2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 23-27, 2022, Kyoto, Japan

Learning Skills to Navigate without a Master: A Sequential Multi-Policy Reinforcement Learning Algorithm

Ambedkar Dukkipati and Rajarshi Banerjee and
Ranga Shaarad Ayyagari and Dhaval Parmar Udaybhai

Abstract— **Solving complex problems using reinforcement learning necessitates breaking down the problem into manageable tasks, and learning policies to solve these tasks. These policies, in turn, have to be controlled by a master policy that takes high-level decisions. Hence learning policies involves hierarchical decision structures. However, training such methods in practice may lead to poor generalization, with either sub-policies executing actions for too few time steps or devolving into a single policy altogether. In our work, we introduce an alternative approach to learn such skills *sequentially* without using an overarching hierarchical policy. We propose this method in the context of environments where a major component of the objective of a learning agent is to prolong the episode for as long as possible. We refer to our proposed method as *Sequential Soft Option Critic*. We demonstrate the utility of our approach on navigation and goal-based tasks in a flexible simulated 3D navigation environment that we have developed. We also show that our method outperforms prior methods such as Soft Actor-Critic and Soft Option Critic on various environments, including the Atari River Raid environment and the Gym-Duckietown self-driving car simulator.**

I. INTRODUCTION

Reinforcement Learning (RL) in the past decade achieved unprecedented success in multiple domains ranging from playing simple Atari games [1] to learning complex strategies and defeating pro players in Starcraft [2] and Go [3]. However, the problem of learning meaningful skills in reinforcement learning remains an open question. The options framework [4] provides a method to automatically extract temporally extended skills for a long horizon task with the use of options, which are sub-policies that can be leveraged by some other policy in a hierarchical manner. The process of learning such temporal abstractions has been widely studied in the broad domain of hierarchical reinforcement learning [5]. In this paper, we provide an alternate approach for learning options sequentially without a higher-level policy and show a better performance on navigation tasks.

In the options framework, each option is defined by a tuple of policies, initiation states, and termination states. The set of termination states of an option is determined by a termination function that maps the state space to its class membership probability of the termination state set. Various advancements in the options framework, like the option critic architecture [6], have significantly improved the convergence of the overall algorithm, but most recent works focus on a fixed set of options that are hard to scale in practical scenarios.

Humans excel at learning skills because while performing tasks, they can apply an inordinate amount of prior information [7]. For an RL agent to learn efficiently in complex environments, it also needs to rely on its previously learned knowledge to continually improve its overall policy. In this spirit, we propose an algorithm to learn policies sequentially so that during the training of any policy, the knowledge obtained till then, in the form of the already trained previous policies, can be leveraged to inform the learning of the current policy. We refer to our method as Sequential Soft Option Critic (SSOC) that is designed to operate in a framework wherein a major component of the goal of the RL agent is to learn diverse skills so as to prolong the episode and survive, as shown in Fig. 1(a). This behavior is incentivized by emitting a reward signal of -1 when the episode ends and a reward signal of 0 for every other time step.

Conceptually one may draw parallels between our approach and curriculum learning [8]. The idea behind curriculum learning is to train a model with a curriculum consisting of a sequence of tasks of increasing complexity rather than simply allowing the model to learn the original task from scratch. In practice, it has been demonstrated that this approach significantly outperforms traditional learning methods [9]. However, the major disadvantage of curriculum-based RL is that it is generally expensive to create a comprehensive curriculum, if not outright impractical. In our approach, when a new option is added, its policy is trained only in states in which the previously trained options are expected to perform poorly. This is done by using the options' termination functions to effectively partition the state space so that each option can learn an optimal policy for some subset of the state space, which is an easier task to accomplish. The options are then chained together with the termination state of the previous option serving as the initial state of the next.

We evaluate our proposed method on three environments: (i) a flexible 3D navigation environment developed by us, (ii) the Duckietown self-driving car simulator [10], and (iii) the Atari River Raid environment.

By conducting extensive experiments, we establish that our method outperforms the Soft Actor-Critic algorithm [11]

This work was supported by the IMPRINT under Grant IMP/2019/000383.

Ambedkar Dukkipati and Ranga Shaarad Ayyagari are with the Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560012, India. (e-mail: {ambedkar,rangaa}@iisc.ac.in). Rajarshi Banerjee and Dhaval Parmar Udaybhai have done this work while at the Department of Computer Science and Automation, Indian Institute of Science, Bangalore 560012, India. (e-mail: {rajarshib,dhavalparmar}@iisc.ac.in).

2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 978-1-6654-7927-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/IROS47612.2022.9981607

and its options counterpart, the Soft Option Critic [12]. Our main contributions are as follows.

(1) We propose a new approach called ‘Sequential Soft Option Critic’ for training options in environments where a primary objective of the agent is to prolong the length of the episode.

(2) We demonstrate the utility of sequentially adding new skills without a policy over options, with experimental results that outperform prior methods in the task of navigation in our simulated 3D environment and the Ducktietown and Atari River Raid environments.

(3) We show that our approach can learn skills to solve complex tasks involving high-level goals in the navigation environment outperforming prior methods.

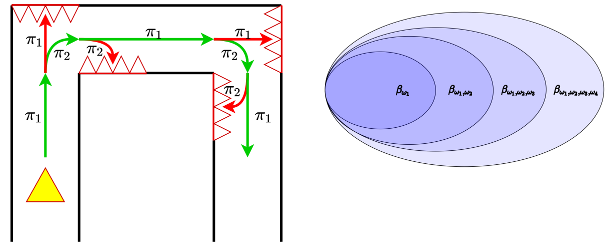
II. RELATED WORK

The concept of temporal abstraction in reinforcement learning has been extensively explored in various works, from humble beginnings with options framework [4], feudal learning [13], hierarchical abstract machines [14], and the MAXQ hierarchical learning algorithm [15], to recent endeavors in imagination augmented agent learning with variational temporal abstraction [16]. Approaches like feudal networks [17] based on feudal learning fused a manager network to choose the direction of navigation in the latent space when learning workers (sub-policies).

The option-critic architecture [6] builds on top of the options framework and makes use of the policy over options to learn its corresponding Q function. This acts as a critic and is used to update the termination functions of the options. Recently Soft Option Actor-Critic Architecture (SOAC) extended this approach by appending intrinsic rewards into the framework [18]. Unlike ,[6] which uses option critic to compute gradients for each sub-policy, in this paper, we study learning sub-policies one at a time.

Hierarchical reinforcement learning with off-policy [19] provided a data-efficient method of training hierarchical policies. Hindsight experience replay [20] has widely been adopted for training policies in sparse reward environments and has also been recently used in multi-level hierarchical reinforcement learning algorithms [21]. Hierarchical reinforcement learning has also shown remarkable success in very complex domains like playing the game of Starcraft [2], although the sub-policies were trained separately and combined together by a master policy, the agent learned to play the game like a pro player [22]. Meta-learning approaches [23] focused on training a meta controller, which would be frequently re-initialized such that it can learn to control the trained sub-policies.

Our approach amounts to partitioning the state space based upon how well a policy performs in it, much like previous iterative approaches [24]. Our method also shares some commonalities with the deep skill chaining algorithm [25] on how new options are added to the existing set of options. Deep skill chaining sequentially learns local skills by chaining them backward from a goal state. However, in our approach, skills are learned to complement the previously acquired ones



(a) (b)

Fig. 1. (a) An illustration of policies learned in our approach. Different policies learn different skills out of necessity to traverse the environment and not terminate the episode. The trajectory spawned by the sequence of policies that correctly learns to avoid terminating the episode is shown in green. The policy π_1 is used when the agent is in the middle of the corridor. When the end of the corridor is reached, π_2 is selected to take a turn and avoid a collision. (b) A representation of the state space partitioned by the termination functions of the options. Each oval corresponds to the set of states classified as non-termination states by the corresponding nested termination function.

such that the agent can traverse to various unseen states. This method shares a striking resemblance with curriculum-based learning approaches [8], but here the curriculum naturally arises from the necessity of traversing the environment. The use of nested termination functions in our framework is inspired by the continual learning architecture in progressive neural networks [26]. We make use of the Soft Actor-Critic algorithm (SAC) [11] that is based on a maximum entropy reinforcement learning framework [27] for training all our policy networks.

III. BACKGROUND

A Markov decision process (MDP) is defined by the tuple $(\mathcal{S}, \mathcal{A}, p, r, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, p is the state transition probability $p(s_{t+1}|s_t, a_t)$ of going to the next state $s_{t+1} \in \mathcal{S}$ from state $s_t \in \mathcal{S}$ given an action $a_t \in \mathcal{A}$, r is the reward function $r(s_t, a_t) \in \mathbb{R}$ that provides a reward signal as the agent traverses the environment, and $\gamma \in [0, 1)$ is the discount factor. The aim is to find an optimal policy π^* such that

$$\pi^* = \arg\max_{\pi} \sum_{t=0}^{T} \mathsf{E}_{(s_t, a_t)\sim\rho_\pi}[\gamma^t r(s_t, a_t)] \,, \qquad (1)$$

where ρ_π is the state-action distribution induced by a policy π. For every policy π, one can define its corresponding Q value function:

$$Q_\pi(s_t, a_t) = r(s_t, a_t) + \gamma \mathsf{E}_{s_{t+1}\sim p(\cdot|s_t, a_t)} V_\pi(s_{t+1}) \,, \qquad (2)$$

where V_π is the value function defined by

$$V_\pi(s_t) = \mathsf{E}_{a_t\sim\pi}[Q_\pi(s_t, a_t)] \,. \qquad (3)$$

A. Soft Actor-Critic

The soft actor-critic algorithm [11] is an off-policy entropy-based reinforcement learning algorithm. The main idea of the entropy-based learning approach is to maximize the entropy of the policy along with the reward. A straightforward way of doing this is by making the reward function depend on the current policy's entropy. Making the reward proportional to the entropy incentivizes greater exploration of the environment, ensuring the policy is far less likely to get stuck in a local optimum. Thus, the optimal policy is defined as

$$\pi^* = \arg\max_{\pi} \sum_{t=0}^{T} \mathsf{E}_{(s_t,a_t)\sim\rho_\pi} \gamma^t [r(s_t,a_t) + \alpha\mathcal{H}(\cdot|s_t)] \, , \quad (4)$$

where α is the temperature variable that accounts for the importance of the entropy and $\mathcal{H}(\cdot|s_t)$ is the entropy of the policy. The above formulation reduces to the standard reinforcement learning objective as $\alpha \to 0$. It is shown that by iteratively using the soft policy evaluation and soft policy improvement, the policy convergences to the optimal policy π^* [11].

The temperature variable α can be treated as a trainable parameter for better performance of the algorithm [28]. This gives the algorithm the flexibility to dictate the relative importance of the policy's entropy. As α decreases, the policy becomes more deterministic in nature. We make use of this property of the soft actor-critic to decide when new policies should be added.

B. The Options Framework

The idea of temporally extended actions has been introduced by [4]. An option $\omega \in \Omega$ is defined by the tuple $(\mathcal{I}_\omega, \pi_\omega, \beta_\omega)$, where Ω is the set of options, π_ω is the policy corresponding to the option, $\mathcal{I}_\omega(\subseteq \mathcal{S})$ is the set of states where the option can be initialized, and $\beta_\omega : S \to [0,1]$ is the termination function of the option. In a typical options framework, k sub-policies, $\pi_{\omega_1}, \pi_{\omega_2}, ...\pi_{\omega_k}$ are initialized, each with its corresponding $\mathcal{I}_\omega$ and β_ω, along with a policy over options π_Ω. In the *call-and-return* approach π_Ω chooses an option ω and π_ω executes actions till it terminates with probability $\beta_\omega(s_t)$ for a given state s_t and the control then returns back to π_Ω. The state transition dynamics are given by

$$\begin{aligned} &P(s_{t+1}, \omega_{t+1}|s_t, \omega_t) \\ &= \sum_a \pi_{\omega_t}(a|s_t) P(s_{t+1}|s_t, a) \big[(1 - \beta_{\omega_t}(s_{t+1})) 1_{\omega_t = \omega_{t+1}} \\ &\qquad\qquad + \beta_{\omega_t}(s_{t+1}) \pi_\Omega(\omega_{t+1}|s_{t+1})\big] \, . \quad (5) \end{aligned}$$

Unlike other approaches where the next option is chosen by a master policy π_Ω, in our proposed approach, the next option to be executed depends on the termination state space of the previous options.

IV. THE PROPOSED ALGORITHM

A. Class of environments

In this paper, we consider environments in which a major component of the task of the agent is to prolong the duration of the episode, which is incentivized by using a simple reward function $r(s_t, a_t, s_{t+1})$, whose value is -1 if s_{t+1} is the final state of the episode, and 0 otherwise. Many real-life problems can be effectively cast as reinforcement learning problems with such a reward function and minimal information from the environment. Examples of such applications include autonomous vehicle navigation while avoiding a collision [29], [30] and drone navigation [31]. Tasks that require the agent to maintain an equilibrium in an ever-changing environment may be cast into our framework using this simple reward function. Practical examples may involve tasks like assembly line automation with increasing levels of complexity.

A consequence of such a reward signal is that as the agent fails less frequently in the process of learning better policies, it becomes harder to train it owing to the increasing sparsity of the failure states. Our approach is designed to overcome this by learning new policies near states where failure occurs without disturbing the already learned policies.

In our proposed approach, the policies learned need only be locally optimal. The main challenge of the learning algorithm then becomes determining how likely it is for the current policy to fail so as to switch to another policy. Unlike other option learning algorithms, the proposed strategy is to sequentially train one option at a time. To learn from a minimal amount of data while training, we do not train a (master) policy over the options but rely on the termination functions of individual options to determine which option should be chosen for execution. The states in which a new policy is trained are determined by the termination functions, which are treated as binary classifiers. New policies are learned only in states that are classified as termination states for all the previous options, and this is achieved by nesting the termination functions of the options. We explain this procedure in detail in the later subsections.

Ideally, one can continue to add more options into the set of all options until $V_\Omega(s_t) \approx 0$, where state s_t belongs to the marginal state distribution induced by the set of options Ω learned by the algorithm. However, from a pragmatic perspective, we threshold the maximum number of options to be learned. We now describe in detail how each component of the framework is adopted in our approach.

B. Training Policies

In the proposed approach, the policies are trained sequentially, i.e., firstly, a single policy and its termination function are trained until semi-optimality, only then another policy is added along with its termination function and trained until semi-optimality, and so on. Once a policy is trained, it is not modified again. Later in this section, we go into detail about how each policy is trained and what we consider to be the semi-optimality of a policy.

Corresponding to any given policy, depending on its performance, the state space is partitioned into termination and non-termination states for that policy. Suppose we have a trained option ω_1. The next option ω_2 need only be trained in those states that are classified as termination states by β_{ω_1}, i.e., the new policy is only focused on learning to operate in states where the previous policy failed. In order to make use of the previously learned policy, we also incentivize the new policy to traverse towards the states that are deemed non-termination states by $\beta_{\omega_1}(s_t)$. For this, in general, given trained options $\omega_1, \ldots, \omega_{i-1}$, we train a new policy π_{ω_i} with the following reward function

$$r_{\pi_{\omega_i}}(s_t, a_t, s_{t+1}) = \begin{cases} 1 & \text{if } \widetilde{\beta}_{\dots\omega_{i-1}}(s_{t+1}, .) = 0, \\ r(s_t, a_t, s_{t+1}) & \text{otherwise,} \end{cases} \tag{6}$$

where $\widetilde{\beta}_{\dots\omega_{i-1}}$ is a 'nested termination function', which acts as a termination function corresponding to all the previously trained options $\omega_1, \dots, \omega_{i-1}$ together. The learning of this nested termination function is described in detail in the next subsection. The reason for giving a $+1$ 'inter-option' reward whenever the new policy enters states with $\widetilde{\beta}_{\dots\omega_{i-1}}(s_{t+1}, .) = 0$ is because the desired objective is to incentivize the new policy to enter states where it can easily switch to some other policy that has already been fully trained and hence, is presumably more capable of traversing those states. This encourages the agent to leverage previously gained knowledge instead of trying to relearn it. The training of the new policy is limited to those state-action pairs that are vital for prolonging the episode and cannot be delegated to previous policies. It is to be noted that if the policy remains in states for which $\widetilde{\beta}_{\dots\omega_{i-1}}(s_{t+1}, .) = 0$, i.e, termination states of previous policies, then the reward given to the policy is the unchanged $\{0, -1\}$ sparse reward of the environment.

This modification of the reward function described by Equation 6 is applied for training every policy other than the first one. We included ablation studies in our experiments to show that this inter-option reward is an important reason behind our approach outperforming the Soft Actor-Critic algorithm in the navigation environment.

Each policy is updated as in soft-actor critic, by using the information projection on the exponential of the soft Q-value

$$\pi_\omega^{new} = \arg\min_{\pi' \in \prod} D_{KL}\left(\pi'(.|s_t) \middle\| \frac{\exp(\frac{1}{\alpha} Q^{\pi_\omega}(s_t, .))}{Z^{\pi_\omega}(s_t)}\right), \tag{7}$$

where $Z^{\pi_\omega}(s_t)$ normalizes the distribution.

In this work, we make use of the trainable α of the soft actor-critic algorithm as a measure to decide whether to stop training a policy and add a new option. α is updated so as to minimize the following cost function [28]

$$J(\alpha) = \mathsf{E}_{a_t \sim \pi_t}[-\alpha \log \pi_t(a_t|s_t) - \alpha\bar{\mathcal{H}}], \tag{8}$$

where $\bar{\mathcal{H}}$ is a hyperparameter set to $-\text{dim}(\mathcal{A})$, where $\mathcal{A}$ is the action space. If α is constrained to satisfy $\alpha \geq 0$, the optimal value of α that satisfies the above objective is 0, since the entropy $\mathsf{E}_{a_t \sim \pi_t}[-\log \pi_t(a_t|s_t)] \geq 0 > \bar{\mathcal{H}}$. So the value of α monotonically decreases as the training progresses.

As the value of α decreases, there is a smaller incentive for the agent to maximize the entropy. This results in the learned policy becoming less exploratory and more deterministic. We determine a policy to be sufficiently trained for it to function as a semi-optimal policy when $\alpha < \alpha_{min}$, where $\alpha_{min} \in [0, 1]$ is a threshold. All new sub-policies are initialized with $\alpha = 1$ to ensure maximum exploration near states where policies are initialized. Essentially we augment a new policy when the temperature variable α of the currently trained policy is low enough to warrant it to be considered an optimal policy near its initialization states.

Once a policy π_ω is trained enough to be deemed semi-optimal, we fix that policy and no longer train it. The rationale for this is that it is much easier to train new options in the context of fixed already trained options rather than trying to learn new options while simultaneously updating old options.

C. Learning Termination Functions

For each option ω, $\widetilde{\beta}_\omega(s_t)$ is either 1 or 0, depending on whether option ω should terminate at state s_t or not. This can be implemented by learning a real-valued function $\beta_\omega(s_t)$ and using it as a binary classifier. Suppose the learning algorithm has already trained $i-1$ options, and the aim is to learn a new option ω_i. We train its new policy as if a single policy is being trained in all states s_t satisfying $\widetilde{\beta}_\omega(s_t) = 1$ for all $\omega \in \Omega_{old} = \{\omega_1, \omega_2, \dots\omega_{i-1}\}$, i.e., in states that are classified as termination states by all the previous options. For this, rather than training β_{ω_i} for the i^{th} option, we train a nested termination function $\beta_{\omega_1,\omega_2,\dots,\omega_i}$, which is the termination classifier for the set of options $\{\omega_1, \omega_2, \dots, \omega_i\}$. As the new policies are trained in the termination states of the previous set of options, the set of non-termination states for the set of options keeps expanding, as shown in Fig. 1(b). Without nested functions, as the number of options increases, it becomes more difficult to accurately update the individual termination functions of each policy, which in turn makes it difficult to incorporate new skills. In the proposed approach, as long as the last nested termination function is correctly updated, new policies can always be learned in the termination states of that function. Using $\widetilde{\beta}_{\omega_1\dots,\omega_i}(s_t)$ and $\widetilde{\beta}_{\omega_1\dots,\omega_{i-1}}(s_t)$, we can obtain $\widetilde{\beta}_{\omega_i}(s_t)$ as

$$\widetilde{\beta}_{\omega_i}(s_t) = (1 - \widetilde{\beta}_{\omega_1\dots,\omega_{i-1}}(s_t)) \vee \widetilde{\beta}_{\omega_1\dots,\omega_i}(s_t) \ . \tag{9}$$

The above equation imposes a simple constraint on choosing options for execution by not allowing a new policy to execute actions in states which are classified as a non-terminating state by the previous nested termination function since that means there is already some trained option more suited to execute in that state.

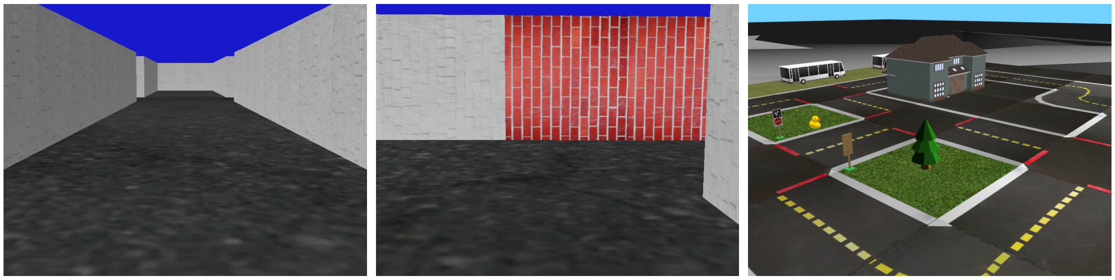
Since $\widetilde{\beta}_{\omega_1,\dots,\omega_i}$ has to take values in 0 or 1 (non-termination and termination respectively), we learn corresponding continuous functions $\beta_{\omega_1,\dots,\omega_i}$ with range $[0, 1]$ and use a threshold to assign $\widetilde{\beta}_{\omega_1,\dots,\omega_i}$ the value 1 if $\beta_{\omega_1,\dots,\omega_i}$ exceeds the threshold, 0 otherwise.

$\beta_{\omega_1\dots\omega_i}$ is trained like a standard Q-value function using the negative of the reward signal emitted by the environment,

$$r_\beta(s_t, a_t, s_{t+1}) = \begin{cases} 1 & \text{if } s_{t+1} \text{ is the last state in the episode, and} \\ 0 & \text{otherwise.} \end{cases} \tag{10}$$

We overload the notation of $\beta_{\omega_1\dots\omega_i}$ by using $\beta_{\omega_1\dots\omega_i}(s_t, a_t)$ as the termination function rather than $\beta_{\omega_1\dots\omega_i}(s_t)$, since we train it like a Q-value function and because it is a better estimator. This function is learned using the update rule:

$$\beta_{\omega_1\dots}(s_t, a_t) \leftarrow r_\beta(s_t, a_t, s_{t+1}) + \gamma_\beta \beta_{\omega_1\dots}(s_{t+1}, a_{t+1}), \tag{11}$$



(a) (b) (c)

Fig. 2. (a) & (b) The input to the neural network in our 3D navigation environment. These images obtained from the simulation are scaled (and also transformed to grayscale in the case of (a)) and appended with K previous images, and sent to the agent as input. (c) A view of the Duckietown environment.

where $\gamma_\beta \in [0,1]$ is the termination discount factor. It largely depends on the problem domain and directly impacts the newer policies that are trained by influencing the partition of the termination and non-termination states.

As new policies are incorporated, $r_\beta(s_t, a_t)$ becomes an increasingly sparse reward since the agent learns strategies to avoid failure, and episodes become relatively longer. In such instances, simply training $\beta_{\omega_1 \ldots \omega_i}(s_t)$ using the above update can bias it to predict all states as non-termination states and thus effectively prevent new policies from learning. To avoid this, after the new policy has been semi-optimally trained, the termination function is trained using the binary cross-entropy loss as

$$\beta_{\omega_{1\ldots}} \leftarrow \underset{\beta_{\omega_1 \ldots \omega_i}}{\arg\min} - \mathsf{E}_{(s_t, a_t)} \left[y_t \log \beta_{\omega_1 \ldots \omega_i}(s_t, a_t) \right. \\ \left. + \; (1 - y_t) \log(1 - \beta_{\omega_1 \ldots \omega_i}(s_t, a_t)) \right]. \tag{12}$$

Here the labels y_t are the solution to (11) if the policy that generates the actions is fixed. They are obtained by unrolling a trajectory $\{s_0, ..., s_T\}$ and labeling each state s_t with $y_t = \gamma_\beta^{T-t}$, where γ_β is the discount factor.

As we fix the policies after training them, the cross-entropy update gives us a better estimation as compared to the previous temporal difference updates. To make $\beta_{\omega_1 \ldots \omega_i}(s_t, a_t)$ unbiased, an equal number of probable termination and non-termination states are sampled for training. The major advantage of adopting nested termination functions is that we only need to modify the latest termination function to correctly reflect if a state is a termination state or not. That will then be used to determine the states in which the next new policy will be trained. Given a training set Ω of options, an option is chosen during execution according to the constraint given in (9).

V. EXPERIMENTAL RESULTS

We compare our approach on several environments against the Soft Actor critic (SAC) [11] and the Soft Option-Critic (SOC) [12] algorithms. Each experimental plot has been plotted by taking the mean of 5 different experiments, and the corresponding bounds are given $\pm$std$/2$.

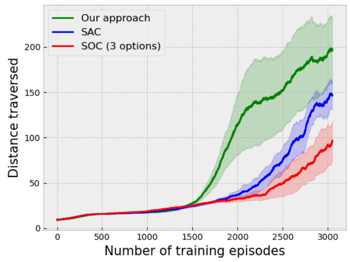
A. The 3D navigation environment

We have created a 3D simulated environment using the Panda3D game engine [32]. This collision avoidance environment consists of long corridors twisting and turning as the agent navigates inside as a vehicle that moves with a constant velocity. The primary challenge is that agent has to understand whether the left or right turn is coming up as taking a wrong turn will inevitably result in a collision and end the episode. The results on this environment are given in Fig. 3(a). In Fig. 4 we show a snapshot of the learned policies navigating in the environment. The first option simply learns to move straight in the environment, while the second and third policies learn to take the correct turns.

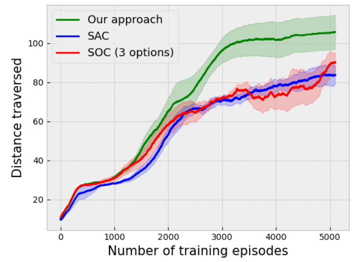
We have also validated our approach on a goal-based version of this environment, in which the agent is also given as input a number representing a destination and is guided towards it by arrows placed in the environment. The results for this setting are given in Fig. 3(g). We have not used an inter-option reward for this setting. We have also validated our method on an instruction-based task in which the agent is given a one-hot vector denoting the direction to take at an intersection, with the results given in Fig. 3(i) compared against Soft Actor-Critic with Hindsight Experience Replay. For this setting, we have also used a $+1$ reward for going in the correct direction and $-\frac{1}{2}$ for going in the wrong direction. Additionally, Fig. 3(b) shows the performance of our approach on a colored version of the environment in which the agent has to take the correct turn depending on the color of the walls in front of it. It should understand the concept of taking a turn in the direction of the colored wall if the color is green and in the opposite direction if the color is red.

B. Other environments

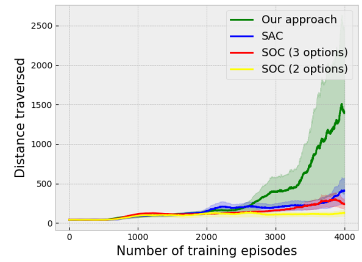
We have tested our algorithm on two more environments: (i) Gym-Duckietown [10], a self-driving car simulator that is a complex environment consisting of multiple immediate turns and various objects like houses, trees, etc., resulting in a large variance in the observations from the environment, and (ii) the Atari River Raid environment, a top-down shooting game in which the goal is to maneuver a plane to destroy or avoid obstacles. The results of the experiments on these environments are given in Fig. 3(c) and 3(e) respectively.



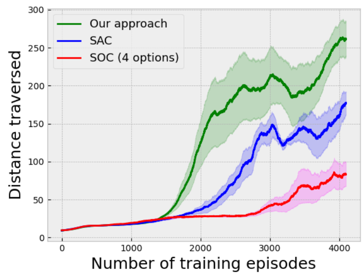
(a) 3D environment with 3 options



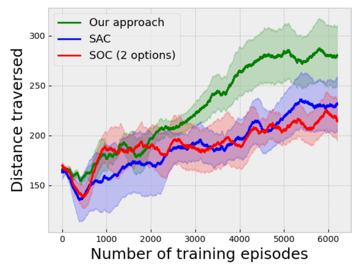
(b) Color 3D environment with 3 options



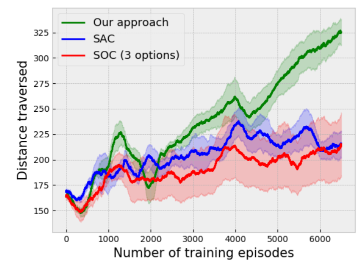
(c) Duckietown environment with 2 options



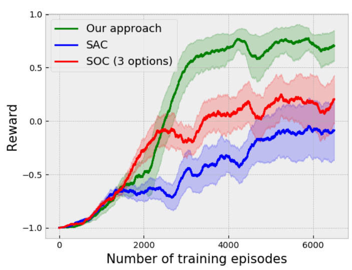
(d) 3D navigation environment with 4 options



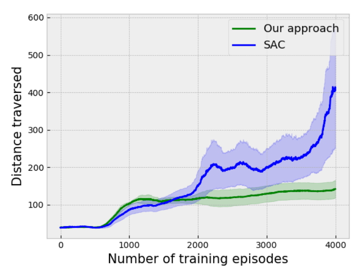
(e) Atari environment with 2 options



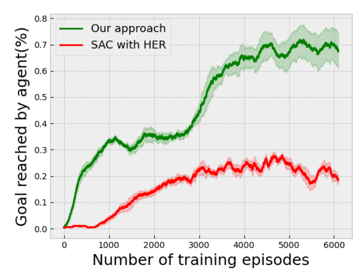
(f) Atari River Raid environment with 3 options



(g) Goal based navigation in the 3D environment



(h) Duckie Town environment without inter-option rewards



(i) Instruction based navigation in the 3D environment

Fig. 3. The results obtained on various environments

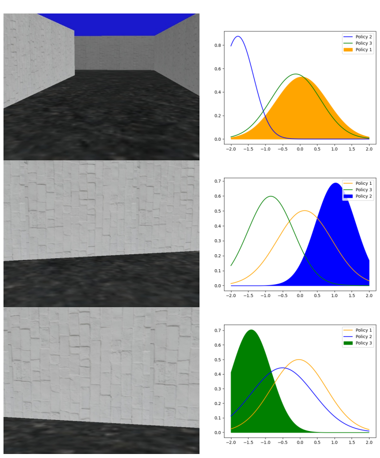


Fig. 4. Outputs of the three learned policies when fed the corresponding input shown on the left. π_{ω_1}: orange, π_{ω_2}: blue, π_{ω_3}: green. Each policy outputs a Gaussian distribution, the active policy is the one filled with color. A positive value for the output action corresponds to turning right, and a negative value indicates a left turn.

Fig. 3(h) show the results of our algorithm when the inter-option reward is removed, showing that it is an essential component of our algorithm in a navigation environment.

C. Implementation

The policy and Q functions are implemented as neural networks that take, at each time step, scaled images seen by the agent from the past K time steps, where $K = 10$ for the navigation environment and $K = 4$ for the Duckietown and Riverraid environments. In environments where the color of the image is not explicitly required as part of the goal specification, the input images are transformed to grayscale before being passed to the networks. The outputs of the policy networks are the parameters of a Gaussian distribution for the case of continuous actions in the navigation and Duckietown environments, and parameters of a categorical distribution for the case of discrete actions in the Riverraid environment.

TABLE I

HYPERPARAMETERS

Parameter	Value
learning rate	3×10^{-4}
discount factor(γ)	0.99
replay buffer size	10^4
target smoothing coefficient(τ)	0.005
number of frames in input state (K)	10
batch size	16
alpha threshold (α_{min})	0.1
termination discount factor (γ_β)	0.95

Two critic networks are used, and the smaller of their predicted values is taken as the Q value to tackle overestimation. Additionally, two additional networks are used as target (soft) Q networks, whose parameters follow those of the critics as exponential moving averages with smoothing coefficient τ.

VI. CONCLUSIONS

In this paper, we proposed an algorithm called 'Sequential Soft Option Critic' that allows adding new skills dynamically without the need for a higher-level master policy. This can be applicable to environments where a primary component of the objective is to prolong the episode. We show that this algorithm can be used to effectively incorporate diverse skills into an overall skill set, and it outperforms prior methods in several environments.

VII. ACKNOWLEDGMENTS

Authors would like to thank Shubham Gupta for many useful discussions on this topic.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. G. I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *Neural Information Processing Systems*, 2013.

[2] O. Vinyals, T. Ewalds, S. Bartunov, P. Georgiev, A. S. Vezhnevets, M. Yeo, A. Makhzani, H. Küttler, J. Agapiou, J. Schrittwieser, *et al.*, "Starcraft ii: A new challenge for reinforcement learning," *arXiv preprint arXiv:1708.04782*, 2017.

[3] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[4] R. S. Sutton, D. Precup, and S. Singh, "Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.

[5] Y. Flet-Berliac, "The promise of hierarchical reinforcement learning," *The Gradient*, 2019.

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

[7] R. Dubey, P. Agrawal, D. Pathak, T. Griffiths, and A. Efros, "Investigating human priors for playing video games," in *International Conference on Machine Learning*, 2018, pp. 1349–1357.

[8] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual International Conference on Machine Learning*, 2009, pp. 41–48.

[9] J. L. Elman, "Learning and development in neural networks: The importance of starting small," *Cognition*, vol. 48, no. 1, pp. 71–99, 1993.

[10] M. Chevalier-Boisvert, F. Golemo, Y. Cao, B. Mehta, and L. Paull, "Duckietown environments for openai gym," https://github.com/duckietown/gym-duckietown, 2018.

[11] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International Conference on Machine Learning*, 2018, pp. 1861–1870.

[12] E. Lobo and S. Jordan, "Soft options critic," *arXiv preprint arXiv:1905.11222*, 2019.

[13] P. Dayan and G. E. Hinton, "Feudal reinforcement learning," in *Advances in neural information processing systems*, 1993, pp. 271–278.

[14] R. Parr and S. J. Russell, "Reinforcement learning with hierarchies of machines," in *Advances in neural information processing systems*, 1998, pp. 1043–1049.

[15] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *Journal of artificial intelligence research*, vol. 13, pp. 227–303, 2000.

[16] T. Kim, S. Ahn, and Y. Bengio, "Variational temporal abstraction," in *Advances in Neural Information Processing Systems*, 2019, pp. 11 570–11 579.

[17] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu, "Feudal networks for hierarchical reinforcement learning," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 3540–3549.

[18] C. Li, X. Ma, C. Zhang, J. Yang, L. Xia, and Q. Zhao, "Soac: The soft option actor-critic architecture," *arXiv preprint arXiv:2006.14363*, 2020.

[19] O. Nachum, S. S. Gu, H. Lee, and S. Levine, "Data-efficient hierarchical reinforcement learning," in *Advances in Neural Information Processing Systems*, 2018, pp. 3303–3313.

[20] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. McGrew, J. Tobin, O. P. Abbeel, and W. Zaremba, "Hindsight experience replay," in *Advances in neural information processing systems*, 2017, pp. 5048–5058.

[21] A. Levy, G. Konidaris, R. Platt, and K. Saenko, "Learning multi-level hierarchies with hindsight," in *Proceedings of International Conference on Learning Representations*, 2019.

[22] Z.-J. Pang, R.-Z. Liu, Z.-Y. Meng, Y. Zhang, Y. Yu, and T. Lu, "On reinforcement learning for full-length game of starcraft," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 4691–4698.

[23] K. Frans, J. Ho, X. Chen, P. Abbeel, and J. Schulman, "Meta learning shared hierarchies," in *International Conference on Learning Representations*, 2018.

[24] D. J. Mankowitz, T. A. Mann, and S. Mannor, "Iterative hierarchical optimization for misspecified problems (ihomp)," *arXiv preprint arXiv:1602.03348*, 2016.

[25] A. Bagaria and G. Konidaris, "Option discovery using deep skill chaining," in *the NeurIPS 2019 Workshop on Deep Reinforcement Learning*, 2019.

[26] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell, "Progressive neural networks," *arXiv preprint arXiv:1606.04671*, 2016.

[27] B. D. Ziebart, "Modeling purposeful adaptive behavior with the principle of maximum causal entropy," 2010, aAI3438449.

[28] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," in *arXiv preprint arXiv:1812.05905*, 2018.

[29] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine, "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–8.

[30] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "Safe, multi-agent, reinforcement learning for autonomous driving," *arXiv preprint arXiv:1610.03295*, 2016.

[31] K. Kang, S. Belkhale, G. Kahn, P. Abbeel, and S. Levine, "Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 6008–6014.

[32] M. Goslin and M. R. Mine, "The panda3d graphics engine," *Computer*, vol. 37, no. 10, pp. 112–114, 2004.