Reverse Curriculum Generation for Robotic Manipulation with Reinforcement Learning

Carlos Florensa
Computer Science, UC Berkeley
florensa@berkeley.edu

David Held
Computer Science, UC Berkeley
davheld@berkeley.edu

Xinyang Geng
Computer Science, UC Berkeley
young.geng@berkeley.edu

Markus Wulfmeier
Department of Engineering Science
University of Oxford
markus@robots.ox.ac.uk

Pieter Abbeel
OpenAI
Computer Science, UC Berkeley
International Computer Science Institute (ICSI)
pabbeel@berkeley.edu

Abstract: Many robotics tasks require a robot to manipulate objects into a desired configuration. For example, we might want a robot to align a gear onto an axle or turn a key into a lock. These tasks present considerable difficulties for reinforcement learning approaches, since the natural reward function for these tasks is sparse and prohibitive amounts of exploration are required to receive it. Past approaches tackle these problems by manually designing a task-specific reward shaping function to help guide the learning. We propose a method to learn these tasks without requiring any prior task knowledge other than obtaining a single state in which the task is achieved. The robot is trained "backwards," gradually learning to reach the goal from a set of starting positions increasingly far from the goal. Our method automatically generates a curriculum of starting positions that adapts to the agent's performance, leading to efficient training on such tasks. We demonstrate our approach on difficult fine-grained manipulation tasks not solvable by state of the art reinforcement learning methods.

Keywords: Reinforcement Learning, Robotic Manipulation, Automatic Curriculum Generation

1 Introduction

Reinforcement Learning (RL) is a powerful learning technique for training an agent to optimize a reward function. Reinforcement learning has been demonstrated on complex tasks such as locomotion [1], Atari games [2], racing games [3], and other robotic manipulation tasks [4].

However, there are many tasks for which the reward function is either sparse or non-convex, which can create difficulties for learning-based approaches. For example, suppose a robot needs to learn to put a ring onto a peg, using a reward function based on the distance between the ring and the bottom of the peg. Such a task will be difficult to solve directly with reinforcement learning; the agent will likely learn a locally optimal policy that places the ring next to the bottom of the peg, and the agent will never learn that it needs to first lift the ring over the top of the peg. There are many such tasks in which the reward function is either sparse or non-convex, leading to difficulties for reinforcement learning.

One possible approach to such tasks is to engineer a more complex reward function that can help guide the policy towards the correct solution. Such "reward engineering" can be achieved using

reward shaping [5]. However, designing an efficient shaping function is often time-consuming and requires human effort and experimentation to find the correct shaping function for each task.

We avoid "reward engineering" by using a key insight: many robotics tasks are easier to undo than they are to perform directly. For example, in the above example, if a ring is already on a peg, it is simple to pull the ring off of the peg, even though it was hard to discover how to place the ring onto the peg directly. For such tasks, the set of states that achieve the task is relatively small, so they are hard to discover directly. However, for an agent placed into this set of successful states, it can easily move out of this set to other states in the environment. Many fine-grained manipulation tasks have this property.

We take advantage of this insight to develop a "backwards learning" approach for solving such difficult manipulation tasks. We first place the robot into a state that successfully achieves the task. Then, the robot learns "backwards" - it moves itself out of this state, and then discovers how to return to the goal state. Initially, the policy can only reach the goal from states that are nearby, but gradually, the policy learns how to achieve the task from increasingly distant initial positions. This method of learning backwards is reminiscent of dynamic programming, in which a problem is divided into subproblems that are sometimes solved in reverse order.

In this paper, we present an efficient and principled framework for performing such "reverse learning." Our method automatically generates a curriculum of initial positions for achieving the task. This curriculum is learned automatically by observing the performance of our agent at each step of the training process. Our method requires no prior knowledge of the task other than providing a single successful state that achieves the task.

The contributions of this paper include:

- A novel problem definition of finding the optimal start-state distribution for solving tasks with sparse reward functions
- A novel and practical approach for computing an initial state distribution that varies over the course of training, leading to an automatic curriculum of initial state distributions
- A demonstration that such an approach works to solve difficult tasks for fine-grained robotic manipulation

2 Related Work

Curriculum-based approaches with manually designed schedules have been explored in supervised learning [6, 7, 8, 9] to split particularly complex tasks into smaller, easier-to-solve sub-problems. One particular type of curriculum learning explicitly enables the learner to reject examples which it currently considers too hard [10, 11]. This type of adaptive curriculum has mainly been applied for supervised tasks and most curriculum approaches in reinforcement learning still rely on pre-specified task sequences [12].

In contrast, we automatically create a curriculum by discovering which tasks are easy or difficult for a given policy by inverting the problem and training from successively harder initial positions. Our approach is similar to work done by Tedrake et al. [13], which sequentially composes locally stabilizing controllers by growing a tree of stabilized trajectories backwards from the goal state. This can be viewed as a "funnel" which takes initial states to the goal state via a series of locally valid policies [14].

Other recent work has proposed using a given baseline performance for several tasks to gauge which tasks are the hardest and require more training [15]. However, this framework can only handle finite sets of tasks and requires dense rewards. Our method trains a policy that generalizes to a set of continuously parameterized tasks and is shown to perform well even under sparse rewards by not allocating training effort to tasks that are too hard for the current performance of the agent.

Finally, an interesting asymmetric self-play strategy has recently been proposed that is concurrent to our work [16]. Contrary to our approach,which is designed to generate multiple training tasks at the right level of difficulty, the asymmetric component of their method can lead to biased exploration [16]. Furthermore, the approach is designed as an exploration bonus for a single target task; in contrast, we define a new problem of efficiently optimizing a policy across a range of starting positions.

The idea of directly influencing state distributions to accelerate training has been addressed in the past. Kakade and Langford [17] introduce the idea of a 'restart distribution' from which new states are drawn, which when chosen well can optimize for faster policy convergence. A similar idea by Kearns et al. [18] explores the of using a 'generative model' - a type of simulator - which enables computing new states. Our work is in similar spirit of finding a distribution for initial states that accelerates training progress, which is done by finding states from which the current policy has a chance of directly solving the task but does not always succeed.

In addition, the idea was explored in the field of motion planning with approaches to extend planning to compute bidirectional search trees and grow an additional search tree from the goal position has been successfully applied in sampling-based planning literature to address high-dimensional configuration spaces and cluttered environments [19, 20, 21]. In particular, the approach has been shown to be beneficial given environments with non-convex guiding heuristics [21]. Initial position generation relates to these approaches as we address similar challenges with non-convexity in the dense rewards or even the limitation to sparse feedback. However, our goal is determining a general policy to reach the goal from all possible initial positions instead of one particular start-end point configuration without access to the kinematics or dynamics model of the environment.

3 Problem Definition

We consider the general problem of learning a policy that will move a system into a specified goal-space, from any starting state sampled from a given distribution. In this section we first briefly introduce the general reinforcement learning framework and then we will formally define our problem statement.

3.1 Preliminaries

We define a discrete-time finite-horizon discounted Markov decision process (MDP) by a tuple $M = (\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho_0, T)$, in which $\mathcal{S}$ is a state set, $\mathcal{A}$ an action set, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}_+$ is a transition probability distribution, $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is a bounded reward function, $\rho_0 : \mathcal{S} \to \mathbb{R}_+$ is an initial state distribution, and T is the horizon. Our aim is to learn a stochastic policy $\pi_\theta : \mathcal{S} \times \mathcal{A} \to \mathbb{R}_+$ parametrized by θ. The objective is to maximize its expected return, $\eta_{\rho_0}(\pi_\theta) = \mathbb{E}_{s_0 \sim \rho_0} R(\pi, s_0)$ with the expected reward from s_0 being $R(\pi, s_0) := \mathbb{E}_{\tau|s_0}[\sum_{t=0}^{T} r(s_t, a_t)]$, where $\tau = (s_0, a_0, \ldots)$ denotes the whole trajectory, $a_t \sim \pi_\theta(a_t|s_t)$, and $s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_t)$. Reinforcement Learning tackles this problem with methods like policy search, in which trajectories are iteratively collected and used to improve the current policy [22, 23, 24].

3.2 Goal-oriented tasks

We consider the general problem of reaching a certain goal space $S^g \subset \mathcal{S}$ from any starting position in $S^0 \subset \mathcal{S}$. This simple, high-level description can be translated into an MDP without further domain knowledge by using a binary reward function $r(s_t) = \mathbb{1}\{s_t \in S^g\}$ and a uniform distribution over the initial states $\rho_0 = \text{Unif}(S^0)$. We terminate the episode when the goal is reached. This implies that the return $R(\pi, s_0)$ associated to every starting state s_0 is the probability of reaching the goal at some time-step $t \in 0 \ldots T$.

$$R(\pi, s_0) = \mathbb{E}_{\pi(\cdot|s_t)} \mathbb{1}\Big\{ \bigcup_{t=0}^{T} s_t \in S^g | s_0 \Big\} = \mathbb{P}\Big(\bigcup_{t=0}^{T} s_t \in S^g \;\Big|\; \pi, s_0 \Big) \tag{1}$$

As advocated by Rajeswaran et al. [25], it is important to be able to train an agent to achieve the goal from a large set of initial states S^0. An agent trained in this way would be much more robust than an agent that is trained from just a single starting position, as it could recover from undesired deviations of the intended trajectory. Therefore, we will choose our set of initial states S^0 to be a wide area around the goal. For example, in the robotic fine-grained manipulation tasks considered in this paper, we want to insert a key into a key-hole or a ring onto a peg, regardless of the initial configuration of the robot or object (within a reasonable radius around the goal).

Our goal space for these tasks will be defined to be a small set of states around the desired configuration (e.g. key in the key-hole or ring on the peg). As discussed above, the sparsity of this

reward function makes learning difficult. If a naive dense reward function is defined for these tasks (such as measuring the distance between the ring and the bottom of the peg), this reward function will likely lead to a locally optimal learned policy, such as learning to push the ring against the side of the peg. To make those methods work, reward shaping [5] could be used, but it is difficult and time-consuming to engineer an effectively shaped reward function for each task. In the following subsection we introduce three assumptions, and the rest of the paper describes how we can leverage these assumptions to efficiently learn to achieve complex goal-oriented tasks directly from sparse reward functions.

3.3 Adapting the initial state distribution

In this work we study how to exploit three assumptions that hold true in a wide range of practical learning problems (especially if learned in simulation):

Assumption 1 *We can arbitrarily reset the agent into any start state $s_0 \in S^0$, allowing us to choose the initial state of any trajectory.*

Assumption 2 *At least one state s^g is provided such that $s^g \in S^g$.*

Assumption 3 *The Markov Chain induced by taking uniformly sampled random actions needs to have a communication class including all starting states S^0 and the given goal state s^g (there is a non-zero probability of traversing between all states in this Markov chain).*

The first assumption has been considered previously (e.g. access to a generative model in Kearns et al. [18]) and is deemed to be a considerably weaker assumption than having access to the full transition model of the MDP. For the second assumption, note that we only assume access to one state s^g in the goal region; we do not require a description of the full region nor trajectories leading to it. Finally, Assumption 3 ensures that the goal can be reached from any of the relevant starting positions, and that those starting positions can also be reached from the goal; this assumption is satisfied by many robotic problems of interest, as long as there are no major irreversibilities in the system.

The use of Assumption 1 to improve the learning in MDPs that require large exploration has already been demonstrated by Kakade and Langford [17]. Nevertheless, they do not propose a concrete procedure to choose a distribution ρ from which to sample the initial states in order to maximally improve on the objective in (1). In the next sections we introduce a method to continuously adapt the initial state distribution to the current performance of the policy. This approach can be understood as automatically building a curriculum of initial state distributions. We demonstrate the value of this method for challenging robotic manipulation tasks.

4 Methodology

In a wide range of goal-oriented RL problems, reaching the goal from an overwhelming majority of starting states in S^0 requires a prohibitive amount of on-policy or undirected exploration. On the other hand, it is usually easy for the learning agent (i.e. our current policy π_i) to reach the goal S^g from states nearby a goal state s^g. Therefore, learning from these states will be fast because the agent will perceive a strong signal, even under the indicator reward introduced in Section 3.2. Once the agent knows how to reach the goal from these nearby states, it can train from even further states and bootstrap its already acquired knowledge. This backwards expansion is inspired from classical RL methods like Value Iteration or Policy Iteration [26], although in our case we do not assume knowledge of the transition model and our environments have high-dimensional continuous action-state spaces. In the following subsections we propose a method that leverages the Assumptions from the previous section and the above reasoning to automatically adapt the starting state distribution, generating a curriculum of starting state distributions that can be used to tackle problems unsolvable by standard RL methods.

4.1 Policy Optimization with modified start state distribution

Policy gradient strategies are well suited for robotic tasks with continuous and high dimensional action-spaces [27]. In policy gradient algorithms, at every iteration i the policy is improved based

on the reward collected by executing the current policy π_i. Nevertheless, applying policy gradients directly on the original MDP does very poorly in our challenging manipulation tasks because the goal might never be reached from the starting positions in S^0. In such cases, no reward will be received, and the policy will not improve. Therefore, we propose to adapt a distribution ρ_i from which starting states s_0 are sampled to train policy π_i. Based on the observation above, to maximize the learning signal we should expand this distribution backwards from the goal, based on the current performance of the policy π_i.

More concretely and analogously to Held et al. [28], we postulate that in goal-oriented environments, a strong learning signal is obtained when training from initial states $s_0 \sim \rho_i$ from which the agent reaches the goal sometimes, but not always. We call these starting states "good" states. More formally, at training iteration i, we would like to sample from $\rho_i = \text{Unif}(S_i^0)$ where $S_i^0 = \{s_0 : R_{\min} < R(\pi_i, s_0) < R_{\max}\}$. The hyper-parameters $R_{\min}$ and $R_{\max}$ are easy to tune due to their interpretation as bounds on the probability of success, derived from Eq. (1). Unfortunately, sampling uniformly from S_i^0 is intractable because no closed-form description of S_i^0 is provided. Nevertheless, at least at the beginning of training, states nearby a goal state s_g (provided by Assumption 2) are more likely to be in S_i^0. Then, after some iterations of training from those starting states, some will be completely mastered (i.e. they will no longer be in S_{i+1}^0), but others will still need more training. To find more "good" states, we follow the same reasoning: the states nearby these remaining $s \in S_{i+1}^0$ are likely to also be in S_{i+1}^0.

Algorithm 1: Policy Training

Input : Policy π_0, goal state $s^g \in S^g$, ρ_0
Output: Policy π_N
$starts_{\text{old}} \leftarrow [s_g]$;
$starts$, $rews \leftarrow [s_g]$, [1];
for $i \leftarrow 1$ **to** $N-1$ **do**
  $starts \leftarrow$ `SampleNearby`$(starts, m_{\text{new}})$;
  $starts$.append[sample$(starts_{\text{old}}, m_{\text{old}})$];
  $\rho_i \leftarrow$ Unif$(starts)$;
  $\pi_{i,+1}$, $rews \leftarrow$ `train_pol`(ρ_i, π_{i-1});
  $starts \leftarrow$ `select`$(starts, rews)$;
  $starts_{\text{old}}$.append[$starts$];
  `evaluate`(π_{i+1}, ρ_0);
end

Our generic algorithm that implements this procedure is detailed in Algorithm 1. `train_pol` applies the RL algorithm of choice; in our case we use 5 iterations of TRPO [29]. `select`$(starts, rews)$ selects a subset of the previously used start states, based on the rewards $rews$ obtained by the policy π_{i+1} . Technically, to check which of the states $s_0 \in starts$ is in S_i^0 we should execute some trajectories from each of those states to estimate $R(s_0, \pi_{i+1})$, but this considerably increases the sample complexity. Instead, we use the rollouts collected during `train_pol`, which is found to give a good enough estimate and not drastically decrease learning performance of the overall algorithm. Our candidate for the `SampleNearby` procedure is described in the next subsection. We also maintain a replay buffer $starts_{\text{old}}$ of previously considered start states from which we uniformly sample m_{old} candidate start states at every iteration. As already shown by Held et al. [28], this is an important feature to avoid catastrophic forgetting.

4.2 Sampling "nearby" feasible states

Procedure 2: `SampleNearby`

Input : $starts$, covariance Σ, T_B, M, m_{new}
Output: $starts_{\text{new}}$
while len$(starts) < M$ **do**
  $s_0 \sim \text{Unif}(starts)$;
  for $t \leftarrow 1$ **to** T_B **do**
    $a_{t+1} = a_t + \epsilon_t$ with $\epsilon_t \sim \mathcal{N}(0, \Sigma)$;
    $s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_t)$;
    $starts$.append(s_{t+1});
  end
end
$starts_{\text{new}} \leftarrow$ sample$(starts, m_{\text{new}})$

For robotic manipulation tasks with complex contacts and constraints, applying noise in state-space $s' = s + \epsilon$, $\epsilon \sim \mathcal{N}$ may yield many infeasible start positions s'. For example, even small random perturbations of the joint angles of a seven degree-of-freedom arm generate large modifications to the end-effector position, potentially placing it in an infeasible state that intersects with surrounding objects. For this reason, the concept of "nearby" states might be unrelated to the Euclidean distance $\|s' - s\|^2$ between these states. Instead, we have to understand proximity in terms of how likely it is to reach one state from the other by taking actions in the MDP.

Therefore, we choose to generate new states s' from a certain seed state s by applying noise in action space. This means we use Assumption 1 to reset the system to state s, and from there we execute short "Brownian motion" rollouts of horizon T_B taking actions $a_{t+1} = a_t + \epsilon_t$ with $\epsilon_t \sim \mathcal{N}(0, \Sigma)$. This method of generating "nearby" states is detailed in Procedure 2. The total sampled states M should be large enough such that the m_{new} desired states $starts_{\text{new}}$, obtained by subsampling, extend in all directions around the input states $starts$. All states visited during the rollouts are guaranteed to be feasible and can then be used as starting states to keep training the policy.

The general method described above keeps expanding the region of the state-space from which the policy can reach the goal reliably, sampling more heavily nearby the starting states that need more training to be mastered and avoiding start states that are yet too far to receive any reward under the current policy. Now, thanks to Assumption 3, the Brownian motion used to generate further and further starting states will eventually reach all starting states in S^0, and therefore our method improves the metric η_{ρ_0}.

5 Experimental Results

We investigate the following questions in our experiments:

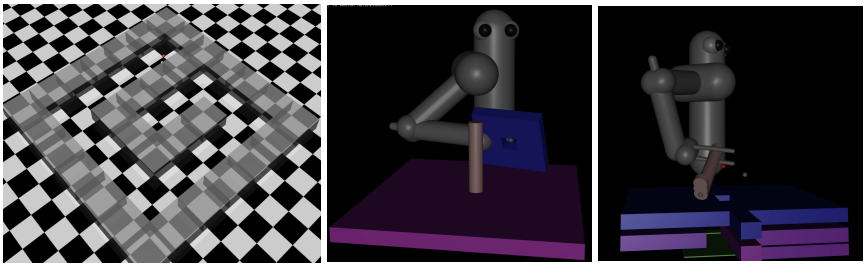
- Does the performance of the policy on the target initial state distribution ρ_0 improve by training on distributions ρ_i growing from the goal?
- Does focusing the training on "good starts" speed up learning?
- Is Brownian motion a good way to generate "good starts" from previous "good starts"?

We use the below task settings to explore these questions. The hyperparameters used in our experiments are described in the appendix.

Navigation in a maze: (Fig. 1a) A point-mass agent (in orange) must navigate to the goal position $(4, 4)$ at the end of a G-shaped maze (in red). The target initial state distribution from which we seek to reach the goal is uniform over all feasible (x, y) positions in the maze.

Ring on Peg: (Fig. 1b) A 7 DOF robot must learn to place a "ring" (actually a square disk with a hole in the middle) on top of a tight-fitting round peg. The task is complete when the ring is within 3 cm of the bottom of the 15 cm tall peg. The target initial state distribution from which we seek to reach the goal is uniform over all feasible joint positions for which the center of the ring is within 40 cm of the bottom of the peg.

Key in hole: (Fig. 1c) A 7 DOF robot must learn to insert a key into a key-hole. The task is completed when the distance between three reference points at the extremities of the key and its corresponding targets is below 3cm. In order to reach the target, the robot must first insert the key at a specific orientation, then rotate it 90 degrees clockwise, push forward, then rotate 90 degrees counterclockwise. The target initial state distribution from which we seek to reach the goal is uniform over all feasible joint positions such that the tip of the key is within 40 cm of key-hole.



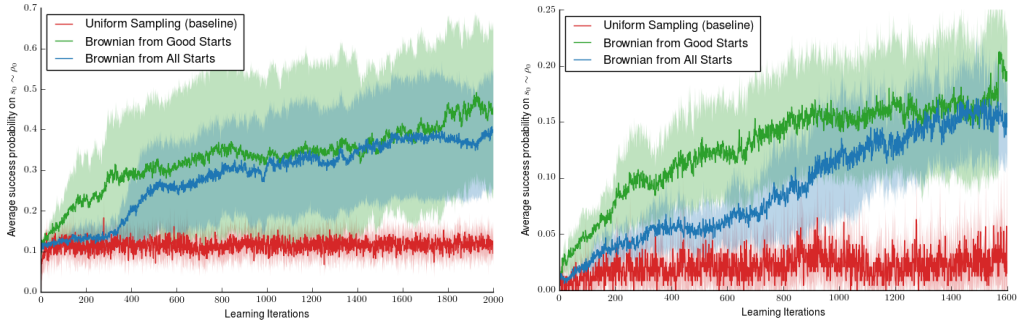
(a) Point-mass maze task (b) Ring on Peg task (c) Key insertion task

Figure 1: Task images. Videos of the final performance obtained by our algorithm are available in the website of the project: https://sites.google.com/view/startgeneration

5.1 Improve learning by modifying the initial state distribution

Trying to tackle the original MDP directly yields very poor success. In Figure 2, the *Uniform Sampling (baseline)* red curves show the average return of policies learned with TRPO without

modifying the initial state distribution. We see that the final average probability of reaching the goal is around 2% for the key task and 10% for the ring task. These success probabilities correspond to reliably reaching the goal only from very nearby positions: when the ring is already on the peg or when the key is initialized very close to the final position. None of the learned policies can reliably reach the goal from more distant starting positions. On the other hand, the two other learning curves presented correspond to methods that modify the initial state distribution on which the policy is trained. The training algorithm itself (as well as its hyperparameters) and the evaluation metric are the same.



(a) Ring on Peg task

(b) Key insertion task

Figure 2: Learning curves for the robotics tasks (mean and standard deviation over 5 random seeds).

In the case of the maze navigation task, we also observe that applying TRPO directly on the original MDP may perform very poorly. Indeed, the *Unform Sampling* red learning curve in Figure 3 has a very high variance because some policies only learn how to perform well from one side of the goal. A more thorough analysis of this behavior is provided in the Appendix. Our methods have a more reliable learning. We conclude that training on a different initial state distribution ρ_i can improve training on the original MDP with a fixed initial state distribution ρ_0.

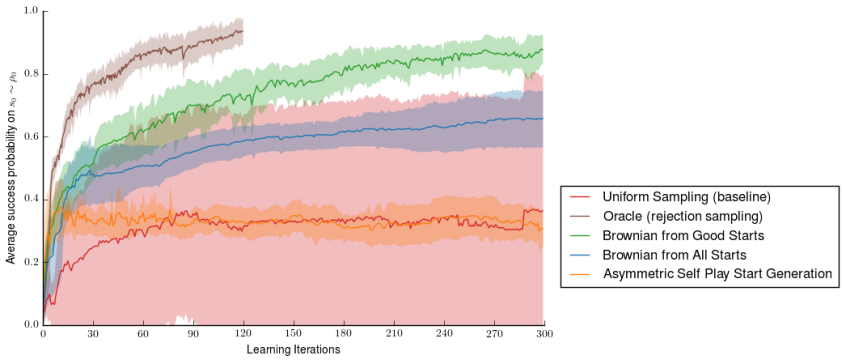


Figure 3: Learning curves for the maze tasks (mean and standard deviation over 5 random seeds).

5.2 Improve learning by training on "good" starts

In Figures 2 and 3 we see how applying our Algorithm 1 to modify the initial state distribution considerably improves learning (*Brownian on Good States*, in green) and final performance on the original MDP. Two elements are involved in this improvement: first, the backwards expansion from the goal, and second, the concentration of training efforts on "good" starts. To test the relevance of this second element, we ablate our method by running our `SampleNearby` Procedure 2 on all states from which the policy was trained in the previous iteration. In other words, the `select` function in Algorithm 1 is replaced by the identity, returning all *starts* independently of the rewards *rews* they obtained during the last training iteration. The resulting algorithm performance is shown as the *Brownian from All Starts* blue curve in Figures 2 and 3. As expected, this method is still better than not modifying the initial state distribution but has a slower learning than running `SampleNearby` around the estimated good starts.

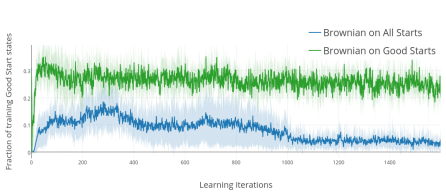
As mentioned in Sec. 4, we would ideally like to sample initial states from $\rho_i = \text{Unif}(S_i^0)$ where $S_i^0 = \{s_0 : R_{\min} < R(\pi_i, s_0) < R_{\max}\}$. Unfortunately, sampling directly from S_i^0 is infeasible because no closed-form description of S_i^0 is provided. Instead, we evaluate states in S_{i-1}^0, and we use Brownian motion to find nearby states, to approximate S_i^0. However, these states form only a subset of all states in S_i^0.

We can evaluate the amount of error in this approximation by exhaustive sampling states in the lower dimensional maze task. To do so, at every iteration we can samples states s_0 uniformly from the state-space $\mathcal{S}$, empirically estimates their return $R(s_0, \pi_i)$, and rejects the ones that are not in the set $S_i^0 = \{s_0 : R_{\min} < R(\pi_i, s_0) < R_{\max}\}$. This exhaustive sampling method is an order of magnitude more expensive in terms of sample complexity, so it would not be of practical use; however, we can see the results of such an approach in Fig. 3, called "Oracle (rejection sampling)"; training on states sampled in such a manner improves further the learning rate and final performance. Thus we can see that our approximation of using states in S_{i-1}^0 to find states in S_i^0 leads to some loss in performance, at the benefit of a greatly reduced computation time.

Finally, we compare to another way of generating start positions based on the asymmetric self-play method of Sukhbaatar et al. [30]. The basic idea is to train another policy, "Alice" that proposes start positions to the learning policy, "Bob". As can be seen, this method performs very poorly in the maze task, and our investigation shows that "Alice" often gets stuck in a local optimum, leading to poor start states suggestions for "Bob". A commonly observed failure case is "Bob" suddenly learning how to reach the goal from a large part of the state-space, making it very hard for Alice to find new positions that are hard for Bob. In the original paper, the method was demonstrated only on discrete action spaces, in which a multi-modal distribution for Alice can be maintained; even in such settings, the authors observed that Alice can easily get stuck in local optima. This problem is exacerbated when moving to continuous action spaces defined by a unimodal Gaussian distribution.

5.3 Brownian motion to generate "nearby" states

Here we evaluate if running our Procedure 2 `SampleNearby` around "good" starts yields more good starts than running `SampleNearby` from all previously visited states. This can clearly be seen in Fig. 4a for the ring task. We also observe in Fig. 4b that running `SampleNearby` on all previous start states generates more starting states with no reward. This is the cause of the decrease in performance of this approach shown in Figures 2 and 3.



(a) Fraction of Good Starts used during training

(b) Fraction of Low Reward Starts used during training

Figure 4: Fraction of Good Starts and Low Reward Starts generated

6 Conclusions and Future Directions

We propose a method to automatically adapt the initial state distribution on which an agent is trained, such that the performance on the original problem is efficiently optimized. We leverage three assumptions commonly satisfied in simulated tasks to tackle hard goal-oriented problems that state of the art RL methods cannot solve.

A limitation of the current approach is that it generates start states that grow from a single goal uniformly outwards, until they cover the original initial state distribution $\text{Unif}(S^0)$. Nevertheless, if the target set of initial states S^0 is far from the goal and we have some prior knowledge, it would be interesting to bias the generated start distributions ρ_i towards the desired start distribution. A promising future line of work is to combine the present automatic curriculum based on start state generation with goal generation [28], similar to classical results in planning [31].

It can be observed in the videos of our final policy for the manipulation tasks that the agent has learned to *exploit* the contacts instead of avoiding them. Therefore, the learning based aspect of the presented method has a huge potential to tackle problems that classical motion planning algorithms could struggle with, such as environments with non-rigid objects or with uncertainties in the the task geometric parameters. We also leave as future work to combine our curriculum-generation approach with methods like domain randomization [32] to obtain policies that are transferable to the real world.

Acknowledgments

Carlos was founded by the La Caixa Fellowship. Markus -¿ acknowledge the support of the UKs Engineering and Physical Sciences Research Council (EPSRC) through the Doctoral Training Award (DTA) as well as the support of the Hans-Lenze-Foundation.

References

[1] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *ICLR*, 2016.

[2] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[4] S. Levine, C. Finn, T. Darrell, and P. Abbeel. End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17(39):1–40, 2016.

[5] A. Y. Ng, D. Harada, and S. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pages 278–287, 1999.

[6] Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48. ACM, 2009.

[7] W. Zaremba and I. Sutskever. Learning to execute. *CoRR*, abs/1410.4615, 2014.

[8] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Advances in Neural Information Processing Systems 28*, pages 1171–1179. Curran Associates, Inc., 2015.

[9] A. Graves, M. G. Bellemare, J. Menick, R. Munos, and K. Kavukcuoglu. Automated Curriculum Learning for Neural Networks. 2017. URL https://arxiv.org/pdf/1704.03003.pdfhttp://arxiv.org/abs/1704.03003.

[10] M. P. Kumar, B. Packer, and D. Koller. Self-paced learning for latent variable models. In *Advances in Neural Information Processing Systems 23*, pages 1189–1197. 2010.

[11] L. Jiang, D. Meng, Q. Zhao, S. Shan, and A. G. Hauptmann. Self-paced curriculum learning. In *AAAI*, volume 2, page 6, 2015.

[12] A. Karpathy and M. Van De Panne. Curriculum learning for motor skills. In *Canadian Conference on Artificial Intelligence*, pages 325–330. Springer, 2012.

[13] R. Tedrake, I. R. Manchester, M. Tobenkin, and J. W. Roberts. Lqr-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research*, 29 (8):1038–1052, 2010.

[14] R. R. Burridge, A. A. Rizzi, and D. E. Koditschek. Sequential composition of dynamically dexterous robot behaviors. *The International Journal of Robotics Research*, 18(6):534–555, 1999.

[15] S. Sharma and B. Ravindran. Online Multi-Task Learning Using Biased Sampling. 2017.

[16] S. Sukhbaatar, I. Kostrikov, A. Szlam, and R. Fergus. Intrinsic Motivation and Automatic Curricula via Asymmetric Self-Play. 2017.

[17] S. Kakade and J. Langford. Approximately Optimal Approximate Reinforcement Learning. *International Conference in Machine Learning*, 2002. URL http://www.cs.cmu.edu/~./jcl/papers/aoarl/Final.pdf.

[18] M. Kearns, Y. Mansour, and A. Y. Ng. A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes. *Machine Learning*, 49(2/3):193–208, 2002. ISSN 08856125. doi:10.1023/A:1017932429737. URL http://link.springer.com/10.1023/A:1017932429737.

[19] I. Pohl. *Bi-directional and heuristic search in path problems*. PhD thesis, Department of Computer Science, Stanford University, 1969.

[20] D. Hsu, J.-C. Latombe, and R. Motwani. Path planning in expansive configuration spaces. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 3, pages 2719–2726. IEEE, 1997.

[21] J. J. Kuffner and S. M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 2, pages 995–1001. IEEE, 2000.

[22] D. P. Bertsekas, D. P. Bertsekas, D. P. Bertsekas, and D. P. Bertsekas. *Dynamic programming and optimal control*, volume 1. Athena scientific Belmont, MA, 1995.

[23] J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural networks*, 21(4):682–697, 2008.

[24] I. Szita and A. Lörincz. Learning tetris using the noisy cross-entropy method. *Learning*, 18 (12), 2006.

[25] A. Rajeswaran, K. Lowrey, E. Todorov, and S. Kakade. Towards generalization and simplicity in continuous control. abs/1703.02660, 2017. URL http://arxiv.org/abs/1703.02660.

[26] R. S. Sutton and A. G. Barto. Reinforcement learning: An introduction. 1998.

[27] M. P. Deisenroth, G. Neumann, J. Peters, et al. A survey on policy search for robotics. *Foundations and Trends in Robotics*, 2(1–2):1–142, 2013.

[28] D. Held, X. Geng, C. Florensa, and P. Abbeel. Automatic goal generation for reinforcement learning agents. abs/1705.06366, 2017. URL http://arxiv.org/abs/1705.06366.

[29] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust region policy optimization. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 1889–1897, 2015.

[30] S. Sukhbaatar, I. Kostrikov, A. Szlam, and R. Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.

[31] J. Kuffner and S. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, pages 995–1001. IEEE. ISBN 0-7803-5886-4. doi:10.1109/ROBOT.2000.844730. URL http://ieeexplore.ieee.org/document/844730/.

[32] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. mar 2017. URL http://arxiv.org/abs/1703.06907.

[33] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel. Benchmarking deep reinforcement learning for continuous control. *International Conference on Machine Learning (ICML)*, 2016.

A Experiment Implementation Details

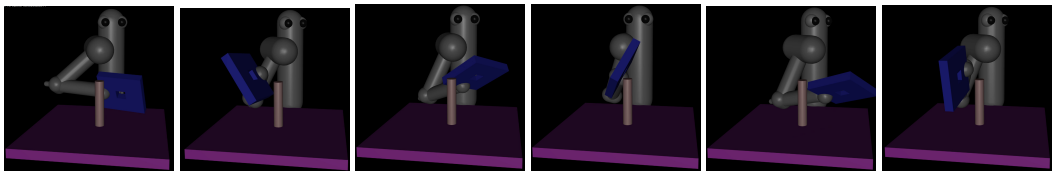
A.1 Hyperparameters

Below we describe the hyperparemeters used for our method. Each iteration, we generate new start states, (as described in Section 4.2 and Procedure 2), which we append to the "seed states" until we have a total of $M = 10000$ start states. We then subsample these down to $m_{new} = 200$ new start states. These are appended with 100 sampled old start states (as described in Section 4.1 and Procedure 1), and these states are used to initialize our agent when we train our policy. The "Brownian motion" rollouts have a horizon of T_B = 50 timesteps, and are obtained by taking random actions sampled from a standard normal distribution (e.g. a 0-mean Guassian with a variance of 1 in each action dimension).

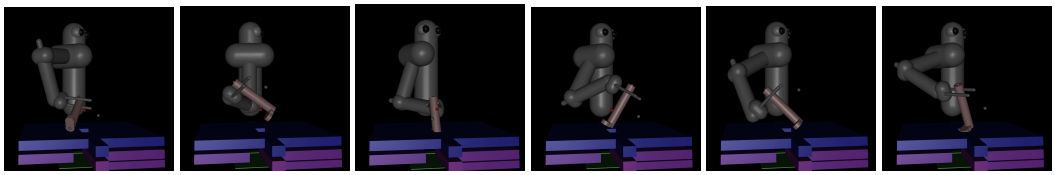
For our method as well as the baselines, we train the policy with TRPO [29], implemented with rllab [33]. We use a TRPO step-size of 0.01 and a linear baseline. For all tasks, we train with a batch size of 20,000 timesteps. All experiments use a maximum horizon of $T = 500$ timesteps, and the episode ends as soon as the agent reaches a goal state. We define the goal set S^g to be a ball around the goal state, in which the ball has a radius of 0.03 m for the ring and key tasks and 0.3 m for the maze task. In our definition of S_i^0, we use $R_{\min} = 0.1$ and $R_{\max} = 0.9$. We use a discount factor $\gamma = 0.995$ for the optimization, in order to encourage the policy to reach the goal as fast as possible.

A.2 Performance metric

The tasks are described as being able to reach the specified goal from any feasible starting position within a certain radius of that goal: $s_0 \in S^0$. For the maze navigation task this is straight forward as the designer gives a concrete description of the feasible (x, y) space, so we can directly sample from it. Nevertheless, it is not trivial to uniformly sample from such feasible starting positions for the robotics tasks. In particular, the state space is in joints angles and angular velocities of the 7 DOF arm, but the physical constraints of these contact-rich environments are given by the geometries of the task. Therefore, uniformly sampling from the angular bounds mostly yields infeasible states, with some part of the arm or the end-effector intersecting with other objects in the scene. In order to approximate uniformly sampling from S^0, we use Assumption 2, giving a feasible goal state s^g. We simply run our `SampleNearby` procedure initialized with $starts = [s_g]$ with a very large M and long time horizons T_B. This large aggregated state data-set is saved and samples from it are used to evaluate the performance of our algorithm. Figures 5a and 5b show six sampled starting states from the data sets used to evaluate the ring task and the key task. These data sets are too large to include in the supplementary material but will be available upon publication for future reproducibility and benchmarking.



(a) Uniformly sampled start positions for ring task. There are 39,530 states in the data-set, of which 5,660 have the ring with its hole already in the peg



(b) Uniformly sampled start positions for key task. There are 544,575 states in the data-set, of which 120,784 have the key somewhere inside the key-hole

Figure 5: Samples from the test distribution for the manipulation tasks

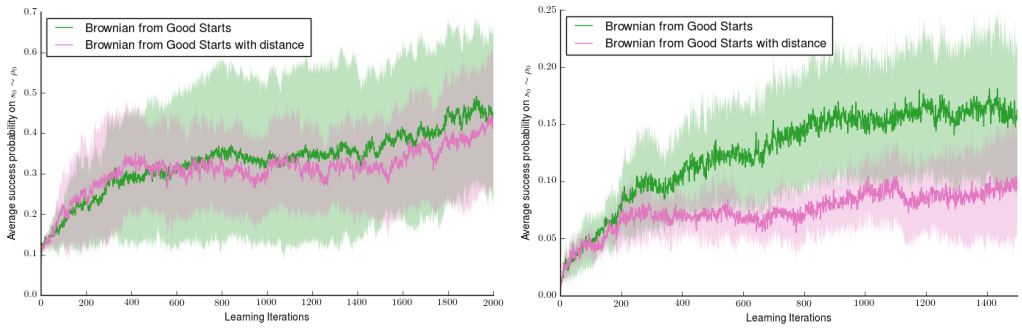
Given the quasi-static nature of the tasks considered, we generate only initial angle positions, and we set all initial velocities to zero. Generating initial velocities is a fairly simple extension of our approach that we leave for future work.

B Other methods

B.1 Distance reward shaping

Although our policies are trained with sparse rewards, the policy optimization steps can use any kind of reward shaping available. In the the robotics tasks considered in this paper, the goal is defined in terms of a reference state, and hence it seems natural to try to use the distance to this state as a reward. However, we have found that this modification does not actually improve training. For the starting states near to the goal, the policy can learn to reach the goal simply from the indicator reward introduced in Section 3.2. For the states that are further away, the distance to the goal is actually not a useful metric to guide the policy; hence, the distance reward actually guides the policy updates towards a suboptimal local optimum, leading to poor performance.

In Fig. 6 we see that the ring task is not much affected by the additional reward, whereas the key task suffers considerably if this reward is added.



(a) Ring on Peg task

(b) Key insertion task

Figure 6: Learning curves for the robotics manipulation tasks

B.2 Failure cases of Uniform Sampling for maze navigation

In the case of the maze navigation task, we observe that applying TRPO directly on the original MDP incures in very high variance across learning curves. We have observed that some policies only learned how to perform well from a certain side of the goal. The reason for this is that our learning algorithm (TRPO) is a batch on-policy method; therefore, at the beginning of learning, uniformly sampling from the state-space might give a batch with very few trajectories that reach the goal and hence it is more likely that they all come from one side of the goal. In this case, the algorithm will update the policy to go in the same direction from everywhere, wrongly extrapolating from these very few successful trajectories it received. This is less likely to happen if the trajectories for the batch are collected with a different initial state distribution that concentrates more uniformly around the goal, as the better learning progress of the other curves show.