Transfer Reinforcement Learning across Homotopy Classes

Zhangjie Cao*[1] and Minae Kwon*[2] and Dorsa Sadigh[3]

***Abstract*—The ability for robots to transfer their learned knowledge to new tasks—where data is scarce—is a fundamental challenge for successful robot learning. While fine-tuning has been well-studied as a simple but effective transfer approach in the context of supervised learning, it is not as well-explored in the context of reinforcement learning. In this work, we study the problem of fine-tuning in transfer reinforcement learning when tasks are parameterized by their reward functions, which are known beforehand. We conjecture that fine-tuning drastically underperforms when source and target trajectories are part of different *homotopy classes*. We demonstrate that fine-tuning policy parameters across homotopy classes compared to fine-tuning within a homotopy class requires more interaction with the environment, and in certain cases is impossible. We propose a novel fine-tuning algorithm, Ease-In-Ease-Out fine-tuning, that consists of a relaxing stage and a curriculum learning stage to enable transfer learning across homotopy classes. Finally, we evaluate our approach on several robotics-inspired simulated environments and empirically verify that the Ease-In-Ease-Out fine-tuning method can successfully fine-tune in a sample-efficient way compared to existing baselines.**

***Index Terms*—Learning (artificial intelligence), Intelligent systems, Learning systems**

I. Introduction

One of the goals of transfer learning is to efficiently learn policies in tasks where sample collection is cheap and then transfer the learned knowledge to tasks where sample collection is expensive. Recent deep reinforcement learning (Deep RL) algorithms require an extensive amount of data, which can be difficult, dangerous, or even impossible to obtain [29], [37], [47], [30]. Practical concerns regarding sample inefficiency make transfer learning a timely problem to solve, especially in the context of RL for robotics. Robots should be able to efficiently transfer knowledge from related tasks to new ones. For instance, consider an assistive robot that learns to feed a patient with a neck problem. The robot could not learn a sophisticated feeding policy when directly trained with a disabled patient in-the-loop, due to the limited number of interactions with the patient. Instead, the robot can learn how to feed abled-bodies, where it is easier to obtain data, and transfer the learned knowledge to the setting with the disabled patient using only a few samples.

We study transfer in the reinforcement learning setting where different tasks are parameterized by their reward function. While this problem and its similar variants have been studied using approaches like meta-RL [14], [45], [8], [16], [20], multitask learning [34], [42], and successor features [1], fine-tuning as an approach for transfer learning in RL is still not well-explored. Fine-tuning is an important method to study for two reasons. First, it is a widely-used transfer learning approach that is very well-studied in supervised learning [28], [21], [46], but the limits of fine-tuning have been less studied in RL. Second, compared to peer approaches, fine-tuning does not require strong assumptions about the target domain, making it a general and easily applicable approach. *Our goal is to broaden our understanding of transfer in RL by exploring when fine-tuning works, when it doesn't, and how we can overcome its challenges.* Concretely, we consider fine-tuning to be more efficient when it requires less interactive steps with the target environment.

In this paper, we find that fine-tuning does not always work as expected when transferring between rewards whose corresponding trajectories belong to different homotopy classes. A homotopy class is traditionally defined as a class of trajectories that can be continuously deformed to one another without colliding with any barriers [4], see Fig. 1 (a). In this work, we generalize the notion of barriers to include any set of states that incur a large negative reward. These states lead to phase transitions (discontinuities) in the reward function. We assume that we know these barriers (and therefore homotopy classes) beforehand, which is equivalent to assuming knowledge of the reward functions. Knowing the reward function a-priori is a commonly made assumption in many robotics tasks, such as knowing goals [24], [23], [33] or having domain knowledge of unsafe states beforehand [18], [44]. Also, reinforcement learning algorithms naturally assume that the reward function is available [40]. Generalizing the notion of barriers allows us to go beyond robotics tasks classically associated with homotopy classes, e.g., navigation around barriers, to include tasks like assistive feeding. *Our key insight is that fine-tuning continuously changes policy parameters and that leads to continuously deforming trajectories.* Hence, fine-tuning across homotopy classes will induce trajectories that intersect with barriers. This will introduce a high loss and gradients that point back to the source policy parameters. So it is difficult to fine-tune the policy parameters across homotopy classes. To address this challenge, we propose a novel Ease-In-Ease-Out fine-tuning approach consisting of two stages: a Relaxing

Manuscript received October 15, 2020; revised January 6, 2021; accepted January 11, 2021.

This paper was recommended for publication by Editor Dana Kulic upon evaluation of the Associate Editor and Reviewers' comments.

This work was supported by NSF Award Number 2006388 and the DARPA Hicon-Learn project.

[1]caozj@cs.stanford.edu, [2]minae@cs.stanford.edu, and [3]dorsa@stanford.edu. The authors are with the Department of Computer Science, Stanford University, Stanford, CA 94305, USA.

$*$ means equal contribution.

Digital Object Identifier (DOI): see top of this page.

Stage and a Curriculum Learning Stage. In the Relaxing Stage, we relax the barrier constraint by removing it. Then, in the Curriculum Learning Stage, we develop a curriculum starting from the relaxed reward to the target reward that gradually adds the barrier constraint back.

The contributions of the paper are summarized as follows:

- We introduce the idea of using homotopy classes as a way of characterizing the difficulty of fine-tuning in reinforcement learning. We extend the definition of homotopy classes to general cases and demonstrate that fine-tuning across homotopy classes requires more interaction steps with the environment than fine-tuning within the same homotopy class.
- We propose a novel Ease-In-Ease-Out fine-tuning approach that fine-tunes across homotopy classes, and consists of a relaxing and a curriculum learning stage.
- We evaluate Ease-In-Ease-Out fine-tuning on a variety of robotics-inspired environments and show that our approach can learn successful target policies with less interaction steps than other fine-tuning approaches.

II. Related Work

Fine-tuning. Fine-tuning is well-studied in the space of supervised learning [25], [17], [32], [26], [19], [12], [36]. Approaches such as L^2-SP penalize the Euclidean distance of source and fine-tuned weights [27]. Batch Spectral Shrinkage penalizes small singular values of model features so that untransferable spectral components are repressed [7]. Progressive Neural Networks (PNN) transfer prior knowledge by merging the source feature into the target feature at the same layer [35]. These works achieve state-of-the-art fine-tuning performance in supervised learning; however, directly applying fine-tuning methods to transfer RL does not necessarily lead to successful results as supervised learning and reinforcement learning differ in many factors such as access to labeled data or the loss function optimized by each paradigm [2]. We compare our approach with these fine-tuning methods for transfer RL.

In fine-tuning for robotics, a robot usually pre-trains its policy on a general source task, where there is more data available, and then fine-tunes to a specific target task. Recent work in vision-based manipulation shows that fine-tuning for off-policy RL algorithms can successfully adapt to variations in state and dynamics when starting from a general grasping policy [22]. As another example, RoboNet trains models on different robot platforms and fine-tunes them to unseen tasks and robots [10]. A key difference is that our work proposes a systematic approach using homotopy classes for discovering when fine-tuning can succeed or fail. This is very relevant to existing literature in this domain, as our approach can explain *why* a general policy, e.g., a general grasping policy, *can* or *cannot* easily be fine-tuned to more specific settings.

Transfer Reinforcement Learning. There are several lines of work for transfer RL including successor features, meta-RL and multitask learning. We refer the readers to [41] for a comprehensive survey. We compare these works to our approach below.

Successor Features. Barreto et al., address the same reward transfer problem as ours by learning a universal policy across tasks based on successor features [1]. However, this work makes a number of assumptions about the structure of the reward function and requires that the rewards between source and target tasks be close to each other, while our work has no such constraints.

Meta-RL. Meta learning provides a generalizable model from multiple (meta-training) tasks to quickly adapt to new (meta-test) tasks. There are various Meta RL methods including RNN-based [14], [45], [8], gradient-based [16], [20], [8], or meta-critic approaches [39]. The gradient-based approach is the most related to our work, which finds policy parameters (roughly akin to finding a source task) that enable fast adaptation via fine-tuning. Note that all meta-RL approaches assume that agents have access to environments or data of meta-training tasks, which is not guaranteed in our setting. Here our focus is to discover when fine-tuning is generally challenging based on homotopy classes. In our experiments we compare our algorithm to core fine-tuning approaches rather than techniques that leverage ideas from fine-tuning or build upon them.

Multitask learning. Other works transfer knowledge by simultaneously learning multiple tasks or goals [34]. In these works, transfer is enabled by learning shared representations of tasks and goals [11], [42], [34], [31]. In our work, we consider the setting where tasks are learned sequentially.

Regularization. Cobbe et al's work [9] proposes a metric to quantify the generalization ability of RL algorithms and compare the effects of different regularization techniques on generalization. The paper compares the effects of deeper networks, batch normalization, dropout, L2 Regularization, data augmentation and stochasticity (ϵ-greedy action selection and entropy bonus). The proposed techniques are designed for general purpose transfer reinforcement learning but are not specially designed for transfer reinforcement learning across homotopy classes. We compare our approach against using deeper networks, dropout, and entropy bonuses in our Navigation and Lunar Lander experiments and show that these techniques alone are not sufficient to transfer across homotopy classes (see supplementary materials).

III. Fine-tuning across Homotopy Classes

In transfer reinforcement learning, our goal is to fine-tune from a source task to a target task. We formalize a task using a Markov Decision Process $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, p, \mathcal{R}, \rho_0, \gamma \rangle$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ is the transition probability, ρ_0 is the initial state distribution, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and γ is the discount factor. We denote $\mathcal{M}_s$ as the source task and $\mathcal{M}_t$ as the target task. We assume that $\mathcal{M}_s$ and $\mathcal{M}_t$ only differ on reward function, i.e., $\mathcal{R}_s \neq \mathcal{R}_t$. These different reward functions across the source and target task can for instance capture different preferences or constraints of the agent. A stochastic policy $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ defines a probability distribution over the action in a given state. The goal of RL is to learn an optimal policy π^*, which maximizes the expected discounted return $\eta_\pi = \mathbb{E}_{\xi \sim \pi}[G_\tau(\xi)] = \mathbb{E}_{s_0 \sim \rho_0, \pi}\left[\sum_{\tau=0}^{\infty} \gamma^\tau \mathcal{R}(s_\tau, a_\tau)\right]$. We define a trajectory to be the sequence of states the agent has visited over time $\xi = \{s_0, s_1, \dots\}$, and denote ξ^* to be a

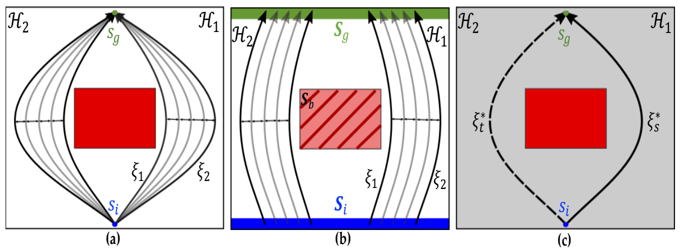

Fig. 1: (a) Homotopy classes. ξ_1 and ξ_2 are part of the same homotopy class because they can be continuously deformed into each other. **(b)** Generalized homotopy classes with expanded definitions of start, end, and barrier states. **(c)** Fine-tuning problem from *left* side to the *right* side. The goal is to find π_t^* that produces ξ_t^*.

trajectory produced by the optimal policy, π^*. We assume that the optimal policy for the source environment π_s^* is available or can be easily learned. Our goal is then to leverage knowledge from π_s^* to learn the optimal policy π_t^* for task $\mathcal{M}_t$. We aim to learn π_t^* with substantially fewer training samples than other comparable fine-tuning approaches.

A. Homotopy Classes

Homotopy classes are formally defined by homotopic trajectories in navigation scenarios in [4]:

Definition III.1. Homotopic Trajectories and Homotopy Class. Two trajectories ξ_1, ξ_2 connecting the same initial and end points s_i, s_g are homotopic if and only if one can be continuously deformed without intersecting with any barriers. Homotopic trajectories are clustered into a homotopy class. [1]

Fig. 1 (a) illustrates a navigation scenario with two homotopy classes $\mathcal{H}_1$ and $\mathcal{H}_2$ separated by a red barrier. ξ_1 and ξ_2 can be continuously deformed into each other without intersecting the barrier, and hence are in the same homotopy class.

Generalization. The original definition of homotopy classes is limited to navigation scenarios with deterministic trajectories and the same start and end states. We generalize this definition to encompass a wider range of tasks in three ways.

Firstly, we account for tasks where there could be more than one feasible initial and end state. We generalize the initial and end points s_i, s_g to a set of states $\mathbf{S}_i$ and $\mathbf{S}_g$, where $\mathbf{S}_i$ contains all the possible starting states and $\mathbf{S}_g$ contains all possible ending states as shown in Fig. 1 (b).

Secondly, we generalize the notion of a barrier to be a set of states that are penalized with large negative rewards $\mathbf{S}_b = \{s | \mathcal{R}(s,a) \leftarrow \mathcal{R}'(s,a) - M\}$, where M is a large positive number and $\mathcal{R}'(s,a)$ is the reward without any barriers. Large negative rewards correspond to any negative phase transitions or discrete jumps in the reward function. Importantly, the generalized 'barrier' allows us to define homotopy classes in tasks without physical barriers that penalize states with large negative rewards (see our Assistive Feeding experiment). Although source and target tasks differ in reward functions, they share the same barrier states.

[1]Even though the presence of a single obstacle introduces infinitely many homotopy classes, in most applications we can work with a finite number of them, which can be formalized by the concept of Z_2-homology [6]. For algorithms that compute these homology classes see [6].

Thirdly, we need to generalize the notion of continuously deforming *trajectories* to *trajectory distributions* when considering stochastic policies. We appeal a distribution distance metric, Wasserstein-∞ (W_∞) metric, that penalizes *jumps* (discontinuities) between trajectory distributions induced by stochastic policies. We can now define our generalized notion of homotopic trajectories.

Definition III.2. General Homotopic Trajectories. Two trajectories ξ_1, ξ_2 with distributions μ_1 and μ_2 and with the initial states $s_i \in \mathbf{S}_i$ and the final states $s_g \in \mathbf{S}_g$ are homotopic if and only if one can be continuously deformed into the other in the W_∞ metric without receiving large negative rewards. Definitions for the W_∞ metric and W_∞-continuity are in Section I of the supplementary materials.

General homotopic trajectories are depicted in Fig. 1 (b). The generalized definition of a homotopy class is the set of general homotopic trajectories. Note that using the W_∞ metric is crucial here. Homotopic equivalence of stochastic policies according to other distances like total variation, KL-divergence, or even W_1 is usually trivial because distributions that even have a tiny mass on all deterministic homotopy classes become homotopically equivalent. On the other hand, in the W_∞ metric, the distance between distributions that tweak the percentages, even by a small amount, would be at least the minimum distance between trajectories in different deterministic homotopy classes, which is a constant. So to go from one distribution over trajectories to another one with different percentages, one has to make a *jump* according to the W_∞ metric.

B. Challenges of Fine-tuning across Homotopy Classes

Running Example. We explain a key optimization issue caused by barriers when fine-tuning across homotopy classes. We illustrate this problem in Fig. 1 (b). An agent must learn to navigate to its goal $s_g \in \mathbf{S}_g$ without colliding with the barrier. Assuming that the agent only knows how to reach s_g by swerving right, denoted by ξ_s^*, we want to learn how to reach s_g by swerving left (i.e., find π_t^*).

We show how the barrier prevents fine-tuning from source to target in Fig. 2. This figure depicts the loss landscape for the target task with and without barriers. All policies are parameterized by a single parameter $\theta \in \mathbb{R}^2$ and optimized with the vanilla policy gradient algorithm [3]. Warmer regions indicate higher losses in the target task whereas cooler regions indicate lower losses.

Policies that collide with barriers cause large losses shown by the hump in Fig. 2 (b). Gradients point away from this large loss region, so it is difficult to cross the hump without a sufficiently large step size. In contrast, in Fig. 2 (a), the loss landscape without the barrier is smooth, so fine-tuning is easy to converge. Details on the landscape plots are in Section IV of the supplementary materials.

We now formally investigate how discontinuities in trajectory space caused by barriers affect fine-tuning of model-free RL algorithms. We let the model parameterized by θ to induce a policy π_θ, and define the loss for the model to be $\ell(\theta)$. We assume that $\ell(\theta)$ is high when the expected return $\eta_{\pi_\theta} =$

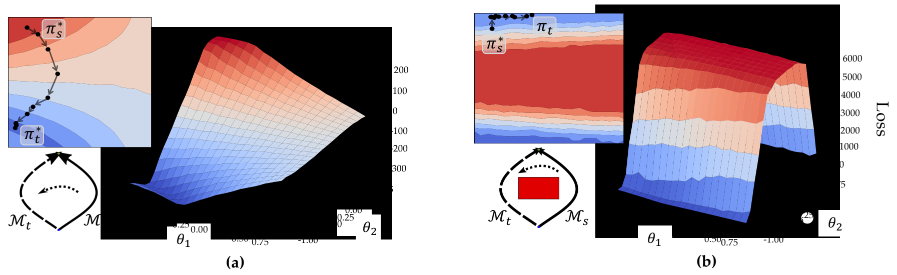

Fig. 2: **(a)** Loss landscape of our running example without a barrier. The top-down pictures illustrate the gradient steps taken when fine-tuning from source to target tasks. **(b)** Loss landscape with a barrier. Barriers create gradients away from it that make it difficult to fine-tune from source to target tasks.

$\mathbb{E}_{\xi\sim\pi_\theta}[G(\xi)]$ is low. This assumption is satisfied in common model-free RL algorithms such as vanilla policy gradient. We optimize our policy using gradient descent with step size α: $\theta_{k+1} = \theta_k - \alpha\nabla_\theta \ell(\theta)|_{\theta_k}$. We can now define what it means to fine-tune from one task to another.

Let θ_s^* be the optimal set of parameters that minimizes the cost function on the source task. Using $\ell^t(\theta)$ as the loss for target reward, fine-tuning from $\mathcal{M}_s$ to $\mathcal{M}_t$ for n gradient steps is defined as: $\theta_1 \leftarrow \theta_s^*$ and $\theta_{k+1} = \theta_k - \alpha\nabla_\theta \ell^t(\theta)|_{\theta_k}$ for $k = 1, \ldots, n$.

We consider a policy to have successfully fine-tuned to $\mathcal{M}_t$ if the received expected return is less than ϵ away from the expected reward of the optimal target policy π_t^* for some small ϵ, i.e., $|\eta^t_{\pi_\theta} - \eta^t_{\pi_t^*}| < \epsilon$.

We now theoretically analyze why it is difficult to fine-tune across homotopy classes. Due to the space limit, we only include our main theorem and remark in the paper. We refer readers to the supplementary materials for the proofs.

Definition III.3. **W_∞-continuity of policy.** A policy π_θ parameterized by θ is W_∞-continuous if the mapping $(\theta) \mapsto \pi_\theta(s,a)$, which maps a vector of parameters in a metric space to a distribution over state-actions is continuous in W_∞ metric.

Definition III.4. **W_∞-continuity of transition probability function.** An MDP $\mathcal{M}$ with transition probability function p is called W_∞-continuous if the mapping $(s,a) \mapsto p(s,a,\cdot)$ which maps a state-action pair in a metric space to a distribution over states is continuous in W_∞ metric.

Theorem 1. *Assume that π_θ is a parametrized policy for an MDP $\mathcal{M}$. If both π_θ and $\mathcal{M}$ are W_∞-continuous, then a continuous change of policy parameters θ results in a continuous deformation of the induced random trajectory in the W_∞ metric. However, continuous deformations of the trajectories do not ensure continuous changes of their corresponding policy parameters.*

Note that the theorem also applies to deterministic policies. For deterministic policies W_∞-continuity is the same as the classical notion of continuity. Theorem 1 bridges the idea of changes in policy parameters with trajectory deformation. To use this theorem, we need assumptions on the learning rate α and bounds on the gradients. Specifically for any θ_1 and θ_2 induced by policies in two different homotopy classes, we need to assume: $\|\theta_1 - \theta_2\| > \alpha \max(\nabla_\theta \ell^t(\theta)|_{\theta_1}, \nabla_\theta \ell^t(\theta)|_{\theta_2})$. With such small enough learning rate α, fine-tuning will always induce trajectories that visit barrier states, $\mathbf{S}_b$.

Remark 2. *Intuitively, the conclusion we should reach from Theorem 1 is that fine-tuning model parameters across homotopy classes is more difficult or even infeasible in terms of number of interaction steps in the environment compared to fine-tuning within the same homotopy class; this is under the assumptions that the transition probability function and policy of $\mathcal{M}$ are W_∞-continuous, learning rate is sufficiently small, and gradients are bounded* [2].

IV. Ease-In-Ease-Out fine-tuning Approach

Our insight is that even though there are states with large negative rewards that make fine-tuning difficult across homotopy classes, there is still useful information that can be transferred across homotopy classes. Specifically, we first *ease in* or *relax* the problem by removing the negative reward associated with barriers, which enables the agent to focus on fine-tuning towards target reward without worrying about large negative rewards. We then *ease out* by gradually reintroducing the negative reward via a *curriculum*. We assume the environment is alterable in order to remove and re-introduce barrier states. In most cases, this requires access to a simulator, which is a common assumption in many robotics applications [13], [15], [5], [38]. We assume that during the relaxing stage as well as each subsequent curriculum stage, we are able to converge to an approximately optimal policy for that stage using reinforcement learning.

Ease In: Relaxing Stage. In the relaxing stage, we remove the barrier penalty in the reward function, i.e., $\forall s \in \mathbf{S}_b, \mathcal{R}_t^{\text{relax}}(s,a) = \mathcal{R}'(s,a)$. We denote the target MDP with relaxed reward function as $\mathcal{M}_t^{\text{relax}}$. Note that we do not physically remove the barriers, so the transition function does not change. We start from π_s^* and train the policy in $\mathcal{M}_t^{\text{relax}}$ to obtain π^*_{relax}. The relaxation removes large losses incurred by the barriers, making fine-tuning much easier than naïve fine-tuning.

Ease Out: Curriculum Learning Stage. The relaxing stage finds an optimal policy π^*_{relax} for $\mathcal{M}_t^{\text{relax}}$. We now need to learn the optimal policy for original target MDP $\mathcal{M}_t$ that actually penalizes barrier states with a large penalty $-M$. We develop two curricula to gradually introduce this penalty.

(1) Reward Weight (general case). We design a general curriculum that can be used for any environments by gradually increasing the penalty from 0 to M using a series of values

[2]Modern optimizers and large step sizes can help evade local minima but risk making training unstable when step sizes are too large.

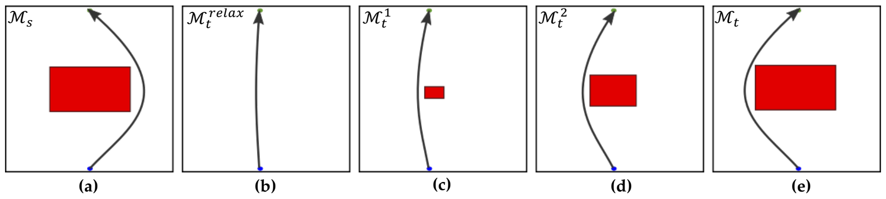

Fig. 3: Ease-In-Ease-Out fine-tuning approach for the single barrier set case. The red represents the negative reward associated with the barrier. **(a)** Source task. **(b)** Relaxing stage. The resulting policy produces trajectories that lean toward the left but remain fairly centered. **(c)-(e)** Curriculum learning stage with $K = 3$. We introduce larger subsets of the barrier states set $\mathbf{S}_b$ and fine-tune. This results in trajectories that are slowly pushed towards the left.

$\alpha_1, \ldots, \alpha_K$ satisfying $0 < \alpha_1 < \alpha_2 < \cdots < \alpha_K = 1$. We redefine our reward function to include intermediary values:

$$\mathcal{R}^{\text{cur}}(s, a; \alpha_k) = \begin{cases} \mathcal{R}'(s, a) - \alpha_k M & s \in \mathbf{S}_b \\ \mathcal{R}'(s, a) & s \notin \mathbf{S}_b \end{cases} \tag{1}$$

This allows us to define a sequence of corresponding tasks $\mathcal{M}_t^0, \ldots, \mathcal{M}_t^K$ where $\mathcal{M}_t^0 \equiv \mathcal{M}_t^{\text{relax}}$ and $\mathcal{M}_t^K \equiv \mathcal{M}_t$. For each new task $\mathcal{M}_t^k$, we initialize the policy with the previous task's optimal policy $\pi^*_{\theta_{k-1}}$ and train it using the reward $\mathcal{R}^{\text{cur}}(s, a; \alpha_k)$. The detailed algorithm is shown in Algorithm 1 in Section III of the supplementary materials.

(2) Barrier Set Size. When there is only a single barrier set $\mathbf{S}_b$ (i.e., $\mathbf{S}_b$ is connected), we can also build a curriculum around the set itself. Here, we keep the $-M$ penalty but gradually increase the set of states that incur this penalty. We can guarantee that our algorithm always converges as we discuss in our analysis section below.

To build a curriculum, we can choose any state $s \in \mathbf{S}_b$ as our initial set and gradually inflate this set to $\mathbf{S}_b$ by connecting more and more states together [3]. For example, we can connect new states that are within some radius of the current set. This allows us to define a series of connected sets $\mathbf{S}_{b_1}, \ldots, \mathbf{S}_{b_K}$ satisfying $\emptyset \subset \mathbf{S}_{b_1} \subset \mathbf{S}_{b_2} \subset \cdots \subset \mathbf{S}_{b_K} = \mathbf{S}_b$. We can then similarly redefine our reward function and parameterize it by including intermediary barrier sets $\mathbf{S}_{b_k}$

$$\mathcal{R}^{\text{cur}}(s, a; \mathbf{S}_{b_k}) = \begin{cases} \mathcal{R}'(s, a) - M & s \in \mathbf{S}_{b_k} \\ \mathcal{R}'(s, a) & s \notin \mathbf{S}_{b_k} \end{cases} \tag{2}$$

Note that the sets $\mathbf{S}_{b_k}$ only change the reward associated with the states, not the dynamics.

Curriculum learning by evolving barrier set size is more interpretable and controllable than the general reward weight approach since for each task $\mathcal{M}_t^k$, an agent learns a policy that avoids a subset of states, $\mathbf{S}_{b_k}$. In the general reward weight approach, it is unclear which states the resulting policy will never visit. A shortcoming of the barrier set size approach is that the convergence guarantee is limited to single barriers because if we have multiple barriers, we may not find a initial set $\mathbf{S}_{b_1}$ as described in Lemma 4. The algorithm for the barrier set approach follows the same structure as Algorithm 1.

Analysis. For both curriculum learning by reward weight and barrier set size, if the agent can successfully find an optimal policy at every intermediary task, then we can find π_t^* for $\mathcal{M}_t$. For the reward weight approach, we cannot prove that at every stage k, the optimal policy for $\mathcal{M}_t^k$ is guaranteed to be obtained, but we can still have the following proposition:

[3]A connected path is defined differently for continuous and discrete state spaces. For example, in continuous state spaces, a connected path means a continuous path.

Proposition 3. *For curriculum learning by reward weight, in every stage, the learned policy achieves a higher reward than the initialized policy evaluated on the final target task.*

Though the reward weight approach is not guaranteed to achieve the optimal policy in every curriculum step, the policy improves with respect to the final target reward. Each curriculum step is much easier than the original direct fine-tuning problem, which increases the possibility for successful fine-tuning. For the barrier set size approach, we prove that in every stage, the optimal policy for each stage is achievable. To learn an optimal policy in each stage, finding $\mathbf{S}_{b_1}$ is key:

Lemma 4. *There exists $\mathbf{S}_{b_1}$ that divides the trajectories of π_s^* and π_{relax}^* into two homotopy classes.*

We propose an approach for finding $\mathbf{S}_{b_1}$ in Algorithm 2 in Section III of the supplementary materials.

Proposition 5. *A curriculum starting with $\mathbf{S}_{b_1}$ as described in Lemma 4 and inflating to $\mathbf{S}_b$ with sufficiently small changes in each step, i.e., small enough for reinforcement learning to find trajectories that should not visit barrier states, can always learn the optimal policy π_t^* for the final target reward.*

V. Experiments

We evaluate our approach on four axes of complexity: (1) the size of barrier, (2) the number of barriers, (3) barriers in 3D environments, and (4) barriers that are not represented by physical obstacles but by a set of 'undesirable' states.

To evaluate these axes, we use various domains including navigation (Figs. 4, 5), lunar lander (Fig. 6 Left), fetch reach (Fig. 6 Right), mujoco ant (Fig. 7), and assistive feeding task (Fig. 8). We compare our approach against naïve fine-tuning (Fine-tune) as well as three state-of-the-art fine-tuning approaches: Progressive Neural Networks (PNN) [35], Batch Spectral Shrinkage (BSS) [7], and L^2-SP [27]. We also add training on the target task from a random initialization (Random) as a reference, but we do not consider Random as a comparable baseline because it is not a transfer learning algorithm. We evaluate all the experiments using the total number of interaction steps it takes to reach within some small distance of the desired return in the target task. We report the average number of interaction steps over in units of 1000 (lower is better). We indicate statistically significant differences ($p < 0.05$) with baselines by listing the first letter of those baselines. We ran Navigation (barrier sizes), and Fetch Reach experiments with 5 random seeds and the rest with 10 random seeds. If more than half of the runs exceeded the maximum number of interaction steps without reaching the desired target task reward, we report that the task is unachievable with the maximum number of interaction steps. Finally, we use stochastic policies which is why our source and target policies may not be symmetrical. Experiment details are in Section V of the supplementary materials.

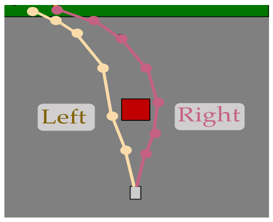

Fig. 4: Navigation environment with barrier size 5.

	Barrier Sizes			
	1	3	5	7
Ours	117.4 ± 128.6	162.6 ± 70.5^{f}	**102.7** ± 87.8^{l,b}	**112.3** ± 111.3^{l,b,f}
PNN	**92.2** ± 102	**138.6** ± 92.1	159.8 ± 90.6	119.2 ± 125
L^2-SP	138.2 ± 61.3	>256	>256	>256
BSS	>256	>256	>256	>256
Fine-tune	141.1 ± 53	>256	157 ± 100	241 ± 27.5
Random	54.6 ± 61.5	88.4 ± 59.4	145 ± 74.8	77.1 ± 40.6

TABLE I: Larger barrier sizes make fine-tuning more challenging. Our approach performs comparably with small sizes and outperforms other methods with large sizes. We only use one curriculum step, so the reward weight and the barrier set size approaches are the same.

1. Navigation. We address the first two axes by analyzing our problem under varying barrier sizes and varying number of homotopy classes. We experiment with our running example where an agent must navigate from a fixed start position s_i to the goal set $\mathbf{S}_g$ (green area).

Varying Barrier Sizes. We investigate how varying the size of the barrier affects the fine-tuning problem going from Right to Left. Here, we use a one-step curriculum so the barrier set size and reward weight approaches are the same. Table I demonstrates that when barrier sizes are small (1,3), our approach is not the most sample efficient, but remains comparable to other methods. With larger barrier sizes (5, 7), we find that our method requires the least amount of training updates. **This result suggests that our approach is especially useful when barriers are large (i.e., fine-tuning is hard). When fine-tuning is easy, simpler approaches like starting from a random initialization can be used.**

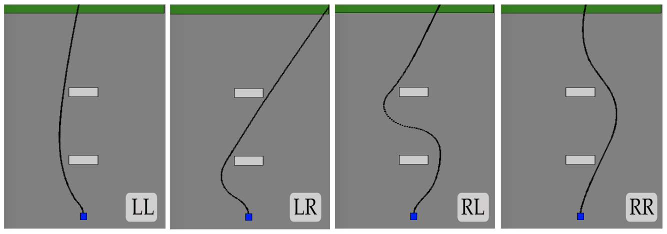

Fig. 5: Navigation environment with four homotopy classes.

Four Homotopy Classes. We next investigate how multiple homotopy classes can affect fine-tuning. As shown in Fig. 5, adding a second barrier creates four homotopy classes: *LL*, *LR*, *RL*, and *RR*. We experiment with both barrier set size and reward weight approaches and report results when using *LL* as our source task in Table II. Results for using *LR*, *RL*, and *RR* as the source task are included in the supplementary materials. We can observe that the proposed Ease-In-Ease-out approach outperforms other fine-tuning methods. Having multiple barriers does not satisfy the single barrier assumption, so our reward weight approach performs better on average than the barrier set size approach. Note that in *LL* → *LR*, *Random* performs best, which implies that the task is easy to learn from scratch and no transfer learning is needed. **We conclude that while increasing the number of barrier sets can result in a more challenging fine-tuning problem for other methods, it does not negatively affect our approach.**

	Transfer Tasks		
	LL → LR	LL → RL	LL → RR
Ours:barrier	88.1±3.2	52.1±14.2	**48.1**±13.2^{p,l,b,f}
Ours:reward	**63.2**±9.1^{p,l,b,f}	**47.1**±10.9^{p,l,b,f}	56.5±9.9
PNN	101.9± 37.2	>300	119.2 ± 36.4
L^2-SP	130.6± 28.6	>300	>300
BSS	>300	>300	>300
Fine-tune	141.2±12.1	>300	>300
Random	43.5±4.1	>300	169.4±27.1

TABLE II: Fine-tuning with multiple homotopy classes.

2. Lunar Lander. Before exploring 3D environments that differ significantly from the navigation environment, we conducted an experiment in Lunar Lander. The objective of the game is to land on the ground between the two flags without crashing. As shown in Fig. 6 (Left), this environment is similar to the navigation environments in that we introduce a barrier which creates two homotopy classes: Left and Right. However, the main difference is that the agent is controlled by two lateral thrusters and a main engine.

	Lunar Lander	
	L → R	R → L
Ours:barrier	80.46±46.58	80.23±39.76
Ours:reward	**75.13**±34.25^{p,b,f}	**38.43**±6.46^{p,l,b,f}
PNN	117.35±3.35	128.59±44.56
L^2-SP	124.54±69.99	94.59±51.23
BSS	>300	>300
Fine-tune	>300	>300
Random	232.32± 48.21	162.92±49.54

TABLE III

Our approach outperforms baselines in the Lunar Lander domain.

Results are shown in Table III. We observe that while L^2-SP suffers from a large variance and PNN needs many more steps, both our reward weight approach and barrier set size approach outperform the fine-tuning methods. The reward weight approach has a small standard deviation and performs stably. Note that Random requires large amount of interaction steps, meaning that training the landing task is originally quite difficult and needs transfer reinforcement learning. **Our approach significantly reduces the number of steps needed to learn the optimal policy in both directions.**

2. Fetch Reach. We address the third axis by evaluating our Ease-In-Ease-Out fine-tuning approach on a more realistic Fetch Reach environment [5]. The Fetch manipulator must fine-tune across homotopy classes in $\mathbb{R}^3$. In the reaching task, the robot needs to reach either the orange or blue tables by stretching right or left respectively. The tables are separated by a wall which creates two homotopy classes, as shown in Fig. 6. Our results are shown in Table IV. We find that our approach was the most efficient compared to baseline methods. One reason why the baselines did not perform well was that the wall's large size and its proximity to the robot caused it to collide often, making it particularly difficult to fine-tune across homotopy classes. We found that even training from a random initialization proved difficult. For this reason, we had to relax the barrier constraint to obtain valid Left and Right source policies.

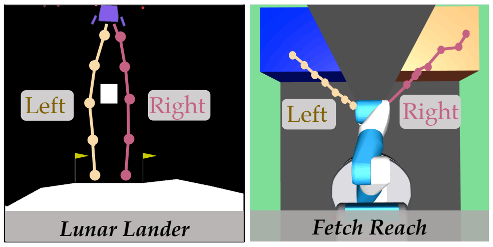

Fig. 6: (Left) Lunar lander environment with two homotopy classes. (Right) Fetch reach environment. The robot must learn to reach to the right or left of the wall.

	Fetch Reach	
	L $\rightarrow$ R	R $\rightarrow$ L
Ours:barrier	**308.7**$\pm$167.7^{p,b}	**274**$\pm$130.5^{p,l,b,f}
PNN	>500	>500
L^2-SP	>500	>500
BSS	>500	>500
Fine-tune	>500	>500
Random	>500	>500

TABLE IV: Our approach overcomes challenging domains where the barrier is extremely close to the robot and collision (and negative rewards) during training is frequent. Other methods are not able to find good policies as efficiently.

3. Mujoco Ant. Finally, we explore whether our algorithm can generalize beyond navigation-like tasks that are traditionally associated with homotopy classes. We demonstrate two examples–Mujoco Ant and Assistive Feeding–where barrier states correspond to undesirable states rather than physical objects. In the Mujoco Ant environment [43], the barrier states correspond to a set of joint angles $\{x \in \frac{\pi}{4} \pm 0.2 \text{ rad}\}$ that the ant's upper right leg cannot move to. The boundary of the barrier states are shown by the red lines in Fig. 7. In our source task, the ant moves while its upper right joint remains greater than $\frac{\pi}{4} + 0.2$ rad. We call this orientation *Down*. Our goal is to transfer to the target task where the joint angle is less than $\frac{\pi}{4} - 0.2$ rad, or *Up*. Results are shown in Table V. We do not evaluate the other direction, *Up* $\rightarrow$ *Down*, because this direction was easy for all of our baselines to begin with, including our own approach. **We find that our approach was the most successful in fine-tuning across the set of joint angle barrier states**.

4. Assistive Gym. We use an assistive feeding environment [15] to create another type of non-physical barrier in the robot's range of motion. In Fig. 8 (right), we simulate a disabled person who cannot change her head orientation by a large

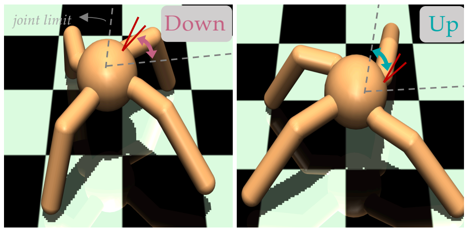

Fig. 7: Mujoco Ant environment with a non-physical barrier. The red lines are the barrier states, or the joint angles the leg cannot move to. The grey dotted lines are the upper right leg's joint limits.

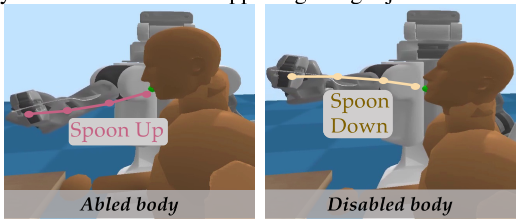

Fig. 8: Assistive Feeding environment. The barrier states represent the horizontal spoon orientation. These states are undesirable for feeding because it misplaces the food in the human's mouth.

	Mujoco Ant	**Assistive Feeding**
	Down $\rightarrow$ Up	Up $\rightarrow$ Down
Ours	**1420.0**$\pm$268.8^{p,l,b,f}	**416**$\pm$32^{p,l,b,f}
PNN	>10000	>2000
L^2-SP	>10000	>2000
BSS	>10000	>2000
Fine-tune	2058.5$\pm$535.2	>2000
Random	2290.4$\pm$585.8	494$\pm$28

TABLE V: Our approach works well in more general environments where barriers represent undesirable states instead of physical objects. We only use one curriculum step, so the reward weight and the barrier set size approaches are the same.

amount. The goal is to feed the person using a spoon. Here, we can easily train a policy on an abled body with a normal head orientation, as in Fig. 8 (left). However, we have limited data for the head orientation of the disabled person (the chin is pointing upwards as it is common in patients who use a head tracking device). To feed a disabled body, the spoon needs to point down, while for an abled body, the spoon needs to point up. The barrier states correspond to holding the spoon in any direction between these two directions when close to the mouth, which may 'feed' the food to the user's nose or chin. This environment is an example of settings with limited data in the target environment, i.e., interacting with the disabled person. It also shows a setting with no physical barriers, and the 'barrier states' correspond to the spoon orientations in between, which can be uncomfortable or even unsafe. **As shown in Table V, Our Ease-In-Ease-Out fine-tuning approach learns the new policy for the disabled person faster than training from scratch while the other fine-tuning methods fail to learn the target policy.**

VI. Discussion

Summary. We introduce the idea of using homotopy classes to characterize the difficulty of fine-tuning between tasks with different reward functions. We propose a novel Ease-In-Ease-Out fine-tuning method that first relaxes the problem and then forms a curriculum. We extend the notion of homotopy classes, which allows us to go beyond navigation environments and apply our approach on more general robotics tasks. We demonstrate that our method requires less samples on a variety of domains and tasks compared to other fine-tuning baselines. **Limitations and Future Work**. Our work has a number of limitations. This includes the need for accessing the barrier states a priori. However, our assistive gym example is a step towards considering environments where barrier states are not as clearly defined a priori. In the future, we plan to apply our methods to other robotics domains with non-trivial homotopy classes by directly finding the homotopy classes [6] and then using our algorithm to fine-tune.

VII. Acknowledgements

We would like to thank NSF Award Number 2006388 and the DARPA Hicon-Learn project for their support.

References

[1] A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul, H. P. van Hasselt, and D. Silver, "Successor features for transfer in reinforcement learning," in *NeurIPS*, vol. 30, 2017, pp. 4055–4065.
[2] A. G. Barto and T. G. Dietterich, "Reinforcement learning and its relationship to supervised learning," *Handbook of learning and approximate dynamic programming*, 2004.
[3] J. Baxter and P. L. Bartlett, "Infinite-horizon policy-gradient estimation," *JAIR*, 2001.
[4] S. Bhattacharya, M. Likhachev, and V. Kumar, "Topological constraints in search-based robot path planning," *Autonomous Robots*, 2012.
[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.
[6] C. Chen and D. Freedman, "Measuring and computing natural generators for homology groups," *Computational Geometry*, 2010.
[7] X. Chen, S. Wang, B. Fu, M. Long, and J. Wang, "Catastrophic forgetting meets negative transfer: Batch spectral shrinkage for safe transfer learning," in *NeurIPS*, vol. 32, 2019, pp. 1908–1918.
[8] I. Clavera, A. Nagabandi, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn, "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," in *ICLR*, 2019. [Online]. Available: https://openreview.net/forum?id=HyztsoC5Y7
[9] K. Cobbe, O. Klimov, C. Hesse, T. Kim, and J. Schulman, "Quantifying generalization in reinforcement learning," in *ICML*, 2019.
[10] S. Dasari, F. Ebert, S. Tian, S. Nair, B. Bucher, K. Schmeckpeper, S. Singh, S. Levine, and C. Finn, "Robonet: Large-scale multi-robot learning," in *CoRL*, vol. 100, 2020, pp. 885–897.
[11] M. P. Deisenroth, P. Englert, J. Peters, and D. Fox, "Multi-task policy search for robotics," in *ICRA*, 2014.
[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *CVPR*, 2009.
[13] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Proceedings of the 1st Annual Conference on Robot Learning*, vol. 78, 2017, pp. 1–16.
[14] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "Rl^2: Fast reinforcement learning via slow reinforcement learning," *arXiv preprint arXiv:1611.02779*, 2016.
[15] Z. Erickson, V. Gangaram, A. Kapusta, C. K. Liu, and C. C. Kemp, "Assistive gym: A physics simulation framework for assistive robotics," *ICRA*, 2020.
[16] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *ICML*, 2017.
[17] C. Finn, X. Y. Tan, Y. Duan, T. Darrell, S. Levine, and P. Abbeel, "Deep spatial autoencoders for visuomotor learning," in *ICRA*, 2016.
[18] J. Garcia and F. Fernández, "Safe exploration of state and action spaces in reinforcement learning," *JAIR*, 2012.
[19] A. Gupta, A. Murali, D. P. Gandhi, and L. Pinto, "Robot learning in homes: Improving generalization and reducing dataset bias," in *NeurIPS*, vol. 31, 2018, pp. 9094–9104.
[20] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, "Meta-reinforcement learning of structured exploration strategies," in *NeurIPS*, vol. 31, 2018, pp. 5302–5311.
[21] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, 2006.
[22] R. Julian, B. Swanson, G. S. Sukhatme, S. Levine, C. Finn, and K. Hausman, "Efficient adaptation for end-to-end vision-based robotic manipulation," in *Visual Learning and Reasoning for Robotic Manipulation Workshop*, 2020.
[23] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine, "Scalable deep reinforcement learning for vision-based robotic manipulation," in *CoRL*, vol. 87, 2018, pp. 651–673.
[24] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *IJRR*, 2013.
[25] S. Lange, M. Riedmiller, and A. Voigtländer, "Autonomous reinforcement learning on raw visual input data in a real world application," in *IJCNN*, 2012.
[26] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *JMLR*, 2016.
[27] X. Li, Y. Grandvalet, and F. Davoine, "Explicit inductive bias for transfer learning with convolutional networks," in *ICML*, vol. 80, 2018, pp. 2825–2834.
[28] G. Mesnil, Y. Dauphin, X. Glorot, S. Rifai, Y. Bengio, I. Goodfellow, E. Lavoie, X. Muller, G. Desjardins, D. Warde-Farley, *et al.*, "Unsupervised and transfer learning challenge: a deep learning approach," in *Unsupervised and Transfer Learning workshop in ICML*, 2011.
[29] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," in *Deep Learning Workshop*, 2013.
[30] R. R. Murphy, *Disaster robotics*. MIT press, 2014.
[31] A. V. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine, "Visual reinforcement learning with imagined goals," in *NeurIPS*, vol. 31, 2018, pp. 9191–9200.
[32] L. Pinto and A. Gupta, "Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours," in *ICRA*, 2016.
[33] M. Plappert, M. Andrychowicz, A. Ray, B. McGrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, *et al.*, "Multi-goal reinforcement learning: Challenging robotics environments and request for research," *arXiv preprint arXiv:1802.09464*, 2018.
[34] S. Ruder, "An overview of multi-task learning in deep neural networks," *arXiv preprint arXiv:1706.05098*, 2017.
[35] 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.
[36] F. Sadeghi, A. Toshev, E. Jang, and S. Levine, "Sim2real viewpoint invariant visual servoing by recurrent control," in *CVPR*, 2018.
[37] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, 2017.
[38] B. Shen, F. Xia, C. Li, R. Martín-Martín, L. Fan, G. Wang, S. Buch, C. D'Arpino, S. Srivastava, L. P. Tchapmi, *et al.*, "igibson, a simulation environment for interactive tasks in large realisticscenes," *arXiv preprint arXiv:2012.02924*, 2020.
[39] F. Sung, L. Zhang, T. Xiang, T. Hospedales, and Y. Yang, "Learning to learn: Meta-critic networks for sample efficient learning," *arXiv preprint arXiv:1706.09529*, 2017.
[40] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
[41] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey," *JMLR*, 2009.
[42] Y. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu, "Distral: Robust multitask reinforcement learning," in *NeurIPS*, vol. 30, 2017, pp. 4496–4506.
[43] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *IROS*, 2012.
[44] M. Turchetta, F. Berkenkamp, and A. Krause, "Safe exploration in finite markov decision processes with gaussian processes," in *NeurIPS*, vol. 29, 2016, pp. 1–5.
[45] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," in *CogSci*, 2017.
[46] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *NeurIPS*, vol. 27, 2014, pp. 3320–3328.
[47] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *ICRA*, 2017.