2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Deep Adversarial Reinforcement Learning for Object Disentangling

Melvin Laux[1], Oleg Arenz[1], Jan Peters[1,2] and Joni Pajarinen[1,3]

***Abstract*— Deep learning in combination with improved training techniques and high computational power has led to recent advances in the field of reinforcement learning (RL) and to successful robotic RL applications such as in-hand manipulation. However, most robotic RL relies on a well known initial state distribution. In real-world tasks, this information is however often not available. For example, when disentangling waste objects the actual position of the robot w.r.t. the objects may not match the positions the RL policy was trained for. To solve this problem, we present a novel adversarial reinforcement learning (ARL) framework. The ARL framework utilizes an adversary, which is trained to steer the original agent, the protagonist, to challenging states. We train the protagonist and the adversary jointly to allow them to adapt to the changing policy of their opponent. We show that our method can generalize from training to test scenarios by training an end-to-end system for robot control to solve a challenging object disentangling task. Experiments with a KUKA LBR+ 7-DOF robot arm show that our approach outperforms the baseline method in disentangling when starting from different initial states than provided during training.**

I. INTRODUCTION

Deep reinforcement learning (DRL) methods have achieved remarkable performance in playing Atari games [1], [2], Go [3], Chess [4], Dota 2 [5] or Starcraft 2 [6]. However, tackling such tasks required a large number of policy roll-outs—typically in the order of billions. This huge sample complexity is in stark contrast to the number of roll-outs that are feasible when training a policy on a real robot, which is typically in the range of tens or hundreds. Still, deep reinforcement learning has proved its usefulness also for robot applications such as in-hand manipulation [7], object manipulation [8], quadruped locomotion [9] or autonomous driving [10]. These results were made possible by using a variety of techniques to reduce the required number of real-world interactions, such as (pre-)training in simulation, initializing from demonstrations, or incorporating prior knowledge in the design of the Markov Decision Process (MDP). However, policies that are trained with few real-world interactions can be prone to failure when presented with unseen states. It is, thus, important to carefully set the initial states of the robot and its environment so that it also includes challenging situations.

Furthermore, unlike simulations, the state of robot and environment can not be easily reset. Using a human to manually set up the environment after each roll-out can often introduce a significant bottleneck in the data collection. Hence, successful applications of reinforcement learning to robotics typically involved tasks that can be easily reset to their initial states, for example playing table tennis against a ball-cannon [11] or grasping objects from a bin [12]. Based on these observations, we investigate how to *automatically* set up the environment to a *challenging* state for the agent.

We present a novel adversarial learning framework for reinforcement learning algorithms: Adversarial reinforcement learning (ARL). This framework helps to learn RL policies that can generalize to new situations by employing an adversarial policy to steer the agent to unknown or difficult situations. Through this adversary driven exploration of the

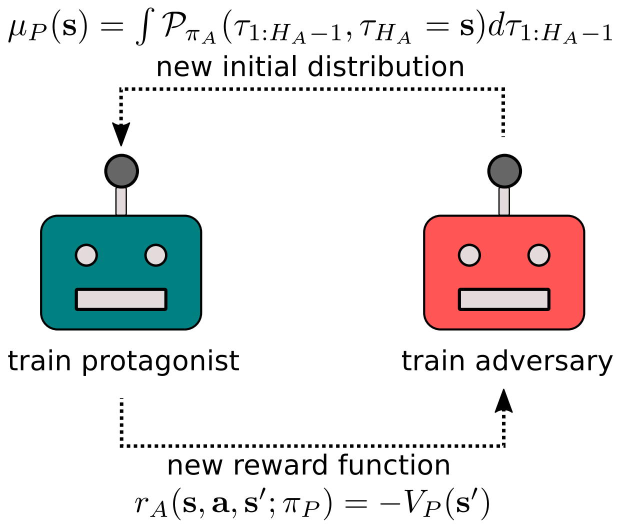
$\mu_P(\mathbf{s}) = \int \mathcal{P}_{\pi_A}(\tau_{1:H_A-1}, \tau_{H_A} = \mathbf{s}) d\tau_{1:H_A-1}$
new initial distribution
train protagonist
train adversary
new reward function
$r_A(\mathbf{s}, \mathbf{a}, \mathbf{s}'; \pi_P) = -V_P(\mathbf{s}')$

Fig. 1. During training our Adversarial Reinforcement Learning (ARL) approach loops over two steps. First, ARL trains the adversary to maximize its current reward function, which depends on the protagonist's current value function $V_P(\mathbf{s}')$. The adversary's updated policy generates a new challenging initial state distribution $\mu_P(\mathbf{s})$ for the protagonist. Next, the protagonist is trained to maximize expected reward when starting from this new state distribution. The ARL process leads to a protagonist that can cope with a challenging initial state while keeping the initial state distribution valid: the adversary was able to explore those states.

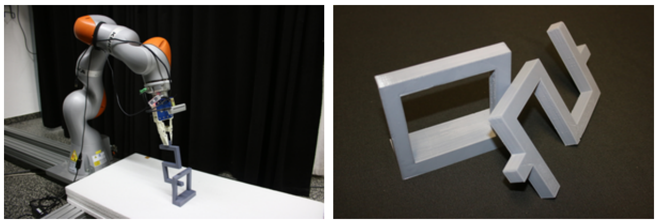

Fig. 2. We show the benefits of our ARL approach in both simulation and in a real robot task where the robot tries to disentangle an object from another one. Left: The KUKA LBR+ 7-DOF robot arm with attached SAKE gripper disentangling an "S-shape" object from an "O-shape" object. Right: The 3D printed "O-shape" and "S-shape" objects.

[1]Intelligent Autonomous Systems, TU Darmstadt, Germany
[2]MPI for Intelligent Systems, Tuebingen, Germany
[3]Learning for Intelligent Autonomous Robots, Tampere University

state space, the agent is more likely to learn a more general policy than by using purely random exploration. While we assume irreducibility of the MDP to prevent the adversary from creating impossible tasks for the protagonist, we do not make any assumptions on the reinforcement learning algorithm. An overview of the framework is illustrated in Figure 1.

We investigate the task of disentangling waste [13]. There is an enormous potential of robots in the waste processing industry by reducing human labor and allowing for better segregation and thus recycling of waste. Furthermore, there is high interest in the nuclear industry to employ robots for segregating nuclear waste, which is inherently dangerous for humans. To apply reinforcement learning to waste disentangling it would be desirable to automate the generation of challenging entanglements. However, designing such procedure by hand is not straightforward and, thus, it is interesting to investigate our learning-based approach.

Contribution.

- We propose an adversarial framework (ARL) for reinforcement learning that alternately trains an agent for solving the given task and an adversary for steering the protagonist to challenging states.
- We evaluate ARL in simulation as well as on a real robot on the object disentangling task. For the robot experiment, a KUKA iiwa R820 in combination with a SAKE EzGripper is tasked to disentangle two 3D-printed objects as shown in Figure 2.

II. RELATED WORK

Adversarial learning. The idea of embedding adversaries into the training process has been shown to work successfully in the field of supervised learning [14], [15] and imitation learning [16]. Adversarial approaches have also been previously incorporated in RL methods to learn more robust policies by adding adversary-generated noise to the policy's actions during training [17], [18], using an adversarial robot to interfere with the protagonist [19] or training an adversary to perturb the protagonist's observations to cause the agent to fail [20], [21], [22]. Gleave et al. investigate the vulnerability of DRL methods against adversarial attacks in the context of multi-agent environments [23]. They show how attacks can be carried out effectively by training adversarial policies to generate natural adversarial observation via self-play. *Robust adversarial reinforcement learning* (RARL) framework by Pinto et al. aims to train policies that are robust to the sim-to-real gap by interpreting dynamics model errors of the simulator as an additional noise and jointly training an adversary and a protagonist [17]. The adversary's goal is to learn adversarial actions that are added to the protagonist's actions at each time step as additional noise. The adversary's reward at each time-step is the negative reward of the protagonist, thus, encouraging the adversary to apply harmful noise. By training the protagonist in the presence of such an adversary, the protagonist is required to learn a policy that is robust to harmful perturbations, making it more robust to model inaccuracies of the simulator and, thus, to the sim-to-real-gap. However, in contrast to our method, such approach is in general not able to steer the agent to far away states and thus does not tackle initial state distribution mismatch. Our method is different to RARL as our method does not train the adversary to add noise to the protagonist's actions, but steer the agent to difficult states from which the protagonist takes over to solve the task without any additionally generated interference by the adversary.

Curriculum learning. Curriculum learning methods learn to solve tasks in a variety of contexts by presenting context-based tasks to the learning agent in an order of increasing complexity instead of presenting tasks randomly [24]. Held et al. train a generative adversarial net (GAN) to produce increasingly difficult contexts [25]. Florensa et al. learn a reverse curriculum for goal-oriented tasks, in which a goal state is provided that needs to be reached from any initial state in the environment [26]. The idea of this method is to maintain a set of initial positions that is initialized to the provided set of goal states. New candidates for initial states are generated by random walks beginning from states of the current set. These candidates are added to the set if they have the desired difficulty according to a simple heuristic or dropped if deemed too easy. This approach makes the assumptions that the environment's goal-state is known during training and that the environment can be reset to arbitrary states in the environment, which our approach does not. The POET algorithm uses evolution strategies to generate a range of related environments to learn policies that can transfer from one environment to another [27]. Sukhbaatar et al. train an adversary to create an automatic curriculum for the protagonist by interleaving standard RL training with adversarial self-play episodes that provide an intrinsic motivation to explore the environment [28]. This context-based approach splits the training process into two different types of episodes. In self-play episodes, an adversarial policy controls the agent and interacts with the environment until it selects a stop-action. At this point in the episode, the protagonist takes over and is tasked with either reversing the adversary's trajectory or to replay it, the desired state being provided to the policy as context. During self-play episodes, no extrinsic reward is provided to the agents: the adversary is rewarded positively for each time step that the protagonist requires to reach the desired state and negatively for each own action. This reward structure encourages the adversary to generate increasingly difficult self-play scenarios for the protagonist. The protagonist's reward in self-play episodes is the negative number of steps required to reach the desired state. The second type of episodes simply consist of the target task that the protagonist is expected to learn. The context in such episodes is set to zero and an additional flag is set to inform the policy of current type of episode. We attempted to test the self-play approach in our maze environment. However, we were unable to achieve better results than standard SAC in the continuous maze environment despite best efforts. We hypothesize that the approach does not scale well to continuous state and action spaces. These findings are in line with the observations made by [26].

III. ADVERSARIAL REINFORCEMENT LEARNING

We consider an irreducible MDP with state space $\mathcal{S}$, action space $\mathcal{A}$, system dynamics $\mathcal{P}(\mathbf{s}'|\mathbf{s},\mathbf{a})$, initial state distribution $\mu(\mathbf{s})$, reward function $r(\mathbf{s},\mathbf{a},\mathbf{s}')$, discount factor γ and a finite horizon H_p. The goal of RL methods is to find a parameterized policy $\pi_\theta(\mathbf{a}|\mathbf{s})$ that maximizes the accumulated discounted reward when sampling the state at time step zero from the initial distribution $\mu(\mathbf{s})$ and afterwards following the policy π_θ and system dynamics $\mathcal{P}$, i.e.,

$$J(\pi) = \mathbb{E}_{\mu,\pi_\theta,\mathcal{P}}\left[\sum_{t=0}^{H_p-1} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t)\right]. \tag{1}$$

However, we assume that the initial state distribution $\mu(\mathbf{s})$ is unknown and resetting the robot to it is infeasible. Instead, we assume that the environment can only be reset according to a reset distribution $\mu_R(\mathbf{s})$ which is typically less diverse and less challenging. For example, we want to solve the robot disentangling task for a wide range of entanglements but are only able to reset the robot (and its attached object) to a small number of manually specified positions.

The main idea of our approach is to reduce the distribution mismatch (covariate shift) between the reset distribution and the unknown initial distribution by steering the agent into regions in which the current policy performs poorly. To steer the agent into these regions, we train an adversarial agent that chooses the protagonist's initial position by interacting with the environment. The adversary's task is to find areas in the environment's state space that are challenging for the protagonist. The adversary is rewarded for moving the agent to situations that are expected to yield low returns for the protagonist under its current policy. At the beginning of each episode, the agent is controlled by the adversarial policy for a fixed number of time steps after which the protagonist takes over again. The adversary's and protagonist's policies are trained jointly to allow the adversary to adapt to the changing protagonist policy.

Our learning framework consists of an MDP environment E of horizon H_P and two policies, the *protagonist* π_P and the *adversary* π_A, which both interact with the same environment. The ARL framework aims to learn a protagonist's policy π_P that performs well in as many areas of the environment as possible. We extend the original RL framework, to encourage directed exploration of difficult areas. At the beginning of each training episode, the agent acts for H_A time steps under policy π_A and subsequently follow the protagonist policy π_P for H_P time steps. We can formulate the protagonist's objective function in the following way:

$$J(\pi_P) = \mathbb{E}_{\mu_P,\pi_P,\mathcal{P}}\left[\sum_{t=0}^{H_P-1} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t)\right], \tag{2}$$

where the protagonist's initial state probability $\mu_P(\mathbf{s}_0)$ is produced by executing the adversary for H_A steps starting from a state that is sampled from the reset distribution $\mu_R(\mathbf{s})$.

The adversary's goal is to find states $\mathbf{s}$ in the environment for which the protagonist's expected return is low. The protagonist's state-value function V^π estimates the expected return of following policy π starting in state $\mathbf{s}$. Thus, we can define a reward function $r(\mathbf{s},\mathbf{a},\mathbf{s}')$ that evaluates the adversary's action through the protagonist's negative state-value of the reached state, that is,

$$r_A(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t; \pi_P) = -V_P(\mathbf{s}'), \tag{3}$$

where V_P is the current protagonist's estimated state-value function. It is to note that we do not only provide reward based on the adversary's final state but also for immediate steps in order to provide a more shaped reward. With this choice of reward function, the adversary is encouraged to explore the environment to find states in which the protagonist's policy is expected to perform poorly. We can formulate the adversary's objective as

$$J(\pi_A) = \mathbb{E}_{\mu,\pi_A,\mathcal{P}}\left[\sum_{t=0}^{H_A-1} \gamma^t r_A(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}'_t; \pi_P)\right]. \tag{4}$$

We jointly train both agents by alternating between improving the adversary's policy and the protagonist's policy. Figure 1 presents a high-level visualization of the ARL learning framework. Our implementation is based on the soft actor-critic [9] RL algorithm which is off-policy and, thus, we describe our implementation for off-policy reinforcement learning. It is, however, also possible to use the ARL framework in combination with on-policy RL methods. Training begins by updating the adversary for K_A episodes. At the start of each episode, the environment is reset to a state sampled from μ_R. Next, at each timestep t, the adversary selects an action using its policy $\mathbf{a}_t^{(A)} \sim \pi_A$ and observes the next state $\mathbf{s}_{t+1}$. The adversary's reward is computed using the current approximation of the protagonist's state-value function, i.e. $r_t^{(A)} = -V_P(\mathbf{s}')$. The tuple $(\mathbf{s}_t, \mathbf{a}_t^{(A)}, r_t^{(A)}, \mathbf{s}_{t+1})$ is added to the adversary's replay buffer, from which a minibatch of transitions is sampled to update the adversary's policy. After H_A steps, the protagonist begins to interact with the environment. Note that the environment is not reset before the protagonist begins its interactions, thus the protagonist's initial position is $\mathbf{s}_{H_A}$. The protagonist collects tuples $(\mathbf{s}_t, \mathbf{a}_t^{(P)}, r_t^{(P)}, \mathbf{s}_{t+1})$ for H_P time steps. The tuples are added to the protagonist's replay buffer D_A. However, the protagonist's policy is not updated during these rollouts. Once the protagonist has completed its interaction with the environment, the environment is reset. This process is repeated for K_A episodes in which the adversary's policy is updated at each time step, while the protagonist simply collects experiences to store in its replay buffer D_P. After K_A rollouts, it is the protagonist's turn to be improved. At the beginning of each rollout, the environment is reset. Next, the adversary interacts with the environment for H_A time steps to generate the protagonist's starting position, while storing the experienced transitions in its replay buffer D_A. After the adversary has completed H_A

actions, it is the protagonist's turn again. For H_P time steps t, the protagonist samples action $\mathbf{a}_t^{(P)} \sim \pi_P(\mathbf{s}_t)$, observes the new state $\mathbf{s}_{t+1}$ and receives the reward $r_t^{(P)}$. Next, the tuple $(\mathbf{s}_t, \mathbf{a}_t^{(P)}, r_t^{(P)}, \mathbf{s}_{t+1})$ is added to the replay buffer D_P and the protagonist's policy π_P and state-value function approximation V_P are updated using a minibatch of samples drawn from D_P. This sequence of alternating between training the adversary and training the protagonist is repeated for a fixed number of iterations N. It should be noted that the ARL framework introduces three additional hyperparameters: K_A, K_P and H_A. Furthermore, it is possible to use any off-policy RL method that approximates the protagonist's state value function V_P in this learning framework. Algorithm 1 shows the pseudocode of our proposed method.

Algorithm 1: Off-policy adversarial reinforcement learning

```
Input: Arbitrary initial policies $\pi_A$ and $\pi_P$ with
       replay buffers $D_A$ and $D_P$, Environment $E$,
for $i \leftarrow 0$ to $N$ do
    for $j \leftarrow 0$ to $K_A$ do
        reset($E$) ;
        for $t \leftarrow 0$ to $H_A$ do
            $(\mathbf{s}_t, \mathbf{a}_t^{(A)}, r_t^{(A)}, \mathbf{s}_{t+1}) \leftarrow$ step($E$, $\pi_A$));
            $D_A \leftarrow D_A \cup (\mathbf{s}_t, \mathbf{a}_t^{(A)}, r_t^{(A)}, \mathbf{s}_{t+1})$;
            $\pi_A \leftarrow$ train($\pi_A$, $D_A$) ;
        for $t \leftarrow 0$ to $H_P$ do
            $(\mathbf{s}_t, \mathbf{a}_t^{(P)}, r_t^{(P)}, \mathbf{s}_{t+1}) \leftarrow$ step($E$, $\pi_P$));
            $D_P \leftarrow D_P \cup (\mathbf{s}_t, \mathbf{a}_t^{(P)}, r_t^{(P)}, \mathbf{s}_{t+1})$;
    for $j \leftarrow 0$ to $K_P$ do
        reset($E$) ;
        for $t \leftarrow 0$ to $H_A$ do
            $(\mathbf{s}_t, \mathbf{a}_t^{(A)}, r_t^{(A)}, \mathbf{s}_{t+1}) \leftarrow$ step($E$, $\pi_A$));
            $D_A \leftarrow D_A \cup (\mathbf{s}_t, \mathbf{s}_t^{(A)}, r_t^{(A)}, \mathbf{s}_{t+1})$;
        for $t \leftarrow 0$ to $H_P$ do
            $(\mathbf{s}_t, \mathbf{a}_t^{(P)}, r_t^{(P)}, \mathbf{s}_{t+1}) \leftarrow$ step($E$, $\pi_P$));
            $D_P \leftarrow D_P \cup (\mathbf{s}_t, \mathbf{a}_t^{(P)}, r_t^{(P)}, \mathbf{s}_{t+1})$;
            $\pi_P \leftarrow$ train($\pi_P$, $D_P$) ;
```

IV. EXPERIMENTAL RESULTS

A. Environments

1) Object disentangling: In the first environment, which we call the *Robot-arm disentangling environment* (RADE), the agent controls a seven degree-of-freedom KUKA LBR iiwa R820 robotic arm which is equipped with a SAKE EzGripper end-effector, holding an "S"-shaped object which is entangled with a second, "O"-shaped object. Figure 2 shows the experimental hardware setup. The MDP's state space is represented as a 7-dimensional vector of the current robot joint positions. The agent's goal is to disentangle the two objects through direct joint actions, i.e. providing joint deltas for all seven joints at each time step, in a fixed number of steps without colliding. The task solved if the euclidean distance d between the objects' centers exceeds a threshold of 0.5 meters. The agent receives a positive reward for successfully solving the disentangling task. Collisions are penalized with a negative reward that depends on the current time step, where early collisions are punished more severely than later ones. Otherwise, a small action penalty is given to the agent to encourage smaller steps. Thus, the reward function for the disentangling task is formalized as,

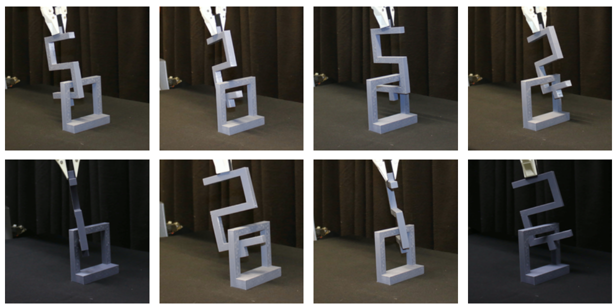
Fig. 3. The top row of images shows the 4 initial positions from the training set, the bottom row the entanglements from the test set. Note that for both sets, the S-shape is positioned in the two top corners of the O-shape, once from each side. The test set consists of slight variations of the positions from the training set.

$$r(\mathbf{s}, \mathbf{a}, \mathbf{s}') = \begin{cases} -\frac{1-\gamma^{H \;\; t+1}}{1-\gamma}, & \text{if collision detected,} \\ 1, & \text{if } d \geq 0.5, \\ -\|\mathbf{a}\|_2, & \text{otherwise.} \end{cases} \tag{5}$$

We evaluated the learned protagonist polices in two distinct ways to measure and compare their respective performances. The first method evaluates policies in simulation on a test set of initial positions. This test set consists of four different joint configurations in which the objects are entangled. These configurations have not been used as initial state for training. To measure how well an agent can generalize to new situations, we evaluate the protagonist's performance over 100 episodes while sampling the initial position uniformly from this test set. Figure 3 shows the possible initial position from both the training set and the test set. Additionally, we tested the learned policy's performance by evaluating it in the real world environment. The initial states were sampled from the same distribution as during training, i.e. $\mu(\mathbf{s}) = \mu_R(\mathbf{s})$.

2) Continuous maze: Additionally, we evaluated our approach in a continuous maze environment, in which the agent must navigate a complex maze to reach a goal position. While this toy example does not require DRL methods to be solved and could easily be trained uniformly on the complete state space in simulation, we chose to examine the ARL method on this task to easily visualize how the addition of an adversary affects the protagonist's performance over time. Furthermore, it shows that our method is applicable to a variety of tasks other than object disentangling. In the

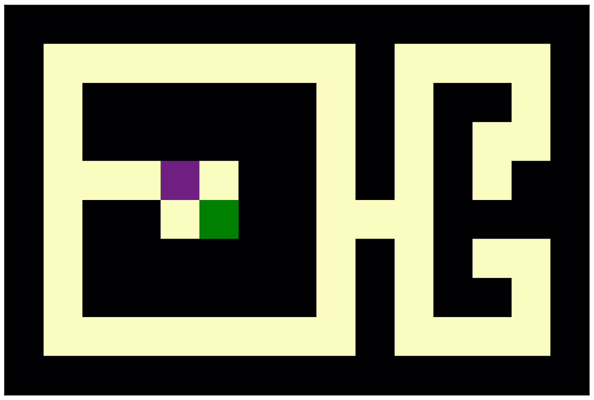

Fig. 4. Illustration of the continuous maze environment. The agent's task is to navigate a point mass to the green goal square. Actions are continuous position deltas, observations are continuous 2D coordinates. In real-world environments, creating meaningful training scenario distributions requires domain knowledge. Instead, generating initial states close to the tasks goal state requires little domain knowledge and human labor. Thus, during training, the agent's start position is sampled from the reset distribution (purple). The adversary then moves to another state which becomes the initial state for the protagonist. The task's true initial state distribution, the uniform distribution over all states in the maze (light yellow), is only used for evaluation.

maze environment, the agent observes the current state as continuous 2D coordinates in the maze. The available actions to the agent are continuous velocities which are capped at a maximum of 1 square per time step. The agent receives a positive reward if the goal square is reached or a small negative movement penalty otherwise. Actions that would result in collisions with the maze walls ignore the component of the action that causes the collision. The agent's task in this environment is to reach the goal position from any point in the maze within 100 time steps. During training, only a limited set of training scenarios is available, as the initial position of the agent is sampled from a single square close to the goal. The maze environment is illustrated in Figure 4.

3) Research questions: We designed our experiments in both environments to answer the following research questions:

1) Does ARL generalize better than standard RL?
2) How robust is ARL w.r.t. its hyperparameters?

B. Implementation details

In order to facilitate experiments, we created a simulated version of the disentangling environment. The existing robot simulator and controller *Simulation Lab* (SL) [29] was used, both, for simulation and to control the real robots. Furthermore, we implemented our environments as extensions to the existing *OpenAI gym* environments [30]. Using this standardized environment interface allowed us to easily use readily available implementations of state-of-the-art RL methods like Soft Actor-Critic (SAC) from the *OpenAI baselines* [31] and *stable-baselines* [32] projects for our experiments.

TABLE I

MEAN RETURN AND SUCCESS RATE ($\pm$ STANDARD ERROR), OVER 10 EVALUATION RUNS, FOR SAC AND ASAC ON THE TRAINING AND TEST SETS. MAXIMUM VALUES ARE IN BOLD FONT. ASAC OUTPERFORMS SAC ON THE TEST SET.

Method	Training		Test set	
	return	success rate (%)	return	success rate (%)
SAC	0.44 ± 0.29	90.5 ± 4.84	-3.90 ± 0.39	17.0 ± 6.63
ASAC10 ($H_A = 20$)	0.20 ± 0.51	86.4 ± 8.65	-3.71 ± 0.60	20.20 ± 10.11
ASAC100 ($H_A = 20$)	0.07 ± 0.43	84.2 ± 7.31	-2.60 ± 0.72	39.00 ± 12.27
ASAC1000 ($H_A = 20$)	-0.42 ± 0.63	76.0 ± 10.67	-3.83 ± 0.45	18.2 ± 7.64
ASAC10 ($H_A = 5$)	-0.18 ± 0.35	80.0 ± 6.01	-3.57 ± 0.41	22.6 ± 6.96
ASAC100 ($H_A = 5$)	-0.51 ± 0.49	74.34 ± 8.28	-3.03 ± 0.43	31.78 ± 7.26
ASAC1000 ($H_A = 5$)	-0.26 ± 0.25	78.6 ± 4.25	-2.74 ± 0.37	36.7 ± 6.21
ASAC10 ($H_A = 1$)	0.30 ± 0.29	88.1 ± 4.99	$\mathbf{-2.56 \pm 0.31}$	$\mathbf{39.7 \pm 5.27}$
ASAC100 ($H_A = 1$)	$\mathbf{0.70 \pm 0.05}$	$\mathbf{94.88 \pm 0.83}$	-3.01 ± 0.28	32.13 ± 4.78
ASAC1000 ($H_A = 1$)	0.00 ± 0.33	83.1 ± 5.60	-2.81 ± 0.36	35.0 ± 6.39

C. Results

To answer the previously stated research questions, we carried out a small sensitivity study, comparing various hyperparameter settings. Our main focus in these experiments was to investigate how the number of training episodes per iteration, K_A and K_P, and the adversary's horizon, H_A, affect the learning process. To analyze the effects of these hyperparameters, we trained three different variants of adversarial SAC with varying choices of K_A and K_P. Hereby, we chose to set $K_A = K_P$ to allow an equal number of training episodes for both protagonist and adversary:

- **ASAC10**: Adversarial SAC with $K_A = K_P = 10$
- **ASAC100**: Adversarial SAC with $K_A = K_P = 100$
- **ASAC1000**: Adversarial SAC with $K_A = K_P = 1000$

Note that all policies were trained for the same number of episodes. Thus, the number of training iterations N depends on the choice of K_A and K_P, i.e. to train ASAC10 for ten thousand episodes, one thousand iterations were required, while training ASAC1000 for the same amount of episodes only 10 training iterations are required.

1) Disentangling results: In the disentangling environment, we tested all three variants of ASAC using three different horizons for the adversary and compared them against a baseline of standard SAC. In the following, we will present and discuss the results of this sensitivity study. We used feedforward neural network policies with 2 hidden layers of 64 units for all disentangling experiments. All SAC hyperparameters were set equally, performing 5 optimization steps for every step in the environment with a learning rate α of 0.0003. Each policy was trained for ten thousand episodes.

Table I shows the performance of the learned protagonist policies on both the training set and the test set in simulation. The left side of the table shows the mean scores and percentages of solving the task of the final 100 episodes of training. Despite the presence of an adversary, all variants of ASAC achieve a similar performance to SAC on the training set. We can observe that ARL leads to policies that are able to solve the majority of examples presented to it. The performance of most ASAC methods is lower than that of SAC since SAC policies were trained on the ideal initial state distribution. We evaluated the final policies of 10 runs on the test set (25 trials per test scenario) in simulation. We can observe that all ASAC policies achieve better performance than SAC. ASAC

TABLE II

MEAN RETURN AND SUCCESS RATE ($\pm$ STANDARD ERROR), OVER 5 EVALUATION RUNS ON THE REAL ROBOT, FOR SAC AND SINGLE STEP ASAC10 ON THE TRAINING AND TEST SETS. ASAC OUTPERFORMS SAC ON THE TRAINING SET AND THE TEST SET.

Method	Training		Test set	
	return	success rate (%)	return	success rate (%)
SAC	-0.44 ± 0.55	80.0 ± 9.35	-4.31 ± 0.36	10.0 ± 6.12
ASAC10	$\mathbf{0.41 \pm 0.59}$	$\mathbf{90.0 \pm 10.00}$	$\mathbf{-2.44 \pm 1.02}$	$\mathbf{41.67 \pm 17.28}$

using a 1-step adversary can solve the presented test scenario twice as often as SAC. We hypothesize that the slightly better performance of ASAC with a 1-step adversary, compared to those with larger horizon is caused by the similarity of the training and test scenarios, thus only a single action of the adversary generates a new meaningful scenario. While larger horizons enable the adversary to potentially find a greater variety of difficult scenarios, they also require the adversary to explore areas of the environment that are trivial to solve for the protagonist, e.g. already disentangled positions.

We chose the method that showed the best performance on the test set in simulation, namely ASAC10 with $H_A = 1$, to be evaluated and compared against SAC in the real-world environment on both the train and test set. We evaluated five learned policies of each method for three trials per scenario, totalling 120 rollouts on the real robot. Table II shows the mean score and success rate of the policies. We can observe that ASAC not only solves the test scenarios 4 times as often, but also outperforms SAC on the training set. We believe that, due to the extended training set generated by the adversary, the protagonist was able to learn a more general policy that is more robust to the sim-to-real gap than SAC. Furthermore, the presence of an adversary improved the learned policy's performance on the test set in all disentangling experiments, indicating that ARL helps alleviate the problem of overfitting in DRL. Although simpler methods to generate extended training sets exist, e.g. random shaking at the beginning of each episode, such methods generally require domain knowledge about the given task and do not steer training to areas that require improvement. The ARL framework required no domain knowledge and actively forces the current policy's weak points to be improved.

2) Maze results: In the following, we present the findings of the evaluation of the ARL framework in the maze environment. We evaluated the same three variants of ASAC and compared their performance against standard SAC and an additional baseline using a random adversary (RA). In all experiments, we set H_A to 100 time steps, allowing the adversary to reach any point in the maze regardless of its start position. We evaluate the protagonist's learnt policy by testing its performance from uniformly distributed starting positions over the complete state space.

Table III shows the mean episode return and success rate on the complete maze of ASAC, SAC and RA. We can observe that all ASAC methods clearly outperform SAC and RA, being able to solve the maze task from twice as many different start positions. While SAC only explores the

TABLE III

MEAN RETURN AND SUCCESS RATE ($\pm$ STANDARD ERROR), OVER 5 EVALUATION RUNS, FOR SAC, ASAC AND RA. MAXIMUM VALUES ARE IN BOLD FONT. ASAC OUTPERFORMS SAC AND RA.

Method	return	success rate (%)
SAC	-3.47 ± 0.16	27.44 ± 3.23
Random Adversary	-3.43 ± 0.19	28.08 ± 3.67
ASAC10	-2.21 ± 0.56	55.60 ± 12.19
ASAC100	-2.44 ± 0.59	51.33 ± 12.90
ASAC1000	$\mathbf{-2.08 \pm 0.63}$	$\mathbf{59.21 \pm 13.97}$

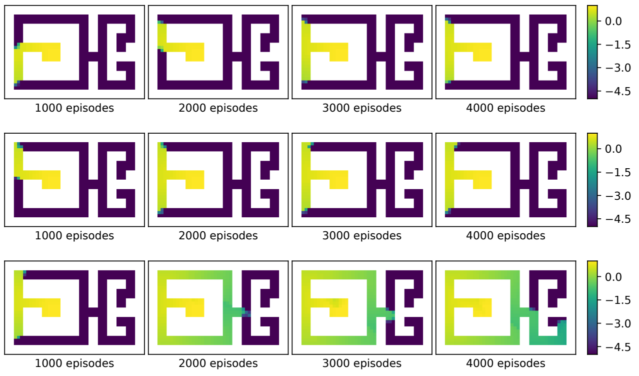

Fig. 5. Example progression of mean return after 1000, 2000, 3000 and 4000 training episodes depending on the agents initial position. Top row shows the random adversary baseline, middle shows SAC and the bottom row shows ASAC10. Guided by the adversarial policy, ASAC continuously progresses to explore distant regions of the maze. SAC and RA only explore close to the reset square.

maze close to the reset state, the random walk adversary is not able to extend the training set of scenarios in a way that aids the protagonist's learning process. Figure 5 shows heatmaps of the mean return of the protagonist the training process of ASAC, SAC and RA. While all methods are able to learn to solve task from the left side of the maze, only ASAC learns a policy that is able to solve the maze from almost everywhere in the maze. These results indicate that the adversary successfully pushes the protagonist to difficult situations resulting in a more general protagonist policy.

V. CONCLUSION

This paper introduced a novel adversarial learning framework, ARL, that trains agents to learn more general policies by steering the agent to difficult regions of the environment through adversarial policy. The approach implicitly generates new training scenarios for the protagonist, making the protagonist less likely to overfit the predefined set of training scenarios and more robust to the sim-to-real gap. We evaluated ARL on the robot disentangling task and the continuous maze. Our experiments included a sensitivity study to investigate the effects of changing hyperparameters, namely the adversary's horizon H_A and the number of training episodes per iteration K_A and K_P for each policy. The results of the sensitivity study show that the performance of ARL is robust to the choice of K_A and K_P as all tested choices led to significantly improved performance on the test scenarios. The adversary's horizon impacts the learning process as adversaries with too small horizons H_K lack the ability to reach all parts of the environment's state space. However, even a single step adversary can improve the protagonist's ability to generalize.

Our results provide several natural starting points for potential future work. First, investigating how ARL performs using other base algorithms instead of SAC remains a question to be answered. Additionally, we plan on evaluating our method one additional tasks to examine its overall generality and to further investigate the effects of hyperparameter choices. Most notably, we intend to examine the effects of different ratios between adversary and protagonist training episodes on performance and learning speed. As our previous experiments used low-dimensional state representations such as robot joint positions, we plan to test our approach using high dimensional visual inputs, i.e. camera images, to learn more general policies w.r.t. object shapes. Another follow-up would be to incorporate a shared critic that evaluates the actions of both the protagonist and the adversary to further improve learning speed and data-efficiency. Finally, we plan to evaluate whether our approach is able to increase robustness against adversarial attacks.

ACKNOWLEDGMENT

Calculations for this research were conducted on the Lichtenberg high performance computer of the TU Darmstadt.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 12 2013.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[3] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–503, 2016.

[4] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[5] OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. de Oliveira Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with large scale deep reinforcement learning," 2019.

[6] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[7] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, *et al.*, "Solving rubik's cube with a robot hand," *arXiv preprint arXiv:1910.07113*, 2019.

[8] Y. Zhu, Z. Wang, J. Merel, A. Rusu, T. Erez, S. Cabi, S. Tunyasuvunakool, J. Kramár, R. Hadsell, N. de Freitas, *et al.*, "Reinforcement and imitation learning for diverse visuomotor skills," *arXiv preprint arXiv:1802.09564*, 2018.

[9] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, "Soft actor-critic algorithms and applications," *ArXiv*, vol. abs/1812.05905, 2018.

[10] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *CoRR*, vol. abs/1604.07316, 2016.

[11] J. Kober, A. Wilhelm, E. Oztop, and J. Peters, "Reinforcement learning to adjust parametrized motor primitives to new situations," no. 4, pp. 361–379, 2012.

[12] S. Levine, P. Pastor, A. Krizhevsky, and D. Quillen, "Learning hand-eye coordination for robotic grasping with large-scale data collection," in *International Symposium on Experimental Robotics*. Springer, 2016, pp. 173–184.

[13] J. Pajarinen, O. Arenz, J. Peters, and G. Neumann, "Probabilistic approach to physical object disentangling," *IEEE Robotics and Automation Letters*, vol. 5, pp. 5510–5517, 2020.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 2672–2680.

[15] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[16] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 4565–4573.

[17] L. Pinto, J. Davidson, R. Sukthankar, and A. Gupta, "Robust adversarial reinforcement learning," *ICML*, 2017.

[18] X. Pan, D. Seita, Y. Gao, and J. F. Canny, "Risk averse robust adversarial reinforcement learning," *CoRR*, vol. abs/1904.00511, 2019.

[19] L. Pinto, J. Davidson, and A. Gupta, "Supervision via competition: Robot adversaries for learning tasks," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1601–1608.

[20] A. Pattanaik, Z. Tang, S. Liu, G. Bommannan, and G. Chowdhary, "Robust deep reinforcement learning with adversarial attacks," *CoRR*, vol. abs/1712.03632, 2017.

[21] S. Huang, N. Papernot, I. Goodfellow, Y. Duan, and P. Abbeel, "Adversarial attacks on neural network policies," *arXiv preprint arXiv:1702.02284*, 2017.

[22] A. Mandlekar, Y. Zhu, A. Garg, F. F. Li, and S. Savarese, "Adversarially robust policy learning: Active construction of physically-plausible perturbations," 09 2017, pp. 3932–3939.

[23] A. Gleave, M. Dennis, N. Kant, C. Wild, S. Levine, and S. Russell, "Adversarial policies: Attacking deep reinforcement learning," *CoRR*, vol. abs/1905.10615, 2019.

[24] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," vol. 60, 01 2009, p. 6.

[25] D. Held, X. Geng, C. Florensa, and P. Abbeel, "Automatic goal generation for reinforcement learning agents," 05 2017.

[26] C. Florensa, D. Held, M. Wulfmeier, and P. Abbeel, "Reverse curriculum generation for reinforcement learning," *CoRR*, vol. abs/1707.05300, 2017.

[27] R. Wang, J. Lehman, J. Clune, and K. O. Stanley, "Paired open-ended trailblazer (POET): endlessly generating increasingly complex and diverse learning environments and their solutions," *CoRR*, vol. abs/1901.01753, 2019.

[28] S. Sukhbaatar, I. Kostrikov, A. Szlam, and R. Fergus, "Intrinsic motivation and automatic curricula via asymmetric self-play," *CoRR*, vol. abs/1703.05407, 2017.

[29] S. Schaal, "The sl simulation and real-time control software package," 2009.

[30] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[31] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov, "Openai baselines," https://github.com/openai/baselines, 2017.

[32] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu, "Stable baselines," https://github.com/hill-a/stable-baselines, 2018.