done in Bacon et al., 2016. In appendix, section 7.2.3 we prove that

Definition 9: Termination gradient - Normal method

$$\nabla_\psi U(s) = \sum_{s'} \eta_\gamma(\bar{s}')\nabla_\psi \beta_\psi(\bar{s}')[r(\hat{s}') - \gamma v(s')] \tag{4.40}$$

This equation has a very intuitive interpretation. First, as in standard policy gradients, we can perform updates according to the visitation distribution just by taking samples from the policy and the termination function. Second, the gradient of β is multiplied by the difference between the reward we would have collected by stopping and the discounted average return we would get by continuing. We can think of this term as a kind of "advantage". When the reward we could have collected is higher than the expected return, the "advantage" is positive, which makes the gradient positive and increases the probability of terminating. On the opposite, when the expected return we would get by continuing is higher than the reward we could have collected, the "advantage and the gradient are negative, so the termination probability decreases. This gradient is also interesting as it already considers the (intrinsic) reward $r_t^I = r(s_t|a_{t-1}, s_{t-1})$ at each time step, even if we did not collect it.

We can also write a sample-based version of the update rule which does not use function approximation

Definition 10: Termination gradient - sampled return baseline

$$\nabla_\psi U(s) = \sum_{s'} \eta(\bar{s}')\nabla_\psi \beta_\psi(\bar{s}')[r_t^I - \gamma G_{t+1}] \tag{4.41}$$

This formulation can be convenient as often in practice the value function estimate can be very imprecise, and cause feedback loops in our case, so grounding the updates on the actual return obtained can help. We should take care however to the last state of a trajectory. For a trajectory $s_0, a_0, r_1, s_1, ..r_T, s_T$, is G_T just equal to r_T we do not have any $V(s_T)$ anymore. this would mean the last term of the "advantage" would always be zero.

Here we show a second way to way to update the gradient of the termination function. Given 4.18, we have that $\sum_{k=0}^{\infty} P^{(k)}(\bar{s}'|\bar{s}) = \frac{P(\hat{s}'|\bar{s})}{\beta_\psi(s')}$

Definition 11: Termination gradient - Normal(last step) method

$$\nabla_\psi U(s) = \sum_{s'} \sum_{k=0}^{\infty} P^{(k)}(s'|s) \nabla_\phi \beta_\psi(\bar{s}')(r(\hat{s}') - \gamma v(s')) \tag{4.42}$$

$$= \sum_{s'} \frac{P(\hat{s}'|s)}{\beta_\psi(s')} \nabla_\phi \beta_\psi(\bar{s}')(r(\hat{s}') - \gamma v(s')) \tag{4.43}$$

$$= \sum_{s'} P(\hat{s}'|s) \nabla_\phi \log \beta_\psi(\bar{s}')(r(\hat{s}') - \gamma v(s')) \tag{4.44}$$

This Which updates the termination function according to the termination-state distribution $P(\hat{s}'|s)$, and can avoid a problem that happens in practice during training when using function approximation.

4.5.1 Actor-Critic with Termination

We can derive an actor-critic update from the policy gradients formulation, similarly to section 2.1.4

$$\mathbb{E}_{\pi,\beta}\left[\log \pi(A_t|s)(G_t - v(s)\right] \tag{4.45}$$

$$\approx \mathbb{E}_{\pi,\beta}\left[\log \pi(A_t|s)(r_{t+1} + \gamma v(s') - v(s)\right] \tag{4.46}$$

Notice that this update still relies on the sampling of the termination function, and for each episode, the non zero reward is the one on the last state visited as $r_t = b_t r_t^I$. To allow updates at each time step, we can use again the expected return G_t^b under the termination function (or alternatively, use $U(\bar{s})$

$$\mathbb{E}_{\pi,\beta}\left[\log \pi(a|s)(\beta_\psi(\bar{s}')r^I(\hat{s}') + \gamma(1 - \beta_\psi(\bar{s}'))v(s') - v(s)\right] \tag{4.47}$$

The estimate of the baseline will be thus updated towards the target $\beta_\psi(\bar{s}')r^I(\hat{s}') + \gamma(1 - \beta_\psi(\bar{s}'))v(s')$ for each visited next state s'.

As the update of the termination function in 4.40 happens independently for each state transition from s_t to s_{t+1}, the update remains the same for the Actor-Critic.

4.5.2 Alternative Termination update: Policy Gradient

Here we show a third way to update termination function $\beta_\psi(s)$, by using policy gradients with the log-derivative trick. Consider the probability of a trajectory $\tau = (s_0, a_0, s_1, a_1, .., s_T)$

$$p(\tau) = \beta_\psi(\bar{s}_T)p(s_0)\prod_{t=1}^{T-1}(1-\beta_\psi(\bar{s}_t))\pi(a_t|s_t)p(s_{t+1}|s_t, a_t) \tag{4.48}$$

Taking the gradient of the log probability, we have

$$\nabla_\psi \log p(\tau) = \nabla_\psi \log(\beta_\psi(\bar{s}_T)) + \sum_{t=1}^{T-1}\nabla_\psi \log\left(1-\left(\beta_\psi(\bar{s}_t)\right)\right) \tag{4.49}$$

$$= \sum_{t=1}^{T}\nabla_\psi \log p_\beta(s_t) \tag{4.50}$$

where we have introduced the notation

$$p_{\beta\psi}(\bar{s}_t) = \begin{cases} (1-\beta_\psi(\bar{s}_t)) & t < T \\ \beta_\psi(\bar{s}_t) & t = T \end{cases} \tag{4.51}$$

in order to reduce the clutter for the next derivations. Furthermore, this notation allows us to see the connection of the termination function with a standard actions, although they are shifted by 1. Using the log-derivative trick as in 2.25

$$\nabla_\psi \mathbb{E}_{\tau\sim\pi_\theta,\beta_\phi}[R(\tau)] = \mathbb{E}_{\tau\sim\pi_\theta,\beta_\phi}\left[G(\tau)\cdot\sum_{t=1}^{T}\nabla_\psi \log p_\beta(\bar{s}_t)\right] \tag{4.52}$$

We show in appendix 7.2.4 that

Definition 12: Termination Gradient - Policy gradient method

$$\nabla_\psi \mathbb{E}_{\tau\sim\pi_\theta,\beta_\phi}[R(\tau)] = \mathbb{E}_{\tau\sim\pi_\theta,\beta_\phi}\left[\sum_{t'=1}^{T}\nabla_\psi \log p_\beta(\bar{s}_t)G_T\right] \tag{4.53}$$

This result is very interesting because it is the equivalent to standard policy gradient methods for updating a policy $\pi(a|s)$, where probability of an action a is updated based on its log probability $\log\pi(a|s)$ and the return G_T. In this result, the action corresponds to terminating, and the probability is indeed $p_\beta(\bar{s}$.

4.6 Practical algorithm

In this section we show a pseudocode algorithm for learning Temporally Extended Options in the framework described in this chapter. This algorithm is built on top of the one for fixed length options described in chapter 3.2.2, but differs in the way episodes terminate: single episodes terminate based on the termination function β_ψ as explained in section 4.2. Furthermore, so far we have discussed how to update options one at a time, given the intrinsic reward. Learning multiple options at the same time can be easily done by running episodes with each of them, as the rewards can be collected regardless of other options. However, it is important to always update the discriminator q_ϕ by running episodes of multiple options at the same time, in order to not bias its outputs towards one only option. This is why in the algorithm batches of option and trajectories are collected and used for updates.

Algorithm 2: Temporally Extended Variational Option Discovery

```
Generate initial policy $\pi_\theta$, discriminator $q_\phi$, and termination function $\beta_\psi$
for $k = 0, 1, 2, \ldots K$ do
    Sample option-final state pairs $\mathscr{D} = \left\{\left(\omega^n, s_f^n\right)\right\}_{n=1,\ldots,N}$, by first sampling
        $\omega \sim \Omega$ an option and then rolling out a trajectory $\tau$ in the environment
        $N$ times, obtaining final states$\{s_f^n\}_{n\ldots N}$
    Calculate intrinsic rewards $r_I^n = -\log p(\omega^n) + \log q_\phi(\omega^n|s_f^n)$
    Update policy with any reinforcement learning algorithm to maximize
        the rewards $\{r_I^n\}_{n=1,\ldots,N}$ using batch $\mathscr{D}$
    Update termination function $\beta_\psi(s)$ in each state of $\tau$ to maximize
        the rewards $\{r_I^n\}_{n=1,\ldots,N}$ using batch $\mathscr{D}$ by using eq. 4.40, 4.44, or 4.53
    Update discriminator by supervised learning: maximize $\mathbb{E}\left[\log q_\phi(\omega|s_f)\right]$
        using batch $\mathscr{D}$
    Reset environment to the initial conditions
end for
```

5 Experiments

5.1 General setup

In this chapter, we evaluate the described model on different environments and show that options of different length can be learned with the framework described in the previous chapter. We compare the results obtained with termination function to the ones without, showing that our approach is not only more general, but also more suited for certain tasks, such as navigation in an environment with stochastic dynamics.

In particular, in section 5.3.1 we show that we can learn Temporally Extended Options that maximize the intrinsic reward with the use of a termination function. In section 5.3.2 we show how such options correctly terminate without wasting time steps idle, while also learning options of different length. In section 5.3.3, we compare different methods to achieve Temporally Extended Options

5.1.1 Implementation

We implement the general algorithm described in 4.6 by using as objective $I(s_f|\omega)$, the mutual information between the final state of the option and the option itself. We choose to use this objective because it is the most simple, and can be maximized in the environments we use (unlike Eysenbach et al. (2018), where if trajectories overlap the objective cannot be maximized, as the objective is the sum of mutual information at each step), allowing a clearer interpretation of the results. The objective is effectively equivalent to the objective used in Gregor et al. (2016), as they keep the starting state fixed.

We run the algorithm for N iterations. At each iteration, we run a batch of N_o episodes. However, we do not run one episode per option, but instead sample the N_o options from a discrete uniform distribution to introduce some noise in the updates. This choice is due to the fact that in practice running one episode per option resulted in options staying close to the initial state and not learning diverse behavior for a longer time at the beginning of the training. The updates of $\pi(a|s), v(s), \beta(s)$, together with the loss $-\log q$ are averaged over all options present in the batch. For each experiment, we impose a maximum duration of episodes of 50, in order to avoid doing unnecessary computations as most states are reachable in at most 32 steps. Notice that the method works as well with higher values of max duration, and we choose 50 as a trade-off between efficiency of computations while still allowing us to show that options converge even when allowed to run for more steps than necessary.

We implement all the components of the RL agent $(\pi(a|s), v(s), \beta(s'))$, as neural networks. We use the same architecture for all the components of the model, a three hidden layer network with 300 neurons and ReLU activation function, where the first layer is shared between all three of $\pi(a|s), v(s), \beta(s')$. The discriminator similarly is a neural network with 3-5 layers depending on the environment

5.1.2 Initialization of the Termination Function

A critical part of the algorithm is ensuring that options explore enough. One obstacle to this is that the standard initialization of $\beta(s)$ with a sigmoid function $\sigma(x) = \frac{1}{1+e^{-x}}$ outputs values around 0.5 when using neural networks with classical initializations (He et al., 2015) (Glorot and Bengio, 2010) and small inputs. This makes the options terminate too often at the beginning when the policies are still random, and does not allow proper exploration of the state space, preventing options from learning any meaningful behavior. Thus, we use a simple trick to make sure that the initial values of $\beta(s)$ are small enough to allow proper initial exploration: we shift the input values of the sigmoid by a constant value c. Thus, we use $\beta(s) = \sigma(f(s) - c)$ as termination probability, where $f(s)$ is the output of a neural network. With $c = 2$, initial values are around 0.12. This means that for the first few episodes, the distribution of the length of the episodes will be close to a geometric distribution [1] with parameter $p = 0.1192$, giving an average length of 8.4 steps, and that 92% of episodes will be of length less than or equal to 20.

5.2 Gridworld environment description

For the first experiments, we use simple discrete state space gridworlds. The implementation is a custom modification of the popular MiniGrid-gridworld implementation (Chevalier-Boisvert et al., 2018). The gridworlds consist of NxM discrete states, with N rows and M columns, where the upper left corner is identified as $(0,0)$ and the bottom right corner is (N, M). The agent can use 4 actions: up, down, left and right. The borders of the grid are occupied by walls, and the agent cannot ever be positioned on the first or last column or row of the grid. If the agent selects an action that would make it go in the position of the wall, it instead stays

[1] https://en.wikipedia.org/wiki/Geometric_distribution

on the same current state, so it does not move. For example, if in the state $(1,1)$ the agent performs the actions up or left, it stays in position $(1,1)$.

As a state space representation, we use the normalized coordinates (x, y) of the current location of the agent. The coordinates can go from 0 to M for the width of the environment, and from 0 to N for the height. For the normalization, we subtract by $\lfloor(M/2)$ (and floor($N/2$)), then divide by M (and N), so that the center of the grid has coordinates of $(0,0)$ (for even N and M), and the values are in the interval $[-1,1]$. This is done in order to have a range of inputs which is not dependent on the size of the environment and so that the inputs are in the range neural networks typically work best with. Notice that this representation refers only to the input of the neural networks, and in our discussion we will still refer to states ranging from 0 to N for the rows and from 0 to M for the columns.

In this section, we use two environments to demonstrate the capabilities of the proposed method. The first environment we use is a discrete grid size $3x35$, which means that the the actual states reachable by the agent are on a 1×33 row, surrounded by walls. Since we effectively have only one row, we only allow the agent to move using two actions: left and right. The agent always starts in the cell $(1,1)$, at the center left of the space. The reason we use this environment is to show that we can successfully learn options that go to different parts of the space, even further away in one dimension. In this environment, we use 4 options for all experiments. In this and the following sections, this environment will be referred to as Line environment.

The second environment consists of a 33x33 empty grid, with walls on each side, so that the agent can actually move only between cells $(1,1)$ and $(N-1, M-1)$. The agent starts in the middle of the grid, and has all four directional actions available. In this environment, we use 8 options as there are more directions where the agent can go to. From this point on, we call this environment **Square** environment.

5.3 Analysis of learned options

In this chapter we will be evaluating the learned options according to different criteria, listed below. We introduce a concise terminology about these criteria to make the following discussion more clear.

- **Mutual Information (MI)** $I(s', \omega)$, which ranges from 0 to $\log N_o$ (where N_o is the number of options), and gives an estimate of the effective number of options learned (by taking its exp, which ranges from 1 to N_o). Since in our case this equals to $I(s_f, \omega) = H(\omega) - H(\omega|s_f)$, as explained in the previous chapter, we will be keeping track of the entropy $H(\omega|s_f)$, which has to be minimized to 0 in order for the mutual information to be maximized.

- **Discriminator loss**, which is $-\log q_\phi(\omega|s_f)$, the negative output of the discriminator for a single option ω and final state s_f. Its expectation over options and final states is the conditional entropy $H(\omega|s_f)$. We expect learned option to have close to zero discriminator loss, meaning that given a state, we are almost sure of which option lead to that state.

- **Option length**: the number of steps taken by the options to terminate (or 50 if it does not terminate before 50 steps). We expect the learned options to have various lengths: some options will terminate after a few steps, close to the initial state, while others will terminate further away.

- **Option distance**: the distance between the initial and final state of the option. As with the option length, we expect options to reach states with various distances from the initial state.

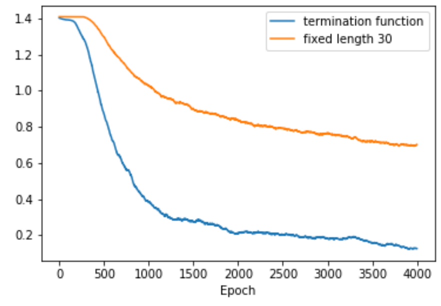
- **Option difference**: the difference between option length and option distance. Optimally, we would like options to have an option difference of zero, so that the length of an option is the same as the option distance, which means that the option terminated as soon as it reached its final state and did not waste time steps.

5.3.1 Mutual Information of options

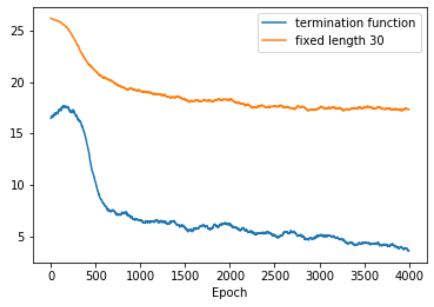
The first way to assess the quality of the learned options is the mutual information $I(s', \omega)$. We run the algorithm as described in the previous sections and report results for the model with highest final mutual information over 7 seeds. We do so by showing the discriminator loss on the sampled final states s_f, averaged over all options in each iteration.

In plots 5.1a and 5.1c we show the discriminator loss $-\log q_\phi(\omega|s_f)$ and the Option Difference for both environments. We can see that the agent manages to correctly minimize the discriminator loss, thus maximizing the mutual information. In particular, for the Line environment we achieve a mutual information of 1.25, where the maximum value is $\log(4) \sim 1.386$. This results in the effective number of options being $\exp(1.25) = 3.5$.The reason for this to be slightly lower than 4 is that the discriminator q_ϕ is not a perfect approximation of the true $p(\omega|s_f)$. We can see in fact that in 5.2 every option corresponds clearly to a different behavior. However in some cases it can happen that the discriminator is not fully aligned with the options, which causes the mutual information to not be fully maximized.

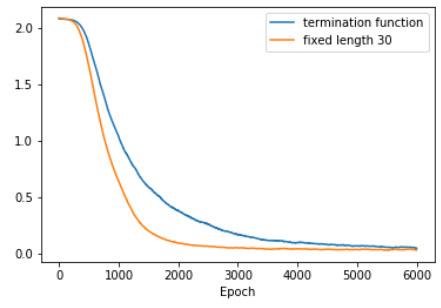
In the same plots, we also show the performance of the agent with fixed length options. This does not manage to minimize the loss for the Line environment 5.1a, but it does so for the Square environment 5.1c. The main difference is however in the option difference: it is clear from 5.1d that even though the options are different from each other, they waste steps idle without any possibility of terminating.



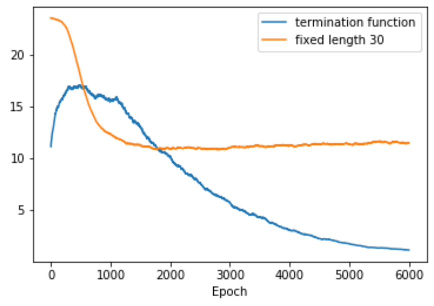
(a) Line environment: Average discriminator loss $-\log q_\phi(\omega|s)$ per epoch, averaged over all options ω_i, $i \in [1,4]$



(b) Line environment: Option Difference per epoch, averaged over all options ω_i, $i \in [1,4]$



(c) Square environment: Average return loss $-\log q_\phi(\omega|s)$ per epoch, averaged over all options ω_i, $i \in [1,8]$



(d) Square environment: Option Difference per epoch, averaged over all options ω_i, $i \in [1,8]$

Fig. 5.1: Comparison between fixed length options and Temporally Extended Options. For both environments, fixed length options waste a large amount of steps idle even when the goal has been reached causing a high option difference (right side), which is instead minimized by our Temporally Extended Options.

To obtain a better understanding of the options learned by the agent, we to look at the plots resulting from the Line environment in figure 5.2. First, we are interested in knowing where the learned options terminate. In the figure we can see one example of the termination distribution $P^o(s_f)$ (approximated by calculating the visitation counts over 200 samples), for each option, where s_0 is the state $(1,1)$. We can see that indeed the learned options end up in parts of the state space which are located at different distances from the starting point. This demonstrates that the method proposed for learning options can work in practice: options that cover the state space can be found, without the need to specify their length. In the 2nd column, we can see the resulting discriminator, which is also the reward function for each option. The discriminator correctly assign probability mass in the 4 regions of the state space where the options terminate. In the 3rd column we see the resulting termination function, which is also increasing around the states where each option terminates. States where the termination function is 1 are typically never reached, and in the future work chapter we discuss how this observation can lead to interesting outcomes.

in 7.3c we see the results for the square environment. The options learned to go in different directions (left- right, up) and to different corners (up-right, bottom-left and bottom-right). Again, the discriminator $p(\omega|s)$ and the termination function $\beta(s)$ correctly have high values close to the states where options terminate.

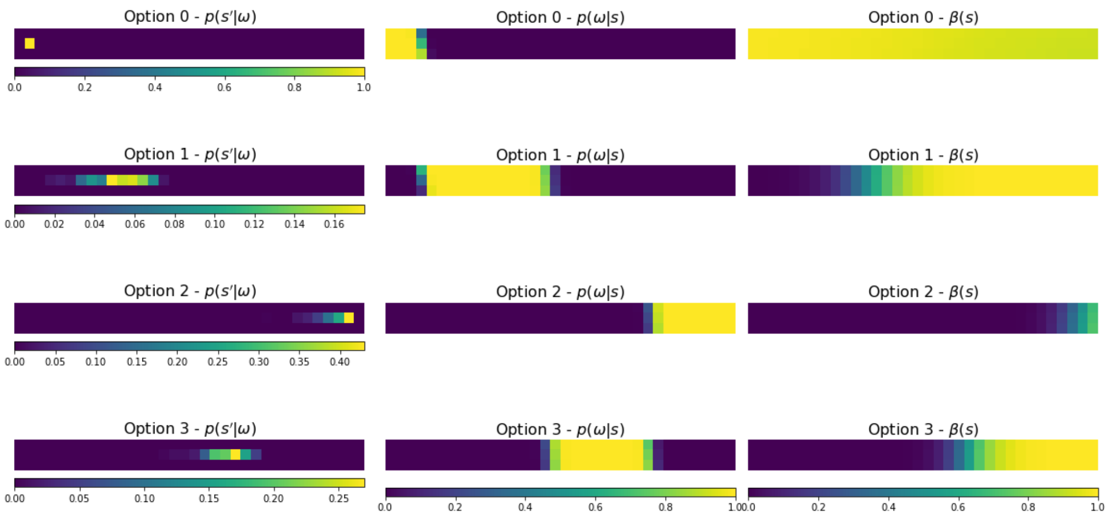


Fig. 5.2: Visual plots of the options learned in the line environment. On the left: transition probabilities $p(s_f|\omega)$, indicating the states where the agent will be at the end of an option. Center: learned discriminator $q(\omega|s)$, which gives an estimate of the probability of the option having terminated in s. Right: termination probability $\beta(s)$

5.3.2 Length of options

In order to understand the usefulness of the proposed method, we have to investigate not only whether the mutual information is maximized, but also look at the efficiency of the learned options in terms of time. In fact, it would be not useful to have options that reach a state but stay idle for many steps without terminating. Thus, we need to make sure that options only last for the minimal amount of steps needed to achieve their goal. For this reason, we want to understand how the *length* of options relates to the position of the state they terminate in, which can be also phrased as the *distance* between their initial state and the final state. In an optimal scenario, we would like the Option Difference to be close to zero.

We first show in figures 5.3 that in the experiments described in the previous section, all options eventually learn to terminate without spending too many steps idle (additional results for the Square environment are shown in fig 7.1 in the appendix). Although the initial length of options is much higher than the distance of the final states reached, towards the end of the training all options learn to terminate in the states they end in without wasting time.

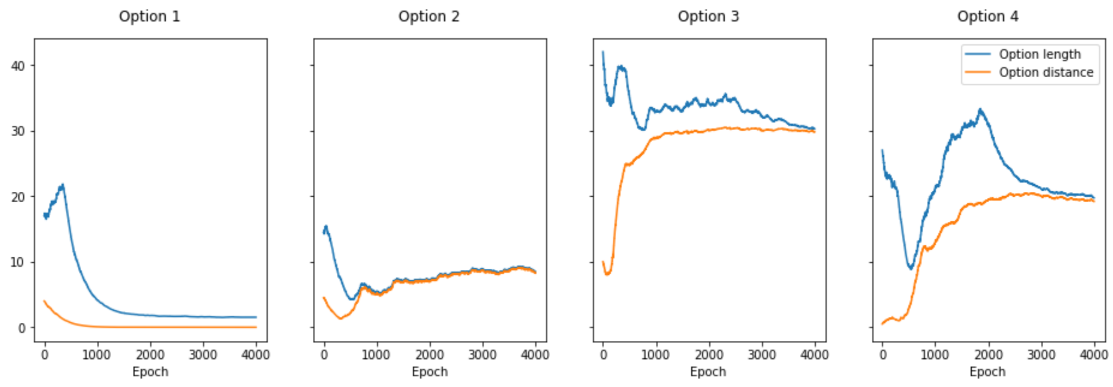


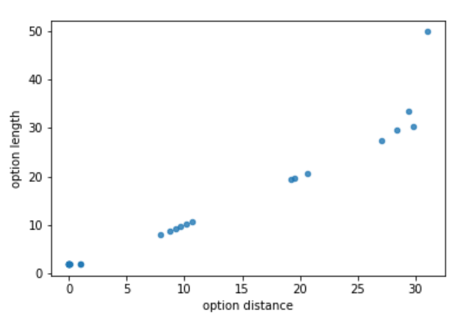
Fig. 5.3: Line environment: Length of options compared to the distance of the final state reached. Eventually, all options learn to terminate in the states they end in without any overhead of steps.

To have a better insight on the relationship between the length and distance of options, we run the algorithm many times with the same parameters, and report such values for each option. In figure 5.3.2, we can see that options of various lengths can be learned. For both environments, these plots show us peculiar behavior or the model proposed. For the Line environment, we clearly see 4 clusters of options. One set of options tends to finish in states close to the initial state, one close to the most distant state to the right and the other two around the middle left and middle right of the environment. This seems to be the easiest way for the algorithm to optimize the objective, consistently emerging over different runs. The only exception is for options that end up in the rightmost state, where there is a wall: in this case, the length of the option does not correctly converge to the actual length of the option. Finally, 5.4d shows that most of the options have as distance less than 2 steps more than necessary to reach the final state.

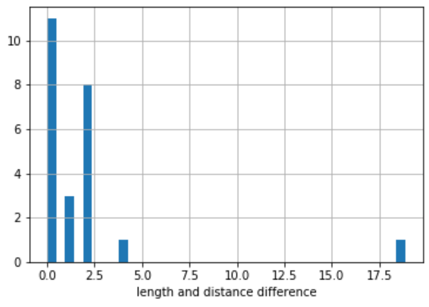
Regarding the Square environment in 5.4c, we have that options instead can lead to states at different distances in different runs of the algorithm, as in 2 dimensions there are many more ways to partition the state space. Interestingly, we have again that far away states close to the walls the length of the option does not match the distance of the state reached. As for the Line environment, in the Square environment too most options last up to 2 steps more than needed to reach the final state as shown in 5.4d . This is a very interesting behavior of the algorithm, and seems to be consistent over different environments: when an option reaches a state where it gets stuck (such as a wall in our case), then it can happen that the length of the option never converges to the actual distance of the state. Although this happens relatively little in the experiments shown above, this tends to happen more often with different hyperparemeter choices. Investigating this behavior, this appears to be linked to the fact that the value function $v(s)$ becomes higher than the highest possible reward $\log(N_o)$ where N_o is the number of options. This makes the term $r(s) - v(s)$ of 4.40 be always negative, increasing decreasing then $\beta(s)$, the probability that the option terminates in that state. For this reason, we ran experiments with $v(s)$ that was capped at a maximum value in different ways, but the behavior continued to be present. This leaves open the possibility that when receiving a very negative reward, the termination function obtains learns to output values very close to zero, and never gets updated correctly later. It is in fact a known problem of sigmoids as activation functions in neural network that when they reach very small values, gradients become very small and the network hardly changes values.

5.3.3 Comparison of termination methods

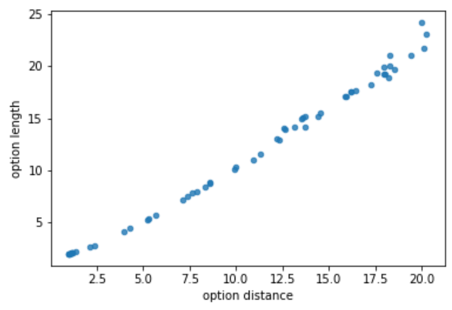
We have seens so far that termination function as described in 4.40 produces options that correctly maximize the mutual information, while learning to navigate to states at different distances. We now want to compare the different ways of updating the termination function introduced in the previous chapter. Furthermore, we also analyze how the other approach to



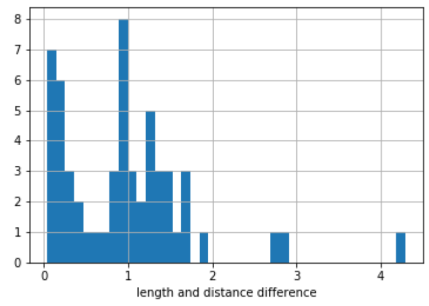
(a) Line environment: Scatter plot of option length and option distance



(b) Line environment: Histogram of the option difference. Most of the learned options have less than 2 steps difference.



(c) Square environment: Scatter plot of option length and option distance



(d) Square environment: Histogram of the option difference. Most of the learned options have less than 2 steps difference.

Fig. 5.4: These plots are referred to the experiments ran before, showing options obained over 7 runs with the same hyperparemeter configuration. Top row results are from the Line environment, bottom row are from the Square environment.

learn temporally extended options, namely a termination action, performs in comparison to the termination function methods.

We use the 3 methods described in the previous chapter for updating the termination function.

- **Normal** method uses 4.40 to update the termination at each step, using the reward model r_i for each visited state, regardless of whether the agent terminated in that state or not.

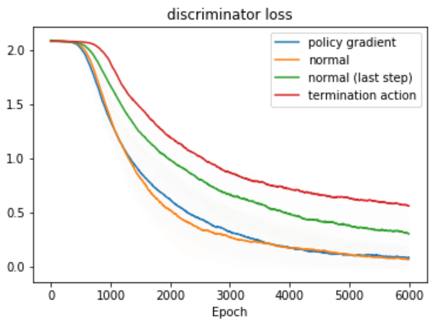
- **Normal (last step)** method uses 4.44 to update the termination function only in the last state visited, where the agent actually terminated the episode, using the intrinsic reward r_i of the last state.

- **policy-gradient** uses 4.53 to update the termination function at each step, but using only the discounted episode return, as typical of policy gradient updates, which correspond to the discounted intrinsic reward of the last state in our case.

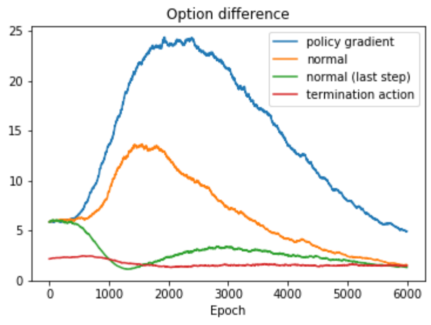
Finally, the alternative to a termination function is a **termination action**, which is added to the set of possible actions available to the policy $\pi(a|s)$. This is straightfowards and does not cause any additional modification to Reinforce, the underlying algorithm used.

We run all the methods described and report their conditional entropy loss $-\log(\omega|s_f)$ over 7 seeds in 5.5a, together with the Option Difference in 5.5b In terms of maximization of the mutual information, we can see that the termination action method does not perform as well as the other methods. At the end of the learning, it reaches a mutual information of 1.54, which corresponds to learning only 5 options out of 8. Although the difference between distance and length in 5.5b is low, this holds only because the options terminate always very close to the initial state. Thus, we can say that the termination action method did not succeed in the task. A similar situation holds for the Normal (only last step) method, which obtains a mutual information of 1.78, which corresponds to 5.8 options, where the options are again located close to the initial state of the agent.

The two methods that successfully maximize the mutual information are the Normal and Policy Gradient one, as shown in 5.5a. They however differ in the length and distance of the options learned. In particular, the Policy Gradient method tends to have options with much higher length than needed while learning, and exhibits higher difference between length and distance in the end. The Normal approach also tends to have high length of options during training, but finally reaches a point where length matches the distance almost perfectly. Relevant quantities for comparison are given in table 5.1. From this, we can say that the log-trick method managed



(a) Loss $-\log q_\phi(\omega|s)$ for different termination methods over time



(b) Option difference over time

Method	MI	Number of options	Option Distance	Option Length	Option Difference
Normal	2.02	7.5	10.6	11.8	1.2
Normal (last step)	1.78	5.9	6.1	7.2	1.1
Policy Gradient	2.00	7.4	12.8	16.5	3.7
term. action	1.54	4.7	7.0	8.7	1.7

Tab. 5.1: Comparison of different termination methods after 6000 iterations for the Square environment

to learn different options, albeit not totally efficient in terms of duration; Instead, the Normal method successfully achieves the objective of learning different options of different length.

The fact that the termination action performs worse than other methods can be explained by the fact that it only updates the values of the termination function when the termination action is actually sampled. Furthermore, action probabilities and termination probabilities are tied together by the softmax in the policy, which might makes it harder to optimize compared to having termination independent from actions. This is because if an action is more rewarding than terminating, its probability will increase, decreasing the probability of terminating instead in that state, which causes the termination action getting sampled less, eventually leading to options that never terminate.

The Normal method performing better than others is expected, as it updates the termination function at each time step using the information of the intrinsic reward in that state even without actually terminating in it. The log-trick also updates the termination function at each time step, although it only uses the information of the last state of the episode, where the agent terminated. This adds more variance in the updates, and can be an explanation why it is less efficient in matching option length with distance of the final state. Finally, the Normal (last step) method also uses the intrinsic reward of the final state, but it only updates the termination function in the final state too, making it even less efficient,

From this comparison, we can state that the most efficient method for learning temporally extended options is the Normal method described in 4.40.

6 Conclusion

6.1 Discussion

In this thesis we have shown that Temporally Extended Options can be learned for discrete state space tasks, and that the resulting options correctly maximize the mutual information $I(s_f, \omega)$ between final states and options, resulting in options that cover different parts of the state space. This was until now only doable using fixed length options, which constrain the duration of options to a fixed number. We have seen how these kind of options are limited and less general for different reasons: they require prior knowledge for deciding their length, and they waste time steps even after they reached their goal. By using a termination function $\beta(s)$, we have managed to learn options that do not require prior information and also do not waste time once they reached their goal.

From our experiments, it can be seen that between the two alternatives for learning Temporally Extended Options, the termination function approach is preferred for two reasons. The first is that its alternative, a termination action, is not directly applicable to continuous environments. The second is that, as shown in the previous chapter, the termination action often does not manage to optimize the objective, and fails to learn different options in practice. We have shown how to derive different methods for updating termination functions, and empirically evaluated their results on different environments. The Normal method (eq. 4.40) was found to successfully and consistently maximize the intrinsic reward while also obtaining a low Option Difference, which means that the options learned do not waste any time steps and correctly terminate whenever they reach a state where they should terminate.

For simplicity, although in Chapter 4 we introduced our methods to maximize a general objective $I(\omega, \tau)$, we have based our experiments on the objective $I(\omega, s_f)$, taking into account only the final state of a trajectory and discarding the rest. Using different parts of a trajectory to determine termination can be easily done by changing the termination function $\beta(s)$ and discriminator $\log q_\phi(\omega|\tau)$ to $\beta(\tau)$ and $\log q_\phi(\omega|\tau)$ so that they take as inputs a general representation of a trajectory τ.

6.2 Future work

As this thesis showed that termination function can be a viable and efficient approach to learn Temporally Extended Options in simple environments, there are multiple directions of research that should be explored in order to fully understand the potential of this approach.

In this thesis, we only used discrete state gridworlds. A future direction of work is exploring how Temporally Extended Options can be learned in practice in continuous state spaces. In our preliminary experiments, which are not included in this thesis, continuous state spaces are more difficult to handle as the discriminator can learn to split the environment in very close fine grained regions, thus making the options learned useless. Thus, the main challenge in trying to solve continuous state problems is finding a correct balance between the learning of discriminator $q_\phi(\omega|s)$ the quality of options learned.

Another point to be addressed is the sample efficiency of the method. Our objective in this thesis was introducing a termination function for learning temporally extended options, without caring about sample efficiency. Thus we have chosen to stick to one of the most simple and straightforward approaches for RL problems, namely Reinforce (Williams, 1992), which is known for having high variance and thus being usually not sample efficient. After showing that the method can work in practice with Reinforce, sample efficiency can be increased by using some of its improvements, such as Actor-Critic and PPO (Schulman, Wolski, et al., 2017). Another way to increase the sample efficiency of the method involves performing updates of the discriminator using not only the final state of a trajectory, but also the other ones where termination did not happen.

All the results shown above involve using a termination function whose values range from 0 to 1. However, this can be considered a limitation for some cases. In particular, the learned termination can have very steep changes of value from 0 to 1 in close states, leading to options that terminate almost deterministically in the same state. This can be not desired for different reasons: for example, we found that often In order to obtain options that can end in multiple states, we would need to avoid such changes in the values of the termination function. One simple but effective way to do this is limiting the maximum value of the termination function, so it can end options only with a certain probability p in each state. This can be particularly useful to control the amount of noise we expect to have in the final state reached by options and can be suited for environments where the typical duration of an episode is much higher than the one explored in this thesis.

One of the main applications of this work is Hierarchical Reinforcement Learning. In particular, our method can be used to initialize options used by hierarchical methods, and has been developed specifically to be fully applicable to the Option-Critic (Bacon et al., 2016), in contrast to fixed length options, which cannot not be directly used for this purpose as they do not learn a termination function. An initialization of this kind can heavily increase the performance of Option-Critic with respect to training from scratch, as a similar outcome has been observed for pretraining goal conditioned policies in Sharma et al. (2019) Eysenbach et al. (2018).

Finally, the most interesting and promising direction of research is applying the termination function to goal oriented hierarchical methods, such as those described in Section 3. This requires having a termination function that is both dependent on the state and on the current goal. This could be beneficial also to stabilize the training: in our current method, terminating leads to high reward only if the option has high reward in a certain part of the state space. This region is however not explicit, and can change over time. Instead, in goal oriented methods the termination needs to happen depending on the current goal, thus can have high rewards since the beginning earlier than with our method. As in recent years most of the successful methods in Hierarchical Reinforcement Learning are goal oriented methods, we conjecture that applying a termination function in the way discussed in this thesis to such methods can further extend the capabilities of hierarchical models concretely even for more difficult tasks.

Bibliography

Achiam, Joshua, Harrison Edwards, Dario Amodei, and Pieter Abbeel (2018). “Variational Option Discovery Algorithms”. *CoRR* abs/1807.10299. arXiv: 1807.10299 (cit. on pp. 1, 2, 21–23).

Andrychowicz, Marcin, Filip Wolski, Alex Ray, et al. (2017). “Hindsight experience replay”. *Advances in Neural Information Processing Systems*, pp. 5048–5058 (cit. on p. 14).

Aubret, Arthur, Laetitia Matignon, and Salima Hassas (2019). “A survey on intrinsic motivation in reinforcement learning”. *arXiv e-prints*, arXiv:1908.06976, arXiv:1908.06976. arXiv: 1908.06976 [cs.LG] (cit. on pp. 18–20).

Bacon, Pierre-Luc, Jean Harb, and Doina Precup (2016). “The Option-Critic Architecture” (cit. on pp. 2, 15, 16, 26, 27, 35, 53, 66).

Bellemare, Marc G, Yavar Naddaf, Joel Veness, and Michael Bowling (2013). “The arcade learning environment: An evaluation platform for general agents”. *Journal of Artificial Intelligence Research* 47, pp. 253–279 (cit. on p. 6).

Bousmalis, K., A. Irpan, P. Wohlhart, et al. (2018). “Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping”. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4243–4250 (cit. on p. 1).

Burda, Yuri, Harrison Edwards, Amos J. Storkey, and Oleg Klimov (2018). “Exploration by Random Network Distillation”. *CoRR* abs/1810.12894 (cit. on p. 18).

Burda, Yuri, Harri Edwards, Deepak Pathak, et al. (2018). “Large-scale study of curiosity-driven learning” (cit. on p. 18).

Chevalier-Boisvert, Maxime, Lucas Willems, and Suman Pal (2018). *Minimalistic Gridworld Environment for OpenAI Gym*. https://github.com/maximecb/gym-minigrid (cit. on p. 42).

Eysenbach, Benjamin, Abhishek Gupta, Julian Ibarz, and Sergey Levine (2018). “Diversity is all you need: Learning skills without a reward function” (cit. on pp. 1, 2, 21, 23, 41, 53).

Florensa, Carlos, Yan Duan, and Pieter Abbeel (2017). “Stochastic Neural Networks for Hierarchical Reinforcement Learning” (cit. on p. 23).

Fox, Roy, Sanjay Krishnan, Ion Stoica, and Ken Goldberg (2017). “Multi-level discovery of deep options” (cit. on p. 19).

Garcia, Carlos E, David M Prett, and Manfred Morari (1989). “Model predictive control: theory and practice—a survey”. *Automatica* 25.3, pp. 335–348 (cit. on p. 24).

Glorot, Xavier and Yoshua Bengio (2010). “Understanding the difficulty of training deep feedforward neural networks”. *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pp. 249–256 (cit. on p. 42).

Gregor, Karol, Danilo Jimenez Rezende, and Daan Wierstra (2016). “Variational intrinsic control” (cit. on pp. 1, 2, 21–23, 27, 41).

Haarnoja, Tuomas, Kristian Hartikainen, Pieter Abbeel, and Sergey Levine (2018). “Latent space policies for hierarchical reinforcement learning” (cit. on p. 13).

Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, and Sergey Levine (2018). “Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor” (cit. on p. 23).

Harb, Jean, Pierre-Luc Bacon, Martin Klissarov, and Doina Precup (2018). “When waiting is not an option: Learning options with a deliberation cost”. *Thirty-Second AAAI Conference on Artificial Intelligence* (cit. on p. 17).

Harutyunyan, Anna, Will Dabney, Diana Borsa, et al. (2019). “The Termination Critic” (cit. on pp. 17, 26, 30, 31, 65).

He, Kaiming, Xiangyu Zhang, Shaoqing Ren, and Jian Sun (2015). “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification”. *Proceedings of the IEEE international conference on computer vision*, pp. 1026–1034 (cit. on p. 42).

Hessel, Matteo, Joseph Modayil, Hado Van Hasselt, et al. (2018). “Rainbow: Combining improvements in deep reinforcement learning”. *Thirty-Second AAAI Conference on Artificial Intelligence* (cit. on p. 6).

Houthooft, Rein, Xi Chen, Yan Duan, et al. (2016). “Vime: Variational information maximizing exploration”. *Advances in Neural Information Processing Systems*, pp. 1109–1117 (cit. on p. 18).

Jain, Arushi, Khimya Khetarpal, and Doina Precup (2018). “Safe Option-Critic: Learning Safety in the Option-Critic Architecture” (cit. on p. 17).

Jinnai, Yuu, Jee Won Park, David Abel, and George Konidaris (2019). “Discovering options for exploration by minimizing cover time” (cit. on p. 20).

Jinnai, Yuu, Jee Won Park, Marlos C Machado, and George Konidaris (2020). “Exploration in Reinforcement Learning with Deep Covering Options” (cit. on p. 20).

Klyubin, Alexander S, Daniel Polani, and Chrystopher L Nehaniv (2005). “Empowerment: A universal agent-centric measure of control”. *2005 IEEE Congress on Evolutionary Computation*. Vol. 1. IEEE, pp. 128–135 (cit. on p. 23).

Krishnan, Sanjay, Roy Fox, Ion Stoica, and Ken Goldberg (2017). “Ddco: Discovery of deep continuous options for robot learning from demonstrations” (cit. on p. 19).

Kulkarni, Tejas D, Karthik Narasimhan, Ardavan Saeedi, and Josh Tenenbaum (2016). “Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation”. *Advances in neural information processing systems*, pp. 3675–3683 (cit. on p. 14).

Levine, S., Chelsea Finn, T. Darrell, and P. Abbeel (2016). “End-to-End Training of Deep Visuomotor Policies”. *J. Mach. Learn. Res.* 17, 39:1–39:40 (cit. on p. 1).

Levy, Andrew, Robert Platt Jr., and Kate Saenko (2018). “Hierarchical Reinforcement Learning with Hindsight”. *CoRR* abs/1805.08180 (cit. on p. 13).

Levy, Andrew, George Konidaris, Robert Platt, and Kate Saenko (2017). “Learning multi-level hierarchies with hindsight” (cit. on p. 14).

Machado, Marios C, Marc G Bellemare, and Michael Bowling (2017). “A laplacian framework for option discovery in reinforcement learning”. *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 2295–2304 (cit. on p. 19).

Machado, Marlos C, Clemens Rosenbaum, Xiaoxiao Guo, et al. (2017). “Eigenoption discovery through the deep successor representation” (cit. on p. 20).

Mankowitz, Daniel J, Timothy A Mann, and Shie Mannor (2016). “Adaptive skills adaptive partitions (ASAP)”. *Advances in Neural Information Processing Systems*, pp. 1588–1596 (cit. on p. 20).

Mnih, Volodymyr, Adria Puigdomenech Badia, Mehdi Mirza, et al. (2016). “Asynchronous methods for deep reinforcement learning”. *International conference on machine learning*, pp. 1928–1937 (cit. on p. 9).

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, et al. (2013). “Playing atari with deep reinforcement learning”. *arXiv preprint arXiv:1312.5602* (cit. on pp. 5, 6).

Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Andrei A Rusu, et al. (2015). “Human-level control through deep reinforcement learning”. *Nature* 518.7540, pp. 529–533 (cit. on pp. 1, 5, 6).

Mohamed, Shakir and Danilo Jimenez Rezende (2015). “Variational information maximisation for intrinsically motivated reinforcement learning”. *Advances in neural information processing systems*, pp. 2125–2133 (cit. on p. 22).

Nachum, Ofir, Shixiang Shane Gu, Honglak Lee, and Sergey Levine (2018). “Data-efficient hierarchical reinforcement learning”. *Advances in Neural Information Processing Systems*, pp. 3303–3313 (cit. on pp. 13, 14).

Nachum, Ofir, Shixiang Gu, Honglak Lee, and Sergey Levine (2018). “Near-optimal representation learning for hierarchical reinforcement learning” (cit. on pp. 14, 15).

Nachum, Ofir, Haoran Tang, Xingyu Lu, et al. (2019). “Why Does Hierarchy (Sometimes) Work So Well in Reinforcement Learning?” (Cit. on p. 14).

Neal, R. (1992). “Connectionist Learning of Belief Networks”. *Artif. Intell.* 56, pp. 71–113 (cit. on p. 24).

Neal, Radford M (1990). “Learning stochastic feedforward networks”. *Department of Computer Science, University of Toronto* 64.9 (cit. on p. 24).

Pathak, Deepak, Pulkit Agrawal, Alexei A Efros, and Trevor Darrell (2017). “Curiosity-driven exploration by self-supervised prediction”. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 16–17 (cit. on p. 18).

Peng, Xue Bin, Michael Chang, Grace Zhang, Pieter Abbeel, and Sergey Levine (2019). “Mcp: Learning composable hierarchical control with multiplicative compositional policies”. *Advances in Neural Information Processing Systems*, pp. 3681–3692 (cit. on p. 13).

Pong, Vitchyr H, Murtaza Dalal, Steven Lin, et al. (2019). “Skew-fit: State-covering self-supervised reinforcement learning” (cit. on pp. 21, 24).

Puterman, Martin L (2014). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons (cit. on p. 30).

Co-Reyes, John, YuXuan Liu, Abhishek Gupta, et al. (2018). “Self-Consistent Trajectory Autoencoder: Hierarchical Reinforcement Learning with Trajectory Embeddings”. *Proceedings of the 35th International Conference on Machine Learning*. Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. Stockholmsmässan, Stockholm Sweden: PMLR, pp. 1009–1018 (cit. on pp. 1, 21, 24).

Riemer, Matthew, Miao Liu, and Gerald Tesauro (2018). “Learning abstract options”. *Advances in Neural Information Processing Systems*, pp. 10424–10434 (cit. on p. 15).

Savinov, Nikolay, Anton Raichuk, Raphaël Marinier, et al. (2018). “Episodic curiosity through reachability” (cit. on p. 18).

Schaul, Tom, John Quan, Ioannis Antonoglou, and David Silver (2015). “Prioritized experience replay”. *arXiv preprint arXiv:1511.05952* (cit. on p. 6).

Schrittwieser, Julian, Ioannis Antonoglou, Thomas Hubert, et al. (2019). “Mastering atari, go, chess and shogi by planning with a learned model”. *arXiv preprint arXiv:1911.08265* (cit. on p. 1).

Schulman, John, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz (2015). “Trust region policy optimization”. *International conference on machine learning*, pp. 1889–1897 (cit. on p. 9).

Schulman, John, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel (2015). “High-dimensional continuous control using generalized advantage estimation”. *arXiv preprint arXiv:1506.02438* (cit. on p. 9).

Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov (2017). “Proximal policy optimization algorithms”. *arXiv preprint arXiv:1707.06347* (cit. on pp. 9, 52).

Sharma, Archit, Shixiang Gu, Sergey Levine, Vikash Kumar, and Karol Hausman (2019). “Dynamics-aware unsupervised discovery of skills” (cit. on pp. 1, 24, 53).

Silver, David, Aja Huang, Chris J Maddison, et al. (2016). “Mastering the game of Go with deep neural networks and tree search”. *nature* 529.7587, p. 484 (cit. on pp. 1, 5).

Silver, David, Thomas Hubert, Julian Schrittwieser, et al. (2017). “Mastering chess and shogi by self-play with a general reinforcement learning algorithm”. *arXiv preprint arXiv:1712.01815* (cit. on p. 5).

Silver, David, Julian Schrittwieser, Karen Simonyan, et al. (2017). “Mastering the game of go without human knowledge”. *Nature* 550.7676, pp. 354–359 (cit. on p. 5).

Smith, Matthew, Herke Hoof, and Joelle Pineau (2018). “An inference-based policy gradient method for learning options”. *International Conference on Machine Learning*, pp. 4703–4712 (cit. on pp. 17, 26).

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction*. MIT press (cit. on pp. 7, 9, 35, 64).

Sutton, Richard S, David A McAllester, Satinder P Singh, and Yishay Mansour (2000). “Policy gradient methods for reinforcement learning with function approximation”. *Advances in neural information processing systems*, pp. 1057–1063 (cit. on pp. 7, 34).

Sutton, Richard S, Doina Precup, and Satinder Singh (1999). “Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning”. *Artificial intelligence* 112.1-2, pp. 181–211 (cit. on pp. 15, 26).

Tang, Charlie and Russ R Salakhutdinov (2013). “Learning stochastic feedforward neural networks”. *Advances in Neural Information Processing Systems*, pp. 530–538 (cit. on p. 24).

Thomas, Philip (2014). “Bias in natural actor-critic algorithms”. *International conference on machine learning*, pp. 441–448 (cit. on p. 32).

Van Hasselt, Hado, Arthur Guez, and David Silver (2016). “Deep reinforcement learning with double q-learning”. *Thirtieth AAAI conference on artificial intelligence* (cit. on p. 6).

Vezhnevets, Alexander Sasha, Simon Osindero, Tom Schaul, et al. (2017). "Feudal networks for hierarchical reinforcement learning". *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, pp. 3540–3549 (cit. on p. 14).

Vinyals, Oriol, Igor Babuschkin, Wojciech M Czarnecki, et al. (2019). "Grandmaster level in StarCraft II using multi-agent reinforcement learning". *Nature* 575.7782, pp. 350–354 (cit. on p. 1).

Wang, Ziyu, Tom Schaul, Matteo Hessel, et al. (2015). "Dueling network architectures for deep reinforcement learning". *arXiv preprint arXiv:1511.06581* (cit. on p. 6).

Warde-Farley, David, Tom Van de Wiele, Tejas Kulkarni, et al. (2018). "Unsupervised control through non-parametric discriminative rewards" (cit. on pp. 21, 24).

Williams, Ronald J (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". *Machine learning* 8.3-4, pp. 229–256 (cit. on pp. 8, 52).

Xu, Haitao, Brendan McCane, Lech Szymanski, and Craig Atkinson (2020). "MIME: Mutual Information Minimisation Exploration" (cit. on p. 18).

7 Appendix

7.1 Additional plots

In this section we show the plots we omitted from the main discussion for the sake of clarity of explanation.

7.1.1 Square environment: Option Length

These are the plots for option length and distance analogous to the one shown in figure 5.3 for the Line environment.

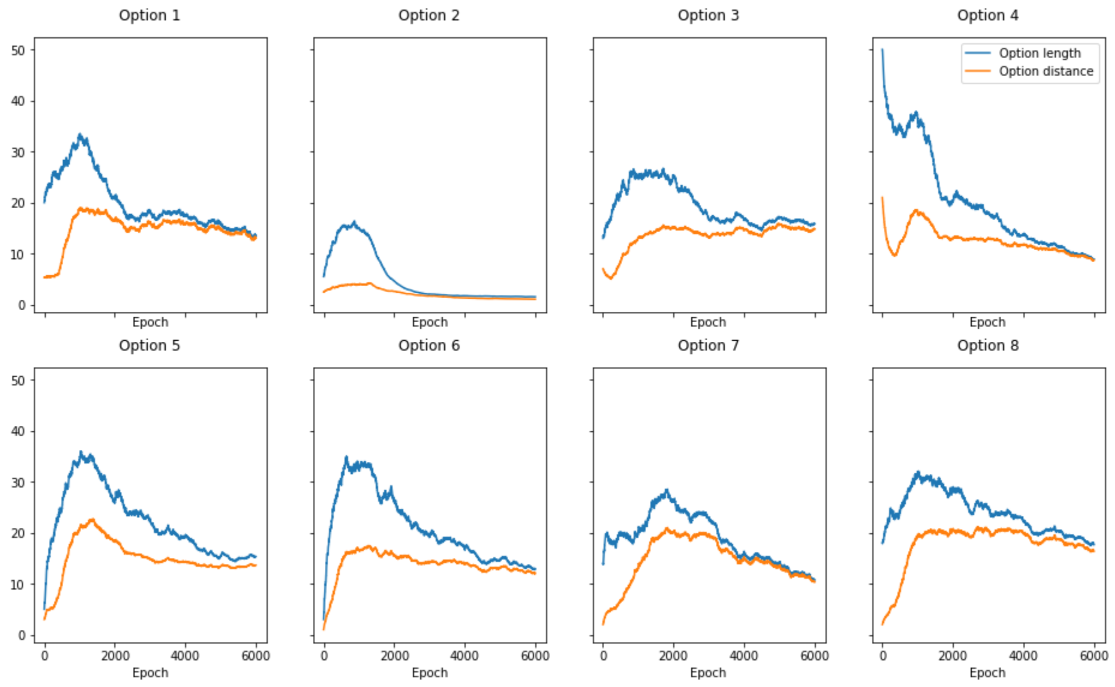
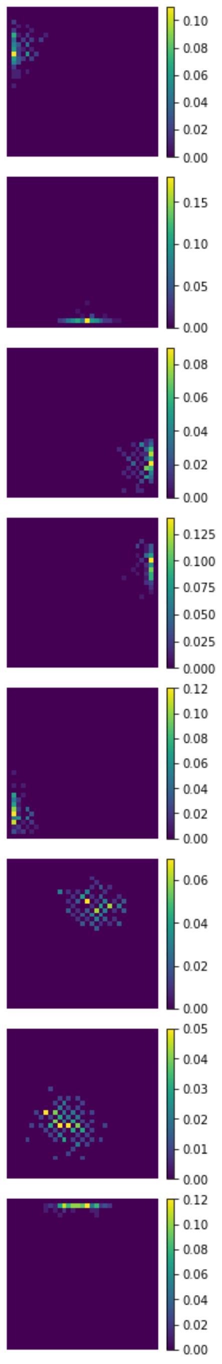


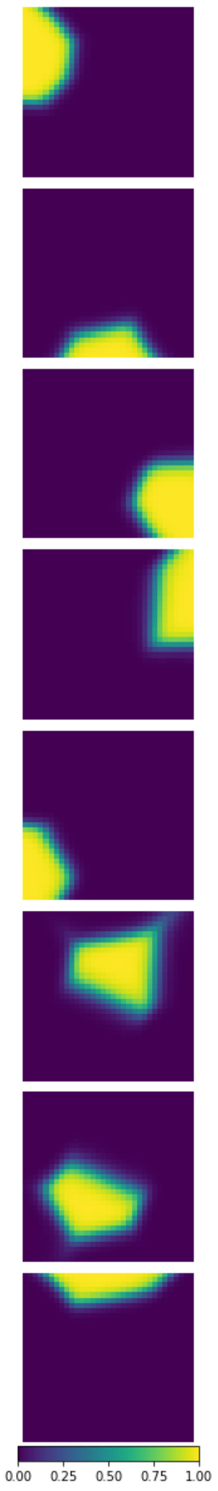
Fig. 7.1: Square environment: Length of options compared to the distance of the final state reached. Eventually, all options learn to terminate in the states they end in without any overhead of steps.

7.1.2 Square environment: Fixed length option plots

7.1.3 Square environment: option plots with termination function

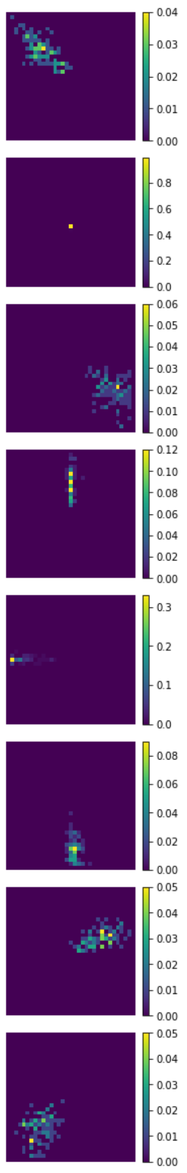


(a) $p(s|\omega)$ for all states in the Square environment. The values are obtained by sampling 100 episodes for each option

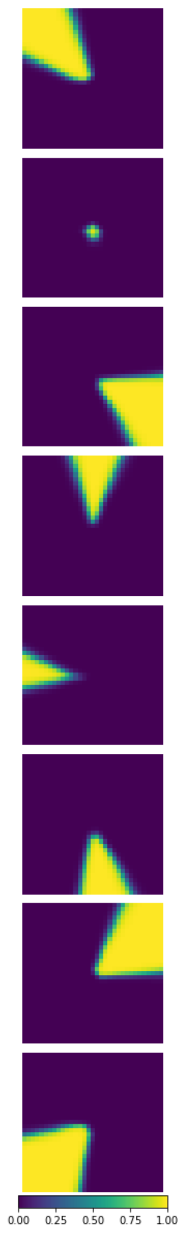


(b) Learned discriminator $q_\phi(\omega|s)$ for each state

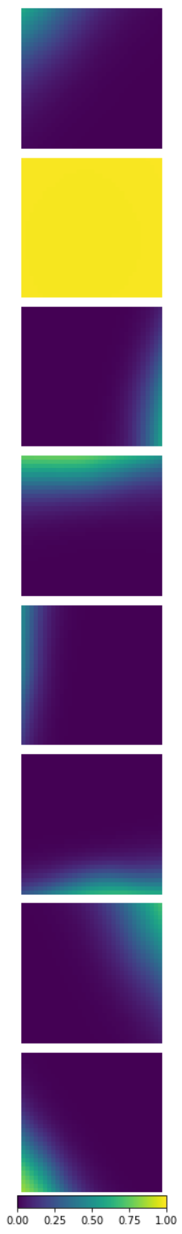
Fig. 7.2: Visual plots of the options learned in the Square environment, with options of fixed length 30



(a) $p(s|\omega)$ for all states in the Square environment. The values are obtained by sampling 100 episodes for each option



(b) Learned discriminator $q_\phi(\omega|s)$ for each state



(c) Termination probability $\beta(s)$ for each state

Fig. 7.3: Visual plots of the options learned in the Square environment, using termination function

7.2 Miscellaneous derivations

7.2.1 Policy update with termination function

Here we prove 4.31 in a similar way to the derivation of policy update in section 13.2 of Sutton and Barto, 2018.

Definition 13: Policy gradient with termination - derivation

$$
\begin{aligned}
&\nabla_\theta v_\pi(s) \\
&= \nabla_\theta \left[\sum_a \pi(a|s) q_\pi(s,a)\right] = \sum_a \nabla_\theta \pi(a|s) q_\pi(s,a) + \sum_a \pi(a|s) \nabla_\theta q_\pi(s,a) \\
&= \sum_a \nabla_\theta \pi(a|s) q_\pi(s,a) + \sum_a \pi(a|s) \nabla_\theta \sum_{s'} p(\bar{s}'|s,a) U(s') \\
&= \sum_a \nabla_\theta \pi(a|s) q_\pi(s,a) + \sum_a \pi(a|s) \nabla_\theta \sum_{s'} p(\bar{s}'|s,a) \Big[\beta(\bar{s}') r(s',a,s) \\
&\qquad + (1-\beta(\bar{s}'))\gamma V(s') \Big] \\
&= \sum_a \nabla_\theta \pi(a|s) q_\pi(s,a) + \sum_a \pi(a|s) \sum_{s'} p(\bar{s}'|s,a) \left[(1-\beta(\bar{s}'))\gamma \nabla_\theta v_\pi(s')\right] \\
&= \sum_a \nabla_\theta \pi(a|s) q_\pi(s,a) + \gamma \sum_{s'} P^{(1)}(s'|s) \left[\nabla_\theta v_\pi(s')\right] \\
&= \sum_a \nabla_\theta \pi(a|s) q_\pi(s,a) + \gamma \sum_{s'} P^{(1)}(s'|s) \Big[\sum_{a'} \nabla_\theta \pi(a'|s') q_\pi(s',a') + \\
&\qquad \gamma \sum_{s''} P^{(1)}(s''|s') \nabla_\theta v_\pi(s') \Big] \\
&= \sum_a \nabla_\theta \pi(a|s) q_\pi(s,a) + \gamma \sum_{s'} P^{(1)}(s'|s) \sum_{a'} \nabla_\theta \pi(a'|s') q_\pi(s',a') \\
&\qquad + \gamma^2 \sum_{s''} P^{(2)}(s''|s') \left[\nabla_\theta v_\pi(s')\right] \\
&= \dots \\
&= \sum_{s \in S} \sum_{k=0}^{\infty} \gamma^k P^{(k)}(s|s) \sum_a \nabla_\theta \pi(a|s) q_\pi(s,a) \\
&= \sum_s \eta_\gamma(s) \sum_a \nabla \pi(a|s) q_\pi(s,a)
\end{aligned}
$$

Where we have defined $\eta(s)_\gamma = \sum_{k=0}^{\infty} \gamma^k P^{(k)}(s'|s)$, which represents the discounted counts for the number of times state s is visited on average during an episode.

7.2.2 Decomposition of Transition probabilities

Here we show the derivation of the result in 4.53. The derivation has similarities with appendix A and B.1 of (Harutyunyan et al., 2019).

Definition 14: Decomposition of Transition probabilities

$$
\begin{aligned}
P\left(\bar{s}_f|\bar{s}_s\right) &= \beta\left(\bar{s}_f\right)\mathbb{I}_{\bar{s}_f=\bar{s}_s} + (1-\beta(\bar{s}_s))\sum_{\bar{s}} p^{\pi}(\bar{s}|s_s)\,P\left(\bar{s}_f|\bar{s}\right) && (7.1)\\
&= \beta\left(\bar{s}_f\right)\mathbb{I}_{\bar{s}_f=\bar{s}_s} + (1-\beta(\bar{s}_s))\sum_{\bar{s}} p^{\pi}(\bar{s}|s_s)\,P\left(\bar{s}_f|\bar{s}\right)\Big[&& (7.2)\\
&\qquad \beta\left(\bar{s}_f\right)\mathbb{I}_{\bar{s}_f=\bar{s}} + (1-\beta(\bar{s}))\sum_{\bar{s}'} p^{\pi}\left(\bar{s}'|s\right)P\left(\bar{s}_f|\bar{s}'\right)\Big] && (7.3)\\
&= \beta\left(\bar{s}_f\right)\mathbb{I}_{\bar{s}_f=\bar{s}_s} + (1-\beta(\bar{s}_s))\,p^{\pi}\left(\bar{s}_f|s_s\right)\beta\left(\bar{s}_f\right) && (7.4)\\
&\qquad + (1-\beta(\bar{s}_s))\sum_{\bar{s}} p^{\pi}(\bar{s}|s_s)(1-\beta(\bar{s}))\sum_{\bar{s}'} p^{\pi}\left(\bar{s}'|s\right)P\left(\bar{s}_f|\bar{s}'\right) && (7.5)\\
&= \ldots && (7.6)\\
&= \beta\left(\bar{s}_f\right)\left(P^{(0)}\left(\bar{s}_f|\bar{s}_s\right) + P^{(1)}\left(\bar{s}_f|\bar{s}_s\right) + \ldots\right) && (7.7)\\
&= \beta\left(\bar{s}_f\right)\sum_{k=0}^{\infty} P^{(k)}\left(\bar{s}_f|\bar{s}_s\right) && (7.8)
\end{aligned}
$$

7.2.3 Termination function gradient - Normal method derivation

Here we show how to derive 4.40. This derivation is very similar to the termination gradient derivations in appendix of Bacon et al. (2016).

Definition 15: Termination gradient - Normal method derivation

$$\nabla_\psi U(\bar{s}') \tag{7.9}$$

$$= \nabla_\psi \left[\beta(\bar{s}')r(\hat{s}') + (1-\beta(\bar{s}'))\gamma v(s')\right] \tag{7.10}$$

$$= \nabla_\psi(1-\beta(\bar{s}'))\gamma v(s') + \gamma(1-\beta(\bar{s}'))\nabla_\psi v(s') + \nabla_\psi \beta(\bar{s}')r(\hat{s}'|s,a) \tag{7.11}$$

$$= \nabla_\psi \beta(\bar{s}')(r(\hat{s}') - \gamma v(s')) + \gamma(1-\beta(\bar{s}'))\nabla_\psi\left[v(s')\right] \tag{7.12}$$

$$= \nabla_\psi \beta(\bar{s}')(r(\hat{s}') - \gamma v(s')) \tag{7.13}$$

$$+ \gamma(1-\beta(\bar{s}'))\nabla_\psi\Big[\sum_a \pi(a|s')[\sum_{s''} P(s''|s',a)U(s'')]\Big] \tag{7.14}$$

$$= \nabla_\psi \beta(\bar{s}')(r(\hat{s}') - \gamma v(s')) \tag{7.15}$$

$$+ \gamma(1-\beta(\bar{s}'))\sum_a \pi(a|s')\sum_{s''} P(s''|s',a)\nabla_\psi U(\bar{s}'') \tag{7.16}$$

$$= \nabla_\psi \beta(\bar{s}')(r(\hat{s}') - \gamma v(s')) + \gamma \sum_{s''} P^{(1)}(\bar{s}''|s')\nabla_\psi U(\bar{s}'') \tag{7.17}$$

$$= \nabla_\psi \beta(\bar{s}')(r(\hat{s}') - \gamma v(s')) + \gamma \sum_{s''} P^{(1)}(s''|s')\Big[\tag{7.18}$$

$$\nabla_\psi \beta(\bar{s}'')(r(s''|s',a) - \gamma v(s'')) + \gamma \sum_{s'''} P^{(1)}(\bar{s}'''|s'')\nabla_\psi U(\bar{s}''')\Big] \tag{7.19}$$

$$= \sum_{s''}\sum_{k=0}^{\infty} \gamma^k P^{(k)}(\bar{s}''|s')\nabla_\psi \beta(\bar{s}'')(r(\hat{s}') - \gamma v(s'') \tag{7.20}$$

$$= \sum_{s'}\sum_{k=0}^{\infty} \gamma^k P^{(k)}(\bar{s}'|s)\nabla_\psi \beta(\bar{s}')(r(\hat{s}') - \gamma v(s')) \tag{7.21}$$

$$= \sum_{s'} \eta_\gamma(s')\nabla_\psi \beta(\bar{s}')(r(\hat{s}') - \gamma v(s')) \tag{7.22}$$

$$\nabla_\psi U(s') = \sum_{s'} \eta(s')_\gamma \nabla_\psi \beta(\bar{s}')[r(\hat{s}') - \gamma V(s')] \tag{7.23}$$

7.2.4 Termination function gradient - Policy gradient derivation

Here we derive the result of 4.53. The derivation follows standard policy gradient derivations.

Definition 16: Termination gradient - Policy gradient derivation

$$\nabla\psi\mathbb{E}_{\tau\sim\pi_\theta,\beta\psi}[G(\tau)] = \mathbb{E}_{\tau\sim\pi_\theta,\beta\psi}\left[G(\tau)\cdot\sum_{t=1}^{T}\nabla\psi\log p_\beta(\bar{s}_t))\right] \tag{7.24}$$

$$= \mathbb{E}_{\tau\sim\pi_\theta,\beta\psi}\left[\left(\sum_{t=1}^{T} r_t\right)\cdot\nabla\psi\left(\sum_{t=1}^{T}\nabla\psi\log p_\beta(\bar{s}_t)\right)\right] \tag{7.25}$$

$$= \sum_{t=1}^{T}\mathbb{E}_{\tau_{1:t}}\mathbb{E}_{\tau_{t+1:T}}\left[r_t\left(\sum_{t'=1}^{t}\nabla\log p_\beta(\bar{s}_t) + \sum_{t'=t+1}^{T}\nabla\log p_\beta(\bar{s}_t)\right)|\tau_{1:T}\right] \tag{7.26}$$

$$= \sum_{t=1}^{T}\mathbb{E}_{\tau_{1:t}}\left[r_t\left(\sum_{t'=1}^{t}\nabla\log p_\beta(\bar{s}_t) + \underbrace{\mathbb{E}_{\tau_{t+1:T}}}_{t'}\sum_{t'=t+1}^{T}\nabla\log p_\beta(\bar{s}_t)|\tau_{1:t}\right]\right. \tag{7.27}$$

$$= \mathbb{E}_{\tau\sim\pi_\theta,\beta\psi}\left[\sum_{t=1}^{T} r_t\sum_{t'=1}^{t}\nabla\log p_\beta(\bar{s}_t)\right] \tag{7.28}$$

By rearranging terms in the two summations, we have

$$\mathbb{E}_{\tau\sim\pi_\theta,\beta\psi}\left[\sum_{t=1}^{T} r_t\sum_{t'=1}^{t}\nabla\log p_\beta(\bar{s}_t)\right] = \mathbb{E}_{\tau\sim\pi_\theta,\beta\psi}\left[\sum_{t'=1}^{T}\nabla\log p_\beta(\bar{s}_t)\sum_{t=t'}^{T} r_t\right] \tag{7.29}$$

$$= \mathbb{E}_{\tau\sim\pi_\theta,\beta\psi}\left[\sum_{t'=1}^{T}\nabla\log p_\beta(\bar{s}_t)G_t\right] \tag{7.30}$$

List of Definitions

List of Algorithms

List of Figures

List of Tables