Option2Vec: Learning Temporal-State Abstraction Embeddings on the MDP

Anonymous Author(s)
Affiliation
Address
email

Abstract

Temporally extended actions refer to actions that last for multiple time steps. The option framework provides a natural scheme to incorporate extended actions into reinforcement learning. Learning an option, however, requires modeling the initiation set, intra-option policy, and termination condition, respectively, which is sub-optimal for knowledge representation and is computationally expensive. To address these issues, we consider temporal and state abstraction jointly. In particular, we formulate the option framework as a Hidden-Markov-Model-style Probabilistic Graphical Model (PGM), thus enabling each option to be parameterized as a compact embedding vector. To optimize this PGM, we first propose a novel *Markovian Option-Value function*, and prove it is an unbiased estimation of the conventional value function. We then derive a two-stage policy gradient theorem based on the above value function. Finally, we implement this PGM upon the Transformer architecture to encode options into fixed-length embeddings, named Option2Vec. Extensive experiments on challenging *Mujoco* environments demonstrate Option2Vec's efficiency and effectiveness: under widely used configuration, with merely 15.8% parameters, Option2Vec achieves competitive, if not better, performance compared to the state-of-the-art baselines on all finite horizon and transfer learning environments. Moreover, Option2Vec significantly outperforms all baselines on infinite horizon environments while exhibiting smaller variance, faster convergence, and interpretability.

1 Introduction

Human is capable of scheduling daily tasks from macro timescale to micro timescale [36]. In reinforcement learning, the capability of modeling courses behind such temporal hierarchies of long-term actions is referred to as temporal abstraction. The option framework [31] is a popular solution that addresses this problem by formulating temporal abstractions as *options*. In previous works [31, 38, 13, 6, 18, 3], options are defined as abstract high-level actions, whose executions are temporally extended to a variable amount of time. Although such a definition is straightforward, learning an option will require specifying the *initiation set*, *intra-option policy*, and *termination function* respectively, which can be less effective for knowledge representation [2] and computationally expensive [11, 38, 20].

Consider an example of training robots to cook mashed potatoes. The process typically consists of three stages: washing potatoes, boiling potatoes, and mashing potatoes. Each stage persists for a variable length of time. Learning options as high-level actions is analogous to training three robots, *i.e.*, three distinct action policies with each policy dedicated to one specific stage. Once an option (the stage) is selected, its *intra-option policy* will be trained independently and only be locally available

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

to the current stage. Because of this, neither knowledge nor parameters are shared among options which is clearly inefficient for computation.

Instead of representing options as temporally extended high-level actions, we turn to represent options as hidden variables, *i.e.*, courses of temporally extended actions, which are invariant of insignificant temporal and state transitions. Inspired by word embeddings in Word2Vec [21], we parameterize such hidden variables as option embeddings (*i.e.*, semantic representation of options), which the action policy is trained to decode. In the above robotic example, learning options as embeddings is analogous to learning three instruction rules (encoded by the embeddings) for three stages, respectively. Each rule describes at that stage what actions to emit, when to stop, and which rule should be executed next. Instead of training three dedicated robots, we only need to train one robot, *i.e.*, action policy, which can read abstract rules and decode them into actual actions. Under this formulation, knowledge of options is compactly encoded into embeddings. Computational cost can also be significantly reduced by sharing one action policy to decode all different option embeddings.

Learning temporal-state abstraction embeddings of options in the context of reinforcement learning, however, is challenging since it requires developing a novel Markov Decision Process (MDP) and derive corresponding optimization algorithms. To solve this problem, we first reformulate the option framework as a Hidde-Markov-Model-style [4] Probabilistic Graphical Model (PGM), in which option embeddings are parameters of a mixture distribution. Although similar formulations have been employed as "one-step option" in previous works [10, 28, 27, 17, 38], we first identify that under this setting the conventional *Value Function* no longer yields the Bellman equation [30]. Instead, we propose a novel *Markovian Option-Value Function*, prove that it is an unbiased estimation of the conventional value function, and derive a novel Bellman equation. Based on the Bellman equation, we then derive an efficient two-stage policy gradient theorem [30] which can be combined with any MDP-style [38] policy optimization algorithms (such as PPO [35]) for learning options. Finally, inspired by Transformer [34], we implement this PGM as the Option2Vec, a simple yet effective Attention based Encoder-Decoder architecture. Empirical studies on challenging locomotion environments demonstrate Option2Vec's efficiency and effectiveness. In the absence of termination function, the embedding representations of options significantly outperform its counterparts in all environments. Under widely used configuration, with merely 15.8% parameters, Option2Vec achieves competitive, if not better, performance compared to the state-of-the-art baselines on all finite horizon and transfer learning environments. Moreover, Option2Vec significantly outperforms all baselines on infinite horizon environments while exhibiting smaller variance and faster convergence. We also show that option embedding is interpretable, which is an important property (*e.g.* ensuring safety to human) for applying RL agents in real-world applications.

2 Background

A Markov Decision Process [23] $M = \{\mathbb{S}, \mathbb{A}, R, P, \gamma\}$ consists of a state space $\mathbb{S}$, an action space $\mathbb{A}$, a state transition function $P(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) : \mathbb{S} \times \mathbb{A} \to \mathbb{S}$, a discount factor $\gamma \in \mathbb{R}$, and a reward function $R(\mathbf{s}, \mathbf{a}) = \mathbb{E}[r|\mathbf{s}, \mathbf{a}] : \mathbb{S} \times \mathbb{A} \to \mathbb{R}$ which is the expectation of the reward $r \in \mathbb{R}$ received from the environment after executing action $\mathbf{a}_t$ at state $\mathbf{s}_t$. A policy $\pi = P(\mathbf{a}|\mathbf{s}) : \mathbb{A} \times \mathbb{S} \to [0, 1]$ is a probability distribution defined over actions conditioning on states. A discounted return is defined as $G_t = \sum_k^N \gamma^k r_{t+k+1}$, where $\gamma \in (0, 1)$ is a discounting factor. The value function $V[\mathbf{s}_t] = \mathbb{E}_{\tau\sim\pi}[G_t|\mathbf{s}_t]$ is the expected return starting at state $\mathbf{s}_t$ and the trajectory $\tau = \{\mathbf{s}_t, \mathbf{a}_t, r_{t+1}, \mathbf{s}_{t+1}, \dots\}$ follows policy π thereafter. The action-value function is defined as $Q[\mathbf{s}_t, \mathbf{a}_t] = \mathbb{E}_{\tau\sim\pi}[G_t|\mathbf{s}_t, \mathbf{a}_t]$.

The option framework [31] enables learning temporal abstractions by introducing a set of options on the MDP. We use the bold-case $\mathbf{o}$ to denote random variables and the light-italic-case o to denote a realized instantiation. In the option framework, options are defined as high-level actions whose executions can extend to a variable amount of time and are developed on a Semi-Markov Decision Process (SMDP). Specifically, in a set of options $\Omega = \{\omega_1, \omega_2, ...\}$, each *option* is a triple $\omega_o = (\mathbb{I}_o, P_o(\mathbf{a}|\mathbf{s}), P_o(\mathrm{b}|\mathbf{s}))$, where the subscript $o \in \mathbb{O} = \{1, 2, \dots, K\}$ is an integer *option index* and K is the number of options; $\mathbb{I}_o$ is the *initiation set*, it defines a subset of state space $\mathbb{I}_o \subseteq \mathbb{S}$ where ω_o can be initiated; $P_o(\mathbf{a}|\mathbf{s}) : \mathbb{A} \times \mathbb{S} \to [0, 1]$ is the *intra-option policy* from which actions are sampled; $P_o(\mathbf{b}|\mathbf{s}) : \mathbb{S} \to [0, 1]$ is the *termination function* where $\mathbf{b} \in \{0, 1\}$ is a binary variable, $\beta_o = P_o(\mathbf{b} = 1|\mathbf{s})$ is the probability to terminate ω_o. We refer to this formulation as $Option^{\text{Triple}}$.

The execution of an option is temporally extended to variable amount of time by following a *call-and-return* mode: at the start of an execution, given state $\mathbf{s}_t$, an *option index* o is selected according to the *master policy* $P(\mathbf{o}|\mathbf{s})$ where $\mathbf{o} \in \mathbb{O}$ is used to sample the index of which option ω_o to be executed,

then actions are sampled by following the activated option ω_o's *intra-option policy* $\mathbf{a} \sim P_o(\mathbf{a}|\mathbf{s})$ thereafter, until at some time the *termination function* determines to stop $P_o(\mathrm{b} = 1|\mathbf{s})$ and repeats this procedure all over again. We use $\tau = \{\mathbf{s}_0, \mathbf{o}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{o}_1, \mathbf{a}_1, \ldots\}$ to denote the trajectory of the option framework and $\mathbf{1}$ is an indicator function and is only true when $\mathbf{o}_t = o_{t-1}$ (notice that o_{t-1} is a realization at $\mathbf{o}_{t-1}$). Dynamics of the option framework can be written as:

$$P(\tau) = P(\mathbf{s}_0)P(\mathbf{o}_0)P_{o_0}(\mathbf{a}_0|\mathbf{s}_0)\prod_{t=1}^{\infty} P(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_{t-1})P_{o_t}(\mathbf{a}_t|\mathbf{s}_t) \\ [P_{o_{t-1}}(\mathbf{b}_t = 0|\mathbf{s}_t)\mathbf{1}_{\mathbf{o}_t = o_{t-1}} + P_{o_{t-1}}(\mathbf{b}_t = 1|\mathbf{s}_t)P(\mathbf{o}_t|\mathbf{s}_t)]. \quad (1)$$

Although easy to understand, $Option^{\text{Triple}}$ is less effective in representing knowledge compactly: since each option is trained distinctively, there is no knowledge nor parameters sharing between options, and thus is computationally expensive to learn. In this paper, we address this issue by encoding knowledge of options into more compact and effective representations, *i.e.*, option embeddings.

3 Methodology

As explained in Section 2, options are defined as high-level actions that are temporally extended on the SMDP. In this section, we propose a novel perspective of options, *i.e.*, redefining options as hidden variables whose values are invariant of insignificant temporal-state transitions, and represent options as embeddings (referred to as $Option^{\text{Embed}}$) on which the action policy is conditioned. For this purpose, we need to develop a novel Decision Process which enables representing knowledge of an option (*e.g.*, where to initiate, what actions to emit and when to terminate) as a semantic vector, and then derive optimization algorithms for learning this Decision Process. We achieve these in three steps. In Section 3.1, we first reformulate the option framework as a Hidden-Markov-Model-style (HMM-style) Probabilistic Graphical Model (PGM) [15] as shown in Figure 1, and define an *option policy* as the mixture distribution over hidden variables. In Section 3.2, we show how to encode knowledge of options into embeddings, *i.e.*, parameters of the *option policy*. In Section 3.3, we define the Decision Process based on the PGM and derive optimization algorithms.

It is worth mentioning that, although similar HMM-style formulations have been employed in earlier works [10, 28, 27, 17, 38] as "one-step option", our work is the first one which discovers that under this setting, conventional *Value Function* $V[\mathbf{s}_t]$ no longer yields the Bellman equation [30]. Instead, we propose a novel *Markovian Option-Value Function* $V[\mathbf{s}_t, \bar{\mathbf{o}}_t]$ and prove that it is an unbiased estimation of $V[\mathbf{s}_t]$, whose variance is up-bounded by $V[\mathbf{s}_t]$. We further derive policy gradient theorems and show that our theorems follow a two-stage optimization scheme [38], which allows us to employ sample efficient MDP-style policy optimization algorithms (*e.g.*, PPO [35]).

3.1 An HMM-style Formulated Option Framework

In this subsection we show how to reformulate the option framework defined in Section 2 as an HMM-style PGM (Figure 1), and enable representing option embeddings as parameters of the hidden variables' mixture distribution. We first follow Bishop [4]'s formulation of mixture distributions (Chapter 9) and redefine the integer *option index* $\mathbf{o} \in \mathbb{O}$ as a K-dimensional one-hot vector $\bar{\mathbf{o}} \in \mathbb{O} = \{0, 1\}^K$, where K is the number of options. For $\bar{\mathbf{o}}_t = o_t$, by definition only the entry $o_t = 1$ and all the other entries of $\bar{\mathbf{o}}_t$ are 0. We then reformulate *intra-option policy* of all options $\omega \in \Omega$ as a mixture distribution by introducing an extra dependency on $\bar{\mathbf{o}}$:

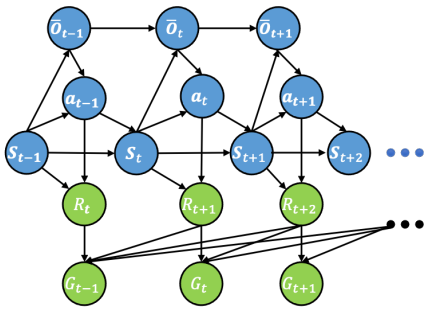


Figure 1: PGM for Option2Vec.

$$P(\mathbf{a}_t|\mathbf{s}_t, \bar{\mathbf{o}}_t) = \prod_{o \in \bar{\mathbf{o}}_t} P_o(\mathbf{a}_t|\mathbf{s}_t)^o, \quad (2)$$

such that $P_{o_t}(\mathbf{a}_t|\mathbf{s}_t) = P(\mathbf{a}_t|\mathbf{s}_t, \bar{\mathbf{o}}_t = o_t)$. For the same reason, we can reformulate the *termination function* as $P(\mathbf{b}_t|\mathbf{s}_t, \bar{\mathbf{o}}_{t-1}) = \prod_{o \in \bar{\mathbf{o}}_{t-1}} P_o(\mathrm{b}_t|\mathbf{s}_t)^o$. We reformulate the indicator function $\mathbf{1}_{\mathbf{o}_t = o_{t-1}}$ in Eq. (1) is now a degenerate probability distribution [23] $P(\bar{\mathbf{o}}_t|\bar{\mathbf{o}}_{t-1}) = \begin{cases} 1 & \text{if } \bar{\mathbf{o}}_t = \bar{\mathbf{o}}_{t-1} \\ 0 & \text{if } \bar{\mathbf{o}}_t \neq \bar{\mathbf{o}}_{t-1} \end{cases}$ and extend the *master policy* $P(\mathbf{o}_t|\mathbf{s}_t)$ in Eq. (1) to an HMM-style *mixture master policy* $P(\bar{\mathbf{o}}_t|\mathbf{s}_t, \mathbf{b}_t, \bar{\mathbf{o}}_{t-1}) = P(\bar{\mathbf{o}}_t|\mathbf{s}_t)^{\mathrm{b}_t} P(\bar{\mathbf{o}}_t|\bar{\mathbf{o}}_{t-1})^{1-\mathrm{b}_t}$ with $\mathbf{b}_t$ and $\bar{\mathbf{o}}_{t-1}$ as hidden variables. We use

$\bar{\tau} = \{\mathbf{s}_0, \bar{\mathbf{o}}_0, \mathbf{a}_0, \mathbf{s}_1, \bar{\mathbf{o}}_1, \mathbf{a}_1, \ldots\}$ to denote the trajectory. Substituting above terms into Eq. (1), the option framework is now simplified as a HMM-style PGM (Figure 1):

$$
\begin{aligned}
P(\bar{\tau}) &= P(\mathbf{s}_0)P(\bar{\mathbf{o}}_0)P(\mathbf{a}_0|\mathbf{s}_0, \bar{\mathbf{o}}_0) \prod_{t=1}^{\infty} P(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_{t-1}) P(\mathbf{a}_t|\mathbf{s}_t, \bar{\mathbf{o}}_t) \\
&\quad \sum_{\mathbf{b}_t} P(\mathbf{b}_t|\mathbf{s}_t, \bar{\mathbf{o}}_{t-1}) P(\bar{\mathbf{o}}_t|\mathbf{b}_t, \mathbf{s}_t, \bar{\mathbf{o}}_{t-1}) \\
&= P(\mathbf{s}_0)P(\bar{\mathbf{o}}_0)P(\mathbf{a}_0|\mathbf{s}_0, \bar{\mathbf{o}}_0) \prod_{t=1}^{\infty} P(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_{t-1}) P(\mathbf{a}_t|\mathbf{s}_t, \bar{\mathbf{o}}_t) P(\bar{\mathbf{o}}_t|\mathbf{s}_t, \bar{\mathbf{o}}_{t-1}) \qquad (3)
\end{aligned}
$$

Note that the *termination function* diminishes as a natural result of the marginalization over the termination variable $\mathbf{b}_t$. Eq. (3) suggests replacing the *termination function* and *master policy* with a single new term $P(\bar{\mathbf{o}}_t|\mathbf{s}_t, \bar{\mathbf{o}}_{t-1})$ which is referred as *option policy* in our paper. Since the *termination function* is executed at every time step, this replacement will not introduce extra computational cost. However, while the *option policy* is responsible for both selecting options and temporally extending an option's execution, the one-hot vector $\bar{\mathbf{o}}_{t-1}$ contains little context information for the *option policy* to exploit. Although earlier works [10, 28, 27, 17, 38] have employed "one-step option" as $P(\mathbf{o}_t|\mathbf{s}_t, \mathbf{o}_{t-1})$ in derivations of Eq. (1), the integer *option index* $\mathbf{o}$ also contains little information, and thus as shown in Section 5.1, in their applications the *termination function* and *master policy* are still implemented for better empirical performance. To address this problem, in Section 3.2 we represent options as compact real-valued vectors, *i.e.*, option embeddings.

3.2 Representing Options as Temporal-State Abstraction Embeddings

In this subsection we answer two questions: (1) how to encode knowledge of $Option^{\text{Triple}}$ with option embeddings, *i.e.*, $Option^{\text{Embed}}$; and (2) how to achieve temporal abstractions by replacing the *call-and-return* mode with a simple *clustering* mechanism. We argue that for learning temporal abstractions, high-level actions, *i.e.*, $Option^{\text{Triple}}$, and the *call-and-return* mode are not mandatory: temporal abstractions can be achieved by replacing $Option^{\text{Triple}}$ with the a newly defined temporal-state transition-invariant embeddings of hidden variables on which the action policy is conditioned.

We start by reviewing the weaknesses of defining options as high-level actions. Specifically, each $Option^{\text{Triple}}$, *i.e.*, a set of distributions, is analogously a point on a statistical manifold [1], which is computationally expensive to learn. Most literatures ignore *the initiation* set because of difficulties in learning it from data [12]. Learning *master policy* is analogously learning classification hyperplanes [20] on the statistical manifold. For M hyperplanes, theoretically there are 2^M options to be learned as shown in Figure 2 (left). As pointed out by Bacon [2] (Chapter 3.6), $Option^{\text{Triple}}$ cannot at represent knowledge compactly and requires more samples to achieve good performance.

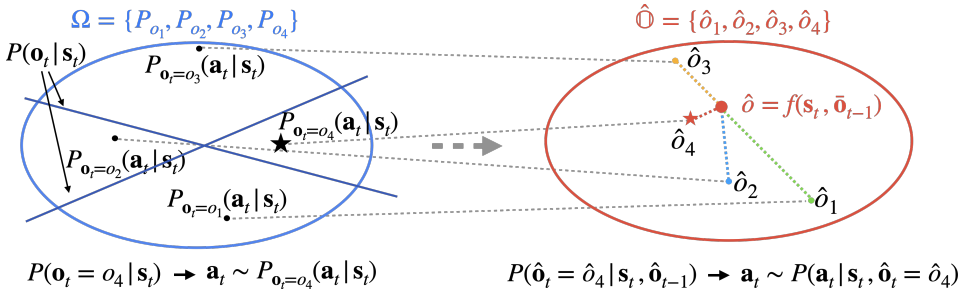


Figure 2: $Option^{\text{Triple}}$ Classification on Statistical Manifold (left) v.s. $Option^{\text{Embed}}$ Clustering on Parametric Space (right). On Statistical Manifold, for M options there are M intra-option policies to learn (for clarity we omit $P_o(\mathbf{b}|\mathbf{s})$ and $\mathbb{I}_o$). Selecting options is analogously learning classification hyperplanes (blue lines). On Parametric Space, for M options there are M embedding centroids to learn yet all embeddings share one action policy (decoder). Selecting options is analogously assigning the closest centroid to the point $[\mathbf{s}_t, \mathbf{o}_{t-1}]$.

On the other hand, as suggested by Eq. (3), temporal abstractions can be achieved more efficiently only by a simple *option policy* $P(\bar{\mathbf{o}}_t|\mathbf{s}_t, \bar{\mathbf{o}}_{t-1})$ as long as there is sufficient context information encoded in $\bar{\mathbf{o}}$. Inspired by word embeddings in Word2Vec [21], we address this issue by representing options as more compact and computationally efficient embeddings (*i.e.*, semantic space of options). Specifically, we define a parameter matrix $\boldsymbol{W}_{\mathbf{o}} = [\hat{\mathbf{o}}_1, ..., \hat{\mathbf{o}}_K] \in \mathbb{R}^{D \times K}$, where D is the dimension of the vector and K is number of options. Each real-valued vector $\hat{\mathbf{o}}$ is referred to as an $Option^{\text{Embed}}$. As in transformer [34], the option embedding is retrieved from the embedding matrix by $\hat{\mathbf{o}} = \boldsymbol{W}_{\mathbf{o}}\bar{\mathbf{o}}$. Under this formulation, the hidden variable $\bar{\mathbf{o}}$ remains the random variable of the *option policy*, while

the option embedding matrix $\boldsymbol{W}_{\mathbf{o}}$ is simply the distribution's parameters:

$$P(\bar{\tau}) = P(\mathbf{s}_0)P(\bar{\mathbf{o}}_0)P(\mathbf{a}_0|\mathbf{s}_0, \bar{\mathbf{o}}_0)\prod_{t=1}^{\infty} P(\mathbf{s}_t|\mathbf{s}_{t-1}, \mathbf{a}_{t-1})P(\mathbf{a}_t|\mathbf{s}_t, \hat{\mathbf{o}}_t)P(\bar{\mathbf{o}}_t|\mathbf{s}_t, \bar{\mathbf{o}}_{t-1}; \boldsymbol{W}_{\mathbf{o}}) \tag{4}$$

where the action policy employs $\boldsymbol{W}_{\mathbf{o}}$ as a realized hidden variable rather than parameters, such that $P(\mathbf{a}_t|\mathbf{s}_t, \hat{\mathbf{o}}_t = \boldsymbol{W}_{\mathbf{o}}\bar{\mathbf{o}}) = P(\mathbf{a}_t|\mathbf{s}_t, \bar{\mathbf{o}}_t, \boldsymbol{W}_{\mathbf{o}})$. Note that rather than a mixture of K distributions defined by Eq. (2), the action policy is now a single decoder which is shared by all option embeddings. All local information of an $Option^{\text{Triple}}$ (where to initiate, what actions to emit and when to terminate) are parameterized as an embedding vector, *i.e.*, $Option^{\text{Embed}}$.

As shown in Figure 2 (right), $Option^{\text{Embed}}$s are actually clustering centroids on a parametric space, which is homeomorphic to the statistical manifold. The *option policy* maps the observed pair $[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]$ to a point (detailed implementation is available in Section 4) on the parametric space. The decision of selecting an option $\hat{\mathbf{o}}_t$ can be made by simply assigning the point to the closest centroid $\hat{\mathbf{o}}^*$, whose distance is estimated efficiently by employing the Attention mechanism [34]. Because a vector is closest to itself, this mechanism has a natural tendency to continue $\hat{\mathbf{o}}_t = \hat{o}_{t-1}$, yet a significantly different state $\mathbf{s}_t$ will pull the point far enough from $\hat{\mathbf{o}}_{t-1}$ and result in another option centroid being assigned. Because the parametric space is now an ambient space of both state space and option space, $Option^{\text{Embed}}$ combines advantages from both temporal abstraction and state abstraction [14].

3.3 Policy Gradient-based Optimization Algorithms

Computing gradients of Eq. (4) directly is intractable because it depends on the state transition probability that is unknown under the model-free RL setting. We address this issue by first following the RL literature [30] and define value functions as expected returns conditioned on the PGM's dynamics. We then propose a novel *Markovian option-value function* to enable the derivation of the Bellman equation. We finally derive a sample efficient policy gradient-based theorems [30], which do not involve the state distribution's gradients, along the expansion of the Bellman equation. Due to space limitations, we provide the details of all proofs in Appendix 3.

Following the RL literature [30], we derive the Bellman equation by recursively expanding value functions. Similar to the value function in Section 2, we define the *option-value function* as the expected return $G_t = \sum_k^N \gamma^k r_{t+k+1}$ conditioned on the current state and option:

$$Q_O[\mathbf{s}_t, \bar{\mathbf{o}}_t] = \mathbb{E}[G_t|\mathbf{s}_t, \bar{\mathbf{o}}_t] \tag{5}$$

We can further expand the *option-value function* as the expectation of an *option-action-value function* $Q_A[\mathbf{s}_t, \bar{\mathbf{o}}_t, \mathbf{a}_t]$ by exploiting conditional independencies encoded in the PGM:

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

In order to establish the connection between the current value and its successors, *i.e.*, deriving the Bellman equation, we need to further expand the $Q_A[\mathbf{s}_t, \bar{\mathbf{o}}_t, \mathbf{a}_t]$ to the next time step:

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

However, as shown in Eq. (7), the last term has an extra dependency on the hidden variable $\bar{\mathbf{o}}_t$. As a result, the conventional value function $V[\mathbf{s}_t]$ does not yield the recursive formulation required for deriving the Bellman equation. This issue has not been identified in related literatures [10, 28, 27, 17, 38] which employ the *option policy* as "one-step option". In order to address this issue, we propose a novel *Markovian option-value function* $\bar{V}[\mathbf{s}_{t+1}, \bar{\mathbf{o}}_t] = \mathbb{E}[G_t|\mathbf{s}_{t+1}, \bar{\mathbf{o}}_t]$ and prove that:

Proposition 3.1. *$\bar{V}[\mathbf{s}_t, \bar{\mathbf{o}}_{t-1}]$ is an unbiased estimation of $V[\mathbf{s}_t]$.*

Proposition 3.2. *The variance of $\bar{V}[\mathbf{s}_t, \bar{\mathbf{o}}_{t-1}]$ is up-bounded by $V[\mathbf{s}_t]$.*

Proofs are available in Appendix 3. The variance-reduction effect is empirically verified in Section 5. We can then employ the *Markovian option-value function* $\bar{V}[\mathbf{s}_t, \bar{\mathbf{o}}_{t-1}]$ to derive the Bellman equation:

$$\bar{V}[\mathbf{s}_{t+1}, \bar{\mathbf{o}}_t] = \mathbb{E}[G_{t+1}|\mathbf{s}_{t+1}, \bar{\mathbf{o}}_t] = \sum_{\bar{\mathbf{o}}_{t+1}} P(\bar{\mathbf{o}}_{t+1}|\mathbf{s}_{t+1}, \bar{\mathbf{o}}_t)Q_O[\mathbf{s}_{t+1}, \bar{\mathbf{o}}_{t+1}]. \tag{8}$$

Expanding Q_O in Eq. (8) through Eq. (5) to Eq. (7) gives the recursive formulation of the Bellman equation. With the Bellman equation in hand, we now are able to derive policy gradient theorems (Note that to keep notations uncluttered, we use $\theta_{\bar{o}}$ to denote *option policy*'s parameters $P(\bar{\mathbf{o}}_t|\mathbf{s}, \bar{\mathbf{o}}_{t-1}; \theta_{\bar{o}})$ and θ_a to denote action policy's parameters $P(\mathbf{a}_t|\mathbf{s}_t, \bar{\mathbf{o}}_t; \theta_a)$):

Theorem 3.3. ***Option Policy Gradient Theorem:*** *Given a stochastic* option policy *differentiable in its parameter vector $\theta_{\bar{o}}$, the gradient of the expected discounted return with respect to $\theta_{\bar{o}}$ is:*

$$\frac{\partial \bar{V}[\mathbf{s}_t, \bar{\mathbf{o}}_{t-1}]}{\partial \theta_{\bar{o}}} = \mathbb{E}[\ \frac{\partial P(\bar{\mathbf{o}}'|\mathbf{s}', \bar{\mathbf{o}})}{\partial \theta_{\bar{o}}} Q_O[\mathbf{s}', \bar{\mathbf{o}}'] \mid \mathbf{s}_t, \bar{\mathbf{o}}_{t-1}], \tag{9}$$

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

Theorem 3.4. ***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:*

$$\frac{\partial Q_O[\mathbf{s}_t, \bar{\mathbf{o}}_t]}{\partial \theta_a} = \mathbb{E}[\ \frac{\partial P(\mathbf{a}|\mathbf{s}, \bar{\mathbf{o}})}{\partial \theta_a} Q_A[\mathbf{s}, \bar{\mathbf{o}}, \mathbf{a}] \mid \mathbf{s}_t, \bar{\mathbf{o}}_t]. \tag{10}$$

Detailed proofs of these two theorems are available in Appendix 4.2. Similar to DAC [38], our gradient theorems enable a two-stage optimization scheme (Appendix 5) for learning options, in which sample efficient MDP-based algorithms (such as PPO [35]) can be employed directly. It is worth to clarify that the derivation of our theorems is significantly different from DAC. In DAC, the random variable $\mathbf{o}$ is part of the augmented space $\mathbb{S}^H$ of the high-level MDP. On the contrary, we employ an HMM-style PGM and $\bar{\mathbf{o}}$ is a hidden variable of the *option policy* and the *action policy*.

4 The Option2Vec Architecture

As illustrated in Section 3.2, our main contributions are representing options as embeddings ($Option^{\text{Embed}}$) and replacing the *call-and-return* mode with the *clustering mechanism*. Inspired by the *Transformer* [34], in this section we implement above mechanisms as Option2Vec, a simple yet effective Multi-Head Attention (MHA) [34] based Encoder-Decoder architecture as shown in Figure 3. Due to space limitations, we explain MHA in Appendix 7. In Option2Vec, we implement the *option policy* $P(\bar{\mathbf{o}}_t|\mathbf{s}_t, \bar{\mathbf{o}}_{t-1}; \boldsymbol{W}_{\mathbf{o}})$ as the encoder and treat the option embedding matrix $\boldsymbol{W}_{\mathbf{o}}$ as encoder's parameters. Specifically, we define the *option encoder* as:

$$\bar{\mathbf{o}}_t \sim Categorical(Clustering(\mathbf{s}_t, \bar{\mathbf{o}}_{t-1}, \boldsymbol{W}_{\mathbf{o}})) \tag{11}$$

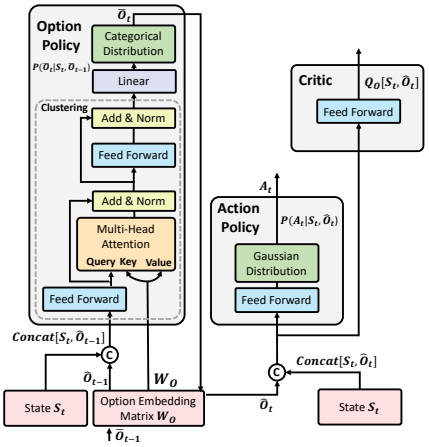


Figure 3: The Option2Vec Architecture.

where $Categorical(\cdot)$ is a K-dimensional categorical distribution, K is the number of options, and distances between the pair $[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]$ (where $\hat{\mathbf{o}}_{t-1} = \boldsymbol{W}_{\mathbf{o}}\bar{\mathbf{o}}_{t-1}$) and all clustering centroids $\hat{\mathbf{o}}$ in embedding matrix $\boldsymbol{W}_{\mathbf{o}} = [\hat{\mathbf{o}}_1, ..., \hat{\mathbf{o}}_K]$ are measured by an efficient MHA-based *clustering module*:

$$Clustering(\mathbf{s}_t, \bar{\mathbf{o}}_{t-1}, \boldsymbol{W}_{\mathbf{o}}) = \text{FFN}(\text{MHA}(\text{Query} = \text{FFN}([\mathbf{s}_t, \hat{\mathbf{o}}_{t-1}]), \text{Key=Value} = \boldsymbol{W}_{\mathbf{o}})) \tag{12}$$

where the concatenated vector $[\mathbf{s}_t, \hat{\mathbf{o}}_{t-1} = \boldsymbol{W}_{\mathbf{o}}\bar{\mathbf{o}}_{t-1}]$ is mapped back to a point in the parametric space by employing a simple Feed-Forward Network. Option2Vec largely improves the option framework's scalability: under the $Option^{\text{Triple}}$ formulation, adding one option means adding two distributions (normally implemented as neural networks); on the contrary, in Option2Vec adding one option is as simple as adding one embedding vector $\hat{\mathbf{o}}$. Because the *clustering module* is MHA-based, the number of parameters for the network stays unchanged with respect to the number of embeddings in $\boldsymbol{W}_{\mathbf{o}}$. With all knowledge of an option (*e.g.*, where to initiate, what actions to emit, and when to terminate) encoded as an embedding vector $Option^{\text{Embed}}$, the *action policy* can be simply implemented as one decoder, which learns to decode $\hat{\mathbf{o}}_t$ and $\mathbf{s}_t$ into primary actions $\mathbf{a}_t$.

$$\mathbf{a}_t \sim Gaussian(\text{FFN}([\mathbf{s}_t, \hat{\mathbf{o}}_t])) \tag{13}$$

which is shared by all $Option^{\text{Embed}}$. This design choice largely improves Option2Vec's sample efficiency and speed of convergence: unlike in $Option^{\text{Triple}}$ that only the activated option ω_o's *intra-option policy* gets updated, the *action policy* learns to decode $Option^{\text{Embed}}$ at every time step.

Because of the *Markovian option-value function* $\bar{V}[\mathbf{s}_{t+1}, \bar{\mathbf{o}}_t]$ is an expectation of the *option-value function* $Q_O[\mathbf{s}_{t+1}, \bar{\mathbf{o}}_{t+1}]$ in Eq. (8), we only need to model only one critic function: $Q_O = \text{FFN}(\mathbf{s}_t, \bar{\mathbf{o}}_t)$, where Q_O is also a decoder of $\mathbf{s}_t$ and $\bar{\mathbf{o}}_t$. We summarize the detailed algorithm in Appendix 5 and upload our code in supplemental materials.

5 Experiments

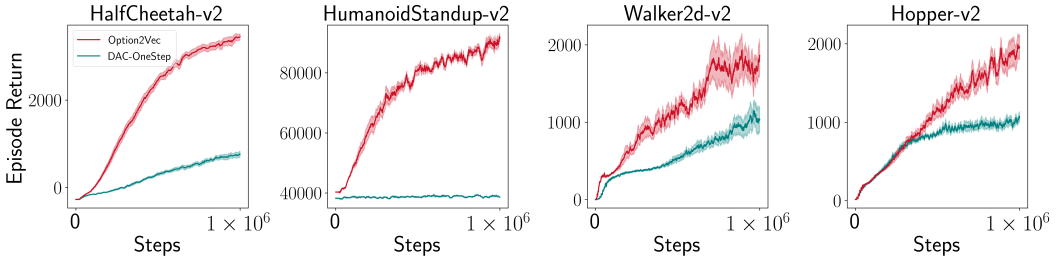
In this section, we design experiments to answer five questions: (Q1) Under the "one-step option" setting (Section 3.1), whether $Option^{\text{Embed}}$ is a more compact and effective representation than $Option^{\text{Triple}}$? (Q2) Beyond "one-step option" setting, whether Option2Vec can achieve better performance than other option variants and non-option baselines? (Q3) Does $Option^{\text{Embed}}$ have a performance boost over other option variants in transfer learning settings? (Q4) Is $Option^{\text{Embed}}$ interpretable? (Q5) Whether Option2Vec can temporally extend options under the *clustering* mechanism?

For baselines, we follow DAC [38]'s open source implementations and compare our algorithm with six baselines, five of which are option variants, *i.e.*, DAC+PPO, AHP+PPO [18], IOPG [29], PPOC [13] and OC [3] The non-option baseline is PPO [26]. All baselines' parameters used by DAC remain unchanged other than the maximum number of training steps: Option2Vec only needs 1 million steps to converge rather than the 2 million used in DAC. For single task learning, experiments are conducted on all OpenAI Gym MuJoCo environments (10 environments) [5]. For transfer learning, we follow DAC and run 6 pairs of transfer learning tasks based on DeepMind Control Suite [32]. Figures are plotted using DAC's original script: curves are averaged over 10 independent runs and smoothed by a sliding window of size 20. Shaded regions indicate standard deviations. 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 6.

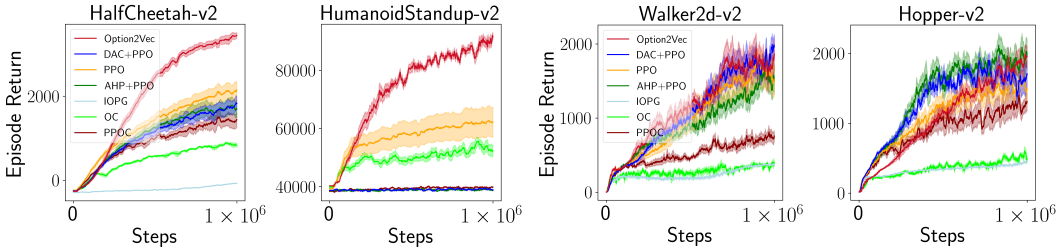
For a fair comparison, we follow DAC and use four options in all implementations. We fix the embedding dimension of $Option^{\text{Embed}}$ to 40 in all environments, *i.e.*, $\boldsymbol{W}_{\mathbf{o}} \in \mathbb{R}^{40\times 4}$. Under this configuration, Option2Vec only use 15.8% parameters in all experiments (single task and transfer learning) compared to the other option variants. Our code is available in supplemental materials.

5.1 Single-Task Learning (Q1 & Q2)

To answer Q1, we need to compare the effectiveness of $Option^{\text{Embed}}$ and $Option^{\text{Triple}}$ under the same "one-step option" setting. As explained in Section 3.1, although DAC is derived under the "one-step option" setting, it still employs *termination function* and *master policy* for better performance. Therefore, we implement a "DAC-OneStep" that only employs the *option policy* (code is available in supplemental materials) and compare its performance with ours in Figure 4a. Experiments show that $Option^{\text{Embed}}$ outperforms $Option^{\text{Triple}}$ in all environments by a large margin. This is because, as explained in Section 3.1, under the "one-step option" setting, the $Option^{\text{Triple}}$'s integer *option index* $\mathbf{o}$ in Eq. (1) contains little context information for the *option policy* to exploit compared to the $Option^{\text{Embed}}$. Moreover, an $Option^{\text{Embed}}$ vector only has 40 parameters to train compared to $Option^{\text{Triple}}$'s 19K parameters (in DAC each *intra-option policy* is a 3-layer FFN). Therefore, $Option^{\text{Embed}}$ is a more compact and effective representation of options compared to $Option^{\text{Triple}}$.



(a) Q1: Option2Vec v.s. DAC-OneStep



(b) Q2: Infinite horizon environments **(c)** Q2: Finite horizon environments

Figure 4: Single-task episodic returns in 4 different environments (*i.e.*, HalfCheetah-v2, HumanoidStandup-v2, Walker2d-v2, and Hopper-v2). Results in all 10 environments are available in Appendix 2.1.

To answer Q2, we compare Option2Vec against five different option variants (*i.e.*, DAC+PPO [38], AHP+PPO [18], IOPG [29], PPOC [13] and OC [3]) and PPO [26]. It is surprising that Option2Vec shows two different performance on infinite (Figure 4b) and and finite (Figure 4c) horizon environments. Previous literatures [13, 29, 8, 38] find that option-based algorithms do not have advantages over hierarchy-free algorithms on single-task environments. Option2Vec is also able to achieve comparable performance with state of the arts (Figure 4c) but with only 15.8% parameters. More importantly, on infinite horizon environments (Figure 4b), Option2Vec's performance significantly outperforms all baselines with respect to episodic return, convergence speed, variance between steps, and variance between 10 runs (Proposition 3.2). Because theoretically infinite and finite horizon environments are identical [30], we do not have a theoretical explanation for this performance difference. In Appendix 1 we conceptually explain that this might because conventional value functions are insufficient to approximate environments in which hidden variables $\mathbf{o}$ only affect rewards but not states. In conclusion, experiment results show that by employing embeddings and the *clustering* mechanism (Section 3.2), Option2Vec is at least as effective as other option variants yet with significantly less computational cost, while has a significant advantage on infinite horizon.

5.2 Transfer Learning (Q3)

We run 6 pairs of transfer learning tasks constructed in DAC based on DeepMind Control Suite [32]. Each pair contains two different tasks. To keep consistent with DAC, we train all models one million steps on the first task and switch to the second (with $Option^{\text{Embed}}$ frozen) to run another one million steps. Results are reported in Figure 5. On the transfer learning (the second) task, Option2Vec's performance ranks the first in 5 out of 6 environments. This shows Option2Vec's advantages in knowledge reuse tasks and its performance is at least comparable with other option variants with significantly less computational cost.

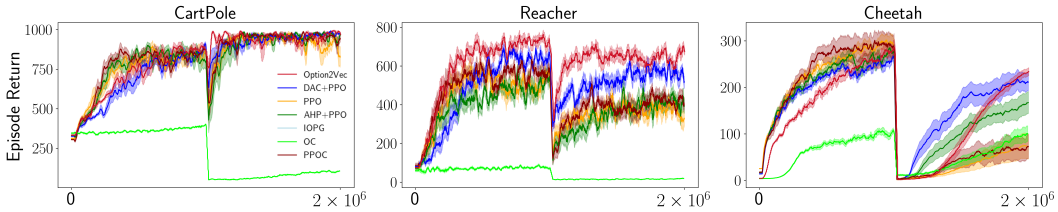


Figure 5: Transfer learning results on three pairs of tasks. All 6 pairs of results are available in Appendix. 2.1

5.3 Interpretation of Option Embeddings (Q4)

Interpretability is a key property (*e.g.*, ensuring safety to human, *etc.*) to apply RL agents in real-world applications. $Option^{\text{Embed}}$ has a nature advantage over $Option^{\text{Triple}}$ on interpretability: as word embeddings [34], $Option^{\text{Embed}}$ learns a semantic space of options with each dimension encodes a particular property and can be interpreted explicitly. As in the capsule network [25], we first reason each embedding dimension's semantic by adding perturbations on to it, and inspecting perturbations' effects on primary actions $\mathbf{a}$ (Figure 6(b)). Once each dimension is understood, embeddings become straight forward to interpret by simply inspecting on which dimensions each embedding $\hat{\mathbf{o}}$ (Figure 6(a)) has significant weights, and interpreting properties of those dimensions (Figure 6(c)). Due to space limitations, more details and GIFs are provided in Appendix 2.3.

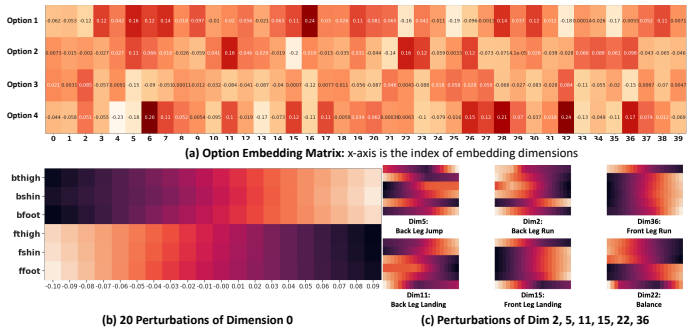


Figure 6: Interpretation of option embeddings

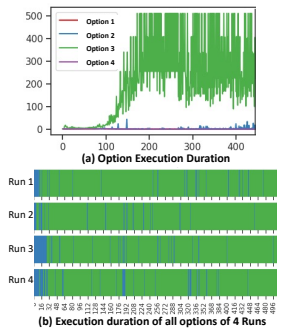


Figure 7: Option Duration Patterns

5.4 Temporal Extension (Q5)

We explained in Section 3.2 that *clustering mechanism* is able to temporally extend an option without the *termination function*. In this subsection we empirically demonstrate this in Figure 7 (more details in Appendix 2.2). At the start of training, all options' durations are short, while Option 3's duration quickly grows. Although this proves that Option2Vec does temporally extend an option, it also shows that Option2 quickly dominates the whole episode. The "dominant skill problem" is not unique to Option2Vec, it is a long identified problem of the option framework [7]. However, as shown in the video[1], Option2Vec actually learns distinguishable options. Option 3 is a running forward skill thus it dominates the whole episode. Option 2 is mainly used to recover from falling down thus its duration decreases with training. In Appendix 1, we argue that the dominant skill problem is actually a problem of learning options at multi-level granularities. Since distances between embeddings are trivial to measure, $Option^{\text{Embed}}$ potentially provides an elegant solution to this problem. We focus this paper on proposing Option2Vec and will further explore this problem in future works.

6 Related Works

To discover options automatically, Sutton et al. [31] 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. AHP [18] is 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. OC [3] proposes a policy gradient based framework for explicitly learning intra-option policies and *termination function*s in an intra-option manner. However, for the master policy's policy gradients learning, OC still remains SMDP-style. DAC [38] reformulated the option framework into two augmented MDPs. Under this formulation, all policies can be modeled explicitly and learned in MDP-style. Policy gradient theorems we derive also follow DAC's efficient two-stage optimization scheme yet is significantly different from DAC as explained in Section 3.3. All above option variants employ computationally expensive $Option^{\text{Triple}}$ and *call-and-return* mode, while Option2Vec employs more simple yet efficient $Option^{\text{Embed}}$ and *clustering mechanism*. We must appreciate that Bacon [2] in his Ph.D. thesis (Chapter 3.5, 3.6) first conceptually discussed the possibility of introducing distributed representations into the option framework. However, to the best of our knowledge, Option2Vec is the first concrete work that enables learning options as distributed representations and deriving learning algorithms.

As for state abstractions, cognitive science experiments [36] conducted with human participants prove that human structure, generalize, and adapt past knowledge to new environments by employing both state abstraction and temporal abstraction. However, existing RL literature [9, 19, 33] only use latent variables to learn state abstractions. Typically, PEARL [24] learns a latent context vector for each task under the meta-reinforcement learning framework to improve the agent's sample and transfer learning efficiency. However, embeddings learned by RL frameworks only encode state abstraction while Option2Vec is the first option variant learns abstractions at both state and temporal dimensions.

7 Conclusions

In this paper, we proposed a compact and effective embedding representation for the option framework, $Option^{\text{Embed}}$. For learning $Option^{\text{Embed}}$, we developed a novel *Markovian Option-Value function* and derived sample efficient policy gradient theorems based on this value function. We implemented the whole mechanism as Option2Vec, a simple yet effective, transformer-like Attention-based Encoder-Decoder architecture. Empirical studies showed that $Option^{\text{Embed}}$ can significantly outperform $Option^{\text{Triple}}$ under the same "One-step option" setting in all environments. We also showed that Option2Vec achieves at least comparable performance to other option variants and non-option baselines on finite and transfer learning environments but with only 15.8% parameters, while has a significant performance boost on infinite environments. Moreover, option embedding has good interpretability, which is a key property for applying RL agents in real-world applications.

It is also worth mentioning that, the $Option^{\text{Embed}}$ establishes a direct connection to causal reinforcement learning [16, 22]. If trained under a model-based setting, $Option^{\text{Embed}}$ is a minimal causal feature set (Theorem 3 [37]) on both temporal and state dimensions. We will investigate the causal properties of Option2Vec in our future works.

[1] https://www.youtube.com/watch?v=F26tcSsIoMA

References

[1] Amari, S.-i. Differential geometrical theory of statistics. *Amari et al. ABNK+87*, pp. 19–94, 1987.

[2] Bacon, P.-L. *Temporal Representation Learning*. PhD thesis, McGill University Libraries, 2018.

[3] Bacon, P.-L., Harb, J., and Precup, D. The option-critic architecture. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[4] Bishop, C. M. *Pattern recognition and machine learning*. springer, 2006.

[5] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

[6] Daniel, C., Van Hoof, H., Peters, J., and Neumann, G. Probabilistic inference for determining options in reinforcement learning. *Machine Learning*, 104(2-3):337–357, 2016.

[7] Haarnoja, T., Hartikainen, K., Abbeel, P., and Levine, S. Latent space policies for hierarchical reinforcement learning. In *International Conference on Machine Learning*, pp. 1851–1860. PMLR, 2018.

[8] Harb, J., Bacon, P.-L., Klissarov, M., and Precup, D. When waiting is not an option: Learning options with a deliberation cost. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[9] Hausman, K., Springenberg, J. T., Wang, Z., Heess, N., and Riedmiller, M. Learning an embedding space for transferable robot skills. In *International Conference on Learning Representations*, 2018.

[10] Henderson, P., Chang, W.-D., Bacon, P.-L., Meger, D., Pineau, J., and Precup, D. Optiongan: Learning joint reward-policy options using generative adversarial inverse reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[11] Jong, N. K., Hester, T., and Stone, P. The utility of temporal abstraction in reinforcement learning. In *AAMAS (1)*, pp. 299–306. Citeseer, 2008.

[12] Khetarpal, K., Klissarov, M., Chevalier-Boisvert, M., Bacon, P.-L., and Precup, D. Options of interest: Temporal abstraction with interest functions. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 4,444–4,451, 2020.

[13] Klissarov, M., Bacon, P.-L., Harb, J., and Precup, D. Learnings options end-to-end for continuous action tasks. *arXiv preprint arXiv:1712.00004*, 2017.

[14] Knoblock, C. A. Learning abstraction hierarchies for problem solving. In *AAAI Conference on Artificial Intelligence*, pp. 923–928, 1990.

[15] Koller, D. and Friedman, N. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[16] Kolobov, A., Weld, D. S., et al. Discovering hidden structure in factored mdps. *Artificial Intelligence*, 189:19–47, 2012.

[17] Lee, S.-H. and Seo, S.-W. Learning compound tasks without task-specific knowledge via imitation and self-supervised learning. In *International Conference on Machine Learning*, pp. 5,747–5,756. PMLR, 2020.

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

[19] Li, Y., Song, J., and Ermon, S. Infogail: Interpretable imitation learning from visual demonstrations. In *Advances in Neural Information Processing Systems*, pp. 3,812–3,822, 2017.

[20] Mankowitz, D. J., Mann, T. A., and Mannor, S. Adaptive skills, adaptive partitions (asap). *arXiv preprint arXiv:1602.03351*, 2016.

[21] Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pp. 3,111–3,119, 2013.

[22] Perez, C. F., Such, F. P., and Karaletsos, T. Generalized hidden parameter mdps transferable model-based rl in a handful of trials. *arXiv preprint arXiv:2002.03072*, 2020.

[23] Puterman, M. L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

[24] Rakelly, K., Zhou, A., Finn, C., Levine, S., and Quillen, D. Efficient off-policy meta-reinforcement learning via probabilistic context variables. In *International Conference on Machine Learning*, pp. 5,331–5,340, 2019.

[25] Sabour, S., Frosst, N., and Hinton, G. E. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pp. 3,856–3,866, 2017.

[26] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[27] Shankar, T. and Gupta, A. Learning robot skills with temporal variational inference. In *International Conference on Machine Learning*, pp. 8624–8633. PMLR, 2020.

[28] Sharma, A., Sharma, M., Rhinehart, N., and Kitani, K. M. Directed-info gail: Learning hierarchical policies from unsegmented demonstrations using directed information. *arXiv preprint arXiv:1810.01266*, 2018.

[29] Smith, M., Hoof, H., and Pineau, J. An inference-based policy gradient method for learning options. In *International Conference on Machine Learning*, pp. 4,703–4,712, 2018.

[30] Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

[31] Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[32] Tassa, Y., Tunyasuvunakool, S., Muldal, A., Doron, Y., Liu, S., Bohez, S., Merel, J., Erez, T., Lillicrap, T., and Heess, N. dmcontrol: Software and tasks for continuous control, 2020.

[33] Tirumala, D., Noh, H., Galashov, A., Hasenclever, L., Ahuja, A., Wayne, G., Pascanu, R., Teh, Y. W., and Heess, N. Exploiting hierarchy for learning and transfer in kl-regularized rl. *arXiv preprint arXiv:1903.07438*, 2019.

[34] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5,998–6,008, 2017.

[35] Witoonchart, P. and Chongstitvatana, P. Application of structured support vector machine backpropagation to a convolutional neural network for human pose estimation. *Neural Networks*, 92:39–46, 2017.

[36] Xia, L. and Collins, A. G. E. Temporal and state abstractions for efficient learning, transfer and composition in humans. *bioRxiv*, 2020.

[37] Zhang, A., McAllister, R., Calandra, R., Gal, Y., and Levine, S. Learning invariant representations for reinforcement learning without reconstruction. *arXiv preprint arXiv:2006.10742*, 2020.

[38] Zhang, S. and Whiteson, S. DAC: The double actor-critic architecture for learning options. In *Advances in Neural Information Processing Systems*, pp. 2,012–2,022, 2019.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No] Our contribution is about making the option framework more efficient (environmental friendly) and interpretable (ensuring safety in real-world applications). We did think about potential negative impacts but really hard to come up with some.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] Due to page limitations, we defer full set of assumptions to Appendix (supplemental materials).
   (b) Did you include complete proofs of all theoretical results? [Yes] Due to page limitations, we defer complete proofs to Appendix (supplemental materials).
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] In supplemental material
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] In supplemental material
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes] In supplemental material
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] Section 5
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [No] We use standard RL environments and MIT licensed DAC's codebase.
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [No] We use standard RL environments and MIT licensed DAC's codebase.
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] We use standard RL environments and MIT licensed DAC's codebase.
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [No]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [No]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [No]