End-to-End Robotic Reinforcement Learning without Reward Engineering

Avi Singh, Larry Yang, Kristian Hartikainen, Chelsea Finn, Sergey Levine

University of California, Berkeley

Email: {avisingh, larrywyang, hartikainen, cbfinn, svlevine}@berkeley.edu

Abstract—The combination of deep neural network models and reinforcement learning algorithms can make it possible to learn policies for robotic behaviors that directly read in raw sensory inputs, such as camera images, effectively subsuming both estimation and control into one model. However, real-world applications of reinforcement learning must specify the goal of the task by means of a manually programmed reward function, which in practice requires either designing the very same perception pipeline that end-to-end reinforcement learning promises to avoid, or else instrumenting the environment with additional sensors to determine if the task has been performed successfully. In this paper, we propose an approach for removing the need for manual engineering of reward specifications by enabling a robot to learn from a modest number of examples of successful outcomes, followed by actively solicited queries, where the robot shows the user a state and asks for a label to determine whether that state represents successful completion of the task. While requesting labels for every single state would amount to asking the user to manually provide the reward signal, our method requires labels for only a tiny fraction of the states seen during training, making it an efficient and practical approach for learning skills without manually engineered rewards. We evaluate our method on real-world robotic manipulation tasks where the observations consist of images viewed by the robot's camera. In our experiments, our method effectively learns to arrange objects, place books, and drape cloth, directly from images and without any manually specified reward functions, and with only 1-4 hours of interaction with the real world. Videos of learned behavior are available at sites.google.com/view/reward-learning-rl/.
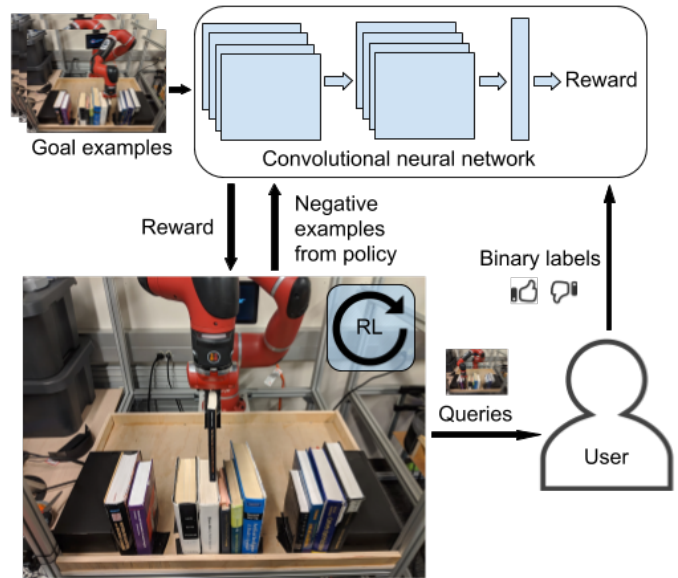
Fig. 1. **Illustration of our approach.** We learn a reward function on high-dimensional observations (i.e. pixels) using a neural network classifier. The classifier receives a set of goal examples (images) from a user that specify the desired outcome. Negative examples for classifier training are obtained from the policy. Our method trains a policy to place a book on the bookshelf via reinforcement learning w.r.t. the learned reward. During this process, the robot periodically queries the user with images, and the user provides a binary label indicating whether or not this image corresponds to a successful outcome. This allows the robot to learn tasks with a modest number of examples and queries, and without manually designing reward functions.

I. INTRODUCTION

Reinforcement learning holds the promise of enabling robotic systems to improve continuously through experience. A robot equipped with a sufficiently powerful reinforcement learning algorithm can become more and more proficient at a given task as it practices it directly in the real world. With end-to-end algorithms that enable skill learning directly from raw sensory observations, such as images, this capability can be tremendously powerful: every aspect of the robot's estimation and control pipeline can improve from experience and become better and better suited to the task at hand. However, in order to enable such continual improvement, the robot must have a way to evaluate whether it is succeeding at the task – it needs a reward function. Unfortunately, in practice, the design of reward functions for robotic skills is very challenging, especially when learning skills from raw observations such as images: manually defining reward functions typically requires manually-designed perception systems or instrumentation of the environment (e.g., by placing additional sensors) [34, 44, 21, 35]. This either precludes learning directly in open-world environments that are not instrumented, or makes learning entirely contingent on the reward perception system, which does not improve or adapt over the robot's lifetime. In many scenarios, the need for manual engineering of reward functions defeats the point of end-to-end learning from pixels, if the reward function itself requires a prior perception pipeline or instrumentation.

Inverse reinforcement learning methods [47, 41, 11, 17, 12] seek to automate reward definition by learning a reward function from expert demonstrations. However, these methods have limitations of their own: collecting expert demonstrations puts a significant data collection burden on the user, and demonstrations are usually non-intuitive for a person to provide (often involving kinesthetically moving a robot [11] or teleoperating it [39]). More importantly, demonstrating how

a task is done defeats a central goal of reinforcement learning: autonomously discovering how to perform skills through trial and error. An alternative way to specify goals is to provide images of the goal, and then train a goal classifier on this data [40, 42]. The success probabilities from this classifier can then be used as a reward for training reinforcement learning agents to achieve the specified goal. While this is appealing in principle, a naïvely-trained classifier can easily be exploited by a reinforcement learning algorithm: the RL algorithm can visit parts of the observation space that the classifier was not trained on, causing it to output incorrect probabilities [13].

A recent approach, variational inverse control with events (VICE) [13], mitigates the exploitation issues of naïve classifiers by adversarially mining negative examples from the policy's own experience, analogously to adversarial imitation learning techniques [11, 17, 12]. However, VICE relies entirely on the positive outcome examples provided at the beginning of training to understand the task, which in practice means that a large number of examples is needed. Further, because VICE relies on on-policy RL for training the policy and for gathering negative examples for the classifier, it requires millions of samples, which may be impractical in the real world. In this paper, we address both of these issues to enable end-to-end reinforcement learning on real robots from pixel observations, and without any task-specific engineering for obtaining rewards. To remove the reliance on a large amount of positive examples provided up front, we elicit a small number of additional queries from a human user as the robot collects additional experience. These active queries are selected based on uncertainty estimates from the classifier that is being used as a reward function, and allow us to learn effective rewards from a small number of initial examples. Further, we extend both the policy learning and the classifier training procedure in VICE to the off-policy setting, which allows us to learn robotic skills in the real world with only 1-4 hours of interaction time, entirely from image observations.

The primary contribution of this paper is a framework for learning robotic skills from high-dimensional observations, such as images, without hand-designing reward functions. Our method uses a small number of examples of positive outcomes (without demonstrations), followed by a modest number of additional binary active queries, where the robot asks the user if a particular outcome is successful or not. Our approach is based on efficient off-policy reinforcement learning, making it well-suited for real-world learning. Our experiments demonstrate that our method can learn a variety of real-world robotic manipulation skills, directly from images, and directly in the real world. Results include draping cloth over an object, placing books on a bookshelf, and pushing mugs onto a coaster. Learning requires minimal user supervision and only 1-4 hours of interaction time, which is substantially less than that of prior work [16, 19, 31, 25, 8].

II. RELATED WORK

Reinforcement learning has been applied to a wide variety of robotic manipulation tasks, including grasping objects [19],

in-hand object manipulation [30, 38, 32, 23], manipulating fluids [35], door opening [44, 3], and cloth folding [28]. However, applications of RL in the real world require considerable effort to design and evaluate the reward function. For example, using thermal cameras for tracking fluids [35], mocap sensors [21] or computer vision systems [34] for tracking objects, and accelerometers for determining the state of a door [44]. Since such instrumentation needs to be done for any new task that we may wish to learn, it poses a significant bottleneck to widespread adoption of reinforcement learning for robotics, and precludes the use of these methods directly in open-world environments that lack this instrumentation.

Data-driven approaches for reward specification [29, 1, 11, 17, 12, 48, 33, 9, 27, 4, 18] seek to overcome this issue, but typically require demonstration data to acquire rewards. Such data can be onerous and time-consuming for users to provide. Recent work on active learning for inverse RL has sought to reduce the required number of demonstrations [2, 6, 26, 5], but still requires some number of demonstrations to be provided manually. Our method only requires a modest number of examples of successful outcomes, followed by binary queries where the user indicates whether a particular outcome that the robot achieved is successful or not. Both of these can be provided easily, without any teleoperation or kinesthetic teaching. Related to our method, Daniel et al. [7] propose to learn rewards from active queries that elicit numerical scores. In contrast to this approach, our method uses only binary success queries, which are easier to provide, and can be readily combined with deep networks for learning skills from image observations. Another line of research that queries binary feedback from humans is that of learning from human preferences [18, 4], but these techniques have so far proven to be quite expensive in the deep reinforcement learning setting in terms of both the supervision needed from humans, and the overall sample complexity. Even in simulation with low-dimensional observations, Christiano et al. [4] make about one thousand queries from humans (we make 50-75 such queries in the real world), and require tens of millions of timesteps of interaction with the environment (we require around tens of thousands of such interactions). Furthermore, comparing trajectories is often a harder form of feedback to elicit from humans, especially towards the start of training where all trajectories might be equally undesirable.

Classifier training serves as an alternative to inverse reinforcement learning for data-driven reward specification [40, 42, 31]. However, using classifier-based rewards for reinforcement learning is prone to exploitation by RL agents, as such agents quickly capitalize on any imperfections in the learned classifier [13, 42]. VICE [13] overcomes this issue issue by adversarially mining negatives form the learned policy, but usually requires a large number of samples to learn due to using an on-policy algorithm [36] for policy improvement and classifier updates. Furthermore, it usually utilizes a large number of goal examples (on the order of 50K for image observations) for successfully learning the task. Our approach overcomes both of these issues, and we demonstrate that it

can be used for practical real-world reinforcement learning.

III. PRELIMINARIES

Our aim is to make it possible to specify reward functions for robotic reinforcement learning with a small number of outcome examples (e.g., photographs of a successful task outcome in the case of image-based RL), followed by a modest number of *active queries*, where the robot asks the user if a particular outcome is a success or not. Here, we summarize the framework of robotic reinforcement learning and introduce how classifiers can be used as reward functions. In RL, the goal is to learn a policy in a Markov decision process, which in a robotic control problem consists of a state space $\mathcal{S}$ and an action space $\mathcal{A}$. In our case, the state space consists of image observations, and the action space consists of desired end-effector motion. The policy, denoted $\pi(\mathbf{a}_t|\mathbf{s}_t)$, chooses actions, and the state is assumed to evolve according to the unknown dynamics $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$. This generates a trajectory of states and actions $\tau : (\mathbf{s}_0, \mathbf{a}_0, \mathbf{s}_1, \mathbf{a}_1, ...)$. The goal in reinforcement learning is to optimize the expected total reward of the trajectory distribution induced by the policy.

a) Maximum Entropy RL: The particular reinforcement learning framework that we will use, called maximum entropy RL [46, 14, 15], also maximizes the entropy of the resulting distribution, resulting in the modified objective

$$J(\pi) = \sum_{t=0}^{T} E_{\tau \sim \pi} \left[r(\mathbf{s}_t, \mathbf{a}_t) - \log \pi(\mathbf{a}_t|\mathbf{s}_t) \right]. \quad (1)$$

While our framework could be combined with standard RL as well, the maximum entropy framework offers two benefits: first, maximum entropy RL algorithms, such as the off-policy soft actor-critic (SAC) algorithm that we use in our experiments [15], tend to produce stable and robust policies for real-world reinforcement learning, and second, maximum entropy RL makes it straightforward to integrate our method with VICE [13], a prior approach for classifier-based rewards, which we discuss in Section V-A. An outline of SAC is presented in Algorithm 1. SAC uses a replay buffer $\mathcal{R}$ to store past transitions, and trains both a critic and a maximum entropy actor on batches sampled from this buffer, while at the same time collecting experience with the current stochastic policy, which is represented as a Gaussian with the mean given by a neural network function of the state.

Algorithm 1 Soft actor-critic (SAC)

1: Initialize policy π, critic Q
2: Initialize replay buffer $\mathcal{R}$
3: **for** each iteration **do**
4:     **for** each environment step **do**
5:         $\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
6:         $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
7:         $\mathcal{R} \leftarrow \mathcal{R} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}$
8:     **for** each gradient step **do**
9:         Sample from $\mathcal{R}$
10:         Update π and Q according to Haarnoja et al. [15]

---

b) Classifier-Based Rewards: Engineering reward functions for RL algorithms is difficult, especially when using image observations, because it requires identifying the state of the objects in the world and formulating a reliable success condition programmatically. Indeed, it is often easier for users to state whether a given outcome is successful or not than to write a program that will do so automatically. However, current RL algorithms require so many episodes that labeling the reward in each one manually would be impractical. A reasonable alternative is to use a goal classifier [42, 40], where the user provides a dataset of example states (e.g., images) before training the policy, denoted $\mathcal{D} := \{(\mathbf{s}_n, y_n)\}$, and a binary classifier $g(\mathbf{s})$ is trained to predict whether a given state is a success or failure. The classifier-based RL framework is then summarised in Algorithm 2. Here, y_n represents the binary success or failure label, and $\mathcal{L}$ denotes a binary classification loss (for example, the cross-entropy loss). Once trained, the classifier can be used to provide a reward during reinforcement learning. If the classifier provides a distribution $p_g(y|\mathbf{s})$, then a particularly convenient form for the reward is given by $\log p_g(y|\mathbf{s})$. As discussed in prior work [13], this has an appealing theoretical interpretation based on a connection to control as inference [24], and in practice can provide some degree of shaping, as log-probabilities often increase smoothly as the agent approaches the goal.

Algorithm 2 Classifier-based rewards for RL

Require: : $\mathcal{D}_i := \{(\mathbf{s}_n, y_n)\}$
1: Update the parameters of g to minimize $\sum_n \mathcal{L}(g(\mathbf{s}_n), y_n)$
2: Run RL or planning, using reward derived from $\log p_g(y|\mathbf{s})$

Prior work generally requires both successful and failed examples to be part of $\mathcal{D}$ [42, 40]. When a policy is trained with this classifier, the policy can learn to *exploit* the classifier, reaching states that are different from those that the classifier was trained on and fool it into outputting a success label erroneously [13]. The degree of exploitation is strongly dependent on how the negative examples are provided, and can only be avoided if a comprehensive set of negative examples covering the entire state space is supplied. In the next section, we will describe an approach which does not require a comprehensive set of negative examples up front, and instead uses a modest number of active queries from the user to address the exploitation problem.

IV. REINFORCEMENT LEARNING WITH ACTIVE QUERIES

The goal of our method, which we call reinforcement learning with active queries (RAQ), is to learn robotic skills via reinforcement learning without requiring hand-engineered reward functions, using data that can be easily obtained from the user. More specifically, we train classifiers to distinguish between goal and non-goal observations, and use them to compute rewards. Instead of learning this classifier from a pre-specified static dataset alone (as done in prior work [40, 42]), we introduce an active learning framework that queries a user for binary success labels for states that it would like to obtain

ground truth labels for. This addresses two major challenges with classifier-based rewards: it removes the need for the user to provide a comprehensive set of negative examples up front, and it mitigates the classifier exploitation problem. Let $\{\mathbf{s}_n\}_{n=1}^t$ denote the set of states that the agent encounters over the learning process, where t is the total number of environment steps that the agent has taken so far. At any given step t, our algorithm decides which states from $\{\mathbf{s}_n\}_{n=1}^t$ (if any) it should query a label for. We first introduce and motivate our query mechanism, and we then show how it can be combined with a classifier-based reward learning technique to obtain a practical algorithm for reinforcement learning in the real world.

A. Active Queries

If the robot requests user labels for every single state it sees, it will have a very accurate reward. However, a typical RL run will collect tens or even hundreds of thousands of states worth of data, as discussed in Section VII, and labeling all of these states is impractical. Minimizing the required number of queries depends critically on the mechanism that decides which state should be labeled. The active learning literature provides a few potential mechanisms based on uncertainty, such as the maximum entropy heuristic. In practice, we found that the maximum entropy heuristic does not actually produce very good results, since the goal is not so much to obtain accurate goal classification everywhere, but rather to eliminate the "exploitation" problem, such that the classifier does not output false positives. To that end, we found that the most effective mechanism to select which states to label was to select the previously-unlabeled states with the highest probability of success according to the classifier. Recall that our reward is provided by a binary classifier, which specifies a distribution $p_g(y|\mathbf{s})$, where y is a binary variable indicating success. Following Fu et al. [13], the reward is given by $\log p_g(y|\mathbf{s})$. We can select the state $\mathbf{s}_k$ to label from the set of observed states according to

$$k = \arg\max_t \ \log p_g(y=1|\mathbf{s}_t) \ \forall t \text{ since last query.}$$

For most practical tasks, this query mechanism is also much more selective than the maximum entropy rule. First, negative examples are much easier to obtain than positive examples for most tasks, so requesting labels for only the potential positive examples avoids superfluous queries for high-entropy states that are unlikely to be informative of success. Second, because the policy is explicitly trying to visit states with high $p_g(y|\mathbf{s})$, classifier exploitation is due to false positives rather than negatives [13]. Since only states with positive predictions can be false positives, querying for labels for these states is an effective mechanism for mitigating classifier exploitation.

Aside from selecting which states to label, we must also choose how often to request labels. We adopt a simple scheme where labels are queried at fixed intervals. We found that simply choosing a query frequency based on the expected training length and a query budget was sufficient. For the real-world robot experiments presented this paper, we adjust the

frequency so that we make between 25 to 75 active queries for a single run of a reinforcement learning experiment. Details on this can be found in Section VII.

B. Classifier-Based RL with Active Queries

We now explain how we combine our active query framework with classifier-based rewards for reinforcement learning. Our approach is summarized in Algorithm 3. Similar to standard classifier reward-based RL, we first train a classifier g on an initial dataset $\mathcal{D}$. The RL algorithm then uses $\log p_g(y|\mathbf{s})$ as the reward, and runs for a predefined number of time steps, at which point we select a new state to query by selecting the state with the largest value for $\log p_g(y|\mathbf{s})$. This labeled state is added to the dataset $\mathcal{D}$, and the classifier is then fine-tuned. We then continue training with RL, and repeat the process. This procedure is repeated until convergence or until a fixed budget of samples or queries is exceeded.

Algorithm 3 Reinforcement learning with active queries (RAQ)

Require: initial $\mathcal{D} := \{(\mathbf{s}_n, y_n)\}$
 1: Initialize policy π, critic Q
 2: Initialize replay buffer $\mathcal{R}$
 3: **for** each iteration **do**
 4:      **for** each environment step **do**
 5:          $\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t|\mathbf{s}_t)$
 6:          $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$
 7:          $\mathcal{R} \leftarrow \mathcal{R} \cup \{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})\}$
 8:      **for** each gradient step **do**
 9:          Sample from $\mathcal{R}$
10:          Compute rewards: $r(\mathbf{s}_t) \leftarrow \log p_g(y_t|\mathbf{s}_t)$
11:          Update π and Q according to Haarnoja et al. [15]
12:      **if** active query **then**
13:          $k \rightarrow \arg\max \log p_g(y_t|\mathbf{s}_t)$ for all t since the last query
14:          **if** $\mathbf{s}_k$ is a successful outcome **then**
15:             $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_k, 1)\}$
16:          **else**
17:             $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_k, 0)\}$
18:          Update g to minimize $\sum_n \mathcal{L}(g(\mathbf{s}_n), y_n)$

V. OFF-POLICY VICE WITH ACTIVE QUERIES

While standard RAQ, as described in Section IV, is effective in mitigating the classifier exploitation problem and can enable reinforcement learning with classifier-based rewards, it only utilizes a very small fraction of the data that is collected when running RL. RL algorithms typically collect tens or hundreds of thousands of transitions when learning to solve a robotic task in the real world, but RAQ only makes tens of queries, barely using 0.1% of the data collected. Ideally, we would like to make use of *all* the data collected during RL. In this section, we first review the basic VICE algorithm, which was proposed by Fu et al. [13]. VICE is classifier-based reward specification framework that uses on-policy RL with policy gradients, and generally requires a large number of positive outcome examples. However, VICE can effectively overcome the classifier exploitation problem, and does so by using all of the data collected during RL without making any active queries. We first show how VICE can be extended into the
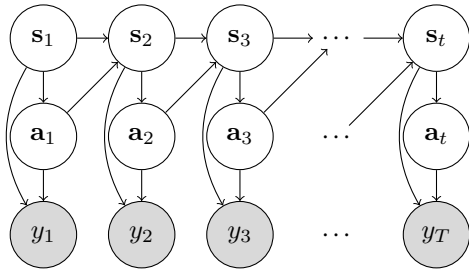
Fig. 2. A graphical model framework for VICE. The node y_t is a binary random variable that denotes whether an event happens at a given time step or not.

off-policy setting, providing a practical method for robotic RL. We then discuss how RAQ can be combined with VICE. The resulting method, which we call VICE-RAQ, combines the best properties of both techniques.

A. VICE

VICE [13] is based on a formulation of reinforcement learning as inference in a graphical model, which is shown in Figure 2. In addition to the states and actions, this graphical model also includes binary event variables, y_t. Intuitively, these variables denote whether a particular event has taken place at time t, which in our case corresponds to successful completion of the task. We can formulate the problem of learning a policy that succeeds at the task as inference in this graphical model, where the policy corresponds to $p(\mathbf{a}_t|\mathbf{s}_t, y_{1:T} = 1)$. This follows from the framework of maximum entropy reinforcement learning [47, 24, 13], and corresponds exactly to the maximum entropy objective in Equation 1 with the reward given by $\log p(y_t = 1|\mathbf{s}_t, \mathbf{a}_t)$, which is also the reward used by RAQ.

Learning the event probabilities in VICE corresponds to an optimization that is similar to maximum entropy inverse reinforcement learning [46]. In the case of high-dimensional and continuous state spaces, as is the case for robotic reinforcement learning, a scalable way to implement maximum entropy inverse RL is to utilize adversarial inverse reinforcement learning (AIRL) [12]. AIRL alternates between training a *discriminator* to discriminate between the positive examples and the current policy's rollouts, and optimizing the policy with respect to the maximum entropy objective in Equation 1, using $\log p(y_t = 1|\mathbf{s}_t, \mathbf{a}_t)$ as the reward. The discriminator in AIRL is parameterized by ψ and given by the following equation:

$$D_\psi(\mathbf{s}, \mathbf{a}) = \frac{\exp(f_\psi(\mathbf{s}, \mathbf{a}))}{\exp(f_\psi(\mathbf{s}, \mathbf{a})) + \pi(\mathbf{a}|\mathbf{s})}. \qquad (2)$$

As shown by Fu et al. [13], $f_\psi(\mathbf{s}, \mathbf{a})$ recovers $\log p(e = 1|\mathbf{s}, \mathbf{a})$ at convergence of this adversarial learning procedure. The basic VICE algorithm is implemented as an on-policy reinforcement learning procedure, typically using policy gradient methods such as TRPO [36]. This makes it difficult to use for real-world robotic learning. Furthermore, VICE requires the success examples to include both the state $\mathbf{s}$ and action $\mathbf{a}$, which is unnatural for a user to provide. We defer further

details about the basic VICE algorithm to prior work due to space constraints [13]. In the following sections, we describe our novel extension of VICE that lifts both of these limitations, and then present our complete VICE-RAQ algorithm, which combines VICE with active queries.

B. Off-Policy VICE

In order to make VICE practical to use for real-world robotic learning, we must extend it so as to make it compatible with efficient off-policy deep reinforcement learning methods, and remove the need to obtain ground truth action labels for the positive examples, so that the user can readily specify examples simply by showing the robot example images of successful outcomes. Extending VICE to the off-policy setting first requires an off-policy reinforcement learning algorithm that can optimize the maximum entropy objective in Equation 1. The soft actor-critic algorithm [15] provides one such method. Next, we need a way to train the discriminator. While in principle this would require importance sampling if using off-policy data from the replay buffer $\mathcal{R}$, prior work has observed that adversarial IRL can drop the importance weights both in theory [10] and in practice [22]. We adopt the same approach, and sample negative examples for the discriminator directly from $\mathcal{R}$ without importance weighting.

The standard VICE algorithm also requires the user-specified success examples to consist of state-action tuples $(\mathbf{s}, \mathbf{a})$, since both $\mathbf{s}$ and $\mathbf{a}$ are required to update the discriminator $D_\theta(\mathbf{s}, \mathbf{a})$ when it has the form in Equation 2. Even when $f_\psi(\mathbf{s})$ does not depend on the action, the term $\pi(\mathbf{a}|\mathbf{s})$ in the denominator does. Providing the actions is unnatural to the user, since the examples consist of isolated individual states showing successful outcomes (e.g., images of successful outcomes). Therefore, we remove the need for the user to specify actions by integrating out the actions for the positive examples in the VICE discriminator update, by using the current policy $\pi(\mathbf{a}|\mathbf{s})$. This amounts to sampling the actions for the positives, denoted $\mathbf{a}_i^E$, from $\pi(\mathbf{a}|\mathbf{s}_i^E)$ as shown on line 11 in Algorithm 4. At convergence, since $\pi(\mathbf{a}|\mathbf{s})$ approaches the expert's policy, this simplification produces the same action distribution, and therefore this update has the same fixed point as when the user supplies ground truth actions.

C. Off-Policy VICE-RAQ

Since the set of positives in VICE is fixed at the start of the algorithm, it typically requires a large set of positive examples provided by the user to begin with in order to prevent overfitting, sometimes as many as several thousand [13]. By integrating VICE with our active query framework, we can substantially decrease the number of examples that are required, at the cost of several tens of binary queries, as in standard RAQ. To integrate RAQ with VICE, we simply add the active queries, as in Algorithm 3, and append the labeled state to the example set if the label is positive. If the label is negative, there is no need to append the state, since VICE already uses all sampled states as negatives. The full VICE-RAQ algorithm is summarized in Algorithm 4. We start
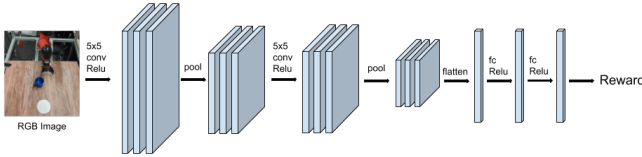
Fig. 3. Our convolutional neural network architecture. The same architecture is used for the policy, critic, and the learned reward function.

out with a dataset $\mathcal{D}$ consisting of positive examples. Every iteration, we collect data from the environment and use it to update f_ψ, the policy, and the Q-function. At fixed intervals, we query the user using our active query mechanism discussed in Section IV-A, and update our dataset $\mathcal{D}$ if the queried state is labeled as a successful outcome. We continue running RL and updating the event probabilities f_ψ, utilizing both our initial dataset and any positives that we obtained from successful queries. This procedure continues until the policy converges, or after a specific period of time.

Algorithm 4 Off-Policy VICE-RAQ with soft actor-critic

Require: : $\mathcal{D}_i := \{(\mathbf{s}_n, 1)\}$
1: Initialize f_ψ (described in Equation 2)
2: Initialize policy π, critic Q
3: Initialize replay buffer $\mathcal{R}$
4: **for** each iteration **do**
5:     **for** each environment step **do**
6:         $\mathbf{a}_t \sim \pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$
7:         $\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1} | \mathbf{s}_t, \mathbf{a}_t)$
8:         $\mathcal{R} \leftarrow \mathcal{R} \cup \{(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1})\}$
9:     **for** each gradient step for f_ψ **do**
10:         Sample positives from $\mathcal{D}$
11:         Sample action labels $\mathbf{a}_i^E \sim \pi(\mathbf{a} | \mathbf{s}_i^E)$
12:         Sample negatives from $\mathcal{R}$
13:         Update f_ψ using Equation 2 as discriminator
14:     **for** each gradient step for the policy π **do**
15:         Sample from $\mathcal{R}$
16:         Compute rewards: $r(\mathbf{s}_t) \leftarrow f_\psi(\mathbf{s}_t)$
17:         Update π and Q according to Haarnoja et al. [15]
18:     **if** active query **then**
19:         $k \to \arg\max f_\psi(\mathbf{s}_t)$ for all t since the last query
20:         **if** $\mathbf{s}_k$ is a successful outcome **then**
21:             $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_k, 1)\}$

VI. VICE-RAQ FOR IMAGE-BASED MANIPULATION

We implemented our methods on top of a standard open-source implementation of the soft actor-critic algorithm [16]. The policy and the critic for each task are represented using convolutional neural networks, shown in Figure 3. It consists of two convolutional layers, each of which is followed by a max-pooling layer, with 8 filters in each of the convolutional layers for simulated tasks, and 32 filters per layer for real world tasks. The flattened output of the convolutional layers is followed by two fully-connected hidden layers with 256 units each. The ReLU non-linearity is applied after each of the convolutional and fully-connected layers. The reward function

$f_\psi(\mathbf{s})$ is also represented using a convolutional neural network with the same architecture.

We use log-probabilities from a neural network-based classifier as reward for reinforcement learning. However, neural networks are known to drastically change their outputs even with small changes to the input [37]. Thus, they provide a hard decision boundary between different classes, which in our case is similar to running RL with sparse rewards. On the other hand, if the output probabilities of the classifier smoothly transition between positive and negative labels, this would provide a more shaped reward, increasing both the stability and efficiency of the reinforcement learning process. To this end, we found *mixup* regularization to be particularly well-suited for smoothing the classifier predictions [45]. We briefly summarize this technique here. Let $\mathbf{s}_i, \mathbf{s}_j$ be any two inputs to the classifier—either from the replay buffer, or from the set of human-provided goal examples—and let y_i, y_j be the corresponding labels. Mixup regularization takes these input/output pairs, and generates the following *virtual* training distribution:

$$\tilde{\mathbf{s}} = \lambda \mathbf{s}_i + (1 - \lambda)\mathbf{s}_j$$
$$\tilde{y} = \lambda y_i + (1 - \lambda)y_j, \tag{3}$$

where $\lambda \sim \text{Beta}(\alpha, \alpha)$. The mixup hyperparameter α controls the extent of mixup, and higher α corresponds to a higher level of mixup (i.e., the sampled λ are closer to 0.5 than to 0). Zhang et al. [45] showed that mixup enables smoother transitions between different classes by encouraging linear behavior, and our experiments indicate that it indeed makes the learned reward function smoother and more amenable to reinforcement learning.

VII. EXPERIMENTS

We perform experiments both in simulation and in the real world. Videos of learned behavior and simulation results can be found on our project website[1].

Our real world experiments aim to study end-to-end reinforcement learning from pixels without manually engineered rewards nor instrumentation of the environment to measure rewards. We evaluate all of our methods, RAQ, off-policy VICE, and VICE-RAQ, on three complex tasks from vision: pushing a mug onto a coaster, draping a cloth over a box, and a task that requires the robot to insert a book onto a shelf between other books. We also provide a naïve classifier-based baseline (summarised in Algorithm 2) as a comparison for all of the tasks. The goal of these experiments is to verify that VICE-RAQ can successfully learn complex tasks, including non-prehensile manipulation (pushing), tasks with multiple success conditions (where the book can be placed in one of several locations), and tasks with deformable objects (cloth draping). For all our experiments, we use end-effector position control on a 7 DoF Sawyer robotic manipulator, and our observations consist of an RGB image of size 84×84. We do not make use of robot joint angles or end effector-positions. The final success rates of the trained policies for

---

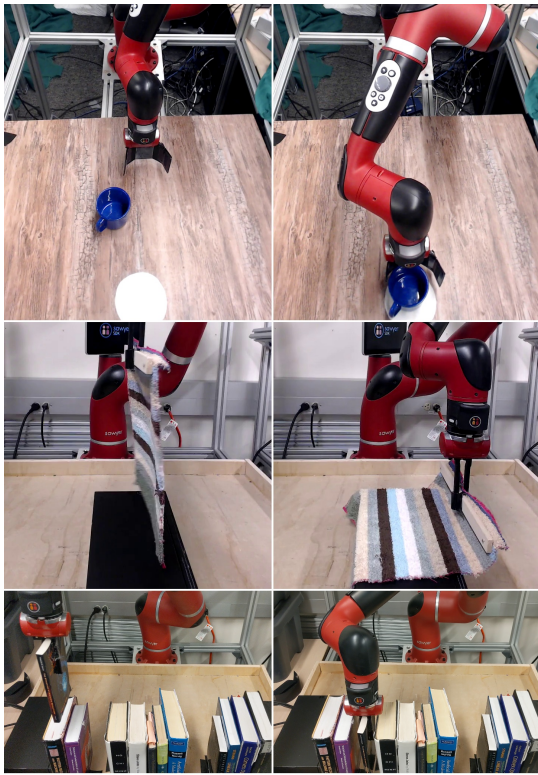[1] https://sites.google.com/view/reward-learning-rl/home

Fig. 4. **Real-world tasks.** The left column depicts possible starting states of the task, while the right column depicts possible goal states. The top row shows the *Visual Pusher* task, in which the goal is to push a mug onto a coaster, and the initial position of the mug is randomized. The middle row presents the *Visual Draping* task, where the goal is to drape a cloth over an object. The bottom row depicts the *Visual Bookshelf* task, where the goal is to inset a book in one of the multiple empty slots in the bookshelf.

	VICE-RAQ (ours)	RAQ (ours)	VICE (ours)	Naïve Classifier
visual pushing	100%	60%	0%	0%
visual draping	100%	100%	100%	0%
visual bookshelf	100%	0%	60%	0%

Fig. 5. Results on the real world robot experiments. For all tasks, the reported numbers are success rates, indicating whether or not the object was successfully pushed to the goal, whether the cloth was successfully draped over the able, and whether the book was placed correctly on the shelf, averaged across 10 trials. In all cases, VICE-RAQ succeeds at learning the task, while VICE and RAQ succeed at some tasks while failing at others.

each of these tasks are shown in Figure 5. We first discuss the individual tasks, and then the experimental results.

a) Visual Pushing: This non-prehensile manipulation task (depicted in Figure 4) requires the robot to push a mug onto a coaster. The position of the mug and coaster must be inferred from the images, and the initial position of the mug varies between different trials. In order to succeed, the robot should push the mug such that is gets placed completely within the coaster. Here, the robot must make use of the images to determine whether it has achieved the goal successfully, and the increased challenge of non-prehensile manipulation allows to better differentiate the performance of the different methods.
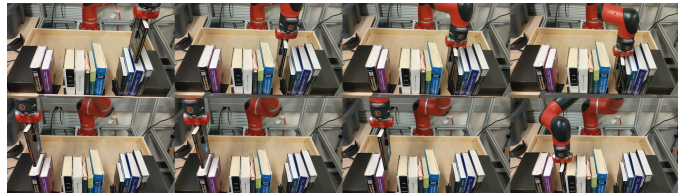

Fig. 6. In this figure, we demonstrate how classifiers are more expressive than goal images for describing a task. The goal for this task is place a book in any empty slot in a bookshelf, and the initial position of the robot arm holding the book is randomized. The top row shows a rollout when the book starts on the right, while the bottom row shows a rollout when the book starts on the left. Here, we see that our method learns a policy to insert the book in different slots in the bookshelf depending on where the book is at the start of a trajectory. The robot usually prefers to put the book in the nearest slot, since this maximizes the reward that it can obtain from the classifier. On the other hand, if we were using goal images to specify the task, the robot would always place the book in one of the two slots, regardless of the starting position of the book.

b) Visual Draping: This task requires draping a cloth over a box – essentially a miniaturized version of a tablecloth draping task. This task is depicted in Figures 4 and 7. The robot starts with holding the cloth in its gripper over the box. In order to succeed, it must drape the cloth smoothly, without crumpling it and without creating any wrinkles. In order to demonstrate the challenges associated with this task, we ran a baseline that only used the robot's end effector position as observation and a hand-defined a reward function on this observation (Euclidean distance to goal). We observed that this baseline failed to achieve the objective of this task, as it simply moved the end effector in a straight line motion to the goal, while this task cannot be solved using any straight line trajectory. See Figure 7 for more details.

c) Visual Bookshelf: In this task, the goal is to insert a book into an empty slot on a bookshelf. The task is depicted in Figures 4 and 6. The initial position of the arm holding the book is randomized, requiring the robot to succeed from any starting position. Crucially, the bookshelf has several open slots, which means that, from different starting positions, different slots may be preferred. We chose this task to emphasize that a goal classifier is fundamentally different from a goal state: there is not a single "goal image" that represents success at the task, but rather a condition on the position of the book that can be fulfilled in different ways (see Figure 6). The successful outcome examples correspondingly illustrate successful placements in both positions.

We provide 80 success examples each for the pushing, draping and book placing tasks. We query once every 250 timesteps for the pushing and book placing task, while querying once every 500 timesteps for the draping task. The *Visual Pusher* experiments are run for 6.2K timesteps (about 90 minutes of real world time), the *Visual Draping* experiments are run for 25K timesteps (about 4 hours), and the *Visual Bookshelf* experiment is run for 19K timesteps (about 3 hours). We therefore make 25 queries for the pushing experiment, 50 queries for the draping experiment, and 75 queries for the book placing experiment. Note that all of these tasks are learned
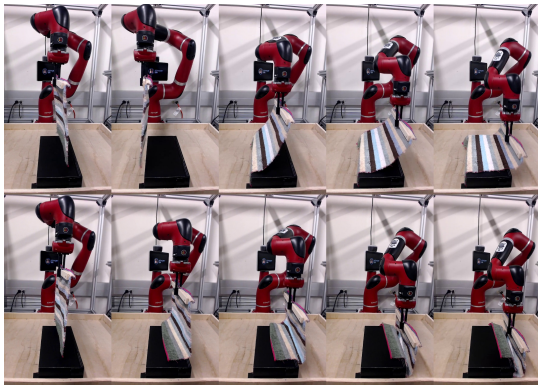
Fig. 7. In this figure, we demonstrate why learning a reward function on pixels is necessary for solving complex tasks in the real world. The task here is to drape a cloth over a box. The top row shows a rollout from the final policy trained by our method, while the bottom row shows a rollout from a policy trained on a hand-defined reward on robot state alone. Our policy is able to successfully drape the cloth over the box, while the policy trained without vision only sees the end-effector position, which it succeeds in moving to the right place, but fails to drape the cloth on the box.

directly from raw images, making these training times very efficient as compared to prior methods, including methods that use ground truth rewards [19, 16, 25, 31]. We hypothesize that the regularized classifiers learned by our method provide favorable reward shaping that makes image-based RL not only more practical, by not requiring engineered rewards, but also substantially more efficient.

The results of our experiments are provided in Figure 5. All the tasks are evaluated in terms of success rates.

The *Visual Pushing* task requires the robot to interpret the RGB camera images and deal with variability in the mug placement. For this task, VICE-RAQ obtains a success rate of 100%, while RAQ only obtains a success rate of 60%. Both off-policy VICE and naïve classifier fail to solve this task. This indicates that including active queries in the classifier training process is helpful for obtaining good rewards, both with and without VICE. These experiments further show that VICE-RAQ can improve upon RAQ via leveraging the data collected by the policy during the reinforcement learning process.

For the *Visual Draping* task, we observe that all of our reward-learning methods (off-policy VICE, VICE-RAQ and RAQ) are able to solve the task, and only the naïve classifier baseline fails. We also compare to a baseline that is based on a reward defined on the robot state alone, in this case, the 3D position of the end-effector (the reward being the Euclidean distance to a goal end-effector position). We observe that this baseline fails to solve the task (see Figure 7). This indicates that performing this task requires the robot to actually pay attention, visually, to the deformations in the cloth, in order to perform the draping successfully. Manually designing reward functions for such deformable object manipulation tasks is generally extremely difficult, but all variants of our method are able to handle it successfully.

For the *Visual Bookshelf* task, where success corresponds to whether the book was successfully placed in an empty slot

on the bookshelf or not), we see that RAQ alone is unable to solve this task, while off-policy VICE learns a policy that only succeeds sporadically. The policy learned with VICE-RAQ solves this task consistently, indicating that the combination of query labels and negative labels from all visited states from VICE provide improved classifier training.

VIII. DISCUSSION AND FUTURE WORK

In this paper, we proposed an approach to reinforcement learning without hand-programmed reward functions. Our method, which we call VICE-RAQ, constructs a reward function from a modest number of user-provided examples of successful outcomes, which in practice might consist simply of pictures of the scene where the task has been successfully completed. Such examples are often substantially easier for a user to provide than either hand-programmed reward functions or full demonstrations. The initial reward is constructed out of a classifier trained on these examples and adversarially mined negatives. Beyond the initially provided success examples, our method uses a modest number of active queries, where the user is asked to label outcomes achieved by the robot as either successful or not. These additional queries are also simple to provide, and roughly correspond to the user directly reinforcing the robot's behavior. However, the user does not need to label all of the robot's experience – only about 50 queries are used in our experiments, out of tens of thousands of transitions.

While our method improves substantially on both naïve classifier rewards and VICE in our experiments, it does have a number of limitations. First, the requirement to obtain labels from the user imposes additional assumptions. Second, the number of queries required—around 50 per training run— is still non-trivial. It is quite possible that a more intelligent query criterion could reduce this number further, and a promising direction for future work is to incorporate techniques for quantifying model uncertainty in the classifier, such as Bayesian neural networks. Lastly, our method does not benefit from any shared structure *between* tasks. In reality, tasks have considerable shared structure, and a classifier that incorporates data from multiple tasks, analogously to prior work on meta-learning [42, 43], could in principle further reduce the number of queries and examples that is needed.

By enabling robotic reinforcement learning without user-programmed reward functions or demonstrations, we believe that our approach represents a significant step towards making reinforcement learning a practical, automated, and readily usable tool for enabling versatile and capable robotic manipulation. By making it possible for robots to improve their skills directly in real-world environments, without any instrumentation or manual reward design, we believe that our method also represents a step toward enabling lifelong learning for robotic systems that learn directly "in the wild." This capability can make it feasible in the future for robots to acquire broad and highly generalizable skill repertoires directly through interaction with the real world.

REFERENCES

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *Machine Learning, Proceedings of the Twenty-first International Conference (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*, 2004.

[2] Daniel S. Brown, Yuchen Cui, and Scott Niekum. Risk-aware active inverse reinforcement learning. In *Conference on Robot Learning (CoRL)*, 2018.

[3] Yevgen Chebotar, Mrinal Kalakrishnan, Ali Yahya, Adrian Li, Stefan Schaal, and Sergey Levine. Path integral guided policy search. In *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*, pages 3381–3388, 2017.

[4] Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. In *NIPS*, 2017.

[5] Robert Cohn, Edmund H. Durfee, and Satinder P. Singh. Comparing action-query strategies in semi-autonomous agents. In *AAAI Conference on Artificial Intelligence AAAI*, 2011.

[6] Yuchen Cui and Scott Niekum. Active reward learning from critiques. In *International Conference on Robotics and Automation (ICRA)*, 2018.

[7] Christian Daniel, Malte Viering, Jan Metz, Oliver Kroemer, and Jan Peters. Active reward learning. In *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014. doi: 10.15607/RSS.2014.X.031.

[8] Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex X. Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control. *CoRR*, 2018. URL http://arxiv.org/abs/1812.00568.

[9] Ashley D Edwards. *Perceptual Goal Specifications for Reinforcement Learning*. PhD thesis, Georgia Institute of Technology, 2017.

[10] Chelsea Finn, Paul Christiano, Pieter Abbeel, and Sergey Levine. A connection between generative adversarial networks, inverse reinforcement learning, and energy-based models. abs/1611.03852, 2016.

[11] Chelsea Finn, Sergey Levine, and Pieter Abbeel. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning (ICML)*, 2016.

[12] Justin Fu, Katie Luo, and Sergey Levine. Learning robust rewards with adversarial inverse reinforcement learning. In *International Conference on Learning Representations (ICLR)*, 2018.

[13] Justin Fu, Avi Singh, Dibya Ghosh, Larry Yang, and Sergey Levine. Variational inverse control with events: A general framework for data-driven reward definition. In *Advances in Neural Information Processing Systems*, 2018.

[14] Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. Reinforcement learning with deep energy-based policies. In *International Conference on Machine Learning (ICML)*, 2017.

[15] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. 2018.

[16] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Vikash Kumar Jie Tan, Henry Zhu, Abhishek Gupta, Pieter Abbeel, and Sergey Levine. Soft actor-critic algorithms and applications. Technical report, 2018.

[17] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[18] Ashesh Jain, Brian Wojcik, Thorsten Joachims, and Ashutosh Saxena. Learning trajectory preferences for manipulators via iterative improvement. In *NIPS*, 2013.

[19] Dmitry Kalashnikov, Alex Irpan, Peter Pastor, Julian Ibarz, Alexander Herzog, Eric Jang, Deirdre Quillen, Ethan Holly, Mrinal Kalakrishnan, Vincent Vanhoucke, and Sergey Levine. Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation. In *Conference on Robot Learning (CoRL)*, 2018.

[20] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference for Learning Representations (ICLR)*, 2015.

[21] Petar Kormushev, Sylvain Calinon, and Darwin G Caldwell. Robot motor skill coordination with em-based reinforcement learning. In *International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2010.

[22] Ilya Kostrikov, Kumar Krishna Agrawal, Debidatta Dwibedi, Sergey Levine, and Jonathan Tompson. Discriminator-actor-critic: Addressing sample inefficiency and reward bias in adversarial imitation learning. *arxiv: 1809.02925*, 2018.

[23] Vikash Kumar, Abhishek Gupta, Emanuel Todorov, and Sergey Levine. Learning dexterous manipulation policies from experience and imitation. *CoRR*, abs/1611.05095, 2016.

[24] Sergey Levine. Reinforcement learning and control as probabilistic inference: Tutorial and review. *ArXiv Preprint*, abs/1805.00909, 2018.

[25] Sergey Levine, Peter Pastor, Alex Krizhevsky, Julian

Ibarz, and Deirdre Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. *I. J. Robotics Res.*, 2018.

[26] Manuel Lopes, Francisco S. Melo, and Luis Montesano. Active learning for reward estimation in inverse reinforcement learning. In *Machine Learning and Knowledge Discovery in Databases, European Conference, ECML PKDD*, 2009.

[27] Anirudha Majumdar, Sumeet Singh, Ajay Mandlekar, and Marco Pavone. Risk-sensitive inverse reinforcement learning via coherent risk models. In *Proceedings of Robotics: Science and Systems*, 2017.

[28] Jan Matas, Stephen James, and Andrew J. Davison. Sim-to-real reinforcement learning for deformable object manipulation. In *Conference on Robot Learning (CoRL)*, 2018.

[29] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML 2000), Stanford University, Stanford, CA, USA, June 29 - July 2, 2000*, pages 663–670, 2000.

[30] OpenAI. Learning dexterous in-hand manipulation. In *arXiv preprint arXiv:1808.00177*, 2018.

[31] Lerrel Pinto and Abhinav Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 3406–3413. IEEE, 2016.

[32] Aravind Rajeswaran, Vikash Kumar, Abhishek Gupta, Giulia Vezzani, John Schulman, Emanuel Todorov, and Sergey Levine. Learning complex dexterous manipulation with deep reinforcement learning and demonstrations. In *RSS*, 2018.

[33] Nathan D. Ratliff, J. Andrew Bagnell, and Martin Zinkevich. Maximum margin planning. In *ICML 2006*, 2006.

[34] Andrei A. Rusu, Matej Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. In *Conference on Robot Learning (CoRL)*, 2017.

[35] Connor Schenck and Dieter Fox. Visual closed-loop control for pouring liquids. In *International Conference on Robotics and Automation (ICRA)*, 2017.

[36] John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust Region Policy Optimization. In *International Conference on Machine Learning (ICML)*, 2015.

[37] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *ICLR 2014*, 2014. URL http://arxiv.org/abs/1312.6199.

[38] Herke van Hoof, Tucker Hermans, Gerhard Neumann, and Jan Peters. Learning robot in-hand manipulation with tactile features. 2015.

[39] Dizan Vasquez, Billy Okal, and Kai Oliver Arras. Inverse reinforcement learning algorithms and features for robot navigation in crowds: An experimental comparison. In

2014 IEEE/RSJ International Conference on Intelligent Robots, 2014.

[40] Mel Vecerik, Oleg Sushkov, David Barker, Thomas Rothrl, Todd Hester, and Jon Scholz. A practical approach to insertion with variable socket position using deep reinforcement learning. *arxiv: 1810.01531*, 2018.

[41] Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. Maximum entropy deep inverse reinforcement learning. In *arXiv preprint arXiv:1507.04888*, 2015.

[42] Annie Xie, Avi Singh, Sergey Levine, and Chelsea Finn. Few-shot goal inference for visuomotor learning and planning. In *Conference on Robot Learning (CoRL)*, 2018.

[43] Kelvin Xu, Ellis Ratner, Anca Dragan, Sergey Levine, and Chelsea Finn. Learning a prior over intent via meta-inverse reinforcement learning. *arXiv:1805.12573*, 2018.

[44] Ali Yahya, Adrian Li, Mrinal Kalakrishnan, Yevgen Chebotar, and Sergey Levine. Collective robot reinforcement learning with distributed asynchronous guided policy search. In *International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[45] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *ICLR 2018*, 2018. URL http://arxiv.org/abs/1710.09412.

[46] Brian Ziebart. Modeling purposeful adaptive behavior with the principle of maximum causal entropy. *PhD thesis, Carnegie Mellon University*, 2010.

[47] Brian Ziebart, Andrew Maas, Andrew Bagnell, and Anind Dey. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2008.

[48] Brian D. Ziebart, Andrew L. Maas, J. Andrew Bagnell, and Anind K. Dey. Maximum entropy inverse reinforcement learning. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008, Chicago, Illinois, USA, July 13-17, 2008*, pages 1433–1438, 2008.