THE SKILL-ACTION ARCHITECTURE: LEARNING ABSTRACT ACTION EMBEDDINGS FOR REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

The option framework, one of the most promising Hierarchical Reinforcement Learning (HRL) frameworks, is developed based on the Semi-Markov Decision Problem (SMDP) and employs a triple formulation of the option (i.e., an action policy, a termination probability, and an initiation set). These design choices, however, mean that the option framework: 1) has low sample efficiency, 2) cannot use more stable Markov Decision Problem (MDP) based learning algorithms, 3) represents abstract actions implicitly, and 4) is expensive to scale up. To overcome these problems, here we propose a simple yet effective MDP implementation of the option framework: the Skill-Action (SA) architecture. Derived from a novel discovery that the SMDP option framework has an MDP equivalence, SA hierarchically extracts skills (abstract actions) from primary actions and explicitly encodes these knowledge into skill context vectors (embedding vectors). Although SA is MDP formulated, skills can still be temporally extended by applying the attention mechanism to skill context vectors. Unlike the option framework, which requires M action policies for M skills, SA's action policy only needs one decoder to decode skill context vectors into primary actions. Under this formulation, SA can be optimized with any MDP based policy gradient algorithm. Moreover, it is sample efficient, cheap to scale up, and theoretically proven to have lower variance. Our empirical studies on challenging infinite horizon robot simulation environments demonstrate that SA not only outperforms all baselines by a large margin, but also exhibits smaller variance, faster convergence, and good interpretability. On transfer learning tasks, SA also outperforms the other models and shows its advantage on reusing knowledge across tasks. A potential impact of SA is to pave the way for a large scale pre-training architecture in the reinforcement learning area.

1 INTRODUCTION

Reinforcement Learning (RL) is a paradigm for imitating human's trial-and-error learning process: RL trains an agent to maximise rewards by taking actions in and receiving feedback from an environment. RL has achieved human-level performance in playing video and board environments (Mnih et al., 2015; Silver et al., 2016). However, while humans can abstract the hierarchical complexity of actions through interactions with the environment and make decisions at both macro and micro time scales, conventional RL agents have limited abilities to solve complex tasks (Daniel et al., 2016): they only learn the most primitive actions and makes decisions at the smallest time scale. Hierarchical Reinforcement Learning (HRL) attempts to resolve this gap between humans and RL by decomposing complex tasks into a hierarchy of abstracted actions at multiple time scales.

An HRL agent typically learns abstractions of actions on two levels: skills and primary actions. Skills are higher-level abstracted actions. Their executions are temporally extended to a variable amount of time. Primary actions are lower-level actions defined by the environment. They are executed at every time step. For example, for a humanoid robot, walking and jumping are two abstract skills, while movements of each joint are primary actions. One of the most promising HRL frameworks is the option framework (Sutton et al., 1999). The option framework has achieved great success in representing actions at different time scales (Bacon et al., 2017), speeding and scaling up learning (Bacon, 2018), improving exploration (Harb et al., 2018) and and facilitating transfer learning (Zhang & Whiteson, 2019).

In the option framework, an option is a primary action level sub-policy consisting of an action policy, a termination probability, and an initiation set. A master policy (Zhang & Whiteson, 2019) (aka. policy-over-options (Sutton et al., 1999)) is used to compose those options and thus is a skill-level policy. The option framework is formulated as a Semi-Markov Decision Problem (SMDP) (Puterman, 1994): an option sampled from a master policy is executed through a variable amount of time (until its termination function determines to stop). As highlighted in the literature, the SMDP formulation has the following limitations:

1. Sample inefficiency (Zhang & Whiteson, 2019): a) For policy gradient based algorithms, the master policy cannot be updated until stop. As a result, one update consumes various time steps of samples. b) At each time step, only one (the executed) option's policies can be updated.
2. Large variance (Bacon, 2018; Zhang & Whiteson, 2019; Haarnoja et al., 2018): SMDP algorithms are notoriously hyperparameters sensitive. Due to the SMDP formulation, more stable Markov Decision Process (MDP) policy gradient algorithms cannot be used.
3. Expensive to scale up (Riemer et al., 2018): for M options, there are $2M$ action and termination policies. Each policy is a neural network that could have millions of parameters to train.

To address these problems, we propose a simple yet effective MDP implementation of the option framework, the Skill-Action (SA) architecture. The idea behind SA originates from a new discovery that the SMDP option framework has an MDP equivalence, which is achieved by adding extra dependencies into the master policy. However, those extra dependencies still prevent the master policy from being updated at every time step. Based on this equivalence, a "skill policy" which marginalizes those dependencies away is derived and hence can be updated at each time step.

In SA, knowledge of a skill is explicitly represented as a skill context vector (similar to an embedding vector (Vaswani et al., 2017) in Natural Language Processing (NLP) or capsule (Sabour et al., 2017) in Computer Vision (CV)): each dimension encodes a particular property of the skill[1]. The skill policy is similar to a compatibility function: it is used to replace the master policy and termination function while improving their functionalities by employing the attention mechanism (Vaswani et al., 2017). At each time step, the skill policy measures the compatibility (suitability) of all skills with the current state and the executed skill from the last step. If the previous skill still fits the current situation, then the skill policy tends to continue with it; otherwise, a new skill with better compatibility will be sampled. Unlike the option framework, which requires M action policies for M skills, SA's action policy only needs one decoder to decode any skill context vector into primary actions. With this formulation, the entire framework is MDP-based while the skill can still be temporally extended, and its scalability, as well as stability, are significantly improved. All of these design choices have precursors in the existing literature (HRL (Sutton et al., 1999; Bacon, 2018; Zhang & Whiteson, 2019); CV (Kosiorek et al., 2019); NLP (Vaswani et al., 2017)). Our contribution is establishing them in reinforcement learning settings.

Compared to the SMDP option framework, SA has following advantages:

1. Sample efficiency: a) SA is MDP formulated, thus sample at each time step can be used to update the skill policy. b) Only one action policy decoder is needed. It learns to decode each dimension of the skill context vector at each time step whichever skill is activated.
2. Small variance: a) The skill value upon arrival function (Eq. (5)) is theoretically and empirically proven to have smaller variance than the conventional value function. b) SA only needs to train two (skill and action) policy networks. c) SA can employ more stable MDP based policy gradient algorithms (e.g. PPO (Schulman et al., 2017)).
3. Scalability: a) Regardless of the number of skills, only two policies needs to be trained. b) Adding one more skill is as cheap as adding a context vector.
4. Transfer Learning: SA outperforms the other models in 5 out of 6 environments and shows its advantage on reusing knowledge across tasks.

[1]For example, the first dimension may encode the orientation of a primary action. A jump skill context vector may have a large first dimension value which instructs the robot to emit primary actions vertically. A walk skill may have a small value and emit actions horizontally.

5. Improved interpretability and exploration: a) Unlike the option framework encodes abstract knowledge implicitly in action policies, knowledge of a skill is explicitly encoded in each dimension of the skill context vector. b) As shown in experiments, SA has better exploration than the other option formulations.

2 Related Works

To discover options automatically, Sutton et al. (1999) proposed Intra-option Q-learning to update the master Q value function at every time step. However, all policies under this formulation are approximated implicitly using the Q-learning method. Levy & Shimkin (2011) proposed to unify the Semi-Markov process into an augmented Markov process and explicitly learn an "overall policy" by applying MDP-based policy gradient algorithms. However, their method for updating the master policy is still SMDP-style thus sample inefficient. Bacon et al. (2017) proposed a policy gradient based Option Critic (OC) framework for explicitly learning intra-option policies and termination functions in an intra-option manner. However, for the master policy's policy gradient learning, OC still remains SMDP-style. Klissarov et al. (2017) attempted to combine OC with PPO in an intra-option learning manner (PPOC). However, as we show in Appendix A.4.2, due to the SMDP formulation, gradients they use for updating master policy are inconsistent. Zhang & Whiteson (2019) reformulated the option framework into two augmented MDPs. Under this formulation all policies can be modeled explicitly and learned in MDP-style. However, their model is still expensive to scale up. On single task environments, DAC has no significant advantages over other baselines.

We must appreciate that Bacon ((Bacon, 2018); Chapter 3.6) conceptually discussed a vectorized option representation and directly approximated the marginalized master policy. However, no concrete formulations and policy gradients theorems were developed in their work. Daniel et al. (2016) proposed an MDP-formulated PGM similar to ours in Appendix A.4. However, unlike in this work, they did not prove the equivalence between the MDP-formulation and SMDP by employing conditional independencies. Furthermore, their learning algorithm is EM based while ours is policy gradient based. Our work is motivated by capsule networks Kosiorek et al. (2019) (more details in Appendix A.1) and is developed independently from literatures above.

With respect to optimization, Zhang & Whiteson (2019) pointed out that a large margin of performance boost of DAC comes from Proximal Policy Optimization (Schulman et al., 2017) (PPO). Since SA is MDP-based, it can be optimized directly with the PPO objective. Recent works show that off-policy methods have large advantages over on-policy methods on sample efficiency and performance(Haarnoja et al., 2018; Wulfmeier et al., 2020). SA is a general purpose framework and also compatible with off-policy algorithms. For a fair comparison with previous works (Zhang & Whiteson, 2019) we only employs on-policy algorithms. Learning with off-policy algorithms remains for future work.

This paper devises SA as a candidate for a general large-scale pre-training framework in reinforcement learning. Main efforts are spent on proving the equivalence between MDP and SMDP, extending the conventional value function to "skill value upon arrival function", and deriving policy gradient theorems. Our extensive experiments show that SA achieves significant sample efficiency improvements on infinite horizon single task environments and shows obvious performance advantages on transfer learning tasks. Although SA empirically shares two innate limitations with the conventional option framework (Levy & Shimkin, 2011; Klissarov et al., 2017; Smith et al., 2018; Harb et al., 2018; Zhang & Whiteson, 2019): (1) failure to improve the performance and the sample efficiency on finite horizon environments (section 4.1); (2) "the dominant skill problem" (Zhang & Whiteson, 2019) (section 4.2), in Appendix A.1 we conceptually show that SA-style wide (higher-order dependencies) value functions could be a solution to both limitations. This is mainly because these limitations are caused by the insufficiency of the conventional value functions in approximating values have temporal latent variables dependencies. Please refer to Appendix A.1 for detailed explanation.

3 The Skill-Action Architecture

In this section, we propose a simple MDP (Puterman, 1994) implementation of the option framework: the Skill-Action (SA) architecture. To overcome limitations of the SMDP option framework,

we first prove an MDP equivalence to the SMDP. Briefly, we propose a novel MDP "mixture master policy" (Appendix A.4.2). Unlike the conventional SMDP master policy only depends on the current state, the mixture master policy has extra dependencies on the termination function and the previously activated option. We then prove that the MDP has identical optimal properties with the SMDP option framework (Sutton et al., 1999) and identical policy gradients with the option-critic architecture (Bacon et al., 2017). Due to page limitations, we provide detailed theorems and proofs in Appendix A.4.

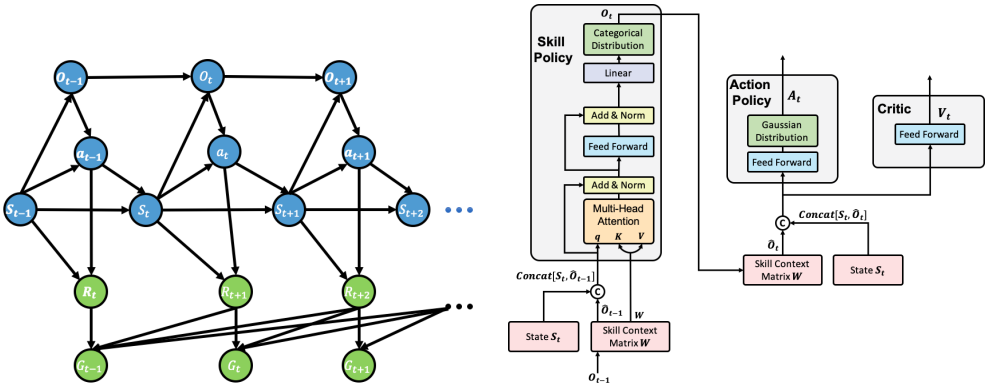
Theorem 3.1. *The SMDP formulated option framework, which employs Markovian options, has an underlying MDP equivalence.*

Proposition 3.2. *The MDP formulation has identical value functions with the SMDP option framework (Sutton et al., 1999).*

Proposition 3.3. *The MDP formulation has identical policy gradients with the option-critic architecture (Bacon et al., 2017).*

Although the mixture master policy (Eq. 18) is MDP formulated, the master policy's (Eq. 17) gradients are still blocked by its dependency on the termination function. To overcome this, we present a marginalized derivation of the equivalence: the Skill-Action (SA) architecture. SA marginalizes the termination function away and models the marginalized policy (Eq. 21) directly with a "skill policy" (Eq. 2), which is used to replace both of the master policy and termination function while implements their functionalities with the attention mechanism (Vaswani et al., 2017). Section 3.1 describes the dynamics (Markov process) of SA. Section 3.2 defines value functions on top of the dynamics, thus formulates the MDP. Policy gradient theorems are then derived. Section 3.3 implements SA by employing neural networks and the Multi-Head Attention mechanism (Vaswani et al., 2017), which enables SA to temporally extend skills in the absence of the termination function.

3.1 Dynamics of the Skill-Action Architecture



(a) PGM of SA (b) Network Architecture of SA

Figure 1: The Skill-Action (SA) Architecture

In this section, we define the dynamics (Markov process) of SA. We first introduce MDP notations. A Markov decision process $M = \{\mathbb{S}, \mathbb{A}, r, P, \gamma\}$ consists of a state space $\mathbb{S}$, an action space $\mathbb{A}$, a state transition function $P(\mathbf{s}_{t+1}|\mathbf{s}_t) : \mathbb{S} \to \mathbb{S}$, a reward function $r(\mathbf{s}, \mathbf{a}) : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$, and a discount factor $\gamma \in \mathbb{R}$. A policy $\pi = P(\mathbf{a}|\mathbf{s}) : \mathbb{S} \to \mathbb{A}$ is a probability distribution defined over actions conditioning on states. An expected discounted return is defined as $G_t = \sum_k^N \gamma^k R_{t+k+1}$, where $R \in \mathbb{R}$ is the actual reward received from the environment. The value function $V[\mathbf{s}_t] = \mathbb{E}_\pi[G_t|\mathbf{s}_t]$ is the expected return starting at state $\mathbf{s}_t$ and following policy π thereafter. The action-value function is defined as $Q[\mathbf{s}_t, \mathbf{a}_t] = \mathbb{E}_{\tau_\pi}[G_t|\mathbf{s}_t, \mathbf{a}_t]$. An Markov decision process together with value functions defined on it are referred to as an MDP (Puterman, 1994).

Having defined notations of MDP, we propose the dynamics of SA. Specifically, a **skill index vector** $\mathbf{o} \in \mathbb{Z}_2^M$ is an M-dimensional one-hot vector, where M denotes the total number of skills to learn. Each entry $\mathrm{o} \in \{0, 1\}$ is a binary random variable. $\mathrm{o}_i = 1$ means that the i-th skill is activated. A **skill context matrix** (Kosiorek et al., 2019) $\boldsymbol{W}_S \in \mathbb{R}^{M \times E}$ is a learnable parameter containing M

rows of E dimensional real vectors, the i-th row of $\boldsymbol{W}_S$ corresponds to the i-th skill o_i, and different columns encode different properties of a skill. A **skill context vector** $\hat{\mathbf{o}}_t$ is defined as:

$$\hat{\mathbf{o}}_t = \boldsymbol{W}_S^T \cdot \mathbf{o}_t,\ \hat{\mathbf{o}}_t \in \mathbb{R}^E. \tag{1}$$

The **skill policy** is defined as:

$$P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}; \boldsymbol{W}_s) : \mathbb{S} \times \mathbb{R}^E \to \mathbb{R}^E, \tag{2}$$

which is a probability distribution over skill context vector $\hat{\mathbf{o}}_t$ conditioned on state $\mathbf{s}_t$ and previous sampled skill context vector $\hat{\mathbf{o}}_{t-1}$, with $\boldsymbol{W}_S$ as its learnable parameters.

The **action policy** is defined as:

$$P(\mathbf{a}_t|\mathbf{s}_t, \hat{\mathbf{o}}_t) : \mathbb{S} \times \mathbb{R}^E \to \mathbb{A}, \tag{3}$$

which is a probability distribution over the action random variable $\mathbf{a}_t \in \mathbb{A}$ conditioned on the skill context vector $\hat{\mathbf{o}}_t$ and state $\mathbf{s}_t$, and decodes them into primary actions.

With both skill and action policies in hand, the dynamics of the SA are defined as a Probabilistic Graphical Model (PGM) (Koller & Friedman, 2009) (Figure 1 (a)):

$$\begin{aligned} P(\tau) =& P(\mathbf{s}_0)P(\hat{\mathbf{o}}_0)P(\mathbf{a}_0|\mathbf{s}_0, \hat{\mathbf{o}}_0) \\ & \prod_{t=1}^{\infty} P(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_{t-1})P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})P(\mathbf{a}_t|\mathbf{s}_t, \hat{\mathbf{o}}_t), \end{aligned} \tag{4}$$

where $P(\tau) = P(\mathbf{s}_0, \hat{\mathbf{o}}_0, \mathbf{a}_0, \mathbf{s}_1, \hat{\mathbf{o}}_1, \mathbf{a}_1, \ldots)$ denotes the joint distribution of the PGM. Note that under this formulation, $P(\tau)$ is actually an Hidden Markov Model (HMM) with $\mathbf{s}_t$, $\mathbf{a}_t$ as observable random variables and $\hat{\mathbf{o}}_t$ as latent variables.

3.2 MDP OF THE SKILL-ACTION ARCHITECTURE

With SA's dynamics in hand, in this section, we first propose a novel "skill value upon arrival function" and theoretically prove that it has a smaller variance than the conventional value function. This property is empirically justified in Section 4.1 and further discussed in Appendix A.1. Then, we derive the recursive formulation of value functions and formulate the MDP. Based on the MDP, skill and action policies' gradients theorems are finally derived.

Rather than use the conventional value function $V[\mathbf{s}_t]$, we define the **skill value upon arrival function** $V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]$ (derivations in Appendix A.5) as:

$$V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}] = \mathbb{E}[G_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}] = \sum_{\hat{\mathbf{o}}_t} P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t]. \tag{5}$$

Proposition 3.4. *$V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]$ is an unbiased estimation of $V[\mathbf{s}_t]$.*

Proposition 3.5. *The variance of $V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]$ is less than or equal to $V[\mathbf{s}_t]$.*

Proof. See Appendix A.5 □

Eq. (5) states that the skill value function upon arrival is an expectation over skill value function $Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t]$ conditioned on previous skill $\hat{\mathbf{o}}_{t-1}$. The **skill value function** $Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t]$ is defined as:

$$Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t] = \mathbb{E}[G_t|\mathbf{s}_t, \hat{\mathbf{o}}_t] = \sum_{\mathbf{a}_t} P(\mathbf{a}_t|\mathbf{s}_t, \hat{\mathbf{o}}_t)Q_A[\mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t], \tag{6}$$

and the **skill-action value function** $Q_A[\mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t]$ is defined as (derivations in Appendix A.5):

$$Q_A[\mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t] = \mathbb{E}[G_t|\mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t] = r(s, a) + \gamma \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)V[\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t], \tag{7}$$

where $\gamma \in \mathbb{R}$ is a discounting factor. Expanding (Eq. 7) with (Eq. 5) gives us a recursion formulation from which Bellman equations and policy gradient theorems are derived. To keep notations uncluttered, we use θ_o to denote skill policy's parameters (Eq. 2) and θ_a to denote action policy's parameters (Eq. 3). The skill and action policies' gradient theorems are:

Theorem 3.6. ***Skill Policy Gradient Theorem:*** *Given a stochastic skill policy differentiable in its parameter vector θ_o, the gradient of the expected discounted return with respect to θ_o is:*

$$\frac{\partial V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]}{\partial \theta_o} = \mathbb{E}[\ \frac{\partial P(\hat{\mathbf{o}}'|\mathbf{s}', \hat{\mathbf{o}})}{\partial \theta_o} Q_O[\mathbf{s}', \hat{\mathbf{o}}'] \mid \mathbf{s}_t, \hat{\mathbf{o}}_{t-1}], \tag{8}$$

where $\hat{\mathbf{o}}'$ is one time step later than $\hat{\mathbf{o}}$.

Theorem 3.7. ***Action Policy Gradient Theorem:*** *Given a stochastic action policy differentiable in its parameter vector θ_a, the gradient of the expected discounted return with respect to θ_a is:*

$$\frac{\partial Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t]}{\partial \theta_a} = \mathbb{E}[\ \frac{\partial P(\mathbf{a}|\mathbf{s}, \hat{\mathbf{o}})}{\partial \theta_a} Q_A[\mathbf{s}, \hat{\mathbf{o}}, \mathbf{a}] \mid \mathbf{s}_t, \hat{\mathbf{o}}_t]. \tag{9}$$

Proof. See Appendix A.6.2 □

Compared to MDP formulated algorithms, SMDP option frameworks are sample inefficient and notoriously unstable to hyperparameters (Zhang & Whiteson, 2019). The Skill and Action policies' gradient theorems enable SA to be compatible with any MDP policy gradient algorithms, and thus has much better stability and convergence. Given the great success of PPO (Schulman et al., 2017), in this paper we directly apply it to our learning algorithm (Algorithm 1 in Appendix A.7).

3.3 Networks Architecture

After deriving the MDP of SA, we present a simple neural network implementation of the Skill-Action Architecture (Figure 1). Unlike the conventional SMDP option framework, which employs a termination function and an SMDP master policy to temporally extend the execution of an option, SA implements the temporal extension functionality by employing the Multi-Head Attention (MHA) mechanism (Vaswani et al., 2017) (due to page limitations, we briefly explain MHA in Appendix A.9). At each time step, the skill policy $P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}; \boldsymbol{W}_s)$ attends to (measures the compatibility of) all skill context vectors in $\boldsymbol{W}_s$ according to $\mathbf{s}_t$ and $\hat{\mathbf{o}}_{t-1}$. If $\hat{\mathbf{o}}_{t-1}$ still fits $\mathbf{s}_t$, then the skill policy assigns a larger attention weight to $\hat{\mathbf{o}}_{t-1}$, thus has a tendency to continue with it. Otherwise, a new skill with better compatibility will be sampled. The action policy is as simple as one decoder to decode $\hat{\mathbf{o}}_t$ and $\mathbf{s}_t$ into primary actions $\mathbf{a}_t$. The attention mechanism together with skill context vectors enable SA to temporally extend skills even in the absence of termination functions.

Specifically, a **skill policy** (Eq. (2)) uses a concatenation of current state $\mathbf{s}_t$ and previous skill context vector $\hat{\mathbf{o}}_{t-1}$ as the query for MHA. Both key and value matrices are the skill context matrix $\boldsymbol{W}_S$. In this way, we have:

$$\hat{\mathbf{s}}_{t-1} = linear(\text{Concat}[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]), \tag{10}$$

$$\mathbf{d}_t^O = \text{MHA}(\hat{\mathbf{s}}_{t-1}, \boldsymbol{W}_S, \boldsymbol{W}_S), \tag{11}$$

$$\mathbf{o}_t \sim Categorical(\mathbf{o}_t|\mathbf{d}_t^O), \tag{12}$$

where the $linear$ layer simply projects the concatenated vector to E dimension. MHA is employed to attend to (measures the compatibility of) all skills in $\boldsymbol{W}_s$ according to $\mathbf{s}_t$ and $\hat{\mathbf{o}}_{t-1}$. The **skill density vector** $\mathbf{d}_t^O$ is then used as densities for a Categorical distribution $P(\mathbf{o}_t|\mathbf{d}_t^O)$, from which the new one-hot **skill index vector** $\mathbf{o}_t$ is sampled from. We can retrieve the **skill context vector** by $\hat{\mathbf{o}}_t = \boldsymbol{W}_S^T \cdot \mathbf{o}_t$. With the skill context vector $\hat{\mathbf{o}}_t$ in hand, the **action policy** can be designed as simple as a multi-layers Feed-Forward Networks (FFN) decoder:

$$\mathbf{d}_t^A = \text{FFN}(\mathbf{s}_t, \hat{\mathbf{o}}_t), \tag{13}$$

$$\mathbf{a}_t \sim P(\mathbf{a}_t|\mathbf{d}_t^A), \tag{14}$$

where $\mathbf{d}_t^A$ is a density vector and P is an arbitrary probability distribution (works for both discrete and continuous situations).

Similar to Zhang & Whiteson (2019), because of the **skill value upon arrival function** $V(\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})$, (Eq. 5) is an expectation of the **skill value function** $Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t]$ (Eq. 6). It is sufficient for us to model only one critic function:

$$Q_O = \text{FFN}(\mathbf{s}_t, \hat{\mathbf{o}}_t), \tag{15}$$

where Q_O is implemented as a multi-layer FFN. We summarize the detailed learning procedures in Algorithm 1 in Appendix A.7.

Since $\hat{\mathbf{o}}_t$ encodes all context of a skill, SA only needs one action policy decoder to decode the activated skill context vectors $\hat{\mathbf{o}}_t$ and current state $\mathbf{s}_t$ into primary actions $\mathbf{a}_t$. This design choice largely improves the scalability of SA: adding one more skill is as cheap as adding a skill context vector. Moreover, unlike the option framework, in which only the activated option's action policy gets updated, the action policy learns to decode each dimension of skill context vectors at every time step. This design choice largely improves sample efficiency and enables SA to converge faster than the conventional option framework.

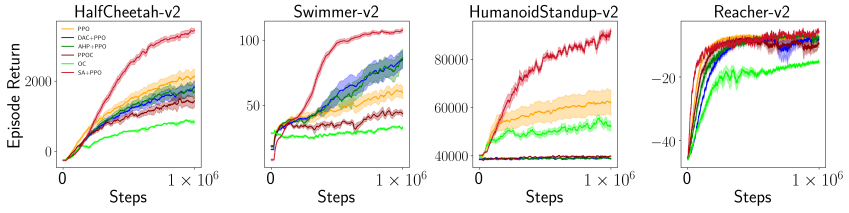
4 EXPERIMENTS

In this section, we design experiments to answer three questions: 1) Can SA outperform other baselines (regarding episodic returns, stability, and scalability)? 2) Can SA temporally extend skills without the termination function? 3) Can skill context vectors be easily interpreted?

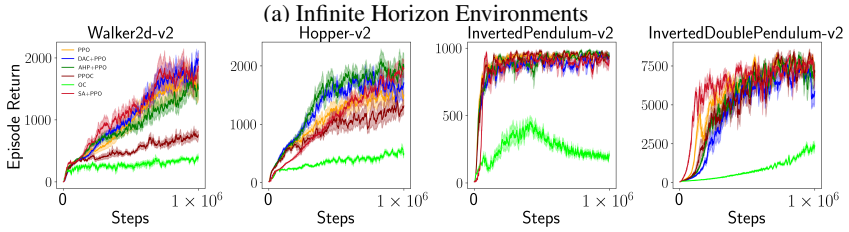
Experiments are conducted on all OpenAI Gym MuJoCo environments (10 environments) (Brockman et al., 2016). We follow DAC (Zhang & Whiteson, 2019) and compare our algorithm with five baselines, four of which are option implementations, i.e., DAC+PPO (Zhang & Whiteson, 2019), AHP+PPO (Levy & Shimkin, 2011), PPOC (Klissarov et al., 2017) and OC (Bacon et al., 2017). The last baseline is PPO (Schulman et al., 2017). All baselines' parameters used in DAC[2] remain unchanged. The only difference is the maximum number of training steps: SA only needs 1 million steps to converge rather than the 2 million used in DAC. For a fair comparison, we use four skills for SA and four options for other option implementations. All experiments are run on an Intel Core i9-9900X CPU @ 3.50GHz with a single thread and process. Our implementation details are summarized in Appendix A.8.

4.1 PERFORMANCE

In Figure 2, we report episodic returns on infinite horizon and finite horizon[3] environments separately. For a fair comparison, we use exactly the same plotting script as used in DAC: curves are averaged over 10 independent runs and smoothed by a sliding window of size 20. Shaded regions indicate standard deviations. Performance of all ten environments is shown in Appendix (Figure 6).



(a) Infinite Horizon Environments



(b) Finite Horizon Environments

Figure 2: Episodic Returns. X-axis is time step. Y-axis is Episodic Return

It is extremely interesting that SA shows two completely different kinds of behaviors on infinite and finite horizon environments. According to previous option framework implementations (Klissarov

[2] All implementations are from DAC's open source repo https://github.com/ShangtongZhang/DeepRL/tree/DAC. Note that the author list of this paper does not have any overlap with DAC. We have open sourced our implementation in supplementary materials.

[3] We refer environments with the environment-over condition to finite horizon environments, and infinite vice versa.

et al., 2017; Smith et al., 2018; Harb et al., 2018; Zhang & Whiteson, 2019), on single task environments, option-based algorithms do not have a distinguishable performance boost over hierarchy-free algorithms. SA also has similar behavior and achieves comparable performance to the best baseline algorithm on most finite horizon environments, as shown in Figure 2 (b). We conjecture that finite horizon environments contain less significant temporal relationships than infinite horizon environments (Appendix A.1).

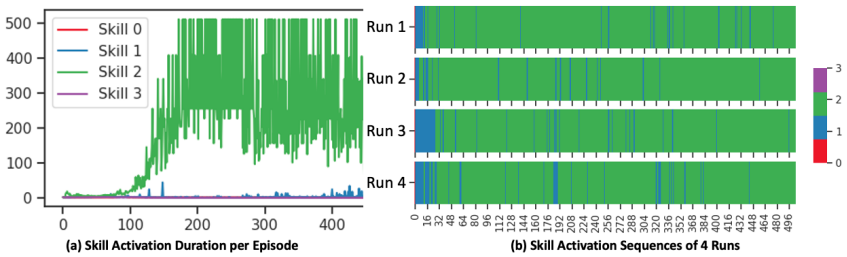


Figure 3: Skill Duration Patterns

On infinite horizon environments as shown in Figure 2 (a), SA's performance significantly outperforms all baselines by a large margin in various aspects. For episodic return, e.g., HumanoidStandup, all option implementations barely converge, while SA is 240% better than DAC and AHP[4]. For convergence, SA has the fastest convergence speed. On the first two environments, which are also reported in DAC, SA only takes 40% of time steps of DAC and AHP to reach similar episodic returns. This acceleration is because: 1) SA is MDP formulated, the skill policy is updated at each time step; 2) SA only has one action policy decoder; 3) the action decoder learns to decode skill context vectors whichever skill is activated. For stability, all 10 runs of SA converges to a similar level while the other have much larger standard deviations. This property is theoretically justified by Proposition 3.5 and further discussed in Appendix A.1.

4.2 Temporal Extension

It is logical to ask whether SA is capable of temporal extension without the termination function. To illustrate this, we plot the average duration of each skill during training episodes of the HalfCheetah environment in Figure 3 (a) (more details are provided in Appendix A.3.2). At the start of training, all skills' durations are short, while Skill 2's duration quickly grows and dominates the entire episode during later stages. This growth of duration proves that SA can still temporally extend a skill. Moreover, towards the end of the training, the dominant skill's [5] duration starts to decrease while the duration of a secondary skill (Skill 1) starts to increase. This means that during later training stages, SA starts to coordinate different skills. To better explain how SA coordinates skills, we provide a visualization of skill activation sequences in Figure 3 (b).

4.3 Interpretation of Skill Context Vectors

Explicit skill representations not only improve efficiency, scalability, and generalization, but also benefit interpretation. We continue with the HalfCheetah example and demonstrate how easily skill context matrix $\boldsymbol{W}_s$ (Figure 4 (a)) can be interpreted (more details are provided in Appendix A.3.3). We first follow Sabour et al. (2017) and interpret what property is represented by a context vector's dimension by adding perturbations on to it, and inspecting perturbations' affections on the action policy decoder of generating primary actions $\mathbf{a}$ (Figure 4 (b)). Once each dimension is understood, skills become straight forward to interpret by simply inspecting on which dimensions (property) each skill $\hat{\mathbf{o}}$ in Figure 4 (a) has significant weights, and interpreting properties of those dimensions ((Figure 4 (c)). In this way, we can interpret that Skill 2 is a forward movement skill, since it focuses on jumping and running forward, while Skill 1 is a landing skill. These interpretations can further be used to explain skill activation patterns in Figure 3 (b): Skill 2 has the longest duration because it is the major source of all forward movements. Skill 2 occasionally falls back to Skill 1 because, after jumping or running, the HalfCheetah needs to land and balance itself.

[4]Even on Reacher, a simple environment on which most algorithms converge to a similar performance, SA is still 38% better than the second best (AHP).

[5]The dominant skill phenomenon is also reported in other option implementations such as DAC, we give a detailed explanation in Appendix A.1

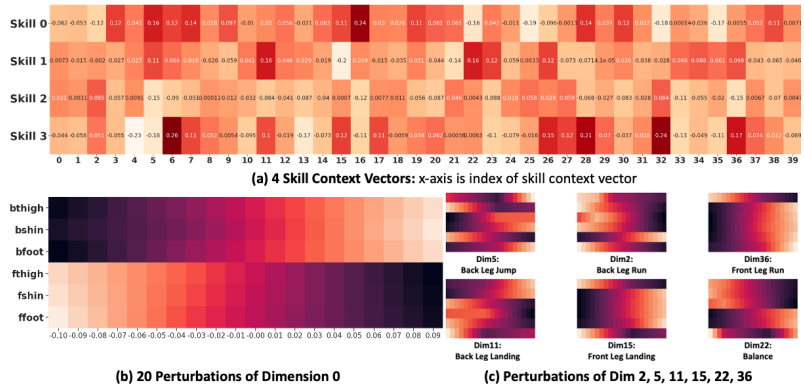


Figure 4: Interpretation of Skill Context Vectors

4.4 Transfer Learning

We follow DAC (Zhang & Whiteson, 2019) and run 6 pairs of transfer learning tasks constructed in DAC based on DeepMind Control Suite (Tassa et al., 2020). Each pair contains two different tasks. We train all models one million steps on the first task and switch to the second (SA's skill context matrix is subsequently frozen) to run another one million steps. Results are reported in Figure 5:

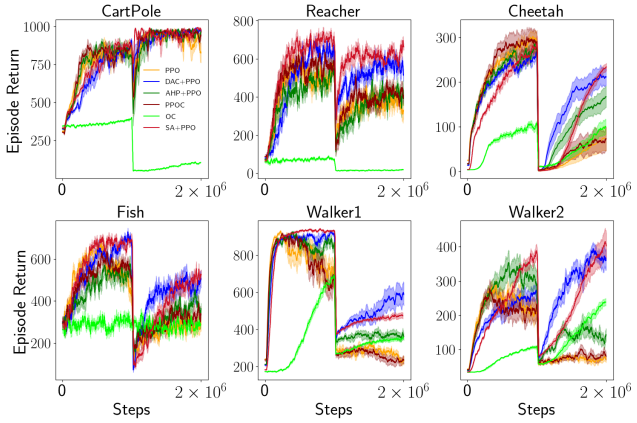


Figure 5: Performance on DAC transfer learning tasks

On the transfer learning (the second) task, SA's performance ranks the first in 5 out of 6 environments. This shows SA's advantages on reusing experience and knowledge across tasks. On the first task, SA's performance is also among the best algorithms in all environments. This further validates SA's advantage on single task as observed in section 4.1.

5 Conclusions

In this paper, we presented a novel MDP equivalence of the SMDP formulated option framework, from which an MDP implementation of the option framework, i.e., the Skill-Action architecture, was derived. We theoretically proved that SA has lower variance than conventional RL models and provided policy gradient theorems for updating SA. Our empirical studies on challenging infinite horizon robot simulation environments demonstrated that SA not only outperforms all baselines by a large margin, but also exhibits smaller variance, faster convergence, and good interpretability. On transfer learning, SA also outperforms the other models in 5 out of 6 environments and shows its advantage on reusing experience and knowledge across tasks.

The final and most important contribution of SA is hierarchically learning of explicit abstract actions' representations with "skill context vectors". This design significantly improves the scalability and interpretability of SA. It is straightforward to extend SA to deeper architectures, which paves the way for a large-scale pre-training and transfer learning architecture in the reinforcement learning area (Appendix A.1).

REFERENCES

Pierre-Luc Bacon. *Temporal Representation Learning*. PhD thesis, McGill University Libraries, 2018.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Christian Daniel, Herke Van Hoof, Jan Peters, and Gerhard Neumann. Probabilistic inference for determining options in reinforcement learning. *Machine Learning*, 104(2-3):337–357, 2016.

Finale Doshi-Velez and George Konidaris. Hidden parameter markov decision processes: A semiparametric regression approach for discovering latent task parametrizations. In *IJCAI: proceedings of the conference*, volume 2016, pp. 1432. NIH Public Access, 2016.

G David Forney. The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, 1973.

Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning. *arXiv preprint arXiv:1910.11956*, 2019.

Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

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

Karol Hausman, Jost Tobias Springenberg, Ziyu Wang, Nicolas Heess, and Martin Riedmiller. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

Matthew D Hoffman, David M Blei, Chong Wang, and John Paisley. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

Taylor W Killian, Samuel Daulton, George Konidaris, and Finale Doshi-Velez. Robust and efficient transfer learning with hidden parameter markov decision processes. In *Advances in neural information processing systems*, pp. 6250–6261, 2017.

Martin Klissarov, Pierre-Luc Bacon, Jean Harb, and Doina Precup. Learnings options end-to-end for continuous action tasks. *arXiv preprint arXiv:1712.00004*, 2017.

Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

Adam Kosiorek, Sara Sabour, Yee Whye Teh, and Geoffrey E Hinton. Stacked capsule autoencoders. In *Advances in Neural Information Processing Systems*, pp. 15512–15522, 2019.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International Conference on Machine Learning*, pp. 3744–3753, 2019.

Kfir Y Levy and Nahum Shimkin. Unified inter and intra options learning using policy gradient methods. In *European Workshop on Reinforcement Learning*, pp. 153–164. Springer, 2011.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Christian F Perez, Felipe Petroski Such, and Theofanis Karaletsos. Generalized hidden parameter mdps transferable model-based rl in a handful of trials. *arXiv preprint arXiv:2002.03072*, 2020.

Martin L Puterman. Markov decision processes: Discrete stochastic dynamic programming. 1994.

Kate Rakelly, Aurick Zhou, Chelsea Finn, Sergey Levine, and Deirdre Quillen. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International conference on machine learning*, pp. 5331–5340, 2019.

Matthew Riemer, Miao Liu, and Gerald Tesauro. Learning abstract options. In *Advances in neural information processing systems*, pp. 10424–10434, 2018.

Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, pp. 3856–3866, 2017.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

Matthew Smith, Herke Hoof, and Joelle Pineau. An inference-based policy gradient method for learning options. In *International Conference on Machine Learning*, pp. 4703–4712, 2018.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. 2018.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Yuval Tassa, Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, and Nicolas Heess. dmcontrol: Software and tasks for continuous control, 2020.

Dhruva Tirumala, Hyeonwoo Noh, Alexandre Galashov, Leonard Hasenclever, Arun Ahuja, Greg Wayne, Razvan Pascanu, Yee Whye Teh, and Nicolas Heess. Exploiting hierarchy for learning and transfer in kl-regularized rl. *arXiv preprint arXiv:1903.07438*, 2019.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pp. 5998–6008, 2017.

Markus Wulfmeier, Abbas Abdolmaleki, Roland Hafner, Jost Tobias Springenberg, Michael Neunert, Tim Hertweck, Thomas Lampe, Noah Siegel, Nicolas Heess, and Martin Riedmiller. Compositional transfer in hierarchical reinforcement learning. 2019.

Markus Wulfmeier, Dushyant Rao, Roland Hafner, Thomas Lampe, Abbas Abdolmaleki, Tim Hertweck, Michael Neunert, Dhruva Tirumala, Noah Siegel, Nicolas Heess, et al. Data-efficient hindsight off-policy option learning. *arXiv preprint arXiv:2007.15588*, 2020.

Shangtong Zhang and Shimon Whiteson. Dac: The double actor-critic architecture for learning options. In *Advances in Neural Information Processing Systems*, pp. 2012–2022, 2019.

A APPENDIX

A.1 LEARNING SKILLS AT MULTI-LEVELS OF GRANULARITY

Implementations of the option framework share some common limitations. When proposing the option framework, Sutton et al. (1999) expected that learning at multi-level of temporal abstraction should be in favor of faster convergence and better exploration. On the contrary, significant improvements on single task environments have not been witnessed in most option implementations (Klissarov et al., 2017; Smith et al., 2018; Harb et al., 2018; Zhang & Whiteson, 2019). To the best of our knowledge, SA is the first option implementation in which these properties are significantly witnessed but only on infinite horizon environments. In this section, we address this problem by first giving a theoretical explanation of why the value function is the main reason for this deficiency in section A.1.1 and how **deep wide value functions** could solve this problem. We then thoroughly explain the motivations of SA, and why it is a promising candidate for a deep wide framework, in section A.1.2. We also give a further explanation of how SA is connected to causality reinforcement learning literature and how a temporal causal reward can be used in objective to further solve this problem in section A.1.3.

A.1.1 PROBLEM STATEMENT AND EVIDENCES

The expectation of improvements of the option framework on single task environment builds on an assumption that, by exploiting hierarchical action and state space, an agent's searching space can be greatly reduced thus accelerates learning and improving exploration. However, as reported in section 4.2, most option frameworks including SA suffer from "the dominant skill problem" (Zhang & Whiteson, 2019) which prevents option frameworks from effectively learning hierarchy in action and state space as well as coordinating between skills.

One root reason for this problem is that conventional value functions $V[S_t]$ and $Q[S_t, O_t, A_t]$ make values depend on temporal latent variables indistinguishable (i.e. Although different skills o_1 and o_2 results to different values, such as $V[S_t, O_{t-1} = o_1] = 10$ and $V[S_t, O_{t-1} = o_2] = -10$. Because they arrive at the same state S_t, they have identical values under conventional value function $V[S_t] = 0$). This deficiency makes option frameworks can only learn skills at very coarse level thus fail to exploit hierarchical information. The solution is to use a **deep wide value function**: enabling the framework to learn fine-grained skills at mutli-levels of granularity (deep) and making value functions depend on latent variables with longer dependencies (e.g. $V[S_t, O_{t-1}]$ and $Q[S_t, O_t, A_t, O_{t-1}]$) (wide).

To have a better understanding the importance of the deep wide value function, let us consider a simple environment which can be easily solved by $Q[s_t, a_t, a_{t-1}]$ but not $Q[s_t, a_t]$.

Suppose we are training a robot which only has a camera sensor to cook thanksgiving turkey. In this setting there are only two states: $\mathbb{S} = \{\text{Raw Turkey Image}, \text{Cooked Turkey Image}\}$. The robot's action space only consists two actions $\mathbb{A} = \{\text{Stuff turkey}, \text{Roast turkey}\}$. As for reward, if the robot roast a stuffed turkey, then the reward is 10. However, if the robot roast an un-stuffed turkey, then the reward is -10. The stuff turkey action receives 0 reward.

The difficulty in this environment is, since the robot only has a camera to capture an image of the turkey, it can only observes either $\{\text{Raw Turkey Image}\}$ or $\{\text{Cooked Turkey Image}\}$. There is no way to look inside the turkey and see if the turkey is stuffed. Under this setting, a robot can never learn to first stuff a turkey and then roast it because $Q[\text{Raw Turkey Image}, \text{Stuff Turkey}] = Q[\text{Raw Turkey Image}, \text{Roast Turkey}] = 0$. Therefore, the robot can only randomly cook a turkey. However, this problem can be easily solved by using a deep wide value function $Q[S_t, A_t, A_{t-1}]$.

The core problem in this setting is, action has no affection on states, it only affects rewards. At first glance this is a Partially Observed MDP (POMDP) problem since the state of whether the turkey is stuffed is un-observed. This is true in all reinforcement learning settings without dependencies on latent variables. However, it goes much deeper in HRL settings.

In HRL, a common formulation is to estimate a latent variable O to encode hierarchical information and makes the policy depends on it $P(A_t|S_t, O_t)$. Since O is a latent variable, it is highly likely that at state S_t, different latent variable $P(A_t|S_t, O_t = O_x)$ and $P(A_t|S_t, O_t = O_y)$ emits the same

action $A_t = A_1$, and thus makes the conventional value function indistinguishable between O_x and O_y.

This phenomenon is especially common around the switching time step of two skills: around switching point, states usually compatible with both old and new skills. Conventional value functions will be especially confused at those moments. This is exactly what we observed in Figure 3: overall, skill 2 is executed consistently. However, there are some random switches to skill 1. And the randomization is increased between around switching time steps. To explicitly show this, we visualized "Run4" into a video: https://www.youtube.com/watch?v=QiLVZvI6NJU. The skill selection is very random at the beginning of the episode as well as around the switching point (the 16th second). These are exactly the most confusing moments of conventional value functions. This is not a cherry-pick result but a common problem. Similar patterns can also be observed here https://youtu.be/xrfxbI3duBM?t=4 in a HumanoidStandUp environment.

Due to the insufficiency of conventional value functions, compatible states have to be different enough to cause distinguishable values of value functions. Therefore, with conventional value functions, SA is only able to learn very coarse skills. For example, as shown in Figure 4 and the video, the HalfCheetah agent is only able learn two skills: one is to run forward, one is to stand up when fall. However it is not able to learn more fine-grained skills such as jump forward and landing. This problem is not limited to SA, but is a common problem in HRL. The solution is to use deep wide value functions.

A.1.2 Deep Wide Skill-Action Architecture (DWSA)

SA is carefully designed to make the most out of deep wide value functions. Compared to other HRL frameworks, SA has following advantages:

1. Stable and unbiased estimation: Thanks to proposition 3.4 and 3.5, the higher the order of the MDPs, the smaller the variance will be. The deep wide value functions stays unbiased estimations of conventional value functions no matter how many dependencies introduced. The current solution in option framework is a biased estimation (Harb et al., 2018) and adding hyper-parameters to the framework.
2. Easy to incorporate wide value functions: Incorporating a deep wide value function is straightforward, SA's **skill value upon arrival function** is already a wide function. The **skill value function** and the **skill-action value function** can be easily extended to wide function by adding a first-order dependency on $\hat{O}_{t-1}$.
3. Easy to incorporate deep value functions: SA is MDP formulated, extending SA to multiple hierarchies is straightforward.
4. Scalability to long time dependencies: SA is MDP formulated, adding more time dependencies is simply to change the 1st-order MDP to higher-order MDPs while both value functions and gradient theorems stay unchanged; SA is attention based, SA can easily attends to thousand time steps without adding any extra complexity to neither skill policy nor action policy.
5. Scalability to multiple hierarchies of skills: SA is attention based and embedding based. Adding skills is as simple as adding skill context matrix. In traditional option frameworks (Riemer et al., 2018), the number of option (note that each option is a neural network) grows at $O(N^L)$ complexity of levels (N is the number of options and L is the number of levels).
6. Interpretability. As shown in section 4.3, skill context vectors learned under SA-based architectures are straightforward to visualize and interpret. This property is especially useful for investigating multi-level granularity skills.

A.1.3 Causality Discovery Rewards

Although theoretically a DWSA can learn multi-level granularity skills, on-policy optimization algorithm is often insufficient for learning such models especially in sparse reward environments. However, SA has a natural connection with causal reinforcement learning thus can exploits causality as a reward in objective function to further facilitate fine grained skill discovery. In this section

we explain how skill embedding vectors learned by SA encodes temporal causality relationships and how to use them to devise causal rewards.

In causal reinforcement learning area, Doshi-Velez & Konidaris (2016) proposed Hidden Parameters MDP (Hi-MDP) in which a skill vector like hidden parameter vector is introduced to learn abstract properties from environments. PEARL (Rakelly et al., 2019) utilizes meta-learning framework to learn a skill representation that encodes abstract properties of a task and updates the framework in an off-policy manner to improve sample efficiency in transfer learning. Killian et al. (2017) extended Hi-MDP by including the hidden parameter vector into transition probability function. Perez et al. (2020) further extended their work by proposing Generalized Hidden Parameter MDPs (GHP-MDPs), a causality discovery framework by including hidden parameter vector into both transition function and value function.

GHP-MDPs is a special case of SA with number of skills $M = 1$. When $M > 1$, SA not only encodes causality relationships between environments and actions but also temporal causality between skills. Since the latent variable is modeled as a skill vector, the distance between different trajectories is straightforward to be calculated and thus can be used as a causal reward to encourage fine-grained and disentangled skills' discovery. To the best of our knowledge, SA is the first RL framework concerns causality in temporal abstraction sequences. We focus this paper on proposing SA, the causality rewarded SA will be discussed in future works.

Another interesting understanding of SA is that, rather than an implementation of the option framework, SA can also be seen as a novel capsule network Kosiorek et al. (2019) trained by policy gradient theorems. In Stacked Capsule Auto-Encoders (SCAE) (Kosiorek et al., 2019), a "capsule" vector encodes a different property (scale, orientation, etc.) of the visual object in each dimension. Kosiorek et al. (2019) proposed to delegate the complexity of part objects detection and part-to-whole objects aggregation by employing the attention mechanism (Lee et al., 2019) on which a generative model is then built to further decode the whole-part relationships. This design choice abstracts the complexity of inference away from the decoder and largely simplified the designation of the generative model.

In this paper, we follow their motivations of learning better representations and utilizing the attention mechanism to simplify the inference problem (sampling new skill without termination function). Moreover, the skill context vector is analogously to a capsule and the skill-action relationship is analogously to the whole-part relationship in the SCAE. Similar to SCAE utilizing the equi-variance property of the whole-part relationship to achieve computing efficiency and better performance, it will be very exciting to investigate potentially "equi-variance" or "invariance" properties existed in skill-action relationships, which might give rise to a novel causal inference architecture in the reinforcement learning area.

A.2 Related Workers In RL

In main text we only discuss a subset of related works from the option framework community. In this section we give a broader discussion of SA with other RL works.

It is worth to mention that, the novel formulation of SA establishes strong connections between causal reinforcement learning, meta-reinforcement learning and transfer learning (Gupta et al., 2019; Hausman et al., 2018). As discussed in section A.1.3, with number of skills $M = 1$, SA falls back to GHP-MDPs (Perez et al., 2020), which introduces an extra latent parameter vector in policy to encode causality between policy and environment properties. Similar settings are also employed by Rakelly et al. (2019) in meta-learning and Tirumala et al. (2019) in imitation learning. However, all these work only exploit skill context vectors without temporal dependencies.

Other than temporal level abstraction, the option framework also encodes action level abstraction. RHPO (Wulfmeier et al., 2019) proposed a hierarchical policy by using Gaussian Mixture Model (GMM) without temporal dependencies between latent variables. HO2 (Wulfmeier et al., 2020). Wulfmeier et al. (2020) extends RHPO (Wulfmeier et al., 2019) to optimize the option framework by employing a trust-region constrained off-policy algorithm. Their experiments prove that frameworks learning both action and temporal abstraction outperforms action-only models by a large margin. HO2 and SA complements each other under the option framework from different aspects: HO2 provides an efficient off-policy algorithm which significantly outperforms on-policy algorithms that

SA suffers from. On the other hand, due to the SMDP formulation, HO2 has to introduce the maximum number of switches as a hyper-parameter and marginalize over it to ensure temporal consistency. With SA's MDP formulation and attention mechanism, HO2 can improve temporal consistency from a much more efficient and data-driven manner.

Unlike HO2 largely benefits from temporal abstraction, our experiments show that one limitation of SA is that its improvements mainly comes from action abstraction rather than temporal. However, as discussed in section A.1, SA gives rise to an elegant solution of exploiting temporal abstraction and coordinating disentangled skills at multiple granularities. This topic is beyond the scope of this paper and remains for future work.

A.3 EXPERIMENTS RESULTS

A.3.1 PERFORMANCE

In this section we provide results for all ten OpenAI Gym Mujoco Environments. Those environments can be classified into two categories: infinite horizon environments (i.e., HalfCheetah, Swimmer, HumanoidStandup and Reacher) and finite horizon environments (the other).

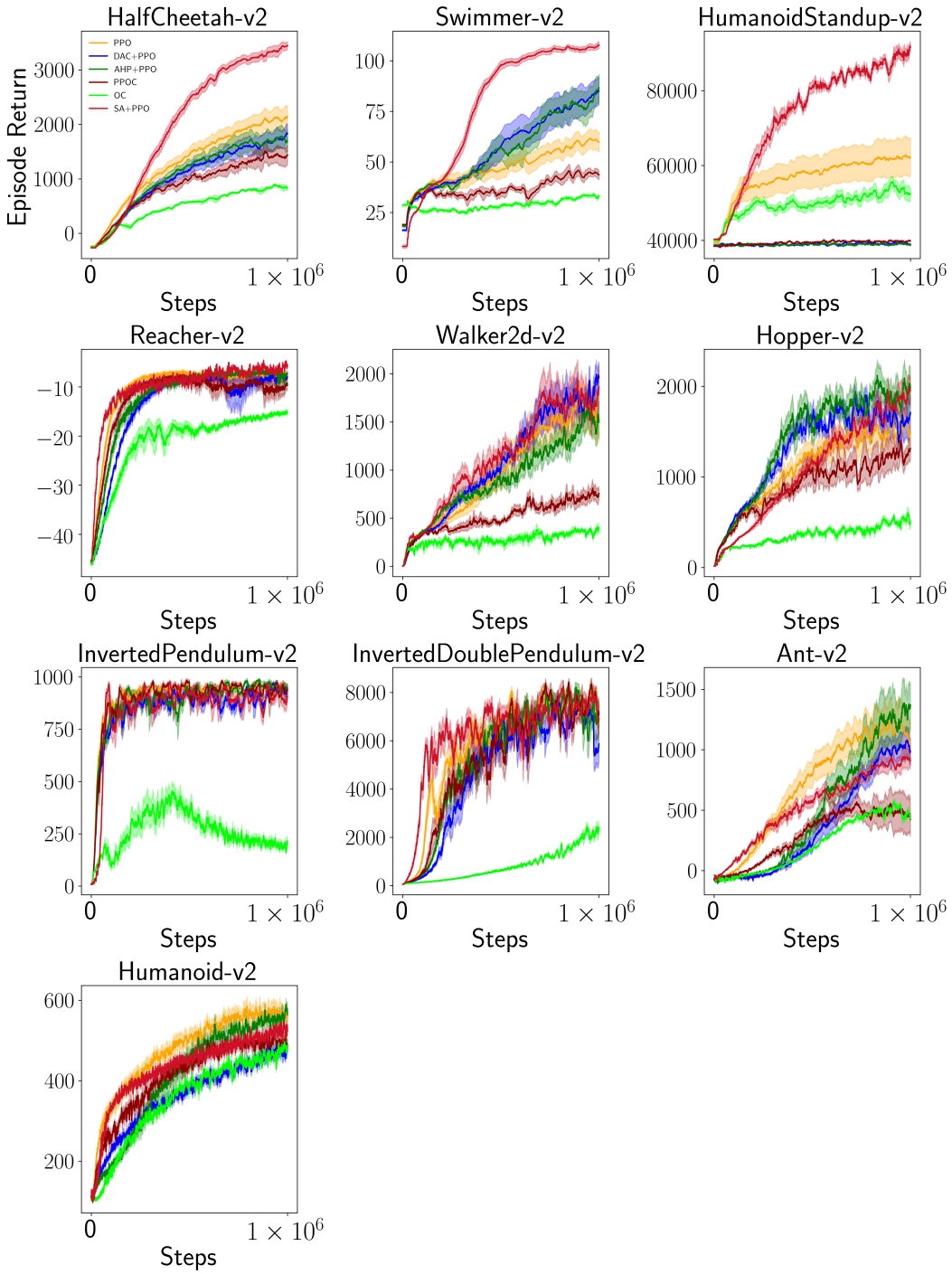


Figure 6: Performance of Ten OpenAI Gym MuJoCo Environments.

A.3.2 TEMPORAL EXTENSION

In Figure 7, we plot the average duration of each skill during 430 training episodes (each episode contains a trajectory of 512 time steps) of the HalfCheetah environment. In this environment, the agent learns to run half of a Cheetah by controlling 6 joints: back thigh, back shin, back foot, front thigh, front shin, and front foot. The faster the Cheetah runs forward, the higher return it gets from the environment. At the start of training, all skills' durations are short. After the 100-th episode, Skill 2's duration quickly grows and dominates the entire episode. The dominant skill phenomenon is also reported in other option implementations such as DAC. One explanation for this domination phenomenon is that for some single task environments, primitive actions might be enough to express the optimal policy, in which case extra levels of abstraction (skills) become overhead. However, because the duration of dominant skill starts to fall at the end of training and SA significantly outperforms PPO which only employs primary actions, these facts indicate that SA has a better capability of automatically discovering abstract actions from primary actions as well as coordinating between them.

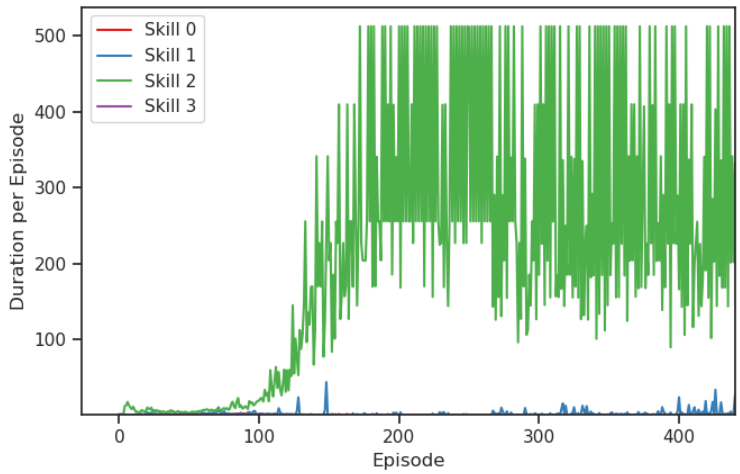


Figure 7: Duration of 4 options during 430 training episodes of HalfCheetah.

To illustrate how SA coordinates skills, we take the HalfCheetah model trained after 1 million steps and independently run it 4 times (4 episodes. each episode contains 512 time steps). Skill activation sequences of 4 runs are then plotted in Figure 8. As we can see that there are some common patterns

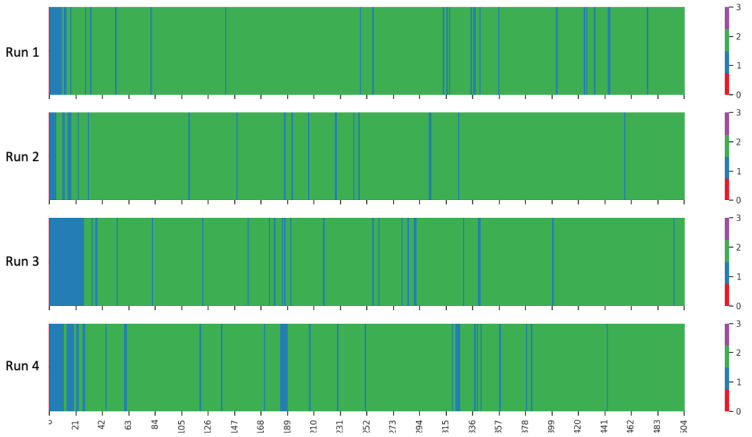


Figure 8: Activated option sequences of 4 independent HalfCheetah runs.

between all 4 independent runs. For example, all runs start with Skill 0 and use Skill 1 at the early

stage. After executing Skill 1 for a short period, they all switch to Skill 2 which has longest durations in all 4 runs. From time to time they will fall back to Skill 1 for short periods and quickly switch to Skill 2 again. This pattern of coordination indicates that Skill 1 and Skill 2 have completely different functionality and SA has the capability of automatically discovering as well as leveraging those skills.

A.3.3 Interpretation of Skill Context Vectors

In this section we continue with the HalfCheetah model used in Section A.3.2 and demonstrate how to interpret skill context vectors as well as skill activation sequences (Figure 8). In HalfCheetah, the agent learns to run half of a Cheetah by controlling 6 joints: back thigh, back shin, back foot, front thigh, front shin, and front foot. The faster the Cheetah runs forward, the higher return it gets from the environment. We interpret skill context vectors and activation patterns by first inspecting what property each dimension of the skill context vector encodes (Figure 10). Once each dimension is understood, skills (Figure 9) become straight forward to interpret by simply inspecting on which dimension (property) they have the most significant weights (Figure 11). These interpretations can further be taken to explain skill activation patterns in Figure 8.

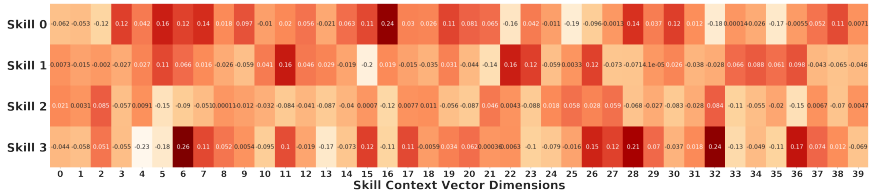


Figure 9: Heatmap of all 4 skill context vectors

As the first step, we follow Sabour et al. (2017) to interpret what property each dimension of the skill context vector in Figure 9 encodes by perturbing each dimension and decode perturbed skill context vectors into primary actions. Specifically, we perturb one dimension by adding a range of perturbations $[-0.1, 0.09]$ by intervals of 0.01 onto it while keep the other dimensions fixed. After perturbation, each skill context vector dimension has 20 perturbed vectors. We then use the action policy decoder to decode all those vectors into primary actions and see how the perturbation affects the primary action. As an illustration, we plot Dimension 0's all 20 perturbed results in Figure 10.

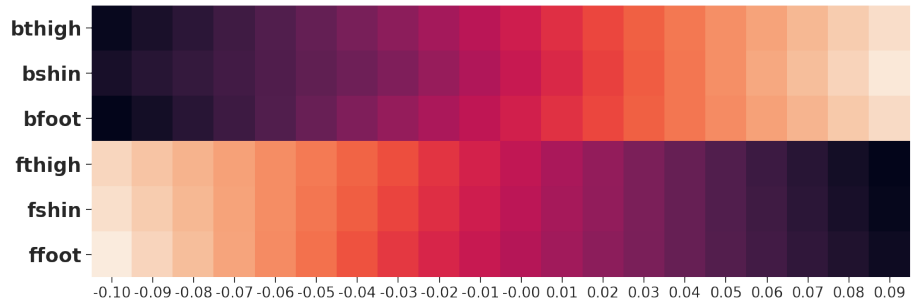


Figure 10: Perturbation on the Dim 0

With visualization of perturbation results in hand, we can interpret what property each dimension encode by inspecting relationships between perturbations and primary actions. In Figure 10, as an example, it is clear that changes on Dim 0 has opposite effect on the back leg and front leg: a larger value on Dim 0 will assign the back leg a larger torque while the front leg a smaller one, and vice versa. This means Dim 0 is has a focus point property: it focuses torque on only one leg.

Once we know how to interpret one dimension, we can move on to interpret the whole skill context vector. Since Skill 1 and Skill 2 are two main skills employed in Figure 8, here we provide an example of how to interpret them. Figure 9 shows that Skill 1 has significant values on dimension 11, 15 and 22. Skill 2 is significant on dimension 2, 5 and 36. We demonstrate these dimensions in the same manner as Figure 10 below:

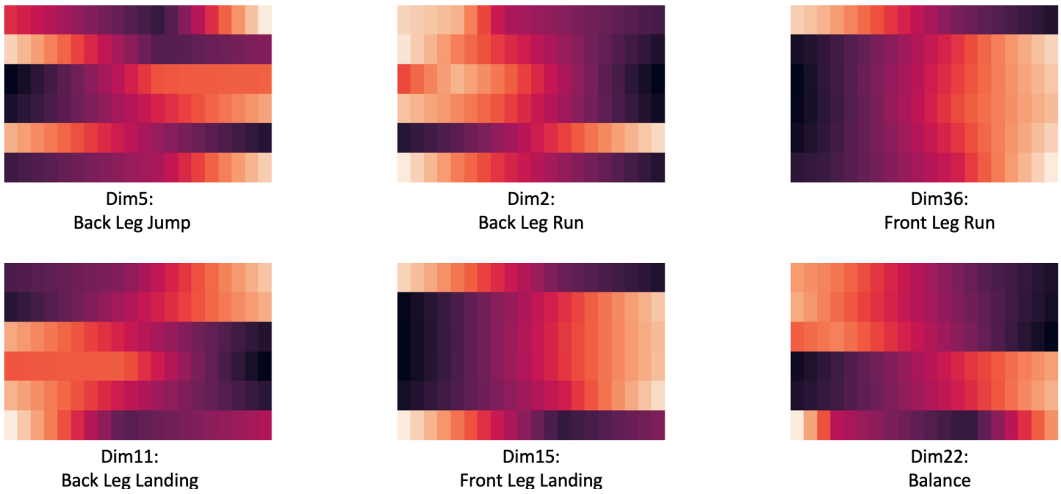


Figure 11: Interpretation of Skill 1 and Skill 2

Subfigures in Figure 11 can be interpreted in the same manner as Figure 10. As an example, from Figure 9 we can see that Skill 1 has a significant small value on Dim 11. In Figure 11, it shows that a smaller Dim 11 will twist the front leg forward and back foot forward while twist back thigh, back shin backward. Composition of these movements is a back leg landing property. Similarly, we can interpret that Dim 15 is a front leg landing property and Dim 22 is a balancing property. Therefore, Skill 1 is focusing on landing from all positions.

Unlike other skill context vectors which have apparent focusing dimensions, Skill 2 has a rather balanced skill context vector. It has no apparently dominant dimension. It only has slightly more significant values on Dim 2, 5, 36, which are focusing on jumping and running properties. Therefore, Skill 2 is more like an "all-weather" skill: it is a skill having very balanced properties with a slightly demonstration on running and jumping.

Interpretations of Skill 1 and 2 above can then be taken to understand skill activation patterns in Figure 8: as an all-weather skill, Skill 2 is the most frequently executed one and has the longest duration. From time to time, when the Cheetah needs to land and balance itself, Skill 1 will be executed. However, since landing skill does not provide power of moving forward and thus has lower returns to continue, once the body is balanced the Cheetah will quickly stop Skill 1's execution and keep running with Skill 2.

A.4 MDP Equivalence to the SMDP Option Framework

In this section, we show that the the conventional Semi-Markov Decision Problem (SMDP) option framework which employs Markovian options actually has an MDP equivalence. We first follow Bishop (2006)'s method and formulate the dynamics of the option framework as an Hidden Markov Model (HMM) (Bishop, 2006) in section A.4.2. With Probability Graphical Model (PGM) (Bishop, 2006) and its conditional independence relationships (Chapter 8.2.1 (Bishop, 2006)) in hand, we then move on to prove that MDP formulation has identical value functions (section A.4.3), bellman equations as well as intra-option policy and termination policy gradients to SMDP formulation (section A.4.4). To the best of our knowledge, this is the first work discovering the option framework's MDP equivalence and deriving the option framework from a PGM view.

A.4.1 BACKGROUND: THE OPTION FRAMEWORK

Sutton et al. (1999) proposed the option framework to demonstrate the temporal abstraction problem. A scalar $\mathrm{o} \in \mathbb{Z}$ denotes the index of an option where $\mathbb{O} \subseteq \{1, 2, \ldots, M\}$ and M is the number of options. An Markovian option is a triple $(\mathbb{I}_o, P_o(\mathbf{a}|\mathbf{s}), P_o(\mathrm{b}|\mathbf{s}))$ in which $\mathbb{I}_o \subseteq \mathbb{S}$ is an initiation set where the option o can be initiated. $P_o(\mathbf{a}|\mathbf{s}) : \mathbb{S} \to \mathbb{A}$ is the intra-option policy which maps environment states $\mathbf{s} \in \mathbb{S}$ to an action vector $\mathbf{a} \in \mathbb{A}$. $P_o(\mathrm{b}|\mathbf{s}) : \mathbb{S} \to \mathbb{Z}_2$ is a termination function where b is a binary random variable. It is used to determine whether to terminate ($\mathrm{b} = 1$) the policy $P_o(\mathbf{a}|\mathbf{s})$ or not ($\mathrm{b} = 0$). Conventionally, $\beta_o = P_o(\mathrm{b} = 1|\mathbf{s})$. Since an option's execution may persist over a variable period of time, a set of options' execution together with its value functions constitutes a Semi-Markov Decision Problem (SMDP) (Puterman, 1994). When an old option is terminated, a new option will be sampled from the master policy (policy-over-options) $o \sim P(o_{t+1}|\mathbf{s}_{t+1}) : \mathbb{S} \to \mathbb{O}$. Due to the SMDP formulation, an option can only be improved when the option terminates.

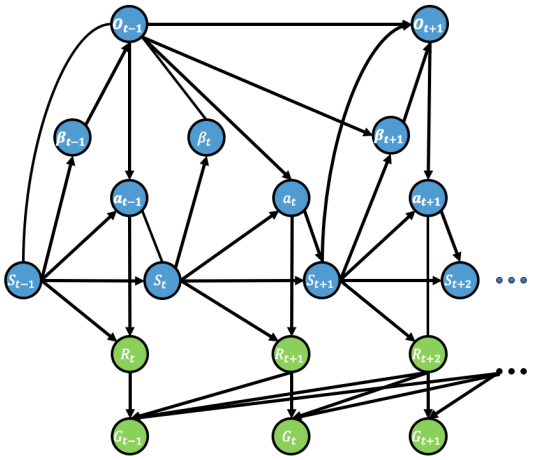


Figure 12: An Illustration of the SMDP Option Framework. An option o_{t-1} is selected by master policy $P(\mathrm{o}_{t-1}|\mathbf{s}_{t-1})$ at time step $t-1$. At time step t, termination function $\beta_{o_{t-1}}(\mathbf{s}_t)$ determines to continue option o_{t-1}. So that there is no random variable o_t at time step t compared to there are random variables $\mathbf{o}$ at every time step in MDP formulation (figure 13).

We refer this as the SMDP-style learning which is sample inefficient and prevents applying SOTA MDP based algorithms such as the Proximal Policy Optimization (PPO) algorithm (Schulman et al., 2017).

A.4.2 HMM DYNAMICS FOR THE OPTION FRAMEWORK

We follow Bishop (2006)'s formulation of mixture distribution and Probabilistic Graphical Models (PGMs). By introducing option variables as latent variables and adding extra dependencies between them, we show that the conventional SMDP version of the option framework (Bacon et al., 2017; Sutton & Barto, 2018; Sutton et al., 1999; Harb et al., 2018; Zhang & Whiteson, 2019) has an MDP equivalence. Following Bishop (2006)'s notation, we use bolded letter $\mathbf{s} \in \mathbb{S}$ to denote a random variable and normal letter s to denote its realization. Without special clarification, a random vector can have either a vector of continuous or discrete entries. Vector $\mathbf{o} \in \mathbb{O}$ is an M-dimensional one-hot vector and each entry $\mathrm{o} \in \{0, 1\}$ is a binary random variable. $P(\mathbf{o}_t|\mathbf{s}_t)$ denotes the probability distribution over one-hot vector $\mathbf{o}$ at time step t conditioned on state $\mathbf{s}_t$. $P(\mathbf{o}_t = \mathrm{o}_t|\mathbf{s}_t)$ denotes a probability entry (a scalar value) of the random variable $\mathbf{o}_t$ with a realization at time step t where $\mathrm{o}_t = 1$ and $\mathrm{o} \in \mathbf{o}_t / \mathrm{o}_t = 0$.

In figure 13, $\mathbf{s} \in \mathbb{S}$, $\mathbf{o} \in \mathbb{O}^M$, $\mathbf{b} \in \mathbb{B}^M$ and $\mathbf{a} \in \mathbb{A}$, denote the state, option, termination and action random variable respectively. $\mathbf{o}$ is an M-dimensional one-hot vector and $\mathbf{b}$ is an M-dimensional binary vector where each entry $\mathrm{b} \in \{0, 1\}$. M is the number of options. R_{t+1} is the actual reward received from the environment after executing action $\mathbf{a}_t$ in state $\mathbf{s}_t$. $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} \cdots$ is the discounted expected return where $\gamma \in \mathbb{R}$ is a discount factor.

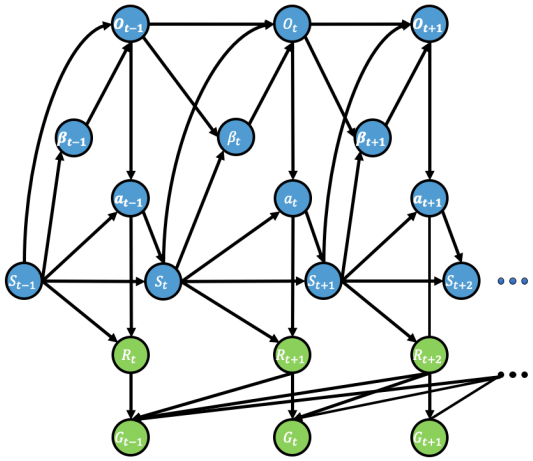


Figure 13: PGM of the MDP Option Framework

The termination policy distribution $P(\mathbf{b}_t|\mathbf{s}_t, \mathbf{o}_{t-1}) : \mathbb{S} \times \mathbb{O} \to \mathbb{B}$ can be formulated as a mixture distribution[6] conditioned on option vector (the one-hot vector) $\mathbf{o}_{t-1}$ and state $\mathbf{s}_t$.

$$P(\mathbf{b}_t|\mathbf{s}_t, \mathbf{o}_{t-1}) = \prod_{\mathrm{i} \in \mathbf{o}_{t-1}} P_{\mathrm{i}}(\mathrm{b}_t|\mathbf{s}_t)^{\mathrm{i}}. \tag{16}$$

Because each option has its own **termination policy** $P_o(\mathbf{b}|\mathbf{s})$, with a slightly abuse of notation, in equation (16) we use $P(\mathbf{b}_t|\mathbf{s}_t, \mathbf{o}_{t-1})$ to denote the termination policy activated at time step t by previous chosen option $\mathbf{o}_{t-1}$. To keep notation uncluttered, we use $\beta_t = P(\mathbf{b}_t = 1|\mathbf{s}_t, \mathbf{o}_{t-1})$ to denote the probability of option $\mathbf{o}_{t-1}$ terminates at time step t and $(1 - \beta_t) = P(\mathbf{b}_t = 0|\mathbf{s}_t, \mathbf{o}_{t-1})$ to denote the probability of continuation.

Conventionally, master policy (Zhang & Whiteson, 2019) (also called "policy-over-options" (Sutton et al., 1999; Bacon et al., 2017))) is defined as:

$$P(\mathbf{o}_t|\mathbf{s}_t). \tag{17}$$

Similarly, we propose a novel **mixture master policy** as a mixture distribution[7]:

$$P(\mathbf{o}_t|\mathbf{s}_t, \mathbf{b}_t, \mathbf{o}_{t-1}) = P(\mathbf{o}_t|\mathbf{s}_t)^{\mathrm{b}_t} P(\mathbf{o}_t|\mathbf{o}_{t-1})^{1-\mathrm{b}_t}, \tag{18}$$

where $P(\mathbf{o}_t|\mathbf{o}_{t-1})$ is a degenerated probability distribution (Puterman, 1994)

$$P(\mathbf{o}_t|\mathbf{o}_{t-1}) = \begin{cases} 1 & \text{if } \mathbf{o}_t = \mathbf{o}_{t-1}, \\ 0 & \text{if } \mathbf{o}_t \neq \mathbf{o}_{t-1}. \end{cases} \tag{19}$$

As shown in equation (18), the master policy only exists when $\mathrm{b}_t = 1$ the option terminates. Therefore, PPOC (Klissarov et al., 2017) uses inaccurate gradients for updating the master policy during an option's execution.

According to the conditional dependency relationships in PGM (figure 13), the joint probability distribution of $\mathbf{o}_t$ and $\mathbf{b}_t$ can be written as:

$$P(\mathbf{o}_t, \mathbf{b}_t|\mathbf{s}_t, \mathbf{o}_{t-1}) = P(\mathbf{b}_t|\mathbf{s}_t, \mathbf{o}_{t-1}) P(\mathbf{o}_t|\mathbf{s}_t, \mathbf{b}_t, \mathbf{o}_{t-1}), \tag{20}$$

[6]Different from conventional formulation which only depends on state $\mathbf{s}_t$, our termination function has an extra dependence on $\mathbf{o}_{t-1}$

[7]Different from conventional formulation which only depends on state $\mathbf{s}_t$, our mixture master policy has extra dependencies on $\mathbf{o}_{t-1}$ and $\mathbf{b}_t$

and the marginal probability distribution can be written as:

$$\begin{aligned} P(\mathbf{o}_t|\mathbf{s}_t, \mathbf{o}_{t-1}) &= \sum_{\mathbf{b}_t} P(\mathbf{b}_t|\mathbf{s}_t, \mathbf{o}_{t-1}) P(\mathbf{o}_t|\mathbf{s}_t, \mathbf{b}_t, \mathbf{o}_{t-1}) \\ &= P(\mathbf{b}_t = 0|\mathbf{s}_t, \mathbf{o}_{t-1}) P(\mathbf{o}_t|\mathbf{o}_{t-1}) + P(\mathbf{b}_t = 1|\mathbf{s}_t, \mathbf{o}_{t-1}) P(\mathbf{o}_t|\mathbf{s}_t) \\ &= (1 - \beta_t) P(\mathbf{o}_t|\mathbf{o}_{t-1}) + \beta_t P(\mathbf{o}_t|\mathbf{s}_t) \\ &= (1 - \beta_t) \mathbf{1}_{\mathbf{o}_t = \mathbf{o}_{t-1}} + \beta_t P(\mathbf{o}_t|\mathbf{s}_t). \end{aligned} \tag{21}$$

The **intra-option (action) policy** distribution can also be formulated as a mixture distribution

$$P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t) = \prod_{\mathrm{i} \in \mathbf{o}_t} P_{\mathrm{i}}(\mathbf{a}_t|\mathbf{s}_t)^{\mathrm{i}}. \tag{22}$$

Therefore, the dynamics of the PGM in figure 13 can be written as:

$$\begin{aligned} P(\tau) =& P(\mathbf{s}_0) P(\mathbf{o}_0) P(\mathbf{a}_0|\mathbf{s}_0, \mathbf{o}_0) \\ & \prod_{t=1}^{\infty} P(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_{t-1}) P(\mathbf{b}_t|\mathbf{s}_t, \mathbf{o}_{t-1}) P(\mathbf{o}_t|\mathbf{b}_t, \mathbf{s}_t, \mathbf{o}_{t-1}) P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t), \end{aligned} \tag{23}$$

where $P(\tau) = P(\mathbf{s}_0, \mathbf{o}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{b}_1, \mathbf{o}_1, \mathbf{a}_1, \ldots)$ denotes the joint distribution of the PGM. Notice that under this formulation, $P(\tau)$ is actually an HMM with $\mathbf{s}_t$, $\mathbf{a}_t$ as observable random variables and $\mathbf{b}_t$, $\mathbf{o}_t$ as latent variables.

It is worth to mention that equation (19) is essentially the indicator function $\mathbf{1}_{\mathbf{o}_t = \mathbf{o}_{t-1}}$ used in conventional SMDP option framework papers and the last line in equation (21) is identical to transitional probability distribution in their formulation. However, as we show in this section, by adding latent variables $\mathbf{o}_{t-1}$ and introducing the dependency between $\mathbf{o}_t$ and $\mathbf{b}_t$, our formulation is essentially an HMM. It opens the door to introduce many well developed PGM algorithms such as message passing (Forney, 1973) and variational inference (Hoffman et al., 2013) to the reinforcement learning framework. As we show below, the nice conditional independence relationships enjoyed by this model also enable us to prove the equivalence between the option framework's SMDP and MDP formulation.

A.4.3 MDP FORMULATION FOR THE OPTION FRAMEWORK

With PGM in hand, we now prove that the HMM formulated MDP option framework has identical value functions with the conventional SMDP option framework(Bacon et al., 2017; Sutton et al., 1999). In this section, we first show that all value functions defined on our PGM are identical to the SMDP formulation. We will prove that the gradients are also the same in next section.

We follow Sutton & Barto (2018)'s notation in this section and write value functions for MDP below:

$$\begin{aligned} V[\mathbf{s}_t] = \mathbb{E}[G_t|\mathbf{s}_t] &= \sum_{G_t} G_t \sum_{\mathbf{o}_t} P(G_t, \mathbf{o}_t|\mathbf{s}_t) \\ &= \sum_{\mathbf{o}_t} P(\mathbf{o}_t|\mathbf{s}_t) \sum_{G_t} G_t P(G_t|\mathbf{s}_t, \mathbf{o}_t) \\ &= \sum_{\mathbf{o}_t} P(\mathbf{o}_t|\mathbf{s}_t) \mathbb{E}[G_t|\mathbf{o}_t, \mathbf{s}_t] \\ &= \sum_{\mathbf{o}_t} P(\mathbf{o}_t|\mathbf{s}_t) Q_O[\mathbf{o}_t, \mathbf{s}_t], \end{aligned} \tag{24}$$

where $V[\mathbf{s}_t]$ is the state value function(Sutton & Barto, 2018) and $Q_O[\mathbf{o}_t, \mathbf{s}_t]$ is the option value function(Bacon et al., 2017; Sutton et al., 1999). Note that in deriving equation (24) we only use summation rule and production rule, the conditional dependency relationships in PGM (figure 13) are not used. The option value function $Q_O[\mathbf{o}_t, \mathbf{s}_t]$ can be further expanded as:

$$\begin{aligned} Q_O[\mathbf{o}_t, \mathbf{s}_t] = \mathbb{E}[G_t|\mathbf{o}_t, \mathbf{s}_t] &= \sum_{\mathbf{a}_t} P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t) \mathbb{E}[G_t|\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t] \\ &= \sum_{\mathbf{a}_t} P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t) Q_U[\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t], \end{aligned} \tag{25}$$

where $Q_U[\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t]$ is the option-action value function.

Proposition A.4.3.1. *MDP formulation has identical state value function $V[\mathbf{s}_t]$ and option value function $Q_O[\mathbf{o}_t, \mathbf{s}_t]$ to SMDP formulations*

Proof. Note that in derivations above we only use summation and production rules. Both equation (24) and (25) are identical to the conventional SMDP option framework. □

From now on, we will continue derivations with conditonal independence relationships encoded in PGM (Chapter 8.2.1 (Bishop, 2006)). We have following conditional independence relationships from PGM (figure 13):

$$\{R_{t+2}, G_{t+1}\} \perp\!\!\!\perp \{\mathbf{b}_{t+1}\} \quad | \{\mathbf{o}_{t+1}\}, \tag{26}$$
$$\{R_{t+2}, G_{t+1}\} \perp\!\!\!\perp \{\mathbf{s}_t\} \quad | \{\mathbf{s}_{t+1}, \mathbf{o}_t\}, \tag{27}$$
$$\{R_{t+2}, G_{t+1}\} \perp\!\!\!\perp \{\mathbf{a}_t\} \quad | \{\mathbf{s}_{t+1}\}, \tag{28}$$
$$\{R_{t+2}, G_{t+1}\} \perp\!\!\!\perp \{\mathbf{o}_t\} \quad | \{\mathbf{s}_{t+1}, \mathbf{o}_{t+1}\}, \tag{29}$$
$$\{R_{t+1}, G_t, \mathbf{s}_{t+1}\} \perp\!\!\!\perp \{\mathbf{o}_t\} \quad | \{\mathbf{a}_t\}. \tag{30}$$

With above conditional independence relationships in hand, we now show that the MDP formulation has identical value functions to the conventional SMDP formulation(Sutton et al., 1999; Bacon et al., 2017).

Proposition A.4.3.2. *MDP formulation has identical option-action value function $Q_U[\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t]$ to SMDP formulations*

$$Q_U[\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t] = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) U[\mathbf{s}_{t+1}, \mathbf{o}_t]. \tag{31}$$

Proof.

$$\begin{aligned}
Q_U[\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t] =& \mathbb{E}[G_t|\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t] \\
=& \mathbb{E}[R_{t+1} + \gamma G_{t+1}|\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t] && \text{by definition of } G_t \\
=& \mathbb{E}[R_{t+1}|\mathbf{s}_t, \mathbf{a}_t] + && \text{use eq (30)} \\
& \gamma \sum_{G_{t+1}} G_{t+1} \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{o}_t, \mathbf{a}_t) P(G_{t+1}|\mathbf{s}_{t+1}, \mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t) \\
=& r(\mathbf{s}_t, \mathbf{a}_t) + \\
& \gamma \sum_{G_{t+1}} G_{t+1} \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) P(G_{t+1}|\mathbf{s}_{t+1}, \mathbf{o}_t) && \text{use eq 27 28 and 30} \\
=& r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \mathbb{E}[G_{t+1}|\mathbf{s}_{t+1}, \mathbf{o}_t] \\
=& r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) U[\mathbf{s}_{t+1}, \mathbf{o}_t].
\end{aligned}$$

□

Proposition A.4.3.3. *MDP formulation has identical option-value function upon arrival $U[\mathbf{s}_{t+1}, \mathbf{o}_t]$ to SMDP formulations*[8]

$$U[\mathbf{s}_{t+1}, \mathbf{o}_t] = (1 - \beta_{t+1}) Q_O[\mathbf{o}_{t+1} = \mathrm{o}_t, \mathbf{s}_{t+1}] + \beta_{t+1} V[\mathbf{s}_{t+1}] \tag{32}$$
$$= Q_O[\mathbf{o}_{t+1} = \mathrm{o}_t, \mathbf{s}_{t+1}] - \beta_{t+1} A[\mathbf{o}_{t+1} = \mathrm{o}_t, \mathbf{s}_{t+1}]. \tag{33}$$

[8]Both equations (32) and (33) is largely used in the conventional SMDP papers(Sutton et al., 1999; Bacon et al., 2017).

Proof.

$$
\begin{aligned}
U[\mathbf{s}_{t+1}, \mathbf{o}_t] =& \mathbb{E}[G_{t+1}|\mathbf{s}_{t+1}, \mathbf{o}_t] \\
=& \sum_{G_{t+1}} G_{t+1} \\
& \sum_{\mathbf{o}_{t+1}} \sum_{\mathbf{b}_{t+1}} P(\mathbf{b}_{t+1}|\mathbf{o}_t, \mathbf{s}_{t+1}) P(\mathbf{o}_{t+1}|\mathbf{b}_{t+1}, \mathbf{o}_t, \mathbf{s}_{t+1}) P(G_{t+1}|\mathbf{o}_{t+1}, \mathbf{b}_{t+1}, \mathbf{o}_t, \mathbf{s}_{t+1}) \\
=& \sum_{\mathbf{o}_{t+1}} \sum_{\mathbf{b}_{t+1}} P(\mathbf{b}_{t+1}|\mathbf{o}_t, \mathbf{s}_{t+1}) P(\mathbf{o}_{t+1}|\mathbf{b}_{t+1}, \mathbf{o}_t, \mathbf{s}_{t+1}) \sum_{G_{t+1}} G_{t+1} P(G_{t+1}|\mathbf{o}_{t+1}, \mathbf{s}_{t+1}) \\
=& \sum_{\mathbf{o}_{t+1}} \left[(1 - \beta_{t+1}) \mathbf{1}_{\mathbf{o}_{t+1} = \mathbf{o}_t} + \beta_{t+1} P(\mathbf{o}_{t+1}|\mathbf{s}_{t+1}) \right] Q_O[\mathbf{o}_{t+1}, \mathbf{s}_{t+1}] \\
=& (1 - \beta_{t+1}) Q_O[\mathbf{o}_{t+1} = \mathrm{o}_t, \mathbf{s}_{t+1}] + \beta_{t+1} V[\mathbf{s}_{t+1}] \\
=& Q_O[\mathbf{o}_{t+1} = \mathrm{o}_t, \mathbf{s}_{t+1}] - \beta_{t+1} A[\mathbf{o}_{t+1} = \mathrm{o}_t, \mathbf{s}_{t+1}].
\end{aligned}
$$

from line 3 to line 4 use equation (26) and (29). From line 4 to line 5 use equation (21) and definition of Q_O. The second last line use equation (24). The last line use the definition of advantage function A. □

Under our MDP formulation, we also propose proposition A.4.3.4. We derive our gradient theorems based on equation (34) in section A.4.4. This important relationship largely simplify derivations than the original paper (Bacon et al., 2017) as well as give rise to the SA.

Proposition A.4.3.4. *The option-value function upon arrival $U[\mathbf{s}_{t+1}, \mathbf{o}_t]$ is an expectation over option value function $Q_O[\mathbf{o}_{t+1}, \mathbf{s}_{t+1}]$ conditioned on previous option O_t*

$$
U[\mathbf{s}_{t+1}, \mathbf{o}_t] = \sum_{\mathbf{o}_{t+1}} P(\mathbf{o}_{t+1}|\mathbf{o}_t, \mathbf{s}_{t+1}) Q_O[\mathbf{o}_{t+1}, \mathbf{s}_{t+1}]. \tag{34}
$$

Proof. Following proof of proposition A.4.3.3,

$$
\begin{aligned}
U[\mathbf{s}_{t+1}, \mathbf{o}_t] =& \sum_{\mathbf{o}_{t+1}} \sum_{\mathbf{b}_{t+1}} P(\mathbf{b}_{t+1}|\mathbf{o}_t, \mathbf{s}_{t+1}) P(\mathbf{o}_{t+1}|\mathbf{b}_{t+1}, \mathbf{o}_t, \mathbf{s}_{t+1}) \sum_{G_{t+1}} G_{t+1} P(G_{t+1}|\mathbf{o}_{t+1}, \mathbf{s}_{t+1}) \\
=& \sum_{\mathbf{o}_{t+1}} P(\mathbf{o}_{t+1}|\mathbf{o}_t, \mathbf{s}_{t+1}) Q_O[\mathbf{o}_{t+1}, \mathbf{s}_{t+1}].
\end{aligned}
$$

□

A.4.4 Gradients for the MDP Option Framework

In above sections, we formulate dynamics of the option framework using HMM and prove the MDP build on it has identical value functions to SMDP formulation. In this section we will prove that both MDP and SMDP formulations (Bacon et al., 2017) share same intra-option and termination gradients. Our derivations is largely simplified by equation (34) compared to previous work.

Let θ_a denote parameter vector for intra-option policies $P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t; \theta_a)$ and θ_b denote parameter vector for termination policies $P(\mathbf{b}_t|\mathbf{s}_t, \mathbf{o}_{t-1}; \theta_b)$. To keep notation uncluttered, we drop the dependency on parameter vector θ in derivations below.

Proposition A.4.4.1. *MDP formulation has identical Intra-Option Policy Gradient with SMDP formulation in (Bacon et al., 2017).*

$$
\begin{aligned}
\frac{\partial Q_O[\mathbf{s}_t, \mathbf{o}_t]}{\partial \theta_a} = \sum_{\mathrm{k}=0}^{\infty} \sum_{\mathbf{s}_{t+k}, \mathbf{o}_{t+k}} & P_\gamma^{(k)}(\mathbf{s}_{t+k}, \mathbf{o}_{t+k}|\mathbf{s}_t, \mathbf{o}_t) \\
& \sum_{\mathbf{a}_{t+k}} \frac{\partial P(\mathbf{a}_{t+k}|\mathbf{s}_{t+k}, \mathbf{o}_{t+k})}{\partial \theta_a} Q_U(\mathbf{s}_{t+k}, \mathbf{o}_{t+k}, \mathbf{a}_{t+k}).
\end{aligned} \tag{35}
$$

Proof. This is a direct result by taking gradient of θ_a with respect to equation (25) by using equation (31) and (34):

$$\begin{aligned}
\frac{\partial Q_O[\mathbf{s}_t, \mathbf{o}_t]}{\partial \theta_a} =& \sum_{\mathbf{a}_t} \frac{\partial P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t)}{\partial \theta_a} Q_U[\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t] + \gamma \sum_{\mathbf{a}_t} P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t) \frac{\partial Q_U[\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t]}{\partial \theta_a} \\
=& \sum_{\mathbf{a}_t} \frac{\partial P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t)}{\partial \theta_a} Q_U[\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t] \\
&+ \gamma \sum_{\mathbf{a}_t} P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t) \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \frac{\partial U[\mathbf{o}_t, \mathbf{s}_{t+1}]}{\partial \theta_a} \\
=& \sum_{\mathbf{a}_t} \frac{\partial P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t)}{\partial \theta_a} Q_U[\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t] \\
&+ \gamma \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{o}_t) \sum_{\mathbf{o}_{t+1}} P(\mathbf{o}_{t+1}|\mathbf{s}_{t+1}, \mathbf{o}_t) \frac{\partial Q_O[\mathbf{o}_{t+1}, \mathbf{s}_{t+1}]}{\partial \theta_a} \\
=& \sum_{\mathbf{a}_t} \frac{\partial P(\mathbf{a}_t|\mathbf{s}_t, \mathbf{o}_t)}{\partial \theta_a} Q_U[\mathbf{o}_t, \mathbf{s}_t, \mathbf{a}_t] + \gamma \sum_{\mathbf{o}_{t+1}, \mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}, \mathbf{o}_{t+1}|\mathbf{s}_t, \mathbf{o}_t) \frac{\partial Q_O[\mathbf{o}_{t+1}, \mathbf{s}_{t+1}]}{\partial \theta_a} \\
=& \sum_{\mathrm{k}=0}^{\infty} \sum_{\mathbf{s}_{t+k}, \mathbf{o}_{t+k}} P_\gamma^{(k)}(\mathbf{s}_{t+k}, \mathbf{o}_{t+k}|\mathbf{s}_t, \mathbf{o}_t) \\
& \sum_{\mathbf{a}_{t+k}} \frac{\partial P(\mathbf{a}_{t+k}|\mathbf{s}_{t+k}, \mathbf{o}_{t+k})}{\partial \theta_a} Q_U(\mathbf{s}_{t+k}, \mathbf{o}_{t+k}, \mathbf{a}_{t+k}).
\end{aligned}$$

□

Proposition A.4.4.2. *MDP formulation has identical Termination Policy Gradient with SMDP formulation in (Bacon et al., 2017).*

$$\frac{\partial U[\mathbf{s}_{t+1}, \mathbf{o}_t]}{\partial \theta_b} = -\sum_{\mathrm{k}=0}^{\infty} \sum_{\mathbf{s}_{t+1+k}, \mathbf{o}_{t+k}} P_\gamma^{(k)}(\mathbf{s}_{t+1+k}, \mathbf{o}_{t+k}|\mathbf{s}_{t+1}, \mathbf{o}_t) \frac{\partial \beta_{t+1+k}}{\partial \theta_b} A[\mathbf{s}_{t+k+1}, \mathbf{o}_{t+k+1} = \mathbf{o}_{t+k}]. \tag{36}$$

Proof. We first show the gradient of θ_b with respect to equation (21) and (25) separately:

$$\frac{\partial P(\mathbf{o}_{t+1}|\mathbf{s}_{t+1}, \mathbf{o}_t)}{\partial \theta_b} = \big[P(\mathbf{o}_{t+1}|\mathbf{s}_{t+1}) - \mathbf{1}_{\mathbf{o}_t = \mathbf{o}_{t-1}}\big] \frac{\partial \beta_{t+1}}{\partial \theta_b} \tag{37}$$

$$\begin{aligned}
\frac{\partial Q_O[\mathbf{o}_{t+1}, \mathbf{s}_{t+1}]}{\partial \theta_b} =& \sum_{\mathbf{a}_{t+1}} P(\mathbf{a}_{t+1}|\mathbf{s}_{t+1}, \mathbf{o}_{t+1}) \sum_{\mathbf{s}_{t+2}} P(\mathbf{s}_{t+2}|\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) \frac{\partial U[\mathbf{s}_{t+2}, \mathbf{o}_{t+1}]}{\partial \theta_b} \\
=& \sum_{\mathbf{s}_{t+2}} P(\mathbf{s}_{t+2}|\mathbf{s}_{t+1}, \mathbf{o}_{t+1}) \frac{\partial U[\mathbf{s}_{t+2}, \mathbf{o}_{t+1}]}{\partial \theta_b}.
\end{aligned} \tag{38}$$

The equation (36) is a direct result by taking gradient of θ_b with respect to equation (34) and using above results:

$$
\begin{aligned}
\frac{\partial U[\mathbf{s}_{t+1}, \mathbf{o}_t]}{\partial \theta_b} =& \sum_{\mathbf{o}_{t+1}} \frac{\partial P(\mathbf{o}_{t+1}|\mathbf{o}_t, \mathbf{s}_{t+1})}{\partial \theta_b} Q_O[\mathbf{o}_{t+1}, \mathbf{s}_{t+1}] + \sum_{\mathbf{o}_{t+1}} P(\mathbf{o}_{t+1}|\mathbf{o}_t, \mathbf{s}_{t+1}) \frac{\partial Q_O[\mathbf{o}_{t+1}, \mathbf{s}_{t+1}]}{\partial \theta_b} \\
=& \sum_{\mathbf{o}_{t+1}} \big[P(\mathbf{o}_{t+1}|\mathbf{s}_{t+1}) - \mathbf{1}_{\mathbf{o}_t = \mathbf{o}_{t-1}}\big] Q_O[\mathbf{o}_{t+1}, \mathbf{s}_{t+1}] \frac{\partial \beta_{t+1}}{\partial \theta_b} \\
&+ \sum_{\mathbf{o}_{t+1}} P(\mathbf{o}_{t+1}|\mathbf{o}_t, \mathbf{s}_{t+1}) \gamma \sum_{\mathbf{s}_{t+2}} P(\mathbf{s}_{t+2}|\mathbf{s}_{t+1}, \mathbf{o}_{t+1}) \frac{\partial U[\mathbf{s}_{t+2}, \mathbf{o}_{t+1}]}{\partial \theta_b} \\
=& \big[V[\mathbf{s}_{t+1}] - Q_O[\mathbf{o}_{t+1} = \mathbf{o}_t, \mathbf{s}_{t+1}]\big] \frac{\partial \beta_{t+1}}{\partial \theta_b} \\
&+ \gamma \sum_{\mathbf{o}_{t+1}, \mathbf{s}_{t+2}} P(\mathbf{s}_{t+2}, \mathbf{o}_{t+1}|\mathbf{s}_{t+1}, \mathbf{o}_t) \frac{\partial U[\mathbf{s}_{t+2}, \mathbf{o}_{t+1}]}{\partial \theta_b} \\
=& - A[\mathbf{o}_{t+1} = \mathbf{o}_t, \mathbf{s}_{t+1}] \frac{\partial \beta_{t+1}}{\partial \theta_b} + \gamma \sum_{\mathbf{o}_{t+1}, \mathbf{s}_{t+2}} P(\mathbf{s}_{t+2}, \mathbf{o}_{t+1}|\mathbf{s}_{t+1}, \mathbf{o}_t) \frac{\partial U[\mathbf{s}_{t+2}, \mathbf{o}_{t+1}]}{\partial \theta_b} \\
=& - \sum_{\mathrm{k}=0}^{\infty} \sum_{\mathbf{s}_{t+1+k}, \mathbf{o}_{t+k}} P_\gamma^{(k)}(\mathbf{s}_{t+1+k}, \mathbf{o}_{t+k}|\mathbf{s}_{t+1}, \mathbf{o}_t) \frac{\partial \beta_{t+1+k}}{\partial \theta_b} A[\mathbf{s}_{t+k+1}, \mathbf{o}_{t+k+1} = \mathbf{o}_{t+k}].
\end{aligned}
$$

□

A.5 DERIVATIONS OF THE SKILL-ACTION ARCHITECTURE'S VALUE FUNCTIONS

Following Bishop (2006)'s notation, we use A, B and C to denote three non-overlapping sets of arbitrarily many random variables. Sets A and B are conditional independent on set C if $P(A, B|C) = P(A|C)P(B|C)$, denoted as $A \perp\!\!\!\perp B \mid C$. We mainly use head-to-tail conditional independence properties (Chapter 8.2.1 (Bishop, 2006)) in this section.

Derivations of Eq. (5):

$$\begin{aligned} V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}] =& \mathbb{E}[G_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}] \\ =& \sum_{\hat{\mathbf{o}}_t} P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})\mathbb{E}(G_t|\mathbf{s}_t, \hat{\mathbf{o}}_t, \hat{\mathbf{o}}_{t-1}) \\ =& \sum_{\hat{\mathbf{o}}_t} P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})\mathbb{E}[G_t|\mathbf{s}_t, \hat{\mathbf{o}}_t] \\ =& \sum_{\hat{\mathbf{o}}_t} P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})Q_O[\hat{\mathbf{o}}_t, \mathbf{s}_t], \end{aligned}$$

where from line 2 to line 3 we use the conditional independence property in PGM that $G_t \perp\!\!\!\perp \hat{\mathbf{o}}_{t-1}|\{\mathbf{s}_t, \hat{\mathbf{o}}_t\}$.

Proof. for Proposition 3.4: By law of total expectation:

$$\mathbb{E}_{\hat{\mathbf{o}}_{t-1}}[V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]] = \mathbb{E}_{\hat{\mathbf{o}}_{t-1}}[\mathbb{E}[G_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]] = \mathbb{E}[G_t|\mathbf{s}_t] = V[\mathbf{s}_t]$$

thus $V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]$ is an unbiased estimator of $V[\mathbf{s}_t]$. □

Proof. for Proposition 3.5: By law of total conditional variance:

$$\begin{aligned} \text{Var}(V[\mathbf{s}_t]) &= \text{Var}([\mathbb{E}[G_t|\mathbf{s}_t]]) = \mathbb{E}[\text{Var}(\mathbb{E}[G_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}])|\mathbf{s}_t] + \text{Var}(\mathbb{E}[\mathbb{E}[G_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]]|\mathbf{s}_t) \\ &= \mathbb{E}[\text{Var}(V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}])|\mathbf{s}_t] + \text{Var}(\mathbb{E}[V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]]|\mathbf{s}_t) \\ &\geq \text{Var}(\mathbb{E}[V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]]|\mathbf{s}_t). \end{aligned}$$

□

Derivations of Eq. (7)

$$\begin{aligned} Q_A[\mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t] =& \mathbb{E}[G_t|\mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t] = \mathbb{E}[R_{t+1} + \gamma G_{t+1}|\mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t] \\ =& r(s, o, a) + \gamma \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t)\mathbb{E}[G_{t+1}|\mathbf{s}_{t+1}, \mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t] \\ =& r(s, a) + \gamma \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\mathbb{E}[G_{t+1}|\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t] \\ =& r(s, a) + \gamma \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)V[\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t], \end{aligned}$$

where from line 2 to line 3 we use the conditional independence property in PGM that $R_{t+1} \perp\!\!\!\perp \hat{\mathbf{o}}_t|\mathbf{a}_t$, $G_{t+1} \perp\!\!\!\perp \mathbf{s}_t|\{\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t\}$ and $G_{t+1} \perp\!\!\!\perp \mathbf{a}_t|\mathbf{s}_{t+1}$. $\gamma \in \mathbb{R}$ is a discounting factor.

A.6 PROOFS FOR THE SKILL-ACTION ARCHITECTURE GRADIENT THEOREMS

A.6.1 PROOF FOR THE SKILL POLICY GRADIENT THEOREM

Proof.

$$\begin{aligned}
\frac{\partial Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t]}{\partial \theta_o} &= \sum_{\mathbf{a}_t} P(\mathbf{a}_t|\mathbf{s}_t, \hat{\mathbf{o}}_t)\big[r(s,a) + \gamma \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)\frac{\partial V[\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t]}{\partial \theta_o}\big] \\
&= \sum_{\mathbf{s}_{t+1}} \gamma P(\mathbf{s}_{t+1}|\mathbf{s}_t, \hat{\mathbf{o}}_t)\frac{\partial V[\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t]}{\partial \theta_o} \\
\frac{\partial V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]}{\partial \theta_o} &= \sum_{\hat{\mathbf{o}}_t} \frac{\partial P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})}{\partial \theta_o} Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t] + \gamma \sum_{\hat{\mathbf{o}}_t} P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})\frac{Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t]}{\partial \theta_o} \\
&= \sum_{\hat{\mathbf{o}}_t} \frac{\partial P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})}{\partial \theta_o} Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t] + \gamma \sum_{\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t} P(\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})\frac{\partial V[\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t]}{\partial \theta_o} \\
&= -\sum_{\mathrm{k}=0}^{\infty} \sum_{\mathbf{s}_{t+k}, \hat{\mathbf{o}}_{t+k-1}} \\
&\quad P_\gamma^{(k)}(\mathbf{s}_{t+k}, \hat{\mathbf{o}}_{t+k-1}|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}) \sum_{\hat{\mathbf{o}}_{t+k}} \frac{\partial P(\hat{\mathbf{o}}_{t+k}|\mathbf{s}_{t+k}, \hat{\mathbf{o}}_{t+k-1})}{\partial \theta_o} Q_O[\mathbf{s}_{t+k}, \hat{\mathbf{o}}_{t+k}] \\
&= \mathbb{E}[\ \frac{\partial P(\mathbf{o}'|\mathbf{s}', \mathbf{o})}{\partial \theta_o} Q_O[\mathbf{s}', \mathbf{o}'] \mid \mathbf{s}_t, \hat{\mathbf{o}}_{t-1}].
\end{aligned}$$

□

A.6.2 PROOF FOR THE ACTION POLICY GRADIENT THEOREM

Proof. Similar to the first equation above, continue expanding gradients of $\frac{\partial Q_O}{\partial \theta_a}$ by equations (5) (6) and (7):

$$\begin{aligned}
\frac{\partial Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t]}{\partial \theta_a} &= \sum_{\mathbf{a}_t} \frac{\partial P(\mathbf{a}_t|\mathbf{s}_t, \hat{\mathbf{o}}_t)}{\partial \theta_a} Q_A[\mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t] + \gamma \sum_{\mathbf{s}_{t+1}} P(\mathbf{s}_{t+1}|\mathbf{s}_t, \hat{\mathbf{o}}_t)\frac{\partial V[\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t]}{\partial \theta_a} \\
&= \sum_{\mathbf{a}_t} \frac{\partial P(\mathbf{a}_t|\mathbf{s}_t, \hat{\mathbf{o}}_t)}{\partial \theta_a} Q_A[\mathbf{s}_t, \hat{\mathbf{o}}_t, \mathbf{a}_t] + \gamma \sum_{\mathbf{s}_{t+1}, \hat{\mathbf{o}}_{t+1}} P(\mathbf{s}_{t+1}, \hat{\mathbf{o}}_{t+1}|\mathbf{s}_t, \hat{\mathbf{o}}_t)\frac{\partial Q_O[\mathbf{s}_{t+1}, \hat{\mathbf{o}}_{t+1}]}{\partial \theta_a} \\
&= -\sum_{\mathrm{k}=0}^{\infty} \sum_{\mathbf{s}_{t+k}, \hat{\mathbf{o}}_{t+k}} \\
&\quad P_\gamma^{(k)}(\mathbf{s}_{t+k}, \hat{\mathbf{o}}_{t+k}|\mathbf{s}_t, \hat{\mathbf{o}}_t) \sum_{\mathbf{a}_{t+k}} \frac{\partial P(\mathbf{a}_{t+k}|\mathbf{s}_{t+k}, \hat{\mathbf{o}}_{t+k})}{\partial \theta_a} Q_A[\mathbf{s}_{t+k}, \hat{\mathbf{o}}_{t+k}, \mathbf{a}_{t+k}] \\
&= \mathbb{E}[\ \frac{\partial P(\mathbf{a}_{t+k}|\mathbf{s}_{t+k}, \hat{\mathbf{o}}_{t+k})}{\partial \theta_a} Q_A[\mathbf{s}_{t+k}, \hat{\mathbf{o}}_{t+k}, \mathbf{a}_{t+k}] \mid \mathbf{s}_t, \hat{\mathbf{o}}_t].
\end{aligned}$$

□

A.7 LEARNING ALGORITHM FOR THE SKILL-ACTION ARCHITECTURE

Algorithm 1: Learning Algorithm for the Skill-Action architecture

Initialize the skill embedding matrix $\boldsymbol{W}_S$
Assign Initial State: $\mathbf{s}_t \leftarrow \mathbf{s}_0$
Assign Initial Skill: $\hat{\mathbf{o}}_{t-1} \leftarrow \hat{\mathbf{o}}_0$

while *Converge* **do**
  # Rollout trajectories and store in replay buffer
  repeat
    Retrieve the skill context vector $\hat{\mathbf{o}}_{t-1} = \boldsymbol{W}_S^T \cdot \hat{\mathbf{o}}_{t-1}$
    Sample $\hat{\mathbf{o}}_t \sim P(\hat{\mathbf{o}}_t|\mathbf{s}_t, \hat{\mathbf{o}}_{t-1})$
    Retrieve the skill context vector $\hat{\mathbf{o}}_t = \boldsymbol{W}_S^T \cdot \hat{\mathbf{o}}_t$
    Sample $\mathbf{a}_t \sim P(\mathbf{a}_t|\mathbf{s}_t, \hat{\mathbf{o}}_t)$
    Compute $Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t]$ and $V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]$
    Take action $\mathbf{a}_t$ in $\mathbf{s}_t$, observe new state $\mathbf{s}_{t+1}$ and reward R_{t+1}
  until *Rollout Length Reached*

  # Compute Advantages for skill & action policies
  Assign t reversely, from $RolloutLength - 1$ to 1
  repeat
    Compute skill Advantage $A_t^O = R_{t+1} + \gamma(V[\mathbf{s}_{t+1}, \hat{\mathbf{o}}_t] - V[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]) + \gamma\lambda A_{t+1}^O$
    Compute action Advantage $A_t^A = R_{t+1} + \gamma(Q_O[\mathbf{s}_{t+1}, \hat{\mathbf{o}}_{t+1}] - Q_O[\mathbf{s}_t, \hat{\mathbf{o}}_t]) + \gamma\lambda A_{t+1}^A$
  until *Rollout Length Reached*

  # λ is the GAE coefficient used in PPO.
  # Optimize PPO Obj
  while $i <$ *PPO Optimization Epochs* **do**
    $\theta_o \leftarrow PPO(\frac{\partial P(\mathbf{o}'|\mathbf{s}', \mathbf{o})}{\partial \theta_o}, A^O)$
    $\theta_a \leftarrow PPO(\frac{\partial P(\mathbf{a}|\mathbf{s}, \mathbf{o})}{\partial \theta_a}, A^A)$
  end
end

A.8 IMPLEMENTATION DETAILS

In this section we summarize our implementation details. For a fair comparison, all baselines: DAC+PPO (Zhang & Whiteson, 2019), AHP+PPO (Levy & Shimkin, 2011), PPOC (Klissarov et al., 2017), OC (Bacon et al., 2017) and PPO (Schulman et al., 2017) are from DAC's open source Github repo: https://github.com/ShangtongZhang/DeepRL/tree/DAC. Hyper-parameters used in DAC (Zhang & Whiteson, 2019) for all these baselines are kept unchanged.

SA Architecture: For all experiments, our implementation of SA is exactly the same as Figure 1 (b). We use Pytorch to build neural networks. Specifically, for skill policy module, we use a skill context matrix $\boldsymbol{W}_S \in \mathbb{R}^{4\times 40}$ which has 4 skills (4 rows) and an embedding size of 40 (40 columns). For Multi-Head Attention, we use Pytorch's built-in MultiheadAttention function[9] with $num\_heads = 1$ and $embed\_dim = 40$. For layer normalization we use Pytorch's built-in function LayerNorm [10]. For Feed Forward Networks (FNN), we use a 2 layer FNN with ReLu function as activation function with input size of 40, hidden size of 64, and output size of 64 neurons. For Linear layer, we use built-in Linear function[11] to map FFN's outputs to 4 dimension. Each dimension acts like a logit for each skill and is used as density in Categorical distribution[12]. For both action policy and critic module, FFNs are of the same size as the one used in the skill policy.

[9] https://pytorch.org/docs/stable/generated/torch.nn.MultiheadAttention.html
[10] https://pytorch.org/docs/stable/generated/torch.nn.LayerNorm.html
[11] https://pytorch.org/docs/stable/generated/torch.nn.Linear.html
[12] https://github.com/pytorch/pytorch/blob/master/torch/distributions/categorical.py

Preprocessing: States are normalized by a running estimation of mean and std.

Hyperparameters of PPO: For a fair comparison, we use exactly the same parameters of PPO as DAC. Specifically:

- Optimizer: Adam with $\epsilon = 105$ and an initial learning rate 3×10^4
- Discount ratio γ: 0.99
- GAE coefficient: 0.95
- Gradient clip by norm: 0.5
- Rollout length: 2048 environment steps
- Optimization epochs: 10
- Optimization batch size: 64
- Action probability ratio clip: 0.2

Computing Infrastructure: We conducted our experiments on an Intel Core i9-9900X CPU @ 3.50GHz with a single thread and process with PyTorch.

A.9 Multi-Head Attention (MHA) Mechanism

Specifically, an attention mechanism is described as the mapping from a query $\mathbf{q} \in \mathbb{R}^E$ and a set of key-value pairs, i.e., $\boldsymbol{K} \in \mathbb{R}^{M \times E}$ and $\boldsymbol{V} \in \mathbb{R}^{M \times E}$ (M and E are total number of skills and embedding dimensions defined in section 3.1), to an output:

$$Attention(\mathbf{q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}(\frac{\mathbf{q}\boldsymbol{K}^T}{\sqrt{E}})\boldsymbol{V} \tag{39}$$

A Multi-Head Attention $\text{MHA}(\mathbf{q}, \boldsymbol{K}, \boldsymbol{V})$ is a linear projection of h (number of heads) concatenated linearly projected $Attention$ outputs:

$$\begin{aligned} \text{MHA}(\mathbf{q}, \boldsymbol{K}, \boldsymbol{V}) &= \text{Concat}[\text{head}_1, \ldots, \text{head}_h]\boldsymbol{W}^H \\ \text{where head}_i &= Attention(\mathbf{q}\boldsymbol{W}_i^q, \boldsymbol{K}\boldsymbol{W}_i^K, \boldsymbol{V}\boldsymbol{W}_i^V) \end{aligned} \tag{40}$$

where projections are parameter matrices $\boldsymbol{W}_i^q \in \mathbb{R}^{E \times E}$, $\boldsymbol{W}_i^K \in \mathbb{R}^{E \times E}$, $\boldsymbol{W}_i^V \in \mathbb{R}^{E \times E}$, $\boldsymbol{W}_i^O \in \mathbb{R}^{hE \times E}$. In this paper we use MHA as one building block as illustrated in Figure 1.