Unsupervised Prototype Learning: Effect-Driven Action Space Discretization for RL

Anonymous authors
Paper under double-blind review

Abstract

Reinforcement learning is typically used to train an agent to perform a task as defined by a task-specific, often carefully crafted, reward function. While this approach achieves good results, the collected data is almost useless for learning policies for other downstream tasks and rewards. We propose a new unsupervised approach to reward-agnostic prototype learning that uses an effect-driven discretization of the action space to discover a set of discrete action prototypes that produce a robust and reliable effect in the environment when executed. We evaluate our method on a simulated stair-climbing reinforcement learning task, and the results show that our set of prototypes generated by an effect-driven action space discretization outperforms uniform grid and randomly sampled discretizations in convergence speed and maximum reward on a RL task with three different reward functions.

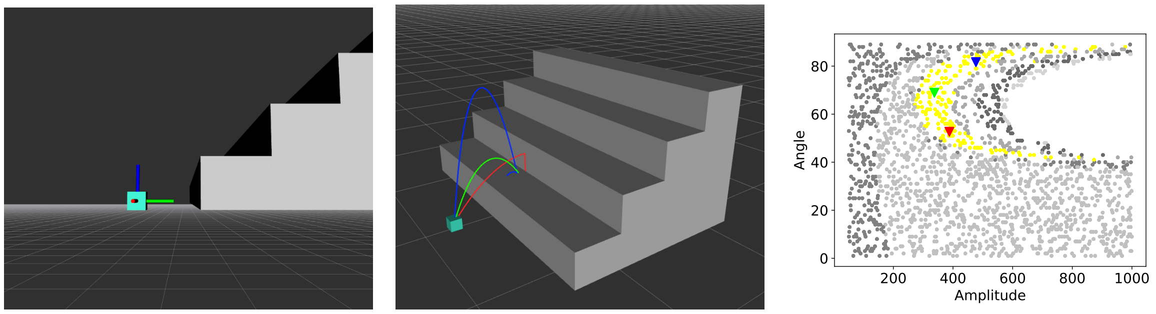
1 Introduction

Reinforcement learning applications such as robotics often exhibit a continuous action space by design. Continuous action spaces imply an infinite number of different actions, which complicates the search for optimal actions. If the action space can be discretized, the underlying control problem would be simplified. Recent work has shown that agent performance can be improved by choosing a discretized distribution for the policy (Tang & Agrawal, 2020; Seyde et al., 2021). Tang & Agrawal (2020) discretizes the policy along the action dimensions, and each action dimension is discretized with a uniform grid. Seyde et al. (2021) replaces the Gaussian policy with a discrete Bernoulli policy and discretizes each action dimension with its two extreme values, so-called bang-bang control. This surprisingly effective approach achieves state-of-the-art performance on several continuous control problems. However, this approach does not translate well to an actual robotics application: maximal torque actions increase wear and, in the worst case, can even break the robot.

This raises the question of whether there is a single discrete distribution that covers all control problems or if each problem has a different optimal distribution for discretization to capture the underlying dynamics of the control task. Our work proposes a different approach where we transform a continuous control environment into a conceptually more straightforward discrete control problem, which is solved using conventional discrete action RL. The action discovery process is guided by the effects that actions have on the environment. It, therefore, does not scale exponentially with the size of the action space, nor is it limited to only two actions per action dimension.

We developed an algorithm to generate what we call "action prototypes". Each prototype is a set of action parameters that allows the agent to produce a specific type of effect. Generating these prototypes involves a) finding out what types of effects are possible in a given environment through exploration, b) building the class of equivalent effects, and c) finding a set of representative action parameter sets that allow achieving such effects reliably.

Contributions Our research contributes to the field by formalizing and implementing an effect-centric action space discretization algorithm. To demonstrate its practicality, we create a stair-climbing environment in Gazebo with a Gym-Wrapper, which provides a continuous action and



(a) Environment initial position (b) Action prototypes (c) One step jump effect cluster

Figure 1: Different actions lead to the same effect on the environment. By discretizing the agents to cover the effect spaces, redundant actions are eliminated, and the action space size is reduced. Our method generates a set of prototypes that let the agent produce a specific type of effect. To test the generated discrete action prototypes, we build a stair-climbing environment (Initial position (a)) where the agent (turquoise cube) can jump up a staircase by applying a force to the center of mass of the cube, parameterized by a force angle and a force magnitude. After a short unsupervised sampling phase, the agent clusters the collected actions based on the effect they produce. We generate a set of prototypes for each effect cluster covering the underlying action distribution of each effect region. (b) showcases three actions selected in one single effect region by our method (i.e., all amount to the effect of a one-step height gain). (c) displays the collected samples in action space with the effect region (highlighted in yellow) of the prototypes from (b) and their respective markers.

state space ideal for action space discretization evaluations. Furthermore, we conduct a comparative evaluation of our proposed effect-centric discretization algorithm and naive baseline discretization approaches in this stair-climbing environment. The results show that our set of prototypes generated by an effect-driven action space discretization outperforms naive baseline discretizations in convergence speed and maximum reward on a RL task with three different reward functions.

The rest of this paper is organized as follows: Section 2 reviews existing work on effect-based representations and action space discretization, Section 3 describes our multi-step approach relying on unsupervised methods to build effect and action representations. In Section 4, we describe the environment in which we evaluate this method on three different reward functions on the environment and compare it to naive action space discretization baselines. Section 5 covers current limitations of our approach and lays out some possible future research directions.

2 Related Work

In robot learning, a large part of the research effort is dedicated to learning (encoding) trajectories useful to solve a task, as shown by several surveys on robot manipulation (Gams et al., 2022; Kroemer et al., 2021). Standard representations include motion primitives, as often done in learning by demonstration, and policies learned with reinforcement learning. The current state-of-the-art approach to learning motions in robotics is to train an end-to-end deep neural network through reinforcement learning (Saeed et al., 2021; Wang et al., 2022; Han et al., 2023). However, these approaches are limited by the large amount of training data needed and the training time required for the learning process to converge (Levine et al., 2016).

For complex real-world tasks, it is often difficult to accurately translate desired behavior into a reward function, which recent work tries to solve by reward function learning (Michaud et al., 2020). To cope with a changing reward function in RL, a reward-agnostic approach is necessary to manage a broad set of potential downstream rewards. Dann et al. (2023) uses multiple alternate reward formulations to drive faster learning. One approach in this work uses reward function disagreement to eliminate actions, which, similarly to discretization, restricts the space of viable actions, leading to better generalization in terms of reward. This method allows the agent to learn a policy that performs well across all pre-selected reward functions.

The dependency on a reward function is lifted when explicitly considering the effects: A step towards learning a relevant effect representation is made by Uyanik et al. (2013). Despite using pre-coded actions, they compute effect codes based on the continuous variations of the features in effect space. This representation produces effect equivalence classes based on the mean and standard deviation of effects produced by one action. No supervision is required, and effect classes are thus autonomously discovered. The limitation here is that actions being hard-coded, some have multiple outcomes, contrary to the idea that the effect defines the action.

Recent efforts in action space discretization focus on adapting the policy to accommodate discrete action selection. Tang & Agrawal (2020) defines the policy via categorical distribution over action dimensions. This approach archives significant gains with state-of-the-art on-policy optimization algorithms (PPO (Schulman et al., 2017), TRPO (Schulman et al., 2015), ACKTR (Wu et al., 2017)), especially for high dimensional tasks with complex dynamics. Seyde et al. (2021) employs Bernoulli policies instead of the standard Gaussian policy and achieves state-of-the-art performance on several continuous control benchmarks. In our work, we try to go even further into the discrete setting by autonomously pre-selecting a set of discrete action prototypes to employ classical discrete control algorithms such as DQN (Mnih et al., 2013).

3 Methodology

Our approach tackles the task of converting a continuous action space into a discrete action space in three stages: unsupervised data collection through random action sampling, effect region clustering, and discrete action prototype generation for each effect region. This results in a set of discrete action prototypes where each prototype performs a reliable effect in the environment. All three stages operate unsupervised, and the resulting discrete actions can be used in a downstream discrete action RL algorithm.

3.1 Unsupervised sample based data collection

The initial setting for this method is an environment with a continuous state space $\mathcal{S}$ defined as $\mathcal{S} = \bigtimes_{i=1}^{n} S_i$ where each S_i is a bounded interval for the ith feature value ($\bigtimes$ denotes the Cartesian Product). We consider a continuous action space $\mathcal{A} = \bigtimes_{i=1}^{n} A_i$ where A_i takes values in a bounded interval. We define an effect e_t at time step t as follows

$$e_t = s_t - s_{t-1} \quad \forall t \,:\, s_t \in \mathcal{S} \tag{1}$$

where $\mathcal{S}$ is a state space and s_t and s_{t-1} are two consecutive states linked by action a_{t-1}. At this stage, the agent samples a random action a_t uniformly from action space $\mathcal{A}$. When performing an action, our method focuses on the effect it produces in the environment and stores the resulting (a_{t-1}, e_t) tuples in Ω. Pseudocode for this stage is given in Algorithm 1.

3.2 Effect region clustering

Effect region clustering is achieved by grouping the effects resulting from the sampled actions into effect classes $\mathcal{C}_k$, as described by Algorithm 2. A different method for grouping the samples into effect categories is selected depending on the specified number of relevant effect dimensions. If only one effect dimension is interesting for the task, simple histogram binning groups the samples into the respective categories. Otherwise, we opt for the K-Means algorithm (MacQueen et al., 1967) to cluster effects in multidimensional space. In order to find the best number of classes to represent the data, we perform multiple iterations of K-Means while increasing the number of generated centroids. The iteration that produces clusters with the highest silhouette score is considered the best representation of possible effects and used as $\mathcal{C}_k$. In each version, labels are amended to each sample generated at the previous stage, resulting in (a_{t-1}, e_t, k) tuples.

Algorithm 1 Unsupervised sample based data collection

```
Require: $\mathcal{S} = \bigtimes_{j=1}^{n} S_j$ and $\mathcal{A} = \bigtimes_{j=1}^{n} A_j$    ▷ Action space $\mathcal{A}$ and state space $\mathcal{S}$
Return: $\Omega$    ▷ Collection of (action, effect) tuples
  $i \leftarrow 0$
  while $N \geq i$ do
      Initialize $s$    ▷ Reset environment at each sample
      Sample $a_i \sim \mathrm{U}(\mathcal{A})$
      Perform action $a_i$ and observe $s'$
      $e_i = s' - s$
      Store $(a_i, e_i)$ to $\Omega$
      $i \leftarrow i + 1$
  end while
```

Algorithm 2 Effect region clustering

```
Require: $N$ action-effect tuples $\Omega$, $\psi$ max clusters
Return: $\mathcal{C}_1, \dots, \mathcal{C}_N$    ▷ $N$ distinct Effect Regions
  if $\dim(e) > 1$ then
      $\theta \leftarrow 0$    ▷ maximal silhouette score
      $\phi \leftarrow 2$    ▷ best number of clusters
      for each $i \in [2, \dots, \psi]$ do
          $s \leftarrow$ silhouette score of KMEANS$(\{e \in \Omega\}, \text{clusters} = i)$
          if $\theta < s$ then
              $\theta \leftarrow s$
              $\phi \leftarrow i$
          end if
      end for
      $\{\mathcal{C}_k : k \in [1, \dots \phi]\} \leftarrow$ KMEANS$(\{e \in \Omega\}, \text{clusters} = \phi)$
  else
      $\text{borders}, \text{bins} \leftarrow \text{Histogram}(\{e \in \Omega\})$
      Create $\mathcal{C}_k$ for non-empty bins
      for each $(e, a) \in \Omega$ do
          Append $(e, a)$ to $\mathcal{C}_k$ where $e \in \text{borders}(\mathcal{C}_k)$
      end for
  end if
```

3.3 Action prototype generation

Each class $\mathcal{C}_k$ constitutes a collection of similar effects, each with respective actions that caused the effect. This one-to-one relationship allows us to group the sampled actions by their associated effect label k. In order to achieve an effect with the effect label k, any action from $\mathcal{C}_k$ can be performed. However, instead of maintaining all these individual actions, we want to find a small number of $p_k \in \mathcal{A}$ for each effect class $\mathcal{C}_k$. All p_k are action prototypes that are available actions to the downstream discrete reinforcement learning agent.

We selected the RGNG (Robust growing neural gas) algorithm by Qin & Suganthan (2004) to generate the action prototypes. This algorithm utilizes a combination of topological learning and outlier resilience, which stays true to the underlying sample topology of the input data.

RGNG needs a maximally allowed number of nodes on initialization. We consider an effect-driven approach to calculate this number based on the underlying effect samples of class $\mathcal{C}_k$. For each class k, we determine the mean μ_k^e and the standard deviation σ_k^e in effect space. After normalizing the mean and standard deviation across all classes, we calculate the number of prototypes ξ_k for each class according to Equation (2).

$$\xi_k = \left\lfloor \frac{(1 - \mathrm{cv}_k) \cdot \mathrm{std}_k}{\min_k((1 - \mathrm{cv}_k) \cdot \mathrm{std}_k)} \right\rfloor \tag{2}$$

$$\mathrm{cv}_k = \frac{\sigma_k^e}{\mu_k^e} \tag{3}$$

where cv_k is the coefficient of variation (defined in Equation (3)). This is a statistical measure of dispersion relative to the mean. A high coefficient of variation means the action collection in $\mathcal{C}_k$ has much uncertainty in effect space compared to the mean effect; therefore, the first term reduces the number of prototypes for actions in unreliable effect regions. The second term reduces the number of prototypes for actions in robust effect regions with little variability in effect space to avoid action overlap. The division by the minimal value ensures that each encountered class gets at least one prototype. Finally, we run either RGNG or calculate the action collections mean μ_k^m to generate the action prototypes p_k for each class $\mathcal{C}_k$

$$p_k = \begin{cases} \mu_k^m & \text{if } \ \xi_k == 1 \\ \mathrm{RGNG}(\xi_k) & \text{if } \ \xi_k > 1. \end{cases} \tag{4}$$

4 Results

We evaluate the algorithm in a simulated environment where a simple agent can move by “jumping”, i.e., applying a force to its center of mass. Figure 1 illustrates the setup. The “jumping” command is

$$a = \{\alpha, \mu\}$$

where α is the angle of the force perpendicular to the stairs, and μ is its amplitude. The agent receives

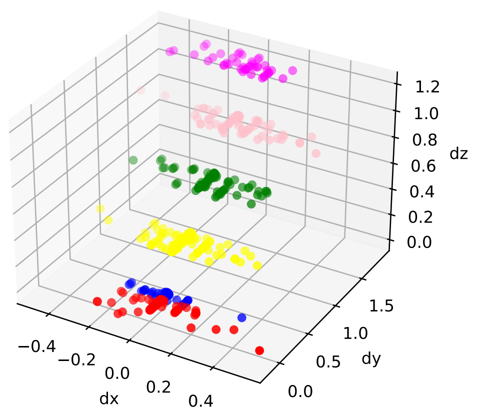
$$s = \{x, y, z, qx, qy, qz, qw, d_{obs}\}$$

as features, where x, y, z encode the position in the world frame, qx, qy, qz, qw is the quaternion representing the orientation and d_{obs} the distance to the next obstacle on the x-axis.

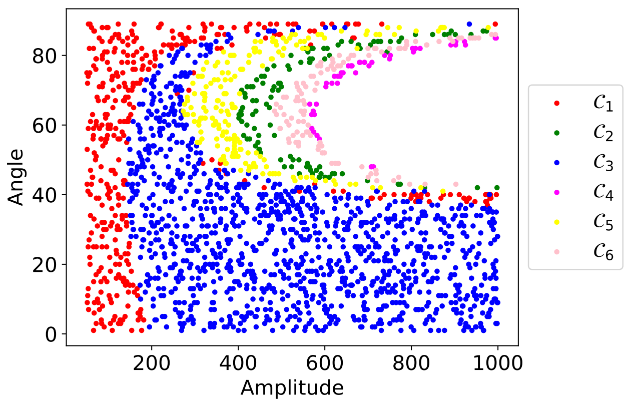
To test if the different sets of prototypes (effect-based, randomly sampled, and uniform grid) are reward-agnostic, we define three reward functions for a downstream RL task on the stair-climbing environment with a staircase 15 steps high in which the agent gets rewarded for height gain at each action. The reward functions reward exclusively: climbing one step at a time, two steps at a time, or three steps at a time.

During the unsupervised data collection phase, we empirically found that 2000 uniformly sampled actions from the initial starting position 1a from the two-dimensional action space are sufficient to form promising action prototypes. A trade-off that has to be considered here is between the time consumption of sampling and the quality of prototypes. Figure 2 shows the k-means clustering result on the collected data in effect space visualized in effect 2a and action space 2b. The method can find meaningful classes on the y-z effect space. Classes $\mathcal{C}_1$ and $\mathcal{C}_3$ correspond to no strict change in the environment where the agent does not reach the first step. In classes $\mathcal{C}_2$, $\mathcal{C}_4$, $\mathcal{C}_5$, $\mathcal{C}_6$, significant changes in the position of the agent are produced. Each subsequent class (in order $\mathcal{C}_5$, $\mathcal{C}_2$, $\mathcal{C}_6$, $\mathcal{C}_4$) represents the agent making it one step higher on the stairs. The empty area in the top right of Figure 2b corresponds to overshooting of the stairs, which we excluded in the final clustering.

Figure 3 shows the action prototype found by our method/other methods. Our effect-based discretization method finds distinct prototypes in significant effect areas. It creates prototypes quantified by the magnitude of the effect combined with the variation in effect, ultimately generating more prototypes in interesting effect regions. The uniform grid and random prototype selection fail to capture a large portion of the high-effect areas and cannot follow the underlying topology in the effect space.

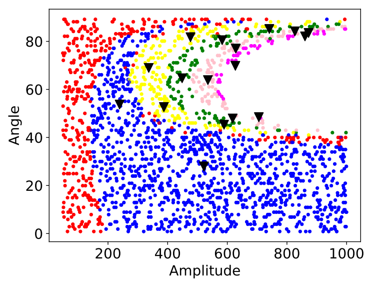


(a) Effect classes in effect space

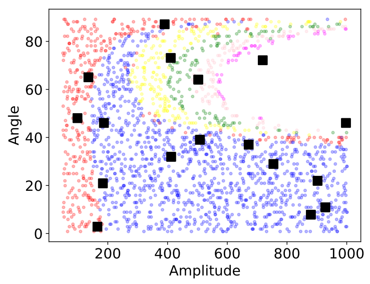


(b) Effect classes in action space

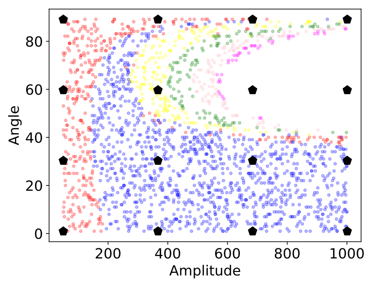
Figure 2: The result of the k-Means effect region clustering in y and z space is visualized in (a) effect space and in (b) action space. (b) shows coherent classes even though the clustering was performed in effect space, which reaffirms these two spaces' correlation. Classes $\mathcal{C}_1$ and $\mathcal{C}_3$ correspond to no strict change in the environment where the agent does not reach the first step. The other classes represent a one-, two-, three-, or four-step height gain.



(a) Effect-centric prototypes



(b) Random prototypes

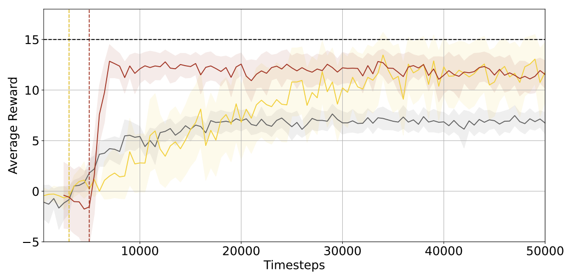


(c) Uniform grid prototypes

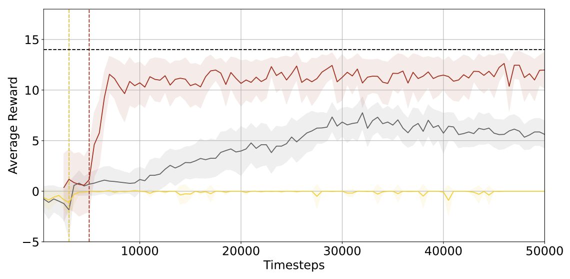
Figure 3: Visualization of the action prototypes (visualized as regular polygons) found by (a) our effect-centric discretization method, (b) random discretization and (c) uniform grid discretization method. The uniform grid and random prototype selection fail to capture a large portion of the high-effect areas and cannot follow the underlying topology in the effect space. The colors of effect classes serve only as visual cues and do not affect methods in (b) and (c).

In our evaluation, we used the SAC and the DQN implementation of Stable-Baselines3 (Raffin et al., 2021) as reinforcement learning agents. As a baseline with continuous action space control, we trained SAC agents (Haarnoja et al., 2018) on all three reward functions (seperately: different reward function, different agent). Additionally, we trained Deep Q-learning (Mnih et al., 2013) agents for each of our effect-based, uniform grid and random prototypes.

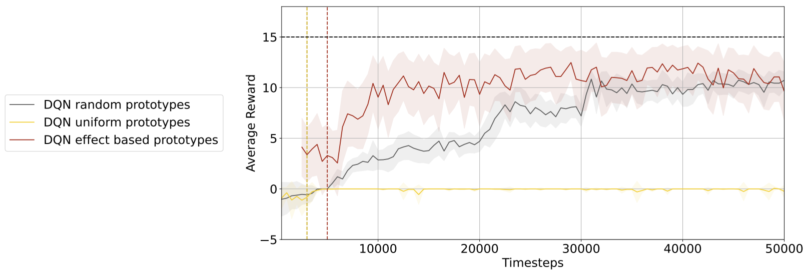
Figure 4 shows the learning curves for all evaluated rewards and discrete RL agents (DQN effect based, DQN random, DQN uniform grid). The plots show the mean reward with its respective standard deviation for each agent evaluated five times for each of the ten seeds after every 500 timesteps on each reward function. The results in this environment show that the *effect-centric* prototype generation is robust and converges reliably to high-reward solutions for all three tested reward functions (4a, 4b, 4c). The agent based on the *uniform grid* discretization method performs well with the one-step reward function (4a) but fails to solve the problem for the two and three-step reward functions (4b, 4c). The uniform grid prototypes rely heavily on the density of the



(a) Learning Curve Comparison "One Step" reward



(b) Learning Curve Comparison "Two Step" reward



(c) Learning Curve Comparison "Three Step" reward

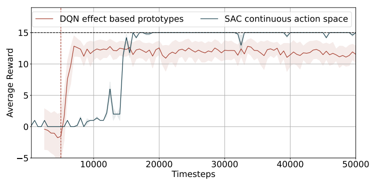
Figure 4: RL learning curve comparisons of all three sets of discrete action prototypes (random, uniform grid, and effect-based) on all three reward functions (subfigures a,b, and c). The vertical dotted lines symbolize the learning start of each DQN agent (the red line is effect-based, and the yellow is random and uniform grid-based). The horizontal black dashed line represents the maximally achievable reward in each plot. The learning curve for the effect-based prototypes is adjusted for the 2000 samples needed in the prototype discovery process. The effect-centric prototypes converge faster than every alternative discretization method. Additionally, the effect-based set of prototypes reliably converges to a high-reward solution while the uniform grid and random sampling method fail or underperform under some reward functions, making them less robust.

discretization grid. A denser grid would potentially allow the algorithm to learn in this environment but would be less feasible for higher dimensional action space environments. The agent based on *random* prototypes manages to learn with each reward function but under-performs for the one and two-step reward functions compared to the effect-based agent (4a, 4b). The randomness allows the method to capture only a few promising action prototypes: since the rewarding actions (depending on which reward is chosen, one-step, two-step, or three-step height gain on the stairs) are not evenly distributed across the action space, a lot of not beneficial actions are chosen as well.

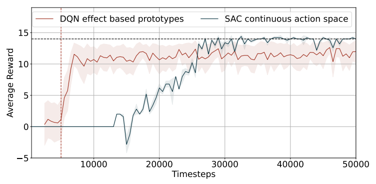
Figure 5 is the same experimental setup as Figure 4 but compares the effect-based DQN agents to continuous action space baseline SAC agents. The effect-centric approach converges significantly faster than the continuous control baseline, with a marginal decrease in long-run performance in all three reward settings. We assume that the SAC baseline agent converges to an optimal return solution due to the more fine-grained control in continuous action space, which the discrete counterpart does not offer.

5 Future Work

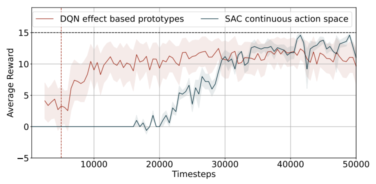
One current limitation is the need for discrete effect spaces for clear effect class separation: discrete effect spaces manifest through fixed boundary conditions in the environment (i.e., immovable stairs), which cause the effect class discovery to form clear boundaries as shown in Figure 2. In contrast, purely continuous effect spaces (e.g., free space navigation) do not exhibit emergent effect category borders. In future work, we want to investigate whether effect-based discretization still yields benefits over the simpler uniform grid prototypes in purely continuous effect spaces.



(a) "One Step" reward task



(b) "Two Step" reward task



(c) "Three Step" reward task

Figure 5: RL learning curve comparisons of DQN with discrete effect-based action prototypes and the SAC continuous control baseline on all three reward functions (subfigures a,b, and c). The vertical red dotted line symbolizes the learning start of the DQN agent. The horizontal black dashed line represents the maximally achievable reward in each plot. The learning curve for the effect-based prototypes is adjusted for the 2000 samples needed in the prototype discovery process. The effect-centric prototypes converge faster than the SAC continuous control baseline. All agents on all the reward functions reliably converge to a high-reward solution. The SAC baseline agent converges to an optimal return solution due to the more fine-grained control in continuous action space.

The naive grid-based discretization approach scales exponentially with the growing number of dimensions of action space. In contrast, the prototypes of the effect-centric discretization approach are not linked to the dimensionality of the action space but to the predefined number of maximum prototypes. In future work, we want to investigate if our approach can capture enough significant prototypes in higher-dimensional settings without exponentially increasing the number of prototypes.

6 Conclusion

In this work, we presented a novel reward-agnostic approach to action space discretization grounded in the action effects. This approach only makes assumptions about the state dimensions where effects are measured and does not rely on the design of a reward function. Our method shows promising preliminary results: in our toy stair-climbing environment, effect-based action space discretization outperforms uniform grid and randomly sampled action space discretization in convergence speed and maximum reward on three downstream RL tasks defined by different reward functions.

References

Christoph Dann, Yishay Mansour, and Mehryar Mohri. Reinforcement learning can be more efficient with multiple rewards. In *International Conference on Machine Learning*, pp. 6948–6967. PMLR, 2023.

Andrej Gams, Tadej Petrič, Bojan Nemec, and Aleš Ude. Manipulation learning on humanoid robots. *Current Robotics Reports*, 3(3):97–109, 2022.

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

Dong Han, Beni Mulyana, Vladimir Stankovic, and Samuel Cheng. A survey on deep reinforcement learning algorithms for robotic manipulation. *Sensors*, 23(7), 2023. ISSN 1424-8220. doi: 10.3390/s23073762. URL https://www.mdpi.com/1424-8220/23/7/3762.

Oliver Kroemer, Scott Niekum, and George Konidaris. A review of robot learning for manipulation: challenges, representations, and algorithms. *J. Mach. Learn. Res.*, 22(1), jan 2021. ISSN 1532-4435.

Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 17(1):1334–1373, 2016.

James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pp. 281–297. Oakland, CA, USA, 1967.

Eric J Michaud, Adam Gleave, and Stuart Russell. Understanding learned reward functions. *arXiv preprint arXiv:2012.05862*, 2020.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

A Kai Qin and Ponnuthurai N Suganthan. Robust growing neural gas algorithm with application in cluster analysis. *Neural networks*, 17(8-9):1135–1148, 2004.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

Muhammed Saeed, Mohammed Nagdi, Benjamin Rosman, and Hiba H. S. M. Ali. Deep reinforcement learning for robotic hand manipulation. In *2020 International Conference on Computer, Control, Electrical, and Electronics Engineering (ICCCEEE)*, pp. 1–5, 2021. doi: 10.1109/ICCCEEE49695.2021.9429619.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization. *CoRR*, abs/1502.05477, 2015. URL http://arxiv.org/abs/1502.05477.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL http://arxiv.org/abs/1707.06347.

Tim Seyde, Igor Gilitschenski, Wilko Schwarting, Bartolomeo Stellato, Martin Riedmiller, Markus Wulfmeier, and Daniela Rus. Is bang-bang control all you need? solving continuous control with bernoulli policies. *Advances in Neural Information Processing Systems*, 34:27209–27221, 2021.

Yunhao Tang and Shipra Agrawal. Discretizing continuous action space for on-policy optimization. In *Proceedings of the aaai conference on artificial intelligence*, volume 34, pp. 5981–5988, 2020.

Kadir Firat Uyanik, Yigit Caliskan, Asil Kaan Bozcuoglu, Onur Yuruten, Sinan Kalkan, and Erol Sahin. Learning social affordances and using them for planning. In *CogSci 2013*, Berlin, Germany, July 31–August 2 2013.

Tianying Wang, En Yen Puang, Marcus Lee, Yan Wu, and Wei Jing. End-to-end reinforcement learning of robotic manipulation with robust keypoints representation, 2022.

Yuhuai Wu, Elman Mansimov, Shun Liao, Roger B. Grosse, and Jimmy Ba. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. *CoRR*, abs/1708.05144, 2017. URL http://arxiv.org/abs/1708.05144.

A Which actions get selected by a trained SAC agent?

To verify the validity of the effect clusters formed by our method, we illustrate the overlap of said effect regions with the actions taken by a fully trained SAC agent. In Figure 6, we plot the actions taken by a SAC agent trained with the one-step at a time reward function (circles) and one trained on the two-steps at a time reward function (squares). The visualization clearly shows that the selected actions overlay the effect regions for the respective effect of one-step height gain and two-step height gain, which are optimal actions to solve the tasks at hand.

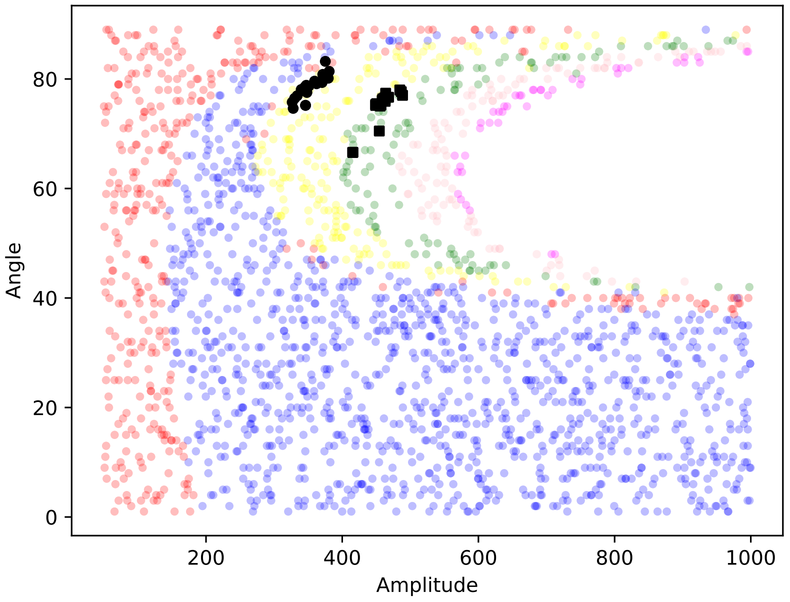


Figure 6: The result of effect region clustering visualized in action space overlaid with actions selected by the two fully trained SAC agents for the one-step (circles) and two-step (squares). The two important effect regions are the yellow (one-step height gain effect) and the green (two-step height gain effect). This illustration clearly shows that the effect regions formed by our method coincide with the optimal actions taken by a trained agent in continuous control.