Diversity for Contingency: Learning Diverse Behaviors for Efficient Adaptation and Transfer

Finn Rietz[1] and Johannes A. Stork[1]

***Abstract*— Discovering all useful solutions for a given task is crucial for transferable RL agents, to account for changes in the task or transition dynamics. This is not considered by classical RL algorithms that are only concerned with finding the optimal policy, given the current task and dynamics. We propose a simple method for discovering all possible solutions of a given task, to obtain an agent that performs well in the transfer setting and adapts quickly to changes in the task or transition dynamics. Our method iteratively learns a set of policies, while each subsequent policy is constrained to yield a solution that is unlikely under all previous policies. Unlike prior methods, our approach does not require learning additional models for novelty detection and avoids balancing task and novelty reward signals, by directly incorporating the constraint into the action selection and optimization steps.**

I. INTRODUCTION

The standard reinforcement learning (RL) approach [1] learns deterministic policies [2, 3, 4] for each task from scratch, despite the notorious sample inefficiency of deep RL algorithms. Instead, it would be preferable to learn transferable and reusable policies and to adapt them to different downstream tasks, with a fraction of data and compute needed compared to learning from scratch. A promising approach for learning transferable RL agents is multi-objective RL (MORL), where vectorized value functions can be shared for many tasks [5, 6, 7]. A key requirement for learning such transferable agents is to allow stochasticity and diversity in the learned behavior [8, 9], as opposed to learning one overly specific, deterministic policy. While MaxEnt RL [10, 11, 5] regularizes policies in an attempt to prevent them from becoming overly specific, entropy-regularized (MO) RL is not sufficient for inducing agents that learn all behaviors that solve the given tasks, as can be seen in Fig. 1. To adapt transferred agents efficiently it is important to discover *all* useful behaviors, to account for possible contingencies, e.g. parts of the middle pathway becoming blocked.

In this paper, we first review MaxEnt RL and methods that learn diverse behaviors, either unsupervised or for a given task. In Sec. III, we then propose a novel method for learning policies that discover different solutions for the given task, accounting for possible contingencies in transfer settings.

*This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

[1]Adaptive and Interpretable Learning Systems Lab, Center for Applied Autonomous Sensor Systems, Örebro University, Sweden, Correspondence: finn.rietz@oru.se

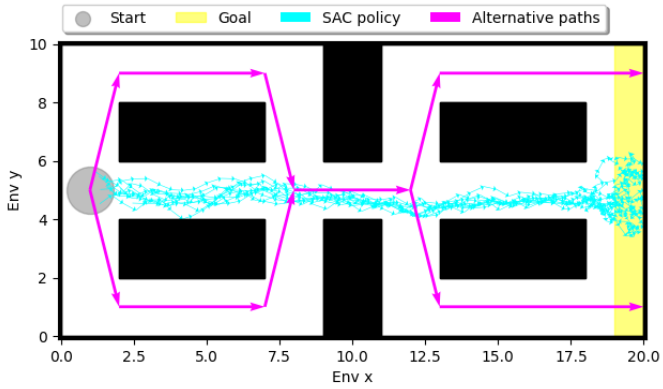

Fig. 1. Rollouts generated by a soft actor-critic agent. The behavior emits local stochasticity but does not learn the alternative paths. Discovering all possible solutions is crucial for transfer and adaptation.

II. RELATED WORK

A. Entropy regularization

A common approach to learning stochastic policies with wide and smooth maxima is MaxEnt RL [11, 10]. Maximum entropy (MaxEnt) RL augments the RL objective by adding a term proportional to the policy's entropy to the reward

$$J(\pi) = \sum_{t=1}^{\infty} \mathbb{E}_{(\mathbf{s}_t, \mathbf{a}_t)} \Big[\gamma^{t-1} r(\mathbf{s}_t, \mathbf{a}_t) + \alpha \mathcal{H}\big(\pi(\cdot \mid \mathbf{s}_t)\big) \Big], \quad (1)$$

where $\mathcal{H}(X) = \mathbb{E}[-\log p(x)]$ is Shannon's entropy, thereby punishing unnecessarily deterministic policies. The coefficient α balances the reward and the entropy terms, thereby giving some control over the stochasticity in the learned policy, however, this coefficient is usually annealed towards zero as training progresses. The primary algorithm for MaxEnt RL is soft actor-critic (SAC) [12, 13], which learns an on-policy, soft Q-function Q^{π}_{soft} for an univariate Gaussian actor model. As can be seen in Fig. 1, SAC learns one behavior (with local variations) but disregards other behaviors that reach the goal. This has two reasons. Firstly, SAC's actor model is unimodal and thus can not capture all possible modes, e.g. at the forks or intersections in the environment. While some prior works [14, 11] can learn multi-modal, entropy-regularized policies, multi-modality is not the key requirement to learning diverse behaviors. The second and more important reason why SAC disregards the other possible behaviors is that they are clearly sub-optimal, since their trajectories are longer and have higher costs compared to driving straight down the middle, from start to goal. RL

is fundamentally only concerned with finding one optimal policy that solves the task, whether alternative solutions are possible is not considered, although this is crucial for transfer RL. In the next section, we review methods that, unlike classical and MaxEnt RL, account for this and explicitly aim to learn diverse behavior alongside the optimal policy.

B. Learning diverse behaviors

Popular approaches to learning diverse behaviors originate from unsupervised *option* [15] discovery [16, 17]. One such method is DIAYN [18], which discovers distinct behaviors in an unsupervised manner and in the absence of a reward function, by maximizing the mutual information between behaviors and states [18]. Similarly, VALOR [19] discovers distinct behaviors by maximizing the mutual information between behaviors and context vectors [19], again without access to a reward function. Both of these methods subsequently use the learned behaviors as low-level options in a hierarchical RL agent [15] to solve downstream tasks efficiently.

In this paper, we instead assume access to the reward function from the beginning and wish to exploit this information during learning, to discover alternative solutions to the given task. In this setting, Zhang, Yu, and Turk [8] learn multiple distinct policies for a task reward function r_{task} by training an autoencoder $\mathbf{D} = \{\mathcal{D}_1, \dots, \mathcal{D}_n\}$ for each available policy $\pi_1, \dots, \pi_n$ on state sequences $\mathbf{S}_i = (\mathbf{s}_t, \mathbf{s}_{t+1}, \mathbf{s}_{t+K})$ of that policy and constructing a *novelty* reward function

$$r_{\text{novel}} = -\exp\big(-w \min_{\mathcal{D}\in\mathbf{D}} ||\mathcal{D}(\mathbf{S}) - \mathbf{S}||^2\big). \tag{2}$$

Zhang, Yu, and Turk [8] then update the policy using the angular bisector of the gradients on the expected novelty and task reward, to ensure that both objectives are improved. Similarly, Zhou et al. [20] learn distinct policies for a given task by constraining policy search to trajectories τ that have low log-likelihood under already learned policies. To promote diverse exploration, Zhou et al. [20] define, in addition to the extrinsic task reward r^{ext}, an intrinsic reward function r^{int} based on learned, policy-specific reward models, to boost diverse exploration:

$$\bar{J}(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\phi(\tau)\sum_{t=1}^{\infty}\gamma^{t-1} r_t^{\text{ext}} + \lambda\big(1-\phi(\tau)\big)\sum_j r_t^{\text{int}}\right], \tag{3}$$

where

$$\phi(\tau) = \prod_{j=1}^{k-1} \mathbb{I}[\text{NLL}(\tau, \pi_j) \geq \delta] \tag{4}$$

is an indicator function on negative log-likelihood of trajectories with threshold δ. While [8, 20] exploit the task reward signal for learning novel policies for the given task, these methods either require learning additional novelty detectors, have to balance multiple reward signals or rely on expensive Monte Carlo updates. In the next section, we propose a simple method for discovering alternative solutions for a given task, while avoiding these shortcomings.

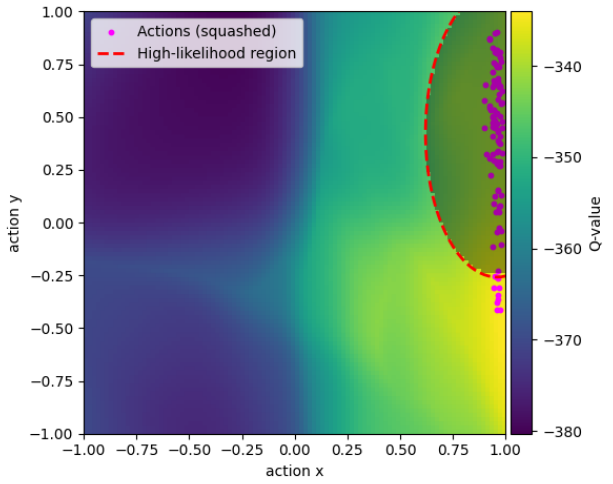

Fig. 2. Q-function, action samples, and high-likelihood region of the agent evaluated at the starting position. Novelty-constraints on policy-likelihood prevent the agent from using actions that fall into the high-likelihood region of prior policies.

III. Learning contingent policies via novelty constraints

We propose an iteratively-constrained algorithm for learning alternative policies for the given task. In each iteration, our algorithm learns a novel policy that attempts to solve the task, while its solution space is constrained to behavior that is unlikely under all previous policies for that task. Unlike [8, 20], we refrain from changing the agent's objective by introducing auxiliary novelty rewards, our agent still maximizes the expected task reward (subject to entropy-regularization), as in Eq. (1). This avoids the trade-off between the (potentially conflicting) objectives of maximizing return and behaving novel. Instead and intuitively, to learn novel behavior, the agent should, in every state, only execute actions that are unlikely under prior policies for the same task. Following this intuition, we constraint policy search in the i-th iteration (i.e. learning of the i-th policy) to a set Π^{π}_{i-1} of policies, where policies in this set only select actions that are unlikely under all prior policies:

$$\pi_i^* = \max_{\pi'} J(\pi') \mid \pi' \in \Pi^{\pi}_{i-1}. \tag{5}$$

To perform policy search as in Eq. 5, the agent needs a way to sample actions from policies in Π^{π}_{i-1}. Implementing Eq. (5) locally and state-based, action selection for policies in Π^{π}_{i-1} is constrained:

$$\begin{aligned} &\mathbf{a} \sim \pi_i(\mathbf{s}) \\ \text{subject to} \quad &\pi_j(\mathbf{s}, \mathbf{a}) \leq \varepsilon_j, \forall j\{1, \dots, i-1\}, \end{aligned} \tag{6}$$

where ε_j are thresholds specifying the maximally allowed action likelihood under policies from previous iterations $1, \dots, i-1$. Fig. 2 provides a visualization of this constraint and how it forbids actions that fall into the high-likelihood region of prior policies. For each previous policy, we define

an indicator function

$$\mathbb{I}_j^{\pi}(\mathbf{s}, \mathbf{a}) = \begin{cases} 1 & \text{if } \pi_j(\mathbf{s}, \mathbf{a}), \leq \varepsilon_j \\ 0 & \text{otherwise,} \end{cases} \tag{7}$$

that can be used to check whether an action $\mathbf{a}$ in state $\mathbf{s}$ satisfies the *novelty constraint* in Eq. (6). With the novelty constraint indicator functions we can project any policy into Π_{i-1}^{π}:

$$\hat{\pi}_i(\mathbf{a} \mid \mathbf{s}) \propto \pi_i(\mathbf{a} \mid \mathbf{s}) \prod_{j=1}^{i-1} \mathbb{I}_j^{\pi}(\mathbf{s}, \mathbf{a}). \tag{8}$$

Projecting policies into Π_{i-1}^{π} and sampling from $\hat{\pi}_i$ via rejection sampling is thus straightforward, however, we still require an algorithm for learning policies π_i whose projections $\hat{\pi}_i$ perform well. We propose such a learning algorithm in the next section.

A. Iterative novelty-constrained SAC

In the i-th iteration of the novelty-constrained setting, the agent's true (novelty-constrained) policy is $\hat{\pi}_i$, to which we only have access via rejection sampling. Thus, a learning algorithm for π_i does not learn the agent's true policy but a proposal distribution for $\hat{\pi}_i$. To account for this, we propose an iterative and novelty-constrained version of SAC [12]. Learning a critic for $\hat{\pi}_i$, the novelty-constrained policy, is straightforward by ensuring that the expectation of future actions in the TD-backup matches the (novelty-constrained) actor:

$$J_Q(\theta_i) = \mathbb{E}_{\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1} \sim \mathcal{D}} \left[\frac{1}{2} \big(Q_{\theta_i}(\mathbf{s}_t, \mathbf{a}_t) - \hat{Q}_{\theta_i}(\mathbf{s}_t, \mathbf{a}_t) \big) \right], \tag{9}$$

with

$$\hat{Q}_{\theta_i}(\mathbf{s}_t, \mathbf{a}_t) = r(\mathbf{s}_t, \mathbf{a}_t) + \gamma \, \mathbb{E}_{\mathbf{a}_{t+1} \sim \hat{\pi}_i} \big[Q_{\bar{\theta}_i}(\mathbf{s}_{t+1}, \mathbf{a}_{t+1}) - \log(\pi_i(\mathbf{a}_{t+1} \mid \mathbf{s}_{t+1})) \big], \tag{10}$$

where $\bar{\theta}_i$ refers to the target network parameter for $\hat{\pi}_i$'s critic. Similarly, the actor update must reflect the novelty constraint and rejection sampling step as well. A key property of SAC is that it updates the actor by minimizing the KL divergence between the actor and the critic:

$$\begin{aligned} J_{\pi}(\phi) &= \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}} \left[\mathrm{D}_{\mathrm{KL}} \left(\pi_{\phi}(\cdot \mid \mathbf{s}_t) \;\middle\|\; \frac{\exp(Q_{\theta_i}(\mathbf{s}, \cdot))}{Z_{\theta_i}(\mathbf{s}_t)} \right) \right] \\ &= \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}, \mathbf{a}_t \sim \pi_{\phi}} \big[\log \pi_{\phi}(\mathbf{a}_t \mid \mathbf{s}_t) - Q_{\theta_i}(\mathbf{s}_t, \mathbf{a}_t) \big]. \end{aligned} \tag{11}$$

In our case, we can still backpropagate through the critic, however, only for action samples that have low likelihood under previous policies and satisfy the novelty constraint (i.e. for which $\prod_{j=1}^{i-1} \mathbb{I}_j^{\pi}(\mathbf{s}, \mathbf{a}) = 1$). Action samples that violate the novelty constraint, i.e. are likely under previous policies, should follow a different gradient because they would be discarded by the rejection sampling step. Since these actions are never executed, the critic never observes a reward signal for those actions and hallucinates unreliable value estimates whose gradients are not suited for learning the novelty-constrained actor. Thus, to account for the rejection sampling step and to encourage learning an actor that respects the novelty constraint in Eq. (6), for actions that violate any of the $i-1$ constraints, we instead use the gradient of the KL divergence between the current policy π_i and the policies whose constraints are violated. This leads to the following actor update for the proposed, novelty-constrained SAC:

$$\begin{aligned} J_{\pi}(\phi_i) = \mathbb{E}_{\mathbf{s}_t \sim \mathcal{D}, \mathbf{a}_t \sim \pi_{\phi_i}} \Bigg[& \prod_{j=1}^{i-1} \mathbb{I}_j^{\pi}(\mathbf{s}, \mathbf{a}) \overbrace{\log \pi_{\phi_i}(\mathbf{a}_t \mid \mathbf{s}_t) - Q_{\theta_i}(\mathbf{s}_t, \mathbf{a}_t)}^{\mathrm{D}_{\mathrm{KL}}(\pi_i || Q)} + \\ & \left(1 - \prod_{j=1}^{i-1} \mathbb{I}_j^{\pi}(\mathbf{s}, \mathbf{a}) \right) \underbrace{\log \pi_{\phi_i}(\mathbf{a}_t \mid \mathbf{s}_t) - \log \pi_{\phi_j}(\mathbf{a}_t \mid \mathbf{s}_t)}_{\mathrm{D}_{\mathrm{KL}}(\pi_i || \pi_j)} \Bigg]. \end{aligned} \tag{12}$$

Thus, our iterative algorithm for learning contingent behaviors operates as follows. The first policy π_1 is learned unconstrained, using normal SAC [5], thus π_1 is the optimal soft policy for the given task. Once π_1 has converged, our algorithm proceeds with the learning of π_2, which is novelty-constrained w.r.t π_1, meaning it has to solve the given task as best as possible while respecting the constraint in Eq. (6). π_2 is learned using the critic update in Eq. (9), the actor update in Eq. (12) and relies on rejection sampling to generate actions from $\hat{\pi}_2$. Once π_2 has converged, π_3 can be learned, being novelty constrained w.r.t π_1 and π_2, and so on. We refer to the additional policies $\pi_2, \ldots, \pi_i$ as "contingency policies" since they can be used in situations where the optimal policy fails when transferred to a new task. In the next section, we show how these additional policies can be used to recover from unforeseen events in the transfer task.

IV. Recovering from unforeseen events using contingency policies

When we transfer a pre-trained agent to a new task, the agent can be exposed to situations that require it to deviate from its behavior learned during pre-training. For example, one such event might be when the middle path in Fig. 1 becomes blocked. Our proposed method accounts for this by learning additional policies during pre-training, to be used in and recover from such situations. To know when we should use one of the contingency policies instead of the optimal policy, we require a method for detecting contingencies, where the optimal pre-trained policy behaves sub-optimally. In the scope of this workshop paper, we simply rely on $\Delta \mathbf{s}$, i.e.changes in the state variable, to detect such events and leave a more sophisticated method as future work. Given we detect a contingency, we propose the following algorithm to recover from such events: First, the agent backtracks for k steps. Then, it runs the first contingency policy for m steps, followed by rolling out the optimal policy. If the agent does not finish the task this way, it repeats this process using all available contingency policies and further backtracking steps. This uninformed process does not require additional knowledge, e.g. a model of environment transition dynamics, and works well in practice, as seen in Figure 3. This is in contrast to a baseline comparison in Fig. 4, where the agent executes

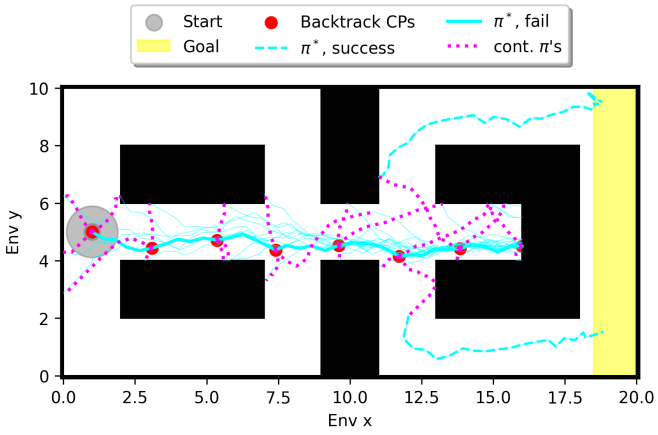

Fig. 3. Rollouts generated by our backtracking algorithm. The algorithm attempts to recover by rolling out the available contingency policies, followed by rollouts of the optimal policy. If unsuccessful, the agent backtracks to the next checkpoints and executes the contingency policies and optimal policy again. This repeats until the task is finished successfully or all checkpoints are exhausted.

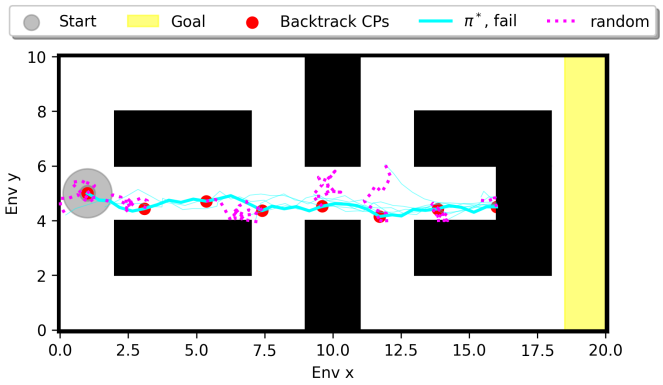

Fig. 4. Rollouts generated by our backtracking algorithm but with random behavior instead of contingent recovery policies.

random actions to recover from the contingencies, instead of the recovery policies learned with our proposed method. There are a number of points that should be addressed in future work. Instead of treating k and m as hyperparameters, it would be preferable to automatically identify states to backtrack to and automatically decide for how long and which contingency policy to execute. For the result in Fig. 3, we manually selected values that were adequate for our simple testing environment, which is not practical for more sophisticated problems. We leave these points, as well as more thorough experimentation and baseline comparisons, as important future work.

References

[1] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[2] John Schulman et al. "Proximal policy optimization algorithms". In: *arXiv preprint arXiv:1707.06347* (2017).

[3] John Schulman et al. "Trust region policy optimization". In: *International conference on machine learning*. PMLR. 2015, pp. 1889–1897.

[4] Timothy P Lillicrap et al. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[5] Tuomas Haarnoja et al. "Composable deep reinforcement learning for robotic manipulation". In: *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2018, pp. 6244–6251.

[6] Jonathan Hunt et al. "Composing entropic policies using divergence correction". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 2911–2920.

[7] André Barreto et al. "Successor Features for Transfer in Reinforcement Learning". In: *Proceedings of the 31st International Conference on Neural Information Processing Systems*. NIPS'17. 2017, pp. 4058–4068.

[8] Yunbo Zhang, Wenhao Yu, and Greg Turk. "Learning Novel Policies For Tasks". In: *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*. Vol. 97. Proceedings of Machine Learning Research. PMLR, 2019, pp. 7483–7492.

[9] Karol Hausman et al. "Learning an embedding space for transferable robot skills". In: *International Conference on Learning Representations*. 2018.

[10] Brian D Ziebart et al. "Maximum entropy inverse reinforcement learning." In: *Aaai*. Vol. 8. Chicago, IL, USA. 2008, pp. 1433–1438.

[11] Tuomas Haarnoja et al. "Reinforcement learning with deep energy-based policies". In: *International conference on machine learning*. PMLR. 2017, pp. 1352–1361.

[12] Tuomas Haarnoja et al. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[13] Tuomas Haarnoja et al. "Soft actor-critic algorithms and applications". In: *arXiv preprint arXiv:1812.05905* (2018).

[14] Yunhao Tang and Shipra Agrawal. "Implicit policy for reinforcement learning". In: *arXiv preprint arXiv:1806.06798* (2018).

[15] Richard S Sutton, Doina Precup, and Satinder Singh. "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning". In: *Artificial intelligence* 112.1-2 (1999), pp. 181–211.

[16] Pierre-Luc Bacon, Jean Harb, and Doina Precup. "The option-critic architecture". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 31. 1. 2017.

[17] Roy Fox et al. "Multi-level discovery of deep options". In: *arXiv preprint arXiv:1703.08294* (2017).

[18] Benjamin Eysenbach et al. "Diversity is All You Need: Learning Skills without a Reward Function". In: *7th International Conference on Learning Repre-*

sentations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net, 2019.

[19] Joshua Achiam et al. "Variational Option Discovery Algorithms". In: *CoRR* abs/1807.10299 (2018). arXiv: 1807.10299.

[20] Zihan Zhou et al. "Continuously Discovering Novel Strategies via Reward-Switching Policy Optimization". In: *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net, 2022.