

---

# Reward-Conditioned Policies

---

**Aviral Kumar, Xue Bin Peng, Sergey Levine**

Department of Electrical Engineering and Computer Sciences

University of California, Berkeley

aviralk@berkeley.edu, xbpeng@berkeley.edu, svlevine@eecs.berkeley.edu

## Abstract

Reinforcement learning offers the promise of automating the acquisition of complex behavioral skills. However, compared to commonly used and well-understood supervised learning methods, reinforcement learning algorithms can be brittle, difficult to use and tune, and sensitive to seemingly innocuous implementation decisions. In contrast, imitation learning utilizes standard and well-understood supervised learning methods, but requires near-optimal expert data. Can we learn effective policies via supervised learning without demonstrations? The main idea that we explore in this work is that non-expert trajectories collected from sub-optimal policies can be viewed as optimal supervision, not for maximizing the reward, but for matching the reward of the given trajectory. By then conditioning the policy on the numerical value of the reward, we can obtain a policy that generalizes to larger returns. We show how such an approach can be derived as a principled method for policy search, discuss several variants, and compare the method experimentally to a variety of current reinforcement learning methods on standard benchmarks.

## 1 Introduction

Reinforcement learning, particularly when combined with high-capacity function approximators such as deep networks, has the potential to automatically acquire control strategies for complex tasks together with the perception and state estimation machinery needed to accomplish them, all the while requiring minimal manual engineering [18, 41]. However, in practice, such reinforcement learning methods suffer from a number of major drawbacks that have limited their utility for real-world problems. Current deep reinforcement learning methods are notoriously unstable and sensitive to hyperparameters [5, 14], and often require a very large number of samples. In this paper, we study a new class of reinforcement learning methods that allow simple and scalable supervised learning techniques to be applied directly to the reinforcement learning problem.

A central challenge with adapting supervised learning methods to autonomously learn skills defined by a reward function is the lack of optimal supervision: in order to learn behaviors via conventional supervised learning methods, the learner must have access to labels that indicate the optimal action to take in each state. The main observation in our work is that *any* experience collected by an agent can be used as optimal supervision *when conditioned on the quality of a policy*. That is, actions that lead to mediocre returns represent “optimal” supervision *for a mediocre policy*. We can implement this idea in a practical algorithm by learning policies that are conditioned on the reward that will result from running that policy, or other quantities derived from the reward, such as the advantage value. In this way, all data gathered by the agent can be used as “optimal” supervision for a particular value of the conditioning return or advantage.

Building on this insight, we propose to learn policies of the form  $\pi_\theta(\mathbf{a}|\mathbf{s}, Z)$ , where  $\theta$  represents the parameters of the policy,  $\mathbf{a}$  represents the action,  $\mathbf{s}$  represents the state, and  $Z$  represents some measure of value – either the total return, or the advantage value of  $\mathbf{a}$  in state  $\mathbf{s}$ . Any data collected

using *any* policy can provide optimal supervision tuples of the form  $(\mathbf{s}, Z, \mathbf{a})$ , and a policy of this form can be trained on such data using standard supervised learning.

Our main contribution is a practical reinforcement learning algorithm that uses standard supervised learning as an inner-loop sub-routine. We show how reward-conditioned policies can be derived in a principled way from a policy improvement objective, discuss several important implementation choices for this method, and evaluate it experimentally on standard benchmark tasks and fully off-policy reinforcement learning problems. We show that some variants of this method can perform well in practice, though a significant gap still exists between this approach and state-of-the-art reinforcement learning algorithms.

## 2 Related Work

Most current reinforcement learning algorithms aim to either explicitly compute a *policy gradient* [44, 48], accurately fit a value function or Q-function [9, 22, 35, 47], or both [10, 21]. While such methods have attained impressive results on a range of challenging tasks [13, 20, 22, 28, 36], they are also known to be notoriously challenging to use effectively, due to sensitivity to hyperparameters, high sample complexity, and a range of important and delicate implementation choices that have a large effect on performance [5, 6, 12, 15, 23, 24, 46].

In contrast, supervised learning is comparatively well understood, and even imitation learning methods can often provide a much simpler approach to learning effective policies when demonstration data is available [3, 27, 33]. Indeed, a number of recent works have sought to combine imitation learning and reinforcement learning [2, 43, 45]. However, when expert demonstrations are not available, supervised learning cannot be used directly. A number of prior works have sought to nonetheless utilize supervised learning in the inner loop of a reinforcement learning update, either by imitating a computational expert (e.g., another more local RL algorithm) [7, 20], the best-performing trajectories [25], or by reweighting sub-optimal data to make it resemble samples from a more optimal policy [31, 32]. In this paper, we utilize a simple insight to make it feasible to use suboptimal, non-expert data for supervised learning: suboptimal trajectories can be used as optimal supervision for a policy that aims to achieve a specified return or advantage value.

The central idea behind our method – that suboptimal trajectories can serve as optimal supervision for other tasks or problems – has recently been explored for *goal*-conditioned policies, both with reinforcement learning [1, 16, 34] and supervised learning [8]. Our approach can be viewed as a generalization of this principle to arbitrary tasks, conditioning on the reward rather than a goal state. Like our method, Harutyunyan et al. [11] also learn the distribution of actions conditioned on future states or the trajectory return, but then utilize such models with standard RL techniques, such as policy gradients, to provide more effective credit assignment and variance reduction. Concurrently with our work, Schmidhuber [37] and Srivastava et al. [42] proposed a closely related algorithm that also uses supervised learning and reward conditioning. While our work is concurrent, we further explore the challenges with this basic design, demonstrate that a variety of careful implementation choices are important for good performance, and provide detailed comparisons to related algorithms.

## 3 Preliminaries

In reinforcement learning, our goal is to learn a control policy that maximizes the expected long term return in a task which is modeled as a Markov decision process (MDP). At each timestep  $t$ , the agent receives an environment state  $\mathbf{s}_t \in \mathcal{S}$ , executes an action  $\mathbf{a}_t \in \mathcal{A}$  and observes a reward  $r_t = r(\mathbf{s}_t, \mathbf{a}_t)$  and the next environment state  $\mathbf{s}_{t+1}$ . The goal of the RL algorithm is to learn a policy  $\pi_\theta(\mathbf{a}_t | \mathbf{s}_t)$  that maximizes the return, which is the cumulative discounted reward  $J(\theta)$ , defined as

$$J(\theta) = \mathbb{E}_{\mathbf{s}_0 \sim p(s_0), \mathbf{a}_0: \infty \sim \pi, \mathbf{s}_{t+1} \sim p(\cdot | \mathbf{a}_t, \mathbf{s}_t)} \left[ \sum_{t=1}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t) \right].$$

Prior reinforcement learning methods generally either aim to compute the derivative of  $J(\pi)$  with respect to the policy parameters  $\theta$  directly via policy gradient methods [48], or else estimate a value function or Q-function by means of temporal difference learning, or both. Our aim will be to avoid complex and potentially high-variance policy gradient estimators, as well as the complexity of temporal difference learning.

---

**Algorithm 1** Generic Algorithm for Reward-Conditioned Policies (RCPs)

---

```
1:  $\theta_1 \leftarrow$  random initial parameters
2:  $\mathcal{D} \leftarrow \emptyset$ 
3:  $\hat{p}_1(Z) \leftarrow$  initial value distribution
4: for iteration  $k = 1, \dots, k_{\max}$  do
5:   sample target value  $\hat{Z} \sim \hat{p}_k(Z)$ .
6:   roll out trajectory  $\tau = \{\mathbf{s}_t, \mathbf{a}_t, r_t\}_{t=0}^T$ , with policy  $\pi_{\theta_k}(\cdot | \mathbf{s}_t, \hat{Z})$ 
7:   for each step  $t$ , label  $(\mathbf{s}_t, \mathbf{a}_t)$  with observed value  $Z_t$ 
8:   store tuples  $\{\mathbf{s}_t, \mathbf{a}_t, Z_t\}_{t=0}^T$  in  $\mathcal{D}$ 
9:    $\theta_{k+1} \leftarrow \arg \max_{\theta} \mathbb{E}_{\mathbf{s}, \mathbf{a}, Z \sim \mathcal{D}} [\log \pi_{\theta}(\mathbf{a} | \mathbf{s}, Z)]$ 
10:   $\hat{p}_{k+1} \leftarrow$  update target value distribution using  $\mathcal{D}$ 
11: end for
```

---

## 4 Reward-Conditioned Policies

The basic idea behind our approach is simple: we alternate between training a policy of the form  $\pi_{\theta}(\mathbf{a}_t | \mathbf{s}_t, Z)$  with supervised learning on all data collected so far, where  $Z$  is an estimate of the return for the trajectory containing the tuple  $(\mathbf{s}_t, \mathbf{a}_t)$ , and using the latest policy to collect more data. We first provide an overview of the generic RCP algorithm, and then describe two practical instantiations of the method.

### 4.1 Reward-Conditioned Policy Training

The generic RCP algorithm is summarized in Algorithm 1. At the start of each rollout, a target value  $\hat{Z}$  is sampled from the current target distribution  $\hat{Z} \sim \hat{p}_k(Z)$ . The current policy  $\pi_{\theta_k}(\mathbf{a} | \mathbf{s}, \hat{Z})$  is then conditioned on  $\hat{Z}$  and used to sample a trajectory  $\tau_k$  from the environment. After a rollout, each timestep  $t$  is relabeled with a new value  $Z_t$  reflecting the actual rewards observed over the course of the rollout. This value can be the observed total reward-to-go, or the estimated advantage at  $(\mathbf{s}_t, \mathbf{a}_t)$ . The tuples  $\{\mathbf{s}_t, \mathbf{a}_t, Z_t\}$  are then added to the dataset  $\mathcal{D}$ , which is structured as a first-in first-out queue. The reward-conditioned policy is then updated via supervised regression on the data in the buffer. Finally, the target return distribution  $\hat{p}(Z)$  is updated using the data in  $\mathcal{D}$ , and the process is repeated. RCP performs policy updates using only supervised regression, leveraging prior suboptimal trajectories as supervision.

We explore two specific choices for the form of the values  $Z$ : conditioning on the **return**, which we refer to as RCP-R, and conditioning on the **advantage**, which we refer to as RCP-A. The return conditioned variant, RCP-R, is the simplest: here, we simply choose  $Z_t$  to be the discounted reward to go along the sampled trajectory, such that  $Z_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'}$ .

A more complex but also more effective version of the algorithm can be implemented by conditioning on the *advantage* of  $\mathbf{a}_t$  in state  $\mathbf{s}_t$ . The advantage function is defined as  $A(\mathbf{s}, \mathbf{a}) = Q(\mathbf{s}, \mathbf{a}) - V(\mathbf{s})$ , where  $V(\mathbf{s})$  is the state value function, and  $Q(\mathbf{s}, \mathbf{a})$  is the state-action value function. Thus, RCP-A uses  $Z_t = A(\mathbf{s}_t, \mathbf{a}_t)$ , with  $Q(\mathbf{s}, \mathbf{a})$  estimated using a Monte Carlo estimate, and  $V(\mathbf{s})$  estimated using a separately fitted value function  $\hat{V}_{\phi}(\mathbf{s})$ . Thus, we have  $Z_t = \sum_{t'=t}^T \gamma^{t'-t} r_{t'} - \hat{V}_{\phi}(\mathbf{s}_t)$ . The value function can be fitted using Monte Carlo return estimates, though we opt for a  $\text{TD}(\lambda)$  estimator, following prior work [29, 39].

An important detail of the RCP algorithm is the update to the target value distribution  $\hat{p}_k(Z)$  on line 10. We will describe the theoretical considerations for the choice of  $\hat{p}_k(Z)$  in Section 4.3, while here we describe the final procedure that we actually employ in our method. We represent  $\hat{p}_k(Z)$  as a normal distribution, with mean  $\mu_Z$  and standard deviation  $\sigma_Z$ . The mean and variance are updated based on the *soft-maximum*, i.e.  $\log \sum \exp$ , of the target value  $Z$  observed so far in the dataset  $\mathcal{D}$ . As per line 5 in Algorithm 1, we sample  $\hat{Z}$  from  $\hat{p}_k(Z)$  for each rollout. For RCP-A, a new sample for  $Z$  is drawn at each time step, while for RCP-R, a sample for the return  $Z$  is drawn once for the whole trajectory.

## 4.2 Implementation and Architecture Details

We opt to use a deterministic policy for evaluation in accordance with the evaluation protocol commonly used in prior RL algorithms [10]. During evaluation, the target value is always chosen to be equal to  $\mu_Z + \sigma_Z$  to avoid stochasticity arising from the target value input.

We model the policy  $\pi_\theta(\mathbf{a}|\mathbf{s}, Z)$  as a three-layer fully-connected deep neural network that takes  $s$  and  $Z$  as inputs and outputs a Gaussian distribution over actions. A simple choice for the architecture of the policy network would be to concatenate the additional scalar target value  $Z$  to the state  $s$ , and then use a standard multi-layer fully connected network. However, prior work has observed that such *conditioning* variables can often be utilized more effectively in an architecture that incorporates multiplicative interactions [4, 26, 30]. Based on this insight, we found that using multiplicative interactions between embeddings of  $Z$  and each intermediate layer of the main policy network, shown in Figure 1, produced substantially better results in our experiments. This design prevents the policy network from ignoring the input target values.

## 4.3 Theoretical Motivation for Reward-Conditioned Policies

In this section, we derive the two variants of RCPs, RCP-R and RCP-A, as approximate solutions to a constrained policy search problem. This derivation resembles REPS [32] and AWR [29].

**Notation.** We denote a trajectory by  $\tau$ , and use  $Z(\tau)$  to denote the return of the trajectory, given by  $Z(\tau) = \sum_t r(\mathbf{s}_t, \mathbf{a}_t)$ . For the purpose of this derivation, we operate in a setting where  $Z(\tau)$  can be stochastic, although deterministic returns are a special case of this scenario. We refer to the joint distribution of trajectories  $\tau$  and returns  $Z$  as  $p(\tau, Z)$ . We denote the joint distribution over trajectories and returns under a sampling policy  $\mu$  as  $p_\mu(\tau, Z)$ .

### 4.3.1 Return-Conditioned Variant (RCP-R)

Our constrained policy search formulation aims to learn a return-conditioned policy  $\pi_\theta(\mathbf{a}|\mathbf{s}, Z)$  that maximizes the discounted long-term return  $J(\theta)$ , under the constraint that the induced trajectory-return marginal  $p_\pi(\tau, Z)$  is close to the marginal of the sampling policy,  $p_\mu(\tau, Z)$ . We will first compute the optimal non-parametric solution  $\pi^*$  to the above described optimization problem and then learn  $\pi_\theta(\mathbf{a}|\mathbf{s}, Z)$  by projecting  $\pi^*$  into the space of parametric policies  $\Pi = \{\pi_\theta(\mathbf{a}|\mathbf{s}, Z)|\theta \in \Theta\}$ . This can be formalized as:

$$\arg \max_{\pi} \mathbb{E}_{\tau, Z \sim p_\pi(\tau, Z)} [Z] \quad (1)$$

$$\text{s.t. } D_{\text{KL}}(p_\pi(\tau, Z) || p_\mu(\tau, Z)) \leq \varepsilon \quad (2)$$

Now, we can derive the supervised regression update for RCPs as a solution to the above constrained optimization. We first form the Lagrangian of the constrained optimization problem presented above with Lagrange multiplier  $\beta$ :

$$\mathcal{L}(\pi, \beta) = \mathbb{E}_{\tau, Z \sim p_\pi(\tau, Z)} [Z] + \beta \left( \varepsilon - \mathbb{E}_{\tau, Z \sim p_\pi(\tau, Z)} \left[ \log \frac{p_\pi(\tau, Z)}{p_\mu(\tau, Z)} \right] \right) \quad (3)$$

Differentiating  $\mathcal{L}(\pi, \beta)$  with respect to  $\pi$  and  $\beta$  and applying optimality conditions, we obtain a non-parametric form for the joint trajectory-return distribution of the optimal policy,  $p_{\pi^*}(\tau, Z)$ :

$$p_{\pi^*}(\tau, Z) \propto p_\mu(\tau, Z) \exp \left( \frac{Z}{\beta} \right) \quad (4)$$

Prior work has used this derivation to motivate a *weighted* supervised learning objective for the policy, where the policy is trained by regressing onto previously seen actions, with a weight corresponding to the exponentiated return  $\exp(Z/\beta)$  [29, 31, 32]. To instead obtain an *unweighted* objective, we

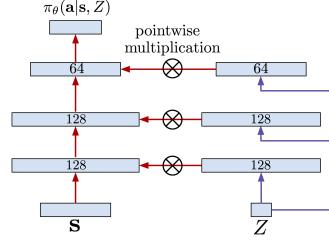


Figure 1: The network architecture used for RCPs in our experiments. Inspired by [4, 26, 30], we use multiplicative interactions between an embedding of  $Z$  and intermediate layers of the policy network.

can instead decompose the joint distribution  $p_\pi(\tau, Z)$  into conditionals  $p_\pi(Z)$  and  $p_\pi(\tau|Z)$ , and use this decomposition to obtain an expression for the trajectory distribution conditioned on the target return  $Z$ . Thus, we can convert Equation 4 into:

$$p_{\pi^*}(\tau|Z)p_{\pi^*}(Z) \propto [p_\mu(\tau|Z)p_\mu(Z)] \exp\left(\frac{Z}{\beta}\right) \quad (5)$$

Equation 5 can be decomposed into separate expressions for the target distribution  $p_{\pi^*}(Z)$  and the conditional trajectory distribution  $p_{\pi^*}(\tau|Z)$ . We obtain a maximum likelihood objective for  $p_{\pi^*}(\tau|Z)$  and an exponentially weighted maximum-likelihood objective for the target distribution  $p_{\pi^*}(\hat{Z})$ .

$$p_{\pi^*}(\tau|Z) \propto p_\mu(\tau|Z) \quad (6)$$

$$p_{\pi^*}(Z) \propto p_\mu(Z) \exp\left(\frac{Z}{\beta}\right) \quad (7)$$

Equation 6 corresponds to fitting a policy  $\pi^*$  to generate trajectories that achieve a particular target return value  $Z$  as depicted in Step 9 in Algorithm 1. Equation 7 corresponds to the process of improving the expected return of a policy by updating the target return distribution to assign higher likelihoods to large values of  $Z$  as shown in Step 10 in Algorithm 1.

For the final steps, we factorize  $p_\pi(\tau|Z)$  as  $p_\pi(\tau|Z) = \prod_t \pi(\mathbf{a}_t|\mathbf{s}_t, Z)p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$ , where the product is over all time steps  $t$  in a trajectory  $\tau$ , and the dynamics  $p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$  are independent of the policy. To train a parametric policy  $\pi_\theta(\mathbf{a}|\mathbf{s}, \hat{Z})$ , we project the optimal non-parametric policy  $p_\pi^*$  computed above onto the manifold of parametric policies, according to

$$\pi_\theta(\mathbf{a}|\mathbf{s}, Z) = \arg \min_{\theta} \mathbb{E}_{Z \sim \mathcal{D}} [\text{DKL}(p_{\pi^*}(\tau|Z) || p_{\pi_\theta}(\tau|Z))] \quad (8)$$

$$= \arg \max_{\theta} \mathbb{E}_{Z \sim \mathcal{D}} \left[ \mathbb{E}_{\mathbf{a} \sim \mu(\mathbf{a}|\mathbf{s}, \hat{Z})} [\log \pi_\theta(\mathbf{a}|\mathbf{s}, Z)] \right] \quad (9)$$

Equation 9 corresponds to a maximum likelihood update for the policy  $\pi_\theta$ . Training is performed only for target return values  $Z$  that have actually been observed and are present in the buffer  $\mathcal{D}$ . We choose to maintain an approximate parametric Gaussian model for  $p_{\pi^*}(Z)$ , and continuously update this model online according to the update in Equation 7. Section 4.2 provides more details on maintaining this model in our practical implementation.

### 4.3.2 Advantage-Conditioned Variant (RCP-A)

In this section, we present a derivation of the advantage-conditioned variant. Our derivation is based on the idea of learning a policy to maximize the *expected improvement* over the sampling policy  $\mu$ . Expected improvement of policy  $\pi(\mathbf{a}|\mathbf{s})$  over another policy  $\mu(\mathbf{a}|\mathbf{s})$  is defined as the difference between their expected long-term discounted returns  $\eta_\mu(\pi) = J(\pi) - J(\mu)$ . Using policy difference lemma [17], we can express expected improvement as:

$$\eta_\mu(\pi) = J(\pi) - J(\mu) = \mathbb{E}_{\mathbf{s}, \mathbf{a} \sim d^\pi(\mathbf{s}, \mathbf{a})} [A_\mu(\mathbf{s}, \mathbf{a})] \approx \mathbb{E}_{\mathbf{s} \sim d^\mu(\mathbf{s}), \mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})} [A_\mu(\mathbf{s}, \mathbf{a})] \quad (10)$$

where the approximate equality holds true if  $\pi$  and  $\mu$  are similar [38].

Analogous to the derivation of RCP-R, for each state-action pair  $(\mathbf{s}, \mathbf{a})$ , we assume that the advantage values are random variables. We denote the advantage random variable for an action  $\mathbf{a}$  at a state  $\mathbf{s}$  with respect to policy  $\pi$  with  $A_\pi(\mathbf{s}, \mathbf{a})$ .

In the case of policies conditioned on advantages, the expected improvement of a policy  $\pi(\mathbf{a}|\mathbf{s}, A)$  over a sampling policy  $\mu(\mathbf{a}|\mathbf{s}, A)$  is given by

$$\eta_\mu(\pi) = \mathbb{E}_{\mathbf{s} \sim d^\pi(\mathbf{s}), A \sim p_\pi(A|\mathbf{s}), \mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s}, A)} [A_\mu(\mathbf{s}, \mathbf{a})] \quad (11)$$

When the policies  $\mu$  and  $\pi$  are close to each other, we obtain a trainable objective, by replacing the intractable state-distribution term  $d^\pi(\mathbf{s})$  in Equation 11 with state distribution  $d^\mu(\mathbf{s})$  of the sampling policy. This approximation has been previously used in the derivation of TRPO [38] and AWR [29]. For a rigorous proof of this approximation, we refer the readers to Lemma 3 in Schulman et al. [38].

Our goal is to learn an advantage-conditioned policy  $\pi(\mathbf{a}|\mathbf{s}, A)$  which maximizes expected improvement while being close to the sampling policy  $\mu(\mathbf{a}|\mathbf{s}, A)$  while staying close to  $\mu$  in distribution. This

is formalized as the following optimization problem:

$$\arg \max_{\pi} \mathbb{E}_{s \sim d^{\mu}(\mathbf{s}), a, A \sim p_{\mu}(a, A | s)} [A] \quad (12)$$

$$\text{s.t. } \mathbb{E}_{s \sim d^{\mu}(\mathbf{s})} [\text{DKL}(p_{\pi}(a, A | s) || p_{\mu}(a, A | s))] \leq \varepsilon \quad (13)$$

Following steps similar to the previous derivation for the return-conditioned variant (RCP-R), we obtain the following maximum-likelihood objective to train a parametric policy  $\pi_{\theta}(a | s, \hat{A})$ , given a sampling policy  $\mu$ , as described in Step 9 of Algorithm 1.

$$\pi_{\theta}(a | s, \hat{A}) = \arg \max_{\theta} \mathbb{E}_{s \sim d^{\mu}(\mathbf{s}), A \sim p_{\mu}(A | s)} [\mathbb{E}_{a \sim \mu(a | s, A)} [\log \pi_{\theta}(a | s, A)]] \quad (14)$$

Further, the target distribution of advantages at any state  $s$  under this procedure is given by:

$$p_{\pi^*}(A | s) \propto p_{\mu}(A | s) \exp\left(\frac{A}{\beta}\right) \quad (15)$$

To summarize, this derivation motivates a maximum-likelihood objective (Equation 14) that trains the policy to choose actions that achieve a particular target advantage value as depicted in Algorithm 1, and the target distribution  $p_{\pi^*}(A | s)$  is updated according to Equation 15 to assign higher likelihoods to actions with higher advantages. Rather than fitting a model to learn a mapping between states and advantages, our model for the target distribution  $p_{\pi^*}(A)$ , as described in Section 4.2 ignores the dependency on states in the interest of simplicity.

#### 4.4 Weighted Maximum Likelihood for Reward-Conditioned Policy Learning

The derivation in Sections 4.3 gives rise to a simple maximum likelihood objective for training the reward-conditioned policy  $\pi_{\theta}(a | s, Z)$ . In contrast to prior work, such as REPS [32] and AWR [29], which use a *return-weighted* maximum likelihood objective to train an *unconditioned* policy, with weights given by exponentiated returns, we expect our *unweighted* maximum-likelihood objective to exhibit less variance, since exponentiated return weights necessarily reduce the effective sample size when many of the (suboptimal) trajectories receive very small weights. However, we can choose to also use weighted likelihood objective with RCPs, and indeed are free to prioritize the samples in  $\mathcal{D}$  to attain better performance. For example, in the case of RCP-A, we can choose to upweight transitions corresponding to highly advantageous actions, rather than training under the data distribution defined by  $\mathcal{D}$ . As we show empirically in Section 5.1, prioritizing transition samples by assigning a weight proportional to exponential target value (either advantage or return) increases sample-efficiency in some cases, although this step is optional in the RCP framework. In practice, we would expect this to also reduce the effective sample size, though we did not find that to be a problem for the benchmark tasks on which we evaluated our method.

## 5 Experimental Evaluation

Our experiments aim to evaluate the performance of RCPs on standard RL benchmark tasks, as well as fully off-policy RL problems. We also present an ablation analysis, which aims to answer the following questions: **(1)** Do RCPs actually achieve a return that matches the value they are conditioned on? **(2)** What is the effect of the policy architecture on the performance of RCPs? **(3)** How does the choice of reweighting method during supervised learning affect performance, and can RCPs perform well with no reweighting at all? **(4)** Are RCPs less sensitive to the size of the replay buffer, as compared to other RL algorithms that use supervised subroutines, such as AWR?

**Experimental setup.** At each iteration, RCP collects 2000 transition samples (i.e. executes 2000 timesteps in the environment), which are appended to the dataset  $\mathcal{D}$ . Unless stated otherwise, for RCPs,  $\mathcal{D}$  is a ring buffer that holds 100k transitions. We also show results with larger buffer sizes in Figure 5. Updates to the policy are performed by uniformly sampling minibatches of 256 samples from  $\mathcal{D}$ . For the advantage-conditioned variant, the value function is updated with 200 gradient steps per iteration, and the policy is updated with 1000 steps.

### 5.1 Performance and Comparisons on Standard Benchmarks

We compare RCP-R and RCP-A to a number of prior RL algorithms, including on-policy algorithms such as TRPO [38] and PPO [40], and off-policy algorithms such as SAC [10] and DDPG [21]. We

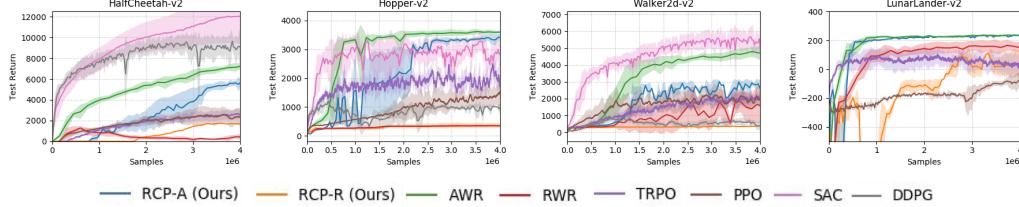


Figure 2: Learning curves of the various algorithms when applied to benchmark tasks. Results are averaged across 5 random seeds. RCP-R performs at par with RWR, and RCP-A is able to learn successful policies for each of the tasks, often outperforming several prior methods.

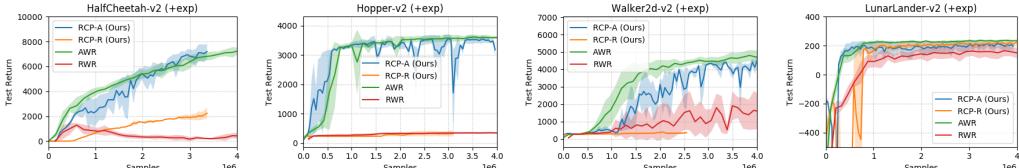


Figure 3: Learning curves for RCP-A and RCP-R with exponential weights for training the policy. AWR is shown for comparison. Results are averaged across 5 random seeds. RCP-A performs similarly to AWR when exponential weighting is used.

also compare to AWR [29], a recently proposed off-policy RL method that also utilizes supervised learning as a subroutine, but does not condition on rewards and requires an exponential weighting scheme during training. When using exponential weighting, both RCP-R and RCP-A resemble AWR, with the main difference being the additional conditioning on returns or advantages. However, RCPs can also use unweighted supervised learning, which can decrease the variance of the supervised learning stage and increase the effective sample size, while AWR requires exponential weighting, without which it can never learn an optimal policy.

Learning curves comparing the different algorithms on three continuous-control and one discrete-action OpenAI gym benchmark tasks are shown in Figure 2. RCP-A substantially outperforms the return-conditioned variant, RCP-R, on all of the tasks, though RCP-R is still able to learn effective policies on the LunarLander-v2 task. While there is still a gap between the performance of RCPs and the best current reinforcement learning algorithms, RCP-A outperforms TRPO and performs comparably or better to PPO. When we additionally incorporate exponential reweighting, as shown in Figure 3, both variants of RCP perform substantially better, and RCP-A performs similarly to AWR, though this is in a sense not surprising, since both methods utilize the same weighted regression step, with the only difference being that the RCP-A policy also receives the advantage values as an input. These results show that, although there is still a gap in performance between RCPs and prior methods, the methods has the potential to learn effective policies on a range of benchmark tasks.

As noted in Section 2, concurrently to our work, Schmidhuber [37] proposed a similar approach, UDRL, though without weighting or advantage conditioning, and reports a final result of around 150 on the LunarLander-v2 task. We can see in Figure 2 that RCPs generally perform better, with RCP-A reaching  $238 \pm 1.3$  on the same task. This suggests that, although conditioning on rewards provides for a simple and effective reinforcement learning method, there are still a number of simple but important design decisions that are essential for good performance.

## 5.2 Performance in Fully Offline Settings

Since RCPs use standard supervised learning and can utilize all previously collected data, we would expect RCPs to be well suited for learning entirely from offline datasets, without on-policy data collection. We follow the protocol described by Kumar et al. [19] and evaluate on static datasets collected from a “mediocre” partially trained policy, with 1 million transition samples per task. RCPs can be trained directly on this dataset, without any modification to the algorithm. We compare to AWR [29] and bootstrapping error accumulation reduction (BEAR) [19], which is a Q-learning method that incorporates a constraint to handle out-of-distribution actions. We also compare to off-policy approximate dynamic programming methods primarily designed for online learning – SAC [10] and TD3 [6], – and PPO [40], which is an importance-sampled policy gradient algorithm.

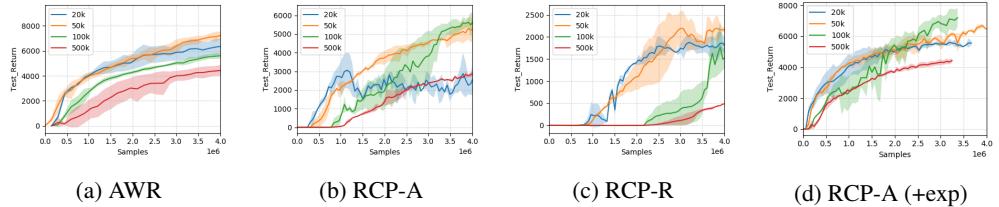


Figure 5: Learning curves demonstrating the effect of varying buffer sizes (20k, 50k, 100k and 500k) on different algorithms: (a) AWR (b) RCP-A (c) RCP-R and (d) RCP-A with exponential weighting on the HalfCheetah-v2 benchmark task. RCP-A generally performs better with larger buffers (compare 50k vs 100k), though performance still degrades with larger buffers.

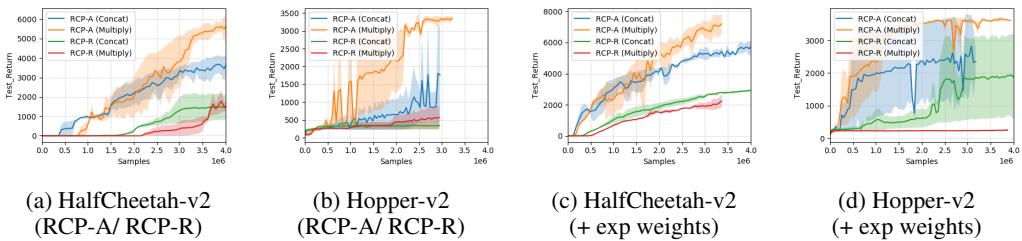


Figure 6: Performance of different architectures on HalfCheetah-v2 and Hopper-v2 environments with replay buffers of size 100k. Figures (c) and (d) correspond to weighted versions of both RCP-A and RCP-R. Note that the architecture *multiply* clearly outperforms *concat* in all cases.

As shown in Figure 4, we find that RCP-A learns effective policies in the purely offline setting on both the environments tested on and achieves performance better than the behavior policy that generated the dataset.

### 5.3 Ablation Experiments

Finally, we perform three ablation experiments to determine the effect of various design decisions for RCP training. The first parameter of variation is the size of the buffer  $\mathcal{D}$  that is used during training. We compare RCP-R, RCP-A, and AWR with different buffer sizes, shown in Figure 5. Note that the performance of AWR degrades significantly as the buffer size increases. This is expected, since AWR constrains the policy against the buffer distribution, therefore larger buffer sizes can result in slower policy improvement. In contrast, RCP-R and RCP-A can handle larger buffers, and perform better with buffers of size 100k as compared to buffers of size 50k, though larger buffers still result in somewhat worse performance. We speculate that this might be due to the fact the low-dimensional and simple benchmark task do not actually require large datasets to train an effective policy, and we might expect larger buffers to be more beneficial on more complex tasks, which we hope to investigate in the future.

In Figure 6, we compare two different architectural choices for both the RCP variants. In the first architecture, labeled *concat* in Figure 6, the target value  $Z$  is concatenated to the state  $s$  and then fed into a three-layer fully-connected network. The second architecture, labeled as *multiply*, is our default choice for experiments in Section 5.1 and uses multiplicative interactions, as discussed in Section 4.2. Learning curves in Figure 6 show that the architecture with multiplicative interactions (*multiply*) leads to better performance across the different environments (HalfCheetah-v2 and Hopper-v2) for both variants (RCP-A and RCP-R).

Finally, we study the relationship between the target value  $\hat{Z}$  that the policy is conditioned and the observed target value  $Z$  achieved by rollouts from the policy. Ideally, we would expect the specified target values of  $Z$  to roughly match the observed value  $\hat{Z}$ , as a reward-conditioned policy is explicitly

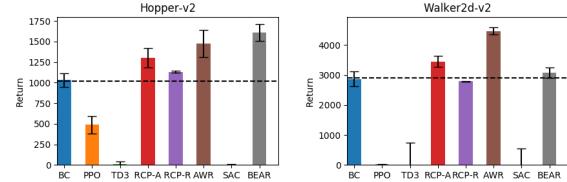


Figure 4: Performance of various algorithms on fully off-policy learning tasks with static datasets. RCP-A learns effective policies that achieve better than dataset average in both cases. RCP-R performs similarly to behavioral cloning (BC).

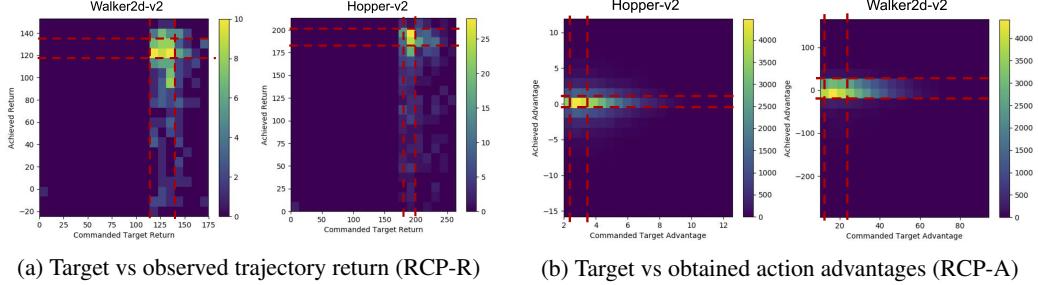


Figure 7: Two-dimensional heatmap visualizing the co-occurrence frequencies of the specified target value  $\hat{Z}$  (x-axis) and the observed value  $Z$  (y-axis) after 2000 epochs of training for (a) RCP-R and (b) RCP-A. The co-occurrence frequencies are empirically estimated using separately executed rollouts that are conditioned on target values sampled from the instantaneous target model  $p_{\pi^*}(Z)$ . Note the similar magnitudes of  $Z$  and  $\hat{Z}$  in most cases.

trained to ensure this (Step 9 of Algorithm 1). In this experiment, we plot a two-dimensional heatmap of co-occurrence frequencies of  $\hat{Z}$  and  $Z_t$  to visualize the relationship between these quantities after about 2000 training iterations for both RCP variants. These heatmaps are shown in Figure 7. We find that both variants of RCP policies achieve returns (or advantages) that are similar enough to the target values they are conditioned on.

## 6 Discussion and Future Work

We presented reward-conditioned policies, a general class of algorithms that enable learning of control policies with standard supervised learning approaches. Reward-conditioned policies make use of a simple idea: sub-optimal trajectories can be regarded as optimal supervision for a policy that does not aim to attain the largest possible reward, but rather to match the reward of that trajectory. By then conditioning the policy on the reward, we can train a single model to simultaneously represent policies for all possible reward values, and generalize to larger reward values.

While our results demonstrate that this approach can attain good results across a range of reinforcement learning benchmark tasks, its sample efficiency and final performance still lags behind the best and most efficient approximate dynamic programming methods, such as soft actor-critic [10], as well as methods that utilize supervised learning in concert with reweighting, such as AWR [29]. We nonetheless expect the simplicity of RCPs to serve as a significant benefit in many practical situations, and we hope that the use of standard supervised learning as a subroutine can also make it easier to analyze and understand the properties of our method. We expect that exploration is likely to be one of the major challenges with reward-conditioned policies: the methods we presented rely on generalization and random chance to acquire trajectories that improve in performance over those previously seen in the dataset. Sometimes the reward-conditioned policies might generalize successfully, and sometimes they might not. Further theoretical and empirical analysis of this generalization behavior may lead to a more performant class of methods, and more optimal sampling strategies inspired by posterior sampling may also lead to better final results. We believe that investigating these directions is an exciting avenue for future work, as it might allow us to devise a new class of reinforcement learning methods that combine the ease of use of supervised learning with the ability to autonomously acquire near-optimal behaviors from only high-level reward specification.

## References

- [1] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS’17*, pages 5055–5065, USA, 2017. Curran Associates Inc. ISBN 978-1-5108-6096-4. URL <http://dl.acm.org/citation.cfm?id=3295222.3295258>.
- [2] B. Balaguer and S. Carpin. Combining imitation and reinforcement learning to fold deformable planar objects. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*,

pages 1405–1412, Sep. 2011. doi: 10.1109/IROS.2011.6094992.

- [3] Felipe Codevilla, Matthias Müller, Alexey Dosovitskiy, Antonio López, and Vladlen Koltun. End-to-end driving via conditional imitation learning. *CoRR*, abs/1710.02410, 2017. URL <http://arxiv.org/abs/1710.02410>.
- [4] Harm de Vries, Florian Strub, Jeremie Mary, Hugo Larochelle, Olivier Pietquin, and Aaron C Courville. Modulating early visual processing by language. In *NIPS*. 2017.
- [5] Justin Fu, Aviral Kumar, Matthew Soh, and Sergey Levine. Diagnosing bottlenecks in deep q-learning algorithms. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2021–2030, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL <http://proceedings.mlr.press/v97/fu19a.html>.
- [6] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1587–1596, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL <http://proceedings.mlr.press/v80/fujimoto18a.html>.
- [7] Dibya Ghosh, Avi Singh, Aravind Rajeswaran, Vikash Kumar, and Sergey Levine. Divide-and-conquer reinforcement learning. In *ICLR*, 2018.
- [8] Dibya Ghosh, Abhishek Gupta, Justin Fu, Ashwin Reddy, Coline Devin, Benjamin Eysenbach, and Sergey Levine. Learning To Reach Goals Without Reinforcement Learning. *arXiv e-prints*, art. arXiv:1912.06088, Dec 2019.
- [9] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine. Continuous deep q-learning with model-based acceleration. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2829–2838, New York, New York, USA, 20–22 Jun 2016. PMLR. URL <http://proceedings.mlr.press/v48/gu16.html>.
- [10] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1861–1870, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR. URL <http://proceedings.mlr.press/v80/haarnoja18b.html>.
- [11] Anna Harutyunyan, Will Dabney, Thomas Mesnard, Mohammad Azar, Bilal Piot, Nicolas Heess, Hado van Hasselt, Greg Wayne, Satinder Singh, Doina Precup, and Remi Munos. Hindsight Credit Assignment. *arXiv e-prints*, art. arXiv:1912.02503, Dec 2019.
- [12] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI’16, pages 2094–2100. AAAI Press, 2016. URL <http://dl.acm.org/citation.cfm?id=3016100.3016191>.
- [13] Nicolas Heess, Dhruva TB, Srinivasan Sriram, Jay Lemmon, Josh Merel, Greg Wayne, Yuval Tassa, Tom Erez, Ziyu Wang, S. M. Ali Eslami, Martin A. Riedmiller, and David Silver. Emergence of locomotion behaviours in rich environments. *CoRR*, abs/1707.02286, 2017. URL <http://arxiv.org/abs/1707.02286>.
- [14] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters, 2017. URL <http://arxiv.org/abs/1709.06560>. cite arxiv:1709.06560Comment: Accepted to the Thirthy-Second AAAI Conference On Artificial Intelligence (AAAI), 2018.
- [15] Matteo Hessel, Joseph Modayil, Hado van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Daniel Horgan, Bilal Piot, Mohammad Gheshlaghi Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *CoRR*, abs/1710.02298, 2017. URL <http://arxiv.org/abs/1710.02298>.
- [16] Leslie Pack Kaelbling. Learning to achieve goals. In *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Chambery, France, 1993. Morgan Kaufmann. URL <http://people.csail.mit.edu/lpk/papers/ijcai93.ps>.

- [17] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, ICML '02, pages 267–274, San Francisco, CA, USA, 2002. Morgan Kaufmann Publishers Inc. ISBN 1-55860-873-7. URL <http://dl.acm.org/citation.cfm?id=645531.656005>.
- [18] 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. *CoRR*, abs/1806.10293, 2018. URL <http://arxiv.org/abs/1806.10293>.
- [19] Aviral Kumar, Justin Fu, George Tucker, and Sergey Levine. Stabilizing off-policy q-learning via bootstrapping error reduction. *CoRR*, abs/1906.00949, 2019. URL <http://arxiv.org/abs/1906.00949>.
- [20] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *J. Mach. Learn. Res.*, 17(1):1334–1373, January 2016. ISSN 1532-4435. URL <http://dl.acm.org/citation.cfm?id=2946645.2946684>.
- [21] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Manfred Otto Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *ICLR*, 2016.
- [22] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, February 2015. ISSN 00280836. URL <http://dx.doi.org/10.1038/nature14236>.
- [23] Rémi Munos, Thomas Stepleton, Anna Harutyunyan, and Marc G. Bellemare. Safe and efficient off-policy reinforcement learning. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 1054–1062, USA, 2016. Curran Associates Inc. ISBN 978-1-5108-3881-9. URL <http://dl.acm.org/citation.cfm?id=3157096.3157214>.
- [24] Ofir Nachum, Mohammad Norouzi, George Tucker, and Dale Schuurmans. Learning gaussian policies from smoothed action value functions, 2018. URL <https://openreview.net/forum?id=B1nLk1-0Z>.
- [25] Junhyuk Oh, Yijie Guo, Satinder Singh, and Honglak Lee. Self-imitation learning. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3878–3887, Stockholm, Sweden, 10–15 Jul 2018. PMLR. URL <http://proceedings.mlr.press/v80/oh18b.html>.
- [26] Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 4797–4805, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.
- [27] Takayuki Osa, Joni Pajarinen, and Gerhard Neumann. *An Algorithmic Perspective on Imitation Learning*. Now Publishers Inc., Hanover, MA, USA, 2018. ISBN 168083410X.
- [28] Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel van de Panne. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Trans. Graph.*, 37(4):143:1–143:14, July 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201311. URL <http://doi.acm.org/10.1145/3197517.3201311>.
- [29] Xue Bin Peng, Aviral Kumar, Grace Zhang, and Sergey Levine. Advantage-Weighted Regression: Simple and Scalable Off-Policy Reinforcement Learning. *arXiv e-prints*, art. arXiv:1910.00177, Sep 2019.
- [30] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. Film: Visual reasoning with a general conditioning layer. In *AAAI*, 2017.
- [31] Jan Peters and Stefan Schaal. Reinforcement learning by reward-weighted regression for operational space control. In *Proceedings of the 24th International Conference on Machine*

*Learning*, ICML '07, pages 745–750, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3. doi: 10.1145/1273496.1273590. URL <http://doi.acm.org/10.1145/1273496.1273590>.

- [32] Jan Peters, Katharina Mülling, and Yasemin Altün. Relative entropy policy search. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, AAAI'10, pages 1607–1612. AAAI Press, 2010. URL <http://dl.acm.org/citation.cfm?id=2898607.2898863>.
- [33] Dean A. Pomerleau. Advances in neural information processing systems 1. chapter ALVINN: An Autonomous Land Vehicle in a Neural Network, pages 305–313. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1989. ISBN 1-558-60015-9. URL <http://dl.acm.org/citation.cfm?id=89851.89891>.
- [34] Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep RL for model-based control. *ICLR*, abs/1802.09081, 2018. URL <http://arxiv.org/abs/1802.09081>.
- [35] Doina Precup, Richard S. Sutton, and Sanjoy Dasgupta. Off-policy temporal difference learning with function approximation. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 417–424, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc. ISBN 1-55860-778-1. URL <http://dl.acm.org/citation.cfm?id=645530.655817>.
- [36] 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 *Proceedings of Robotics: Science and Systems (RSS)*, 2018.
- [37] Juergen Schmidhuber. Reinforcement Learning Upside Down: Don't Predict Rewards – Just Map Them to Actions. *arXiv e-prints*, art. arXiv:1912.02875, Dec 2019.
- [38] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR. URL <http://proceedings.mlr.press/v37/schulman15.html>.
- [39] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *Proceedings of the International Conference on Learning Representations (ICLR)*, 2016.
- [40] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. URL <http://arxiv.org/abs/1707.06347>.
- [41] Avi Singh, Larry Yang, Kristian Hartikainen, Chelsea Finn, and Sergey Levine. End-to-end robotic reinforcement learning without reward engineering. *CoRR*, abs/1904.07854, 2019. URL <http://arxiv.org/abs/1904.07854>.
- [42] Rupesh Kumar Srivastava, Pranav Shyam, Filipe Mutz, Wojciech Jaśkowski, and Jürgen Schmidhuber. Training Agents using Upside-Down Reinforcement Learning. *arXiv e-prints*, art. arXiv:1912.02877, Dec 2019.
- [43] Wen Sun, J. Andrew Bagnell, and Byron Boots. TRUNCATED HORIZON POLICY SEARCH: DEEP COMBINATION OF REINFORCEMENT AND IMITATION. In *International Conference on Learning Representations*, 2018. URL <https://openreview.net/forum?id=ryUlhzWCZ>.
- [44] Richard S Sutton, David A. McAllester, Satinder P. Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. A. Solla, T. K. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems 12*, pages 1057–1063. MIT Press, 2000.
- [45] Qing Wang, Jiechao Xiong, Lei Han, peng sun, Han Liu, and Tong Zhang. Exponentially weighted imitation learning for batched historical data. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6288–6297. Curran Associates, Inc., 2018.

- [46] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Rémi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *CoRR*, abs/1611.01224, 2016. URL <http://arxiv.org/abs/1611.01224>.
- [47] Christopher J. C. H. Watkins and Peter Dayan. Q-learning. In *Machine Learning*, pages 279–292, 1992.
- [48] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Mach. Learn.*, 8(3-4):229–256, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL <https://doi.org/10.1007/BF00992696>.