Replacing Rewards with Examples: Example-Based Policy Search via Recursive Classification

Benjamin Eysenbach[1 2] **Sergey Levine**[2 3] **Ruslan Salakhutdinov**[1]
[1]Carnegie Mellon University, [2]Google Brain, [3]UC Berkeley
beysenba@cs.cmu.edu

Abstract

Reinforcement learning (RL) algorithms assume that users specify tasks by manually writing down a reward function. However, this process can be laborious and demands considerable technical expertise. Can we devise RL algorithms that instead enable users to specify tasks simply by providing examples of successful outcomes? In this paper, we derive a control algorithm that maximizes the future probability of these successful outcome examples. Prior work has approached similar problems with a two-stage process, first learning a reward function and then optimizing this reward function using another RL algorithm. In contrast, our method directly learns a value function from transitions and successful outcomes, without learning this intermediate reward function. Our method therefore requires fewer hyperparameters to tune and lines of code to debug. We show that our method satisfies a new data-driven Bellman equation, where examples take the place of the typical reward function term. Experiments show that our approach outperforms prior methods that learn explicit reward functions.[1]

1 Introduction

In supervised learning settings, tasks are defined by data: what causes a car detector to detect cars is not the choice of loss function (which might be the same as for an airplane detector), but the choice of training data. Defining tasks in terms of data, rather than specialized loss functions, arguably makes it easier to apply machine learning algorithms to new domains. In contrast, reinforcement learning (RL) problems are typically posed in terms of reward functions, which are typically manually designed. Arguably, the challenge of designing reward functions has limited RL to applications with simple reward functions, and has been restricted to users who speak this language of mathematically-defined reward functions. Can we make task specification in RL similarly data-driven?

Whereas the standard MDP formalism centers around predicting and maximizing the future reward, we will instead focus on the problem classifying whether a task will be solved in the future. The user will provide a collection of example success states, not a reward function. We call this problem setting *example-based control*. In effect, these examples tell the agent "What would the world look like if the task were solved?" For example, for the task of opening a door, success examples correspond to different observations of the world when the door is open. The user can find examples of success even for tasks that they themselves do not know how to solve. For example, the user could solve the task using actions unavailable to the agent (e.g., the user may have two arms, but a robotic agent may have only one) or the user could find success examples by searching the internet. As we will discuss in Sec. 3.1, this problem setting is different from imitation learning: we maximize a different objective function and only assume access to success examples, not entire expert trajectories.

[1]Project site with videos and code: https://ben-eysenbach.github.io/rce

35th Conference on Neural Information Processing Systems (NeurIPS 2021), Sydney, Australia.

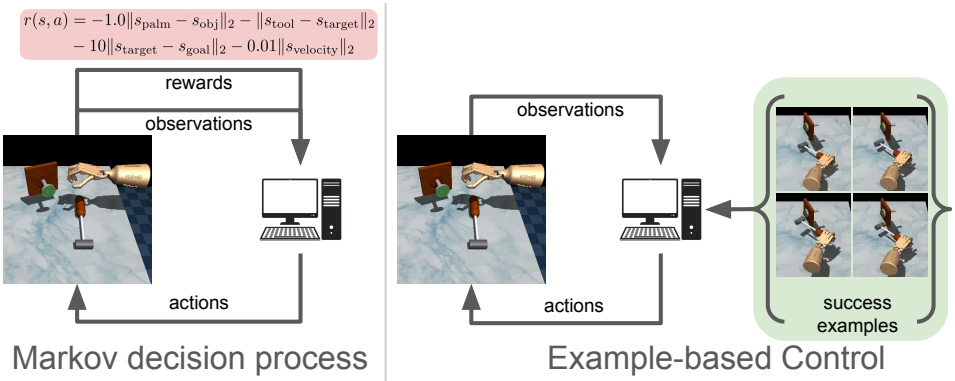


Figure 1: **Example-based control**: Whereas the standard MDP framework requires a user-defined reward function, example-based control specifies tasks via a handful of user-provided success examples.

Learning from examples is challenging because we must automatically identify when the agent has solved the task and reward it for doing so. Prior methods (either imitation learning from demonstrations or learning from success examples) take an indirect approach that resembles inverse RL: first learn a separate model to represent the reward function, and then optimize this reward function with standard RL algorithms. Our method is different from these prior methods because it learns to predict *future* success directly from transitions and success examples, without learning a separate reward function. This key difference has important algorithmic, theoretical, and empirical benefits. **Algorithmically**, our end-to-end approach removes potential biases in learning a separate reward function, reduces the number of hyperparameters, and simplifies the resulting implementation. **Theoretically**, we propose a method for classifying *future* events using a variant of temporal difference learning that we call recursive classification. This method satisfies a new Bellman equation, where success examples are used in place of the standard reward function term. We use this result to provide convergence guarantees. **Empirically**, we demonstrate that our method solves many complex manipulation tasks that prior methods fail to solve.

Our paper also addresses a subtle but important ambiguity in formulating example-based control. Some states might always solve the task while other states might rarely solve the task. But, without knowing how often the user visited each state, we cannot determine the likelihood that each state solves the task. Thus, an agent can only estimate the probability of success by making an additional assumption about how the success examples were generated. We will discuss two choices of assumptions. The first choice of assumption is convenient from an algorithmic perspective, but is sometimes violated in practice. A second choice is a worst-case approach, resluting in a problem setting that we call *robust example-based control*. Our analysis shows that the robust example-based control objective is equivalent to minimizing the squared Hellinger distance (an f-divergence).

In summary, this paper studies a data-driven framing of control, where reward functions are replaced by examples of successful outcomes. Our main contribution is an algorithm for off-policy example-based control. The key idea of the algorithm is to directly learn to predict whether the task will be solved in the future via *recursive classification*, without using separate reward learning and policy search procedures. Our analysis shows that our method satisfies a new Bellman equation where rewards are replaced by data (examples of success). Empirically, our method significantly outperforms state-of-the-art imitation learning methods (AIRL [7], DAC [17], and SQIL [28]) and recent methods that learn reward functions (ORIL [41], PURL [38], and VICE [8]). Our method completes complex tasks, such as picking up a hammer to knock a nail into a board, tasks that *none* of these baselines can solve. Using tasks with image observations, we demonstrate agents learned with our method acquire a notion of success that generalizes to new environments with varying shapes and goal locations.

2 Related Work

Learning reward functions. Prior works have studied RL in settings where the task is specified either with examples of successful outcomes or complete demonstrations. These prior methods typically learn a reward function from data and then apply RL to this reward function (e.g., Fu et al. [8], Ziebart et al. [40]). Most inverse RL algorithms adopt this approach [1, 25, 27, 29, 36, 40], as do more recent methods that learn a *success classifier* to distinguishing successful outcomes from random states [8, 16, 31, 41]. Prior adversarial imitation learning methods [7, 12] can be

viewed as *iteratively* learning a success classifier. Recent work in this area focuses on extending these methods to the offline setting [16, 41], incorporating additional sources of supervision [42], and learning the classifier via positive-unlabeled classification [13, 38, 41]. Many prior methods for robot learning have likewise used a classifier to distinguish success examples [2, 22, 34, 37]. Unlike these prior methods, our approach only requires examples of successful outcomes (not expert trajectories) and does not learn a separate reward function. Instead, our method learns a value function directly from examples, effectively "cutting out the middleman." This difference from prior work removes hyperparameters and potential bugs associated with learning a success classifier. Empirically, we demonstrate that our end-to-end approach outperforms these prior two-stage approaches. See Appendix D for more discussion of the relationship between our method and prior work.

Imitation learning without auxiliary classifiers. While example-based control is different from imitation learning, our method is similar to two prior imitation learning methods that likewise avoid learning a separate reward function [18, 28]. ValueDICE [18], a method based on convex duality, uses full expert demonstrations for imitation learning. In contrast, our method learns from success examples, which are typically easier to provide than full expert demonstrations. SQIL [28] is a modification of SAC [11] that labels success examples with a reward of $+1$. The mechanics of our method are similar to SQIL [28], but key algorithmic differences (backed by stronger theoretical guarantees) result in better empirical performance. Our analysis in Sec. 4.2 highlights connections and differences between imitation learning and example-based control.

Goal-conditioned RL. Goal-conditioned RL provides one way to specify tasks in terms of data, and prior work has shown how goal-conditioned policies can be learned directly from data, without a reward function [4, 15, 21, 30]. However, the problem that we study in this paper, example-based control, is different from goal-conditioned RL because it allows users to indicate that tasks can be solved in many ways, enabling the agent to learn a more general notion of success. Perhaps the most similar prior work in this area is C-learning [4], which uses a temporal-difference update to learn the probability that a goal state will be reached in the future. Our method will use a similar temporal-difference update to predict whether *any* success example will be reached in the future. Despite the high-level similarity with C-learning, our algorithm will be markedly different; for example, our method does not require hindsight relabeling, and learns a single policy rather than a goal-conditioned policy.

3 Example-Based Control via Recursive Classification

We aim to learn a policy that reaches states that are likely to solve the task (see Fig. 1), without relying on a reward function. We start by formally describing this problem, which we will call *example-based control*. We then propose a method for solving this problem and provide convergence guarantees.

3.1 Problem Statement

Example-based control is defined by a controlled Markov process (i.e., an MDP without a reward function) with dynamics $p(\mathbf{s_{t+1}} \mid \mathbf{s_t}, \mathbf{a_t})$ and an initial state distribution $p_1(\mathbf{s_1})$, where $\mathbf{s_t} \in \mathcal{S}$ and $\mathbf{a_t}$ denote the time-indexed states and actions. The variable $\mathbf{s_{t+\Delta}}$ denotes a state Δ steps in the future.

The agent is given a set of *success examples*, $\mathcal{S}^* = \{\mathbf{s}^*\} \subseteq \mathcal{S}$. The random variable $\mathbf{e_t} \in \{0, 1\}$ indicates whether the task is solved at time t, and $p(\mathbf{e_t} \mid \mathbf{s_t})$ denotes the probability that the current state $\mathbf{s_t}$ solves the task. Given a policy $\pi_\phi(\mathbf{a_t} \mid \mathbf{s_t})$, we define the discounted future state distribution:

$$p^\pi(\mathbf{s_{t+}} \mid \mathbf{s_t}, \mathbf{a_t}) \triangleq (1-\gamma) \sum_{\Delta=0}^{\infty} \gamma^\Delta p^\pi(\mathbf{s_{t+\Delta}} = \mathbf{s_{t+}} \mid \mathbf{s_t}, \mathbf{a_t}). \tag{1}$$

Using this definition, we can write the probability of solving the task at a *future* step as

$$p^\pi(\mathbf{e_{t+}} \mid \mathbf{s_t}, \mathbf{a_t}) \triangleq \mathbb{E}_{p^\pi(\mathbf{s_{t+}} \mid \mathbf{s_t}, \mathbf{a_t})}[p(\mathbf{e_{t+}} \mid \mathbf{s_{t+}})]. \tag{2}$$

Example-based control maximizes the probability of solving the task in the (discounted) future:

Definition 1 (Example-based control)**.** *Given a controlled Markov process and distribution over success examples $p(\mathbf{s_t} \mid \mathbf{e_t} = 1)$, the example-based control problem is to find the policy that optimizes the likelihood of solving the task:*

$$\arg\max_\pi p^\pi(\mathbf{e_{t+}} = 1) = \mathbb{E}_{p_1(\mathbf{s_1}), \pi(\mathbf{a_1} \mid \mathbf{s_1})}\left[p^\pi(\mathbf{e_{t+}} = 1 \mid \mathbf{s_1}, \mathbf{a_1})\right].$$

Although this objective is equivalent to the RL objective with rewards $r(\mathbf{s_t}, \mathbf{a_t}) = p(\mathbf{e_t} = 1 \mid \mathbf{s_t})$, we assume the probabilities $p(\mathbf{e_t} \mid \mathbf{s_t})$ are unknown. Instead, we assume that we have samples of successful states, $\mathbf{s}^* \sim p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)$. Example-based control differs from imitation learning because imitation learning requires full expert demonstrations. In the special case where the user provides a single success state, example-based control is equivalent to goal-conditioned RL.

Since interacting with the environment to collect experience is expensive in many settings, we define *off-policy example-based control* as the version of this problem where the agent learns from environment interactions collected from other policies. In this setting, the agent learns from two distinct datasets: **(1)** transitions, $\{(\mathbf{s_t}, \mathbf{a_t}, \mathbf{s_{t+1}}) \sim p_U(\mathbf{s_t}, \mathbf{a_t}, \mathbf{s_{t+1}})\}$, which contain information about the environment dynamics; and **(2)** success examples, $\mathcal{S}^* = \{\mathbf{s}^* \sim p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)\}$, which specify the task that the agent should attempt to solve. Our analysis will assume that these two datasets are fixed. The main contribution of this paper is an algorithm for off-policy example-based control.

An assumption on success examples. The probability of solving the task at state $\mathbf{s_t}$, $p(\mathbf{e_t} = 1 \mid \mathbf{s_t})$, cannot be uniquely determined from success examples and transitions alone. To explain this ambiguity, we define $p_U(\mathbf{s_t})$ as the state distribution visited by the user; note that the user may be quite bad at solving the task themselves. Then, the probability of solving the task at state $\mathbf{s_t}$ depends on how often a user visits state $\mathbf{s_t}$ versus how often the task is solved when visiting state $\mathbf{s_t}$:

$$p(\mathbf{e_t} = 1 \mid \mathbf{s_t}) = \frac{p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)}{p_U(\mathbf{s_t})} p_U(\mathbf{e_t} = 1). \tag{3}$$

For example, the user may complete a task using two strategies, but we cannot determine which of these strategies is more likely to succeed unless we know how often the user attempted each strategy. Thus, any method that learns from success examples *must* make an additional assumption on $p_U(\mathbf{s_t})$. We will discuss two choices of assumptions. The **first choice** is to assume that the user visited states with the same frequency that they occur in the dataset of transitions. That is,

$$p_U(\mathbf{s_t}) = \iint p_U(\mathbf{s_t}, \mathbf{a_t}, \mathbf{s_{t+1}}) d\mathbf{a_t} d\mathbf{s_{t+1}}. \tag{4}$$

Intuitively, this assumption implies that the user has the same capabilities as the agent. Prior work makes this same assumption without stating it explicitly [8, 24, 31]. Experimentally, we find that our method succeeds even in cases where this assumption is violated.

However, many common settings violate this assumption, especially when the user has different dynamics constraints than the agent. For example, a human user collecting success examples for a cleaning task might usually put away objects on a shelf at eye-level, whereas transitions collected by a robot interact with the ground-level shelves more frequently. Under our previous assumption, the robot would assume that putting objects away on higher shelves is more satisfactory than putting them away on lower shelves, even though doing so might be much more challenging for the robot. To handle these difference in capabilities, the **second choice** is to use a *worst-case* formulation, which optimizes the policy to be robust to *any* choice of $p_U(\mathbf{s_t})$. Surprisingly, this setting admits a tractable solution, as we discuss in Sec. 4.2.

3.2 Predicting Future Success by Recursive Classification

We now describe our method for example-based control. We start with the more standard **first choice** for the assumption on $p_U(\mathbf{s_t})$ (Eq. 4); we discuss the second choice in Sec. 4.2. Our approach estimates the probability in Eq. 2 indirectly via a future success classifier. This classifier, $C^\pi_\theta(\mathbf{s_t}, \mathbf{a_t})$, discriminates between "positive" state-action pairs which lead to successful outcomes (i.e., sampled from $p(\mathbf{s_t}, \mathbf{a_t} \mid \mathbf{e_{t+}} = 1)$) and random "negatives" (i.e., sampled from the marginal distribution $p(\mathbf{s_t}, \mathbf{a_t})$). We will use different class-specific weights, using a weight of $p(\mathbf{e_{t+}} = 1)$ for the "positives" and a weight of 1 for the "negatives." Bayes-optimal classifier is

$$C^\pi_\theta(\mathbf{s_t}, \mathbf{a_t}) = \frac{p^\pi(\mathbf{s_t}, \mathbf{a_t} \mid \mathbf{e_{t+}} = 1) p(\mathbf{e_{t+}} = 1)}{p^\pi(\mathbf{s_t}, \mathbf{a_t} \mid \mathbf{e_{t+}} = 1) p(\mathbf{e_{t+}} = 1) + p(\mathbf{s_t}, \mathbf{a_t})}. \tag{5}$$

These class specific weights let us predict the probability of future success using the optimal classifier:

$$\frac{C^\pi_\theta(\mathbf{s_t}, \mathbf{a_t})}{1 - C^\pi_\theta(\mathbf{s_t}, \mathbf{a_t})} = p^\pi(\mathbf{e_{t+}} = 1 \mid \mathbf{s_t}, \mathbf{a_t}). \tag{6}$$

Algorithm 1 Recursive Classification of Examples

Input: success examples $\mathcal{S}^*$
Initialize policy $\pi_\phi(\mathbf{a_t} \mid \mathbf{s_t})$, classifier $C_\theta^\pi(\mathbf{s_t}, \mathbf{a_t})$, replay buffer $\mathcal{D}$
while not converged **do**
  Collect a new trajectory: $\mathcal{D} \leftarrow \mathcal{D} \cup \{\tau \sim \pi_\phi\}$
  Sample success examples: $\{\mathbf{s_t}^{(1)} \sim \mathcal{S}^*, \mathbf{a_t}^{(1)} \sim \pi_\phi(\mathbf{a_t} \mid \mathbf{s_t}^{(1)})\}$
  Sample transitions: $\{(\mathbf{s_t}^{(2)}, \mathbf{a_t}^{(2)}, \mathbf{s_{t+1}}) \sim \mathcal{D}, \mathbf{a_{t+1}} \sim \pi_\phi(\mathbf{a_{t+1}} \mid \mathbf{s_{t+1}})\}$
  $w \leftarrow \frac{C_\theta^\pi(\mathbf{s_{t+1}}, \mathbf{a_{t+1}})}{1 - C_\theta^\pi(\mathbf{s_{t+1}}, \mathbf{a_{t+1}})}$ ▷ Eq. 9
  $\mathcal{L}(\theta) \leftarrow (1-\gamma)\mathcal{CE}(C_\theta(\mathbf{s_t}^{(1)}, \mathbf{a_t}^{(1)}); y = 1) + (1+\gamma w)\mathcal{CE}(C_\theta(\mathbf{s_t}^{(2)}, \mathbf{a_t}^{(2)}); y = \frac{\gamma w}{1+\gamma w})$
  Update classifier: $\theta \leftarrow \theta + \eta \nabla_\theta \mathcal{L}(\theta)$ ▷ Eq. 8
  Update policy: $\phi \leftarrow \phi + \eta \nabla_\phi \mathbb{E}_{\pi_\phi(\mathbf{a_t}|\mathbf{s_t})}[C_\theta(\mathbf{s_t}, \mathbf{a_t})]$
return π_ϕ

Importantly, the resulting method will not actually require estimating the weight $p(\mathbf{e_{t+}} = 1)$. We would like to optimize the classifier parameters using maximum likelihood estimation:

$$\mathcal{L}^\pi(\theta) \triangleq p(\mathbf{e_{t+}} = 1)\, \mathbb{E}_{p(\mathbf{s_t}, \mathbf{a_t} | \mathbf{e_{t+}} = 1)}[\log C_\theta^\pi(\mathbf{s_t}, \mathbf{a_t})] + \mathbb{E}_{p(\mathbf{s_t}, \mathbf{a_t})}[\log(1 - C_\theta^\pi(\mathbf{s_t}, \mathbf{a_t}))]. \quad (7)$$

However, we cannot directly optimize this objective because we cannot sample from $p(\mathbf{s_t}, \mathbf{a_t} \mid \mathbf{e_{t+}} = 1)$. We convert Eq. 7 into an equivalent loss function that we can optimize using three steps; see Appendix A for a detailed derivation. The **first step** is to factor the distribution $p(\mathbf{s_t}, \mathbf{a_t}, \mathbf{e_{t+}} = 1)$. The **second step** is to decompose $p^\pi(\mathbf{e_{t+}} = 1 \mid \mathbf{s_t}, \mathbf{a_t})$ into two terms, corresponding to the probabilities of solving the task at time $t' = t + 1$ and time $t' > t + 1$. We can estimate the probability of solving the task at the next time step using the set of success examples. The **third step** is to estimate the probability of solving the task at time $t' > t + 1$ by evaluating the classifier at the next time step. Combining these three steps, we can *equivalently* express the objective function in Eq. 7 using off-policy data:

$$\mathcal{L}^\pi(\theta) = (1-\gamma)\mathbb{E}_{\substack{p_U(\mathbf{s_t}|\mathbf{e_t}=1) \\ p(\mathbf{a_t}|\mathbf{s_t})}}[\underbrace{\log C_\theta^\pi(\mathbf{s_t}, \mathbf{a_t})}_{(a)}] + \mathbb{E}_{p(\mathbf{s_t}, \mathbf{a_t}, \mathbf{s_{t+1}})}[\underbrace{\gamma w \log C_\theta^\pi(\mathbf{s_t}, \mathbf{a_t})}_{(b)} + \underbrace{\log(1 - C_\theta^\pi(\mathbf{s_t}, \mathbf{a_t}))}_{(c)}], \quad (8)$$

where

$$w = \mathbb{E}_{p(\mathbf{a_{t+1}}|\mathbf{s_{t+1}})}\left[\frac{C_\theta^\pi(\mathbf{s_{t+1}}, \mathbf{a_{t+1}})}{1 - C_\theta^\pi(\mathbf{s_{t+1}}, \mathbf{a_{t+1}})}\right] \quad (9)$$

is the classifier's prediction (ratio) at the next time step. Our resulting method can be viewed as a temporal difference [33] approach to classifying future events. We will refer to our method as **recursive classification of examples (RCE)**. This equation has an intuitive interpretation. The first term *(a)* trains the classifier to predict 1 for the success examples themselves, and the third term *(c)* trains the classifier to predict 0 for random transitions. The important term is the second term *(b)*, which is analogous to the "bootstrapping" term in temporal difference learning [32]. Term *(b)* indicates that the probability of future success depends on the probability of success at the *next* time step, as inferred using the classifier's own predictions.

Our resulting method is similar to existing actor-critic RL algorithms. To highlight the similarity to existing actor-critic methods, we can combine the *(b)* and *(c)* terms in the classifier objective function (Eq. 8) to express the loss function in terms of two cross entropy losses:

$$\min_\theta\, (1-\gamma)\mathbb{E}_{\substack{p(\mathbf{s_t}|\mathbf{e_t}=1), \\ \mathbf{a_t} \sim \pi(\mathbf{a_t}|\mathbf{s_t})}}[\mathcal{CE}(C_\theta^\pi(\mathbf{s_t}, \mathbf{a_t}); y = 1)] + (1+\gamma w)\mathbb{E}_{p(\mathbf{s_t}, \mathbf{a_t}, \mathbf{s_{t+1}})}\Big[\mathcal{CE}\big(C_\theta^\pi(\mathbf{s_t}, \mathbf{a_t}); y = \frac{\gamma w}{\gamma w + 1}\big)\Big]. \quad (10)$$

These cross entropy losses update the classifier to predict $y = 1$ for the success examples and to predict $y = \frac{\gamma w}{1+\gamma w}$ for other states.

Algorithm summary. Alg. 1 summarizes our method, which alternates between updating the classifier, updating the policy, and (optionally) collecting new experience. We update the policy to choose actions that maximize the classifier's confidence that the task will be solved in the future: $\max_\phi \mathbb{E}_{\pi_\phi(\mathbf{a_t}|\mathbf{s_t})}[C_\theta^\pi(\mathbf{s_t}, \mathbf{a_t})]$. Following prior work [5, 35]), we regularized the policy updates by adding an entropy term with coefficient $\alpha = 10^{-4}$. We also found that using N-step returns significantly improved the results of RCE (see Appendix F for details and ablation experiments.). Implementing our method on top of existing methods such as SAC [11] or TD3 [9] requires only changing the standard Bellman loss with the loss in Eq. 10. See Appendix E for implementation details; code is available on the project website.

4 Analysis

In this section, we prove that RCE satisfies many of the same convergence and optimality guarantees (for example-based control) that standard RL algorithms satisfy (for reward-based MDPs). These results are important as they demonstrate that formulating control in terms of data, rather than rewards, does preclude algorithms from enjoying strong theoretical guarantees. Proofs of all results are given in Appendix B, and we include a further discussion of how RCE relates to prior work in Appendix D.

4.1 Bellman Equations and Convergence Guarantees

To prove that RCE converges to the optimal policy, we will first show that RCE satisfies a new Bellman equation:

Lemma 4.1. *The Bayes-optimal classifier C^π for policy π satisfies the following identity:*

$$\frac{C^\pi(\mathbf{s_t}, \mathbf{a_t})}{1 - C^\pi(\mathbf{s_t}, \mathbf{a_t})} = (1-\gamma)p(\mathbf{e_t} = 1 \mid \mathbf{s_t}) + \gamma \mathbb{E}_{\substack{p(\mathbf{s_{t+1}}|\mathbf{s_t},\mathbf{a_t}) \\ \pi(\mathbf{a_{t+1}}|\mathbf{s_{t+1}})}} \left[\frac{C^\pi(\mathbf{s_{t+1}}, \mathbf{a_{t+1}})}{1 - C^\pi(\mathbf{s_{t+1}}, \mathbf{a_{t+1}})} \right]. \tag{11}$$

The proof combines the definition of the Bayes-optimal classifier with the assumption from Eq. 4. This Bellman equation is analogous to the standard Bellman equation for Q-learning, where the reward function is replaced by $(1-\gamma)p(\mathbf{e_t} = 1 \mid \mathbf{s_t})$ and the Q function is parametrized as $Q^\pi_\theta(\mathbf{s_t}, \mathbf{a_t}) = \frac{C^\pi_\theta(\mathbf{s_t},\mathbf{a_t})}{1-C^\pi_\theta(\mathbf{s_t},\mathbf{a_t})}$. While we do not know how to compute this reward function, the update rule for RCE is equivalent to doing value iteration using that reward function and that parametrization of the Q-function:

Lemma 4.2. *In the tabular setting, the **expected** updates for RCE are equivalent to doing value iteration with the reward function $r(\mathbf{s_t}) = (1-\gamma)p(\mathbf{e_t} = 1 \mid \mathbf{s_t})$ and a Q-function parametrized as $Q^\pi_\theta(\mathbf{s_t}, \mathbf{a_t}) = \frac{C^\pi_\theta(\mathbf{s_t},\mathbf{a_t})}{1-C^\pi_\theta(\mathbf{s_t},\mathbf{a_t})}$.*

This result tells us that RCE is equivalent to maximizing the reward function $(1-\gamma)p(\mathbf{e_t} = 1 \mid \mathbf{s_t})$; however, RCE does not require knowing $p(\mathbf{e_t} = 1 \mid \mathbf{s_t})$, the probability that each state solves the task. Since value iteration converges in the tabular setting, an immediate consequence of Lemma 4.2 is that tabular RCE also converges:

Corollary 4.2.1. *RCE converges in the tabular setting.*

So far we have analyzed the training process for the classifier for a fixed policy. We conclude this section by showing that optimizing the policy w.r.t. the classifier improves the policy's performance.

Lemma 4.3. *Let policy $\pi(\mathbf{a_t} \mid \mathbf{s_t})$ and success examples $\mathcal{S}^*$ be given, and let $C^\pi(\mathbf{s_t}, \mathbf{a_t})$ denote the corresponding Bayes-optimal classifier. Define the improved policy as acting greedily w.r.t. C^π: $\pi'(\mathbf{a_t} \mid \mathbf{s_t}) = \mathbb{1}(\mathbf{a} = \arg\max_a C^\pi(\mathbf{s_t}, \mathbf{a}))$. Then the improved policy is at least as good as the old policy at solving the task: $p^{\pi'}(\mathbf{e_{t+}} = 1) \geq p^\pi(\mathbf{e_{t+}} = 1)$.*

4.2 Robust Example-based Control

In this section, we derive a principled solution for the case where $p_U(\mathbf{s_t})$ is not known, which will correspond to modifying the objective function for example-based control. However, we will argue that, in some conditions, the method proposed in Sec. 3.2 is *already* robust to unknown $p_U(\mathbf{s_t})$, if that method is used with online data collection. The goal of this discussion is to provide a theoretical relationship between our method and a robust version of example-based control that makes fewer assumptions about $p_U(\mathbf{s_t})$. This discussion will also clarify how changing assumptions on the user's capabilities can change the optimal policy.

When introducing example-based control in Sec. 3.1, we emphasized that we *must* make an assumption to make the example-based control problem well defined. The exact probability that a success example solves the task depends on how often the user visited that state, which the agent does not know. Therefore, there are many valid hypotheses for how likely each state is to solve the task. We can express the set of valid hypotheses using Bayes' Rule:

$$\mathcal{P}_{\mathbf{e_t}|\mathbf{s_t}} \triangleq \left\{ \hat{p}(\mathbf{e_t} = 1 \mid \mathbf{s_t}) = \frac{p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)p(\mathbf{e_t} = 1)}{p_U(\mathbf{s_t})} \right\}.$$

Previously (Sec. 3.2), we resolved this ambiguity by assuming that $p_U(\mathbf{s_t})$ was equal to the distribution over states in our dataset of transitions. As discussed in Sec. 3.1, many problem settings violate this assumption, prompting us to consider the more stringent setting with no prior information about $p_U(\mathbf{s_t})$ (e.g., no prior knowledge about the user's capabilities). To address this settting, we will assume the *worst* possible choice of $p_U(\mathbf{s_t})$. This approach will make the agent robust to imperfect knowledge of the user's abilities and to mislabeled success examples. Formally, we define the *robust example-based control* problem as

$$\max_{\pi} \min_{\hat{p}(\mathbf{e_t}|\mathbf{s_t}) \in \mathcal{P}_{\mathbf{e_t}|\mathbf{s_t}}} \mathbb{E}_{p^\pi(\mathbf{s_{t+}})}[\hat{p}(\mathbf{e_{t+}} = 1 \mid \mathbf{s_{t+}})] = \max_{\pi} \min_{p_U(\mathbf{s_t})} \mathbb{E}_{p^\pi(\mathbf{s_{t+}})} \left[\frac{p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)}{p_U(\mathbf{s_t})} p(\mathbf{e_t} = 1) \right]. \tag{12}$$

This objective can be understood as having the adversary assign a weight of $1/p_U(\mathbf{s_t})$ to each success example. The optimal adversary will assign lower weights to success examples that the policy frequently visits and higher weights to less-visited success examples. Intuitively, the optimal policy should try to reach many of the success examples, not just the ones that are easiest to reach. Thus, such a policy will continue to succeed even if certain success examples are removed, or are later discovered to have been mislabeled. Surprisingly, solving this two-player game corresponds to minimizing an f-divergence:

Lemma 4.4. *Define $H^2[p(\mathbf{x}), q(\mathbf{x})] = \int (\sqrt{p(\mathbf{x})} - \sqrt{q(\mathbf{x})})^2 d\mathbf{x}$ as the squared Hellinger distance, an f-divergence. Robust example-based control (Eq. 12) is equivalent to minimizing the squared Hellinger distance between policy's discounted state occupancy measure and the* ***conditional*** *distribution $p(\mathbf{s_t} \mid \mathbf{e_t} = 1)$:*

$$\min_{\hat{p}(\mathbf{e_t}|\mathbf{s_t}) \in \mathcal{P}_{\mathbf{e_t}|\mathbf{s_t}}} p^{\pi,\hat{p}}(\mathbf{e_{t+}}) = 1 - \frac{1}{2} H^2[p(\mathbf{s_t}|\mathbf{e_t} = 1), p^\pi(\mathbf{s_{t+}} = \mathbf{s_t})].$$

The main idea of the proof (found in Appendix C) is to compute the worst-case distribution $p_U(\mathbf{s_t})$ using the calculus of variations. Preliminary experiments (Fig. 5 in Appendix C) show that a version of RCE with online data collection finds policies that perform well on the robust example-based control objective (Eq. 12). In fact, under somewhat stronger assumptions, we can show that the solution of robust example-based control is a fixed point of *iterated* RCE (see Appendix C.2). Therefore, in our experiments, we use RCE with online data collection.

5 Experiments

Our experiments study how effectively RCE solves example-based control tasks, especially in comparison to prior methods that learn an explicit reward function. Both RCE and the prior methods receive only the success examples as supervision; no method has access to expert trajectories of reward functions. Additional experiments in Sec. 5.2 study whether RCE can solve tasks using image observations. These experiments test whether RCE can solve tasks in new environments that are different from those where the success examples were collected, and test whether RCE learns policies that learn a general notion of success rather than just memorizing the success examples. We include videos of learned policies online[2] and include implementation details, hyperparameters, ablation experiments, and a list of failed experiments in the Appendix.

We compare RCE against prior methods that infer a reward function from the success examples and then apply an off-the-shelf RL algorithm; some baselines iterate between these two steps. AIRL [7] is a popular adversarial imitation learning method. VICE [8] is the same algorithm as AIRL, but intended to be applied to success examples rather than full demonstrations. We will label this method as "VICE" in figures, noting that it is the same algorithm as AIRL. DAC [17] is a more recent, off-policy variant of AIRL. We also compared against two recent methods that learn rewards from *demonstrations*: ORIL [41] and PURL [38]. Following prior work [16], we also compare against "frozen" variants of some baselines that first train the parameters of the reward function and then apply RL to that reward function without updating the parameters of the reward function again. Our method differs from these baselines in that we do not learn a reward function from the success examples and then apply RL, but rather learn a policy directly from the success examples. Lastly, we compare against SQIL [28], an imitation learning method that assigns a reward of $+1$ to states from demonstrations and 0 to all other states. SQIL does not learn a separate reward function and structurally resembles our method, but is derived from different principles (see Sec. 2.).

[2] `https://ben-eysenbach.github.io/rce`

Figure 2: **Recursive Classification of Examples for learning manipulation tasks**: We apply RCE to a range of manipulation tasks, each accompanied with a dataset of success examples. For example, on the `sawyer_lift` task, we provide success examples where the object has been lifted above the table. We use the cumulative task return (↑ is better) solely for evaluation. Our method (blue line) outperforms prior methods across all tasks.

5.1 Evaluating RCE for Example-Based Control.

We evaluate each method on five Sawyer manipulation tasks from Meta-World [39] and two manipulation tasks from Rajeswaran et al. [26]. Fig. 3 illustrates these tasks. On each task, we provide the agent with 200 successful outcomes to define the task. For example, on the `open_drawer` task, these success examples show an opened drawer. As another example, on the `sawyer_push` task, success examples not only have the end effector touching the puck, but (more importantly) the puck position is different.

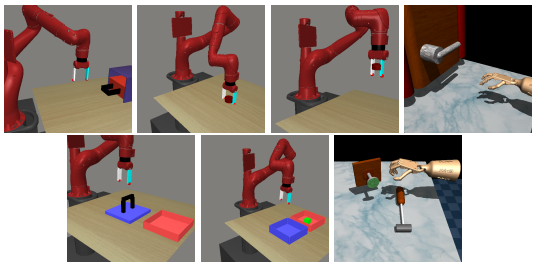


Figure 3: **Manipulation Environments**

We emphasize that these success examples only reflect the final state where the task is solved and are not full expert trajectories. This setting is important in practical use-cases: it is often easier for humans to arrange the workspace into a successful configuration than it is to collect an entire demonstration. See Appendix E.3 for details on how success examples were generated for each task. While these tasks come with existing user-defined reward functions, these rewards are not provided to any of the methods in our experiments and are used solely for evaluation (↑ is better). We emphasize that this problem setting is exceedingly challenging: the agent provided only with examples of success states (e.g., an observation where an object has been placed in the correct location). Most prior methods that tackle similar tasks employ hand-designed reward functions or distance functions, full demonstrations, or carefully-constructed initial state distributions.

The results in Fig. 2 show that RCE significantly outperforms prior methods across all tasks. The transparent lines indicate one random seed, and the darker lines are the average across random seeds. RCE solves many tasks, such as bin picking and hammering, that *none* of the baselines make *any* progress towards solving. The most competitive baseline, SQIL, only makes progress on the easiest two tasks; even on those tasks, SQIL learns more slowly than RCE and achieves lower asymptotic return. To check that all baselines are implemented correctly, we confirm that all can solve a very simple reaching task described in the next section.

5.2 Example-Based Control from Images

Our second set of experiments studies whether RCE can learn image-based tasks and assesses the generalization capabilities of our method. We designed three *image-based* manipulation tasks. The `reach_random_position` task entails reaching a red puck, whose position is randomized in each episode. The `reach_random_size` task entails reaching a red object, but the actual shape of that object varies from one episode to the next. Since the agent cannot change the size of the object and the size is randomized from one episode to the next, it is impossible to reach any of the previously-observed success examples. To solve this task, the agent must learn a notion of success that is more general than reaching a fixed goal state. The third task, `sawyer_clear_image`, entails clearing an object off the table, and is mechanically more challenging than the reaching tasks.

Fig. 4 shows results from these image-based experiments, comparing RCE to the same baselines. We observe that RCE has learned to solve both reaching tasks, reaching for the object regardless of the location and size of the object. This task is mechanically easier than the state-based tasks in Fig. 2,

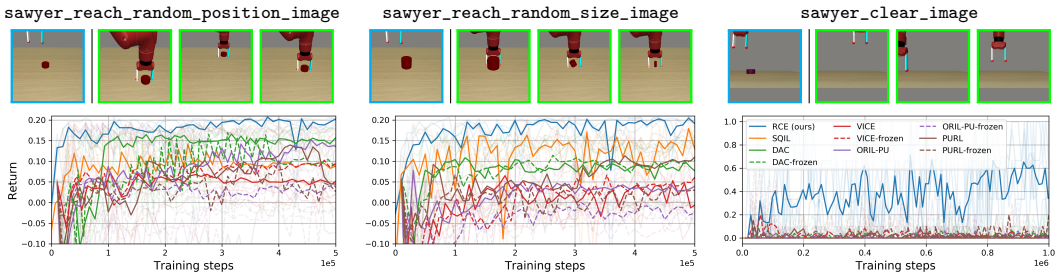


Figure 4: **Example-based control from images**: We evaluate RCE on three manipulation tasks using image-observations. *(Top)* We show examples of the initial state and success examples for each task. *(Bottom)* RCE (blue line) outperforms prior methods, especially on the more challenging clearing task. For the `random_size` task (*center*), this entails reaching for new objects that have different sizes from any seen in the success examples.

and all the baselines make some progress on this task, but learn more slowly than our method. The good performance of RCE on the `reach_random_size` task illustrates that RCE can solve tasks in a new environment, where the object size is different from that seen in the success examples. We hypothesize that RCE learns faster than these baselines because it "cuts out the middleman," learning a value function directly from examples rather than indirectly via a separate reward function. To support this hypothesis, we note SQIL, which also avoids learning a reward function, learns faster than other baselines on these tasks. On the more challenging clearing task, only our method makes progress, suggesting that RCE is a more effective algorithm for learning these image-based control tasks. In summary, these results show that RCE outperforms prior methods at solving example-based control tasks from image observations, and highlights that RCE learns a policy that solves tasks in new environments that look different from any of the success examples.

5.3 Ablation Experiments

We ran seven additional experiments to study the importance of hyperparameters and design decisions. Appendix F provides full details and figures. These experiments highlight that RCE is not an imitation learning method: RCE fails when applied to full expert trajectories, which are typically harder to provide than success examples. Other ablation experiments underscore the importance of using n-step returns and validate the approximation made in Sec. 3.2.

6 Conclusion

In this paper, we proposed a data-driven approach to control, where examples of success states are used in place of a reward function. Our method estimates the probability of reaching a success example in the future and optimizes a policy to maximize this probability of success. Unlike prior imitation learning methods, our approach is end-to-end and does not require learning a reward function. Our method is therefore simpler, with fewer hyperparameters and fewer lines of code to debug. Our analysis rests on a new data-driven Bellman equation, where example success states replace the typical reward function term. We use this Bellman equation to prove convergence of our classifier and policy. We believe that formulating control problems in terms of data, rather than the reward-centric MDP, better captures the essence of many real-world control problems and suggests a new set of attractive learning algorithms.

Limitations and future work. One limitation with RCE is that, despite producing effective policies, the classifier's predictions are not well calibrated. This issue resembles the miscalibration in Q-functions observed in prior work [9, 20]. Second, both RCE and the baselines we compared against all struggled to learn more challenging image-based tasks, such as image-based versions of the tasks shown in Fig. 3. Techniques such as explicit representation learning [19, 23] may be important for scaling example-based control algorithms to high-dimensional tasks.

Acknowledgements. This work is supported by the Fannie and John Hertz Foundation, NSF (DGE1745016, IIS1763562), ONR (N000141812861), and US Army. We thank Ilya Kostrikov for providing the implementation of DAC. We thank Ksenia Konyushkova, Konrad Zolna, and Justin Fu for discussions about baselines and prior work. We thank Oscar Ramirez for help setting up the image-based experiments, Ryan Julian and anonymous reviewers for feedback on drafts of the paper, and Alex Irpan for help releasing code.

References

[1] Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1.

[2] Calandra, R., Owens, A., Upadhyaya, M., Yuan, W., Lin, J., Adelson, E. H., and Levine, S. (2017). The feeling of success: Does touch sensing help predict grasp outcomes? In *Conference on Robot Learning*, pages 314–323. PMLR.

[3] Elkan, C. and Noto, K. (2008). Learning classifiers from only positive and unlabeled data. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 213–220.

[4] Eysenbach, B., Salakhutdinov, R., and Levine, S. (2021). C-learning: Learning to achieve goals via recursive classification. In *International Conference on Learning Representations*.

[5] Fox, R., Pakman, A., and Tishby, N. (2016). Taming the noise in reinforcement learning via soft updates. In *Proceedings of the Thirty-Second Conference on Uncertainty in Artificial Intelligence*, pages 202–211.

[6] Fu, J., Kumar, A., Nachum, O., Tucker, G., and Levine, S. (2020). D4rl: Datasets for deep data-driven reinforcement learning. *arXiv preprint arXiv:2004.07219*.

[7] Fu, J., Luo, K., and Levine, S. (2018a). Learning robust rewards with adverserial inverse reinforcement learning. In *International Conference on Learning Representations*.

[8] Fu, J., Singh, A., Ghosh, D., Yang, L., and Levine, S. (2018b). Variational inverse control with events: A general framework for data-driven reward definition. *Advances in neural information processing systems*, 31:8538–8547.

[9] Fujimoto, S., Hoof, H., and Meger, D. (2018). Addressing function approximation error in actor-critic methods. In *International Conference on Machine Learning*, pages 1587–1596. PMLR.

[10] Guadarrama, S., Korattikara, A., Ramirez, O., Castro, P., Holly, E., Fishman, S., Wang, K., Gonina, E., Wu, N., Kokiopoulou, E., Sbaiz, L., Smith, J., Bartók, G., Berent, J., Harris, C., Vanhoucke, V., and Brevdo, E. (2018). TF-Agents: A library for reinforcement learning in tensorflow. [Online; accessed 25-June-2019].

[11] Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International Conference on Machine Learning*, pages 1861–1870. PMLR.

[12] Ho, J. and Ermon, S. (2016). Generative adversarial imitation learning. In *NIPS*, pages 4565–4573.

[13] Irpan, A., Rao, K., Bousmalis, K., Harris, C., Ibarz, J., and Levine, S. (2019). Off-policy evaluation via off-policy classification. In *Advances in Neural Information Processing Systems*, pages 5437–5448.

[14] Jaakkola, T., Jordan, M. I., and Singh, S. P. (1994). On the convergence of stochastic iterative dynamic programming algorithms. *Neural computation*, 6(6):1185–1201.

[15] Kaelbling, L. P. (1993). Learning to achieve goals. In *IJCAI*, pages 1094–1099. Citeseer.

[16] Konyushkova, K., Zolna, K., Aytar, Y., Novikov, A., Reed, S., Cabi, S., and de Freitas, N. (2020). Semi-supervised reward learning for offline reinforcement learning. *arXiv preprint arXiv:2012.06899*.

[17] Kostrikov, I., Agrawal, K. K., Dwibedi, D., Levine, S., and Tompson, J. (2018). Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning. In *International Conference on Learning Representations*.

[18] Kostrikov, I., Nachum, O., and Tompson, J. (2019). Imitation learning via off-policy distribution matching. In *International Conference on Learning Representations*.

[19] Laskin, M., Srinivas, A., and Abbeel, P. (2020). Curl: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*, pages 5639–5650. PMLR.

[20] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. In *ICLR (Poster)*.

[21] Lin, X., Baweja, H. S., and Held, D. (2019). Reinforcement learning without ground-truth state. *arXiv preprint arXiv:1905.07866*.

[22] Lu, Q., Van der Merwe, M., Sundaralingam, B., and Hermans, T. (2020). Multifingered grasp planning via inference in deep neural networks: Outperforming sampling by learning differentiable models. *IEEE Robotics Automation Magazine*, 27(2):55–65.

[23] Nachum, O., Gu, S., Lee, H., and Levine, S. (2018). Near-optimal representation learning for hierarchical reinforcement learning. In *International Conference on Learning Representations*.

[24] Nasiriany, S. (2020). *DisCo RL: Distribution-Conditioned Reinforcement Learning for General-Purpose Policies*. PhD thesis.

[25] Pomerleau, D. A. (1989). Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, pages 305–313.

[26] Rajeswaran, A., Kumar, V., Gupta, A., Vezzani, G., Schulman, J., Todorov, E., and Levine, S. (2018). Learning Complex Dexterous Manipulation with Deep Reinforcement Learning and Demonstrations. In *Proceedings of Robotics: Science and Systems (RSS)*.

[27] Ratliff, N. D., Bagnell, J. A., and Zinkevich, M. A. (2006). Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, pages 729–736.

[28] Reddy, S., Dragan, A. D., and Levine, S. (2020). {SQIL}: Imitation learning via reinforcement learning with sparse rewards. In *International Conference on Learning Representations*.

[29] Ross, S., Gordon, G., and Bagnell, D. (2011). A reduction of imitation learning and structured prediction to no-regret online learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 627–635.

[30] Schaul, T., Horgan, D., Gregor, K., and Silver, D. (2015). Universal value function approximators. In *International conference on machine learning*, pages 1312–1320.

[31] Singh, A., Yang, L., Finn, C., and Levine, S. (2019). End-to-end robotic reinforcement learning without reward engineering. In *Robotics: Science and Systems*.

[32] Sutton, R. S. (1988). Learning to predict by the methods of temporal differences. *Machine learning*, 3(1):9–44.

[33] Sutton, R. S. (1995). Td models: Modeling the world at a mixture of time scales. In *Machine Learning Proceedings 1995*, pages 531–539. Elsevier.

[34] Vecerik, M., Sushkov, O., Barker, D., Rothörl, T., Hester, T., and Scholz, J. (2019). A practical approach to insertion with variable socket position using deep reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 754–760. IEEE.

[35] Williams, R. J. and Peng, J. (1991). Function optimization using connectionist reinforcement learning algorithms. *Connection Science*, 3(3):241–268.

[36] Wulfmeier, M., Ondruska, P., and Posner, I. (2015). Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888*.

[37] Xie, A., Singh, A., Levine, S., and Finn, C. (2018). Few-shot goal inference for visuomotor learning and planning. In *Conference on Robot Learning*, pages 40–52. PMLR.

[38] Xu, D. and Denil, M. (2019). Positive-unlabeled reward learning. *arXiv preprint arXiv:1911.00459*.

[39] Yu, T., Quillen, D., He, Z., Julian, R., Hausman, K., Finn, C., and Levine, S. (2020). Meta-world: A benchmark and evaluation for multi-task and meta reinforcement learning. In *Conference on Robot Learning*, pages 1094–1100. PMLR.

[40] Ziebart, B. D., Maas, A. L., Bagnell, J. A., and Dey, A. K. (2008). Maximum entropy inverse reinforcement learning. In *Aaai*, volume 8, pages 1433–1438. Chicago, IL, USA.

[41] Zolna, K., Novikov, A., Konyushkova, K., Gulcehre, C., Wang, Z., Aytar, Y., Denil, M., de Freitas, N., and Reed, S. (2020). Offline learning from demonstrations and unlabeled experience. *arXiv preprint arXiv:2011.13885*.

[42] Zolna, K., Reed, S., Novikov, A., Colmenarej, S. G., Budden, D., Cabi, S., Denil, M., de Freitas, N., and Wang, Z. (2019). Task-relevant adversarial imitation learning. *arXiv preprint arXiv:1910.01077*.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] See Sec. 6.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] We do not foresee plausible negative impacts of this work.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix B.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code is available at:
   `https://github.com/rce-anonymous/rce-anonymous.github.io/tree/main/code`
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Appendix A and provided code.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [No] Each experiment took approximately one day on a standard CPU server. The exact compute resources are proprietary.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [N/A]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

A Derivation of Recursive Classification of Examples

In this section we include the full derivation of our method (RCE). Recall that we want to learn a classifier, $C_\theta^\pi(\mathbf{s_t}, \mathbf{a_t})$ to discriminate between "positive" state-action pairs sampled from the conditional distribution $p^\pi(\mathbf{s_t}, \mathbf{a_t} \mid \mathbf{e_{t+}} = 1)$ and "negatives" sampled from a marginal distribution $p(\mathbf{s_t}, \mathbf{a_t})$. We will use different class-specific weights, using a weight of $p(\mathbf{e_{t+}} = 1)$ for the "positives" and a weight of 1 for the "negatives." Then, if this classifier is trained perfectly, the Bayes-optimal solution is

$$C_\theta^\pi(\mathbf{s_t}, \mathbf{a_t}) = \frac{p^\pi(\mathbf{s_t}, \mathbf{a_t} \mid \mathbf{e_{t+}} = 1)p(\mathbf{e_{t+}} = 1)}{p^\pi(\mathbf{s_t}, \mathbf{a_t} \mid \mathbf{e_{t+}} = 1)p(\mathbf{e_{t+}} = 1) + p(\mathbf{s_t}, \mathbf{a_t})}.$$

The motivation for using these class-specific weights is so that the classifier's predicted probability ratio tells us the probability of solving the task in the future:

$$\frac{C_\theta^\pi(\mathbf{s_t}, \mathbf{a_t})}{1 - C_\theta^\pi(\mathbf{s_t}, \mathbf{a_t})} = p^\pi(\mathbf{e_{t+}} = 1 \mid \mathbf{s_t}, \mathbf{a_t}).$$

Importantly, the resulting method will not actually require estimating the weight $p(\mathbf{e_{t+}} = 1)$. We would like to optimize these parameters using maximum likelihood estimation:

$$\mathcal{L}^\pi(\theta) \triangleq p(\mathbf{e_{t+}} = 1)\, \mathbb{E}_{p(\mathbf{s_t}, \mathbf{a_t} \mid \mathbf{e_{t+}} = 1)}[\log C_\theta^\pi(\mathbf{s_t}, \mathbf{a_t})] + \mathbb{E}_{p(\mathbf{s_t}, \mathbf{a_t})}[\log(1 - C_\theta^\pi(\mathbf{s_t}, \mathbf{a_t}))].$$

While we can estimate the second expectation using Monte Carlo samples $(\mathbf{s_t}, \mathbf{a_t}) \sim p(\mathbf{s_t}, \mathbf{a_t})$, we cannot directly estimate the first expectation because we cannot sample from $(\mathbf{s_t}, \mathbf{a_t}) \sim p(\mathbf{s_t}, \mathbf{a_t} \mid \mathbf{e_{t+}} = 1)$. We will address this challenge by deriving a method for training the classifier that resembles the temporal difference update used in value-based RL algorithms, such as Q-learning. We will derive our method from Eq. 7 using three steps.

The **first step** is is to factor the first expectation in Eq. 7

$$p(\mathbf{s_t}, \mathbf{a_t} \mid \mathbf{e_{t+}} = 1)p(\mathbf{e_{t+}} = 1) = p^\pi(\mathbf{e_{t+}} = 1 \mid \mathbf{s_t}, \mathbf{a_t})p(\mathbf{s_t}, \mathbf{a_t}).$$

Substituting this identity into Eq. 7, we obtain

$$\begin{aligned}\mathcal{L}^\pi(\theta) &= \mathbb{E}_{p(\mathbf{s_t}, \mathbf{a_t})}[p^\pi(\mathbf{e_{t+}} = 1 \mid \mathbf{s_t}, \mathbf{a_t}) \log C_\theta^{(t)}] + \mathbb{E}_{p(\mathbf{s_t}, \mathbf{a_t})}[\log(1 - C_\theta^{(t)})] \\ &= \mathbb{E}_{p(\mathbf{s_t}, \mathbf{a_t})}[p^\pi(\mathbf{e_{t+}} = 1 \mid \mathbf{s_t}, \mathbf{a_t}) \log C_\theta^{(t)} + \log(1 - C_\theta^{(t)})]. \end{aligned} \tag{13}$$

The **second step** is to observe that the term $p^\pi(\mathbf{e_{t+}} = 1 \mid \mathbf{s_t}, \mathbf{a_t})$ satisfies the following recursive identity:

$$p^\pi(\mathbf{e_{t+}} = 1 \mid \mathbf{s_t}, \mathbf{a_t}) = (1 - \gamma)p(\mathbf{e_t} = 1 \mid \mathbf{s_t}) + \gamma \mathbb{E}_{\substack{p(\mathbf{s_{t+1}} \mid \mathbf{s_t}, \mathbf{a_t}), \\ \pi_\phi(\mathbf{a_{t+1}} \mid \mathbf{s_{t+1}})}} \left[p^\pi(\mathbf{e_{t+}} = 1 \mid \mathbf{s_{t+1}}, \mathbf{a_{t+1}})\right]. \tag{14}$$

We substitute this identity into Eq. 13 and then break the expectation into two terms.

$$\begin{aligned}\mathcal{L}^\pi(\theta) &= \mathbb{E}_{p(\mathbf{s_t}, \mathbf{a_t})}\left[\left((1 - \gamma)p(\mathbf{e_t} = 1 \mid \mathbf{s_t}) + \gamma \mathbb{E}_{\substack{p(\mathbf{s_{t+1}} \mid \mathbf{s_t}, \mathbf{a_t}), \\ \pi_\phi(\mathbf{a_{t+1}} \mid \mathbf{s_{t+1}})}} \left[p^\pi(\mathbf{e_{t+}} = 1 \mid \mathbf{s_{t+1}}, \mathbf{a_{t+1}})\right]\right) \log C_\theta^{(t)} + \log(1 - C_\theta^{(t)})\right] \\ &= (1 - \gamma)\mathbb{E}_{p(\mathbf{s_t}, \mathbf{a_t})}\left[p(\mathbf{e_t} = 1 \mid \mathbf{s_t}) \log C_\theta^{(t)}\right] \\ &\qquad + \gamma \mathbb{E}_{p(\mathbf{s_t}, \mathbf{a_t})}\left[\mathbb{E}_{\substack{p(\mathbf{s_{t+1}} \mid \mathbf{s_t}, \mathbf{a_t}), \\ \pi_\phi(\mathbf{a_{t+1}} \mid \mathbf{s_{t+1}})}} \left[p^\pi(\mathbf{e_{t+}} = 1 \mid \mathbf{s_{t+1}}, \mathbf{a_{t+1}})\right] \log C_\theta^{(t)} + \log(1 - C_\theta^{(t)})\right]. \end{aligned} \tag{15}$$

The **third step** is to express the first expectation in Eq. 15 in terms of samples of success example, $\mathbf{s}^* \sim p(\mathbf{s_t} \mid \mathbf{e_t} = 1)$:

$$\begin{aligned}\mathbb{E}_{p(\mathbf{s_t}, \mathbf{a_t})}\left[p(\mathbf{e_t} = 1 \mid \mathbf{s_t}) \log C_\theta^{(t)}\right] &= \mathbb{E}_{p(\mathbf{s_t}, \mathbf{a_t})}\left[\frac{p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)}{p(\mathbf{s_t})} p_U(\mathbf{e_t} = 1) \log C_\theta^{(t)}\right] \\ &= p_U(\mathbf{e_t} = 1)\mathbb{E}_{\substack{p(\mathbf{s_t}), \\ p(\mathbf{a_t} \mid \mathbf{s_t})}}\left[\frac{p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)}{p(\mathbf{s_t})} \log C_\theta^{(t)}\right] \\ &= p_U(\mathbf{e_t} = 1)\mathbb{E}_{\substack{p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1), \\ p(\mathbf{a_t} \mid \mathbf{s_t})}}\left[\log C_\theta^{(t)}\right]. \end{aligned}$$

Note that actions are sampled from $p(\mathbf{a_t} \mid \mathbf{s_t}) = p(\mathbf{s_t}, \mathbf{a_t})/p(\mathbf{s_t})$, the average policy used to collect the dataset of transitions. Finally, we substitute this expression back into Eq. 15, and use Eq. 6 to estimate the probability of future success at time $t+1$.

$$\mathcal{L}^\pi(\theta) = (1-\gamma)p_U(\mathbf{e_t}=1)\mathbb{E}_{\substack{p_U(\mathbf{s_t}|\mathbf{e_t}=1),\\ p(\mathbf{a_t}|\mathbf{s_t})}}\left[\log C_\theta^{(t)}\right] + \gamma\mathbb{E}_{p(\mathbf{s_t},\mathbf{a_t})}\left[\mathbb{E}_{\substack{p(\mathbf{s_{t+1}}|\mathbf{s_t},\mathbf{a_t}),\\ \pi_\phi(\mathbf{a_{t+1}}|\mathbf{s_{t+1}})}}\left[\frac{C_\theta^{(t+1)}}{1-C_\theta^{(t+1)}}\right]\log C_\theta^{(t)} + \log(1-C_\theta^{(t)})\right].$$

Thus, we have arrived at our final objective (Eq. 8).

B Proofs

In this section we include proofs of the theoretical results in the main text.

Proof of Lemma 4.1

Proof. To start, we recall that Eq. 6 says that the Bayes-optimal classifier satisfies

$$\frac{C^\pi(\mathbf{s_t}, \mathbf{a_t})}{1-C^\pi(\mathbf{s_t}, \mathbf{a_t})} = p^\pi(\mathbf{e_{t+}} = 1 \mid \mathbf{s_t}, \mathbf{a_t}).$$

Substituting this identity into both the LHS and RHS of the recursive definition of $p^\pi(\mathbf{e_{t+}} = 1 \mid \mathbf{s_t}, \mathbf{a_t})$ (Eq. 14), we obtain the desired result:

$$\frac{C^\pi(\mathbf{s_t}, \mathbf{a_t})}{1-C^\pi(\mathbf{s_t}, \mathbf{a_t})} = (1-\gamma)p^\pi(\mathbf{e_t}=1 \mid \mathbf{s_t}) + \gamma\mathbb{E}_{\substack{p(\mathbf{s_{t+1}}|\mathbf{s_t},\mathbf{a_t})\\ \pi(\mathbf{a_{t+1}}|\mathbf{s_{t+1}})}}\left[\frac{C^\pi(\mathbf{s_{t+1}}, \mathbf{a_{t+1}})}{1-C^\pi(\mathbf{s_{t+1}}, \mathbf{a_{t+1}})}\right].$$

□

Proof of Lemma 4.2.1

Proof. We will show that RCE is equivalent to using value iteration using the Bellman equation above, where $(1-\gamma)p(\mathbf{e_{t+1}} = 1 \mid \mathbf{s_t}, \mathbf{a_t})$ takes the role of the reward function. For given transition $(\mathbf{s_t}, \mathbf{a_t}, \mathbf{s_{t+1}})$, the corresponding TD target y is the expected value of three terms (Eq. 8): with weight $(1-\gamma)p(\mathbf{e_t} = 1 \mid \mathbf{s_t})$ it is assigned $y = 1$; with weight $\gamma\mathbb{E}[w]$ it is assigned label $y = 1$; and with weight 1 it is assigned label $y = 0$. Thus, the expected value of the TD target y can be written as follows:

$$\mathbb{E}[y \mid \mathbf{s_t}, \mathbf{a_t}, \mathbf{s_{t+1}}] = \frac{(1-\gamma)p(\mathbf{e_t}=1 \mid \mathbf{s_t}) \cdot 1 + \gamma\mathbb{E}[w] \cdot 1 + 1 \cdot 0}{(1-\gamma)p(\mathbf{e_t}=1 \mid \mathbf{s_t}) + \gamma\mathbb{E}[w] + 1}.$$

Thus, the assignment equation can be written as follows:

$$C^\pi(\mathbf{s_t}, \mathbf{a_t}) \leftarrow \mathbb{E}[y \mid \mathbf{s_t}, \mathbf{a_t}, \mathbf{s_{t+1}}] = \frac{(1-\gamma)p(\mathbf{e_t}=1 \mid \mathbf{s_t}) + \gamma\mathbb{E}[w]}{(1-\gamma)p(\mathbf{e_t}=1 \mid \mathbf{s_t}) + \gamma\mathbb{E}[w] + 1}. \tag{16}$$

While this gives us an assignment equation for C, our Bellman equation is expressed in terms of the *ratio* $C/(1-C)$. Noting that the function $C/(1-C)$ is strictly monotone increasing, we can write the same assignment equation for the ratio as follows:

$$\frac{C^\pi(\mathbf{s_t}, \mathbf{a_t})}{1-C^\pi(\mathbf{s_t}, \mathbf{a_t})} \leftarrow \frac{\mathbb{E}[y \mid \mathbf{s_t}, \mathbf{a_t}, \mathbf{s_{t+1}}]}{1-\mathbb{E}[y \mid \mathbf{s_t}, \mathbf{a_t}, \mathbf{s_{t+1}}]} = (1-\gamma)p(\mathbf{e_t}=1 \mid \mathbf{s_t}) + \gamma\mathbb{E}[w].$$

Recalling that $w = \frac{C(\mathbf{s_{t+1}}, \mathbf{a_{t+1}})}{1-C(\mathbf{s_{t+1}}, \mathbf{a_{t+1}})}$, we conclude that our temporal difference method is equivalent to doing value iteration using the Bellman equation in Eq. 11. □

Proof of Corollary 4.2.1

Proof. Lemma 4.2 showed that the update rule for our method is equivalent to value iteration. Value iteration is known to converge in the tabular setting [14, Theorem 1], so our method also converges. □

Proof of Lemma 4.3 The proof of Lemma 4.3 is nearly identical to the standard policy improvement proof for Q-learning.

Proof.

$$
\begin{aligned}
p^\pi(\mathbf{e_t}_+ = 1) &= \mathbb{E}_{p_0(s),\pi(a_0|s_0)}[p^\pi(\mathbf{e_t}_+ = 1 \mid s_0, a_0)] \\
&\leq \mathbb{E}_{p_0(s),\pi'(a_0|s_0)}[p^\pi(\mathbf{e_t}_+ = 1 \mid s_0, a_0)] \\
&= \mathbb{E}_{p_0(s),\pi'(a_0|s_0)}[(1-\gamma)p(e_0 = 1 \mid s_0, a_0) + \gamma \mathbb{E}_{p(s_1|s_0,a_0),\pi(a_1|s_1)}[p^\pi(\mathbf{e_t}_+ = 1 \mid s_1, a_1)]] \\
&\leq \mathbb{E}_{p_0(s),\pi'(a_0|s_0)}[(1-\gamma)p(e_0 = 1 \mid s_0, a_0) + \gamma \mathbb{E}_{p(s_1|s_0,a_0),\pi'(a_1|s_1)}[p^\pi(\mathbf{e_t}_+ = 1 \mid s_1, a_1)]] \\
&= \mathbb{E}_{\substack{p_0(s),\pi'(a_0|s_0) \\ p(s_1|s_0,a_0),\pi'(a_1|s_1)}} [(1-\gamma)p(e_0 = 1 \mid s_0, a_0) + \gamma p(e_1 = 1 \mid s_1, a_1) \\
&\qquad + \gamma^2 \mathbb{E}_{p(s_2|s_1,a_1),\pi(a_2|s_2)}[p^\pi(\mathbf{e_t}_+ = 1 \mid s_2, a_2)]] \\
&\ldots \\
&\leq p^{\pi'}(\mathbf{e_t}_+ = 1)
\end{aligned}
$$

□

C Robust Example-Based Control

Our strategy for analyzing robust example-based control will be to first compute an analytic solution to the inner minimization problem. Plugging in the optimal adversary, we will find that we weight the success examples inversely based on how often our policy utilizes each of the potential solution strategies.

C.1 Proof of Lemma 4.4

Proof. First, we solve the inner minimization problem. Note that the adversary's choice of an example probability function $\hat{p}(\mathbf{e_t} \mid \mathbf{s_t})$ is equivalent to a choice of a marginal distribution, $p_U(\mathbf{s_t})$. We can this write the inner minimization problem as follows:

$$
\min_{p_U(\mathbf{s_t})} \int p^\pi(\mathbf{s_t}_+ = \mathbf{s_t}) \frac{p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)\cancel{p(\mathbf{e_t} = 1)}}{p_U(\mathbf{s_t})} d\mathbf{s_t}
$$

As before, we can ignore the constant $p(\mathbf{e_t} = 1)$. We solve this optimization problem, subject to the constraint that $\int p_U(\mathbf{s_t}) d\mathbf{s_t} = t$, using calculus of variations. The corresponding Lagrangian is

$$
\mathcal{L}(p_U, \lambda) = \int \frac{p^\pi(\mathbf{s_t}_+ = \mathbf{s_t}) p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)}{p_U(\mathbf{s_t})} d\mathbf{s_t} + \lambda \left(\int p_U(\mathbf{s_t}) d\mathbf{s_t} - 1 \right).
$$

Setting the derivative $\frac{d\mathcal{L}}{dp_U(\mathbf{s_t})} = 0$, we get

$$
-\frac{p^\pi(\mathbf{s_t}_+ = \mathbf{s_t}) p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)}{p_U^2(\mathbf{s_t})} + \lambda = 1 \implies p_U(\mathbf{s_t}) = \frac{\sqrt{p^\pi(\mathbf{s_t}_+ = \mathbf{s_t}) p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)}}{\int \sqrt{p^\pi(\mathbf{s_t}_+ = \mathbf{s_t}') p_U(\mathbf{s_t}' \mid \mathbf{e_t} = 1)} d\mathbf{s_t}'}.
$$

Note that $\frac{d^2\mathcal{L}}{dp_U(\mathbf{s_t})^2} > 0$, so this stationary point is a minimum. We can therefore write the worst-case probability function as

$$
\begin{aligned}
\hat{p}(\mathbf{e_t} = 1 \mid \mathbf{s_t}) &= \frac{p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)}{p_U(\mathbf{s_t})} p(\mathbf{e_t} = 1) \\
&= \sqrt{\frac{p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)}{p^\pi(\mathbf{s_t}_+ = \mathbf{s_t})}} \int \sqrt{p^\pi(\mathbf{s_t}_+ = \mathbf{s_t}') p_U(\mathbf{s_t}' \mid \mathbf{e_t} = 1)} d\mathbf{s_t}' p(\mathbf{e_t} = 1).
\end{aligned}
$$

Intuitively, this says that the worst-case probability function is one where successful states $\mathbf{s_t} \sim p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)$ are downweighted if the current policy visits those states more frequently (i.e., if

$p^\pi(\mathbf{s_{t+}} = \mathbf{s_t})$ is large). Substituting this worst-case example probability into Eq. 12, we can write the adversarial objective as follows:

$$\max_\pi \int p^\pi(\mathbf{s_{t+}} = \mathbf{s_t}) \sqrt{\frac{p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)}{p^\pi(\mathbf{s_{t+}} = \mathbf{s_t})}} d\mathbf{s_t} \int \sqrt{p^\pi(\mathbf{s_{t+}} = \mathbf{s_t}')p_U(\mathbf{s_t}' \mid \mathbf{e_t} = 1)} d\mathbf{s_t}' p(\mathbf{e_t} = 1)$$
$$= \left(\int \sqrt{p^\pi(\mathbf{s_{t+}} = \mathbf{s_t})p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)} d\mathbf{s_t} \right)^2 p(\mathbf{e_t} = 1)$$

Since $p(\mathbf{e_t} = 1)$ is assumed to be a constant and $(\cdot)^2$ is a monotone increasing function (for non-negative arguments), we can express this same optimization problem as follows:

$$\max_\pi \int \sqrt{p^\pi(\mathbf{s_{t+}} = \mathbf{s_t})p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)} d\mathbf{s_t}.$$

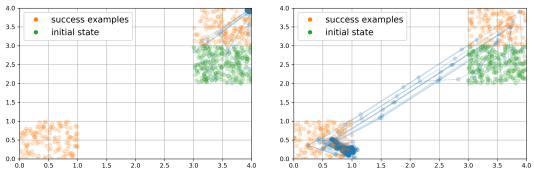
Using a bit of algebra, we can show that this objective is a (scaled and shifted) squared Hellinger distance:

$$\begin{aligned}
&\int \sqrt{p^\pi(\mathbf{s_{t+}} = \mathbf{s_t})p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)} d\mathbf{s_t} \\
&= 1 - \frac{1}{2} \int \left(\sqrt{p^\pi(\mathbf{s_{t+}} = \mathbf{s_t})} - \sqrt{p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)} \right)^2 d\mathbf{s_t} \\
&= 1 - \frac{1}{2} \int \left(\sqrt{p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)} \left(\sqrt{\frac{p^\pi(\mathbf{s_{t+}} = \mathbf{s_t})}{p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)}} - 1 \right) \right)^2 d\mathbf{s_t} \\
&= 1 - \frac{1}{2} \mathbb{E}_{p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)} \left[\left(1 - \sqrt{\frac{p^\pi(\mathbf{s_{t+}} = \mathbf{s_t})}{p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1)}} \right)^2 \right] \\
&= 1 - \frac{1}{2} H^2(p_U(\mathbf{s_t} \mid \mathbf{e_t} = 1), p^\pi(\mathbf{s_{t+}} = \mathbf{s_t})).
\end{aligned}$$

□

C.2 Robust Example-Based Control and Iterated RCE

As noted in Sec. 4.2, we found empirically that the policy found by RCE with online data collection was similar to the optimal policy for the robust example-based control objective. We visualize that experiment in Fig. 5. In this 2D navigation task there are two sets of success examples (orange circles). When we apply RCE to this task using a uniform distribution of transitions, the agent navigates directly towards the closer success examples. However, when we run RCE in an iterative fashion, periodically collecting data from the current policy, the final policy visits both sets of success examples. This behavior is similar to what we would expect from the optimal policy to robust example-based control We hypothesize that this happens because our implementation of RCE periodically collects new data using the current policy, violating the assumption that the dataset of transitions $p(\mathbf{s_t}, \mathbf{a_t}, \mathbf{s_{t+1}})$ is fixed. When the policy frequently visits one success example, that state will be included many times in the dataset of transitions $p(\mathbf{s_t}, \mathbf{a_t}, \mathbf{s_{t+1}})$ at the next iteration, so the classifier's predictions will decrease for that state.



(a) Offline RCE (b) Iterated RCE

Figure 5: A 2D navigation task with two sets of success states. *(Left)* When we apply RCE to this task without online data collection, the agent navigates directly toward the closer set of success states. *(Right)* When we run RCE in an iterated fashion, periodically collecting new transitions using the current policy, the learned policy visits both sets of success states. The behavior of iterated RCE is similar to what we would expect from robust example-based control.

We now show that (under a very strong assumption), the solution to the robust example-based control problem is a fixed point of an iterative version of RCE. The iterated version of RCE is defined by

alternating between collecting a dataset of transitions using the current policy and then running RCE (Alg. 1) on those trajectories:

$$\pi \leftarrow \text{RCE}(\mathcal{S}^*, \mathcal{D}), \qquad \mathcal{D} \leftarrow \{\tau \sim \pi\}. \tag{17}$$

Lemma C.1. *Let π be an optimal policy for robust example-based control (Eq. 12). Assume that π visits each success example with probability proportional to how often that state occurs as a success example:*

$$\exists\, 0 < c \leq 1 \;\; s.t. \;\; p^\pi(\mathbf{s_{t+}} = s) = c \cdot p(\mathbf{s_t} = s \mid \mathbf{e_t} = 1)\; \forall s \in \mathcal{S}. \tag{18}$$

Then π is a fixed point for iterated RCE.

Proof. If $p^\pi(\mathbf{s_{t+}}) = p(\mathbf{s_t} \mid \mathbf{e_t} = 1)$, then Eq. 3 tells us that $p(\mathbf{e_t} = 1 \mid \mathbf{s_t})$ is a constant for all states. In this setting, all policies are optimal under our objective (Def. 1), so the current policy π is also optimal. Thus, any policy satisfying $p^\pi(\mathbf{s_{t+}}) = c \cdot p(\mathbf{s_t} \mid \mathbf{e_t} = 1)$ is a fixed point of iterated RCE. □

When we are given a discrete set of success examples, the assumption in Eq. 18 is equivalent to assuming that the optimal robust policy spends an equal fraction of its time at each success example.

Lemma C.2. *Assume there exists a policy π such that $p^\pi(\mathbf{s_{t+}}) = p(\mathbf{s_t} \mid \mathbf{e_t} = 1)$. Then the solution to robust example-based control is a fixed point of iterated RCE.*

Proof. As shown in Lemma 4.4, the robust example-based control corresponds to minimizing the squared Hellinger distance, an f-divergence. All f-divergences are minimized when their arguments are equal (if feasible), so the solution to robust example-based control is a policy π satisfying $p^\pi(\mathbf{s_{t+}}) = p(\mathbf{s_t} \mid \mathbf{e_t} = 1)$.

Now, we need to show that such a policy is a fixed point of iterated RCE. If $p^\pi(\mathbf{s_{t+}}) = p(\mathbf{s_t} \mid \mathbf{e_t} = 1)$, then Eq. 4 tells us that $p(\mathbf{e_t} = 1 \mid \mathbf{s_t})$ is a constant for all states. In this setting, all policies are optimal under our objective (Def. 1), so the current policy π is also optimal. Thus, any policy satisfying $p^\pi(\mathbf{s_{t+}}) = p(\mathbf{s_t} \mid \mathbf{e_t} = 1)$ is a fixed point of iterated RCE. □

While this result offers some explanation for why iterated RCE might be robust, the result is quite weak. First, the assumption that $p^\pi(\mathbf{s_{t+}}) = p(\mathbf{s_t} \mid \mathbf{e_t} = 1)$ for some policy π is violated in most practical environments, as it would imply that the agent spends every timestep in states that are success examples. Second, this result only examines the fixed points of robust example-based control and iterated RCE, and does not guarantee that these methods will actually converge to this fixed point. We aim to lift these limitations in future work.

D Connection with Success Classifiers and Success Densities

In this section we show that objective function in prior methods for example-based control implicitly depend on where success examples came from.

Nasiriany [24] learn a reward function by fitting a density model to success examples, $\hat{p}(s^*) \approx p(\mathbf{s_t} \mid \mathbf{e_t} = 1)$. Using this learned density as a reward function is *not* the same as our approach. Whereas our method implicitly corresponds to the reward function $r(\mathbf{s_t}) = (1-\gamma)p(\mathbf{e_t} = 1 \mid \mathbf{s_t})$, the reward function in Nasiriany [24] corresponds to $\exp r(\mathbf{s_t}) = p(\mathbf{s_t} \mid \mathbf{e_t} = 1) = p(\mathbf{e_t} = 1 \mid \mathbf{s_t})p_U(\mathbf{s_t})/p(\mathbf{e_t} = 1)$. The additional $p_U(\mathbf{s_t})$ term in the (exponentiated) Disco RL reward function biases the policy towards visiting states with high density under the $p_U(\mathbf{s_t})$ marginal distribution, *regardless of whether those states were actually labeled as success examples*.

In contrast, VICE [8] learns a classifier to distinguish success examples $s^* \sim p(\mathbf{s_t} \mid \mathbf{e_t} = 1)$ from "other" states sampled from $q(\mathbf{s_t})$. The predicted probability ratio from the Bayes-optimal classifier is

$$\frac{p(\mathbf{s_t} \mid \mathbf{e_t} = 1)}{q_E(\mathbf{s_t})} = \frac{p(\mathbf{e_t} = 1 \mid \mathbf{s_t})p_U(\mathbf{s_t})}{p(\mathbf{e_t} = 1)q(\mathbf{s_t})}.$$

This term depends on the user's state distribution $p_U(\mathbf{s_t})$, so without further assumptions, using this probability ratio as a reward function does not yield a policy that maximizes the future probability of

success (Eq. 2). We can recover example-based control if we make the additional assumption that $p_U(\mathbf{s_t}) = q(\mathbf{s_t})$, an assumption not made explicit in prior work. Even with this additional assumption, VICE differs from RCE by requiring an additional classifier.

In summary, these prior approaches to example-based control rely on auxiliary function approximators and do not solve the example-based control problem (Def. 1) without additional assumptions. In contrast, our approach is simpler and is guaranteed to yield the optimal policy.

E Experimental Details

E.1 Implementation

This method is straightforward to implement on top of any actor-critic method, such as SAC or TD3, only requiring a few lines of code to be added. The only changes necessary are to (1) sample success in the train step and (2) swap the standard Bellman loss with that in Eq. 10. Unless otherwise noted, all experiments were run with 5 random seeds. When plots in the paper combine the results across seeds, this is done by taking the average.

We based our implementation of RCE off the SAC implementation from TF-Agents [10]. We modified the critic loss to remove the entropy term and use our loss (Eq. 10) in place of the Bellman loss, modified the TD targets to use n-step returns with $n = 10$, and modified the actor loss to use a fixed entropy coefficient of $\alpha = 1e-4$. To implement n-step returns, we replaced the target value of $y = \frac{\gamma w^{(t)}}{\gamma w^{(t)}+1}$ with $y = \frac{1}{2}\left(\frac{\gamma w^{(t)}}{\gamma w^{(t)}+1} + \frac{\gamma^{10} w^{(t+10)}}{\gamma^{10} w^{(t+10)}+1}\right)$. We emphasize that implementing RCE therefore only required modifying a few lines of code. Unless otherwise noted, all hyperparameters were taken from that implementation (version v0.6.0). For the image-based experiments, we used the same architecture from CURL [19], sharing the image encoder between the actor and classifier. Initial experiments found that using random weights for the image encoder worked at least as well as actually training the weights of the image encoder, while also making training substantially faster. We therefore used a random (untrained) image encoder in all experiments and baselines on the image-based tasks. We used the same hyperparameters for all environments, with the following exception:

- `sawyer_bin_picking`: The SAC implementation in TF-Agents learns two independent Q values and takes the minimum of them when computing the TD target and when updating the policy, as suggested by Fujimoto et al. [9]. However, for this environment we found that the classifier's predictions were too small, and therefore modified the actor and critic losses to instead take the maximum over the predictions from the two classifiers. We found this substantially improved performance on this task.

E.2 Baselines.

We implemented the SQIL baseline as an ablation to our method. The two differences are (1) using the standard Bellman loss instead of our loss (Eq. 10) and (2) not using n-step returns. We implemented all other baselines on top of the official DAC repository [17]. For fair comparison with our method, the classifier was only conditioned on the state, not the action. These baselines differed based on the following hyperparameters:

Method name	gradient penalty	discriminator loss	absorbing state wrapper
DAC	10	`cross_entropy`	yes
AIRL	0	`cross_entropy`	no
ORIL-PU	0	`positive_unlabeled`	no
PURL	0	`positive_unlabeled_margin`	no

Table 1: Classifier-based baselines

The done bit wrapper is the idea discussed in DAC [17, Sec. 4.2]. The `cross_entropy` loss is the standard cross entropy loss. The `positive_unlabeled` loss is the standard positive-unlabeled loss [3]; following Zolna et al. [41, Appendix B, Eq. 5] we used $\eta = 0.5$. The

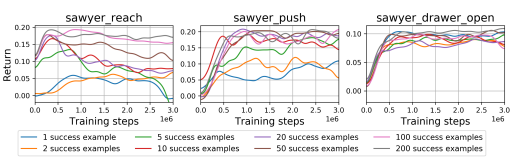
positive_unlabeled_margin loss is taken from Xu and Denil [38, Eq. 9], as are the hyper-parameters of $\eta = 0.5, \beta = 0.0$.

E.3 Environments

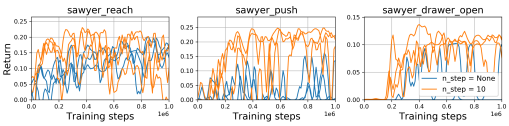
For both the state-based experiments and the image-based experiments, we choose tasks using two design criteria: (1) we wanted to include a breadth of tasks, and (2) we wanted to include the most difficult tasks that could be learned by either our method or any of the baselines. For the image-based tasks, we were unable to get the prior methods to learn tasks more complex than those included in Fig. 4.

Our experiments used benchmark manipulation tasks from Metaworld (Jan. 31, 2020) [39] and D4RL (Aug. 26, 2020) [6, 26]. Unless otherwise mentioned, we used the default parameters for the environments. The reward functions for some of the Sawyer tasks contain additional reward shaping terms. As we describe below, we remote these shaping terms to more directly evaluate success.

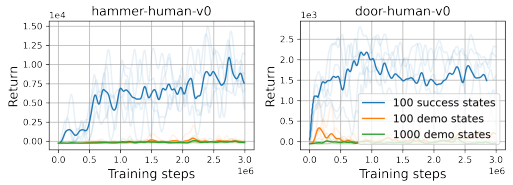
1. sawyer_drawer_open: This task is based on the SawyerDrawerOpenEnv from Yu et al. [39]. We generated success examples by sampling the drawer Y coordinate from unif(low=-0.25, high=-0.15) and moving the gripper to be touching the drawer handle. Episodes were 151 steps long. For evaluation, we used the net distance that the robot opened the drawer (displacement along the negative Y axis) as the reward.
2. sawyer_push: This task is based on the SawyerReachPushPickPlaceEnv from Yu et al. [39], using task_type = 'push'. We generated success examples by sampling the puck XY position from unif(low=(0.05, 0.8), high=(0.15, 0.9)) and moving the gripper to be touching the puck. Episodes were 151 steps long. For evaluation, we used the net distance that the puck traveled towards the goal as the reward.
3. sawyer_lift: This task is based on the SawyerReachPushPickPlaceEnv from Yu et al. [39], using task_type = 'reach'. We generated success examples by sampling the puck Z coordinate from unif(low=0.08, high=0.12) and moving the gripper such that the puck was inside the gripper. Episodes were 151 steps long. For evaluation, we used the Z coordinate of the puck at the final time step.
4. sawyer_box_close: This task is based on the SawyerBoxCloseEnv from Yu et al. [39]. We generated success examples by positioning the lid on top of the box and positioning the gripper so that the lid handle was inside the gripper. Episodes were 151 steps long. For evaluation, we used the net distance that the lid traveled along the XY plane towards the center of the box.
5. sawyer_bin_picking: This task is based on the SawyerBinPickingEnv from Yu et al. [39]. We generated success examples by randomly positioning the object in the target bin, sampling the XY coordinate from unif(low=(0.06, 0.64), high=(0.18, 0.76)) and positioning the gripper so that the object was inside the gripper. Episodes were 151 steps long. For evaluation, we used the net distance that the object traveled along the XY plane towards the center of the target bin.
6. door-human-v0, hammer-human-v0: These tasks are described in [26]. We generated success examples by taking a random subset of the last 50 observations from each trajectory of demonstrations provided for these tasks in [6]. We used the default episode length of 200 for these tasks. For evaluation, we used the default reward functions for these tasks (which are a few orders of magnitude larger than the net-distance rewards used for evaluating the sawyer tasks.).
7. sawyer_reach_random_position_image, sawyer_reach_random_size_image: These image-based reaching tasks are based on the SawyerReachPushPickPlaceEnv task from Yu et al. [39], using task_type = 'reach'. We generated success examples by positioning the gripper at the object. Episodes were 51 steps long. For evaluation, we used the net distance traveled towards the object as the reward. The sawyer_reach_random_position_image modified the environment to initialize each episode by randomly sampling the object position from unif(low=(-0.02, 0.58), high=(0.02, 0.62)). The sawyer_reach_random_size_image modified the environment to initialize each episode by randomly resizing the object by sampling a radius and half-height from unif(low=(0.1, 0.005), high=(0.05, 0.045)).



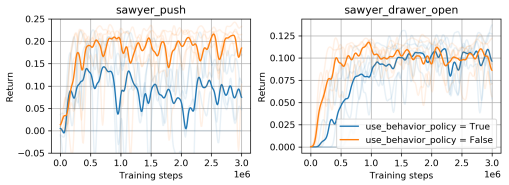
(a) Number of success examples



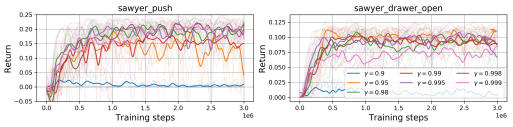
(b) N-step returns



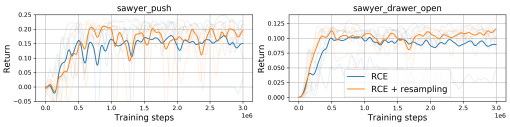
(c) Using all states from demonstrations.



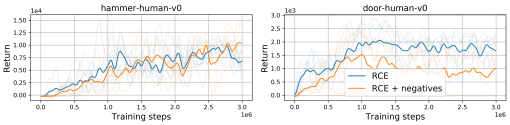
(d) Actions for success examples



(e) Ablation of discount factor.



(f) Sampling new success examples.



(g) Using negative examples.

Figure 6: **Ablation Experiments**

8. `sawyer_clear_image`: This task is based on the `SawyerReachPushPickPlaceEnv` from Yu et al. [39], using `task_type = 'push'`. We generated success examples by simply deleting the object from the scene and setting the gripper to a random position drawn from `unif(low=(-0.2, 0.4, 0.02), high=(0.2, 0.8, 0.3))`. For evaluation, we used a reward of 1 if the object was out of sight, as defined by having a Y coordinate less than 0.41 or greater than 0.98.

F Ablation Experiments and Visualization

We ran a number of ablation experiments to further understand our method.

Number of success examples. We studied performance when we used fewer than 200 success examples. As shown in Fig. 6a, we can achieve comparable performance using 100 examples for the `sawyer_reach` task, 20 examples for the `sawyer_push` task, and just 1 example for the `sawyer_drawer_open` task. We ran each experiment with 5 random seeds, but only plot the mean across seeds to decrease clutter.

N-step returns. An important implementation detail of RCE is using n-step returns. Fig. 6b shows that using n-step returns substantially improved performance on the `sawyer_push` and `sawyer_drawer_open` tasks, and had a smaller positive effect on the `sawyer_reach` task. We used n-step returns with $n = 10$ for all other experiments with our method.

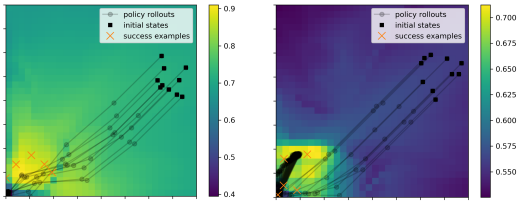
Applying RCE to expert trajectories. RCE is intended to be applied to a set of success examples. Applying this same method to states sampled randomly from expert demonstrations does not work, as shown in Fig. 6c. Not only is requiring intermediate states from demonstrations more onerous for users, but also RCE performs worse in this setting. Interestingly, we found the same trend for the SQIL baseline, which (unlike our method) was designed to use states from demonstrations, not just success states. Thus, while SQIL is designed as an imitation learning method, it may actually be solving a version of example based control.

Sampling actions for success examples. We studied the approximation made in Sec. 3.2, where we sampled actions for the success examples using our current policy instead of the historical average policy. We fit a new behavior policy to our replay buffer using behavior cloning and used this behavior policy to sample actions for the success examples. Fig. 6d shows that simply using our current policy instead of this behavior policy worked at least as well, while also being simpler to implement.

New success examples during training. RCE is intended to be applied in the setting where a set of success examples is provided before training. Nonetheless, we can apply RCE to a set of success examples that is updated throughout training. We implemented this variant of RCE, sampling the success set anew every 100k iterations. Fig. 6f shows that sampling new success examples online during training does not affect the performance of RCE.

Negative examples. In this ablation experiment, we modified RCE by training the classifier to predict 0 for a set of "negative" states. We implemented this variant of RCE, using samples from the initial state (the task is never solved at the initial state) as negatives. Fig. 6g shows that this variant performs about the same on the hammer task, but worse on the door task.

Discount factor, γ. As shown in Fig. 6e, the performance of RCE is indistinguishable for a wide range of discount factors (0.95 to 0.998). RCE does perform worse for very small (0.9) and very large (0.999) discount factors. Note that we did not tune this parameter for the experiments in the main paper, and used a value of $\gamma = 0.99$ for all experiments there.



(a) 20k iterations (b) 80k iterations

Figure 7: Visualizing the classifier's predictions and policy rollouts throughout training. To avoid visual clutter, we plot a random subset of the success examples.

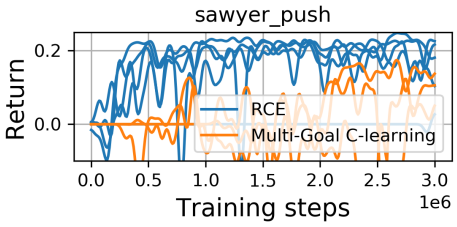


Figure 8: We compare RCE against an alternative approach based on goal-conditioned RL.

F.1 Visualizing Learning Dynamics

To better illustrate the mechanics of our method, we ran RCE on a simple 2D navigation task. We then visualized the classifier's predictions throughout training. As shown in Fig. 7, the classifier's predictions are initially mostly uniform across all states, but later converse to predict much larger values near the success examples and lower values elsewhere.

F.2 Comparison with C-Learning

C-learning [4] solves a different problem than RCE and thus cannot be directly applied to this setting. Specifically, C-learning learns a goal-conditioned policy from a dataset of transitions, whereas RCE learns a single policy (not conditioned on goals) from a dataset of transitions and another dataset of success examples. Nonetheless, we initially tried to solve example-based control problems using a version of C-learning modified to use success examples. However, we found that this method performed quite poorly. For example, on the sawyer-push task, RCE achieves an average return of 0.2 (see Fig. 8) whereas this variant of C-learning achieved a return of about 0.07 (65% worse). The poor performance of this variant of C-learning motivated us to design a new method from the ground-up, specifically designed to solve the example-based control problem. We will include a plot of these initial experiments in the final paper.

G Failed Experiments

This section describes a number of experiments that we tried that did not work particularly well. We did not include any of these ideas in our final method.

1. **Training the image encoder.** For the image-based experiments, we found that *not* training the image encoder and just using random weights often worked better than actually learning these weights.
2. **Alternative parametrization of the classifier.** By default, our classifier predicts values in the range $C_\theta \in [0, 1]$. However, Eq. 6 says that the classifier's predicted probability ratio corresponds to
$$p^\pi(\mathbf{e_{t+}} = 1 \mid \mathbf{s_t}, \mathbf{a_t}) = \frac{C_\theta^{(t)}}{1 - C_\theta^{(t)}} \in [0, 1].$$
We therefore expect that a learned classifier should only make predictions in the range $[0, 0.5]$. We experimented by parametrizing the classifier to only output values in $[0, 0.5]$, but found that this led to no noticeable change in performance.
3. **Classifier regularization.** We experimented with regularizing our classifier with input noise, gradient penalties, mixup, weight decay and label smoothing. These techniques yield improvements that were generally small, so we opted to not include any regularization in our final method to keep it simple.
4. **Regularizing the *implicit* reward.** We can invert our Bellman equation (Eq. 11) to extract the reward function that has been implicitly learned by our method:
$$r(\mathbf{s_t}, \mathbf{a_t}) = \frac{C_\theta^{(t)}}{1 - C_\theta^{(t)}} - \gamma \frac{C_\theta^{(t+1)}}{1 - C_\theta^{(t+1)}}.$$
We experimented with regularizing this implicit reward function to be close to zero or within $[0, 1]$, but saw no benefit.
5. **Normalizing w.** We tried normalizing the TD targets w (Eq. 9) to have range $[0, 1]$ by applying a linear transformation (subtracting the min and dividing by the max-min), but found that this substantially hurt performance.
6. **Hyperparameter robustness.** We found that RCE was relatively robust to the learning rate, the discount factor, the Polyak averaging term for target networks, the batch size, the classifier weight initialization, and the activation function.
7. **Imbalanced batch sizes.** Recall that our objective (Eq. 10) uses a very small weight of $(1 - \gamma)$ for the first term. This small weight means that our effective batch size is relatively small. We experimented with changing the composition of each batch to include fewer success examples and more random examples, updating the coefficients for the loss terms such that the overall loss remained the same. While this does increase the effective batch size, we found it had little effect on performance (perhaps because our method is fairly robust to batch size, as noted above).
8. **Optimistic initialization.** We attempted to encourage better exploration by initializing the final layer bias of the classifier so that, at initialization, the classifier would predict $C_\theta = 1$. We saw no improvements from this idea.
9. **More challenging image-based tasks.** We found that RCE (and all the baselines) failed on image-based versions of the tasks in Fig. 3. Our hypothesis is that this setting is challenging for RCE for the same reason that image-based tasks are challenging for off-policy RL algorithms: it is challenging to learn Q functions via temporal different learning on high-dimensional inputs. In future work, we aim to study the use of explicit representation learning to assist RCE in learning complex image-based tasks.