DisCo RL: Distribution-Conditioned Reinforcement Learning for General-Purpose Policies

Soroush Nasiriany

Electrical Engineering and Computer Sciences
University of California at Berkeley

Technical Report No. UCB/EECS-2020-151
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-151.html

August 13, 2020

DisCo RL: Distribution-Conditioned Reinforcement Learning for General-Purpose Policies

by Soroush Nasiriany

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

Committee:

Professor Sergey Levine
Research Advisor

8/11/2020

(Date)

* * * * * * *

Professor Pieter Abbeel
Second Reader

8/11/2020

(Date)

To my parents, for their unconditional support.

DisCo RL: Distribution-Conditioned Reinforcement Learning for General-Purpose Policies

Soroush Nasiriany*
University of California, Berkeley
snasiriany@berkeley.edu

Abstract: Reinforcement learning is focused on the problem of learning a near-optimal policy for a given task. But can we use reinforcement learning to instead learn general-purpose policies that can perform a wide range of different tasks, resulting in flexible and reusable skills? Contextual policies provide this capability in principle, but the representation of the context determines the degree of generalization and expressivity. Categorical contexts preclude generalization to entirely new tasks. Goal-conditioned policies may enable some generalization, but cannot capture all tasks that might be desired. In this paper, we propose goal distributions as a general and broadly applicable task representation suitable for contextual policies. Goal distributions are general in the sense that they can represent any state-based reward function when equipped with an appropriate distribution class, while the particular choice of distribution class allows us to trade off expressivity and learnability. We develop an off-policy algorithm called distribution-conditioned reinforcement learning (DisCo RL) to efficiently learn these policies. We evaluate DisCo RL on a variety of robot manipulation tasks and find that it significantly outperforms prior methods on tasks that require generalization to new goal distributions.

Keywords: reinforcement learning, multi-task learning

1 Introduction

Versatile, general-purpose robotic systems will require not only broad repertoires of behavioral skills, but also the faculties to quickly acquire new behaviors as demanded by their current situation and the needs of their users. Reinforcement learning in principle enables autonomous acquisition of such skills. However, each skill must be learned individually at considerable cost in time and effort. In this paper, we instead explore how general-purpose robotic policies can be acquired by conditioning policies on task representations. This question has previously been investigated by learning *goal-conditioned* or *universal* policies, which take in not only the current state, but also some representation of a *goal* state. However, such a task representation cannot capture many of the behaviors we might actually want a versatile robotic system to perform, since it can only represent behaviors that involve reaching individual states. For example, for a robot packing items into a box, the task is defined by the position of the items relative to the box, rather than their absolute locations in space, and therefore does not correspond to a single state configuration. How can we parameterize a more general class of behaviors, so as to make it possible to acquire truly general-purpose policies that, if conditioned appropriately, could perform any desired task?

To make it possible to learn general-purpose policies that can perform any task, we instead consider conditioning a policy on a full *distribution* over goal states. Rather than reaching a specific state, a policy must learn to reach states that have high likelihood under the provided distribution, which may specify various covariance relationships (e.g., as shown in Figure 1, that the position of the items should covary with the position of the box). In fact, we show that, because optimal policies are invariant to additive factors in reward functions, arbitrary goal distributions can represent *any* state-dependent reward function, and therefore any task. Choosing a specific distribution class provides a natural mechanism to control the expressivity of the policy. We may choose a small distribution class to narrow the range of tasks and make learning easier, or we may choose a large distribution class to expand the expressiveness of the policy.

*Please refer to Section 8 for acknowledgement of collaborators and their contributions

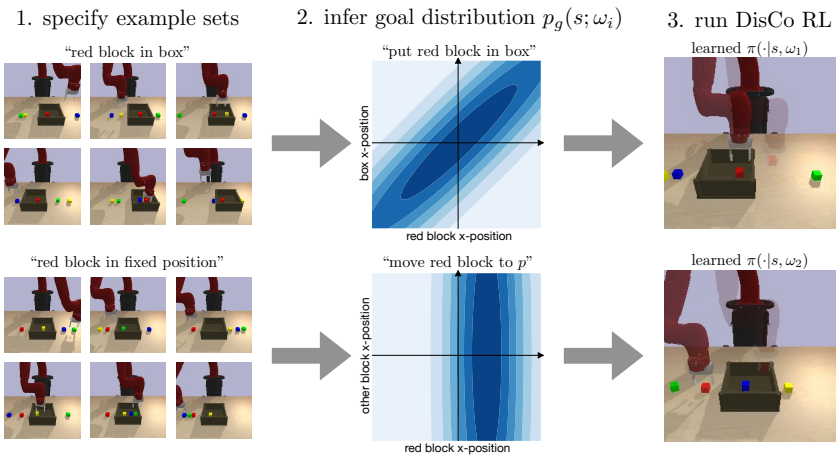


Figure 1: The distribution parameters ω can be inferred from data and then passed to a DisCo policy. Distribution-conditioned RL can express a broad range of tasks, from defining relationships between different state components (top) to simply ignoring a dimension (bottom).

We show that distribution-conditioned policies can be trained efficiently by sharing data from a variety of tasks and relabeling the goal distribution parameters based on the posterior likelihood of other goal distributions. In this way, the same data can be utilized to train a policy for a wide range of different distributions, each corresponding to a different reward. Lastly, while the distribution parameters can be provided manually to specify tasks, we also present a way to infer these distribution parameter directly from data.

The main contribution of this paper is DisCo RL, an algorithm for learning distribution-conditioned policies. To learn efficiently, DisCo RL uses off-policy training and a novel distribution relabeling scheme. We evaluate on robot manipulation tasks in which a single policy must solve multiple tasks that cannot be expressed as reaching different goal states. We find that conditioning the policies on goal distributions results in significantly faster learning than solving each task individually, enabling policies to acquire a broader range of tasks than goal-conditioned methods.

2 Related Work

In goal-conditioned RL, a policy is given a goal state, and must take actions to reach that state [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. However, as discussed previously, many tasks cannot be specified with a single goal state. To address this, many goal-conditioned methods manually design a goal space that explicitly excludes some state variables [11, 12, 13, 14, 15, 16], for example by only specifying the desired location of an object. This requires manual effort and user insight, and does not generalize to environments with high-dimensional state representations, such as images, where manually specifying a goal space is very difficult. With images, a number of methods learn latent representations for specifying goal states [17, 7, 9, 18, 19], which makes image-based goals more tractable, but does not address the representation issues discussed above. Our work on goal distributions can be seen as a generalization of goal state reaching. Reaching a single goal state g is equivalent to maximizing the likelihood of a delta-distribution centered at g, and ignoring some state dimensions is equivalent to maximizing the likelihood of a distribution that places uniform likelihood across the respective ignored state variables. But more generally, goal distributions capture the set of all reward functions, enabling policies to be conditioned on arbitrary tasks.

A number of prior methods learn rewards [20, 21] or policies [22, 23, 24, 25] using expert trajectories or observations. In this work, we also demonstrate that we can use observations to learn reward functions, but we have different objectives and assumptions as compared to prior work. Many of these prior methods require state sequences from expert demonstrations [22, 23, 24, 25], whereas our work only requires observations of successful outcomes to fit the goal distribution. Fu et al. [21] also only uses observations of successful outcomes to construct a reward function, but focuses on solving single tasks or goal-reaching tasks, whereas we study the more general setting where the policy is conditioned on a goal distribution.

Parametric representations of rewards have also been in the context of successor features [26, 27, 28, 29], which parameterize reward functions as linear combinations of known features. We present a general framework in which arbitrary rewards can be represented as goal distributions rather than feature weights, and also demonstrate that these goal distributions can be learned from data.

Prior work on state marginal matching [30] attempts to make a policy's stationary distribution match a target distribution to explore an environment. In our work, rather than matching a target distribution, we use the log-likelihood of a goal distributions to define a reward function, which we then maximize with standard reinforcement learning.

3 Background

Reinforcement learning (RL) frames reward maximization in a Markov decision process (MDP), defined by the tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, r, p, p_0, \gamma)$ [31], where $\mathcal{S}$ denotes the state space and $\mathcal{A}$ denotes the action space. In each episode, the agent's initial state $\mathbf{s}_0 \in \mathcal{S}$ is sampled from an initial state distribution $\mathbf{s}_0 \sim p_0(\mathbf{s}_0)$, the agent chooses an action $\mathbf{a} \in \mathcal{A}$ according to a stochastic policy $\mathbf{a}_t \sim \pi(\cdot|\mathbf{s}_t)$, and the next state is generated from the state transition dynamics $\mathbf{s}_{t+1} \sim p(\cdot \mid \mathbf{s}_t, \mathbf{a}_t)$. We will use τ to denote a trajectory sequence $(\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \dots)$ and denote sampling as $\tau \sim \pi$ since we assume a fixed initial state and dynamics distribution. The objective of an agent is to maximize the sum of discounted rewards, $\mathbb{E}_{\tau\sim\pi}\left[\sum_{t=1}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)\right]$.

Off-policy, temporal-difference algorithms. Our method can be used with any off-policy temporal-difference (TD) learning algorithm. TD-learning algorithms only need $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$ tuples to train a policy, where $\mathbf{a}_t$ is an action taken from state $\mathbf{s}_t$, and where r_t and $\mathbf{s}_{t+1}$ are the resulting reward and next state, respectively, sampled from the environment. Importantly, these tuples can be collected by any policy, making it an off-policy algorithm. These tuples are typically sampled from a replay buffer $\mathcal{R}$, which consists of tuples generated by all previous environment interactions.

Contextual MDPs. A contextual MDP augments an MDP with a context space $\mathcal{C}$ and context distribution p_c. At the start of each episode, a context is sampled from the context distribution $\mathbf{c} \sim p_c$, and both the reward and the policy are conditioned on this context for the entire episode: $r(\mathbf{s}_t, \mathbf{a}_t, \mathbf{c})$ and $\pi(\mathbf{a}_t \mid \mathbf{s}_t, \mathbf{c})$[2]. We note that contextual MDPs can always be reduced to a standard MDP, by creating an augmented state space that includes both the original state and context, given by $\mathcal{S}' = \mathcal{S} \times \mathcal{C}$. However, the concept of a contextual MDP is useful because it allows us to easily model a policy that must accomplish different tasks at each episode. For example, when the context is a goal that an agent must reach, we recover goal-conditioned reinforcement learning [1, 2]. When the context is a weight vector w that specifies a linear combination of cumulants for the reward, as in $r(s) = w^T\phi(s)$, we recover successor features [27].

The design of the context space and contextual reward function determines the behaviors that these policies can learn. For example, goal-conditioned policies are only trained to reach individual goal states. If we want to train general-purpose policies that can perform arbitrary tasks, is it possible to design a contextual MDP that captures the set of all rewards functions? In theory, the context c could be a reward function and the context space $\mathcal{C}$ could be the set of all reward functions. But in practice, it is unclear how to condition policies on reward functions, since most learning algorithms are not well suited to take functions as inputs. In the next section, we present a promising context space: the space of goal distributions.

4 Distribution-Conditioned Reinforcement Learning

In this section, we show how conditioning policies on a goal distribution results in a contextual MDP that can capture any set of reward functions. Each distribution represents a different reward function, and so choosing a distribution class provides a natural mechanism to choose the expressivity of the contextual policy. We then present distribution-conditioned reinforcement learning (DisCo RL), an off-policy algorithm for training policies conditioned on parametric representation of distributions, and discuss the specific representation that we use.

[2]Some work assumes the dynamics p depend on the context [32]. We assume that the dynamics are the same across all contexts, though our method could also be used with context-dependent dynamics.

4.1 Generality of Goal Distributions

We assume that the goal distribution is in a parametric family, with parameter space Ω, and consider a contextual MDP in which the context space is Ω. At the beginning of each episode, a parameter $\omega \in \Omega$ is sampled from some parameter distribution p_ω. The parameter ω defines the goal distribution $p_g(\mathbf{s}; \omega) : \mathcal{S} \mapsto \mathbb{R}_+$ over the state space. The policy is conditioned on this parameter, and is given by $\pi(\cdot \mid \mathbf{s}, \omega)$. The objective of a distribution-conditioned (DisCo) policy is to reach states that have high log-likelihood under the goal distribution, which can be expressed as

$$\max_\pi \mathbb{E}_{\tau \sim \pi(\cdot|\mathbf{s},\omega)} \left[\sum_t \gamma^t \log p_g(\mathbf{s}_t; \omega) \right]. \tag{1}$$

This formulation generalizes goal-conditioned RL. We can recover the standard goal-conditioned formulation, in which an agent must reach a single goal $\mathbf{s}_g$, by parameterizing the goal distributions by this state, $p_g(\mathbf{s}; \mathbf{s}_g)$. If $p_g(\mathbf{s}; \mathbf{s}_g)$ is the Dirac-delta distribution centered around s_g, we recover goal-conditioned RL with a sparse indicator reward. If $p_g(\mathbf{s}; \mathbf{s}_g)$ is the Gaussian distribution, we recover goal-conditioned RL with a squared distance reward to the goal. However, we can also express much more complex distributions and tasks, as we illustrate in Figure 1. More formally:

Remark 1 *Any standard reward maximization problem can be equivalently written as maximizing the log-likelihood under a goal distribution (Equation 1), up to a constant factor.*

This statement is true because for any reward function of the form $r(\mathbf{s})$, we can define a distribution $p_g(\mathbf{s}) \propto e^{r(\mathbf{s})}$, from which we can conclude that maximizing $\log p_g(\mathbf{s})$ is equivalent to maximizing $r(\mathbf{s})$, up to the constant normalizing factor in the denominator. If the reward function depends on the action, $r(\mathbf{s}, \mathbf{a})$, we can modify the MDP and append the previous action to the state $\bar{\mathbf{s}} = [\mathbf{s}, \mathbf{a}]$, reducing it to another MDP with a reward function of the form $r(\bar{\mathbf{s}})$.

Of course, while any reward can be expressed as the log-likelihood of a goal distribution, a specific fixed parameterization $p_g(\mathbf{s}; \omega)$ may not by itself be able to express any reward. In other words, choosing the goal distribution parameterization is equivalent to choosing the set of reward functions that the conditional policy can maximize. As we discuss in Section 4.3, we can trade off expressivity and ease of learning by choosing an appropriate goal distribution family.

4.2 Learning Distribution-Conditioned Policies

In this section, we discuss how DisCo RL optimizes Equation 1 using an off-policy TD algorithm. As discussed in Section 3, TD algorithms require tuples of state, action, next state, and reward, denoted by $(\mathbf{s}_t, \mathbf{a}_t, r_t, \mathbf{s}_{t+1})$. Off-policy TD algorithms can sample these tuples from a replay buffer [33] of data collected by a policy as its learns. In our case, at the beginning of each episode, a policy is conditioned on a goal distribution parameter ω, collects a trajectory $\tau = [\mathbf{s}_0, \mathbf{a}_1, \cdots]$, and stores it in the replay buffer, denoted as $\mathcal{R}$. We could simply store the parameter ω that was used to collect the trajectory in the replay buffer $\mathcal{R}$, but, as we discuss below, we can greatly improve the sample-efficiency of our method by dynamically relabeling this parameter.

Because TD algorithms are off-policy, we propose to reuse data collected by a policy conditioned on one goal distribution ω to learn about how a policy should behave under another goal distribution ω'. In particular, given a state $\mathbf{s}$ sampled from a policy that was conditioned on some goal distribution parameters ω, we occasionally relabel the goal distribution with an alternative goal distribution $\omega' = RS(\mathbf{s}, \omega)$ for training, where RS is some relabeling strategy. This relabeling is similar to relabeling methods used in goal-conditioned reinforcement learning [1, 11, 13, 5, 6, 15, 7], but applied to goal distributions rather than individual goal states. In the next section, we discuss the specific relabeling strategy RS that we use, which depends on the method for generating goal distributions.

We present a summary of the algorithm in 1. In our experiments, we use soft actor-critic as our RL algorithm [34], though in theory a number of other off-policy algorithms could be used.

4.3 Goal Distribution Parameterization

The choice of goal distribution parameterization is important, as it balances the expressivity of DisCo RL with how easy it is to train the DisCo policy. In our experiments, we evaluate DisCo RL using the

Algorithm 1 Distribution-Conditioned Reinforcement Learning

Require: Policy π_θ, Q-function Q_ϕ, TD algorithm $\mathcal{A}$, relabeling strategy RS, exploration parameter distribution p_ω, replay buffer $\mathcal{R}$.

for $0, \dots, N_{\text{episode}} - 1$ episodes **do**
  Sample initial state $\mathbf{s}_0 \sim p_0$ and distribution parameters $\omega \sim p_\omega(\cdot)$.
  Sample trajectory from environment $\tau \sim \pi(\cdot|\mathbf{s}; \omega)$.
  Store trajectory with distribution parameter (τ, ω) in replay buffer $\mathcal{R}$.
  for $0, \dots, N_{\text{updates}} - 1$ steps **do**
    Sample trajectory and parameter $(\tau, \omega) \sim \mathcal{R}$.
    Sample transition tuple $(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})$ and a future state $\mathbf{s}_h$ from τ, where $t < h$.
    With probability p_{relabel}, relabel the goal distribution parameters $\omega \leftarrow RS(\mathbf{s}_h, \omega)$.
    Compute reward $r = \log p_g(\mathbf{s}_t; \omega)$ and augment states $\hat{\mathbf{s}}_t \leftarrow [\mathbf{s}_t; \omega]$, $\hat{\mathbf{s}}_{t+1} \leftarrow [\mathbf{s}_{t+1}; \omega]$.
    Update Q_ϕ and π_θ using $\mathcal{A}$ and data $(\hat{\mathbf{s}}_t; \mathbf{a}_t, \hat{\mathbf{s}}_{t+1}, r)$.

family of multivariate Gaussian distributions. Given a state space in $\mathbb{R}^n$, the distribution parameters consists of two components, $\omega = (\mu, \Sigma)$, where $\mu \in \mathbb{R}^n$ is the mean vector and $\Sigma \in \mathbb{R}^{n \times n}$ is the covariance matrix. With these parameters, the reward from Equation 1 is given by

$$r(\mathbf{s}; \omega) = -0.5(\mathbf{s} - \mu)^T \Sigma^{-1} (\mathbf{s} - \mu), \tag{2}$$

where we have dropped constant terms that do not depend on the state $\mathbf{s}$. This simple parameterization can express a large number of reward functions. As discussed earlier, with an identity covariance matrix, setting the mean to a specific value corresponds to goal-conditioned RL with a squared-distance reward. However, it also captures a broader set of tasks. As visualized in Figure 1, using this parameterization, the weight of individual state dimensions depend on the values along the diagonal of the covariance matrix. By using off-diagonal covariance values, this parameterization also captures the set of tasks in which state components need to covary, such as when the one object must be placed near another one, regardless of the exact location of those objects. In practice, we found that dropping the constant term and using the square-root before the -0.5 factor performed slightly better, and so we used this reward.

5 Constructing Goal Distributions

The previous section presented an algorithm for learning DisCo RL policies, assuming that a goal distribution is provided. In this section, we discuss two applications of DisCo RL, describing how we obtain goal distributions in each case.

Inferring goal distributions. While a user can manually select the goal distribution parameters ω, this requires a degree of user insight. A more intuitive way to specify the goal distribution is to provide K example observations $\{\mathbf{s}_k\}_{k=1}^K$ in which the task is successfully completed. This supervision can be easier to provide than full demonstrations, which not only specify the task but also must show how to solve the task through a sequence of states $(\mathbf{s}_1, \mathbf{s}_2, \dots)$ or state and actions $(\mathbf{s}_1, \mathbf{a}_1, \mathbf{s}_2, \dots)$. Given this set of examples, we learn a goal distribution via maximum likelihood estimation (MLE), such that

$$\omega^* = \arg\max_{\omega \in \Omega} \sum_{k=1}^{K} \log p_g(\mathbf{s}_k; \omega),$$

and condition the DisCo RL policy on the resulting parameters ω^*. In other words, we choose the Dirac-delta distribution $p_\omega(\omega) = \delta(\omega = \omega^*)$.

For relabeling, given a state $\mathbf{s}$ and existing parameters $\omega = (\mu, \Sigma)$, we would like to provide a strong learning signal by creating a distribution parameter that gives high reward to an achieved state. While a number of distribution relabeling schemes could be used, we found that the simple strategy of keeping the covariance matrix Σ and replacing the mean with the state vector $\mathbf{s}$, as in $RS(\mathbf{s}, \mu, \Sigma) = (\mathbf{s}, \Sigma)$, performed well.

Conditional distributions for multi-stage tasks. While DisCo RL is a more general framework than goal-conditioned RL, it can also be used to facilitate learning goal-directed behavior, in particular when tasks can be decomposed into multiple stages. For example, as we visualize in Figure 2

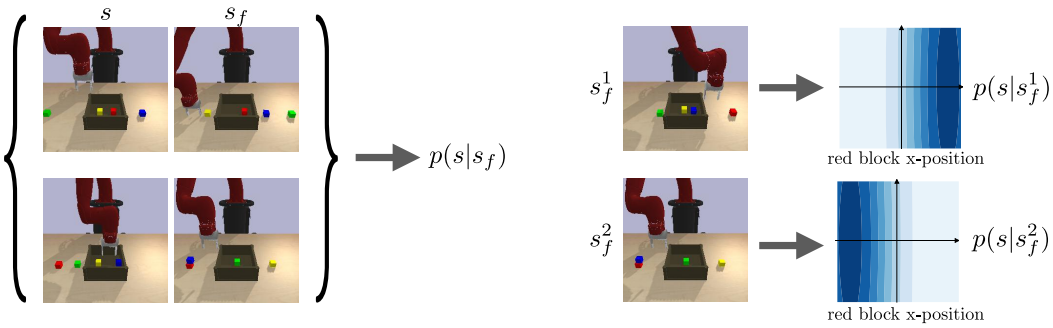


Figure 2: (Left) A robot must arrange objects into a configuration, $\mathbf{s}_f$, that varies between episodes. This task contains sub-tasks, such as first moving the red object to the correct location. Given a final state $\mathbf{s}_f$, there exists a distribution of intermediate states $\mathbf{s}$ in which this first sub-task is completed. We use pairs of states $(\mathbf{s}, \mathbf{s}_f)$ to learn a conditional distribution $p(\mathbf{s} \mid \mathbf{s}_f)$, which (right) defines the first sub-task given the final state, $\mathbf{s}_f$.

a robot that must arrange a dinner table may have a final desired goal state $\mathbf{s}_f$, that specifies the location of every piece of silverware, but this task can be decomposed into several sub-tasks, such as placing one plate in a desired location, each of which can be expressed as a DisCo RL problem. This decomposition can make learning significantly easier, compared to training a single policy to directly reach $\mathbf{s}_f$. In this situation, we need a way to generate a goal distribution for each sub-task, conditioned on a final state $\mathbf{s}_f$.

Formally, we construct a map $h : \mathcal{S} \mapsto \Omega$ that maps a final state $\mathbf{s}_f$ to a goal distribution parameter ω that represents a sub-task for reaching $\mathbf{s}_f$. This map allows us to dynamically generate distributions over the state space conditioning on $\mathbf{s}_f$, by mapping a state $\mathbf{s}_f$ into a distribution parameter $\omega = h(\mathbf{s}_f)$. To obtain the map h, we learn a conditional distribution $p_{\mathbf{S}|\mathbf{S}_f}$, which maps an observation of the random variable $\mathbf{s}_f$ into a probability distribution over the state space. To obtain this conditional distribution, we assume that data is given in the form of pairs of states $\mathcal{D}_{\text{subtask}} = \{(\mathbf{s}^{(k)}, \mathbf{s}_f^{(k)})\}_{k=1}^K$, in which $\mathbf{s}^{(k)}$ correspond to a state where a sub-task is accomplished when trying to reach the final state $\mathbf{s}_f$. We fit a joint Gaussian distribution, denoted by $p_{\mathbf{S},\mathbf{S}_F}(\mathbf{s}, \mathbf{s}_f)$, to these pairs of states, and then compute the conditional distribution in closed form. For details on this process, see Section E.2.

6 Experiments

Our experiments study the following questions: **(1)** How does DisCo RL with a learned distribution compare to using manually-designed rewards and prior work that also uses successful states for computing rewards? **(2)** How does DisCo RL compare to prior methods at training policies to solve multiple tasks? **(3)** Can we apply DisCo RL to solve long-horizon tasks that are decomposed into shorter sub-tasks? **(4)** How do DisCo policies perform when conditioned on goal distributions that were never used for data-collection? We also include ablations for each set of experiments that study the importance of the relabeling strategy presented in Section 4.2.

We study these questions in three simulated manipulation environments, shown in Figure 3. The first environment contains a Sawyer robot, a rectangular tray, and four blocks, which the robot must learn to manipulate. The agent controls the velocity of a Sawyer arm and gripper, and the arm is restricted to move in a 2D plane perpendicular to the table's surface. We also use an IKEA furniture assembly environment from Lee et al. [35]. An IKEA agent controls the velocity of a cursor that can lift and place shelves onto a pole. Shelves are picked up and connected automatically when they are within a certain distance of the cursor or pole. Lastly, we use a two-dimensional "Flat World" environment in which an agent can pick up and place objects at various locations. The states comprise the Cartesian position of all relevant objects and, for the Sawyer task, the gripper state. All plots show mean and standard deviation across 5 seeds, as well as maximum performance with a gray, dashed line.

Baseline and comparisons. To demonstrate the difficulty of the tasks, we compare to goal-conditioned RL (**GCRL**). Many of the tasks are not well suited for GCRL, meaning that the optimal solution to these tasks is not equivalent to reaching a single state. In order to compare to goal conditioned RL we perform *oracle goal setting*, where the features of the state that are irrelevant for the task are set to their initial state $\mathbf{s}_0$. Note that this oracle method receives additional information

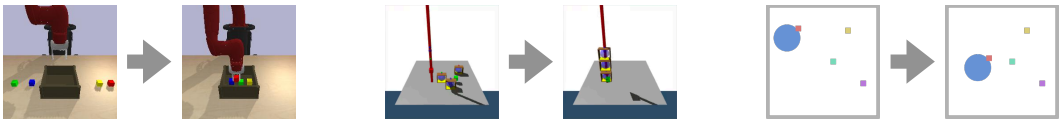


Figure 3: Illustrations of the domains we use for evaluation, in which a policy must (left) control a Sawyer arm to move cubes into and out of a tray (center) attach shelves to a pole using a cursor, and (right) use the blue robot to move objects to different locations. Our tasks requires that the policy ignores objects or move the object based on relative distances, such as moving an object into the tray regardless of the tray's location.

that is not available for our method. For details on the goal-conditioned RL baseline algorithm see Appendix E. We also compare our method to variational inverse control with events (**VICE**) [21], which trains a success classifier on user-provided positive examples, and then uses this classifier as a reward. This comparison is relevant because our method also fits the goal distribution to positive examples. However, while VICE learns one task at a time, our method learns a policy that is conditioned on the goal distribution, and therefore can perform many tasks. Further details about the environments and tasks are in Appendix C.

Distribution inference. Our first set of experiments study the first question by comparing the performance of DisCo RL with a learned goal distribution to using a manually specified goal distribution and existing approaches for solving reinforcement learning problems. To compare to standard reinforcement learning methods, we consider the single task setting, where there is only one reward (or one distribution) on which the policy is evaluated. The task is to control the Sawyer arm and move the red object into the bowl while ignoring the remaining three "distractor" objects. We generated $K = 30$ examples of successful states, by repeatedly randomly sampling the location of the tray, placing the red object inside the tray, and then randomly sampling the location of the three distracting objects. When inferring the goal distribution from data, we regularize the covariance to avoid degeneracy, as discussed in Appendix E. Note that this task cannot be expressed as a single goal state $\mathbf{s}_g$, since the location of the distractors are unimportant. In addition to VICE and GCRL, we compare to DisCo RL using a manually specified parameterization of the goal distribution, as well as soft actor critic [34], which uses the same reward as the manually selected goal distribution, but does not condition a policy on ω. We see in Figure 4 that DisCo RL matches the performance of using a manually specified reward function and outperforms previous methods, indicating that learning a goal distribution can be used to solve single-task problems effectively.

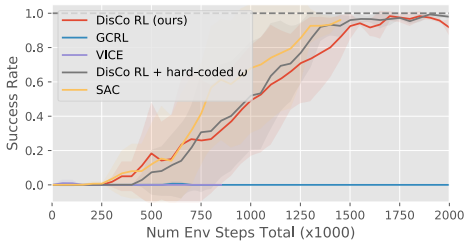


Figure 4: Learning curve on the single-object moving task. DisCo RL with a learned goal distribution performs just as well as using a manually specified goal distribution or reward function.

Multi-task performance. To evaluate how well a single DisCo policy solves multiple tasks specified by different distributions, we consider a task in the Sawyer environment in which the location of the tray is fixed, and the agent must move one randomly chosen object of the four to a target location. To generate goal distributions, we use the conditional distribution formulation presented in Section 5. We construct four different example sets $\mathcal{D}^i_{\text{subtask}}$, for $i = 1, 2, 3, 4$. Each set i contains pairs of state $(\mathbf{s}, \mathbf{s}_f)$, in which object i is in the same location as in the final desired state $\mathbf{s}_f$. See Figure 2 for an illustration. During training, we sample an initial state $\mathbf{s}_0$ and final goal state $\mathbf{s}_f$ uniformly from the set of possible states, and condition the policy on $p_i(\mathbf{s} \mid \mathbf{s}_f)$.

For evaluation, we test how well DisCo RL can solving long-horizon tasks by conditioning the policy on each $p_i(\mathbf{s} \mid \mathbf{s}_f)$ for $H/4$ time steps. We report the cumulative number of tasks that were solved, where each task is considered solved when the respective object is within a minimum distance of its target location specified by $\mathbf{s}_f$. We design an analogous task in the IKEA environment (where moving each shelf corresponds to its own task) and Flat World (moving each object corresponds to its own task). We compare to VICE which runs a separately trained policy for each sub-task in sequence, and GCRL, which is directly conditioned on $\mathbf{s}_f$.

We see in Figure 5 that DisCo RL significantly outperforms VICE and the goal-conditioned baseline and successfully generalizes to new goal distributions at test time. While we were initially surprised by the poor performance of GCRL, we found that the goal-conditioned policy learned to only focus on moving the end effector, as we show in Appendix B. We note that this is a relatively common

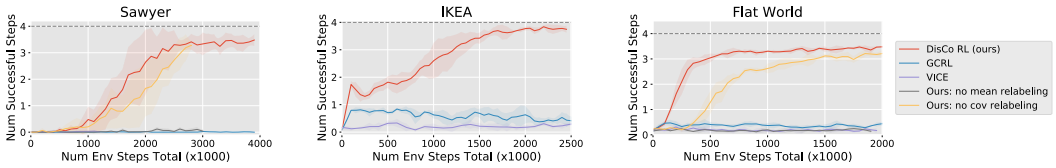


Figure 5: Learning curves for Sawyer (left), IKEA (middle) and Flat World (right) showing the number of cumulative tasks completed across all three domains. We see that DisCo RL significantly out-performs GCRL and VICE. We show in Appendix B that GCRL mainly fails because it only focuses on reaching the target end effector or cursor location.

failure mode of GCRL, which is why many existing GCRL methods manually design a goal space that excludes certain state variables [2, 11, 12, 15, 14].

Off-policy distributions. One of the benefits of DisCo RL is that it is an off-policy method, meaning that a DisCo policy can be trained on goal distributions other than the goal distributions used to collect data, and, in theory, perform well on these goal distributions. The next experiment evaluates this ability of DisCo policies to learn using off-policy data on the Flat World environment. We collect data using four goal distributions and train our policy on the four exploration goal distributions as well as six additional goal distributions in an off-policy manner, and evaluate our policy on all ten goal distributions. The four exploration distributions involve moving single objects, while the six additional distribution require moving pairs of objects. This setting requires that the policy uses its experience when moving individual objects to learn how to move pairs of objects, despite never attempting to move a pair of objects at the same time during exploration. We compare to an on-policy variant, in which we collect data using all ten distributions, and see in Figure 6 that off-policy learning performs comparably to on-policy learning.

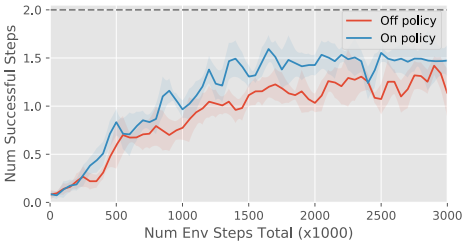


Figure 6: Learning curve when evaluated on off-policy goal distributions. Although these goal distributions were never used to explore, we see that the DisCo policy performs just as well as if it had explored with those distributions.

Ablations. We include ablations that test the importance of relabeling the mean and covariance parameters during training. We see in Figure 5 that relabeling both helps DisCo policies perform better (results on IKEA not available due to computational constraints).

7 Discussion

We presented DisCo RL, a method for learning general-purpose policies specified using a goal distribution. Our experiments show that DisCo policies can solve a variety of tasks using goal distributions inferred from data, and can accomplish tasks specified by goal distributions that were not seen during training. In this work, we studied Gaussian parameterizations of goal distributions, and an exciting direction for future work would be to use more complex distributions to capture a broader range of reward functions. Fortunately, learning to compactly represent complex distributions is an active area of research in machine learning, which could make for a natural goal distribution parameterization. One promising approach would be to learn latent representations of states where Gaussian distributions correspond to complex distributions in the state space, as done with deep latent variables models [36, 37]. Additionally, since distribution-conditioned RL generalizes goal-conditioned RL, an exciting direction for future work would be to apply it in various settings where goal-conditioned RL has been used previously. For example, work on goal-directed exploration [38, 39, 40, 9] could naturally be extended to exploration in goal-distribution space, which can enable an agent to more flexibly explore an environment based on its current state. Similarly, work on hierarchical goal-conditioned methods [41, 42, 12, 6, 5] could be extended to allow more abstract notions of sub-tasks that are specified by a high-level controller.

8 Acknowledgements

I would like to thank my team of collaborators for this project: Vitchyr Pong, Ashvin Nair, Alexander Khazatsky, Glen Berseth, and Sergey Levine.

Each member provided invaluable contributions to this project: (1) SN built the environments and executed the primary experiments, (2) VP led the writing of this paper, (3) AN executed the VICE baseline, and (4) AK provided primary figures. In addition, VP, AN, and GB provided high-level ideas throughout the project and contributed to writing. SL was the primary adviser for this project.

A special thank you to Sergey Levine and Vitchyr Pong for their invaluable advising and mentorship throughout my time as a member of the RAIL lab.

References

[1] L. P. Kaelbling. Learning to achieve goals. In *IJCAI*, volume vol.2, pages 1094 – 8, 1993.

[2] T. Schaul, D. Horgan, K. Gregor, and D. Silver. Universal Value Function Approximators. In *ICML*, 2015.

[3] D. Pathak, P. Mahmoudieh, G. Luo, P. Agrawal, D. Chen, Y. Shentu, E. Shelhamer, J. Malik, A. A. Efros, and T. Darrell. Zero-Shot Visual Imitation. In *ICLR*, 2018.

[4] V. Dhiman, S. Banerjee, J. M. Siskind, and J. J. Corso. Floyd-warshall reinforcement learning: Learning from past experiences to reach new goals. *arXiv:1809.09318*, 2018.

[5] V. Pong, S. Gu, M. Dalal, and S. Levine. Temporal Difference Models: Model-Free Deep RL For Model-Based Control. In *ICLR*, 2018.

[6] O. Nachum, G. Brain, S. Gu, H. Lee, and S. Levine. Data-Efficient Hierarchical Reinforcement Learning. In *NeurIPS*, 2018.

[7] A. Nair, V. Pong, M. Dalal, S. Bahl, S. Lin, and S. Levine. Visual Reinforcement Learning with Imagined Goals. In *NeurIPS*, 2018.

[8] D. Warde-Farley, T. Van De Wiele, T. Kulkarni, C. Ionescu, S. Hansen, and M. Volodymyr. Unsupervised Control Through Non-Parametric Discriminative Rewards. In *ICLR*, 2019.

[9] V. H. Pong, M. Dalal, S. Lin, A. Nair, S. Bahl, and S. Levine. Skew-fit: State-covering self-supervised reinforcement learning. *CoRR*, abs/1903.03698, 2019.

[10] T. Jurgenson, E. Groshev, and A. Tamar. Sub-goal trees–a framework for goal-directed trajectory prediction and optimization. *arXiv:1906.05329*, 2019.

[11] M. Andrychowicz, F. Wolski, A. Ray, J. Schneider, R. Fong, P. Welinder, B. Mcgrew, J. Tobin, P. Abbeel, and W. Zaremba. Hindsight Experience Replay. In *NeurIPS*, 2017.

[12] A. Levy, R. Platt, and K. Saenko. Hierarchical Actor-Critic. *arXiv:1712.00948*, 2017.

[13] P. Rauber, F. Mutz, and J. J. Schmidhuber. Hindsight policy gradients. In *CoRR*, volume abs/1711.0, 2017.

[14] C. Florensa, D. Held, X. Geng, and P. Abbeel. Automatic goal generation for reinforcement learning agents. In *ICML*, 2018.

[15] M. Plappert, M. Andrychowicz, A. Ray, B. Mcgrew, B. Baker, G. Powell, J. Schneider, J. Tobin, M. Chociej, P. Welinder, V. Kumar, and W. Zaremba. Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research. *arXiv:1802.09464*, 2018.

[16] C. Colas, P. Fournier, M. Chetouani, O. Sigaud, and P.-Y. Oudeyer. Curious: intrinsically motivated modular multi-goal reinforcement learning. In *ICML*, pages 1331–1340, 2019.

[17] A. Laversanne-Finot, A. Pere, and P.-Y. Oudeyer. Curiosity driven exploration of learned disentangled goal spaces. In *CoRL*, pages 487–504, 2018.

[18] A. Nair, S. Bahl, A. Khazatsky, V. Pong, G. Berseth, and S. Levine. Contextual imagined goals for self-supervised robotic learning. In *CoRL*, pages 530–539, 2020.

[19] S. Nair, S. Savarese, and C. Finn. Goal-aware prediction: Learning to model what matters. In *ICML*, 2020.

[20] F. Torabi, G. Warnell, and P. Stone. Generative adversarial imitation from observation. *arXiv:1807.06158*, 2018.

[21] J. Fu, A. Singh, D. Ghosh, L. Yang, and S. Levine. Variational inverse control with events: A general framework for data-driven reward definition. In *NeurIPS*, pages 8538–8547, 2018.

[22] F. Torabi, G. Warnell, and P. Stone. Behavioral cloning from observation. In *IJCAI*, 2018.

[23] P. Sermanet, C. Lynch, Y. Chebotar, J. Hsu, E. Jang, S. Schaal, S. Levine, and G. Brain. Time-contrastive networks: Self-supervised learning from video. In *ICRA*, 2018.

[24] Y. Liu, A. Gupta, P. Abbeel, and S. Levine. Imitation from Observation: Learning to Imitate Behaviors from Raw Video via Context Translation. In *ICRA*, 2018.

[25] A. Edwards, H. Sahni, Y. Schroecker, and C. Isbell. Imitating latent policies from observation. In *ICML*, pages 1755–1763, 2019.

[26] T. D. Kulkarni, A. Saeedi, S. Gautam, and S. J. Gershman. Deep successor reinforcement learning. *arXiv:1606.02396*, 2016.

[27] 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*, pages 4055–4065, 2017.

[28] D. Borsa, A. Barreto, J. Quan, D. J. Mankowitz, H. van Hasselt, R. Munos, D. Silver, and T. Schaul. Universal successor features approximators. In *ICLR*, 2018.

[29] A. Barreto, D. Borsa, J. Quan, T. Schaul, D. Silver, M. Hessel, D. Mankowitz, A. Zidek, and R. Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *ICML*, pages 501–510, 2018.

[30] L. Lee, B. Eysenbach, E. Parisotto, E. Xing, S. Levine, and R. Salakhutdinov. Efficient exploration via state marginal matching. In *ICLR*, 2019.

[31] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. 1998.

[32] A. Hallak, D. Di Castro, and S. Mannor. Contextual markov decision processes. *arXiv:1502.02259*, 2015.

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

[34] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. In *ICML*, 2018.

[35] Y. Lee, E. S. Hu, Z. Yang, A. Yin, and J. J. Lim. IKEA furniture assembly environment for long-horizon complex manipulation tasks. *arXiv:1911.07246*, 2019.

[36] D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. In *ICLR*, 2014.

[37] D. J. Rezende, S. Mohamed, and D. Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In *ICML*, 2014.

[38] S. Koenig and R. G. Simmons. The effect of representation and knowledge on goal-directed exploration with reinforcement-learning algorithms. *Machine Learning*, 22(1-3):227–250, 1996.

[39] A. Ecoffet, J. Huizinga, J. Lehman, K. O. Stanley, and J. Clune. Go-explore: a new approach for hard-exploration problems. *CoRR*, abs/1901.10995, 2019.

[40] Z. Ren, K. Dong, Y. Zhou, Q. Liu, and J. Peng. Exploration via hindsight goal generation. In *NeurIPS*, pages 13485–13496, 2019.

[41] P. Dayan and G. E. Hinton. Feudal reinforcement learning. In *NeurIPS*, pages 271–278, 1993.

[42] A. S. Vezhnevets, S. Osindero, T. Schaul, N. Heess, M. Jaderberg, D. Silver, and K. Kavukcuoglu. FeUdal Networks for Hierarchical Reinforcement Learning. In *ICML*, 2017.

[43] A. Singh, L. Yang, K. Hartikainen, C. Finn, and S. Levine. End-to-End Robotic Reinforcement Learning without Reward Engineering. In *RSS*, 2019.

[44] E. Coumans et al. Bullet physics library. *Open source: bulletphysics. org*, 15(49):5, 2013.

[45] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, V. Vanhoucke, G. Brain, and G. Deepmind. Sim-to-Real: Learning Agile Locomotion For Quadruped Robots. In *RSS*, 2018.

[46] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine. Learning agile robotic locomotion skills by imitating animals. *RSS*, 2020.

[47] J. Fu, K. Luo, and S. Levine. Learning Robust Rewards With Adversarial Inverse Reinforcement Learning. In *ICLR*, 2018.

[48] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond Empirical Risk Minimization. In *ICLR*, oct 2018.

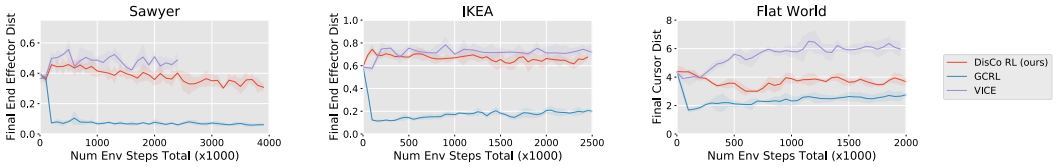
A Generalization to Real-World Robots

While our experiments were only conducted in simulated environments, we expect DisCo RL to generalize to real-world robots. We note that both goal-conditioned RL [7] and VICE [43] have been applied to real-world robot domains, and that Figure 5 demonstrates that DisCo RL can significantly outperform these methods in multi-task settings where tasks are not specified by single goals. We note that these findings are consistent across all three simulated domains, suggesting that these results are general properties of the methods and may apply to other domains, including real-world robotics. We also note that the PyBullet simulator [44] has been successfully applied for sim-to-real transfer [45, 46], suggesting that strong performance in the simulator can generalize to real-world robots.

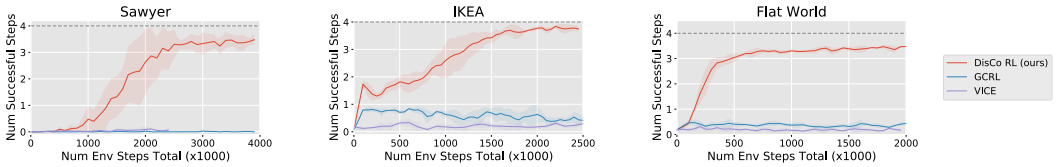
Lastly, in our experiments DisCo RL used 30 to 50 examples to learn goal distributions that specify different robot tasks. Specifying such few number of example successful states is particularly practical for real-world domains, where specifying tasks often requires manually specifying reward functions or adding task-specific instrumentation and sensors.

B Additional Results

The objective of the tasks in Section 6 was to move objects to certain positions, and we found that goal-conditioned reinforcement learning (GCRL) performed poorly. To better understand the behavior of the GCRL policies, we plotted the distance between the final end effector position reached and the one specified by the final goal state $\mathbf{s}_g$. In Figure 7, we see that the GCRL policies quickly learned to minimize the final distance to the end effector position in the goal state $\mathbf{s}_g$. We see that the GCRL policies optimized only the dimensions of the goal state that were easiest to maximize, rather than the important dimensions. VICE had unsatisfactory performance even on the end effector position, and we hypothesize this is due to the fact that VICE needs a substantially higher number of example states.



(a) final end-effector distance



(b) success rate

Figure 7: (a) The distance between the final end-effector position and the position specified in the goal state $\mathbf{s}_g$. (b) The success rate of the methods on the same task (with results copied from Figure 5 for convenience). We see that goal-conditioned RL focuses primarily on moving the end effector to the correct position, while DisCo RL ignores this task-irrelevant dimension and successfully completes the task.

Videos of the final policies for our method and baselines are available on the paper website: `https://sites.google.com/view/disco-rl`

C Environments

Sawyer This environment is based in the PyBullet [44] physics simulator. It consists of a Sawyer robot mounted next to a table, on top of which there is a tray and four blocks. The robot must learn to manipulate the blocks via its gripper. The robot is controlled via position control, and it is restricted to move in a 2D plane. Specifically, the robot arm can move in the YZ coordinate plane and the gripper can open along the X axis, where the X, Y, Z axes move along the front-back, left-right,

and up-down directions of table, respectively. We also constrain the objects to move along the YZ coordinate plane. The agent has access to state information, comprising of the position of the end effector and gripper state, as well as the positions of the objects and tray.

IKEA We adapted this environment from the suite of furniture assembly environments developed by Lee et al. [35]. In our environment, the agent must learn to attach a set of 3 shelves to a pole. It can do so by controlling two end effectors: one end effector that can move the pole, and another end effector that can move the shelves. The former end effector is always attached to the pole, while the latter end effector can selectively attach and detach itself from the shelves. Both end effectors can move via 3D position control, in a $1 \times 1 \times 1$ area for a maximum of 0.05 units (in each direction) per timestep. The end effector interfacing with the shelves can hold onto a shelf by applying a grasp action when it is within the bounding box region of a shelf. Each shelf has a connection point at which point it will attach to the pole, and the pole has a receiving connection point as well. When these two connection points are within 0.2 units away from one another, the shelf automatically attaches to the pole and becomes welded. The objects have 3 degrees of freedom via translations in 3D space. The objects are not allowed to collide with one another – if an action causes them to collide, that action is ignored by the environment and the next state is the same as the current state. As an exception, when an end effector is not grabbing an object, it is allowed to move through objects. The agent has access to state information, comprising the 3D position of the end effectors, shelves, and the pole, and indicator information for whether each end effector is grasping an object.

Flat World This two-dimensional environment consists of a policy and 4 objects, each of which are defined by their XY-coordinate. The policy and objects are in an enclosed 8×8 unit space. The policy's action space is three dimensional: two correspond to relative change in position, for a maximum of 1 unit in each dimension per timestep, and one corresponds to a grab action. The grab action takes on a value between -1 and 1. If this grab action is positive, then the policy picks up the closest object that is within 1 unit of it, or none if there are no such objects. If this grab action is non-positive, then the policy drops any object that it was holding. While an object is grabbed, the object moves rigidly with the policy. The policy can only grab one object at a time, with ties broken by a predetermined, fixed order. The agent has access to state information, comprising the 2D position of the policy and the 4 objects.

D Experimental Details

Distribution inference These experiments were evaluated on the Sawyer environment. In this setting, the agent needs to pick up one of the blocks (specifically, the red block) and place it into the tray. The initial position of the tray and objects vary in each episode. The objective is only to minimize the relative distance between the red block and the tray, and it is important for the agent to ignore the absolute position of the other blocks and the tray. The robot can attempt to slide the tray to a specific goal location, but the tray is heavy and moves very slowly. If it successfully moves the tray to another location, it will not have enough time in the episode to move the red block. Thus, the GCRL baseline will fail for this task unless the goal position of the tray matches the initial location of the tray at the beginning of the episode.

We generated $K = 30$ examples of successful goal states, in which the red block is always inside the tray, and the tray, other objects, hand, and gripper are in random locations and configurations. We provided this set of example goal states as input to the competing baselines. Each baseline used the example states in the following manner:

- DisCo RL: infer a goal distribution from the example states
- GCRL: the example states are the set of goals for exploration rollouts, and the set of goals sampled for relabeling in training
- VICE: a classifier is trained to predict whether a state is optimal, with the example states as the positive examples for the classifier

For evaluation, a trajectory is successful if the red block is placed in the tray. We plot this success metric over time in Figure 4.

Multi-task performance For the multi-task evaluations, we performed experiments in all three of our environments. For each environment, we split the task into several subtasks, as described below:

- Sawyer: move the blocks to their goal locations. Each subtask represents moving one block at a time to its goal location.
- IKEA: move the pole and the shelves to their goal locations. Each subtask represents moving the pole and one of the shelves to their goal locations.
- Flat World: move the objects to their goal locations. Each subtask represents moving one object at a time to its goal location.

We generated an example dataset of successful states for each subtask. For each example state for a particular subtask, we also provided a state representing the final configuration for the entire task (after all subtasks are solved). See Section E.2 for additional details regarding the example sets. The evaluation metrics for each environment are as follows:

- Sawyer: the number of objects that are within 0.10 units of their respective final goal locations
- IKEA: the number of shelves that are connected to the pole, in addition to an indicator for whether the pole is within 0.10 units of its final goal location
- Flat World: the number of objects that are within 1 unit of their respective final goal locations

For evaluation, we provide the GCRL baseline oracle goals. In this setting, the provided goal is identical to the initial state at the beginning of the episode, except for the position of the red block, which we set to be inside the tray. For example, if the state is given by

$$\mathbf{s}_0 = \left[x_0^{\text{EE}}, y_0^{\text{EE}}, x_0^{\text{red-block}}, y_0^{\text{red-block}}, x_0^{\text{blue-block}}, y_0^{\text{blue-block}}, \ldots\right],$$

and we want the red block to move to a position (x^*, y^*), we set the goal to

$$\mathbf{s}_g = \left[x_0^{\text{EE}}, y_0^{\text{EE}}, x^*, y^*, x_0^{\text{blue-block}}, y_0^{\text{blue-block}}, \ldots\right].$$

In theory, this oracle goal encourages the robot to focus on moving the red block to its goal rather than moving the other state components. The GCRL baseline only uses this oracle during evaluation.

Details regarding the distribution of final goal states used for exploration rollouts and relabeling during training, are provided in Table 1. For evaluation, we used $H = 100$ for the IKEA and Flat World environments, and $H = 400$ for the Sawyer environment.

Goal Use Case	**Sawyer**	**IKEA**	**Flat World**
Exploration	50%: objects on ground, 50%: objects in tray	Shelves assembled, pole in random position	Objects in random positions
Training	Same as above	Objects in random positions	Same as above

Table 1: Environment specific final goal distributions.

Off-policy distributions These experiments are conducted on the Flat World domain and extend the multi-task experiments described in the previous section. We created a total of 10 total subtasks, 4 of which require moving a single object to its desired goal location, and 6 of which require moving a pair of objects to their desired goal locations. We provided example sets of size $K = 30$ for each of these 10 subtasks. For the on-policy variant, we explore and relabel with the goal distributions inferred for all 10 subtasks. For the off-policy variant, we only explore with the first 4 goal distributions and train with all 10 goal distributions. For evaluation, we measured the performance of the algorithm for one of the pairwise subtasks. Our evaluation metric measures how many objects (out of the specific pair) the agent was able to successfully move within 1 unit of their respective final goal locations.

E Implementation Details

E.1 General Training Algorithm and Hyperparameters

In our experiments, we use soft actor-critic as our RL algorithm [34]. For specific details on the hyperparameters that we used, see Table 3.

Hyper-parameter	Sawyer	IKEA	Flat World
Number of examples (per subtask) K	30	50	30
Std. dev. of Gaussian noise added to example set data	0.01	0.1	0.01

Table 2: Environment specific hyper-parameters.

Hyper-parameter	Value
horizon H (for training)	100
batch size	2048
discount factor	0.99
Q-function and policy hidden sizes	$[400, 300]$
Q-function and policy hidden activations	ReLU
replay buffer size	1 million
hindsight relabeling probability	80%
target network update speed τ	0.001
number of training updates per episode $N_{\text{updates per episode}}$	100
number of training batches per environment step	1

Table 3: General hyper-parameters used for all experiments.

E.2 DisCo RL

Covariance Smoothing We apply pre-preprocessing and post-processing steps to obtain the distribution parameters used in RL. In the pre-processing phase, we add i.i.d. Gaussian noise to the dataset of examples. The amount of noise that we apply varies by environment – see Table 2 for specific details. After inferring the raw parameters of the Gaussian distribution μ and Σ, we apply post-processing steps to the covariance matrix. We begin by inverting the covariance matrix Σ. For numerical stability, we ensure that the condition number of Σ does not exceed 100 by adding a scaled version of the identity matrix to Σ. After obtaining Σ^{-1}, we normalize its components such that the largest absolute value entry of the matrix is 1. Finally, we apply a regularization operation that thresholds all values of the matrix whose absolute value is below 0.25 to 0. We found this regularization operation to be helpful when the number of examples provided is low, to prevent the Gaussian model from inferring spurious dependencies in the data. We used the resulting μ and Σ^{-1} in Equation 2 for computing the reward.

Conditional distribution details To obtain the conditional distribution used for relabeling and multi stage planning, we assume that data is given in the form of pairs of states $\{(\mathbf{s}^{(k)}, \mathbf{s}_f^{(k)})\}_{k=1}^K$, in which $\mathbf{s}^{(k)}$ correspond to a state where a sub-task is accomplished when trying to reach the final state $\mathbf{s}_f$. We fit a joint Gaussian distribution of the form

$$p_{\mathbf{S},\mathbf{S}_f} = \mathcal{N}\left(\begin{bmatrix}\mu_{\mathbf{s}} \\ \mu_{\mathbf{s}_f}\end{bmatrix}, \begin{bmatrix}\Sigma_{\mathbf{ss}} & \Sigma_{\mathbf{ss}_f} \\ \Sigma_{\mathbf{s}_f\mathbf{s}} & \Sigma_{\mathbf{s}_f\mathbf{s}_f}\end{bmatrix}\right), \mu_{(\cdot)} \in \mathbb{R}^{\dim(\mathcal{S})}, \Sigma_{(\cdot)} \in \mathbb{R}^{\dim(\mathcal{S})\times\dim(\mathcal{S})}$$

to these pairs of states using maximum likelihood estimation. Since the joint distribution $p_{\mathbf{S},\mathbf{S}_f}$ is Gaussian, the conditional distribution $p_{\mathbf{S}|\mathbf{S}_f}$ is also Gaussian with parameters $(\mu, \Sigma) = h(\mathbf{s}_f)$, where h is the standard conditional Gaussian formula:

$$h(\mathbf{s}_f) = \left(\underbrace{\mu_{\mathbf{s}} + \Sigma_{\mathbf{ss}_f}\Sigma^{-1}_{\mathbf{s}_f\mathbf{s}_f}(\mathbf{s}_f - \mu_{\mathbf{s}_f})}_{\mu}, \underbrace{\Sigma_{\mathbf{ss}} - \Sigma_{\mathbf{ss}_f}\Sigma^{-1}_{\mathbf{s}_f\mathbf{s}_f}\Sigma_{\mathbf{s}_f\mathbf{s}}}_{\Sigma}\right). \tag{3}$$

In summary, given a final desired state $\mathbf{s}_f$, we generate a distribution by computing $\omega = h(\mathbf{s}_f)$ according to Equation 3. This conditional distribution also provides a simple way to relabel goal distributions given a reached state $\mathbf{s}_r$: we relabel the goal distribution by using the parameters $\omega' = h(\mathbf{s}_r)$.

Multi-task exploration scheme For training, in 50% of exploration rollouts we randomly selected a single subtask for the entire rollout, and in the other 50% of exploration rollouts we sequentially switched the subtask throughout the rollout, evenly allocating time to each subtask. For switching the subtask, we simply switched the parameters of the subtask μ and Σ. We randomized the order of the subtasks for sequential rollouts.

Relabeling We relabel the parameters of the goal distribution (μ, Σ), and the relabeling strategy we use depends on whether we use conditional goal distributions. For non-conditional distributions, we relabel according to the following strategy:

- 40%: relabel μ to a future state along the same collected trajectory

For conditional distributions, we relabel according to the following strategy:

- 40%: randomly sample $\mathbf{s}_f$ from the environment
- 40%: relabel $\mathbf{s}_f$ to a future state along the same collected trajectory

For our multi-task experiments, whenever we perform relabeling, we also relabel Σ. Specifically, we first randomly sample a task from the set of tasks that we have inferred, and relabel Σ to the covariance matrix for that task.

E.3 GCRL

Our implementation of goal-conditioned RL follows from hindsight experience replay (HER) [11]. Crucially, we perform off-policy RL, in addition to using the relabeling strategies inspired by HER. When provided a batch of data to train on, we relabel the goals according to the following strategy:

- 40%: randomly sampled goals from the environment, or the example sets
- 40%: future states along the same collected trajectory, as dictated by HER

Unlike HER, which used sparse rewards, we use the Euclidean distance as the basis for our reward function:

$$r(\mathbf{s}, \mathbf{s}_g) = -\left\|\mathbf{s} - \mathbf{s}_g\right\| \tag{4}$$

To avoid manual engineering, the space of goals is the same as the space of states. I.e., the dimension of the goal is the same as that of the state, and the corresponding entries in $\mathbf{s}$ and $\mathbf{s}_g$ correspond to the same semantic state features.

E.4 VICE

Variational inverse control with events (VICE) is described in Fu et al. [21]. VICE proposes an inverse reinforcement learning method that extends adversarial inverse reinforcement learning (AIRL) Fu et al. [47]. Like AIRL, VICE learns a density $p_\theta(s, a)$ using a classification problem. However, unlike the usual IRL setting, VICE assumes access to an example set that specifies the task – the same assumption as DisCo RL.

VICE alternates between two phases: updating the reward and running RL. To learn a reward function, VICE solves a classification problem, considering the initial example set as positives and samples from the replay buffer as negatives. The discriminator is:

$$D_\theta(s, a) = \frac{p_\theta(s, a)}{p_\theta(s, a) + \pi(a|s)}. \tag{5}$$

At optimality, the reward recovered $p_\theta(s, a) \propto \pi^*(a|s) = \exp(A(s, a))$, the advantage of the optimal policy [47]. In practice the reward is represented as $p_\theta(s)$, ignoring the dependence on actions. However, actions are still needed to compute the discriminator logits; we follow the method specified in VICE-RAQ [43] to sample actions from $\pi(a|s)$ for all states. We also use mixup [48] as described in VICE-RAQ. Mixup significantly reduces overfitting and allows VICE to successfully learn a neural net classifier even with so few (30 to 50) positive examples. In the RL phase, VICE runs reinforcement learning with $\log p_\theta(s, a)$ as the reward function, actively collecting more samples to use as negatives.

We re-implemented VICE as above and confirmed that it successfully learns policies to reach a single state, specified by examples. However, our results demonstrate that VICE struggles to reach the example sets we use in this work. This issue is exacerbated when the problem is multi-task instead of single-task and the state includes a goal, as in goal-conditioned learning.

In multi-stage tasks, we train a single DisCo RL policy shared among the stages. For VICE, sharing data among different tasks would not respect the adversarial optimization performed by the method. Instead, we train separate policies for each stage without sharing data between policies. Thus, for a task with N stages, VICE is generously allowed to experience $N\times$ the data, as each policy collects its own experience).