Robust Deep Skill Chaining

Author:
Matthew Slivinski

Advisor:
Prof. George Konidaris

Sc.M. Project Report

BROWN

Brown University
Department of Computer Science

May 15, 2020

Abstract

Hierarchical reinforcement learning methods that aim to autonomously discover temporally extended actions (or *skills*) need to be resilient to accommodate various tasks. We propose reliable methods that build upon the deep skill chaining algorithm for discovering skills in high-dimensional, continuous domains. We demonstrate empirically that these methods can replace previous routines while achieving the same performance in a continuous control task.

Acknowledgements

I would like to thank Akhil Bagaria, George Konidaris, and other members of the Intelligent Robot Lab at Brown for their constructive contributions to this project.

A special thanks to my partner, Lauren Marshall, for her tireless encouragement and guidance throughout my program. Additionally, I would like to thank my friends and family for their support.

Contents

1 Introduction

Methods in hierarchical reinforcement learning (RL) [2] focus specifically on the *options* framework [12] for planning, learning, and knowledge representation using high level-skills. Recent work extends the RL framework for acquiring useful skills (*i.e.* option discovery) in low-dimensional continuous domains using *skill chaining* [5] that assembles sequential options to target previously learned options within the environment. Moreover, *deep skill chaining* [1] scales skill chaining to high-dimensional continuous state and action environments using recent work in non-linear function approximation in RL. In order for deep skill chaining to extend to long-horizon tasks, its internal methods must be robust and dependable.

At the core of deep skill chaining (and option discovery in general), it is crucial to learn the set of states from which an option can be executed (*i.e.* the initiation set of an option) sequentially from a learned option policy. In skill chaining, an option's initiation set determines where the agent can execute an option and it serves as an intermediate goal (or salient event) for when the agent can terminate that option (*i.e.* the termination condition).

Current methods within deep skill chaining estimate the initiation set of an option by first gestating samples to fit an estimator then refine this estimator for a fixed period of time. Since options are temporally extended actions, the states represented within an estimate of an option's initiation set can change based on the samples collected throughout learning. If the initiation set of an option is fixed at any time during option discovery, then it does not accurately represent the *true* initiation set of an option since it doesn't factor all learned states.

Consequently, an issue can arise in skill chaining when an option's initiation set estimate changes because of the relationship between the initiation set and termination condition. As an agent learns a policy for an option, if the initiation set estimate changes (implying that the termination condition for the successor option also changes) then the option's policy must change. Subsequently, the option's policy will be incorrect at first where an option's initiation set estimate will decrease in size, resulting in the option's policy not hitting a salient event, and ultimately not solving the original task. While methods of learning an option's initiation set in deep skill chaining outperforms flat agents and other popular hierarchical methods in a series of challenging continuous control tasks, these learning methods are not robust in guaranteeing that a sufficient initiation set has been learned and overall, that the skill chain will not break.

We introduce new approaches for detecting and fixing breaks in the skill chain along with learning a reliable initiation set for each option. These techniques combine insights from deep skill chaining with support vector classification in supervised learning. Since these methods build upon previous work in deep skill chaining, these new methods will also scale to high-dimensional tasks with continuous state and actions spaces. Through various of experiments in the MuJoCo physics simulator [14], we aim to show that our novel approaches are not only leveled in learning performance but are reliably robust compared to previous work.

2 Background and Related Work

The standard RL framework for sequential decision making are finite *Markov Decision Processes* (MDPs) for a discrete time scale, $t = 0, 1, 2, \ldots$. For each time step, t, a learning agent observes a state within the state space, $s_t \in \mathcal{S}$, where the agent chooses a primitive action, $a_t \in \mathcal{A}_{s_t}$. When each a_t is taken, a reward, r_{t+1}, is produced from the reward function, $\mathcal{R}$, (where each step reward is discounted by $\gamma \in [0, 1]$), and a next state, s_{t+1}, is produced from the transition function, $\mathcal{T}$. We consider goal-oriented MDPs with the terminating goal state $g \in \mathcal{S}$ [11] in which the agent can call an indicator function, $\mathbb{1}_g : s \in S \to \{0, 1\}$, to determine if it has reached the goal state of the MDP.

Learning a policy in an MDP can be done by learning the action-value function $Q^\pi(s_t, a_t)$ that defines the expected sum of discounted future rewards of taking action a_t in state s_t under a policy, π

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi \left[r_t + \gamma \max_{a_{t+1}} Q^\pi(s_{t+1}, a_{t+1}) \right]. \tag{1}$$

A common off-policy algorithm, *Q-learning* [16], uses the action-value function to learn a greedy policy, $\pi(s_t) = \arg\max_{a_t} Q(s_t, a_t)$. Recent work scales Q-learning to high-dimensional spaces [8, 15, 6, 13] using non-linear function approximators parameterized by ϕ to learn the action-value function $Q^\pi_\phi(s_t, a_t)$ by minimizing the loss,

$$L(\phi) = \mathbb{E}_\pi \left[\left(Q^\pi_\phi(s_t, a_t) - y_t \right)^2 \right] \tag{2}$$

such that the Q-learning target [15], y_t, is

$$y_t = r_t + \gamma Q_{\phi'} \left(s_{t+1}, \arg\max_{a_{t+1}} Q_\phi(s_{t+1}, a_{t+1}) \right). \tag{3}$$

Specifically, *Deep Q-Learning* (DQN) [8] uses target networks, ϕ', and experience replay [7] to minimize $L(\phi)$ as a standard regression problem.

2.1 The Options Framework

The options framework [12] generalizes primitive actions to include temporally extended actions. An option, o, consists of three main components: an *option policy*, $\pi_o : \mathcal{S} \times \mathcal{A} \to [0, 1]$, that determines the closed-loop actions given a set of states, *initiation set*, $\mathcal{I}_o \subseteq \mathcal{S}$, that determines whether an option can be executed among states, and a *termination condition*, $\beta_o : \mathcal{S} \to [0, 1]$, for which the states an option must terminate.

Compared to MDPs, finite *semi-Markov Decision Processes* (SMDPs) [3] consider continuous-time transitions, $\Omega : \mathcal{S} \times \mathcal{A} \to (0, \infty)$, between each epoch of a decision that represents a history of all possible realizations of states rather than discrete-time transitions. With *semi-Markov* options [12], the policy and termination condition are augmented such that the policy, $\pi_o : \mathcal{S} \times \Omega \to [0, 1]$, and termination condition, $\beta_o : \Omega \to [0, 1]$, are mappings of prior events since the option was initiated with the set of all histories.

2.2 Deep Skill Chaining

Skill chaining first creates an option, o_g, that learns a policy, π_g, to initiate and terminate near a salient event, g, then recursively repeats this process until the start state of the MDP is within one of the options' learned initiation set. That is, skill chaining learns options such that the termination condition, $\beta_{o_i}(s_t)$, for the current option, o_i, is the initiation set, $\mathcal{I}_{o_{i-1}}(s_t)$, of the predecessor option, o_{i-1}, for all $i = 1, 2, \ldots$ options in the skill chain. These sequentially executable options guarantees that if an agent can successfully execute each option in the chain then it will solve the original task.

Deep skill chaining (DSC) extends this intuition through the following core components:

Initiation set classifiers: For each option to learn an estimate of its initiation set, $\hat{\mathcal{I}}_o$, N successful trajectories from the start state to the goal state is collected, where the last K states from each trajectory is segmented, known as the gestation period [5, 9]. Then, a one-class classifier is fitted around these K states. The states labeled within this classifier (*i.e.* label of $+1$) make up $\hat{\mathcal{I}}_o$. Once initially fitted, $\hat{\mathcal{I}}_o$ is refined for a fixed number of episodes by additionally collecting unsuccessful trajectories labeled as negative samples (where successful trajectories are labeled as positive samples) then fitting a two-class classifier on these trajectories. Afterward, a one-class classifier is fitted on the positive predictions from the two-class classifier.

Intra-option policy: Each option learns its own *intra-option policy*, $\pi_o : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^{|\mathcal{A}|}$, that is parameterized by θ_o, as a standard RL problem. *Deep Deterministic Policy Gradient* (DDPG) [6] is used to learn the intra-option policies for continuous real-valued action spaces.

Policy over options: A *global option*, o_G, is created that learns a *policy over options*, $\pi_{\mathcal{O}}$, to determine which option to execute out of all learned options $\mathcal{O}$ or select a primitive action.

To select an option at state s_t out of the set of possible executable options, $\mathcal{O}'(s_t)$, $\pi_{\mathcal{O}'(s_t)}$ chooses an option, $o' \in \mathcal{O}'(s_t)$, that maximizes its option-value function using DQN [8]. That is, given a SMDP Q-learning update [3] from time t to $t + \tau$, and target weights, ϕ, the Q-learning target y_t is

$$y_t = \sum_{t'=t}^{\tau} \gamma^{t'-t} r_{t'} + \gamma^{\tau-t} Q_{\phi'}(s_{t+\tau}, o_{t+\tau}) \tag{4}$$

such that

$$o_{t+\tau} = \underset{o' \in \mathcal{O}'(s_{t+\tau})}{\arg\max}\ Q_{\phi}(s_{t+\tau}, o'). \tag{5}$$

Using these components, the DSC algorithm first creates o_G where $\mathcal{I}_{o_G}$ contains all states within the environment (*i.e.* it can initiate everywhere) and β_{o_G} is true at the goal state of the MDP. When o_G terminates N times, a new option, o_k, is created to target $\beta_k = \mathcal{I}_{o_G}$ N times then trains to learn π_k while learning $\hat{\mathcal{I}}_{o_k}$. Next, a new node is added to the final layer of the DQN parameterized by $\pi_{\mathcal{O}}$ then the network is re-trained. When the start state

is within the initiation set estimate of a learned option, creation of options is halted, and training of all local policies and $\pi_{\mathcal{O}}$ continue for a fixed number of episodes.

3 Robust Deep Skill Chaining

3.1 Initiation Set Classifiers

Currently in DSC, once $\hat{\mathcal{I}}_o$ is learned, it is fixed. When continuous learning is applied, $\hat{\mathcal{I}}_o$ migrates towards the goal state (*i.e.* the *migrating option problem*). Given an option o_i and its predecessor o_{i-1}, this problem occurs due to the fact that as o_i collects samples, it will fit a one-class classifier around the samples that have densely targeted $\beta_{o_i} = \hat{\mathcal{I}}_{o_{i-1}}$. Therefore, as the option covering the goal state fits a compact ball around the goal state, this will cause all the descendent options to move closer to target the goal state as well.

Classifiers: Our proposed approach uses two separate core estimators to estimate an options initiation set; an *optimistic initiation set classifier*, $\hat{\mathcal{I}}_o^{OPT} : \mathcal{S} \rightarrow \{0, 1\}$, and a *pessimistic initiation set classifier*, $\hat{\mathcal{I}}_o^{PES} : \mathcal{S} \rightarrow \{0, 1\}$. The optimistic classifier is a two-class classifier fitted on the predictions from a one-class classifier that is trained on sampled states that have successfully triggered a salient event (or positive samples). Estimating the initiation set in this fashion allows for samples to be tolerantly labeled as positive samples while fitting a well defined boundary around these samples using a one-class classifier. Equally important, the pessimistic classifier is a one-class classifier fitted on the positively predicted samples of the optimistic classifier. The *optimistic set* consists of successfully triggered states that represent a liberal estimate of an option's initiation set from the optimistic classifier while the *pessimistic set* contains densely clustered successfully triggered trajectories from the pessimistic classifier where $\hat{\mathcal{I}}_o^{PES} \subseteq \hat{\mathcal{I}}_o^{OPT}$.

Data collection: To label samples, our method uses the procedure from previous work of classifying positive samples as states from which an option can successfully execute and negative samples as states that have failed to do so. In addition, we label states from o_{i-1}'s pessimistic set, that is

$$\hat{\mathcal{I}}_{o_{i-1}}^{PES}(s) = \mathbb{1} | \forall s \in \mathcal{S} \tag{6}$$

as negative samples for o_i to prevent the migrating option problem. Initially, samples are collected after o_i has had N successful trajectories, where the last K states from these trajectories is used among the samples for the optimistic set classifier. After N successful trajectories have been accomplished, we continuously collect positive samples that represent the last K states for successful trajectories and negative samples that represent the start state for failed trajectories; thus removing the initiation period from previous work.

Initiating and terminating: To initiate o_i, at state s_t at time t, s_t must be within o_i's optimistic set:

$$\hat{\mathcal{I}}_{o_i}^{OPT}(s_t) = \mathbb{1}. \tag{7}$$

Conversely, to terminate o_i, a given state must be within o_i's pessimistic set and o_{i-1}'s optimistic set:

$$\beta_{o_i}(s_t) = \hat{\mathcal{I}}^{PES}_{o_i}(s_t) \wedge \hat{\mathcal{I}}^{OPT}_{o_{i-1}}(s_t) = \mathbb{1}. \tag{8}$$

Termination is performed in this fashion to ensure the likelihood of successfully executing each skill sequentially. In other words, by restricting o_i's densely populated positive samples to intersect with o_{i-1}'s liberal initiation set estimate will yield less execution failures than targeting o_{i-1}'s initiation set estimate alone. Furthermore, this will allow $\pi_{\mathcal{O}}$ to initiate and terminate options in states that are more likely to succeed.

3.2 Chain Fixing Options

Skill chaining assumes that there exists a skill chain of sequentially executed options that covers every state state along a path from the start state to the goal state. In DSC, these options are either locally learned options or the global option chosen by $\pi_{\mathcal{O}}$, where local options are *only learned from the direct predecessor option.*

Consider that there exists a path from the start state to the goal state. Let's assume options o_1 and o_2 initiation sets have been sequentially learned (Figure 1a) and at a given time, $\pi_{\mathcal{O}}$ will choose to sequentially execute o_1 then o_2. Given that any method estimating an option's initiation set can change in size then it is possible for the current option's initiation set estimate to break apart from its predecessor option's initiation set estimate. That is, if o_1 and o_2 break apart, DSC will learn an option, o_{child}, that will target $\beta_{o_3} = \hat{\mathcal{I}}_{o_2}$ while leaving o_1 and o_2 "unlinked" in the skill chain (Figure 1c). By definition, if $\pi_{\mathcal{O}}$ chooses o_1 then o_2 to execute, then this method of creating options will not build a necessary skill chain since only creating child options will not cover the states along the path between o_1 and o_2.

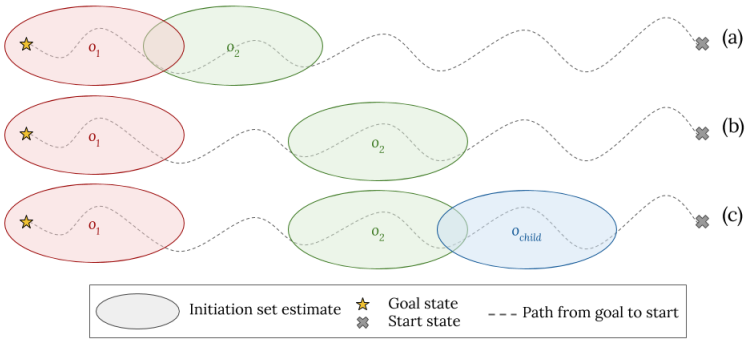


Figure 1: Sequence of when: (a) two sequential options are learned, (b) two option's initiation set estimates break apart, and (c) DSC learns new options.

To handle the skill chain breaking from two learned options, we propose that before learning a child option, a *chain fix option* is created that mends two broken options. First, to detect that a chain break has occurred, we will iterate over all learned options and verify that at least one state is labeled as positive from each o_i's optimistic set and o_{i-1}'s optimistic set.

Formally, if each o_i contains a record of discrete labeled states, $\mathcal{X}_{o_i}$:

$$\hat{I}_{o_i}^{OPT}(x_{o_i}) \wedge \hat{I}_{o_{i-1}}^{OPT}(x_{o_i}) = \mathbb{1}|\exists x_{o_i} \in \mathcal{X}_{o_i}. \tag{9}$$

Continuing the example from Figure 1, if a chain break develops, a chain fix option, o_{fix}, is created to target $\beta_{o_{fix}} = \hat{\mathcal{I}}_{o_1}$ while re-assigning o_2 to target $\beta_{o_2} = \hat{\mathcal{I}}_{o_{fix}}$ (Figure 2b). Once this option has been added to the chain, DSC can continue to create child options from o_2 (Figure 2c).

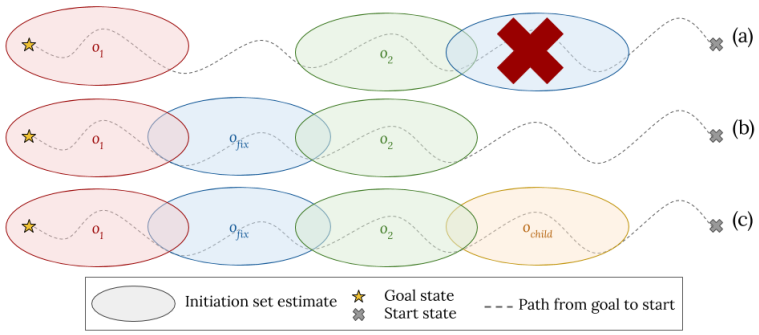


Figure 2: Sequence of (a) not creating a child option first but instead to (b) first create a chain fix option, and then (c) continue to create child options.

4 Comparative Analysis

We compared our methods to previous work in Point Maze [4] to outline a baseline for strong hierarchical reinforcement learning. In this task, a point agent with continuous locomotion must navigate around a U-shaped maze to reach its goal by receiving a -1 reward for every non-goal reaching step along with a sparse, terminating reward of 0 when it reaches the goal location. Since this is sparse reward setting, each option is given a sub-goal reward when it triggers its termination condition.

Migrating option problem: Figure 3 shows how the migration pattern with previous DSC methods in row (a) compare to our proposed methods in row (b) to learn $\hat{\mathcal{I}}_o$. Both previous DSC and robust DSC continuously take in samples with not being limited to the number of options an agent can learn. Note, that since $\hat{\mathcal{I}}_o^{PES}$ from the robust DSC methods represent the dense positive samples within $\hat{\mathcal{I}}_o^{OPT}$, it is not necessary to include it within this analysis. From our results, previous DSC methods learn options that aim to merge with the predecessor option and overtime will result in most of the options to converge to the goal state. Our proposed method learns options that spread out over the environment while preventing all learned options merging to the goal state.

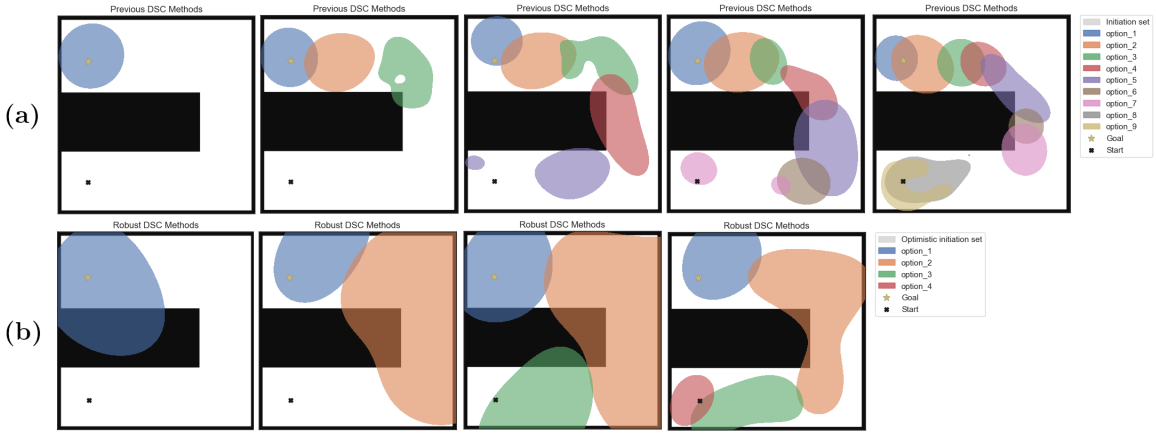


Figure 3: Migration pattern of all option's initiation set estimate with (a) previous DSC methods and (b) robust DSC methods.

Fixing chain breaks: In Figure 4, we illustrate the process of how our methods detect and fix the skill chain between two learned options. When a chain break is detected, we initiate that a chain fix option needs to be learned next in the main DSC control loop in order to precede in creating child options.

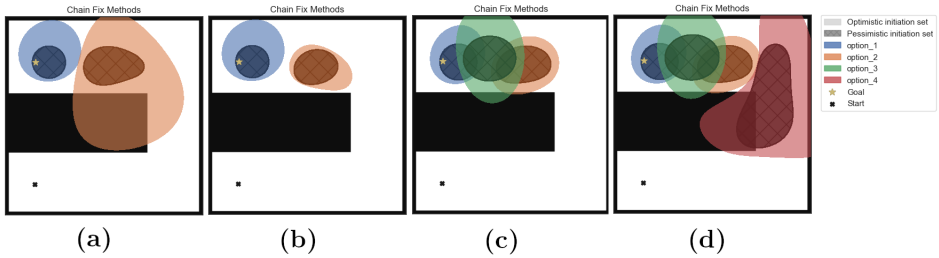
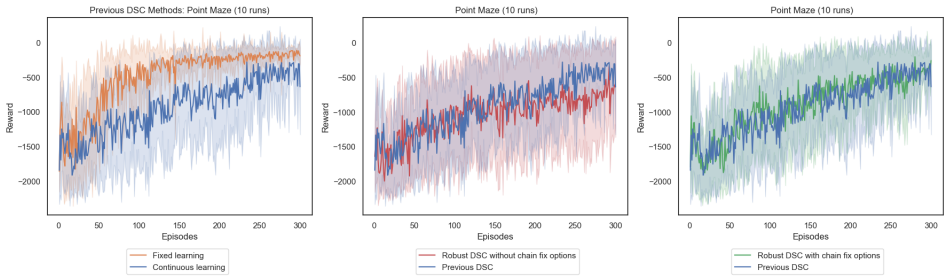


Figure 4: Learned options (a) before a chain break, (b) after a chain break, (c) when a chain fix option is created, and (d) when the process of creating child options continues.

Learning performance: All curves are averaged over 10 runs with the solid lines representing the median reward per episode and error bands representing one standard deviation. Figure 5a shows how the current DSC initiation set classifier methods compare to when the classifier takes in a fixed set of samples to a continuous set samples throughout the lifetime of an agent. The continuously learning agent not only under-performs the fixed learning agent but also has a drastic increase of variation in comparison. Next, we compare how the robust DSC perform with and without chain fixing options to previous methods that are all continuously learning in Figures 5b and 5c. Robust DSC without chain fixing options is matched in performance up to around 150 episodes when robust DSC starts to underperform in Figure 5b. Alternately, robust DSC with chain fixing options performance is relatively aligned to that of previous work. In either case of robust DSC, there is a high

variance of performance.



(a) (b) (c)

Figure 5: Learning curves comparing of (a) previous DSC methods that fix the intake of samples to methods that continuously take in samples, (b) robust DSC with chain fix options to previous DSC, and (c) robust DSC without chain fix options to previous DSC.

5 Discussion and Conclusion

Robust deep skill chaining aims to improve the deep skill chaining algorithm by enhance how options are learned. For this purpose, we show that our methods handle issues that arise with skill chaining with comparable performance to previous methods in one environment.

Deep skill chaining currently fails to handle the migrating option problem. This is due to the mix of imbalanced positive to negative samples, along with the nature of the initiation set classifier targeting densely positive samples. To handle this problem, our method uses the predecessor option's pessimistic set as negative samples for the current option to learn its optimistic initiation set classifier. As a result, learned options spread out, rather than migrating in, covering the environment and certify that all learned options do not converge to the goal state.

If the policy over options chooses to execute two learned options sequentially, and their initiation set estimates do not overlap, then a chain break will occur, resulting in a decrease in performance. We emphasize formally how to detect and fix such a chain break while incorporating these methods into the current DSC algorithm. In practice, these methods effectively connect two learned options that do not have joint initiation set estimates while continuing to learn options that lengthen the skill chain.

We highlight the importance of continuous learning of an option's initiation set where current deep skill chaining techniques perform worse under such conditions, compared to that of fixed learning. While obtaining a more accurate and up-to-date representation of an options initiation set, continuous learning also destabilizes the accuracy of an agent's performance. Since the reward is based on successfully executing either the global option or a locally learned option, continuous learning negatively affects the Q-values which ultimately impacts learning a sufficient policy over options.

Our work is no worse than previous methods with chain fixing options, and relatively worse without chain fixing options, when taking in continuous samples. In both cases, it is difficult

to state if chain fixing options will improve performance. If a chain break occurs between two locally learned options, the policy over options could choose to take the global option instead and learn that it is better to do so over a locally learned option. However, we hypothesize that for long horizon tasks, it will be more beneficial to take locally learned options rather than primitive actions (*i.e.* the global option) since the distribution of the Q-values for the global option will be minimal and sparse, compared the representative and compact distribution from a locally learned option. Future work aims to further develop DSC by understanding how continuous learning effects the performance of an agent and if chain fixing options are a necessity for solving long horizon tasks.

In general, exploration affects how options are learned and can have negative impacts on their learning, especially with chain fixing options. For example, current DSC methods use a linear exploration decay schedule since it assumes that options are created sequentially backward from the goal. If options can be created in-between already learned options (*e.g.* chain fixing options), then a linear decay exploration schedule will negatively impact the how that option learns to get to a salient event, especially if that salient event is far reaching. Thus, a more dynamic exploration schedule is needed to accommodate these types of options which is another avenue for future work.

Skill discovery is at the core of hierarchical reinforcement learning. We demonstrated novel and stable procedures that not only strengthen skill discovery in the DSC algorithm, but to ultimately advance how hierarchies reliably solve long-horizon sequential decision making problems.

A Appendix

A.1 Implementing Optimistic and Pessimistic Initiation Set Classifiers

In order to implement the optimistic and pessimistic initiation set classifiers as described in Section 3.1, we used scikit-learn's `OneClassSVM` and `SVC` packages [10]. These classifiers were learned over the x and y position rather than all state variables due to the position being the only relevance to learning the initiation set in Point-Maze.

A.2 Hyperparameter Settings

The set of hyperparameters relative to DSC, with the exception of the initiation classifiers, were identical to those of previous work. Refer to [1] for more details. The parameters used for initiation classifiers are listed in Tables 1, 2, and 3.

Parameter	**Value**
Kernel	`rbf`
Nu (ν)	0.1
Gamma	`scale`

Table 1: Optimistic Initiation Set Classifier (One-Class SVM)

Parameter	Value
Gamma	`scale`
Class Weight	`balanced`

Table 2: Optimistic Initiation Set Classifier (Two-Class SVM)

Parameter	Value
Kernel	`rbf`
Nu (ν)	0.3
Gamma	`scale`

Table 3: Pessimistic Initiation Set Classifier (One-Class SVM)

The main tuning parameter for the one-class classifiers is ν that sets an upper bound of mislabeled training data. That is, the higher ν is, the more compact the prediction boundary was. For the optimistic classifier to have a wide decision boundary, ν was tuned fairly low. However, since the pessimistic classifier for an option acts as negative samples for the successor option, ν was set low enough to ensure that sequentially learned options sufficiently overlapped to prevent a chain break upon creation. Tuning these parameters was done by visualizing the learned initiation sets during training (e.g. Figure 3). Fine tuning ν for both classifiers is left up to future work.

A.3 Computing Environment and Time

We used 1 NVIDIA GeForce GTX 1650 and 1 Tesla K80 on the Google Cloud compute infrastructure to perform all experiments reported in this paper. With the format `hh:mm:ss`, each run on average took $01:35:45 \pm 00:34:51$ compared to $01:32:36 \pm 00:33:13$ in previous work.

References

[1] Akhil Bagaria and George Konidaris. Options discovery using deep skill chaining. *ICLR*, pages 1–21, 2020.

[2] Andrew G Barto and Sridhar Mahadevan. Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dynamic Systems*, 13(4):41–77, 2003.

[3] Steven J. Bradtke and Michael O. Duff. Reinforcement Learning Method for Continuous-Time Markov Decision Problems. *Advances in Neural Information Processing Systems*, 7:393–400, 1994.

[4] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. *CoRR*, abs/1604.06778, 2016.

[5] George Konidaris and Andrew Barto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems 22 - Proceedings of the 2009 Conference*, pages 1015–1023, 2009.

[6] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. Technical report, 2016.

[7] Long-Ji Lin. *Reinforcement learning for robots using neural networks.* PhD thesis, Carnegie Mellon University, 1993.

[8] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[9] Scott Niekum and Andrew G Barto. Clustering via Dirichlet process mixture models for portable skill discovery. In *Advances in Neural Information Processing Systems 24: 25th Annual Conference on Neural Information Processing Systems 2011, NIPS 2011*, 2011.

[10] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[11] Richard Sutton and Andrew Barto. *Reinforcement Learning: An Introduction.* The MIT Press, 2018.

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

[13] Gerald Tesauro. Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Comput.*, 6(2):215–219, 1994.

[14] Emanuel Todorov, Tom Erez, and Yuval Tassa. MuJoCo: A physics engine for model-based control. In *IEEE International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012.

[15] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double Q-Learning. In *30th AAAI Conference on Artificial Intelligence, AAAI 2016*, pages 2094–2100, 2016.

[16] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. In *Machine Learning*, pages 279–292, 1992.